BTO 2019.001 | Januari 2019

BTO rapport

Datamining voor de zuivering

KWR

BTO

Datamining voor de zuivering

BTO 2019.001 | Januari 2019

Opdrachtnummer
402045-033

Projectmanager
Ing. E. Beerendonk

Opdrachtgever
BTO - Thematisch onderzoek - Zuivering

Kwaliteitsborger(s)
Prof. Dr. Ir. E.R. Cornelissen, Dr. Ir. D. Vries

Auteur(s)
Dr. Ir. B.A. Wols, Dr. Ir. M. Korevaar, Dr. Ir. D. Vries

Verzonden aan
BTO participanten.
Een jaar na publicatie is het rapport openbaar.

Jaar van publicatie
2019

Meer informatie
Wols
T 030 606 9604
E bas.wols@kwrwater.nl

Keywords

Postbus 1072
3430 BB Nieuwegein
The Netherlands

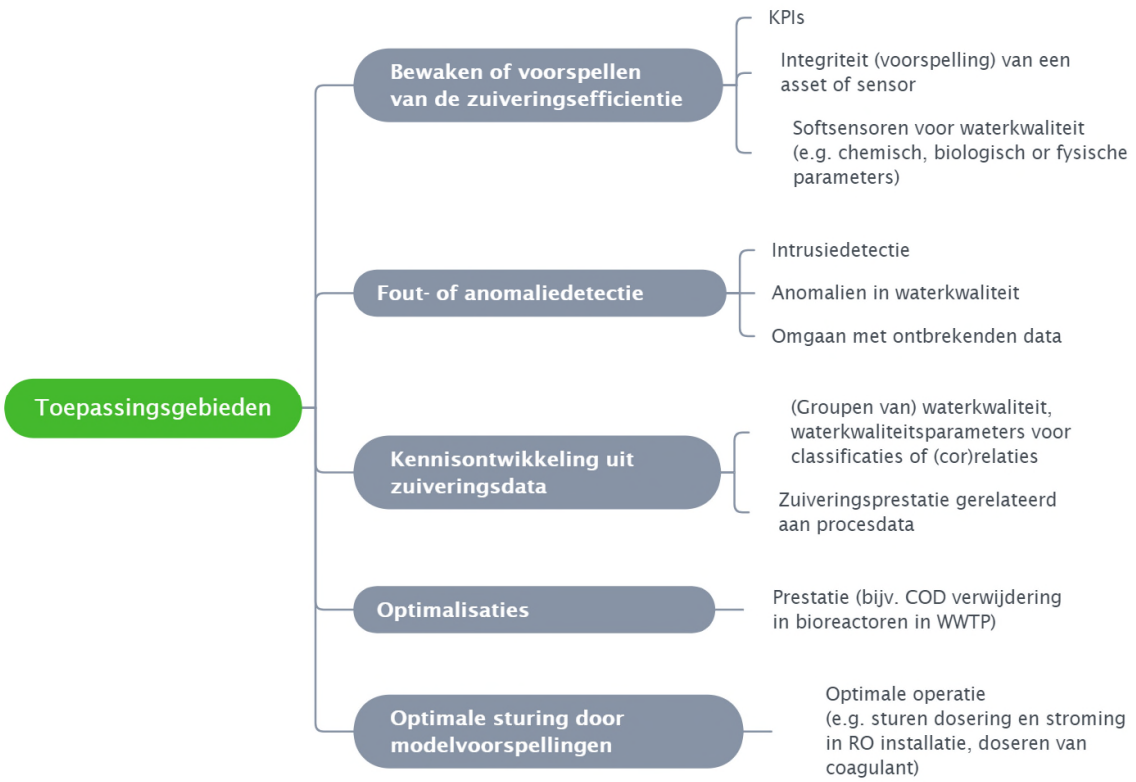
T +31 (0)30 60 69 511
F +31 (0)30 60 61 165
E info@kwrwater.nl
I www.kwrwater.nl

BTO *Managementsamenvatting*

Datamining kan helpen zuivering te optimaliseren

Auteur(s) dr. ir. B.A. Wols, dr. ir. M. Korevaar, dr. ir. D. Vries

Binnen het thema Zuivering is een verkennend onderzoek uitgevoerd naar de mogelijkheden die big data biedt voor de drinkwaterzuivering. Dit levert een overzicht op van verschillende toepassingsgebieden, zoals het karakteriseren van waterkwaliteitsparameters (soft-sensors), het optimaliseren van het doseren van coagulanten en het voorspellen van membraanvervuiling. Een voorbeeld is uitgewerkt: data van een coagulatieproces gevolgd door zandfiltratie is gebruikt om een model te trainen om de drukopbouw in de zandfilters te voorspellen aan de hand van waterkwaliteitsparameters en operationele parameters. Hiermee kunnen filterlooptijd, spoelmomenten en ijzerdosering worden geoptimaliseerd.



Mindmap met overzicht van toepassingsgebieden voor big data in de waterzuivering

Belang: toenemende hoeveelheid data en meer kansen om die te benutten

Vanwege de toenemende digitalisering van de samenleving neemt de hoeveelheid opgeslagen data sterk toe. In veel sectoren worden opgeslagen data al ingezet om processen te optimaliseren. Ook over de drinkwaterzuivering worden steeds meer data verzameld. De vraag rijst in hoeverre big data toegepast kan worden

in de waterzuivering om zuiveringsprocessen te verbeteren.

Aanpak: literatuurscan en aan data uit zuivering ingezet om model te trainen

Een korte literatuurscan is uitgevoerd naar de toepassingsmogelijkheden van big data in de (drink)waterzuivering.
Ook zijn datamining-technieken toegepast op data uit de drinkwaterproductie: door PWN beschikbaar gestelde praktijkdata van een voorzuivering die bestaat uit een coagulatiestap gevolgd door een zandfilter. Deze data omvat waterkwaliteitsgegevens (pH, troebelheid, temperatuur) en operationele gegevens (flow, Fe-dosering) van verschillende parallelle filters over een periode van 2 jaar. Met deze gegevens is een datamodel getraind.

Resultaten: mindmap en model om drukopbouw in zandfilters te voorspellen

Over drinkwaterzuivering is in de literatuur minder te vinden dan over toepassingen van big data in de afvalwaterzuivering, maar genoemd worden onder meer:

- het karakteriseren van waterkwaliteitsparameters (soft-sensors),
- het optimaliseren van het doseren van coagulanten en
- het voorspellen van membraanvervuiling.

Zie ook de mindmap in de figuur.

Het getrainde datamodel voorspelt de drukopbouw in de filters, waarmee vervolgens een voorspelling kan worden gemaakt van de looptijd van het filter op basis van waterkwaliteit en operationele parameters. Bovendien is bepaald hoe de operationele parameters de filtratielooptijd beïnvloeden. De looptijd neemt toe bij hogere Fe-dosering (tot een plateau-waarde is bereikt), lagere troebelheid, lagere flow en bij een pH rond 8.0. Met het data-model kunnen deze verbanden gekwantificeerd worden, zodat het model gebruikt kan worden om filterlooptijd, spoelmomenten en ijzerdosering te optimaliseren.

Implementatie: optimaliseren coagulatie en filtratie met datamodel.

Met het verkregen model kan de spoelduur en ijzerdosering worden geoptimaliseerd. Dit kan leiden tot vermindering van het verbruik van ijzer, opslag van ijzerslib en spoelwaterverlies. De resultaten zijn opgeleverd in de vorm van een interactief werkblad. Aan de hand van dit werkblad kunnen de stappen van de big data analyse worden gevolgd, en kan de gebruiker zelf voorspellingen maken met het model.

Rapport

Dit onderzoek is beschreven in het rapport *Datamining voor de zuivering* (BTO 2019.001).

Meer informatie
B.A. Wols
T 030 606 9604
E bas.wols@kwrwater.nl

KWR
PO Box 1072
3430 BB Nieuwegein
The Netherlands

Voorwoord

Voorliggend rapport is het resultaat van het BTO onderzoek ‘Big data in de zuivering’. Dit project is een verkennend onderzoek (‘kraamkamer’) binnen de themagroep Zuivering. Het rapport omvat een korte beschrijving van literatuuronderzoek en data-analyse die uitgevoerd is met PWN data. Naast een rapport is een interactief werkblad opgeleverd, waarmee de gebruiker stapsgewijs het datamining kan volgen en nadoen, en bovendien voorspellingen kan maken met het opgestelde data-model. Een handleiding

Veel dank gaat uit naar Richard van Daalen (PWN) en Henk van Duist (PWN) voor het beschikbaar stellen van de casestudie, het aanleveren van de data en het meedenken over de data-bewerking en interpretatie.

Bas Wols, Martin Korevaar, Dirk Vries.

Inhoud

1 Literatuurscan

1.1 Betekenis van big data

Met het exponentieel gebruik van het internet en de data die daarbij opgeslagen, opgevraagd en gedeeld wordt, kwamen ook de vragen of en hoe deze data beter kon worden benut. Uitgever O'Reilly Media introduceerde in 2005 de term 'Big Data' als een dataset die van zodanige grootte en complexiteit is dat de traditionele informatietechnologieën niet in staat zijn tot goed datamanagement. Het woord 'big' wordt doorgaans op twee manieren gebruikt: enerzijds als het gaat over letterlijke afmetingen van de gigantische hoeveelheden data ('groot'), en anderzijds als het gaat over figuurlijke afmetingen ('groots'): de grootsheid van mogelijkheden die big data biedt. In een recent BTO onderzoek (Van Alphen, 2017; BTO 2017.041) is een overzicht gegeven van de trends op het gebied van data.

Veelgebruikte definities van big data refereren naar 3 V's (*Volume, Velocity, Variety* (Beyer en Laney z.d.), 4 V's (naast de voorgaande 3 ook *Value*, zie (Dijcks 2012)) of 5 V's (Zhai, Ong, en Tsang 2014) die betrekking hebben op de aspecten van big data:

- *Volume*: grote volume aan data;
- *Velocity*: hoge groei van data;
- *Variety*: combinaties van verschillende data(typen);
- *Veracity*: verschillende niveau's van datakwaliteit;
- *Value*: waarde die de gegevens (kunnen) hebben.

Er zijn vele andere definities in de literatuur te vinden, waarbij aspecten vaak betrekking hebben op technologie (capaciteit voor datamanagement), informatie (hoeveelheden en type data), methodieken (technieken voor *datamining*[1] en analyse), en impact. Met deze thema's in het achterhoofd stellen De Mauro e.a. (2015) de volgende definitie voor:

> Big data: Information assets characterized by such a High Volume, Velocity and Variety to require specific Technology and Analytical Methods for its transformation into Value

1.2 Big data-technieken (datamining)

Dit rapport spitst zich toe op het gebruik van big data-technieken of het zogenoemde 'datamining' voor diverse kwesties en vraagstukken in de zuivering. We lichten kort de technieken toe, uitgebreidere informatie is te vinden in de BTO-rapporten (Vonk en Vries 2016).

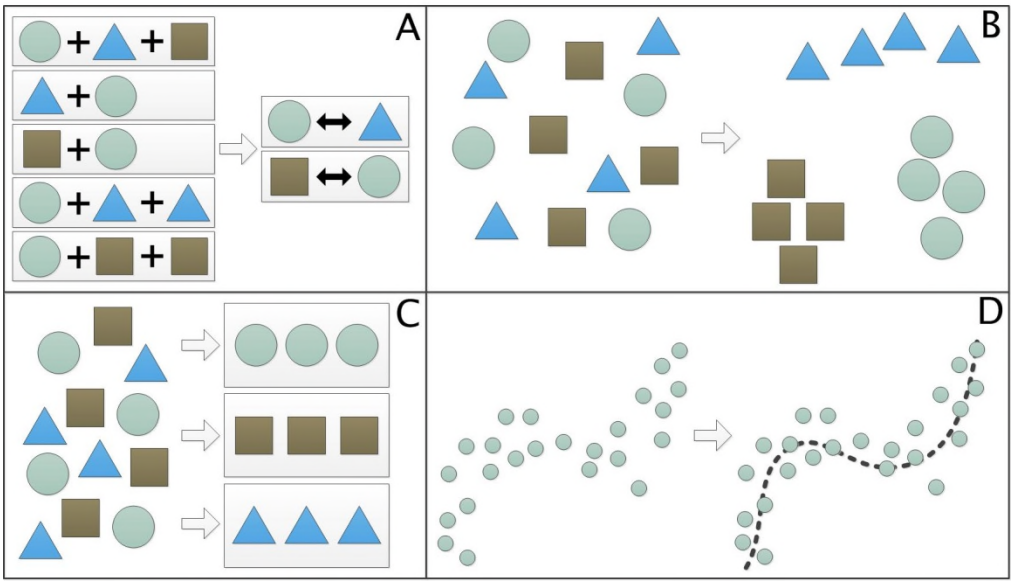
Met big-datatechnieken kunnen verschillende doelen bereikt worden: verbanden en patronen ('association rules'), clusteren, classificatie en regressie (zie Figuur 1). Een zeer grote set aan datamining-technieken is bekend als 'machine learning'. Machine learning omvat volgens een van de eerste definities van deze term de studie naar, en

[1] Met datamining wordt hier verstaan: het gebruik van technieken met als doel: 'knowledge discovery in databases' zoals beschreven in (Vonk en Vries 2016)

implementatie van computer modellen voor een breed scala aan leerprocessen (Carbonell, Michalski, en Mitchell 1983).

