Lab. 3 – Maskininlärning

Kurs:	Digitalisering inom tillverknings-industrin, 20 YH-poäng
Kurskod:	Produktionsteknik
Provkod:	
Lärare:	Pererik Andreasson & Fredrik Lundborg
Uppdaterad:	10 december 2023

Innehåll

1 Mjukstart med maskininlärning

I denna delen av kursen skall du göra några stycken datorlaborationer för att öka din förståelse av de verktyg som kan användas i datadrivna sammanhang. Samtliga övningar i denna laboration är obligatoriska för att få godkänt på kursen. Du behöver inte redovisa allt du har gjort, eftersom en del är övning för att komma igång, men de delar som är explicit markerade i instruktionerna nedan skall lämnas in som Powerpoint eller liknande.

Vissa övningar måste göras i ordning och det är viss progression så att senare övningar förutsätter att tidigare har flutit på bra och att dessa sitter. Sista delen, avnistt 3, kräver lite mättid med din Raspberry, så **börja gärna datainsamlingsdelen av denna i god tid** (omöjligt att få till i sista sekund).

1.1 Data för lineär regression

1. Skapa en ny Jupyter notebook och kopiera in filen `avsvalning.csv` från Blackboard till samma katalog som du skapar Jupyter notebooken i.

2. Läs in din avsvalningsdata till en Pandas dataframe (här kan du med fördel kopiera delar av din kod från labbarna tidigare i kursen). Använd `pd.read_csv()`-funktionen; eftersom avsvalningsdatan inte har några tabellöverskrifter kan du ha `header = None` så får kolumnerna enkla siffror som namn. Här skriver jag inte exakt i detalj hur du skall göra, blir ett lärtillfälle! Har du lyckats ladda in datan bör det se ut som nedan om du använder `.head()`-funktionen på avsvalningsdatan.

	0	1	2
0	2020-09-04 10:23:44.926175	60.0	23.0
1	2020-09-04 10:23:47.502931	66.0	23.0
2	2020-09-04 10:23:52.559373	66.0	23.0
3	2020-09-04 10:23:55.088043	66.0	23.0
4	2020-09-04 10:23:57.617666	66.0	23.0

Ovan ser vi att kolumnerna heter 0, 1 och 2, de har alltså bara fått index som namn eftersom det inte fanns några tabellöverskrifter i avsvalnings-datafilen.

3. Omvandla datum och tid (som är sparat som text) till ett Datetime-objekt så att Python förstår att det är datum och tid. Det gjorde vi i förra labben i kursen med `pd.to_datetime()`-funktionen. Nu skall vi utöka vår förmåga att hitta i Pandas-tabeller och använda index för att hitta rätt kolumn. Förutsatt att du har kallat din Pandas dataframe `cooling` och att `.head()` ser ut som ovan bör följande uttryck fungera felfritt. (Heter din dataframe något annat, byt bara ut `cooling` nedan. Vad du kallar din nya kolumn är valfritt, jag använder `OBJDateTime` för att komma ihåg att det då är Datetime-objekt (alltså riktiga datum och tider och inte text).)

```
cooling["OBJDateTime"] = pd.to_datetime(cooling.iloc[:,0])
```

Här används `.iloc[]` för att hitta kolumner och rader, där första termen, kolon, inom klamrarna betyder alla rader. Andra termen, 0, betyder första kolumnen (index startar på 0 som vanligt). Till exempel om vi vill ha fjärde raden ges detta av uttrycket: `cooling.iloc[3,:]`. Detta behöver ni inte använda, men som förklaring till funktionen `.iloc[]` passar det som exempel. Testa gärna!

