Exercise set 0

Problem 1

Task a

Load the data set in npf_train.csv:

```
import pandas as pd

npf = pd.read_csv("npf_train.csv")
```

Task b

Modify and look at dataframe:

```
npf
```

```
##       id        date    class4  ...  UV_B.std   CS.mean    CS.std
## 0      1  2000-01-17        Ib  ...  0.018122  0.000243  0.000035
## 1      2  2000-02-28  nonevent  ...  0.003552  0.003658  0.000940
## 2      3  2000-03-24        Ib  ...  0.272472  0.000591  0.000191
## 3      4  2000-03-30        II  ...  0.451830  0.002493  0.000466
## 4      5  2000-04-04  nonevent  ...  0.291457  0.004715  0.000679
## ..   ...         ...       ...  ...       ...       ...       ...
## 459  460  2011-08-16  nonevent  ...  0.496816  0.002423  0.000425
## 460  461  2011-08-19  nonevent  ...  0.726461  0.002476  0.000902
## 461  462  2011-08-21  nonevent  ...  0.363890  0.003484  0.000457
## 462  463  2011-08-22  nonevent  ...  0.595032  0.004782  0.001082
## 463  464  2011-08-27  nonevent  ...  0.722553  0.006956  0.000605
##
## [464 rows x 104 columns]
```

```
npf = npf.set_index("date")
npf = npf.drop("id",axis=1)
npf
```

```
##               class4  partlybad  CO2168.mean  ...  UV_B.std   CS.mean    CS.std
## date                                          ...
## 2000-01-17        Ib      False   368.771711  ...  0.018122  0.000243  0.000035
## 2000-02-28  nonevent      False   378.197295  ...  0.003552  0.003658  0.000940
## 2000-03-24        Ib      False   373.043158  ...  0.272472  0.000591  0.000191
## 2000-03-30        II      False   375.643019  ...  0.451830  0.002493  0.000466
## 2000-04-04  nonevent      False   377.661030  ...  0.291457  0.004715  0.000679
## ...              ...        ...          ...  ...       ...       ...       ...
```

```
## 2011-08-16  nonevent      False   381.016623  ...  0.496816  0.002423  0.000425
## 2011-08-19  nonevent      False   383.698146  ...  0.726461  0.002476  0.000902
## 2011-08-21  nonevent      False   379.279128  ...  0.363890  0.003484  0.000457
## 2011-08-22  nonevent      False   384.443758  ...  0.595032  0.004782  0.001082
## 2011-08-27  nonevent      False   382.230839  ...  0.722553  0.006956  0.000605
##
## [464 rows x 102 columns]
```

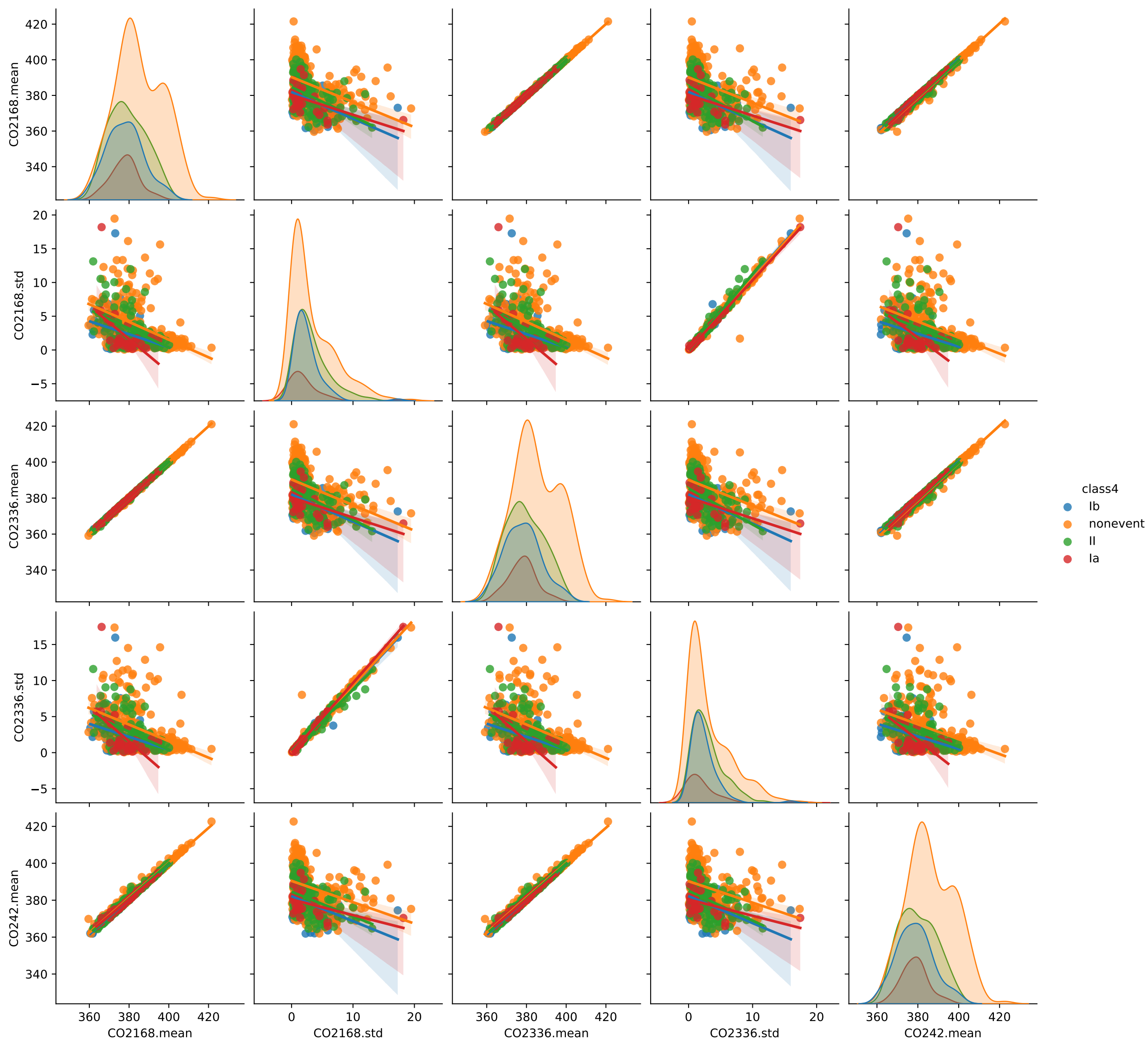
Task c

Plotting: i.-iii.