Voor elk machine learning-probleem worden 3 deelproblemen onderscheiden (Domingos 2012):

- een model ('representation model': een model die de data op een bepaalde manier beschrijft),
- criteria om het model te evalueren;
- een optimalisatie-algoritme: een methode die zoekt naar het optimum van het (wiskundige) probleem die voldoet aan de criteria.



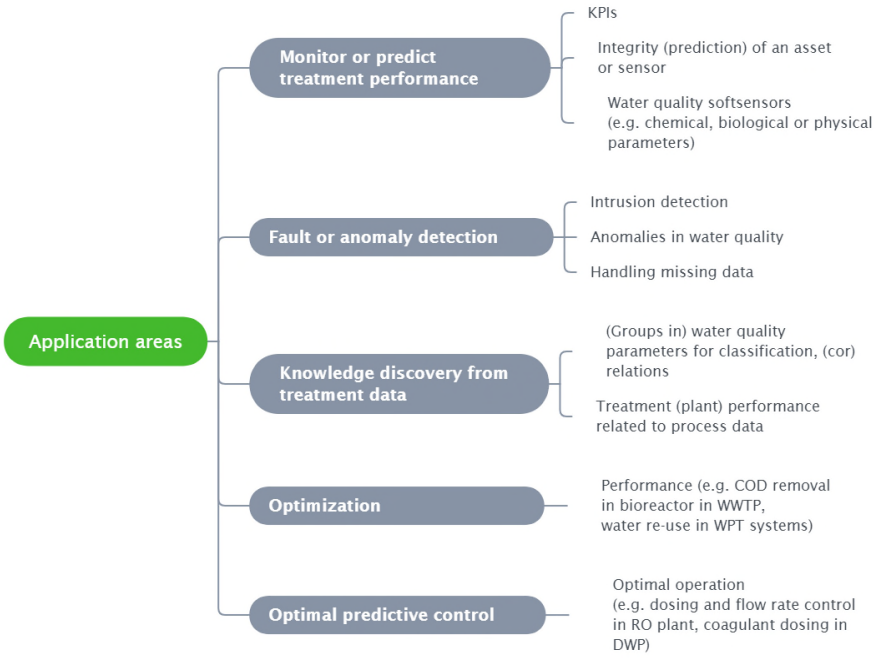
Figuur 1-1: grafische representatie van de doelstellingen die datamining kan hebben: (A) verbanden destilleren, (B) clusteren, (C) classificeren en (D) regressiemodel fitten (uit: Vonk en Vries 2016).

1.3 Big data in de zuivering

Uit een scan van literatuur wordt duidelijk dat datamining (big data-technieken) voor diverse toepassingen kan worden ingezet. We kunnen daarbij een 'passief' en een 'actief' niveau van ingrijpen op het zuiveringsproces onderscheiden:

- *Passief, d.w.z. een model of algoritme grijpt niet in de bedrijfsvoering:*
  - monitoren van zuiveringsprestaties en/of integriteit van assets (waaronder ook sensoren) (*monitor or predict treatment performance*);
  - fout- of anomaliedetectie (*fault or anomaly detection*);
  - modelleren of extraheren van verbanden tussen parameters in het zuiveringsproces en monitoren en informatie voor besluitvorming ondersteunende systemen (*knowledge discovery from treatment data*)
  - optimalisatie van een proces (*optimization*).
- *Actief, d.w.z. het model wordt onderdeel van een bedrijfsvoeringstrategie:*
  - benutten van een *machine learning*-model als onderdeel van een regelstrategie. Voor de regelstrategie wordt vaak optimale besturingstheorie toegepast (*optimal predictive control*).

Met machine learning is een breed palet aan toepassingen mogelijk. Uit een literatuurselectie zijn bovenstaande 5 categorieën benoemd en verder onderverdeeld in de doelstellingen waarvoor machine learning is toegepast, zoals het voorspellen van mangaan in behandeld water (Pinto e.a. 2009) binnen de categorie 'monitoren van zuiveringsprestaties'.



Figuur 1-2: Overzicht van toepassingsgebieden waarbij machine learning is ingezet.

Een overzicht is gemaakt waarbij gegroepeerd is naar de toepassingsgebieden en subcategorieën zoals in Figuur 2. Dit overzicht is weergegeven in Tabel 1. Veel toepassingen van big data worden toegepast in de afvalwaterzuivering. De drinkwaterbehandelingsprocessen die het meest genoemd worden zijn coagulatie en filtratie (Deng et al. 2017; Zhang et al. 2013; Chawakitchareon et al. 2017; Haghiri et al. 2018; Bae et al. 2006). Met behulp van datamining kan een regressiemodel de juiste coagulantdosis voorspellen, en/of waterkwaliteitsparameters (zoals troebelheid, mangaan) na de filtratiestap. Een andere gedocumenteerde toepassing in de drinkwaterzuivering is het voorspellen van membraanvervuiling (Lee at al. 2009). Daarnaast wordt datamining toegepast om waterkwaliteitsparameters te 'meten' (lees: online voorspellen) met behulp van softsensoren, zoals bijvoorbeeld het voorspellen van norovirus aan de hand van direct meetbare waterkwaliteitsparameters zoals pH, geleidbaarheid, temperatuur en regenval.

Tabel 1: overzicht van literatuurbronnen per toepassingsgebied. Gebruikte afkortingen in machine learning-technieken: ANN: Artificial neural network, KNN: K-nearest neighbours, SOM: Self-organizing map, SVM: Support Vector Machine, RF: Random Forests, FL: Fuzzy Logic, PCA: Principle Component Analysis, GA: Genetic Algoritm, EA: Evolutionary Algorithm, GPR: Gaussian process regression, AI: Artificial intelligence, ML : machine learning, PSO: particle swarm optimization.

Application	Subcategorie	Applications of machine learning in (waste)water treatment	Technique	Reference
Monitor/predict treatment performance indicators	Integrity of an asset	Prediction of membrane fouling	GA	Lee at al. 2009
	KPI's	Benchmarking KPI's (energy consumption) in WWTP on a daily basis using AI techniques	FL, ANN, SVR, RF	Torregrossa et al. 2016
		Using clustering of operational (flow, level of deposits) and water quality parameters (pH, redox potential) to monitor state of a acidic chromic wastewater treatment plant	SOM, K-means clustering	García, Hilario López; González, Iván Machón; 2004
	Water quality (soft sensors)	Prediction of manganese and turbidity levels after water treatment using ML classification methods	decision trees, K-means clustering	Pinto et al. 2009
		Softsensors in WWTP (for COD (Chemical Oxygen Demand) and ammonium) and identification of input sensor failure using ML	ANN, SOM, RF, PCA	Dürrenmatt et al. 2012
		Overview of ML techniques to predict different parameters in WWTP	ANN, GA and PSO	Fan et al. 2018
		Prediction of effluent concentrations (e.g. total nitrogen)/process efficiency in WWTP	ANN, SVM	Guo et al. 2015; Curteanu et al. 2014
		Inline measurement of COD and NH4-N in WWTP using ML	RF regression	Kern et al. 2014
		Prediction of norovirus in raw water using parameters such as pH, conductivity, turbidity, rain and temperature	GPR, ANN and FL (adaptive neuro-fuzzy inference)	Mohammed et al. 2017
		Prediction of volatile fatty acid concentrations and state monitoring in WTTPs	clustering, ANN regression	Dixon et al. 2007

		Prediction of Cryptosporidium peaks (0 or 1) by neural networks from 8 water quality parameters	ANN	Brion et al. 2001
Fault/anomaly detection and (sensor) integrity checks	Anomalies in water quality	Determine anomalies in (chemical) water quality Monitoring and fault detection in WWTP Softsensors in WWTP and identification of input sensor failure using ML	? GPR ANN, SOM, RF, PCA	Kusiak, Andrew; Shah, Shital; 2006 Samuelsson et al. 2017 Dürrenmatt et al. 2012
	Handling missing data	Fault detection and handling missing data in coagulation process	ANN, fuzzy logic	Lamrini et al. 2014
	Intrusion detection	Data driven physical modelling for intrusion detection in cyber physical systems	KNN, SVM regression techniques	Junejo, K.N., Yau, D., 2016
Knowledge discovery from treatment data	Groups in water quality parameters	Review of ML related to decision support systems in the urban water supply Classification of water treatment parameters into groups relevant for water treatment Determine relevant physico-chemical parameters for specific water treatment processes Clustering to characterize water quality from large number of parameters	- ANN, reinforcement learning clustering SOM, K-means clustering	Hadjimichael et al. 2016 Dinniy et al. 2017 dos Santos et al. 2018 Juntunen et al. 2013
	Treatment plant performance related	diagnose water treatment efficiency in relation to fluorescence and TOC data Understanding filtration performance using ML Investigating causes of elevated turbidity after filtration Understanding WWTP performance by clustering and trajectory diagrams	PCA, parallel factor analysis, SOM, ANN ANN Classification and regression trees hierarchical clustering	Bieroza, et al. 2012 Tashaouie et al. 2012 Upton et al. 2017 Gilbert et al. 2010

Optimization of treatment process	Performance	Optimization of COD (Chemical Oxygen Demand) removal in bioreactor in WWTP using ML	ANN and GA	Picos-Benítez et al. 2017
Optimal predictive control of treatment (processes)	Optimal operation	Prediction of coagulant dose Optimization of operation and control of RO membrane plant (for desalination) Tuning oxygen setpoint in WWTP automatically using reinforced learning	various ML techniques (proprietary) Hitachi AI reinforcement learning	Deng et al. 2017; Zhang et al. 2013; Chawakitchareon et al. 2017; Haghiri et al. 2018; Bae et al. 2006. Embutsu et al. 2016; Zilouchian 2001 Hernandez-Del-Olmo, F., Gaudioso, E., Nevado, A., 2012

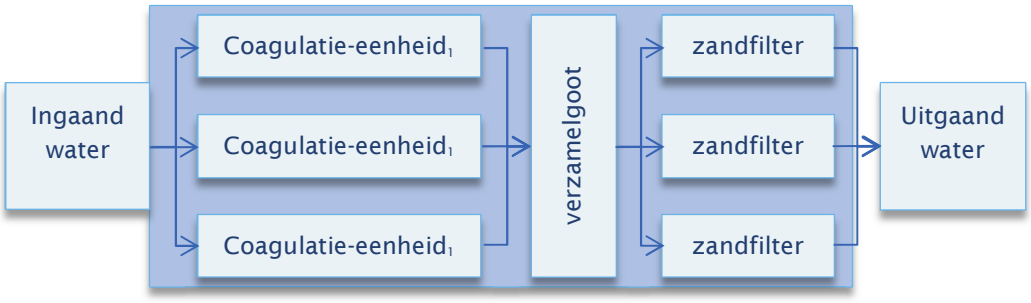
2 Toepassing big data voor de zuivering

2.1 Introductie

Er is praktijkdata beschikbaar gesteld van een (voor)zuivering waarbij oppervlakte water wordt ingenomen. Dit wordt eerst door een coagulatiestap geleid gevolgd door filtratie over een zandfilter (zie Figuur 2-1). Dit heeft tot doel de hoge troebelheid van het water te verlagen. De troebelheid van het inkomende water is seizoensafhankelijk, bijvoorbeeld door verhoogde algengroei in het ingenomen water in de zomer. Een hogere troebelheid betekent een hogere belasting voor de filters waardoor ze sneller verstopt raken. Daarom moeten ze vaker worden gespoeld om de verstopping te verwijderen. Vaker spoelen is ongewenst omdat er meer (spoel)water moet worden gebruikt. Verder zorgt het er ook voor dat tijdens spoelen de filters niet gebruikt worden voor waterproductie zodat de totale capaciteit van de productielocatie vermindert.