4. Nu skall du producera två plottar, en för temperatur och en för relativ luftfuktighet. Använd `.plot()`-funktionen, men måste ges med argument för x och y, annars vet inte Python vad den skall plotta (är blandat text, datum och värden – blir för rörigt och du får felmeddelande). Om du har kallat datumtid-kolumnen från 3 ovan för `OBJDateTime` bör följande argument ge en plot av luftfuktigheten:

```
x = "OBJDateTime", y = 1
```

Här är igen inte hela lösningen med, dessa argument skall alltså ges till `.plot()` på din dataframe. Argumentet för y ges som kolumnindex, för de har inga namn ännu. För att plotta temperaturen byter du bara till `y = 2`. **Dessa två plottar, temperatur mot tid och luftfuktighet mot tid, skall redovisas i din inlämnade PowerPoint så spara dessa separat.**

5. I vår modell vill vi bara ha med avsvalningen, men data samlades även innan maxtemperatur har uppnåtts så vi skapar en ny dataframe som börjar då avsvalningen börjar, d.v.s., på sista raden som temperaturen är 60 °C. Det är lämpligt att spara detta som en egen variabel, som jag nedan kallar `maxtemp`. Det förutsätts att din dataframe heter `cooling` och vi använder `.loc[]` och `.iloc[]` tillsammans med lite Boolesk logik för att hitta de rader som har temp 60 °C.

```
maxtemp = cooling.loc[cooling.iloc[:,2] == 60]
```

   Här hittar `cooling.loc[]` de rader där andra kolumnen (: för alla rader, 2 för kolumn med tempdata) är lika med 60. I min data (den som finns på Blackboard) är detta 15 rader. Om du vill kolla, skriv bara `maxtemp` i en egen cell och kör den så kommer de raderna upp. När vi indexerar i Python kan vi komma åt sista elementet (eller raden, eller kolumnen) genom att använda indexet -1. Alltså om 0 är första indexet och vi inte vet exakt vilket index som är det sista kan vi använda -1. För att hitta sista tillfället då sensorn visar 60 °C kan vi hitta det med `.iloc[-1, 3]`, sista raden i fjärde kolumen. Kolumn nummer tre skall vara datetime-objekten om inget har blivit fel. Vi gör en ny dataframe som i exemplet nedan kallas `cooling_trunc`, där vi bara har med de tider som är senare och lika med tiden vi hittar med `.iloc[-1, 3]`:

```
cooling_trunc = cooling.loc[cooling["OBJDateTime"]
                            >= maxtemp.iloc[-1,3]]
```

   Här hittar alla rader som är senare (i tid) än sista tiden då sensorn visade 60 °C. `cooling_trunc` är en ny dataframe som innehåller samma kolumner som `cooling` men saknar de rader innan sista gången sensorn mätte 60 °C. För att kolla att allt är OK kan vi använda `.head()`, alltså bör `cooling_trunc.head()` ge:

	0	1	2	OBJDateTime
38	2020-09-04 10:25:41.292488	5.0	60.0	2020-09-04 10:25:41.292488
39	2020-09-04 10:25:43.821043	25.0	59.0	2020-09-04 10:25:43.821043
40	2020-09-04 10:25:46.349908	25.0	58.0	2020-09-04 10:25:46.349908
41	2020-09-04 10:25:48.878708	25.0	58.0	2020-09-04 10:25:48.878708
42	2020-09-04 10:25:51.407451	24.0	57.0	2020-09-04 10:25:51.407451

   Där vi ser att första raden har temp 60 °C men sen sjunker temperaturen på nästa mätning. Indexkolumnen börjar på 38 i bilden ovan vilket betyder att vi har klippt bort 38 rader från orginaldatan (första raden är rad 0!). Går att indexera om så den börjar på 0 men det är inte nödvändigt för att fortsätta så vi kan lämna det som det är.

6. I en modell för avsvalningen vill vi inte ha datum och tid på den horisontella axeln utan vi vill ha antal sekunder startandes på noll och fortsätter räkna i bara sekunder och inte gå över till minuter efter 60 sekunder.

