Assignment 5

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

```
+ Training Error (MSE, MAE)
+ Testing Error (MSE, MAE)
```

In [162]:
```
import matplotlib.pyplot as plt # Setup
%matplotlib inline
plt.rcParams['figure.figsize'] = 20, 10
import pandas as pd
import numpy as np
import math
from sklearn import linear_model, metrics
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.metrics import (accuracy_score,
                             classification_report,
                             confusion_matrix, auc, roc_curve
                            )
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [163]:
```
# Import new regression data
wages = pd.read_csv('../data/wages.csv')
wages
```

Out[163]:

	female	urban	edu	exp	wage
0	1	1.0	4.0	17.727152	NaN
1	0	NaN	1.0	7.901757	34497.08594
2	0	0.0	3.0	23.718281	NaN
3	0	1.0	NaN	23.976738	75486.74219
4	0	1.0	2.0	25.446344	51890.10156
...	...	...	...	...	...
2995	1	0.0	2.0	0.000000	67397.41406
2996	0	0.0	3.0	35.154343	NaN
2997	1	1.0	3.0	12.892221	48033.35156
2998	1	1.0	4.0	8.605361	74244.77344
2999	0	1.0	3.0	21.547012	NaN

3000 rows × 5 columns

In [164]:
```
# Data cleaning
wages_clean = wages.dropna()
wages_clean
```

Out[164]:

	female	urban	edu	exp	wage
4	0	1.0	2.0	25.446344	51890.10156
6	1	1.0	4.0	10.086199	66897.09375
8	1	0.0	3.0	0.000000	25374.85156
9	1	1.0	1.0	0.213173	0.00000
10	1	1.0	1.0	0.000000	66817.35156
...	...	...	...	...	...
2989	1	0.0	2.0	10.685445	28155.04688
2990	0	1.0	3.0	14.312721	16781.22070
2995	1	0.0	2.0	0.000000	67397.41406
2997	1	1.0	3.0	12.892221	48033.35156
2998	1	1.0	4.0	8.605361	74244.77344

1973 rows × 5 columns

In [165]:
```
# Building the X values
columns = ['female', 'urban', "edu", "exp"]
X = wages_clean[columns].values

X = np.vstack([X.T, np.ones(len(X))]).T
X
```

Out[165]:
```
array([[ 0.        ,  1.        ,  2.        , 25.44634438,  1.        ],
       [ 1.        ,  1.        ,  4.        , 10.08619881,  1.        ],
       [ 1.        ,  0.        ,  3.        ,  0.        ,  1.        ],
       ...,
       [ 1.        ,  0.        ,  2.        ,  0.        ,  1.        ],
       [ 1.        ,  1.        ,  3.        , 12.8922205 ,  1.        ],
       [ 1.        ,  1.        ,  4.        ,  8.60536099,  1.        ]])
```

In [166]:
```
# Building the Y values
Y = wages_clean['wage']
Y
```

Out[166]:
```
4       51890.10156
6       66897.09375
8       25374.85156
9           0.00000
10      66817.35156
           ...
2989    28155.04688
2990    16781.22070
2995    67397.41406
2997    48033.35156
2998    74244.77344
Name: wage, Length: 1973, dtype: float64
```

In [167]:
```
# Building the train test set for both a linear model and a 5th polyno

X_train, X_test, Y_train, Y_test = train_test_split(wages_clean.drop([
                                                     wages_clean.wage,

linear = linear_model.LinearRegression().fit(X_train, Y_train)