De variabele troebelheid aan de ingang en de looptijd van de filters is ook sterk gekoppeld aan de coagulatiestap. Hierin wordt $FeCl_3$ gedoseerd. Dit slaat direct na doseren neer; hieraan zullen de andere (kleine) deeltjes in het water zich hechten en zo steeds grotere deeltjes vormen. De grote vlokken hopen zich op in de coagulatie-eenheid en worden periodiek verwijderd. Het restant wordt verwijderd door het zandfilter. Bij voorkeur wordt er zo min mogelijk ijzer gedoseerd zodat er minder ijzerslib wordt gevormd (ijzerslib is een afvalproduct). Maar als er *te* weinig ijzer wordt gedoseerd, dan wordt de troebelheid niet voldoende verlaagd; er bestaat dus een optimale dosering. Dat optimum is nu echter niet bekend en kan veraf liggen van de gekozen dosering; deze dosering is gebaseerd op de beoordeling van de experts en operators.

In dit project wordt dat verkent met een datagedreven aanpak. Hiervoor wordt de data van de online sensoren gebruikt. De beschikbare data bestaat uit waterkwaliteitsgegevens (pH, troebelheid, temperatuur) en operationele gegevens (flow, druk, Fe-dosering) van verschillende parallelle filters over een periode van 2 jaar. De pH wordt tijdens de coagulatie en na de coagulatie gemeten, de troebelheid wordt na de coagulatie en na het zandfilter gemeten.



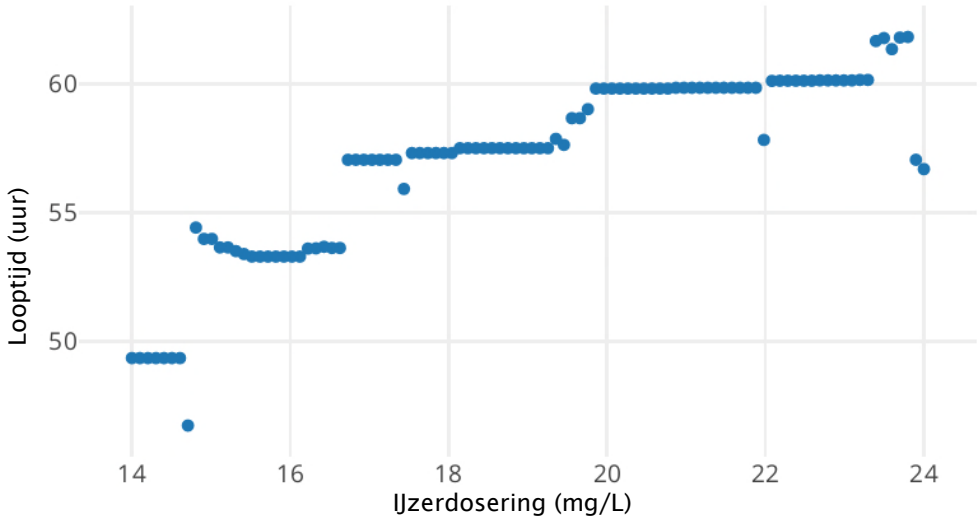
Figuur 2-1Schematisch overzicht van de waterproductielocatie

2.2 Aanpak

De optimale dosering zit dus in een spanningsveld tussen een zo lang mogelijke filterlooptijd en een zo laag mogelijke troebelheid in het uitgaande water. Deze zitten in principe allebei in de dataset. Voor het trainen van een datamodel is het ook belangrijk dat er metingen beschikbaar zijn van de niet-gewenste situatie. Dit is echter voor het geval van troebelheden van het uitgaand water niet het geval; het proces wordt zo gestuurd dat de troebelheid daar *altijd* laag is. Er kan dus geen datamodel worden getraind die voorspellingen doet over de uitgaande troebelheid. Dit is een nadeel van deze aanpak en een die vaak zich voordoet bij machinelearning projecten in de praktijk: doordat de data niet is verzameld voor het doel waar het nu voor gebruikt wordt, is de data eigenlijk niet geschikt om *direct* de gewenste uitvoer te voorspellen. Aan de andere kant is de data wel geschikt om de looptijden te voorspellen omdat er verschillende looptijden aanwezig zijn in de dataset. Dit zal dan ook het onderwerp zijn van dit onderzoek.

2.3 Samenvatting resultaten

Het opgestelde machine learning model maakt voorspellingen van de drukopbouw in zandfilters na spoelen. Met het model kan de spoelduur voorspeld worden en verder geoptimaliseerd worden. Zo kan bijvoorbeeld de voorspelde looptijd worden weergegeven als functie van Fe-dosering, waarbij alle andere parameters constant zijn. Vanuit deze grafiek is dan af te leiden wat de minimale dosering is voor maximale looptijd (zie Figuur 2-2). Hieruit blijkt dat naarmate er meer ijzer gedoseerd wordt de looptijd toeneemt, omdat de belasting van het filter lager is (meer vlokken worden weggenomen in de coagulatiestap). Echter dit vlakt sterk af, zodat er snel teveel ijzer gedoseerd wordt (operators zitten graag aan de veilige kant). Door minder te doseren kan het ijzerverbruik en ijzerslib verminderd worden zonder gevolgen voor de filterlooptijd.



Figuur 2-2 Modelvoorspelling van looptijd van het filter als functie van de doseerdiepte van ijzer (waarbij alle andere parameters constant zijn).

De verdere details van de aanpak zijn verwerkt in een *Jupyter notebook* of werkblad. Dit is een interactief document waarin code kan worden uitgevoerd, maar ook tekst en

plaatjes kunnen worden toegevoegd. Dit document houdt dus code, uitleg en vizualisatie bij elkaar wat het bij uitstek geschikt maar voor presentatie van algoritmes en resultaten van machinelearning projecten. In het interactief werkblad kunnen de stappen van de data analyse gevolgd worden, en kan de gebruiker zelf voorspellingen maken met het model. Een pdf van een statische versie van het werkblad is te vinden in Bijlage II. Voor de interactieve versie moet er software worden geïnstalleerd. De stappen die hiervoor moeten worden gevolgd staan in Bijlage I.

3 Conclusies en aanbevelingen

3.1 Conclusies

Een machine learning model is opgesteld waarmee de drukopbouw in een snelfilter (na coagulatie) voorspeld kan worden. Aan de hand van operationele variabelen, zoals pH, turbiditeit, volumestroom, kan een voorspelling van de drukopbouw gemaakt worden. Hiermee kan tevens de verwachte looptijd van het filter voorspeld worden.

Met het model kan ook de gevoeligheid van de verwachte filterlooptijd voor de diverse operationele en waterkwaliteitsparameters bepaald worden:

- Filterlooptijd neemt toe bij een hogere ijzerdosering, maar vlakt vanaf een waarde van 20 mg/L weer af.
- Filterlooptijd verschilt per individueel filter, dit heeft mogelijk te maken met stromingscondities en/of de geschiedenis van het filter (tijd nadat het zand vervangen is).
- Bij hogere volumestromen neemt de filterlooptijd sterk af als de volumestroom toeneemt.
- Bij toename van de troebelheid neemt de filterlooptijd sterk af.
- Temperatuur heeft nauwelijks effect op de filterlooptijd.
- De filterlooptijd is het langst bij een pH van 8, een iets hogere of lagere pH resulteert in veel lagere looptijden.

Met het data-model kunnen deze verbanden (die voor een groot deel ook te verwachten zijn), gekwantificeerd worden, zodat het proces beter gestuurd kan worden.

3.2 Aanbevelingen

Met de toegepaste machine learning technieken kan voor de looptijden van snelfilters een goede voorspelling gemaakt worden. Het is interessant om deze technieken ook toe te passen op andere zuiveringsprocessen.

Voor het gebruik van het opgestelde model door de waterbedrijven zijn de volgende aanbevelingen relevant:

- Doseren van ijzer kan mogelijk verlaagd worden (er wordt snel overgedoseerd), hiermee kan het ijzerverbruik en de slibvorming verminderd worden.
- Omzetten van het model, zodat het direct een optimale ijzerdosering kan voorspellen.
- Verder onderzoeken wat het verschil in looptijd tussen de diverse filters verklaard.
- Verder onderzoeken wat de grote invloed van pH (rond de 8.0) verklaart.

Specifiek voor het opgestelde model, zijn de volgende verbeteringen aan te bevelen:

- Meenemen van de flocculatiehulpstof in de winter.
- Het toegepaste criterium (terugspoelen bij 1.6 bar) verschilt per filter.
- De gemeten pH's en turbiditeiten van parallelle strengen beter combineren tot een individuele pH of turbiditeit per filter.

- Verder optimalizeren van het machine learning algoritme.

Literatuurlijst

Bae, H; Kim, S; Kim, YJ; . 2006. Decision algorithm based on data mining for coagulant type and dosage in water treatment systems. Water science and technology 53(4-5):321-329

Beyer, Mark A., en Douglas Laney. 2012. “The Importance of ‘Big Data’: A Definition”. 2012. https://www.gartner.com/doc/2057415/importance-big-data-definition.

Bieroza, M., A. Baker, en J. Bridgeman. 2012. “New data mining and calibration approaches to the assessment of water treatment efficiency”. Advances in Engineering Software, CIVIL-COMP, 44 (1): 126–35. https://doi.org/10/b4pcr2.

Brion, G.M., T.R. Neelakantan, en S. Lingireddy. 2001. “Using neural networks to predict peak Cryptosporidium concentrations”. Journal / American Water Works Association 93 (1): 99–105.

Carbonell, Jaime G., Ryszard S. Michalski, en Tom M. Mitchell. 1983. “An overview of machine learning”. In Machine Learning. An artificial intelligence approach, 3–23. San Francisco (CA): Morgan Kaufmann.

Chawakitchareon, P., Boonnao, N., Charutragulchai, P.. 2017. Prediction of alum dosage in water supply by WEKA data mining software. Frontiers in Artificial Intelligence and Applications 292():83-93. https://10.3233/978-1-61499-720-7-83

De Mauro, Andrea, Marco Greco, en Michele Grimaldi. 2015. “What Is Big Data? A Consensual Definition and a Review of Key Research Topics”. In , 97–104. https://doi.org/10.1063/1.4907823.

Deng, X., Lin, C.. 2017. Application of ELM to predict the coagulant dosing in water treatment plants. Water Science and Technology: Water Supply 17(4):1053-1061. https://10.2166/ws.2016.203

Dijcks, Jean Pierre. 2012. “Oracle: Big data for the enterprise”. Oracle.

Dinniy, M.F., A.R. Barakhbah, en E.M. Kusumaningtyas. 2017. “Quality measurement classification for water treatment using neural network with reinforcement programming for weighting optimization”. In , 126–33. https://doi.org/10.1109/KCIC.2016.7883636.

Dixon, Maurice; Gallop, Julian R; Lambert, Simon C; Lardon, Laurent; Healy, Jerome V; Steyer, Jean-Philippe; . 2007. Data mining to support anaerobic WWTP monitoring. Control engineering practice 15(8):987-999

Domingos, Pedro. 2012. “A Few Useful Things to Know About Machine Learning”. Commun. ACM 55 (10): 78–87. https://doi.org/10/cgc9.

dos Santos, F.C.R., Librantz, A.F.H., Sassi, R.J.. 2018. An approach to clustering using the expectation-maximization and selection of attributes relieff applied to water treatment plants process. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 10657 LNCS():558-565. https://10.1007/978-3-319-75193-1_67

Dürrenmatt, D.J., en W. Gujer. 2012. "Data-driven modeling approaches to support wastewater treatment plant operation". Environmental Modelling and Software 30: 47–56. https://doi.org/10/br9kh5.

Embutsu, I., Kageyama, K., Tsuji, S., Moriwaki, N., Ichige, Y.. 2016. Utilization of AI in the water sector: Case study of converting operating history data to values. Hitachi Review 65(6):139-144.

Fan, M., J. Hu, R. Cao, W. Ruan, en X. Wei. 2018. "A review on experimental design for pollutants removal in water treatment with the aid of artificial intelligence". Chemosphere 200: 330–43. https://doi.org/10/gddv4c.

Gibert, K., G. Rodríguez-Silva, en I. Rodríguez-Roda. 2010. "Knowledge discovery with clustering based on rules by states: A water treatment application". Environmental Modelling & Software 25 (6): 712–23. https://doi.org/10/b2crkw.

