Data Mining and Machine Learning Report

Blaine Burke

Link to dataset: https://www.kaggle.com/datasets/mirichoi0218/insurance

1. Regression

1.1. Business Understanding

The health insurance data set I chose allows medical business to predict the charges of someone's health insurance. This allows them as a business to estimate their revenue from insurance and allows them to provide better care for their patients.

By preforming linear regression analysis on the dataset, we can answer many different questions such as:

- Is the price affected by being a smoker?
- Is the price affected by having children different for each sex?
- Is the price affected by age?

1.2. Data Understanding & Preparation

I got this data set from Kaggle.com. There are 7 columns in the dataset with 1138 rows of data. Here is an extract from the data.

age	sex	bmi	children	smoker	region	charges
19	female	27.9	0	yes	southwest	16884.92
18	male	33.77	1	no	southeast	1725.552
28	male	33	3	no	southeast	4449.462

- Age: The current age of the insured person
- Sex: The sex of the insured person
  - Male, Female
- BMI: The current bmi of the insured person
- Children: The current number of children insured
  - 0 - 5
- Smoker: Does the insured person smoke
  - Yes, No
- Region: What region is the insured person from
  - northeast, southeast, southwest, northwest.
- Charges: The cost of the insurance cover.
  - Fortunately, there is no missing data in this dataset.

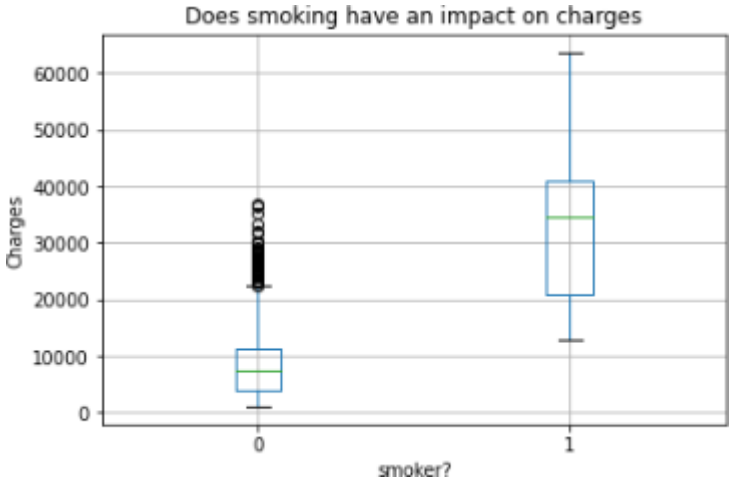
```
In [108]: # Ensure that there is no missing data
          df = pd.read_csv("data/insurance.csv")
          df.count()

Out[108]: age         1338
          sex         1338
          bmi         1338
          children    1338
          smoker      1338
          region      1338
          expenses    1338
          dtype: int64
```

Before I began modelling, I did some analysis on the data as to get a more in depth look at the features and their relationship with the charges.

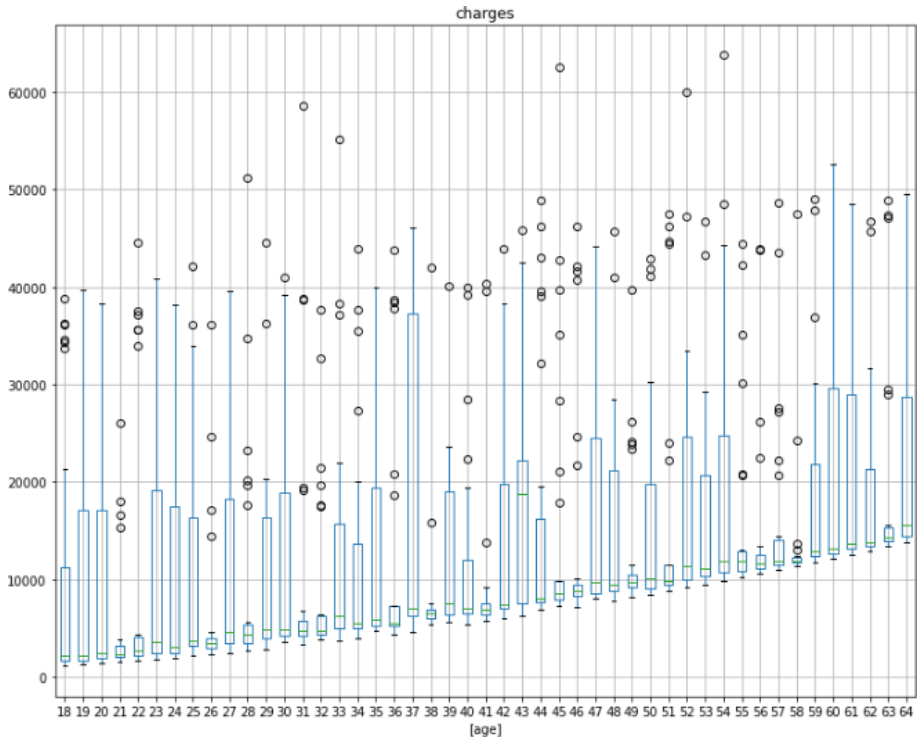
Does Smoker effect Charges?