För att göra detta subtraherar vi tiden som vår nya dataframe börjar på från alla tidsangivelser i hela kolumnen. Eftersom vi har våra tider som datetime-objekt är detta lätt, bara att använda minus för subtraktion! På denna använder vi metoden `.dt.total_seconds()` så får vi det i sekunder endast (inga minuter). Kommandot nedan ger en en-kolumns dataframe med bara sekunder:

```
timecopy = (cooling_trunc["OBJDateTime"]
            - maxtemp.iloc[-1,3]).dt.total_seconds()
```

Nu är vi nästan på mållinjen för vårt dataset med avsvalningsdata, för att undvika `SettingWithCopyWarning` (den rosa som förvirrar oss i onödan) gör vi nu en ny dataframe med snygga kolumnnamn där bara förlupen tid i sekunder är med. För att skapa en ny, tom dataframe (som här kallas `cooling_small`):

```
cooling_small = pd.DataFrame()
```

Sedan fyller vi på denna med de kolumner vi behöver (bara förlupen tid i sekunder, `timecopy` ovan, och temperaturen i kolumn med index 2 från `cooling_trunc`):

```
cooling_small["Time"] = timecopy
cooling_small["Temp"] = cooling_trunc.iloc[:,2]
```

7. Plotta avsvalningskurvan med tid i sekunder (utan minuter) på x-axeln och temperaturen på y-axeln. **Denna plot skall redovisas på din PowerPoint-inlämning.**

1.2 Lineär regression – rät linje

Denna övning bör göras i samma Jupyter-notebook som datainläsningen och -polerandet i övningen ovan. Tänk på att om du stänger av Jupyter-systemet så måste du köra samtliga celler i notebooken när du startar igen så att all data och det du har gjort med den är inläst i datorn så du kan använda det i övningen nedan.

Enklare trender modelleras med en lineär funktion, alltså ett rakt streck. Vårt första försök att skapa en modell och träna på vår avsvalningsdata blir en lineär modell – lineär regression som det heter på statistik- eller för den delen maskininlärningsspråk. Kom ihåg att en lineär funktion ser ut så här:

$$y = k \cdot x + m$$

där y är vårt sökta resultat, i vårt fall ovan är det temperaturen. x är vår oberoende variabel vilket i vårt avsvalningsexperiment motsvarar tiden. Att x är vår oberoende variabel kan vara lite svårt att ta till sig, men det betyder att det är den som *styr*. Det som ändras är tiden (försök stoppa den om du vågar!) och när tiden går sjunker temperaturen. De parametrar som vi har i vår modell är k, lutningen på linjen, och m, var den skär den vertikala axeln. Dessa två

parametrar skall maskininlärningsalgoritmerna ändra på så att linjen passar vår data så bra som möjligt.

1. För att göra detta behöver vi lite fler verktyg i vår verktygslåda, för att göra den lineära modellen använder vi modulen LinearRegression från Scikit Learn. Importera denna till din Jupyter notebook på liknande sätt som tidigare. I samma cell kan du importera modulen numpy så vi har tillgång till dess funktionalitet:

```
from sklearn.linear_model import LinearRegression
import numpy as np
```

2. Vi hämtar ut vår oberoende variabel X (tiden) och vårt resultat y (temperaturen) från vår dataframe som skapades i övningen ovan med kommandona nedan. `np.vstack()` används för att göra rena kolumner, Scikit-learn vill inte ha Pandas-kolumner.

```
X = np.vstack(cooling_small["Time"])
y = np.vstack(cooling_small["Temp"])
```

3. Nu är nästa steg att göra själva maskininlärandet (vifta med trollstaven), alltså skapa en modell och skapa prediktioner om temperaturen baserat på vår skapade modell. Vi skapar modellen med minsta kvadratmetoden (som vi nu nästan kan):

```
model1 = LinearRegression().fit(X, y)
```

   Nu har vi skapat vår modell och behöver göra prediktioner med denna av temperaturen:

```
y_predict1 = model1.predict(X)
```

   Här har vi alltså sparat den tränade modellen i kolumnen vi kallar `y_predict1`, y för att det är y-värden (output), predict för att det är prediktioner och 1 (etta) efteråt för att det är en lineär modell (ett rakt streck). Färdigtrollat! Applåder!