Guo, H., Jeong, K., Lim, J., Jo, J., Kim, Y.M., Park, J.-P., Kim, J.H., Cho, K.H.. 2015. Prediction of effluent concentration in a wastewater treatment plant using machine learning models. Journal of Environmental Sciences (China) 32():90-101. https://10.1016/j.jes.2015.01.007

Hadjimichael, A., Comas, J., Corominas, L.. 2016. Do machine learning methods used in data mining enhance the potential of decision support systems? A review for the urban water sector. AI Communications 29(6):747-756. https://10.3233/AIC-160714

Haghiri S., Daghighi A., Moharramzadeh S.. 2018. Optimum coagulant forecasting by modeling jar test experiments using ANNs. Drinking Water Engineering and Science 11(1):1-8. https://10.5194/dwes-11-1-2018

Hernandez-Del-Olmo, F., Gaudioso, E., Nevado, A.. 2012. Autonomous adaptive and active tuning up of the dissolved oxygen setpoint in a wastewater treatment plant using reinforcement learning. IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews 42(5):768-774. https://10.1109/TSMCC.2011.2162401

Junejo, K.N., Goh, J.. 2016. Behaviour-based attack detection and classification in cyber physical systems using machine learning. CPSS 2016 - Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security, Co-located with Asia CCS 2016:34-43. https://10.1145/2899015.2899016

Junejo, K.N., Yau, D.. 2016. Data driven physical modelling for intrusion detection in cyber physical systems. Cryptology and Information Security Series 14:43-57. https://10.3233/978-1-61499-617-0-43

Juntunen, P., M. Liukkonen, M. Lehtola, en Y. Hiltunen. 2013. "Cluster analysis by self-organizing maps: An application to the modelling of water quality in a treatment process". Applied Soft Computing Journal 13 (7): 3191–96. https://doi.org/10/f4x5pt.

Kern, P., Wolf, C., Gaida, D., Bongards, M., McLoone, S.. 2014. COD and NH4-N estimation in the inflow of Wastewater Treatment Plants using Machine Learning Techniques. IEEE International Conference on Automation Science and Engineering 2014:812-817. https://10.1109/CoASE.2014.6899419

Kusiak, Andrew; Shah, Shital; . 2006. Data-mining-based system for prediction of water chemistry faults. IEEE Transactions on Industrial Electronics 53(2):593-603

Lee, T.-M., H. Oh, Y.-K. Choung, S. Oh, M. Jeon, J.H. Kim, S.H. Nam, en S. Lee. 2009. "Prediction of membrane fouling in the pilot-scale microfiltration system using genetic programming". Desalination 247 (1–3): 285–94. https://doi.org/10/ctk4kd.

López Garcı´ a, Hilario, en Iván Machón González. 2004. "Self-organizing map and clustering for wastewater treatment monitoring". Engineering Applications of Artificial Intelligence 17 (3): 215–25. https://doi.org/10/dj96j7.

Pinto, A., A. Fernandes, H. Vicente, en J. Neves. 2009. "Optimizing water treatment systems using artificial intelligence based tools". WIT Transactions on Ecology and the Environment 125: 185–94. https://doi.org/10/d8n3qx.

Santos, F.C.R. dos, A.F.H. Librantz, en R.J. Sassi. 2018. "An approach to clustering using the expectation-maximization and selection of attributes relieff applied to water treatment plants process". Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 10657 LNCS: 558–65. https://doi.org/10/gddv3q.

Tashaouie, H.R., G.B. Gholikandi, en H. Hazrati. 2012. "Artificial neural network modeling for predict performance of pressure filters in a water treatment plant". Desalination and Water Treatment 39 (1–3): 192–98. https://doi.org/10/5004/dwt.2012.3329.

Torregrossa, D., G. Schutz, A. Cornelissen, F. Hernández-Sancho, en J. Hansen. 2016. "Energy saving in WWTP: Daily benchmarking under uncertainty and data availability limitations". Environmental Research 148: 330–37. https://doi.org/10/f8qj7b.

Upton, A., B. Jefferson, G. Moore, en P. Jarvis. 2017. "Rapid gravity filtration operational performance assessment and diagnosis for preventative maintenance from on-line data". Chemical Engineering Journal 313: 250–60. https://doi.org/10/gddv3s.

Van Alphen, H.J. 2017. "Diepte-artikel data", BTO 2017.041, KWR Watercycle Research Institute.

Vonk, Erwin, en D. Vries. 2016. "Datamining voor assetmanagement - inventarisatie en voorbeelden uit de watersector". BTO 2016.007.

Yu R.-F., Kang S.-F., Liaw S.-L., Chen M.-C.. 2000. Application of artificial neural network to control the coagulant dosing in water treatment plant. Water Science and Technology 42(43193):403-408. https://

Zhai, Y., Y. S. Ong, en I. W. Tsang. 2014. "The Emerging 'Big Dimensionality'". IEEE Computational Intelligence Magazine 9 (3): 14–26. https://doi.org/10/f6chx6.

Zhang, K., Achari, G., Li, H., Zargar, A., Sadiq, R.. 2013. Machine learning approaches to predict coagulant dosage in water treatment plants. International Journal of Systems Assurance Engineering and Management 4(2):205-214. https://10.1007/s13198-013-0166-5

Zilouchian, Ali; Jafar, Mutaz. 2001. Automation and process control of reverse osmosis plants using soft computing methodologies. Desalination 135(1):51-59

Bijlage I Handleiding om het interactieve werkblad te gebruiken

Aanleiding

Binnen het verkennend onderzoek heeft KWR een analyse uitgevoerd op de meetdata van een voorzuivering. Hierbij is machine learning gebruikt om een (black box) beschrijving te vinden van de verschillende gemeten grootheden. De resultaten hiervan zijn opgeleverd in een Jupyter notebook[2]. Dit is een interface waarbij zowel code als ook grafische output alsook (uitgebreide) tekstuele beschrijving in één bestand kan worden gepresenteerd. Om de code in het notebook uit te kunnen voeren moeten er wel verschillende stappen worden doorlopen. Deze stappen worden hier beschreven.

De stappen bestaan uit 4 delen:

1. Het installeren van Python met Anaconda
2. Het ophalen van het zip-bestand met alle files benodigd voor het uitvoeren van de notebook
3. Het aanmaken van een conda environment en het installeren van de juiste Python packages met de package manager anaconda
4. Het Jupyter notebook aanroepen

Het installeren van Python met Anaconda

Ga naar de webpagina van anaconda distributie genaamd miniconda: https://conda.io/miniconda.html. Download de Python 3 versie die hoort bij je besturingssysteem. Voor Windows met 64 bit is dit https://repo.anaconda.com/miniconda/Miniconda3-latest-Windows-x86_64.exe. Voer het bestand uit om miniconda te installeren.

Het ophalen van de benodigde files

Er is één zip-bestand genaamd code_worksheet.zip; deze wordt toegestuurd middels WeTransfer (aan te vragen bij de auteurs of te vinden op BTOnet). Het maakt niet zoveel uit waar deze wordt uitgepakt als maar onthouden wordt waar het is uitgepakt. Het is ook belangrijk dat de mappenstructuur wordt behouden bij het uitpakken. Naar de hoofdmap van het uitgepakte bestand wordt voortaan verwezen met *hoofdmap*.

Het aanmaken van een conda environment en het installeren van de juiste Python packages met de package manager anaconda

Wanneer men gebruik maakt van Python is men afhankelijk van verschillende packages. Niet alle versies van alle packages kunnen altijd met elkaar samenwerken. Daarom is het goed om bij het gebruik van een Jupyter notebook ook altijd de definities van de gebruikte packages te voegen. Deze zijn te vinden in de file *environment.yml* welke is

[2] https://jupyter.org/

te vinden in de *hoofdmap*. Daar is ook het bat-script *create_conda_env.bat* te vinden. Als met de Windows verkenner wordt genavigeerd naar de map kan met dubbelklikken op het bestand *create_conda_env.bat* een environment worden aangemaakt. Hiermee zal een heleboel pakketten worden gedownload en geïnstalleerd; dit kan een tijdje duren.

Het Jupyter notebook aanroepen

Jupyter notebook moet worden aangeroepen vanuit de juiste map (dat is de hoofdmap van het uitgepakte code_worksheet bestand) en vanuit het juiste conda environment (dat is het environment dat in de vorige stap is gecreëerd genaamd machinelearning_worksheet). Dat gaat goed als in de verkenner naar de 'code' map wordt genavigeerd en daar op het bestand *run_jupyter_notebook.bat* wordt geklikt. Er wordt een browser geopend en hierin wordt het Jupyter notebook getoond. De code in dit notebook kan worden uitgevoerd door op de code te klikken (hiermee wordt de *cell*[3] geselecteerd) en vervolgens kan door boven in het menu op *Run* te klikken de cel worden uitgevoerd. Afhankelijk van de code kan dit enige tijd duren. Het is in eerste instantie zaak om de cellen van boven naar beneden uit te voeren. Er kunnen ook meerdere cellen worden geselecteerd en met *Run* worden die dan na elkaar uitgevoerd. Ook kan in het menu *Cell/Run all* er voor worden gekozen om alle cellen uit te voeren. Voor een uitgebreidere uitleg van Jupyter en het gebruik van Jupyter wordt verwezen naar de online documentatie: http://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html en https://www.dataquest.io/blog/jupyter-notebook-tutorial/ vanaf het kopje "The notebook interface".

[3] Een cell in een Jupyter notebook is een blokje code of tekst. Een notebook is opgebouwd uit onder elkaar geplaatste cellen. Voor cellen met code die kan worden uitgevoerd staat *In: [x]* waarbij *x* een nummer is dat aangeeft in welke volgorde cellen zijn uitgevoerd. Als er geen getal voor x is ingevuld, is de cel nog niet uitgevoerd.

Bijlage II Statische versie van het interactieve werkblad

Explore the data of surface water pretreatment site in the Netherlands

This worksheet contains the data analysis that has been done for the coagulation/filtration step at a surface water pretreatment site in the Netherlands. In this worksheet (a part) of the Python code can be run by the user the follow the steps that are done for the data analysis. The user can run each block of code by entering the block and clicking on the run button or type CTRL+Enter.

Import Python modules

Please note that importing the necessary packages might take a few minutes.

In [1]:
```
%load_ext autoreload
%autoreload 2

import pandas as pd
import numpy as np
import os
from pprint import pprint
from plotly import offline as py
from plotly import graph_objs as go
from plotly import tools
from IPython.core.display import display, HTML
py.init_notebook_mode(connected=True)

from sklearn import preprocessing
from sklearn.ensemble import GradientBoostingRegressor
from sklearn import model_selection
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import r2_score
from scipy.stats import randint as sp_randint

from sys import platform
import sys
import pickle
import missingno

# current working directory
from pathlib import Path

import warnings

if not sys.warnoptions:
    warnings.simplefilter("ignore")

notebook_subdir = 'notebooks'
project_subdir = 'vo_ml_def'
cwd = Path.cwd()
if cwd.name == notebook_subdir:
    notebook_dir = cwd
    os.chdir('..')
    plot_or_iplot = py.iplot
else:
    if cwd.name != project_subdir:
        os.chdir(project_subdir)
    notebook_dir = cwd / notebook_subdir
    plot_or_iplot = py.plot

from notebooks.helper_functions import *
from notebooks.filters import filterNonPhysicalValues, filterZeroFlow, filterFlu
shes, filterShutdown, filterSavgol
from notebooks.load_data import *

if not notebook_dir.exists():
    raise FileNotFoundError("directory notebooks is not found, " +
                            "make sure 'jupyter notebook' is called " +
                            "from 'notebooks' directory or its parent" +
                            "notebook_dir = {}".format(notebook_dir))
else:
    print('Initialization successful')
notebook_dir = str(notebook_dir)
w_fig = 600 #900
h_fig = 400 #600
```

```
Initialization successful
```

Load data

Loads the data from a file and calculates some features (e.g. the runtime of each filter).

NB: Use a pickle file in 'notebook dir' instead of raw data if this file already exists. If run for the first time, raw data will be read and stored in a pickle file.

