Rapport: INF161
Prosjekt: Prognostisering av Bergen Bysykkel

I dette prosjektet skal jeg lage et system som kan predikere hvor mange sykler som vil være tilgjengelig på en stasjon én time senere. Prosjektet er delt i tre deler.

Oppsummering: prosjektet viser at RandomForest gir best resultater for å predikere antall ledige sykler én time frem i tid. Resultatene tyder på at modellen fanger opp mønstre knyttet til tid, temperatur og totaltrafikk.

Generelt om mitt prosjekt:
Jeg har lagt til 5 filer:

Navn	Beskrivelse
data_exploration.ipynb	Utorsker data. Viser også hvordan pipelinen fungerer og de endelige stegene.
train.py	Består av to funksjoner: pipeline og modellering. Pipeline gir ferdig datasett og modellering gir ferdig modell.
predict.py	Bruker ferdig modell til å predikere antall sykler.
model.pkl	Ferdig trent modell. Blir brukt i predict.py.
hvordan_bruke.txt	Hvordan kjøre koden.

Del 1: Datautforsking og dataklargjøring
Skriver ned stegene fra data_exploration.ipynb. Utdyper der jeg mener det er nødvendig.

1.1 Lese inn csv.filene.

1.2 Ser hva datasettenne inneholder (describe)

	free_bikes
count	16077.000000
mean	7.669403
std	7.296786
min	0.000000
25%	2.000000
50%	5.000000
75%	12.000000
max	34.000000

Figur 1: stations

	temperature	precipitation	wind_speed
count	11575.000000	11599.000000	11597.000000
mean	7.054609	0.318036	11.352358
std	6.521379	0.743150	7.161594
min	-16.000000	0.000000	0.000000
25%	2.900000	0.000000	5.600000
50%	6.900000	0.000000	9.900000
75%	11.400000	0.200000	15.700000
max	27.900000	9.600000	40.600000

Figur 2: weather

	start_station_latitude	start_station_longitude	end_station_latitude	end_station_longitude
count	952601.000000	952601.000000	952601.000000	952601.000000
mean	60.388004	5.327178	60.388331	5.326808
std	0.009795	0.012697	0.009445	0.012171
min	60.341031	5.273174	60.341031	5.273174
25%	60.381512	5.320002	60.382255	5.320656
50%	60.388910	5.325714	60.389626	5.325714
75%	60.394550	5.333327	60.394550	5.333027
max	60.423343	5.361202	60.423343	5.361202

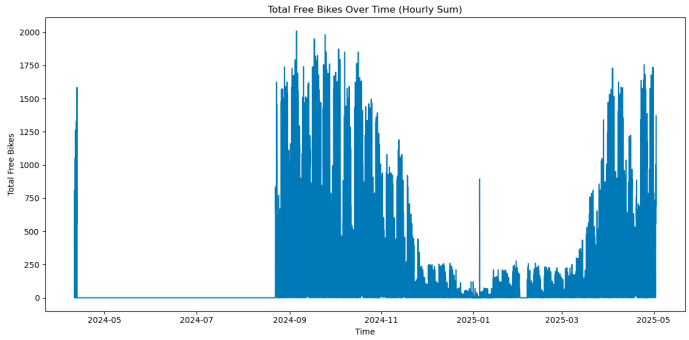
Figur 3: trips

1.2.2 Ser hvor store datasettene er.

1.2.3 Ser hvilken type data vi har.

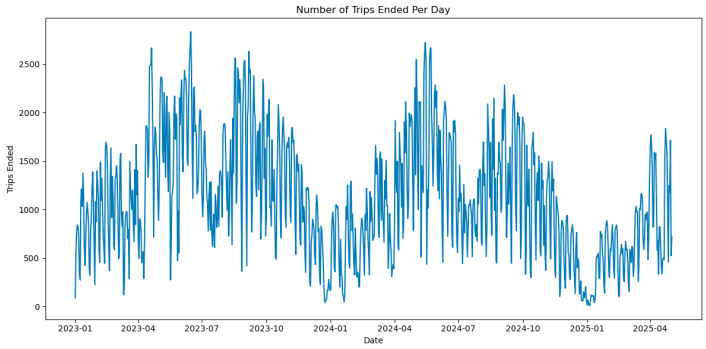
1.3 Visualisering.

Denne visualiseringen er gjort på hele datasettene. Dette er for å se om det er noen store mangler eller hull som burde tas hensyn til.