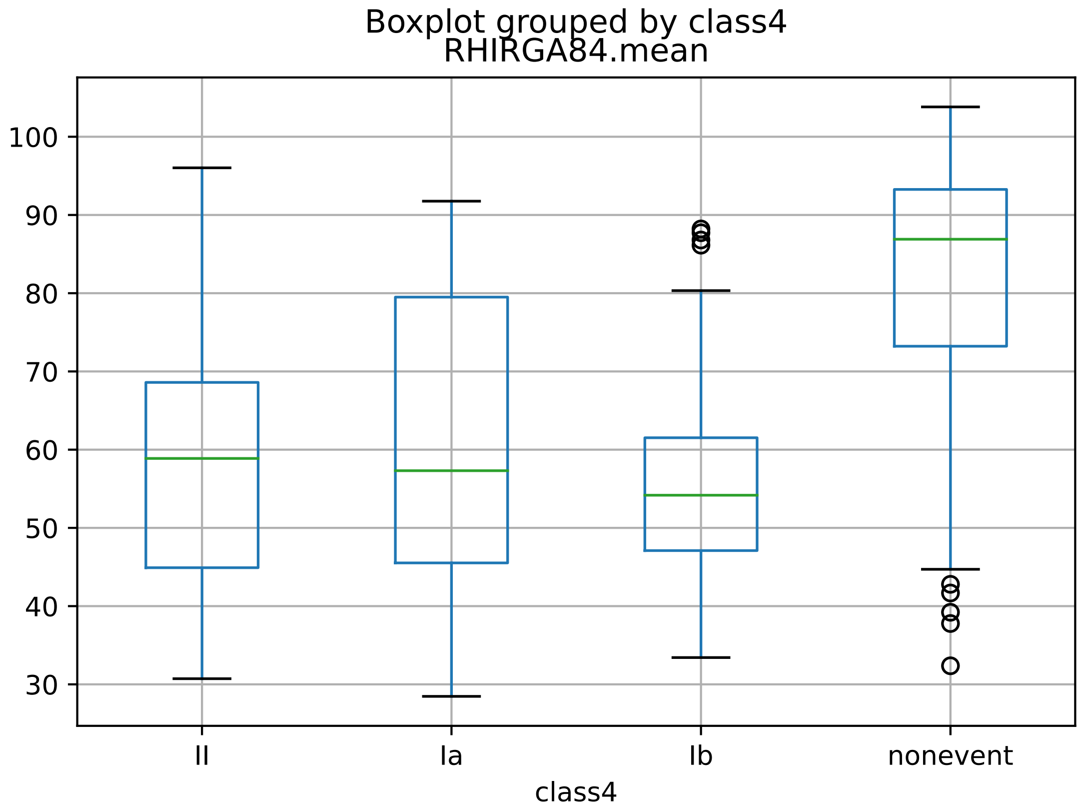
```
import matplotlib.pyplot as plt
import seaborn as sns

npf.describe()
```

```
##        CO2168.mean  CO2168.std  CO2336.mean  ...    UV_B.std     CS.mean      CS.std
## count   464.000000  464.000000   464.000000  ...  464.000000  464.000000  464.000000
## mean    382.072525    3.129971   382.086831  ...    0.366484    0.002963    0.000667
## std      11.080110    3.222030    11.055166  ...    0.287019    0.002146    0.000724
## min     359.579024    0.053968   359.096905  ...    0.003552    0.000243    0.000027
## 25%     374.398155    0.845635   374.389589  ...    0.086265    0.001391    0.000266
## 50%     380.814198    1.952732   380.727947  ...    0.334264    0.002398    0.000476
## 75%     389.048782    4.428063   389.028476  ...    0.589098    0.003910    0.000791
## max     421.511176   19.460521   421.057843  ...    1.055615    0.012670    0.006277
##
## [8 rows x 100 columns]
```

```
npf = npf.drop("partlybad",axis=1)

sns.pairplot(npf,hue="class4",vars=npf.columns[1:6],kind="reg")
```