In this boxplot, we can see that smoking has a large impact on costs. There is a clear skew in the distributions of charges between smokers and non smokers with non smokers averaging at around $8,000 and smokers averaging at aproximatly $35,000.



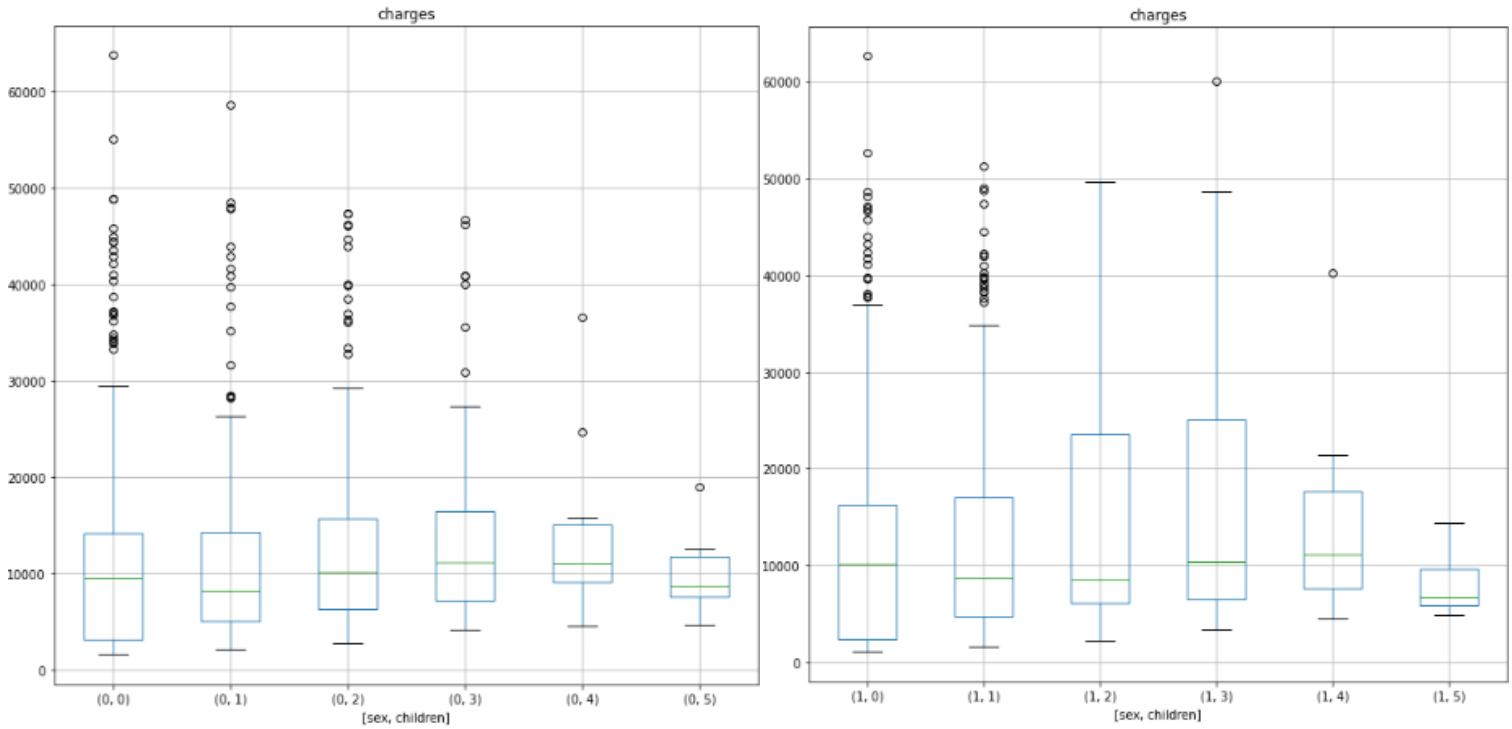
Does Age effect Charges?