4. Nästa steg är att kontrollera hur bra modellen blev, det finns flera bra mått på hur bra en modell är, vi kommer här använda oss av absoluta medelfelet, *mean absolute error* (MAE). Dessutom kommer vi plotta data och modell jämte varandra för att få en visuell jämförelse av vad vårt MAE-värde betyder. Funktionen MAE behvöver vi importera från Scikit Learn:

```
from sklearn.metrics import mean_absolute_error
```

   För att beräkna, ger vi funktionen våra riktiga y-värden och våra predikterade y-värden, `print()`-kommandot gör så att MAE för vår modell skrivs ut:

```
mae_linear = mean_absolute_error(y, y_predict1)
print(mae_linear)
```

Bör bli strax under 4.

5. Sista steget är att plotta data och modell i en liten figur. Nu har vi modellen och vår data i olika tabeller så för detta är det lättast att använda modulen Matplotlib, importera denna så här till din Jupyter notebook:

```
import matplotlib.pyplot as plt          '
```

   Att göra en plot med Matplotlib är ganska lätt, vi använder funktionen `plt.plot()` eller `plt.scatter()` med lämpliga argument. Nedan är den första raden vår data från och andra raden plottar modellen som ett streck. Tredje till femte raden lägger till text så att det ser snyggt ut och så att all information finns i figuren. Kommandot `plt.show()` visar plotten i en figur.

```
plt.scatter(X, y)
plt.plot(X, y_predict1, 'r')
plt.xlabel("Tid (s)")
plt.ylabel("Temperatur ($^\circ$C)")
plt.legend(["Model","Data" ])
plt.show()
```

6. **Redovisa ditt MAE-värde och din figur som visar data och lineär modell i din PowerPoint-inlämning.**

1.3 Lineär regression – kvadratisk modell

En kvadratisk modell innebär en böjd kurva och eftersom avsvalningen är typiskt böjd bör denna passa något bättre än ett rakt streck. Kvadratisk innebär att det är någonting upphöjt till två, t.ex.

$$y = \alpha \cdot x^2 + k \cdot x + m$$

Vi skall igen använda Scikit Learn och `LinearRegression` för att göra regressionen men vi behöver lite andra verktyg för att komma ända dit. Den största skillnaden är att vi behöver göra om våra X-värden från ovan med hjälp av en annan modul från Scikit learn, som heter `PolynomialFeatures`, för att få till en kvadratisk modell. Du kan fortsätta denna övning i samma Jupyter notebook som övningen ovan, vi kommer använda samma data för denna modell.

1. Först importerar vi modulen PolynomialFeatures från Scikit learn:

```
from sklearn.preprocessing import PolynomialFeatures
```

   Sedan bestämmer vi kvadratisk genom göra en modell där vi sätter `degree = 2` (rad ett nedan), därefter gör vi om våra X-värden så de passar i en polynom-modell (rad två nedan):

```
poly2 = PolynomialFeatures(degree=2)
x_poly2 = poly2.fit_transform(X)
```

2. Efter denna omvandling kör vi på med `LinearRegresssion` på samma sätt som i övningen ovan, andra raden tränar vår kvadratiska modell.

```
model2 = LinearRegression().fit(x_poly2, y)
y_predict2 = model2.predict(x_poly2)
```

   Där `model2` blir den kvadratiska modellen (något upphöjt till två), vi har samma y-värden (temperaturen är samma) men vi har ändrat X-värdena något för att passa en kvadratisk modell.

3. Plotta data och prediktioner på samma sätt som i övningen ovan, ta fram MAE med `mean_absolute_error()`. Behöver då givetvis vara med argumentet `y_predict2`. **Redovisa MAE och din figur med data och kvadratisk modell på din PowerPoint- inlämning.**

1.4 Modeller av högre ordning (x^3 och x^4)

Efter att vi har testat lineär och kvadratisk modell (upphöjt till två) skall vi nu testa modeller av högre ordning. Högre ordning betyder att det är upphöjt till ett större tal än två, vi nöjer oss med att testa tre och fyra.