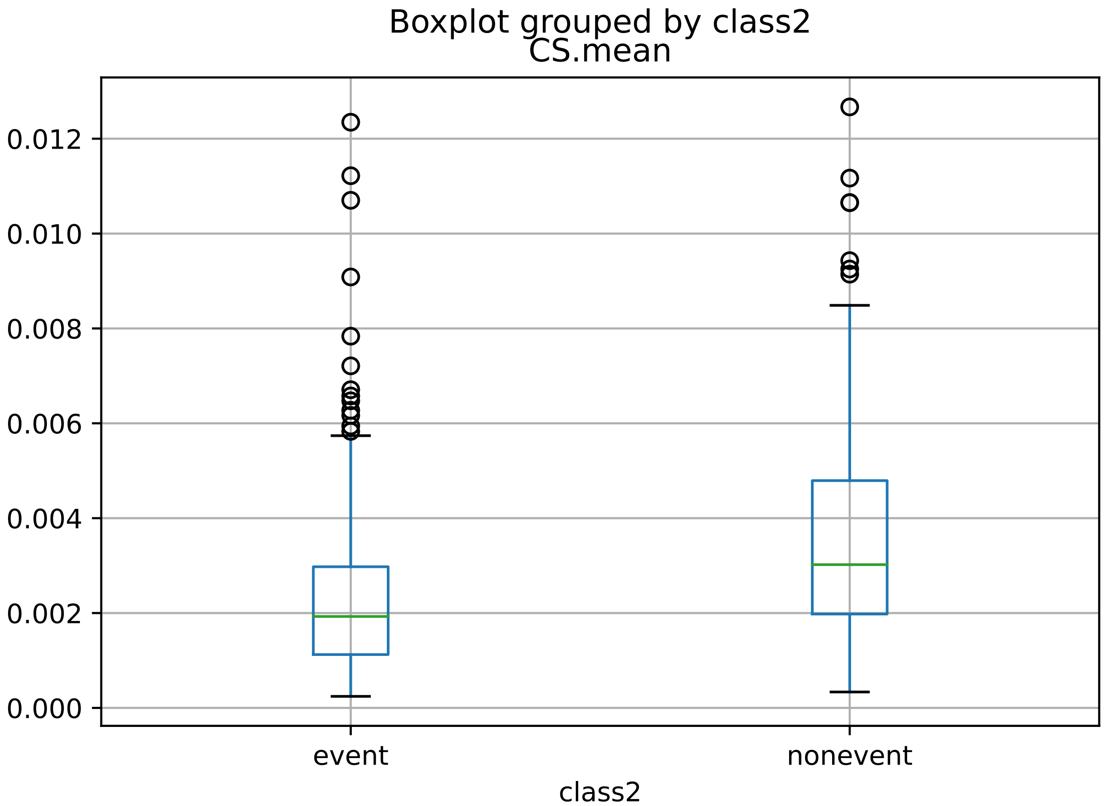



```
npf.boxplot(column="RHIRGA84.mean",by="class4")
```



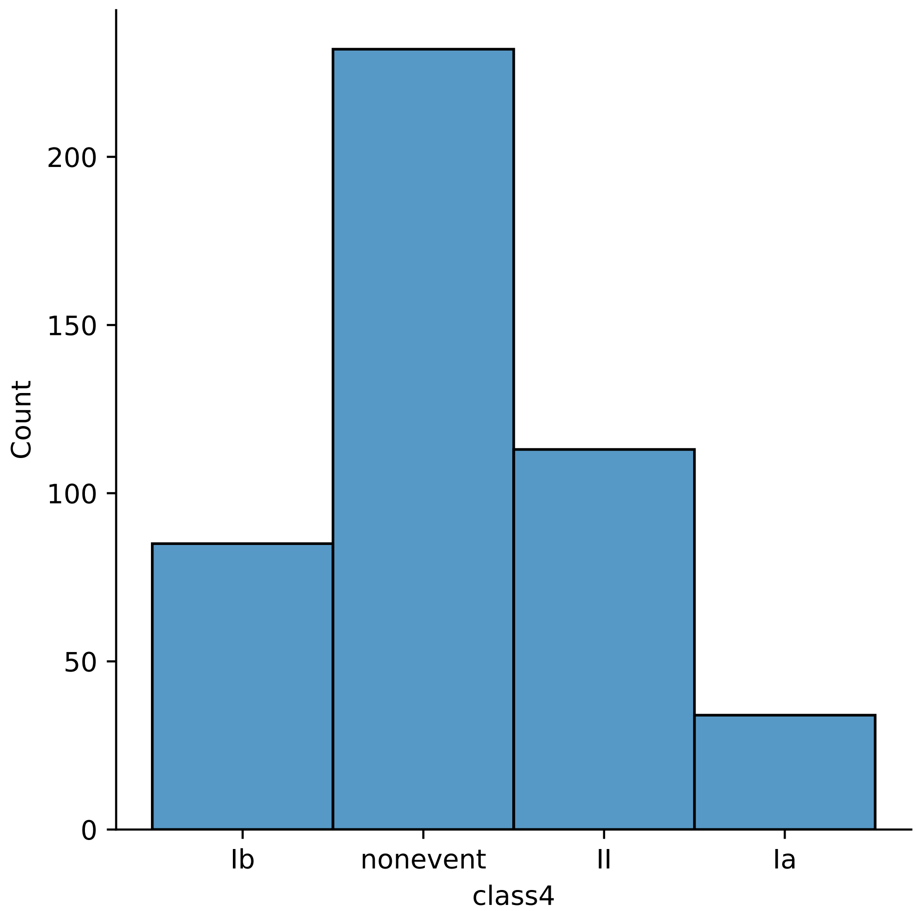
iv.

```
import numpy as np
class2 = np.array(["nonevent", "event"])
npf["class2"] = class2[(npf["class4"]!="nonevent").astype(int)]

npf.boxplot(column="CS.mean", by="class2")
```