X_train5 = PolynomialFeatures(degree=5).fit_transform(X_train)
X_test5 = PolynomialFeatures(degree=5).fit_transform(X_test)

linear5 = linear_model.LinearRegression().fit(X_train5, Y_train)
```

In [168]:
```
# Using the module 3 method for the beta equation
Left  = np.linalg.inv(np.dot(X.T, X))
Right = np.dot(Y.T, X)
np.dot(Left, Right)
```

Out[168]:
```
array([-19255.91623585,   4183.39035646,  14360.31295072,   1783.8555
438 ,
        16665.98627362])
```

In [169]:
```
# Solving for beta
Beta = np.dot(Left, Right)
Beta
```

Out[169]:
```
array([-19255.91623585,   4183.39035646,  14360.31295072,   1783.8555
438 ,
        16665.98627362])
```

In [170]:
```
# Building the linear prediction as from module 3
linear_pred = np.dot(X, Beta)
linear_pred
```

Out[170]:
```
array([94962.60502322, 77027.0338602 , 40491.00888994, ...,
       26130.69593922, 67672.25825721, 74385.43309661])
```

In [171]:
```
# MSE of the training and the linear predition
(
    metrics.mean_squared_error(Y_train, linear5.predict(X_train5)),
    metrics.mean_squared_error(Y_train, linear.predict(X_train)),
    metrics.mean_squared_error(Y, linear_pred)
)
```

Out[171]:
```
(812199518.1965536, 861543881.0522715, 872270130.1301197)
```

In [172]:
```
# MSE of the testing and the linear predition
(
    metrics.mean_squared_error(Y_test, linear5.predict(X_test5)),
    metrics.mean_squared_error(Y_test, linear.predict(X_test)),
    metrics.mean_squared_error(Y, linear_pred)
)
```

Out[172]:
```
(995172935.3428698, 919485585.6104162, 872270130.1301197)
```

In [173]:
```
# MAE of the training and the linear prediction
(
    metrics.mean_absolute_error(Y_train, linear5.predict(X_train5)),
    metrics.mean_absolute_error(Y_train, linear.predict(X_train)),
    metrics.mean_absolute_error(Y, linear_pred)
)
```

Out[173]:
```
(23033.12671110714, 23538.80105399361, 23710.206833882366)
```

In [174]:
```
# MAE of the testing and the linear prediction
(
    metrics.mean_absolute_error(Y_test, linear5.predict(X_test5)),
    metrics.mean_absolute_error(Y_test, linear.predict(X_test)),
    metrics.mean_absolute_error(Y, linear_pred)
)
```

Out[174]:
```
(24850.299210191537, 24423.528864297652, 23710.206833882366)
```

2. Choose a CLASSIFICATION dataset (not the adult.data set, The UCI repository has many datasets as well as Kaggle), perform test/train split and create a classification model (your choice but DecisionTree is fine). Calculate

```
+ Accuracy
+ Confusion Matrix
+ Classifcation Report
```

In [175]:
```
# Loading Data

heart = pd.read_csv('../data/heart+disease/processed.cleveland.data',
                    names=["age", "sex", "cp", "trestbps", "chol", "fb
                           "exang", "oldpeak", "slope", "ca", "thal",
        # age: age in years
        # sex: sex (1 = male; 0 = female)
        # cp: chest pain type: 1: typical angina, 2: atypical angina,
        # trestbps: resting blood pressure (in mm Hg on admission to t
        # chol: serum cholestoral in mg/dl
        # fbs: (fasting blood sugar > 120 mg/dl)  (1 = true; 0 = false
        # restecg: resting electrocardiographic results: Value 0: norm
        # thalach: maximum heart rate achieved
        # exang: exercise induced angina (1 = yes; 0 = no)
        # oldpeak = ST depression induced by exercise relative to rest
        # slope: the slope of the peak exercise ST segment: Value 1: u
        # ca: number of major vessels (0-3) colored by flourosopy
        # thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
        # num: diagnosis of heart disease: Value = 0: absence, Value >

heart
```

Out[175]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	r
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0	
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0	
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
298	45.0	1.0	1.0	110.0	264.0	0.0	0.0	132.0	0.0	1.2	2.0	0.0	7.0	
299	68.0	1.0	4.0	144.0	193.0	1.0	0.0	141.0	0.0	3.4	2.0	2.0	7.0	
300	57.0	1.0	4.0	130.0	131.0	0.0	0.0	115.0	1.0	1.2	2.0	1.0	7.0	
301	57.0	0.0	2.0	130.0	236.0	0.0	2.0	174.0	0.0	0.0	2.0	1.0	3.0	
302	38.0	1.0	3.0	138.0	175.0	0.0	0.0	173.0	0.0	0.0	1.0	?	3.0	

303 rows × 14 columns

In [176]:
```
# Building Train/Test Set and clean the data