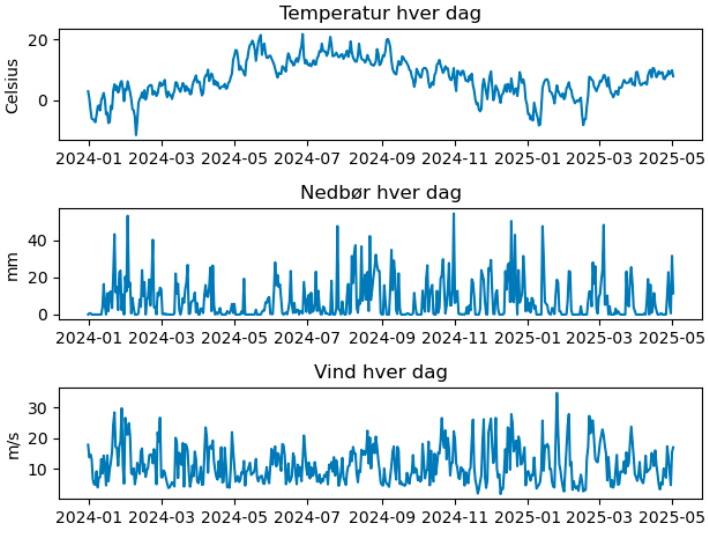
Figur 2: Visualisering av stations

Figur 1: Det er et stort hull i starten av datasettet. Hullet er såpass stort at å fylle det kan bli misvisende eller bare gi feil informasjon. Derfor har jeg valgt å fjerne dette.



Figur 3: Visualisering av trips

Figur 2: Ingen mangler.

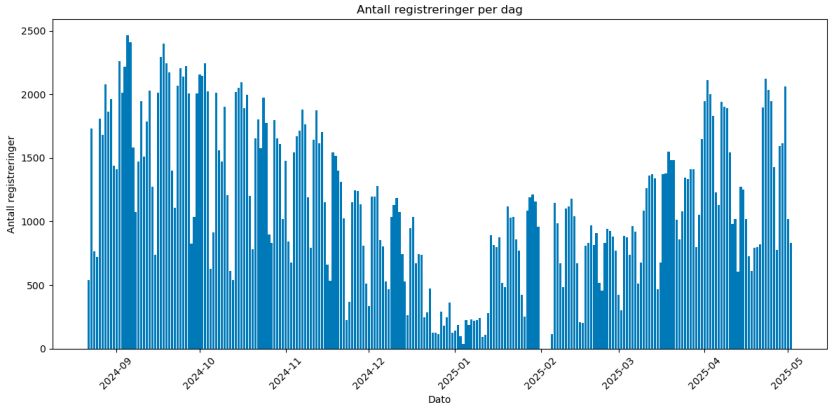


Figur 4: Visualisering av weather

Figur 3: Ingen mangler.

1.4 Fjerner hull i stastions

I stations er det er stort hull som går over flere måneder der det mangler data. Dette skyldes sannsynligvis tekniske årsaker eller lignende, som gjør at de manglende dataene er ikke-tilfeldig. Derfor kan man ikke fylle de med ffill ettersom det vil gi kunstige mønster. Derfor blir det beste å finne og fjerne hullet.



Figur 5: Stations etter fjerning av hull

1.5 Gjør om til tidspunkt og dato

Kolonnene med timestamp og lignende må gjøres om fra strings slik at man kan lettere bruke dem videre. I data_exploration er ikke tiden satt til norsk tid. Dette blir gjort først i de andre filene.

1.6 Deler i train og test

Datautforsking skal bare gjøres på treningsdata så deler i train og test. I datautforskingen er det ikke nødvendig å bruke val og test, så splitter ikke mer enn i 70% og 30% her. I tillegg splittes dataene på tid. Altså vil de første 70% av radene bli treningsdata, og de siste blir test. Dette gjøres slik:

```
stations_train, stations_test = train_test_split(stations_after_gap,
test_size=0.3, shuffle=False)
```

Alle visualiseringer herfra er gjort på bare treningsdata.

1.7 Ser på datasettene

```
Station train: (200614, 8)
Trips train: (666820, 8)
Weather train: (8194, 4)
```

Figur 6: Form på datasettene

1.7.2 Ser på første og siste observasjon

```
Stations:
2024-08-22 15:05:34+00:00
2025-05-02 15:49:25+00:00

Trips:
2023-01-01 04:33:19.884000+00:00
2025-05-02 15:52:52.760000+00:00

Weather:
2023-12-31 23:00:00+00:00
2025-05-02 16:00:00+00:00
```

Figur 7: Start- og sluttidspunkt på datasettene

1.7.3 Sjekker manglende verdier

- Stations: ingen manglende verdier.
- Trips:

	missing	percent
started_at	695	0.1
hour	695	0.1
date	695	0.1
ended_at	665	0.1

Figur 8: Manglede data for trips

- Weather:

	missing	percent
temperature	90	1.10
wind_speed	83	1.01
precipitation	75	0.92

Figur 9: Manglende data for weather

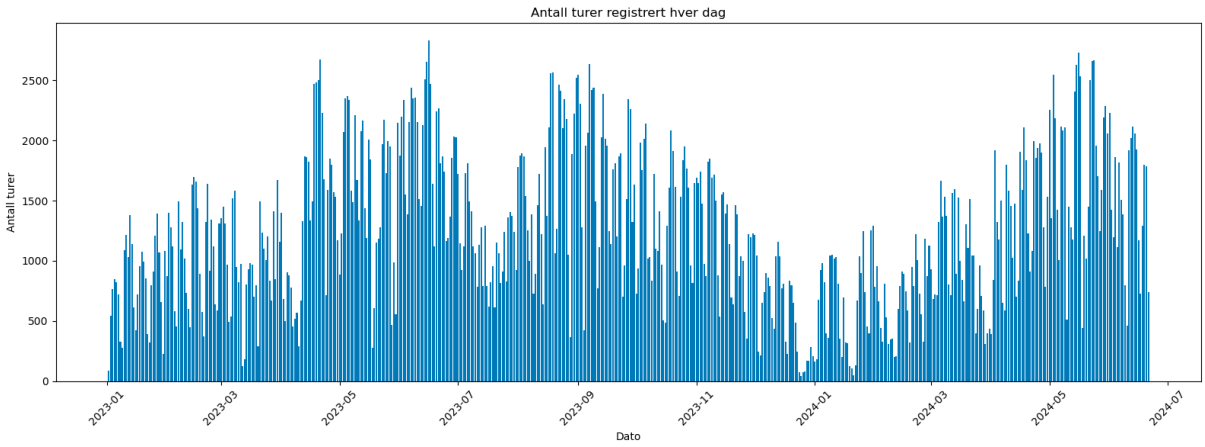
Vi ser her at trips er det største datasettet. Stations inneholder fremdeles registreringer fra alle stasjoner, altså ikke bare de vi skal predikere for, og kommer derfor til å bli mindre etter hvert. I tillegg ser man at trips har registreringer nesten et helt år lenger enn weather, og at stations har et enda kortere tidsrom. Det er en liten prosentdel med manglende data. Alt dette må bli tatt hensyn til senere.

1.8 Trips og weather

1.8.1 Lager total trips per time

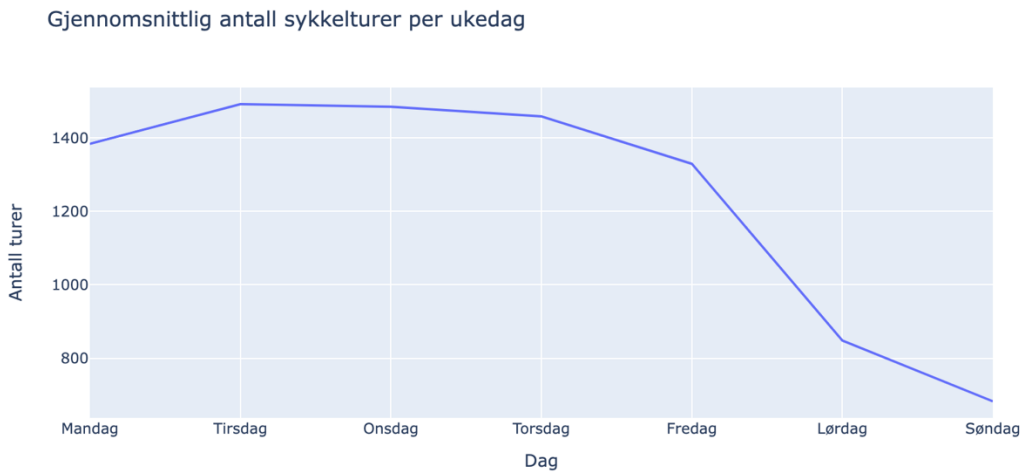
Lager et nytt datasett som heter trips_per_hour som har «hour» og «total_trips» som kolonner. Dette gjøres siden vi skal se på timevis og ikke hver eneste observasjon. Senere kommer til å gjøre dette på stations: om vi har observasjoner 16:38:53 på 5 sykler og 16:49:25 på 8 sykler, så skal registreringen klokken 17:00:00 stå som 8 sykler. Da bruker jeg ceil, og derfor gir det mening å gjøre det samme på resten av dataene.

1.8.2 Ser på hvor mange turer det er hver dag



Figur 10: Turer hver dag

Visualiseringen viser et tydelig dropp i antall turer rundt desember/januar, som gradvis går oppover mot våren. Dette kan ha med at det er kaldt ute og at færre folk ikke sykler like mye på vinteren som resten av året. I tillegg ser man at rundt juli/august går det også ned. En grunn til det kan være at det er mange som bruker bysyklene til jobb, men har sommerferie i dette tidsrommet. Med andre ord kan både temperatur og hvilken måned det er ha noe å si for hvor mange som sykler.



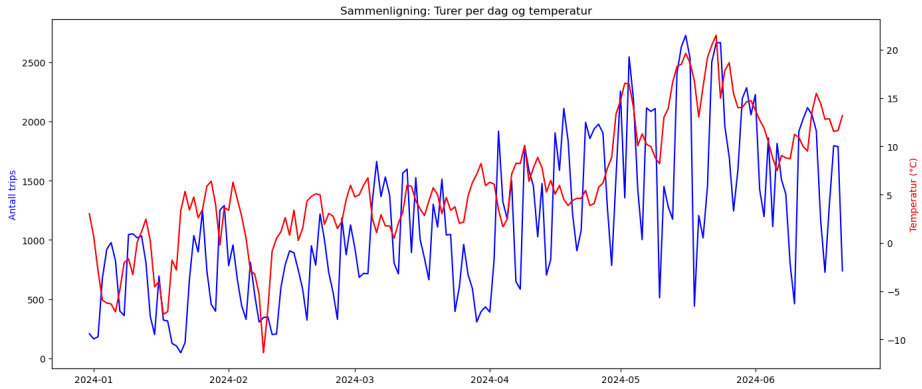
Figur 11: Oversikt over når flest folk sykler

I denne visualiseringen brukte jeg floor istedenfor ceil. Dette er fordi ceil i dette tilfellet ville forskjøvet alt med en dag. I figur 10 viser det at det er flest som sykler mandag-fredag, mens lørdag og søndag har en del mindre. Dette stemmer over ens med teorien om at mange bruker bysykler til og fra jobb.