In [2]:
```
print('read data from file {file} in directory {dir}'.format(file='df_tseries_an
```

```
d_units.pickle', dir=notebook_dir))
print()
df_t_raw, units = loadDfMeasurements(notebook_dir)
```

```
read data from file df_tseries_and_units.pickle in directory D:\TEMP\VO_zuivering\
vo_ml_def\notebooks

Local pickle not found, load original excel files: this may take a while
Read first excel file
Read seconde excel file
Read third excel file
Spoelen zandfilter 31 actief
Spoelen zandfilter 32 actief
Spoelen zandfilter 33 actief
Spoelen zandfilter 34 actief
Spoelen zandfilter 35 actief
Spoelen zandfilter 36 actief
Spoelen zandfilter 41 actief
Spoelen zandfilter 42 actief
Spoelen zandfilter 43 actief
Spoelen zandfilter 44 actief
Spoelen zandfilter 45 actief
Spoelen zandfilter 46 actief
Spoelen zandfilter 51 actief
Spoelen zandfilter 52 actief
Spoelen zandfilter 53 actief
Spoelen zandfilter 54 actief
Spoelen zandfilter 55 actief
Spoelen zandfilter 56 actief
datetime_of_previous_flush_f31
datetime_of_previous_flush_f32
datetime_of_previous_flush_f33
datetime_of_previous_flush_f34
datetime_of_previous_flush_f35
datetime_of_previous_flush_f36
datetime_of_previous_flush_f41
datetime_of_previous_flush_f42
datetime_of_previous_flush_f43
datetime_of_previous_flush_f44
datetime_of_previous_flush_f45
datetime_of_previous_flush_f46
datetime_of_previous_flush_f51
datetime_of_previous_flush_f52
datetime_of_previous_flush_f53
datetime_of_previous_flush_f54
datetime_of_previous_flush_f55
datetime_of_previous_flush_f56
```

Data preparation

Filter data

The first step is to filter the data:

- Removing non physical values
- Remove data points from filters that are shutdown
- Remove data points when there is no flow
- Obtain the moments of flushing

In [3]:
```
df = filterNonPhysicalValues(df_t_raw)
df = filterShutdown(df)
df = filterZeroFlow(df)
df = filterFlushes(df)
df_filtered = df
```

```
Spoelen zandfilter 31 actief
Spoelen zandfilter 32 actief
Spoelen zandfilter 33 actief
Spoelen zandfilter 34 actief
Spoelen zandfilter 35 actief
Spoelen zandfilter 36 actief
Spoelen zandfilter 41 actief
Spoelen zandfilter 42 actief
Spoelen zandfilter 43 actief
Spoelen zandfilter 44 actief
Spoelen zandfilter 45 actief
Spoelen zandfilter 46 actief
Spoelen zandfilter 51 actief
Spoelen zandfilter 52 actief
Spoelen zandfilter 53 actief
Spoelen zandfilter 54 actief
Spoelen zandfilter 55 actief
Spoelen zandfilter 56 actief
```

Add data fields

Add data fields that may be important features in the data mining procedure:

- Add column with number of active filters (vstroom > 10)

- Determine the flow per filter
- Add the week number

In [4]:
```
df = addNumActiveFilters(df)
df = flowPerFilter(df)
df = addWeekNumber(df)
#df_average = calculateAverageOverFilters(df)
```

Use savgol filter

The turbidity (troebelheid) and pH data suffers inaccuracy due to noise. By using a smoothing filter the spikes are reduced and cleaner data can be fed to the machine learning algorithms later on.

This is done by the function `filterSavgol`. The argument `window` determines how subsequent datapoints are used for the filtering. Making the window too large will average out the variation too much; this reduces data quality.

In [5]:
```
window = 15  # must be odd
df, _df = filterSavgol(df, window)
```

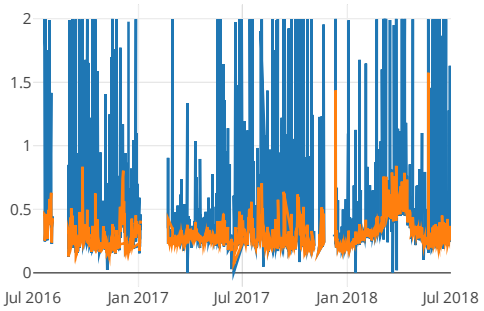
```
Savgol filter applied to columns:
Troebelheid na straat 11
Troebelheid na straat 12
Troebelheid na straat 13
Troebelheid na straat 14
Troebelheid na straat 15
Troebelheid na straat 16
Troebelheid na ZF serie 30
Troebelheid na ZF serie 40A
Troebelheid na ZF serie 40B
Troebelheid na ZF serie 50
PH na straat 11
PH na straat 12
PH na straat 13
PH na straat 14
PH na straat 15
PH na straat 16
PH vlokvormer 11
PH vlokvormer 12
PH vlokvormer 13
PH vlokvormer 14
PH vlokvormer 15
PH vlokvormer 16
```

Explore the data

The following subsections will give a glance of the data and the influence of some smoothing filters.

Plot the effect of savgol smoothing filter

In [6]:
```
col_plot = 'Troebelheid na straat 11'
traces = [go.Scatter(dict(x=_df.index, y=_df[col_plot], mode = 'lines', name='Before smoothing filter')),
          go.Scatter(dict(x=df.index, y=df[col_plot], mode = 'lines', name='After smoothing filter'))]
layout = go.Layout(dict(xaxis=dict(automargin=True),
                        yaxis=dict(automargin=True),
                        width = w_fig, height = h_fig))
py.iplot(go.Figure(data=traces, layout=layout))
```



Export to plot.ly »

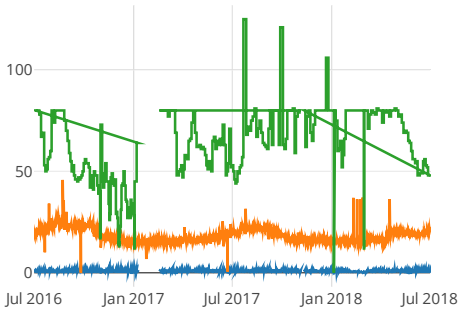
Plot timeseries of: `pressure drop`, `dosing` and `time between`

`flushes`

Variables are plotted over time. In the variable columns_to_plot different variables can be listed that will be plotted. Here, we choose filter 31.

In [7]:
```
columns_to_plot = ['Verschildruk zandfilter 31', 'Doseerdiepte Fe', 'time_betwee
n_flushes_f31']
traces = []
for col_plot in columns_to_plot:
    traces += [go.Scatter(dict(x=_df.index, y=_df[col_plot], mode = 'lines', nam
e=col_plot))]

layout = go.Layout(dict(xaxis=dict(automargin=True),
                        yaxis=dict(automargin=True),
                        width = w_fig, height = h_fig))
py.iplot(go.Figure(data=traces, layout=layout))
#print(df.columns.tolist())
```



Export to plot.ly »

Calculate average quantities per runtime (between flushes)

For the purpose of data mining, all values in a row of the table (also known as dataframe) `df` need to be a valid number. So if for a certain row, only one column has an invalid or unknown number, the whole row needs to be discarded. Consequently, this reduces the available datapoints a lot. To overcome this, all quantities are averaged over each runtime for each filter. Furthermore, we will add the following new quantities for each filter to the dataframe:

- runtime: max of runtime_filter_xx, that is, the total runtime
- dp: pressure drop over one runtime, that is, the difference between the min and max pressure drop (over the bed) in that runtime
- dpdt: dp/runtime, average of the time derivative of the pressure drop over the bed

We have to reorganize the dataframe to streamline the averaging and adding of new quantities.

Show a data view of the current dataframe

Let's see how the dataframe looks like now:

In [8]:
```
print(df.head(2))
```

```
                     Temperatuur ingenomen water  \
2016-07-01 00:00:00                     19.17642
2016-07-01 00:15:00                     19.17642

                     Vstroom totale aanvoer ruwwater  \
2016-07-01 00:00:00                       5633.362793
2016-07-01 00:15:00                       5633.362793

                     Troebelheid na straat 11  Troebelheid na straat 12  \
2016-07-01 00:00:00                       NaN                  0.490279
2016-07-01 00:15:00                       NaN                  0.497087

                     Troebelheid na straat 13  Troebelheid na straat 14  \
2016-07-01 00:00:00                  0.324534                  0.413743
2016-07-01 00:15:00                  0.329130                  0.414927

                     Troebelheid na straat 15  Troebelheid na straat 16  \
2016-07-01 00:00:00                  0.323942                       NaN
2016-07-01 00:15:00                  0.327310                       NaN

                     Troebelheid na ZF serie 30  Troebelheid na ZF serie 40A  \
2016-07-01 00:00:00                    0.029203                     0.027453
2016-07-01 00:15:00                    0.029203                     0.027397

                     ...   runtime_filter_54  time_between_flushes_f55  \
2016-07-01 00:00:00  ...                 NaT                       NaN
```

```
2016-07-01 00:15:00  ...                NaT                       NaN

                     datetime_of_previous_flush_f55  runtime_filter_55  \
2016-07-01 00:00:00                             NaT                NaT
2016-07-01 00:15:00                             NaT                NaT

                     time_between_flushes_f56  datetime_of_previous_flush_f56  \
2016-07-01 00:00:00                       NaN                             NaT
2016-07-01 00:15:00                       NaN                             NaT

                     runtime_filter_56  n_active_filters  flow_per_filter  \
2016-07-01 00:00:00                NaT                 4      1408.340698
2016-07-01 00:15:00                NaT                 4      1408.340698

                     week
2016-07-01 00:00:00    26
2016-07-01 00:15:00    26

[2 rows x 142 columns]
```

Reorganize the dataframe such that a row contains data of a particular rapid sand filter

As you can see, the dataframe `df` has the independent time variable in its index, i.e. each observation at some time instant is ordered along the rows.

We reorganize the data such that each row will correspond to a particular filter. This is done as follows: for every filter, the function `stackFiltersKeepOnlyOwnMeasurement(df)` selects which columns of `df` belong to that filter (including the turbidity after the filter). Then it removes all the columns belonging to other filters and it adds a column `filter_number` containing the particular filter number to `df`. Finally, it removes the filter number from the column labels. This procedure is executed for data belonging to a particular (rapid sand) filter and stored in a seperate dataframe. Afterwards, we are able to concatenate (stack) each dataframe for each filter with each other, using inner joins.

In [9]:
```
print('Shape of data before stacking:', df.shape)
df = stackFiltersKeepOnlyOwnMeasurement(df)
print('Shape of data after stacking:', df.shape)
```

```
Shape of data before stacking: (70081, 142)
31
32
33
34
35
36
41
42
43
44
45
46
51
52
53
54
55
56
Shape of data after stacking: (1261458, 38)
```

Interpolate over time

Then missing values are interpolated over time by the function `interpolate`, this may take some time. Interpolation is only done if the number of adjacent missing values is less than the value set by `limit` (default value: 10).

In [10]:
```
df_interp = df.interpolate(method='time', limit=10)
```

We now average all relevant quantities over the filter runtimes and drop some unneeded columns:

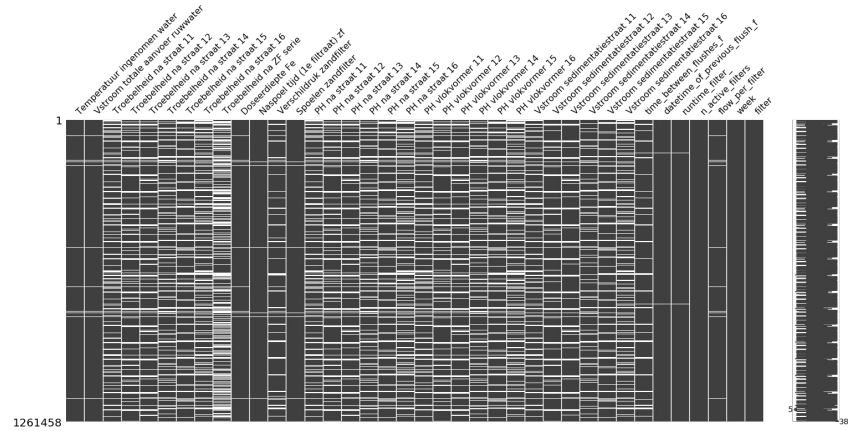
In [11]:
```
col_drop = ['Vstroom sedimentatiestraat',
                           'Troebelheid na straat',
                           'PH na straat',
                           'PH vlokvormer',
                           'Spoelen z',
                           'Naspoel tijd']
df_runtime = getQuantitiesPerRuntimeOfConcatedFilters(df=df_interp,
                                                      min_number_of_timesteps=5
, columns_to_drop=
                                                      col_drop)
```

Check the original dataframe `df` and the processed dataframe `df_runtime`

Let's check how many missing and unknown datapoints dataframe `df` contained before pre-processing. We do this by using a heatmap, where each datapoint is shown in black, and each missing or unknown datapoint is shown in white.