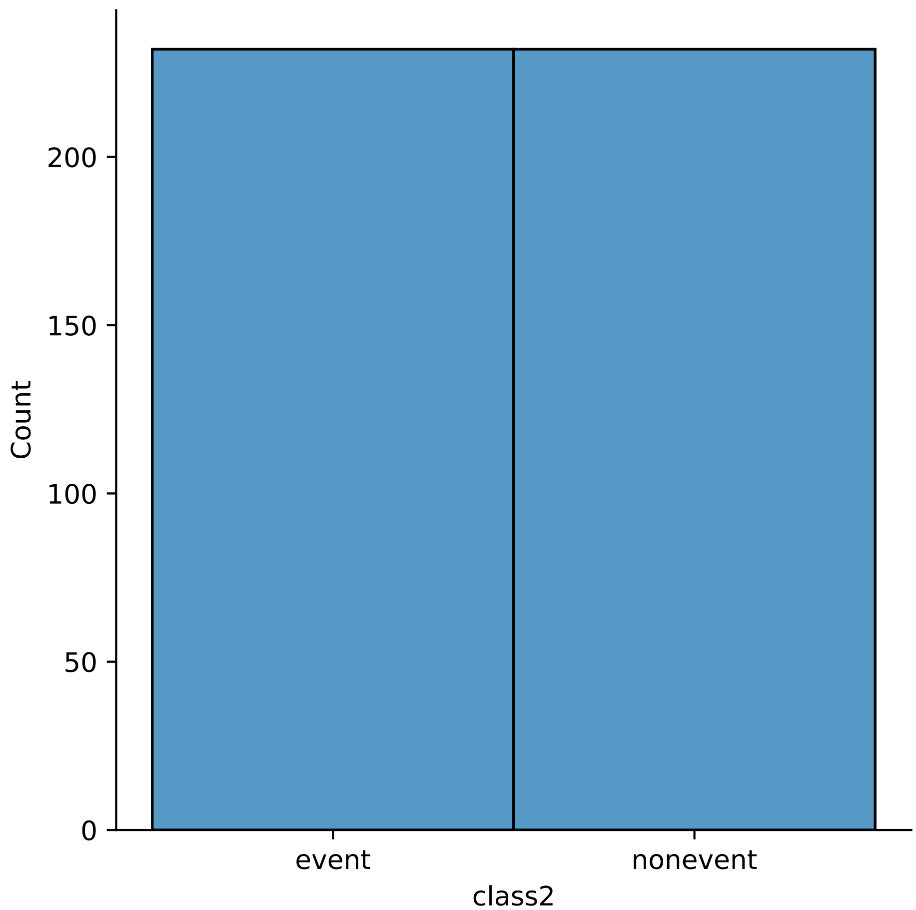


v.

```
sns.displot(npf, x="class4")
```




```
sns.displot(npf, x="class2")
```



vi. Non-events and events occur the same amount. Among the events the most common event type is II, second is Ib and last Ia.

Problem 2

Task a

As an extra model I used Ridge regression.

```
import os
from urllib.request import urlretrieve

import numpy as np
import pandas as pd

from sklearn.dummy import DummyRegressor
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error
```

```
file = "Bias_correction_ucl.csv"

nwp = pd.read_csv(file)
nwp = nwp.drop(["Date", "Next_Tmin"], axis=1)
nwp = nwp.dropna()

#Downsample
nwp, _ = train_test_split(
    nwp, train_size=1000, random_state=42, stratify=nwp["station"]
)

#Splitting data to train and test
X = nwp.drop(["Next_Tmax", "station"], axis=1)
y = nwp["Next_Tmax"]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, train_size=500, random_state=42, shuffle=True, stratify=nwp["station"]
)

models = [DummyRegressor(), LinearRegression(), SVR(), RandomForestRegressor(), Ridge()]
res = pd.DataFrame(index=["dummy", "OLS", "SVR", "RF", "Ridge"])

#Mean squared error function
def loss(X_tr, y_tr, X_te, y_te, m):
    return mean_squared_error(y_te, m.fit(X_tr, y_tr).predict(X_te), squared=False)

#Losses and cross-validation
res["train"] = [loss(X_train, y_train, X_train, y_train, m) for m in models]
res["test"] = [loss(X_train, y_train, X_test, y_test, m) for m in models]
res["cv_10"] = [
    -cross_val_score(
        m, X_train, y_train, cv=10, scoring="neg_root_mean_squared_error"
    ).mean()
    for m in models
]
res["cv_1out"] = [
    -cross_val_score(
        m, X_train, y_train, cv=X_train.shape[0], scoring="neg_root_mean_squared_error"
    ).mean()
    for m in models
]

res
```

```
##           train      test     cv_10   cv_1out
## dummy  3.113638  3.089353  3.109294  2.550849
## OLS    1.419127  1.567541  1.491157  1.167503
## SVR    3.106365  3.087891  3.108758  2.550040
## RF     0.540936  1.503170  1.445121  1.114870
## Ridge  1.419863  1.569854  1.488503  1.164751
```

Task b

Random forest regressor is clearly the best in train, test and validation errors. Close second is Linear regression and Ridge. Ridge regression and linear regression are very similar algorithms and their scores are almost the same in all sets. RMSE on the training data is very similar to the RMSE on the test data. On random forest regressor the difference is, however, quite noticable, giving it's high complexity. CV error is very comparable to the error on the test set. CV error on leave_one_station_out is very noticably lower than 10-fold CV error. Given these observations the linear regression model is very efficient and low in complexity while Support vector regression is as inefficient as the dummy model. Random forest classifier seems very good from the training data, but test and cross-validation data gives us a bit more clearer view and the difference between it and the linear regression model is not too great. The main choice is going to be which of these regressors you use depending on what you value more, time or error.

Task c

Leave-one-station-out cross-validation is cross-validation where you, instead of splitting the dataset into 5-10 parts you split the dataset into how many datapoints you have. Then validate the data compairing all datapoints to one datapoint, one at a time, going through the whole dataset. In Cho et al., the authors used this method to compare to the more common 5-10-fold cross-validation methods.