Jämfört med övning övningarna ovan är det inte speciellt mycket nytt som behöver skrivas, behåll i samma Jupyter notebook och gör en ny cell för x^3- och en för x^4-modellen. Nedan följer en generell beskrivning som passar bägge dessa, det enda som behöver bytas ut är en trea mot en fyra. Det vi behöver ändra är:

```
poly3 = PolynomialFeatures(degree=3)
```

Denna ger oss modell med upphöjt till tre (eller fyra om trean byts ut).

```
x_poly3 = poly3.fit_transform(X)
model3 = LinearRegression().fit(x_poly3, y)
y_predict3 = model3.predict(x_poly3)
```

Plotta data och prediktioner på samma sätt, och ta fram MAE. För bägge dessa modeller (x^3 och x^4): **Redovisa MAE och din figur med data och modeller på din PowerPoint-inlämning.**

1.5 Modelltestning utanför dataområdet

Sista modellen vi provade, x^4, hade när jag gjorde den lågt MAE och följde exempeldatan väldigt väl. Nu skall vi utforska den modellen om vi tänker att tiden tickar på. Vad vi förväntar oss är att modellen förutspår att det kommer fortsätta vara rumstemperatur efter några timmar. Hittills har vi bara gjort prediktioner baserat på våra faktiska, uppmätta X-värden (ursprungligen från `cooling_small["Time"]`). Men dessa sträcker sig bara exakt lika långt som våra mätningar (eftersom varje tid är från en mätning). Vi måste alltså nu skapa en egen kolumn med tider som sträcker sig några timmar, stoppa in i vår modell med lägst MAE och kolla temperaturen efter några timmar baserat på vår modell. Fortsätt i samma Jupyter notebook.

1. Det som är nytt i denna övning är att vi skall generera en Numpy-array som startar på 0 s och slutar efter kanske två timmar. Resten är i stort sett samma som delarna ovan. En timme innehåller 3600 s, så om vi kör på till 5000 s eller 7000 s borde vi få se om modellen visar några andra trender än att behålla rumstemperatur. Vi skall använda Numpy-funktionen `np.linspace()` för att generara ett jämnt intervall av värden och `np.vstack()` för att de skall stå på rätt håll, parametrar till dessa enligt nedan. Här gör vi direkt våra X-värden som krävs för våren modell med av fjärde ordningen och sedan gör vi vår prediktion:

```
xtest = poly4.fit_transform(np.vstack(
                            np.linspace(0, 5000, 100)))
predictiontest = model4.predict(xtest)
```

2. Plotta datan gör vi som ovan, men plotta vår prediktion blir lite annorlunda eftersom vi måste ange vår `xtest` som X-värden:

```
plt.scatter(X, y)
plt.plot(xtest[:,1], predictiontest, 'r')
```

   `xtest[:,1]` behövs för våra X-värden har fått lite fler kolumner när vi transformerar om dem med `PolynomialFeatures`.

3. **Klistra in den figuren i din Powerpoint tillsammans med sluttemperaturen.** Sluttemperaturen kan fås direkt som sista värdet i `predictiontest`, alltså då tiden har nått 5000 s (eller vilken tid du nu valde) via:

```
predictiontest[-1]
```

   Sista värdet som index i Python är alltid -1. (Näst sista -2 etc.)

1.6 Regressionsträd för stabilare modell

Beroende på vilken data du använde övningarna ovan gav vissa av de olika modellerna ganska dåliga resultat. I den här övningen skall du testa att göra en modell baserat på regressionsträd, alltså en variant av beslutsträd (decision trees) som används för att göra en modell av en kurva. Använd i den här delen det något större datasettet `cooling_data_BPA.txt` som finns på Blackboard.