By plotting the age of a person against the charges, we can see that there is a slight increase in charges as someone get older. This is to be expected as people are more susceptible to becoming ill as they get older. However, people can still become ill when young as can be seen through the outliers on the boxplot.



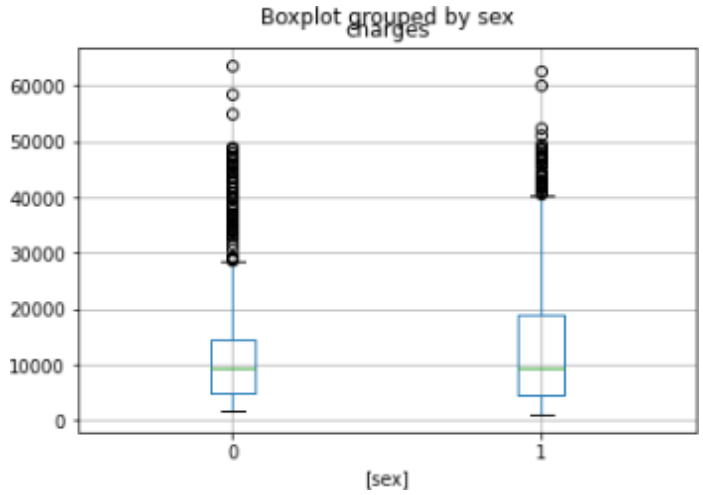
Does Children and sex effect charges?

These plots are an indication that children impact your charges depending on sex if you have more than 1 child. As can be seen below, sex 1 has higher charges when having 2 children or more. This is likely due to the fact that women bear children, increasing their risk of health complications.



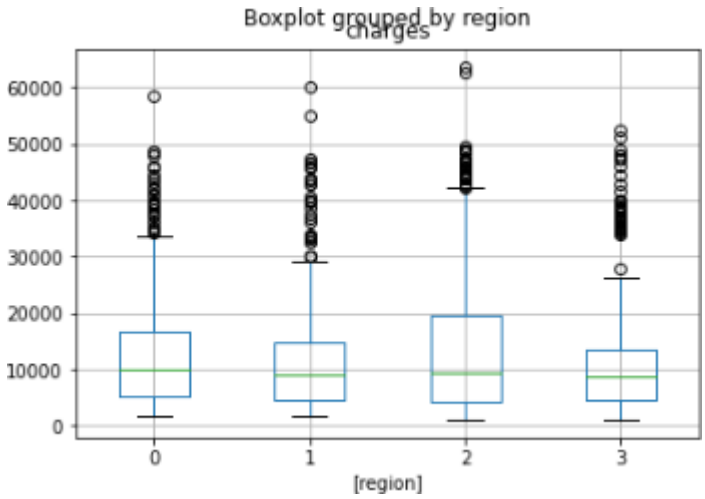
Does Sex effect charges?

It appears that there is a slight effect on the charges depending on sex. This is likely due to women bearing children as shown in the boxplots above.



Does Region effect charges?

It appears that there is a no effect of region on a person's charges.