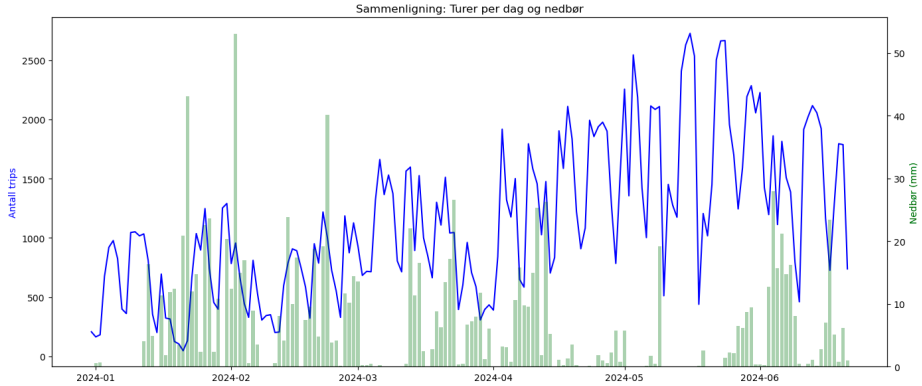
1.8.3 Lager en «hour» kolonne for weather

Samme som for trips, men her har jeg kolonnene «temperature», «precipitation» og «wind_speed». Brukte .agg() her istedenfor ceil fordi det er mer naturlig. Tar gjennomsnittet for temperatur og wind_speed for timen det gjelder, mens for nendbør tar jeg sum.

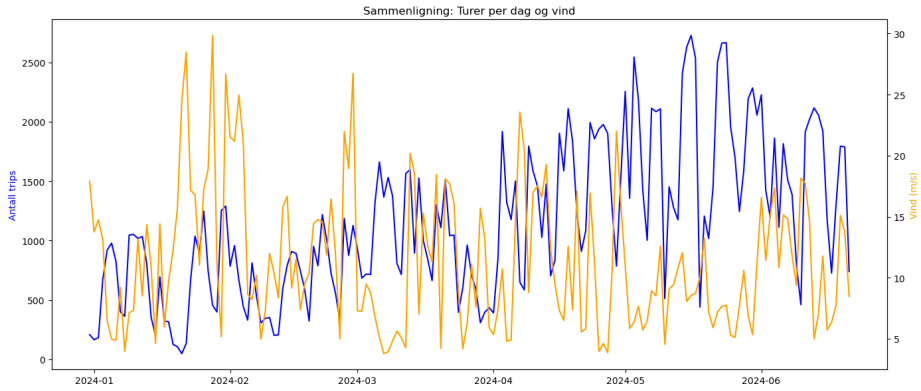
1.8.4 Sammenligning av weather og total_trips



Figur 12: Turer per dag og temperatur



Figur 13: Turer per dag og nedbør



Figur 14: Turer per dag og vind

I visualiseringene ser man at antall turer og temperatur har en positiv korrelasjon, mens antall turer og nedbør kan tyde på at har en negativ korrelasjon. Litt vanskeligere å se noe tydelig korrelasjon med vind.

1.9 Stations

For hver del av stations i data_exploration.ipynb har jeg vist hvordan datasettet ser ut underveis ved bruk av head(). Jeg har ikke limt det inn i rapporten, men det er mulig å se i notebooken.

1.9.1 Utvalgte stasjoner

Tar bare med de utvalgte stasjonene videre. Gikk fra 200 614 rader, til 29453 rader.

1.9.2 Lager en «hour» kolonne

1.9.3 Lager en kolonne som lagrer tidspunktet for siste måling generelt

I utskriften skal man ha med siste observasjon. Jeg valgte derfor å lage en egen kolonne som tar vare på siste tidspunktet til registreringen som gjelder for timen. Her er et eksempel:

	hour	last_obs
0	2024-08-22 16:00:00+00:00	2024-08-22 15:48:44+00:00
1	2024-08-22 17:00:00+00:00	2024-08-22 16:49:50+00:00
2	2024-08-22 18:00:00+00:00	2024-08-22 17:48:55+00:00
3	2024-08-22 19:00:00+00:00	2024-08-22 18:36:40+00:00
4	2024-08-22 20:00:00+00:00	2024-08-22 19:49:00+00:00

Figur 15: Eksempel fra last_obs

1.9.4 En rad per time og en kolonne per stasjon med siste registrering i timen

Koden til dette er:

```
stations_per_hour = (
    stations_filtered
    .sort_values("timestamp")
    .groupby(["hour", "station"])["free_bikes"].last()
    .unstack()
    .add_suffix("_free_bikes")
    .reset_index()
)
```

Først passer jeg på at alt blir sortert etter tid med sort_values. Med groupby får jeg den siste målingen av ledige sykler for hver stasjon i hver time. Ved unstack får jeg en kolonne per stasjon. Jeg hadde først fill_value = 0, men det gir ikke mening at om det var 8 sykler klokken

16, men at det mangler observasjoner 17 så skal antall ledige sykler plutselig være 0. Derfor fyller jeg inn for manglende verdier senere. Jeg kaller kolonnene «_free_bikes».
Jeg viser hva jeg har gjort her for jeg gjør lignende steg flere ganger i koden.