1. Dela tränings- och testdata. Här behövs tre moduler från Scikit learn: `train_test_split` för att dela upp datan i tränings- och testdata, `DecisionTreeRegression` för att göra själva modellen och igen `mean_absolute_error` för att testa om modellen är bra. Vi behöver även Numpy för att konvertera datan från dataframe. Importera de nya på följande sätt:

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
```

   För att dela upp datan i tränings- och testdata använder vi följande kommando:

```
X_train, X_test, y_train, y_test = train_test_split(
                                    np.vstack(multidata["Time"]),
                                    np.vstack(multidata["Temp"]),
                                    test_size = 0.3,
                                    shuffle = True)
```

Till vänster om likhetstecknet ovan skapas variabler som innehåller X- och y-värden (tid i sekunder och temperatur i °C) fördelat på tränings- och testdata. Till höger om likhetstecknet läses data in, här finns X-värdena i kolumnen `''Time''` i dataframen `multidata` och y-värdena i kolumnen `''Temp''` i samam dataframe. För att konvertera från dataframes till ren kolumndata används ovan `np.vstack()`, detta krävs av `DecisionTreeRegressor`. Heter din dataframe eller kolumnerna något annat behöver dessa ändras. De sista två raderna är mycket viktiga, `test_size` bestämmer hur stor del av datan som skall användas för test (30% i det här fallet) och `shuffle` väljer slumpmässigt ut de som skall vara test- och träningsdata. *Kommandot ovan är bara en rad i Python, jag har lagt in radbryt för att det skall få plats och se lite snyggare ut, men påverkar inte funktionen.*

2. Träna träd-modellen med `.fit`:

```
model_tree = DecisionTreeRegressor(max_depth=5).fit(X_train,
    y_train)
```

Här ges den tränade modellen namnet `model_tree` till vänster om likhetstecknet. Till höger om likhetstecknet använder vi vår modell, sätter maxantal nivåer i trädet till fem och stoppar in träningsdatan.

3. Gör prediktion baserat på den tränade modellen och visualisera resultatet.

```
Xpred = np.vstack(np.linspace(0, 5000, 100))
predictions = model_tree.predict(Xpred)
```

På första raden ovan skapar vi test-data (`Xpred`) som innehåller tid från 0 s till 5000 s i 100 steg.

4. Visualisera med `plt.plot` och `plt.scatter` för modell respektive data. **Denna plot skall vara med i din PowerPoint-redovisning av datorlaborationerna.**

5. Testa hur bra modellen är med hjälp av test-datan. För detta använder vi absoluta medelfelet, det som man får är hur stort felet är i genomsnitt på prediktionen:

```
mae_train = mean_absolute_error(y_train,
                                model_tree.predict(X_train))
mae_test = mean_absolute_error(y_test,
                               model_tree.predict(X_test))
```

Det är förväntat att felet är ganska lågt på träningsdatan, men kan vara något (eller betydligt) högre på testdatan. **Redovisa absoluta medelfelet för både tränings och testdata på din PowerPoint.**

2 Klimatförändringar – återbesök av SMHI-data

I den här övningen skall du använda de SMHI-data som du hämtade och gjorde lite statistisk analys på i förra labben. Du skall i denna övning använda din historiska data över medeltemperaturen i juli för att prediktera medeltemperaturen i juli 2030 baserat på en lineär regressionsmodell. Rekommenderat är att du skapar en ny Jupyter notebook och kopierar över inläsningen av SMHI-datan från din notebook i förra delen av kursen till den nya. Kom ihåg att notebooken och datan behöver ligga i samma katalog på datorn för att det skall fungera.

1. Skapa en ny dataframe som innehåller bara juli-data. Beroende på vad du döper dina dataframes till kan raden nedan användas för att skapa denna dataframe. Kom ihåg att du behöver ha gjort om datum i textformat till datum i DateTime-objekt-format för att det skall fungera (görs med `pd.to_datetime()`-funktionen)