1.3. Modelling

When modelling the data, I wished to predict the charges, so I added it as my target variables. When training the data, I used stratify=X['smoker'], to get an equal proportion of smokers and non-smokers.

```
In [139]: X_train, X_test, y_train, y_test = train_test_split( X, y,stratify=X['smoker'], random_state=2)
```

I then created a null predictor function that returns a list of the average charges. I can then use this to get the null RMSE which I can later use to check if my model is better or worse.

```
def nullPredictor(X):
    average = []
    average.append((sum(df['charges']) / len(df['charges'])))
    return [average for X_val in X]

# Null model for regression is the avg of the target.
# As long as model is better than the null value it's good.
```

```
y_null_hat = nullPredictor(y_test)
print("Null RMSE: ",mean_squared_error(y_test, y_null_hat, squared=False))
```

```
Null RMSE:  11584.776450511354
```

For my first model, I used all the data in the dataset to attempt to predict the charges. This model was good with an r-squared value of 0.75 and a root mean squared error value of 5819.72 which is much lower than the null rmse, meaning that the model has predictive power.

```
model = LinearRegression()
model.fit(X_train, y_train)

#As seen in the slopes, sex and region do not have an effect on charges.
# -199.2047536, -453.30789577
print('slopes:', model.coef_)
print('R squared:', model.score(X,y))

yhat = model.predict(X_test)
print('RMSE', mean_squared_error(y_test, yhat, squared=False))
```

```
slopes: [  251.96294361  -199.2047536    341.80626768   520.45449159
 24157.60324768  -453.30789577]
R squared: 0.7504206308636931
RMSE 5819.723034630363
```

For my second model, I wished to find the best possible set of features to predict the charges. To do this, I created a list of lists of all the possible subset combinations of the data. I then trained each subset and recorded their RMSE values in a dictionary. This allows me to find which subset has the lowest RMSE value which was subset 48 with an RMSE of 5800.38.

```
model = LinearRegression()
#Create a dictionary to store the subset number and the rmse value
results = {'feature_set':[],'rmse':[]}

for i, feature in enumerate(testList):

    if len(feature) ==0:
        continue

    X = df[feature]
    y = df['charges']

    if 'smoker' in feature:
        stratify_var = X['smoker']
    else:
        stratify_var = None

    #Train data using the different subsets of features
    X_train, X_test, y_train, y_test = train_test_split(X, y,stratify = stratify_var,random_state=2)

    model.fit(X_train, y_train)

    yhat = model.predict(X_test)

    #Add feature set number and rmse value to list
    results['feature_set'].append(i)
    results['rmse'].append(mean_squared_error(y_test, yhat, squared=False))
```

```
pd.DataFrame(results).sort_values(by = "rmse").head()
```

	feature_set	rmse
47	48	5800.382753
56	57	5801.160092
60	61	5817.981402
62	63	5819.723035
26	27	5819.912608

The subset contains the features age bmi, children and smoker. And when I trained this data, it had an r-squared value of 0.75.

```
model = LinearRegression()

X = df[testList[48]]
y = df['charges']

X_train, X_test, y_train, y_test = train_test_split(X, y,stratify=X['smoker'], random_state=2)

model.fit(X_train, y_train)

#print('intercept:', model.intercept_)
print('slopes:', model.coef_)
print('R squared:', model.score(X,y))

yhat = model.predict(X_test)
print('RMSE', mean_squared_error(y_test, yhat, squared=False))
```

```
slopes: [  254.14892008   327.25029253   494.37878052 24102.55705501]
R squared: 0.7495374581681424
RMSE 5800.382753063383
```

```
print(testList[48])
```

```
['age', 'bmi', 'children', 'smoker']
```

1.4. Evaluation

I was able to create a good model that can predict the charges of someone's health insurance and answers many business questions such as, is the price affected by being a smoker or is the price affected by age as seen in section 1.2.
Both model one and two are good but model is two is better as it uses less data, has the same r-squared value and a lower RMSE value. The data set ['age', 'bmi', 'children', 'smoker'] was the most accurate data set with a rmse value of 5800.

2. Decision Trees

2.1. Business Understanding

The health insurance data set I chose allows medical business to predict the charges of someone's health insurance. This allows them as a business to work through choices to determine the best outcomes for their organization.

2.2. Data Understanding & Preparation

See section 1.2.