1.9.5 Får en rad for hver stasjon hver time
Istedenfor å ha 9 kolonner med «_free_bikes» gjør jeg om til å å ha én kolonne for free_bikes (forklaring under «Prøving og feiling»). I tillegg gjør jeg om til å ha en kolonne for hver stasjon med dummy-variable. Det vil si at for hver time er det 9 registreringer, én for hver stasjon. Dummy-variablene forteller derfor hvilken stasjon hver enkelt rad gjelder.

1.9.6 Lager et grid som fyller inn alle timene
Ser på første og siste målingen i datasettet og passer på at man får målinger hver eneste time mellom dette.

1.9.7 Fyller inn manglende verdier
Her bruker jeg forward-fill. Igjen hadde det ikke gitt mening å fylle inn med f.eks. 0. I disse dataene gir det mening om det er en del manglende registreringer på natten ettersom folk flest sover og ikke sykler da. Derfor vil det gjerne ikke være store endringer på disse tidspunktene og ffill kan da gi mening.

1.9.8 Legger til last_obs i datasettet
Merger på den felles kolonnen «hour» med hensyn på stations. Dvs. at om last_obs mangler noen verdier så vil det stå NaN istedenfor å droppe raden.

1.10 Slår sammen datasettene
Igjen bruker man merge med hensyn på stations ettersom det er det mest «komplette» datasettet.

Oversikt over kolonner før sammenslåing:

station	hour	free_bikes	«navn på stasjoner» x9	last_obs

hour	total_trips

hour	temperature	precipitation	wind_speed

Altså blir kolonne i det siste datasettet:

- hour
- station
- free_bikes
- navn på stasjon x9
- last_obs
- total_trips
- temperature
- precipitation
- wind_speed

1.10.2 Legger til ekstra kolonner
- Endrer navn til «hour» tilbake til «timestamp».
- Lager følgende kolonner:
    - hour: hvilken time det er i døgnet (0-23)
    - day_of_week: hvilken dag det er i uka (0-6)
    - month: hvilken måned det er (0-11)

1.10.3 Lager «target» kolonnen

Target kolonnen vil si antall sykler neste time. I funksjonen forskyver man antall ledige sykler med én time og gjør de verdiene om til target.

1.11 Diverse

1.11.1 Gjør om slik at det ikke er negative verdier

På grunn av at noen modeller ikke tar inn negative verdier, så forskyver jeg alle verdiene med den laveste verdien i datasettet. Med andre ord: om den laveste målte temperaturen er -9.3, så vil alle temperaturmålinger plusses med 9.3 slik at den laveste blir 0.

1.11.2 Rydder i kolonner

Endrer rekkefølgen av kolonnene og tar bare med de jeg ønsker å ha med:

- "timestamp",
- "last_obs",
- "hour",
- "day_of_week",
- "month",
- "station",
- "free_bikes",
- "Akvariet",
- "Damsgårdsveien 71",
- "Dreggsallmenningen Sør",
- "Florida Bybanestopp",
- "Grieghallen",
- "Høyteknologisenteret",
- "Møllendalsplass",
- "Studentboligene",
- "Torgallmenningen",
- "precipitation",
- "wind_speed",
- "temperature_shifted",
- "total_trips",
- "target"

1.12 Manglende data

1.12.1 Sjekker for tomme variable

```
timestamp                   0
last_obs                10764
hour                        0
day_of_week                 0
month                       0
station                     0
free_bikes                  0
Akvariet                    0
Damsgårdsveien 71           0
Dreggsallmenningen Sør      0
Florida Bybanestopp         0
Grieghallen                 0
Høyteknologisenteret        0
Møllendalsplass             0
Studentboligene             0
Torgallmenningen            0
precipitation           15876
wind_speed              16119
temperature_shifted     16119
total_trips             38916
target                      9
dtype: int64
```

Figur 16: Oversikt over tommme variable per kolonne

Fjerner ikke target som mangler ettersom vi predikerer på disse senere.

1.12.2 total_trips
Der det mangler turer setter jeg inn 0. Det hadde ikke gitt mening å ffille her ettersom det kunne gitt feil mønstre. Om det mangler registreringer en time kan det enkelt være for at det ikke er gjort noen turer, f.eks. på natten.

1.12.3 last_obs
Bruker ffill fordi da vet man alltid når den siste registreringen var.