```
juliT = smhidata_datum.loc[pd.DatetimeIndex(
                           smhidata_datum["OBJDatum"]).month==7]
```

   Här är `juliT` min dataframe med bara julidata och `smhidata_datum` är min dataframe med temperaturdata från samtliga år som fanns med. Efter denna behandling bör din dataframe med bara juli-data se ut något i stil med:

	År	Temperatur
0	1961	16.4
1	1961	25.1
2	1961	25.4
3	1961	24.0
4	1961	29.0
...	...	...
4701	2021	17.5
4702	2021	14.7
4703	2021	19.7
4704	2021	17.5
4705	2021	16.8

4706 rows × 2 columns

2. Nu behöver du bara sortera på år och ta medelvärdet sen är du redo att förutspå klimatförändringar. Pandas har en smart funktion som heter `.groupby()` som är högst användbar här. Går givetvis att lösa med en loop eftersom det inte är så många datapunkter det handlar om. För att skapa en mindre, grupperad dataframe med medeltemperaturen i juli varje

år bör följande fungera (beroende på vad dina dataframes och kolumner heter):

```
datat = julitemp.groupby("Year").mean()
```

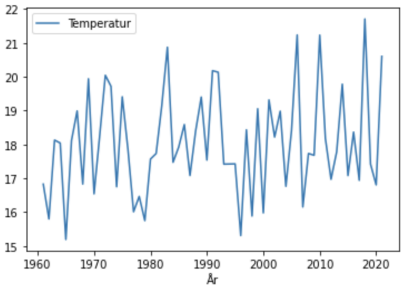
Här grupperar Pandas på årtal och tar medelvärdet av varje år. Följande rader visar de fem första raderna, de fem sista raderna och en plot på årtal.

```
display(datat.head(), datat.tail())
datat.plot()
plt.show()
```

Inklippt blir det så här:

År	Temperatur
1961	16.826882
1962	15.796774
1963	18.127957
1964	18.040860
1965	15.191398

År	Temperatur
2017	16.938710
2018	21.708065
2019	17.417742
2020	16.804839
2021	20.601613



Givetvis beroende på vilken väderstation du har valt, detta är för Oskarshamn.

3. Gör en lineär regressionsmodell för temperaturen över de år som finns i din SMHI- data. Det gör du på liknande sätt som i övningen med lineär regression ovan.

   I din PowerPoint-inlämning för denna övning skall data- och regressionsplot, predikterad temperatur år 2030 och MAE för din modell vara med.
   För att prediktera temperaturen år 2030 behöver vi kalla:

   ```
   model1.predict(np.vstack([2030]))
   ```

   Förutsatt att din modell av medeltemperaturen i juli heter `model1`.

3 Grand finale

Kronan på verket i den här kursen är att du skall som Data Scientist kontrollera isolationen i din bostad (i alla fall på ett ungefär). Nu skall du samla in temperaturdata inomhus och göra en jämförelse med SMHIs utomhusdata för samma tidsperiod för att avgöra om isoleringen är bra eller dålig. I princip består denna uppgift av tre uppgifter, datainsamling, datarensande och datapresentation.

1. Samla in temperaturdata inomhus med din Raspberry över **lite längre tidsperiod**, typiskt en eller två veckor men **minimum två dygn**. Ladda ner datan från Thingspeak.

2. Ladda ner SMHI-data för samma tidsperiod som i punkten ovan, observera att SMHI har senaste tre månaderna och historiska data (äldre än tre månader) i olika filer eftersom SMHI kvalitetskontrollerar innan de lägger till i den historiska datan.

3. Nu bör du besitta tillräckliga kunskaper för att göra snygga dataset med datum, tid, inomhus- och utomhustemperatur. Bestäm själv hur ofta du vill mäta, men det kommer vara SMHI som sätter gränsen där eftersom stationerna inte mäter (eller i alla fall inte sparar) datan jätteofta. Medelvärdet på temperaturen per timme bör räcka. **I din PowerPoint-inlämning bör `.head()` på din slutliga dataframe vara med.**

4. Briljera med dina Data Science-kunskaper och **ha med den snyggaste visualiseringen du kan uppbåda på din ute- och inomhustemperatur** samt korrelationen mellan dessa. Visualiseringen **skall** vara både temperaturer mot datum OCH inomhustemp mot utomhustemp.