Problem 3

Task a

```
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split


def create_dataset(n):
  x = np.random.uniform(-3, 3, n)
  e = np.random.normal(0, 0.4, n)
  fx = 2-x+x**2
  y = fx + e
  df = pd.DataFrame(columns=["x", "y"])
  df["x"] = x
  df["y"] = y
  return df

df = create_dataset(1020)

df
```

```
##             x         y
## 0    1.501313  2.801027
## 1    2.210114  4.365355
## 2   -0.197216  2.146964
## 3    0.067094  2.362992
## 4   -1.675020  6.198171
## ...       ...       ...
```

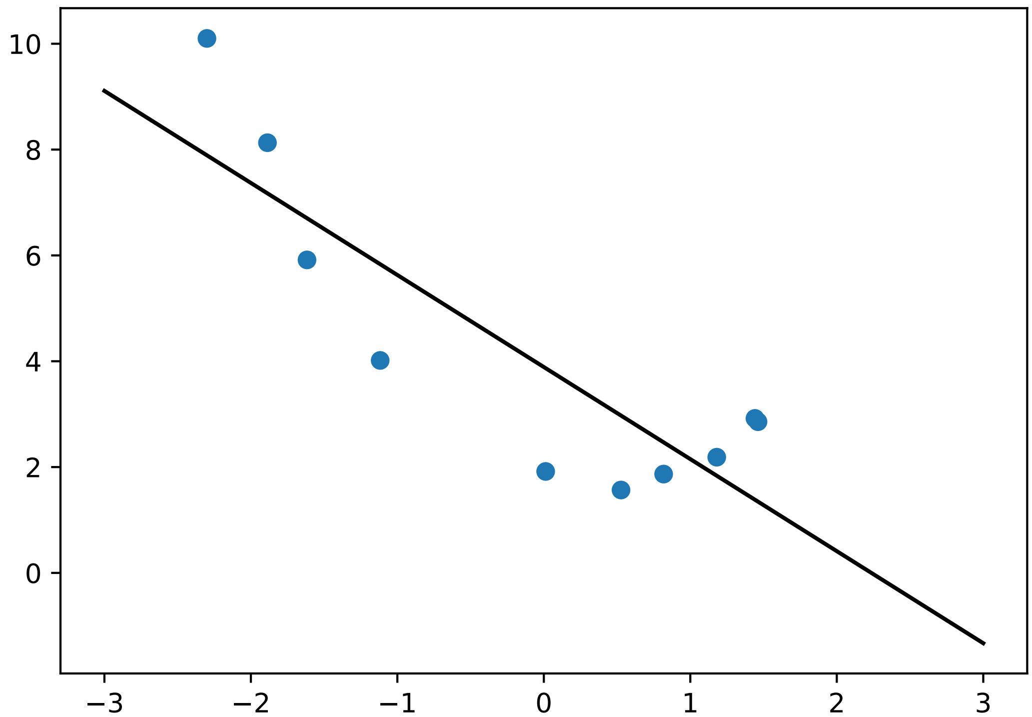
```
## 1015 -1.819594  7.564145
## 1016 -0.761601  3.117326
## 1017  1.866501  3.642278
## 1018  0.518609  1.535702
## 1019  1.493286  2.668158
##
## [1020 rows x 2 columns]
```

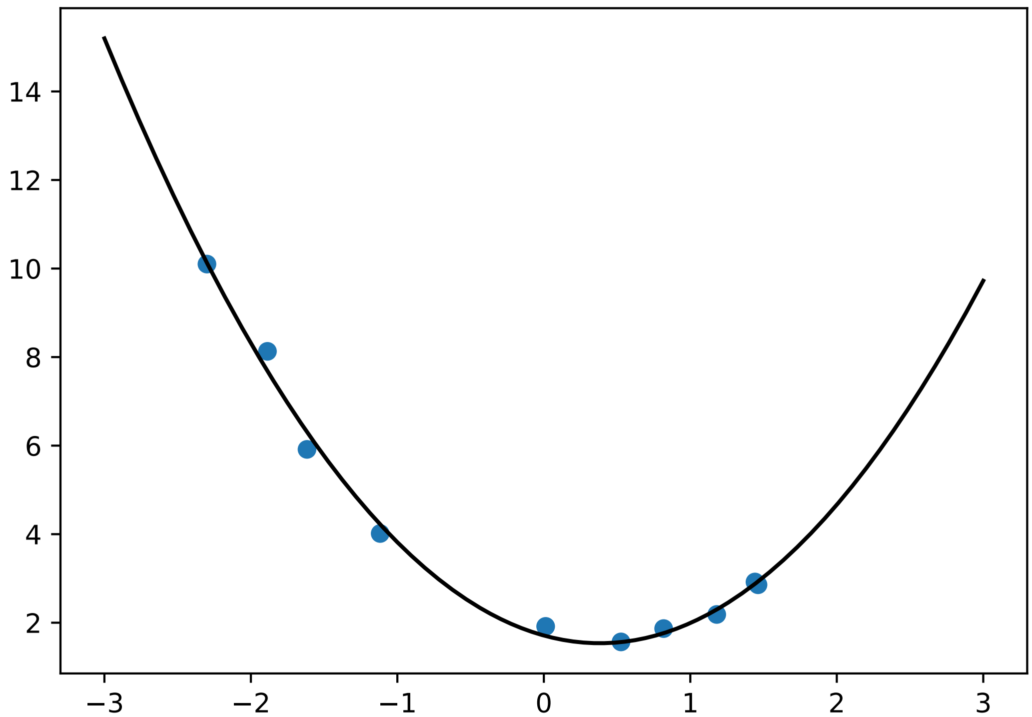
```
X = df["x"]
y = df["y"]

X, X_test, y, y_test = train_test_split(
    X, y, train_size=20, random_state=42, shuffle=True
)

X_train, X_val, y_train, y_val = train_test_split(
    X, y, train_size=10, random_state=42, shuffle=True
)
```