heart_clean = heart[pd.to_numeric(heart["ca"], errors='coerce').notnul


heart_clean = heart_clean[pd.to_numeric(heart_clean["oldpeak"], errors

heart_clean = heart_clean[pd.to_numeric(heart_clean["thal"], errors='c

x_train, x_test, y_train, y_test = train_test_split(heart_clean.drop([
                                                     heart_clean.num, t

model = DecisionTreeClassifier(criterion='entropy')

model.fit(x_train, y_train)
```

Out[176]:
```
▾           DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy')
```

In [177]:
```
# Test Predictions

test_predictions = model.predict(x_test)
```

In [178]:
```
# Accuracy

accuracy_score(y_test, test_predictions)
```

Out[178]:
```
0.5
```

In [179]:
```
# Confusion Matrix

confusion_matrix(y_test, test_predictions)
```

Out[179]:
```
array([[25,  3,  2,  0,  0],
       [ 5,  3,  0,  1,  0],
       [ 3,  3,  1,  1,  1],
       [ 3,  5,  1,  1,  0],
       [ 0,  1,  0,  1,  0]])
```

In [180]:
```python
# Classification Report

print(classification_report(y_test, test_predictions))
```

```
              precision    recall  f1-score   support

           0       0.69      0.83      0.76        30
           1       0.20      0.33      0.25         9
           2       0.25      0.11      0.15         9
           3       0.25      0.10      0.14        10
           4       0.00      0.00      0.00         2

    accuracy                           0.50        60
   macro avg       0.28      0.28      0.26        60
weighted avg       0.46      0.50      0.46        60
```

3. (Bonus) See if you can improve the classification model's performance with any tricks you can think of (modify features, remove features, polynomial features)

I was reading through the documentation on UCI's website for the Heart Disease data and it appears that the scientists using the data only modeled whether or not the observation had any heart disease not the severity. With that in mind I am going to convert the "num" column to a binary column on the presence or absence of heart disease. I wanted to see if the model was better a predicting *any* heart disease instead of the severity. I am not sure if this counts as a performance improvement since it isn't a feature modification.

In [181]:
```
# Adding new column of T/F
heart_clean["heart_disease"] = heart_clean['num'].astype(bool)
heart_clean["heart_disease"],heart_clean['num']
```

Out[181]:
```
(0      False
 1       True
 2       True
 3      False
 4      False
        ...
 297     True
 298     True
 299     True
 300     True
 301     True
 Name: heart_disease, Length: 297, dtype: bool,
 0      0
 1      2
 2      1
 3      0
 4      0
       ..
 297    1
 298    1
 299    2
 300    3
 301    1
 Name: num, Length: 297, dtype: int64)
```

In [187]:
```
# New test train set
x2_train, x2_test, y2_train, y2_test = train_test_split(heart_clean.dr
                                                        heart_clean.heart_

model2 = DecisionTreeClassifier(criterion='entropy')

model2.fit(x2_train, y2_train)
```

Out[187]:
```
▾          DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy')
```

In [188]:
```
# Test Predictions

test2_predictions = model2.predict(x2_test)
```

In [189]:
```
# Accuracy

accuracy_score(y2_test, test2_predictions)
```

Out[189]:
```
0.7166666666666667
```

In [190]:
```
# Confusion Matrix

confusion_matrix(y2_test, test2_predictions)
```

Out[190]:
```
array([[24, 10],
       [ 7, 19]])
```

In [191]:
```
# Classification Report

print(classification_report(y2_test, test2_predictions))

# Better; up from 50% but probably still some room to improve
```

```
              precision    recall  f1-score   support

       False       0.77      0.71      0.74        34
        True       0.66      0.73      0.69        26

    accuracy                           0.72        60
   macro avg       0.71      0.72      0.71        60
weighted avg       0.72      0.72      0.72        60
```