In [12]:
```
missingno.matrix(df)
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0xa01310e7f0>



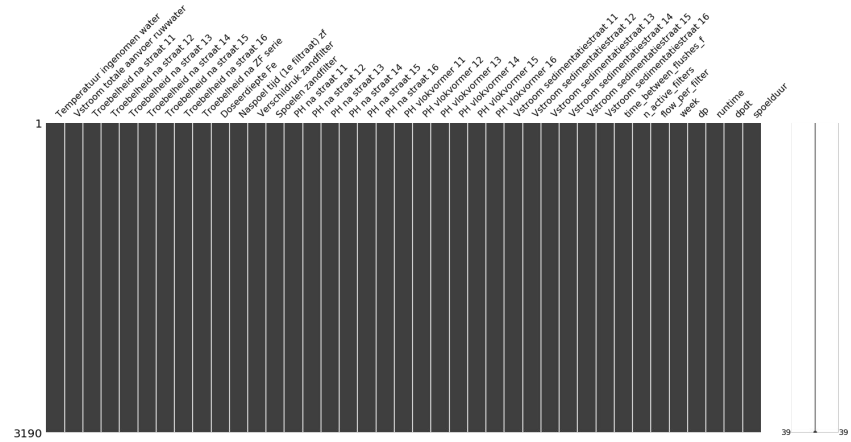
In [13]:
```
print('Number of data points & columns in the interpolated data:', df_interp.shape)
print('Number of filter runtimes:', df_runtime.shape[0])
#print('Columns (features) in data:', df_runtime.columns.tolist())
```

```
Number of data points & columns in the interpolated data: (1261458, 38)
Number of filter runtimes: 3190
```

Now let's check the missing data heatmap of `df_runtime`

In [14]:
```
missingno.matrix(df_runtime)
```

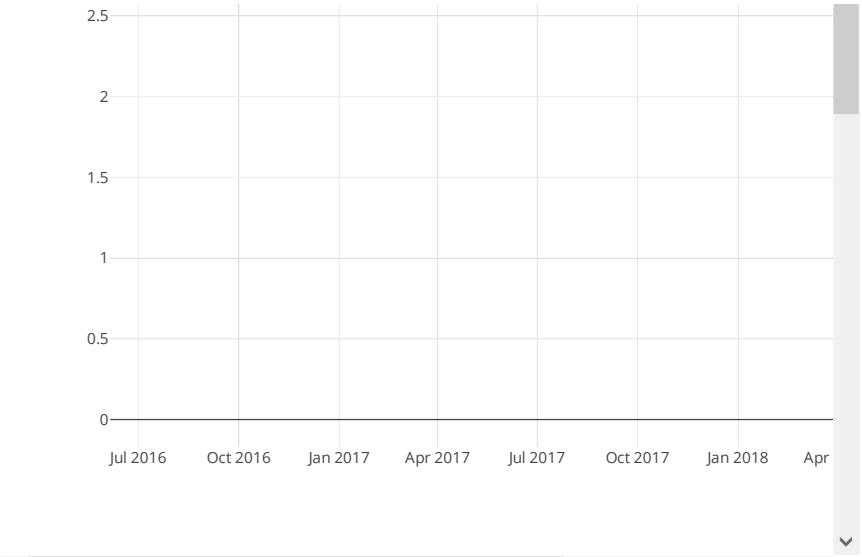
Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0xa00c738320>



Plot pressure drop

We check the pressure drop visually, i.e. we plot `Verschildruk zandfilter` over the complete sampling period. Note that not all runtimes are taken into account.

In [15]:
```
par_to_plot = 'Verschildruk zandfilter' #'Troebelheid na ZF serie' # 'Verschildruk zandfilter'
index_filter = 31
columns_to_plot = [par_to_plot + ' ' + str(index_filter)]
df_idx = df_runtime.xs(index_filter, level='filter')
df_plot = df_interp[df_interp['filter']==index_filter]
traces=[go.Scattergl(dict(x=df_plot.index, y=df_plot['Verschildruk zandfilter'], name='dp')),
        go.Scattergl(dict(x=df_idx.index, y=df_idx['dp'], name='dp averaged', mode='markers'))]
layout = go.Layout(dict(xaxis=dict(title='date'),
                        yaxis=dict(title='dp'),
                        width = w_fig*1.5, height = h_fig*1.5))
py.iplot(go.Figure(data=traces, layout=layout ))
```



Preliminary data analysis

In the next sections, we explore the dataset `df_runtime` and see whether there exist correlation between different features (quantities).

First, we list all the features of `df_runtime` (here, a feature corresponds to a column in `df_runtime`).

In [16]:
```
print(list(df_runtime.columns))
```

```
['Temperatuur ingenomen water', 'Vstroom totale aanvoer ruwwater', 'Troebelheid na
 straat 11', 'Troebelheid na straat 12', 'Troebelheid na straat 13', 'Troebelheid
na straat 14', 'Troebelheid na straat 15', 'Troebelheid na straat 16', 'Troebelhei
d na ZF serie', 'Doseerdiepte Fe', 'Naspoel tijd (1e filtraat) zf', 'Verschildruk
zandfilter', 'Spoelen zandfilter', 'PH na straat 11', 'PH na straat 12', 'PH na st
raat 13', 'PH na straat 14', 'PH na straat 15', 'PH na straat 16', 'PH vlokvormer
11', 'PH vlokvormer 12', 'PH vlokvormer 13', 'PH vlokvormer 14', 'PH vlokvormer 15
', 'PH vlokvormer 16', 'Vstroom sedimentatiestraat 11', 'Vstroom sedimentatiestraa
t 12', 'Vstroom sedimentatiestraat 13', 'Vstroom sedimentatiestraat 14', 'Vstroom
sedimentatiestraat 15', 'Vstroom sedimentatiestraat 16', 'time_between_flushes_f',
 'n_active_filters', 'flow_per_filter', 'week', 'dp', 'runtime', 'dpdt', 'spoelduu
r']
```

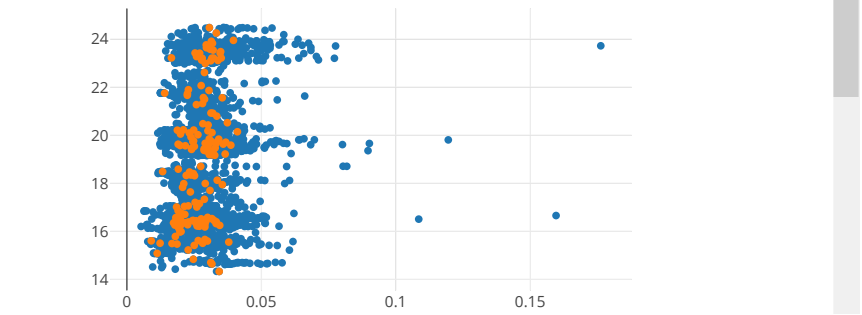
Plot variable averages over filter runtime

Let's have a look on the data first. Plot for all filters and for a specific filter, the averages over the filter runtime. Here we choose to plot on the x axis the feature `dpdt`, on the y axis `Doseerdiepte Fe` and we are interested in filter 31 (`index_filter`).

In [17]:
```
x_col = 'dpdt' # spoelduur
y_col = 'Doseerdiepte Fe' #'Vstroom totale aanvoer ruwwater'
index_filter = 31

_df = df_runtime.xs(index_filter, level='filter')
_df.sort_index(inplace=True)

traces=[go.Scatter(dict(x=df_runtime[x_col], y=df_runtime[y_col], name='all fil
ters', mode='markers')),
        go.Scatter(dict(x=_df[x_col], y=_df[y_col], name='filter '+str(index_fil
ter), mode='markers')),]
layout = go.Layout(dict(xaxis=dict(title=x_col),
                        yaxis=dict(title=y_col),
                        width = w_fig, height = h_fig))
py.iplot(go.Figure(data=traces, layout=layout))
```



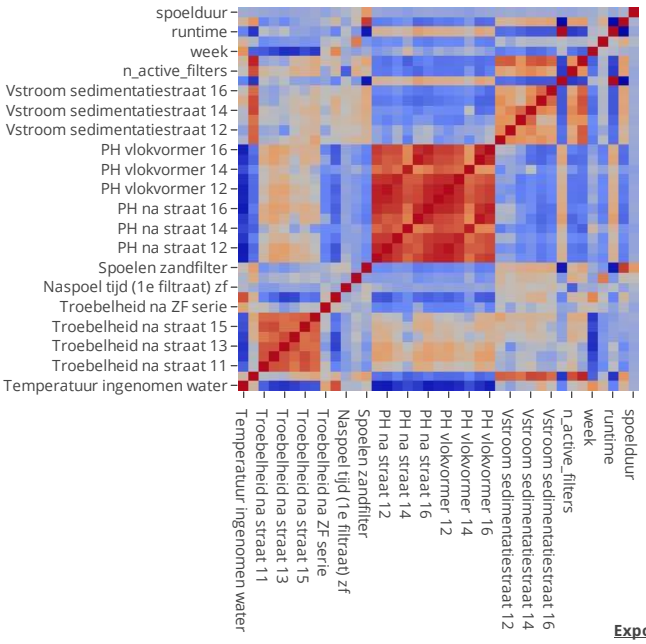
Export to plot.ly »

Explore if there is correlation between the features

We determine the correlation between all parameters and make a (Pearson) correlation heatmap of it.

In [18]:
```
# Determine correlations
corr = df_runtime.corr()

# Make heatmap
traces = [go.Heatmap(dict(z=corr.values,
                   x=corr.columns.values,
                   y=corr.columns.values))]
layout = go.Layout(dict(xaxis=dict(automargin=True),
                        yaxis=dict(automargin=True),
                       width = w_fig, height = w_fig))
py.iplot(go.Figure(data=traces, layout=layout))
```



Export to plot.ly »

We now order the correlations with pressure increase (dpdt) from high to low absolute value. Note that some variables with high correlations are a direct function of pressure increase, for example `runtime` and `time_between_flushes`. These variables need to be removed if we start to do machine learning.

In [19]:
```
corr['dpdt'].abs().sort_values(ascending=False)
```

Out[19]:

dpdt	1.000000
Spoelen zandfilter	0.803733
runtime	0.782982
time_between_flushes_f	0.782982
Vstroom totale aanvoer ruwwater	0.435138
flow_per_filter	0.387800
n_active_filters	0.365795
Vstroom sedimentatiestraat 16	0.341660
Vstroom sedimentatiestraat 14	0.334371
PH na straat 15	0.322205
Vstroom sedimentatiestraat 15	0.322065
PH vlokvormer 15	0.303903
PH na straat 16	0.292457
PH vlokvormer 16	0.283961
Vstroom sedimentatiestraat 13	0.279212
dp	0.269030
PH vlokvormer 12	0.237718
Vstroom sedimentatiestraat 11	0.236950
PH vlokvormer 11	0.221428
Temperatuur ingenomen water	0.215419
PH na straat 11	0.214602
Vstroom sedimentatiestraat 12	0.207103
PH na straat 12	0.198504
Doseerdiepte Fe	0.195444
PH vlokvormer 13	0.191067
Troebelheid na straat 11	0.166841
Troebelheid na straat 13	0.150411
Verschildruk zandfilter	0.149301
PH na straat 13	0.146015

```
PH na straat 14                    0.108785
PH vlokvormer 14                   0.084845
Troebelheid na straat 12           0.073030
Troebelheid na ZF serie            0.060573
Troebelheid na straat 14           0.056006
spoelduur                          0.042880
week                               0.034721
Naspoel tijd (1e filtraat) zf      0.027402
Troebelheid na straat 15           0.011582
Troebelheid na straat 16           0.001537
Name: dpdt, dtype: float64
```

Machine learning

After the data has been filtered and cleaned up, average values during the runtime were taken. Those average values will be used in the machine learning algorithms to find a regression relation between the *operational parameters* and the target feature *pressure increase*.

Machine learning consists of several steps:

- feature selection
- splitting the dataset in a training and a development test set
- normalize the data
- model training and validation
- model evaluation.

Feature selection

For the regression model, features need to be selected and a target column need to be chosen. As a target, `dpdt` is used. The columns (features) that have high co-correlation with other features (e.g. `time_between_flushes`, `runtime`, ...) are removed. Each remaining column is then used as a feature.

Notice that we could choose `runtime` as a target feature, this will train a model that predicts the runtime. However, the runtimes are maximized at 80 hours, while the maximum allowed pressure is often not yet reached by then. In those cases, the runtime is not a good predictor for clogging of the filter. We avoid this problem by using `dpdt`.

In [20]:
```
# Reset index
df_runtime = df_runtime.reset_index().set_index('datetime_of_previous_flush_f')

# Feature selection
_df = df_runtime.copy()
target_col = 'dpdt'
cols_to_keep = list(set(_df.columns)
                    - set([target_col])
                    - set([_ for _ in _df.columns if 'dp' in _.lower()])
                    - set([_ for _ in _df.columns if 'time_between_flushes_f' i
n _.lower()])
                    - set([_ for _ in _df.columns if 'runtime' in _.lower()])
                    - set([_ for _ in _df.columns if 'spoel' in _.lower()])
                    - set([_ for _ in _df.columns if 'verschildruk ' in _.lower
()])
                   )

_df.dropna(how='any', axis=1, inplace=True)
target = _df.loc[:, target_col]
features = _df.loc[:, cols_to_keep]
```

Split the dataset

The dataset will be splitted into a training set, to train the model, and a test set, to validate the trained model. This will be done randomly, whereby 20% of the data will be used as test data.

In [21]:
```
X_train, X_test, y_train, y_test = model_selection.train_test_split(features, t
arget.values, test_size=0.2)
print('Shape of training set:', X_train.shape)
print('Shape of test set:', X_test.shape)
```