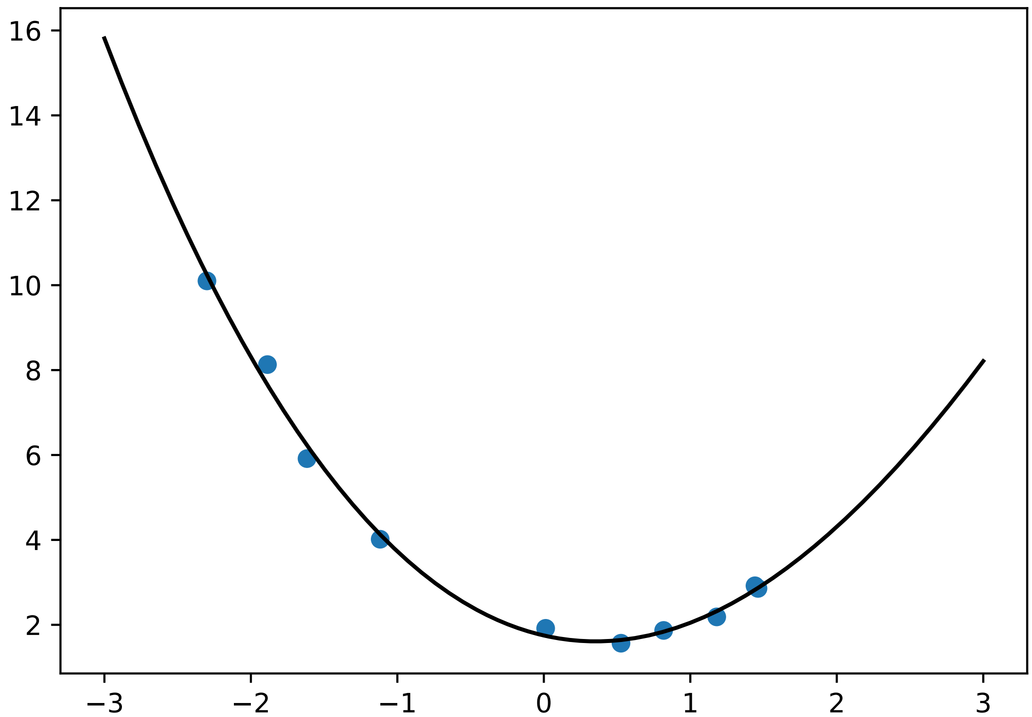
Task b

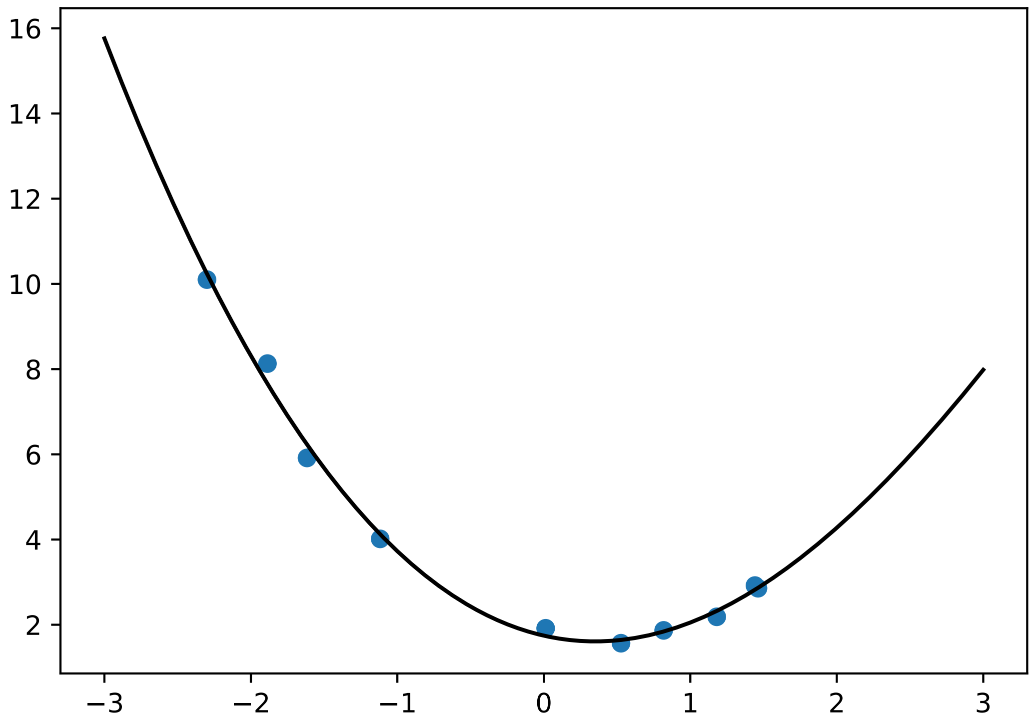
```
import matplotlib.pyplot as plt

polys = [1, 2, 3, 4, 8]
models = []

for poly in polys:
  fig, ax = plt.subplots()
  ax.scatter(X_train, y_train)

  pf = np.polyfit(X_train, y_train, deg=poly)
  models.append(pf)

  lin = np.linspace(-3, 3, num=256)
  ax.plot(lin, np.polyval(pf, lin), color="k")
  plt.show()
```

