In [676…
```
import numpy as np
import pandas as pd
import matplotlib.pylab as plt
%matplotlib inline
```

Assignment 5

1. Choose a regression dataset (bikeshare is allowed), perform a test/train split, and build a regression model (just like in assingnment 3), and calculate the

Training Error (MSE, MAE)

Testing Error (MSE, MAE)

In [677…
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
```

In [678…
```
# Training dataset
df = pd.read_csv('../data/WineQT.csv')
```

Dataset: https://www.kaggle.com/rajyellow46/wine-quality

In [679…
```
df.columns
```

Out[679]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [680…
```
y = df["quality"]
x = df.drop(["quality"], axis = 1)
```

In [681…
```
y.shape, y.size
```

Out[681]:
```
((1142,), 1142)
```

In [682…
```
x.shape, x.size
```

Out[682]:
```
((1142, 11), 12562)
```

In [683…
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.5)
```

In [684…
```
linreg = LinearRegression()
linreg.fit(x_train, y_train)
linreg.coef_, linreg.intercept_
```

Out[684]:
```
(array([ 4.17828651e-02, -1.17572106e+00, -3.26310276e-01,  1.64549005e-02,
        -1.07675841e+00,  2.40359598e-03, -2.92762569e-03, -3.37836342e+01,
        -3.32272777e-01,  8.11536960e-01,  2.80284864e-01]),
 37.48044580851582)
```

In [685…
```
pred = linreg.predict(x_train)
```

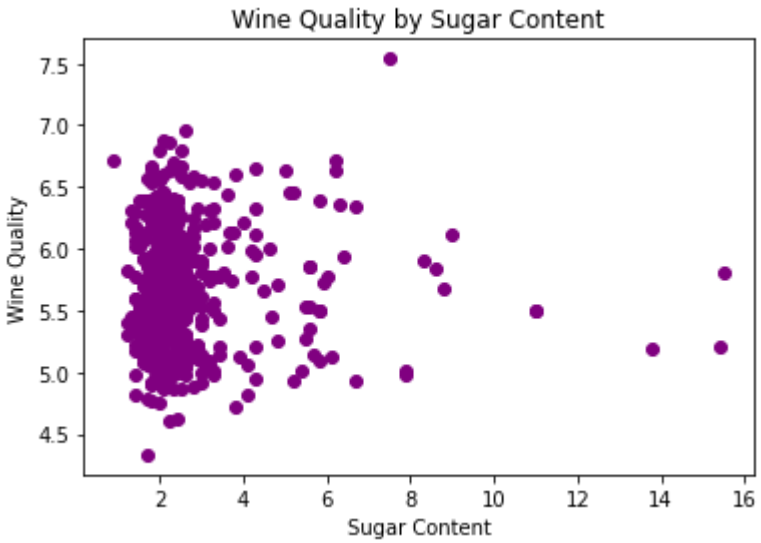
In [686…
```
sugar = x_train["residual sugar"]
```

In [687…
```
sugar.shape, pred.shape
```

Out[687]:
```
((571,), (571,))
```

In [691…
```
plt.scatter(sugar, pred, c = "purple")
plt.title("Wine Quality by Sugar Content")
plt.xlabel("Sugar Content")
plt.ylabel("Wine Quality")
```

Out[691]:
```
Text(0, 0.5, 'Wine Quality')
```



In [692…
```
# Train MSE and MAE
print(mean_squared_error(y_train, pred))
print(mean_absolute_error(y_train, pred))
```

```
0.4264465126020117
0.49899128751283467
```

In [693…
```
# Test MSE and MAE
print(mean_squared_error(y_test, np.dot(x_test, linreg.coef_) + linreg.intercept_))
print(mean_absolute_error(y_test, np.dot(x_test, linreg.coef_) + linreg.intercept_))
```

```
0.38902823650821267
0.4901619857853927
```

2. Choose a classification dataset (not the adult.data set, perform test/train split and create a classification model (your choice but DecisionTree is fine). Calculate:

Accuracy

Confusion Matrix

Classifcation Report

Import libraries, define model, test/train/split

In [729…
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import (accuracy_score,
                             classification_report,
                             confusion_matrix
                            )
```

In [730…
```
model = DecisionTreeClassifier(criterion = "entropy")
```

In [731…
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.50)
```

In [732…
```
x_test.shape, x_train.shape
```

Out[732]:
```
((571, 11), (571, 11))
```

In [733…
```
y_test.shape, y_train.shape
```

Out[733]:
```
((571,), (571,))
```

Train

In [734…
```
model.fit(x_train, y_train)
```

Out[734]:
```
DecisionTreeClassifier(criterion='entropy')
```

Feature Importances

In [735…
```
list(zip(x.columns, model.feature_importances_))
```

Out[735]:
```
[('fixed acidity', 0.08147017493363087),
 ('volatile acidity', 0.15001754033356043),
 ('citric acid', 0.06951872605508914),
 ('residual sugar', 0.061313460722376485),
 ('chlorides', 0.06691291921237352),
 ('free sulfur dioxide', 0.08708687189794129),
 ('total sulfur dioxide', 0.08275321833101687),
 ('density', 0.0632502085969847),
 ('pH', 0.09345163942785982),
 ('sulphates', 0.09201002699099102),
 ('alcohol', 0.15221521349817582)]
```

In [736…
```
predictions = model.predict(x_train)
```

In [737…
```
accuracy_score(y_test, predictions)
```

Out[737]:
```
0.3467600700525394
```

In [738…
```
confusion_matrix(y_test, predictions)
```

Out[738]:
```
array([[  0,   0,   1,   2,   1,   1],
       [  0,   0,   7,   7,   2,   0],
       [  1,   6,  99, 113,  31,   3],
       [  0,   7,  91,  90,  29,   4],
       [  0,   4,  28,  27,   9,   1],
       [  0,   0,   3,   2,   2,   0]], dtype=int64)
```

In [739…
```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         5
           4       0.00      0.00      0.00        16
           5       0.43      0.39      0.41       253
           6       0.37      0.41      0.39       221
           7       0.12      0.13      0.13        69
           8       0.00      0.00      0.00         7

    accuracy                           0.35       571
   macro avg       0.15      0.15      0.15       571
weighted avg       0.35      0.35      0.35       571
```

Test

In [740…
```
test_predictions = model.predict(x_test)
```

In [741…
```
predictions.shape, test_predictions.shape
```

Out[741]:
```
((571,), (571,))
```

In [742…
```
accuracy_score(y_test, test_predictions)
```

Out[742]:
```
0.5148861646234676
```

In [743…
```
confusion_matrix(y_test, test_predictions)
```

Out[743]:
```
array([[  0,   0,   3,   2,   0,   0],
       [  0,   0,   9,   5,   2,   0],
       [  2,  12, 154,  73,  11,   1],
       [  3,   8,  59, 102,  47,   2],
       [  2,   0,   2,  23,  37,   5],
       [  0,   0,   1,   2,   3,   1]], dtype=int64)
```

In [744…
```
print(classification_report(y_test, test_predictions))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         5
           4       0.00      0.00      0.00        16
           5       0.68      0.61      0.64       253
           6       0.49      0.46      0.48       221
           7       0.37      0.54      0.44        69
           8       0.11      0.14      0.12         7

    accuracy                           0.51       571
   macro avg       0.27      0.29      0.28       571
weighted avg       0.54      0.51      0.52       571
```

In [ ]:
```

```