1.12.4 precipitation
For manglende verdier om regn er det rimelig å anta at det ikke har regnet. Derfor fyller jeg inn med 0.

1.12.5 wind_speed og temperature_shifted
Bruker ffill.

Del 1 – train.py:
Alt dette (utenom datautforsking) er slått sammen til pipeline() i train.py.

Annerledes fra data_exploration:

- Setter til norsk tid. Bruker .dt.tz_convert("Europe/Oslo") for at den skal følge sommer og vinterstid.

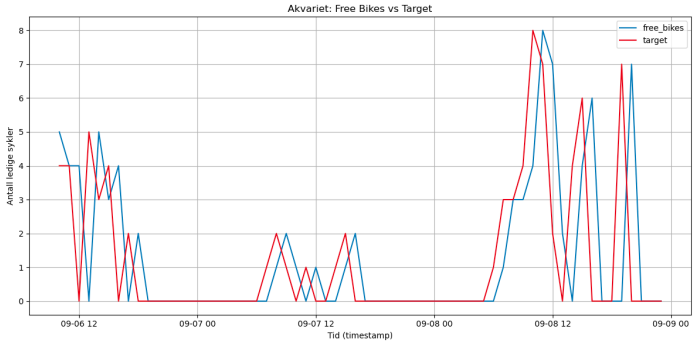
- Funksjonen tar inn stations_data som argument. Dette er fordi jeg på forhånd deler inn i train, val og test. Å dele inn alle tre dataettene på forhånd ville gitt store komplikasjoner senere. Dette er fordi de har forskjellig størrelse og forskjellige tidspunkt de starter på. Så å dele i 70, 15, 15 ville ikke blitt for samme tidspunkt på alle tre. Måten jeg løste det på var å bare dele opp stations. På den måten, når jeg merger med hensyn på stations så vil jeg uansett bare få inn de aktuelle målingene/tidspunktene fra de andre datasettene.

- Funksjonen returnerer df, som er det ferdige df som kan brukes senere. Men! Dette er fremdeles bare enten train, val eller test.

Jeg velger å bare dele opp stations i train, val og test fordi de tre rådatasettene ikke dekker nøyaktig samme tidsrom. Stations fungerer derfor som hoveddatasettet i pipelinen, og kolonen «hour» som felles nøkkel ved sammenslåing. Når jeg merger trips og weather med stations, vil de automatisk bare bidra med observasjoner som overlapper i tid. Dermed får trips og weather naturlig samme oppdeling i train, val og test osm stations uten at jeg trenger å splitte dem separat.

Del 2: Modellering og prediksjon

Sjekker target først:



Figur 17: Sammenligner target med free_bikes (Akvariet)

Ser om det er gjort riktig når jeg lager target-variablene. Alt skal være likt, men forskyvd en time. Dette stemmer.

Funksjonen modellering() bruker pipelinen til å lage train, val og test, for så å dele de inn i X_train og y_train, osv. Noen kolonner må droppes siden ikke alle modeller tar inn tekst og datoer.
Deretter tester jeg på disse modellene:

```
    models = {
        "Baseline" : DummyRegressor(strategy="mean"),
        "LinearRegression" : LinearRegression(),
        "kNN-3" : KNeighborsRegressor(n_neighbors=3),
        "kNN-10" : KNeighborsRegressor(n_neighbors=10),
        "kNN-50" : KNeighborsRegressor(n_neighbors=50),
        "DecisionTree" : DecisionTreeRegressor(random_state=42),
        "RandomForest" : RandomForestRegressor(random_state=42),
        "Lasso" : make_pipeline(PolynomialFeatures(degree=2), Lasso(alpha=0.01,
max_iter=5000)),
        "SVR" : make_pipeline(StandardScaler(), SVR(C=1.0, epsilon=0.2))
    }
```

Regresjon fordi i dette prosjektet skal man predikere kontinuerlige tall istedenfor noe kategorisk. Jeg har valgt å teste mange forskjellige modeller for å få et best mulig resultat.

Grunner:

- Baseline: enkel modell til å sammenligne.
- LinearRegression: vil prøve å finne lineære sammenhenger. Veldig enkel og rask. Litt det samme som baseline ettersom jeg ikke forventer at den skal være så god.
- Knn: Ser på nærmeste verdiene i datasettet. Tester i tillegg ulike k-verdier for å se hva som blir best.
- DecisionTree: ser på ikke-linære sammenhenger. Kan overtilpasse om datasettet er for lite.
- RandomForest: bruker mange decision-trees og er dermed mer stabil. Kan se på både linære og ikke-lineære sammenhenger. Kan se på feature-importance etterpå.
- Lasso: bra mot overtilpassing. Kan se på feature-importance etterpå.
- SVR: god på ikke-lineære data, men krever skalering og kan ta lang tid om datasettet er for stort.

Hyperparameter og lignende er tatt ut fra de obligatoriske innleveringene eller ved å se på modellene på scikit. Random state = 42 er for reproduserbarhet.