2.3. Modelling

I used smoker as my target for Decision tree analysis.

```
X = df.drop('smoker', axis='columns')
y = df['smoker']
print(y.shape)
```

```
(1338,)
```

When preforming Cross validation, the results were very high, ranging from 0.93 to 0.96.

```
# Perform cross validation varying the depth of the tree.
# Make predictions in each loop and calculate F1 and print

# Using F1 because of the unbalanced dataset(80% = no)

for d in range(2,12):
    tree = DecisionTreeClassifier(max_depth=d)
    scores = cross_val_score(tree, X_train, y_train)
    print("Depth: ", d, "Accuracy:", scores.mean(), "F1: ",f1_score(y_test, y_hat, average=None))
```

```
Depth:  2 Accuracy: 0.9282288557213931 F1:  [0.96461825 0.85714286]
Depth:  3 Accuracy: 0.9601194029850746 F1:  [0.96461825 0.85714286]
Depth:  4 Accuracy: 0.9601243781094526 F1:  [0.96461825 0.85714286]
Depth:  5 Accuracy: 0.9630995024875622 F1:  [0.96461825 0.85714286]
Depth:  6 Accuracy: 0.9650995024875622 F1:  [0.96461825 0.85714286]
Depth:  7 Accuracy: 0.9661094527363183 F1:  [0.96461825 0.85714286]
Depth:  8 Accuracy: 0.9661144278606963 F1:  [0.96461825 0.85714286]
Depth:  9 Accuracy: 0.9661044776119402 F1:  [0.96461825 0.85714286]
Depth:  10 Accuracy: 0.959139303482587 F1:  [0.96461825 0.85714286]
Depth:  11 Accuracy: 0.9641194029850746 F1:  [0.96461825 0.85714286]
```

I then created a null predictor function that returns a list full of the string 'no'. I can then use this to get the null accuracy and null F1.

```
def nullPredictor(X):
    return ['no' for X_val in X]

# Null model for regression is the avg of the target.
# As long as model is better than the null value it's good.
```

```
tree = DecisionTreeClassifier(max_depth=7)
tree = tree.fit(X_train,y_train)

#Predict the response for test dataset
y_hat = tree.predict(X_test)
print("Test Accuracy: ", accuracy_score(y_test, y_hat))
print("Test F1: ",f1_score(y_test, y_hat, average=None))
```

```
Test Accuracy:  0.9522388059701492
Test F1:  [0.97014925 0.88059701]
```

```
y_null_hat = nullPredictor(y_test)

print("Null Accuracy: ",accuracy_score(y_test, y_null_hat))
print("Null F1: ",f1_score(y_test, y_null_hat, average=None))
```

```
Null Accuracy:  0.7940298507462686
Null F1:  [0.88519135 0.        ]
```

2.4. Evaluation

As seen below, the Test Accuracy and Test F1 values are higher than the null accuracy and null f1 values. This means the model is good and has predictive power.


```
def nullPredictor(X):
    return ['no' for X_val in X]

# Null model for regression is the avg of the target.
# As long as model is better than the null value it's good.
```

```
tree = DecisionTreeClassifier(max_depth=7)
tree = tree.fit(X_train,y_train)

#Predict the response for test dataset
y_hat = tree.predict(X_test)
print("Test Accuracy: ", accuracy_score(y_test, y_hat))
print("Test F1: ",f1_score(y_test, y_hat, average=None))
```