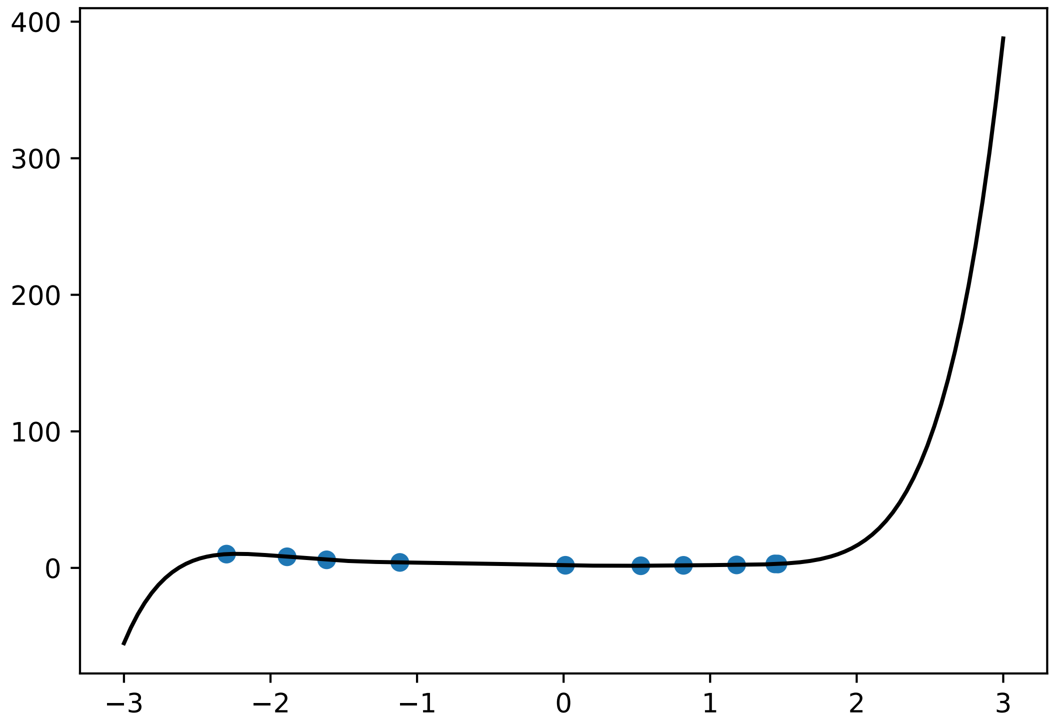
```
## <matplotlib.collections.PathCollection object at 0x0000000044AAD760>
## [<matplotlib.lines.Line2D object at 0x0000000044806C70>]
## <matplotlib.collections.PathCollection object at 0x0000000045B32580>
## [<matplotlib.lines.Line2D object at 0x0000000004743520>]
## <matplotlib.collections.PathCollection object at 0x000000000488BA60>
## [<matplotlib.lines.Line2D object at 0x00000000048B4400>]
## <matplotlib.collections.PathCollection object at 0x00000000049F5430>
## [<matplotlib.lines.Line2D object at 0x00000000049FF8B0>]
## <matplotlib.collections.PathCollection object at 0x0000000045B32130>
## [<matplotlib.lines.Line2D object at 0x0000000004B9DD30>]
```


10
8
6
4
2
0
−3
−2
−1
0
1
2
3


14
12
10
8
6
4
2
−3
−2
−1
0
1
2
3


16
14
12
10
8
6
4
2
−3
−2
−1
0
1
2
3


16
14
12
10
8
6
4
2
−3
−2
−1
0
1
2
3



Task c

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error

def loss(X_tr, y_tr, X_te, y_te, m):
  return mean_squared_error(y_te, np.polyval(m, X_te), squared=False)

res = pd.DataFrame(columns=polys)

res.loc["train"] = [loss(X_train, y_train, X_train, y_train, m) for m in models]
res.loc["val"] = [loss(X_train, y_train, X_val, y_val, m) for m in models]
res.loc["test"] = [loss(X_train, y_train, X_test, y_test, m) for m in models]

res
```

```
##                1          2          3          4          8
## train   1.436676   0.217557   0.205724   0.205706   0.038840
## val     2.709040   0.791217   0.593872   0.579849  19.371303
## test    3.198062   0.732886   0.638341   0.626950  67.070134
```

Task d

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.datasets import make_regression

X_comb = pd.concat([X_train, X_val])
y_comb = pd.concat([y_train, y_val])

X_poly = pd.DataFrame(X_comb)

cross_score = []
for poly in polys:
  linear = LinearRegression()
  Xd = PolynomialFeatures(poly).fit_transform(X_poly)
  cross_score.append(-cross_val_score(linear, Xd, y_comb, cv=5).mean())

res.loc["cv"] = cross_score


res.loc["comb_test"] = [loss(X_comb, y_comb, X_test, y_test, m) for m in models]
res
```

```
##                    1         2         3         4          8
## train       1.436676  0.217557  0.205724  0.205706   0.038840
## val         2.709040  0.791217  0.593872  0.579849  19.371303
## test        3.198062  0.732886  0.638341  0.626950  67.070134
## cv          0.652657 -0.866419 -0.858807 -0.863246   0.375971
## comb_test   3.198062  0.732886  0.638341  0.626950  67.070134
```

Task e

Polynomial of 4 seems to be the winner in every category and would suit the model the best. Even if test set wouldn't be available.

Problem 4

Task a

- Squared bias: Average accuracy of the model increases when flexibility increases.
- Variance: Variance on the other hand decreases as flexibility increases since these models will be more drastic in correlation to the training data.
- Training error: Decreases since more flexible model can adjust to training set better.
- Test error: First decreases but then increases since too flexible model starts making too drastic changes based on the training data.
- Bayes error: Always constant. Just some noice.