```
Shape of training set: (2552, 32)
Shape of test set: (638, 32)
```

Normalize data

In regressing data with different features (with machine learning), we often encounter that each feature has its own variance, maximum and minimum values which can differ greatly with the variance, maximum and minimum values of another feature. To circumvent regressing a machine learning model that only selects features with high minima and maxima, the data is normalized by removing the mean and scaling to unit variance. The feature is then standardized to a normal score (hence, the term `StandardScaler`).

Hence, we scale the train set and test set with a `StandardScaler`.

In [22]:
```
# Make scalers for X and y
scaler_X = preprocessing.StandardScaler().fit(X_train)
scaler_y = preprocessing.StandardScaler().fit(y_train.reshape(-1, 1))
```

```
# Apply scalers
X_train = scaler_X.transform(X_train)
X_test = scaler_X.transform(X_test)
y_train = scaler_y.transform(y_train.reshape(-1, 1)).ravel()
y_test = scaler_y.transform(y_test.reshape(-1, 1)).ravel()
```

Train regression model

Selection of machine learning model algorithms

In the following, we train two model types which we have found to have good performance in previous data mining projects, namely:

a. Gradient boosting regression

b. Neural Networks.

At first, we use default (hyper)parameter model settings to tune the model training. For training of the neural networks, we use two different toolkit implementations: `scikit-learn` and `KERAS`. `KERAS` is a high level programming interface for the `TensorFlow` deep learning toolkit developed by Google.

(a) Gradient boosting regression

First a gradient boosting regression is chosen to predict `dpdt` using standard tuning parameters for the regression model.

In [23]:
```
# Initialize regressor
params = {'n_estimators': 1000, 'max_depth': 4, 'min_samples_split': 2,
          'learning_rate': 0.01, 'loss': 'ls'}
gradbm = GradientBoostingRegressor(**params)

# Apply regression
gradbm.fit(X_train, y_train)
```

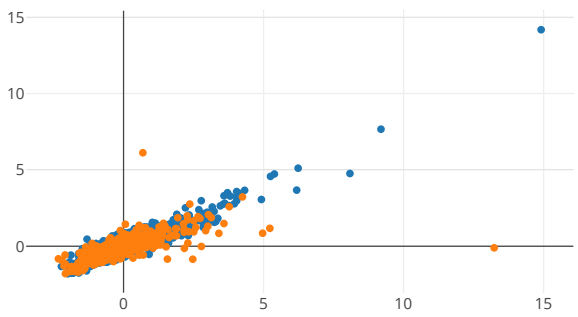
Out[23]:
```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
             learning_rate=0.01, loss='ls', max_depth=4, max_features=None,
             max_leaf_nodes=None, min_impurity_decrease=0.0,
             min_impurity_split=None, min_samples_leaf=1,
             min_samples_split=2, min_weight_fraction_leaf=0.0,
             n_estimators=1000, presort='auto', random_state=None,
             subsample=1.0, verbose=0, warm_start=False)
```

(a) Plot performance of gradient boosting regression model

The performance is plotted as predicted values versus measured values for both the training and test data. A perfect match would give a 1:1 line. Also, the scores (R^2) of the training and test prediction are shown.

In [24]:
```
traces = [go.Scatter(dict(x=y_train, y=gradbm.predict(X_train), mode='markers',
 name='train')),
          go.Scatter(dict(x=y_test, y=gradbm.predict(X_test), mode='markers', n
ame='test'))]
layout = go.Layout(dict(width = w_fig, height = h_fig))

py.iplot(go.Figure(data=traces, layout=layout))
print('R2 score of train set:', gradbm.score(X_train, y_train))
print('R2 score of test set:', gradbm.score(X_test, y_test))
```



Export to plot.ly »

```
R2 score of train set: 0.854027514664644
R2 score of test set: 0.46009756837266375
```

Notice that the R^2 score of the training set is not bad at all with a value above 0.8. However, we see that the score of the test set is rather low. Let's check if we can improve both scores by an optimized

hyperparameter set.

(a) Hyperparameter optimization of gradient boosting regression

The gradient boosting regression model gives already good results (R^2 > 0.8). A better result could be obtained by optimizing the hyperparameters used in the regression model. Because many iterations are needed, this may take a while.

In [25]:
```
model      = GradientBoostingRegressor()
n_iter     = 100
param_dist = {"max_depth": [3, 5, None],
              "max_features": sp_randint(1, 11),
              "min_samples_split": sp_randint(2, 11),
              "n_estimators": [10, 100, 500, 1000, 1500],
              "learning_rate": [0.001, 0.005, 0.01, 0.05, 0.1, 0.2]}

gs = model_selection.RandomizedSearchCV(estimator = model, n_iter = n_iter, par
am_distributions = param_dist,
                                        scoring = 'r2', cv = 5, n_j
obs = -1, verbose=1)

gs.fit(X_train, np.ravel(y_train))

model_best = gs.best_estimator_
predictions_gdbm = model_best.predict(X_test)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   26.9s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:  2.8min
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:  5.9min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:  7.3min finished
```

(a) Plot performance of optimized gradient boosting regression model

Let's see the performance after hyperparameter optimization.

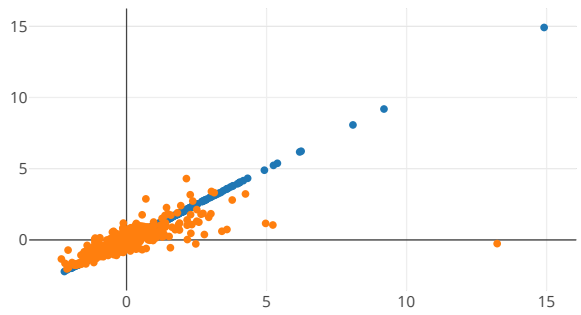
In [26]:
```
print('Optimal hyperparameters:')
cv_best = gs.best_params_
for key, item in cv_best.items():
      print(key + ' : ' + str(item))

traces = [go.Scatter(dict(x=y_train, y=model_best.predict(X_train), mode='marke
rs', name='train')),
          go.Scatter(dict(x=y_test, y=model_best.predict(X_test), mode='markers
', name='test'))]
layout = go.Layout(dict(width = w_fig, height = h_fig))

py.iplot(go.Figure(data=traces, layout=layout))

print('R2 score of train set:', model_best.score(X_train, y_train))
print('R2 score of test set:', model_best.score(X_test, y_test))
```

```
Optimal hyperparameters:
learning_rate : 0.1
max_depth : 5
max_features : 9
min_samples_split : 4
n_estimators : 1500
```



Export to plot.ly »

```
R2 score of train set: 0.9998015270302519
R2 score of test set: 0.5214919678676502
```

Notice that both scores are improved, but still, the test score is rather low. This can mean that either:

- the model over-fits the data, i.e. the amount of features and/or the complexity of the model is too high compared to the number of data observations.

- the train and test set do not have an equal distribution. This seems unlikely since we have used K-fold cross validation.

It seems that the first reason is causing the low test score, since we have got quite an amount of points in the test data. Ways to remediate over-fitting is regularization or feature reduction. Since this case is intended for demonstration purposes, these methods go beyond the current scope.

(b) Train neural network model using `scikit-learn`

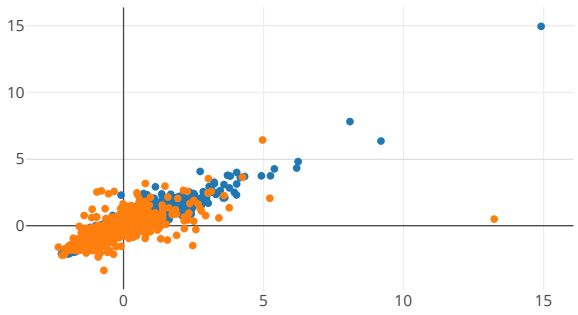
An alternative group of regression models are neural networks. First, the standard Python implementation in `scikit-learn` is used.

In [27]:
```
# Initialize neural network regressor
params = {'hidden_layer_sizes': (64,), 'solver': 'lbfgs','learning_rate_init':
0.001, 'alpha': 0.0001, 'max_iter': 1000}
nn     = MLPRegressor(**params)

# Fit neural network
nn.fit(X_train, y_train)

# Plot performance
traces = [go.Scatter(dict(x=y_train, y=nn.predict(X_train), mode='markers', nam
e='train')),
          go.Scatter(dict(x=y_test, y=nn.predict(X_test), mode='markers', name=
'test'))]
layout = go.Layout(dict(width = w_fig, height = h_fig))

py.iplot(go.Figure(data=traces, layout=layout))
print('R2 score of train set:', nn.score(X_train, y_train))
print('R2 score of train set:', nn.score(X_test, y_test))
```



Export to plot.ly »

```
R2 score of train set: 0.8904366529132673
R2 score of train set: 0.2975402451487785
```

(b) Train deep learning model using `keras`

The `scikit-learn` neural network does not give a better performance than the gradient boosting regression model. As a next step, deep learning algorithms are applied. These are neural networks with more layers (deeper) and may result in a better performance, especially for large data sets. In case large data sets are available and deep networks can be trained, the problem at hand is called a deep learning problem.

Please note that the deep learning model may take some time. The computation time largely depends on the number of epochs. One epoch consists of one full training cycle on the training set.

In [28]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras import backend as K

def coeff_determination(y_true, y_pred):
    SS_res =  K.sum(K.square( y_true-y_pred ))
    SS_tot = K.sum(K.square( y_true - K.mean(y_true) ) )
    return ( 1 - SS_res/(SS_tot + K.epsilon()) )

# generate regression dataset
# define and fit the final model
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mse', optimizer='adam') #, metrics=[coeff_determination])
model.fit(X_train, y_train, epochs=1000, verbose=0, validation_data = (X_test,
y_test)) #, validation_split=0.2)
```

```
Using TensorFlow backend.
```

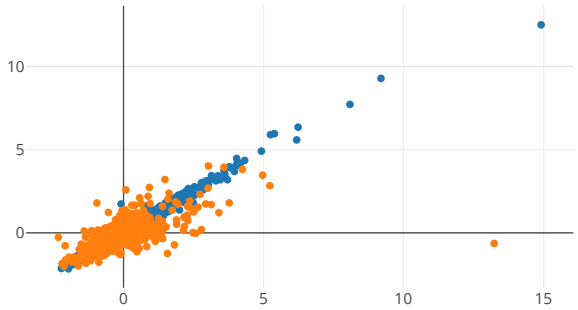
Out[28]: <keras.callbacks.History at 0xa00d54dc18>

In [29]:
```
from sklearn.metrics import r2_score

traces = [go.Scatter(dict(x=y_train, y=model.predict(X_train)[:,0], mode='markers', name='train')),
          go.Scatter(dict(x=y_test, y=model.predict(X_test)[:,0], mode='markers', name='test'))]
layout = go.Layout(dict(width = w_fig, height = h_fig))

py.iplot(go.Figure(data=traces, layout=layout))

print('R2 score of train set:', r2_score(model.predict(X_train)[:,0], y_train))
print('R2 score of test set:', r2_score(model.predict(X_test)[:,0], y_test))
```



Export to plot.ly »

```
R2 score of train set: 0.9784766199434398
R2 score of test set: -0.013721009985332433
```

As we can see, the KERAS deep learning model does not really improve the test score. In fact, the over-fitting problem seems to be much worse.

Evaluation of machine learning

The gradient boosting regression showed a better performance than we have seen with the neural networks. Perhaps designing a better neural network (e.g. by changing hyperparameters, the number of layers, or otherwise...) may help in improving the performance.

For now, we continue with the gradient boosting regression model.

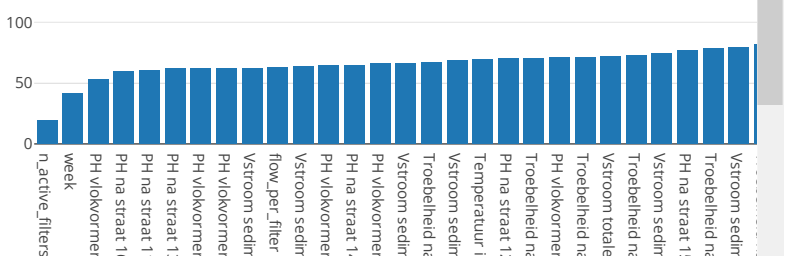
Feature importance

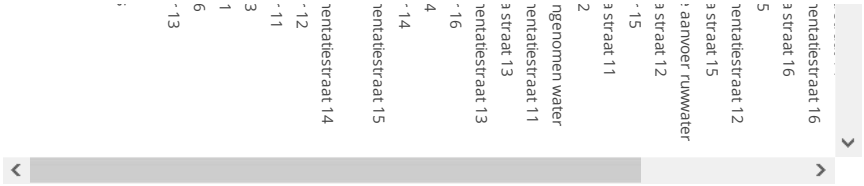
We plot the importance of the different variables in the model. The higher the value, the larger the influence on the predicted pressure increase.