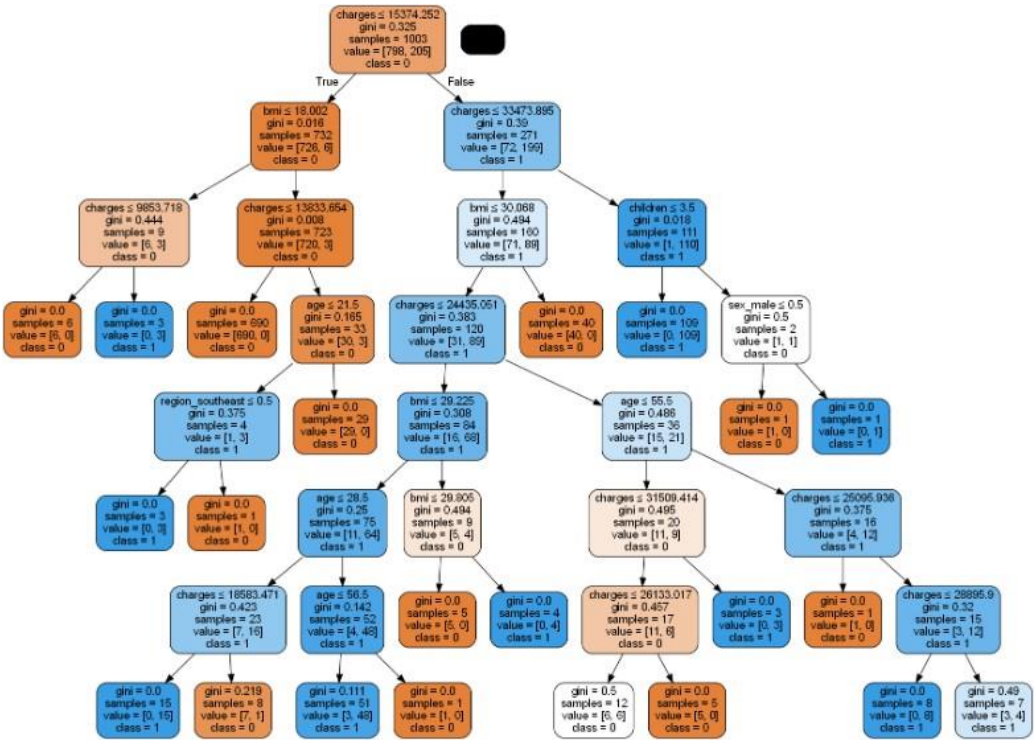
```
Test Accuracy:  0.9522388059701492
Test F1:  [0.97014925 0.88059701]
```

```
y_null_hat = nullPredictor(y_test)

print("Null Accuracy: ",accuracy_score(y_test, y_null_hat))
print("Null F1: ",f1_score(y_test, y_null_hat, average=None))
```

```
Null Accuracy:  0.7940298507462686
Null F1:  [0.88519135 0.        ]
```


In the decision tree, there are a lot of pure nodes with a gini value of 0.0. This is due to the model accurately predicting the target ‘smoker’.



3. kNN

3.1. Business Understanding

See section 2.1.

3.2. Data Understanding & Preparation

See section 1.2.

3.3. Modelling

I used smoker as my target for my kNN analysis.

```
X = df.drop('smoker', axis='columns')
y = df['smoker']
print(y.shape)
```

```
(1338,)
```

3.4. Evaluation

The null accuracy is lower than the Test accuracy, indicating a good, predictive model.

In [155]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5, stratify=y)

clf = KNeighborsClassifier()
clf.fit(X_train, y_train)
print('Test Accuracy', clf.score(X_train, y_train))
```

```
Test Accuracy 0.9401869158878504
```

In [161]:
```
y_null_hat = nullPredictor(y_train)

print("Null Accuracy: ",clf.score(X_train, y_null_hat))
```

```
Null Accuracy:  0.780373831775701
```

The confusion matrix analysis shows an accuracy of 0.94.

```
y_hat = clf.predict(X_test)
cm = confusion_matrix(y_test, y_hat)

tn, fp, fn, tp = cm.ravel()

print("True negatives:", tn, ", False Positives:", fp, ", False Negatives:", fn, ", True Positives:",
print("Confusion Matrix accuracy",(tn + tp) / (tn+fp+fn+tp))
```

```
True negatives: 203 , False Positives: 10 , False Negatives: 7 , True Positives: 48
Confusion Matrix accuracy 0.9365671641791045
```

When scaling the data, the models got high values. The standard scaler had a higher result that the minmax scaler, of 0.94 to 0.91. The test accuracy was 0.96, this is a very high result.

```
# Evaluate min max scaler
scaler = MinMaxScaler()

scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)

clf = KNeighborsClassifier()

scores = cross_val_score(clf, X_train_scaled, y_train, cv=5)
print('Validation accuracy with MinMaxScaler', scores.mean())
```

```
Validation accuracy with MinMaxScaler 0.9149532710280374
```

```
# Evaluate standard scaler
scaler = StandardScaler()

scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)

clf = KNeighborsClassifier()

scores = cross_val_score(clf, X_train_scaled, y_train, cv=5)
print('Validation accuracy with StandardScaler data', scores.mean())
```

```
Validation accuracy with StandardScaler data 0.9383177570093458
```

```
# Finally estimate the test accuracy
scaler = StandardScaler()

scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

clf = KNeighborsClassifier()
clf.fit(X_train_scaled, y_train)
print('Test Accuracy', clf.score(X_test_scaled, y_test))
```

```
Test Accuracy 0.9552238805970149
```