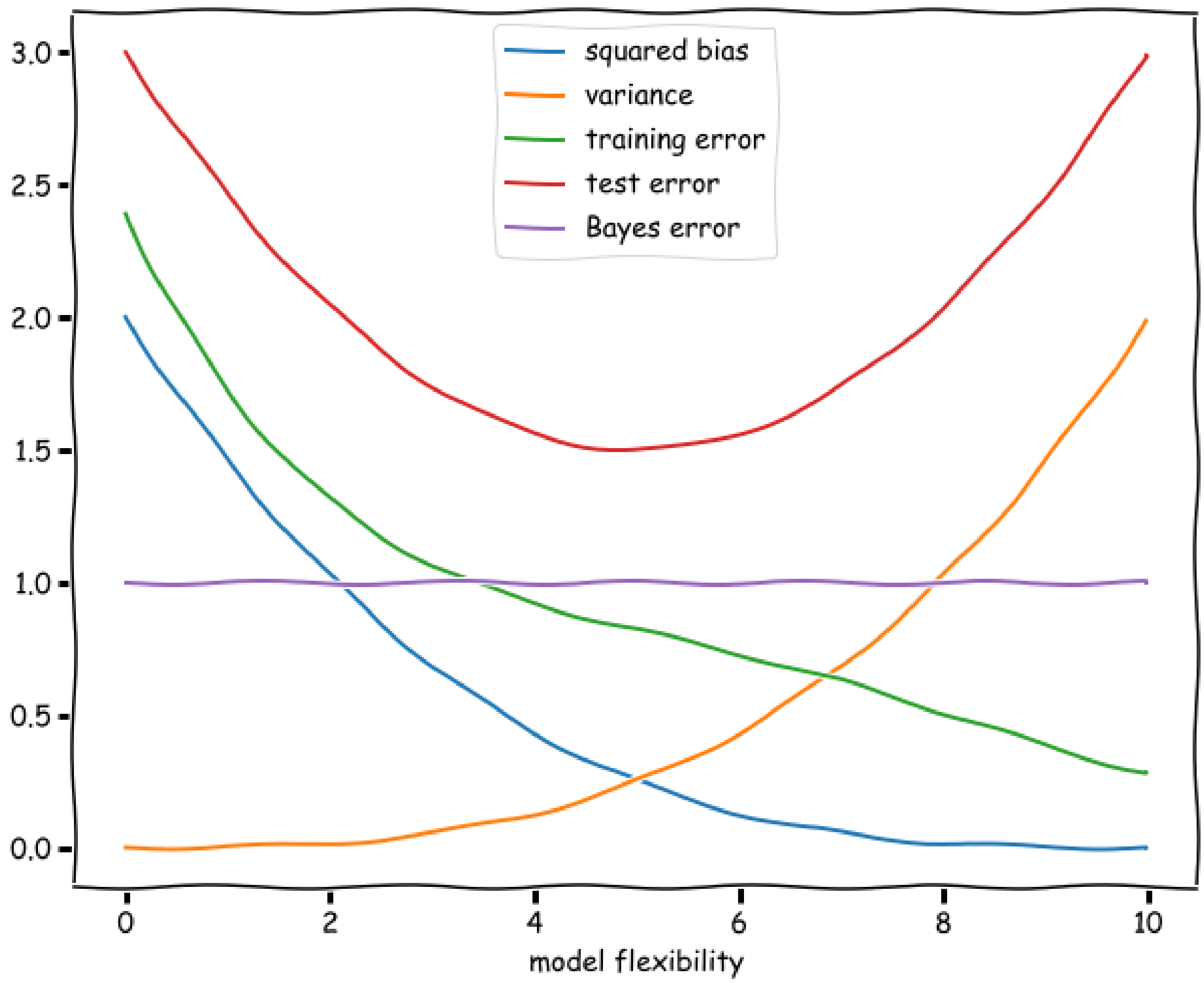


Figure 1: Error curves

Problem 5

Task a

$$E[L_{test}] = E[\frac{1}{m}\sum_{i=1}^{m}(\bar{y}_i - \hat{\beta}^T\bar{x}_i)^2] = \frac{1}{m}\sum_{i=1}^{m}E[(\bar{y}_i - \hat{\beta}^T\bar{x}_i)^2] = \frac{1}{m}(E[(\bar{y}_1 - \hat{\beta}^T\bar{x}_1)^2] + ... + E[(\bar{y}_n - \hat{\beta}^T\bar{x}_n)^2]) = E[(\bar{y}_1 - \hat{\beta}^T\bar{x}_1)^2]$$

Task b

To prove that estimate of L_{test} is an unbiased estimate of the generalization error for the OLS regression, we have to prove that

$$E[\frac{1}{m}\sum_{i=1}^{m}(\bar{y}_i - \hat{\beta}^T x_i)^2] = E[\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2]$$

. From the modification we did to L_{test} in task a we can see that $E[y_i] = E[y]\ \forall i$ and $\hat{\beta}^T x = \hat{y}$. From this we can state that the equation

$$E[L_{test}] = E[(y - \hat{y}^2)]$$

.

Task c

We must prove:

$$E[\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{\beta}^T\bar{x}_i)^2] \leq E[\frac{1}{m}\sum_{i=1}^{m}(\bar{y}_i - \hat{\beta}^T\bar{x}_i)^2]$$

The only term possible to affect this equation is $\hat{\beta}^T$. From Problem 4 task a we can see that the this term favors the training set; Because of $\hat{\beta}^T$ the loss is always less or equal to the test set. So $\hat{\beta}^T_{train} \leq \hat{\beta}^T_{test}$.

Task d

The previous task is related to the generalization problem in machine learning since it means that the difference between test and train results has to be found between bias and variance so that the model is not too fitting to training data (overfitting) or too general (underfitting).

Problem 6

Task a

```python
import numpy as np
import pandas as pd

data = pd.read_csv("co2lite.csv")

y_pred = [np.mean(data["FCO2"]) for i in data["FCO2"]]
difference = data["FCO2"] - y_pred
std = difference.std(ddof=1)
t_value = (np.mean(difference))/(std/np.sqrt(len(difference)))

np.mean(data["FCO2"])
```

```
## -1.4710767763200001
```

```
t_value
```

```
## 2.24863277762917e-16
```

```
std
```

```
## 4.468756230695357
```

With this few datapoints it's not yet possible to state that the true mean of FCO2 is non-zero.

Problem 7

The beginning of this course has really hooked me in and it's starting to seem like it's my favourite course this far. I have learned and understood many of the beginning principles of machine learning and supervised learning. Bits of the math side is always hard but with time I will hopefully understand those as well. I have thought about including a bit of machine learning to my bachelor's thesis so this knowledge will be very useful.