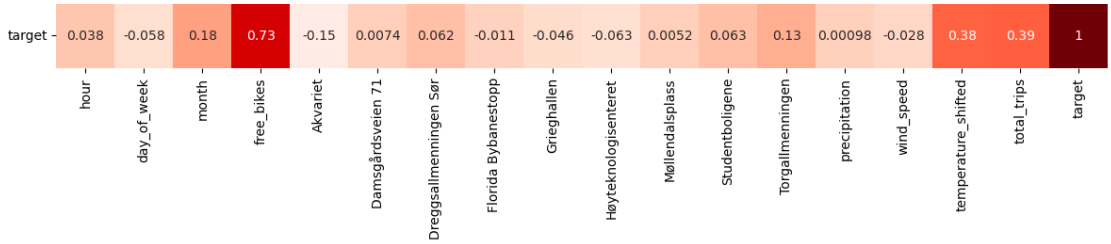
Etter dette trenes alle modellene på treningsdata for så å finne root mean squared error på både trening og validerinsdata. Deretter velges den beste modellen – altså den med lavest RMSE score. Til slutt testes den på testdata, der denne scoren blir lagret og brukt til å vise generaliseringsemne. Den beste modellen blir i tillegg lagret ved hjelp av pickle og brukt senere i predict.py.

Resultat: den beste modellen ble RandomForest.
Utskriften under er hentet fra train.py, men fjernet print i ettertid.

	train	val
RandomForest	1.374760	4.378270
Lasso	4.000796	5.577354
kNN-10	3.705708	5.617608
kNN-50	3.959979	5.659099
SVR	4.215156	5.755681
LinearRegression	4.145534	5.767632
kNN-3	3.152978	5.843889
DecisionTree	0.000000	6.180700
Baseline	6.240864	8.174350

Figur 18: RMSE fra modellering

Variabelutvinning:
I tillegg til å ta bort og legge til noen variable tidligere, brukte jeg en korrelasjonsmatrise for å se hvilke variabler som hadde sterkest sammenheng med target (hele matrisen ligger i data_exploration).



Figur 19: Korrelasjonsmatrise for det ferdige datasettet (treningsdata)

Jo høyere verdi (positiv eller negativ), jo større korrelasjon. Variablene som hører til hver enkelt stasjon skal være med uansett, og jeg ser bort fra dem.

- Høy korrelasjon: month, free_bikes, temperature_shifed og total_trips.
- Lav korrelasjon: hour, day_of_week, precipitation og wind_speed.

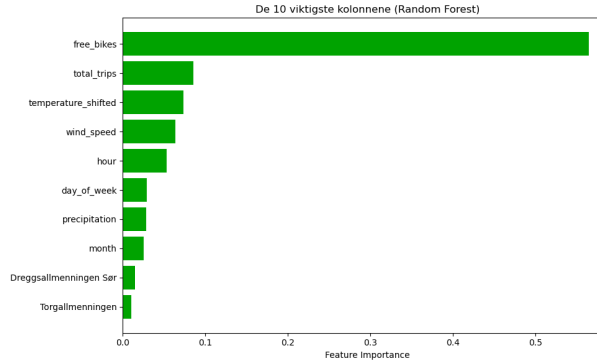
Jeg tester effekten av å gjerne de variablene som hadde lavest korrelasjon for å se om modellen ble påvirket.

Resultat (valideringsdata):

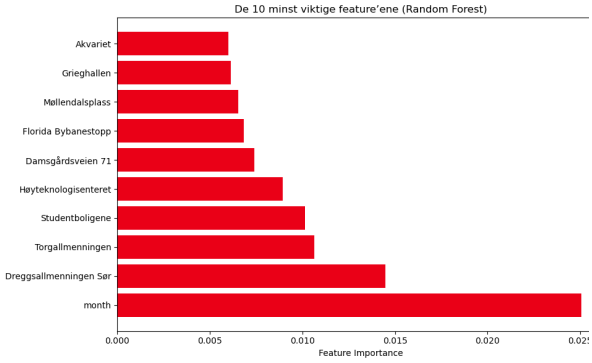
- Uten endring: RMSE = 2.731
- Med endring: RMSE = 2.803

Forskjellen er liten, men modellen presterer litt dårligere når disse fjernes.

Testing av variable:
Etter å ha funnet den beste modellen (RandomForest), undersøkte jeg feature importance for å se hvilke variabler modellen faktisk bruker mest.



Figur 20: Feature Importance med de 10 viktigste kolonnene



Figur 21: Feature importance med de 10 minst viktige kolonnene

Resultatet viser at også variabler med lav korrelasjon, som wind_speed og hour bidrar til modellen. Dette er mest trolig fordi RandomForest ser på ikke-linære sammenhenger. Jeg beholder derfor disse kolonnene. Variablene som bidrar minst er stasjonene, så de er ikke noe å gjøre med.

Imputasjon:
Vi jobber med tidsavhengige data som blir delt inn i trening, validering ogo testdata som respekterer rekkefølgen. Som tidligere sagt har jeg derfor brukt forward-fill på de fleste manglende verdier. Når jeg kommer til modelleringen er det ingen manglende verdier å ta hensyn til, og det blir derfor ikke brukt noen imputasjon i denne delen.