In [30]:
```
# Plot feature importance
feature_importance = model_best.feature_importances_
# make importances relative to max importance
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)

traces = [go.Bar(x = np.array(cols_to_keep)[sorted_idx], y = feature_importance[sorted_idx])]
layout = go.Layout(dict(yaxis=dict(title='Relative Importance', automargin=True),
                        xaxis=dict(automargin=True),
                        width = 800, height = h_fig))

py.iplot(go.Figure(data=traces, layout=layout))
```





Notice that `Troebelheid` and `Doseerdiepte` are quite import in predicting `dpdt`, just what we would expect!

Use the model for prediction and sensitivity analysis

The trained model can be used to investigate the parameter space of the operational conditions. Furthermore we can do a sensitivity analysis of the features, this will be done in the next section.

Model predictions

In the following, the function `make_pred()` is defined that takes care of model predictions. With this function, the user supplies a value of a feature or variable in a dictionary `dict_var`, so that the model can make a prediction for `dpdt` using the given variables. If some variables are not given by the user, the model takes the average value from the data of that variable. The runtime is calculated from the dpdt by assuming that the maximum allowed pressure increase is 1.6 bar.

First, a function is made where a variable and a range of values for that variable can be given.

In [31]:
```
def make_pred(dict_var):
    len_var = len(next(iter(dict_var.values())))-1
    X_pred = df_runtime[cols_to_keep].mean().to_frame().transpose()
    if len_var > 0:
        X_pred = X_pred.append([X_pred]*len_var)
    for key, value in dict_var.items():
        if key in ['PH', 'Troebelheid', 'Vstroom']:
            names = [_ for _ in X_pred.columns if key in _]
        elif key in X_pred.columns:
            names = [key]
        else:
            names = []
        for name in names:
            X_pred[name] = value

    X_pred = scaler_X.transform(X_pred.get_values()) #.reshape(1,-1))

    y_pred = model_best.predict(X_pred)
    y_pred = scaler_y.inverse_transform(y_pred)
    return X_pred, y_pred

def plot_pred(X_pred, y_pred, dict_var):
    traces = [go.Scatter(dict(x=next(iter(dict_var.values())), y=1.6/y_pred, mo
de='markers'))]
    layout = go.Layout(dict(xaxis=dict(title=' '.join([key for (key,val) in dic
t_var.items()])),
                            yaxis=dict(title='runtime (h)'),
                            width = w_fig, height = h_fig))
    py.iplot(go.Figure(data=traces, layout=layout))
```

Let's check the average values first:

In [32]:
```
df_runtime.mean()
```

Out[32]:
```
filter                                43.998433
Temperatuur ingenomen water           14.065085
Vstroom totale aanvoer ruwwater     8035.214801
Troebelheid na straat 11               0.305239
Troebelheid na straat 12               0.373931
Troebelheid na straat 13               0.306712
Troebelheid na straat 14               0.403714
Troebelheid na straat 15               0.380506
Troebelheid na straat 16               0.398960
Troebelheid na ZF serie                0.058715
Doseerdiepte Fe                       18.791084
Naspoel tijd (1e filtraat) zf          6.678833
Verschildruk zandfilter                0.814396
Spoelen zandfilter                     0.009881
PH na straat 11                        7.981515
PH na straat 12                        7.966835
PH na straat 13                        8.008293
PH na straat 14                        7.916681
PH na straat 15                        7.875787
PH na straat 16                        7.838369
PH vlokvormer 11                       8.023241
PH vlokvormer 12                       8.030556
PH vlokvormer 13                       8.027486
PH vlokvormer 14                       7.919956
PH vlokvormer 15                       7.855363
PH vlokvormer 16                       7.824126
```

```
Vstroom sedimentatiestraat 11      1386.973329
Vstroom sedimentatiestraat 12      1378.731517
Vstroom sedimentatiestraat 13      1331.294475
Vstroom sedimentatiestraat 14      1360.705686
Vstroom sedimentatiestraat 15      1350.161160
Vstroom sedimentatiestraat 16      1353.437235
time_between_flushes_f               56.179624
n_active_filters                      5.732119
flow_per_filter                    1387.452427
week                                 25.301775
dp                                    1.456562
runtime                              56.179624
dpdt                                  0.028284
spoelduur                             2.038893
dtype: float64
```

Now, we set some features in `dict_var`. Then, we can make a prediction with `make_pred`.

In [33]:
```
dict_var = {'PH': [8.1],
            'Troebelheid': [0.1],
             'Temperatuur': [19.],
             'Doseerdiepte Fe': [20.],
             'filter': [34]}

X_pred, y_pred = make_pred(dict_var)

print('\nPredicted values:')
print('\ndp/dt [bar/h]:', y_pred[0])
print('runtime [h]:', 1.6/y_pred[0])
```

```
Predicted values:

dp/dt [bar/h]: 0.027087824931066293
runtime [h]: 59.06712717140324
```

Sensitivity analysis of features with respect to `runtime`

We assign a range of values to a particular feature (while the other parameters take the average value from the data), and plot the runtime against this parameter.

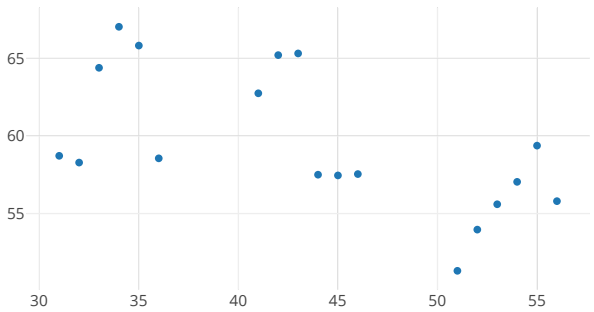
The following features will be analyzed:

- `filter` (filter number)
- `doseerdiepte Fe` (dosing amount of iron)
- `Vstroom` (flow rate)
- `Temperatuur ingenomen water` (temperature of raw water)
- `PH` (pH).

Filter number

Filter number has a large influence, this is also visible if the mean runtime per filter is calculated from the data. The influence of filter number may be related to the history of the filter (specific clogging, last time of sand replacement), differences in hydraulic configurations and loadings of the filters.

In [34]:
```
dict_var = {'filter': df_runtime['filter'].unique() }
X_pred, y_pred = make_pred(dict_var)
plot_pred(X_pred, y_pred, dict_var)
```



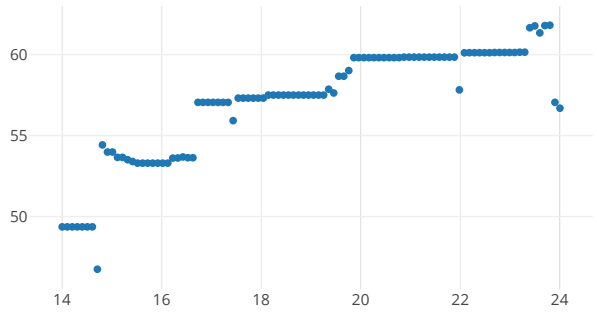
Export to plot.ly »

Dosing

The model predicts that no improvement of runtime is expected by dosing more than 20. By dosing around that optimum means a maximal runtime with a minimum of sludge.

In [35]:
```
dict_var = {'Doseerdiepte Fe': np.linspace(14, 24, 100)}

X_pred, y_pred = make_pred(dict_var)
plot_pred(X_pred, y_pred, dict_var)
```



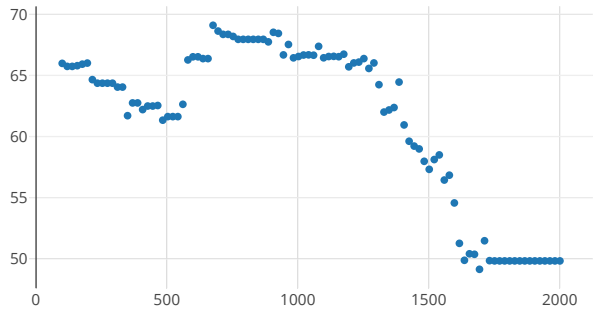
Export to plot.ly »

Flow rate

As expected, the runtime decreases with increasing flow rate (because the treated volume increases per time). However, this seems to be only the case for flows above 1000 and the influence seems to increase with increasing flow.

In [36]:
```
dict_var = {'Vstroom': np.linspace(100., 2000., 100)}

X_pred, y_pred = make_pred(dict_var)
plot_pred(X_pred, y_pred, dict_var)
```

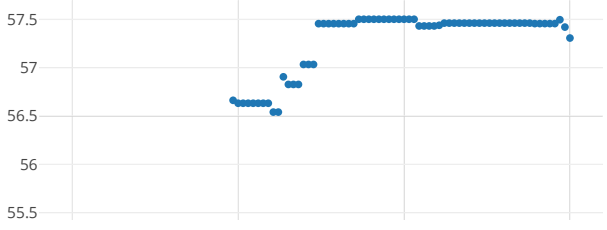


Export to plot.ly »

Temperatuur

The temperature has no significant influence on runtime; changing from 10 to 20 degrees Celsius only increases the runtime with 1.5 hour.

In [37]:
```
dict_var = {'Temperatuur ingenomen water': np.linspace(5, 20, 100)}
X_pred, y_pred = make_pred(dict_var)
plot_pred(X_pred, y_pred, dict_var)
```

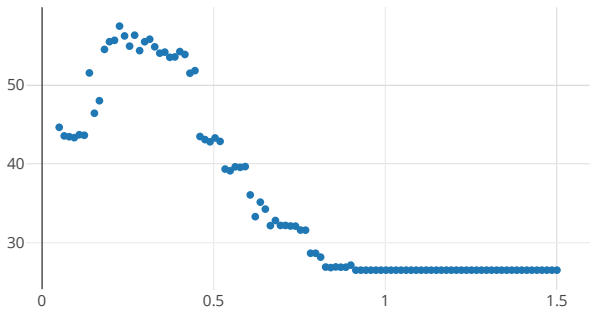


Export to plot.ly »

Troebelheid

As expected, increasing turbidity decreases runtime. On the contrary, at a turbidity of 0.22 a small increase of runtime is seen with increasing turbidity. This may be cause by the little amount of data at that turbidity level and thus less reliable model or a consequence of the other selected parameters like Fe-dosing.

In [38]:
```
dict_var = {'Troebelheid': np.linspace(0.05, 1.5, 100)}
X_pred, y_pred = make_pred(dict_var)
plot_pred(X_pred, y_pred, dict_var)
```

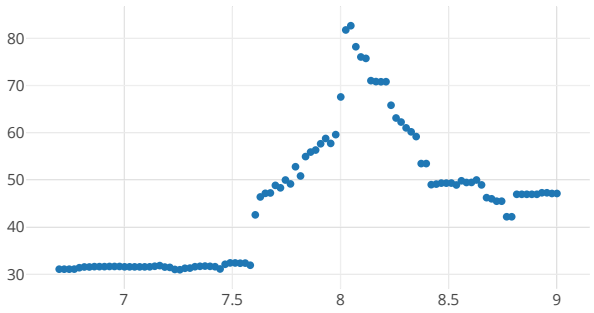


Export to plot.ly »

pH

A very sharp optimum is shown at a pH of 8; small deviation of this optimum decreases runtime considerably. It is comforting to see that the optimal pH is the same as the required pH enforced by legislation.

In [39]:
```
dict_var = {'PH': np.linspace(6.7, 9., 100)}
X_pred, y_pred = make_pred(dict_var)
plot_pred(X_pred, y_pred, dict_var)
```



Export to plot.ly »

Conclusions

A *big data* model is built that predicts the pressure increase in rapid sand filtration after coagulation. From operational variables, such as pH, turbidity, flow, an estimate of the pressure increase can be made. The pressure increase can be used to predict the runtime by assuming a maximum of 1.6 bar is allowed.

The trained model is used to visualize the sensitivity of the runtime against parameters like pH, turbidity

and Fe-dosing. Such an analysis may support the design of guidelines for optimal operating conditions.

Recommendations

To get better model scores, we recommend to research one or more of the following:

- add or aggregate features such that the predicted target feature is more accurate, e.g.:
  - during the winter, a flocculant aid is added. The amount of flocculant aid is not present as a feature in our current data set.
  - aggregate the pH's and turbidities of the different systems that run in parallel;
- Fix that the pressure criterium for back wash (1.6 bar) is not the same for all filters.
- Optimize the machine learning model performance by applying feature reduction techniques or regularization.

Recommendations for use by water companies:

- Use the model for predicting Fe-dosing.
- Further investigate the difference in predicted runtime by the different filters.
- Investigate the large influence of pH (especially around 8.0)