Forventet generaliseringsevne og modellvalg:

```
Beste modell: RandomForest
Forventet RMSE: 3.12
```

Figur 22: Resultat av modellering

Noe som er verdt å legge merke til er at val-rmse er høyere enn test-rmse. Dette trenger ikke bety mer enn at testdataene mulig er mer stabile enn val. Datasettene er tidsbestemt og ulike mønster i treningssettet kan stemme bedre over ens med test enn val.

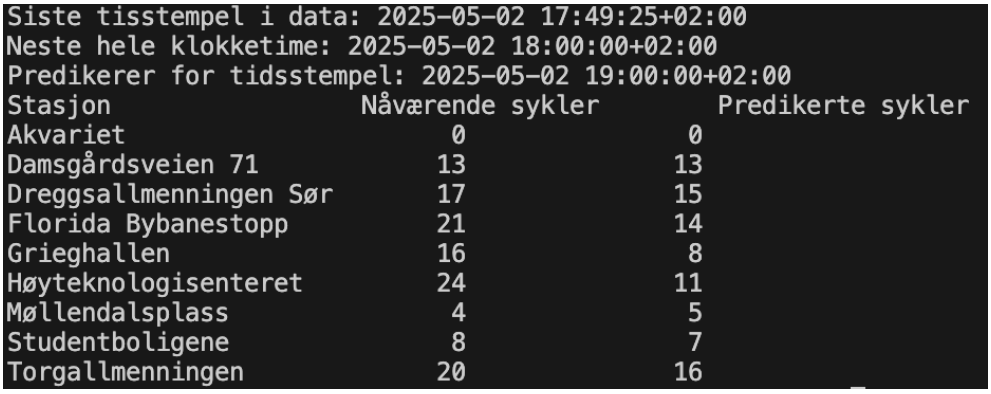
Del 3:Sammensetning og deployment

I filen predict.py henter man først inn den ferdig trente modellen, for så å lese inn stations og ta vekk hullet igjen. Denne gangen deles det ikke opp i train, val og test. Til slutt kjøres hele datasettet gjennom pipelinen.

Jeg gjør det sånn fordi måten jeg har laget pipelinen min skal ikke være avhengig av akkurat dette stations-datasettet. Den slipper å håndtere manglende perioder i selve pipelinen og gjør den mer effektiv. Det kan virke litt tungvint å fjerne hullet i både train.py og predict.py, men nå skal det skal være mulig å kjøre filene individuelt om ønskelig.

Hva som skjer i predict():
Funksjonen henter det siste tidspunktet i datasettet og finner tidspunktet for neste time som skal predikeres. Deretter grupperes data etter stasjon slik at man får den siste registrerte observasjonen per stasjon, og bruker disse som input til modellen for å predikere hvor mange sykler som vil være tilgengelig én time frem i tid. Til slutt skrives resultatet ut.

Resultat:



Figur 23: Utskrift til predict.py

Systemet fungerer som tiltenkt med å hente siste tidsstempel og runde opp til hele neste time, for så å predikere for hver stasjon. Resultatene virker realistiske, selv om noen har større hopp enn andre.

Prøving og feiling under prosjektet:

Modellering: kjørte først GridSearchCV på alle de ulike modellene som skulle testes med ulike hyperparameter, men så at det var unødvendig på dette prosjektet. Endret til å kjøre hver enkelt modell og evaluere med validation data. Dette ble gjort for at det ikke skal ta for lang tid for å kjøre koden.

Kolonne for hver stasjon med {station}_free_bikes: hadde først 9 kolonner med free bikes, altså én for hver stasjon. Når jeg kom til modelleringen, fikk jeg en dårligere rmse-score. Jeg testet med og uten disse kolonnene, og fant ut at det ble best med én kolonne for free_bikes.

Fjerning av rader: fjernet først de siste ni radene i både train, val og test. Dette kunne jeg ikke gjøre ettersom jeg skulle bruke den siste timen til å predikere på. Derfor står disse radene uten target. Endte opp med å fjerne de i modellering før trening av modeller, slik at jeg fikk behold de manglende verdiene på slutten av datasettet for prediksjonen.

Hva kunne bli gjort bedre:

Variabelutvinning: jeg kunne sett endring i RMSE-score for hver enkelt variabel istedenfor å teste på alle 4 jeg vurderte. Det kunne vert at å fjerne 1-2 av de kunne utgjort en positiv forskjell.

Manglende verdier: istedenfor å fylle inn alt av manglende verdier, kunne jeg mulig prøvd ulike imputeringsmetoder.

Ekstra features: hadde jeg hatt bedre tid ville jeg lagt til features «in» og «out» som tok data fra trips. Det ville vist turer per time der «in» viser antall sykler som kommer inn til en stasjon og «out» viser hvor mange sykler som går ut av stasjonen den timen. Vet ikke om det hadde gjort prosjektet bedre, men kunne vert relevant.

Hyperparameter: kunne testet ulike hyperparameter på den beste modellen til slutt.

Bruk av KI:

Jeg brukte KI til å hjelpe med å tolke noen av feilmeldingene. I tillegg ble det brukt til å hjelpe med noen visualiseringer og utskriften i predict.py.