Project : Data Visualization, Choosing K-value and Appreciating Feature Scaling and Standardization

Binary Classification: Diabetes Dataset

Using ML techniques to predict whether a Pima Indian Woman has diabetes or not, based on information about the patient such as blood pressure, body mass index (BMI), age, etc.

Introduction

Scientists carried out a study to investigate the significance of health-related predictors of diabetes in **Pima Indian Women**. The study population was females (21 years and above) of Pima Indian heritage.

The purpose of the study was to find out the factors that are associated with the presence of diabetes in Pima Indians.

To find out the reason behind this, we have to first analyze the relationship between different features, such as the number of times a woman was pregnant, their BMI, prevalence of diabetes, etc.

Exploratory Data Analysis (EDA) and Statistical Analysis

Import Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

sns.set_style('whitegrid')

import warnings
warnings.filterwarnings('ignore')
```

```
# Upload the preprocessed diabetes data CSV file that has been shared with you.
# Run this cell, click on the 'Choose files' button and upload the file.
from google.colab import files
uploaded = files.upload()
```

Browse... No files selected. Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
diabetes_data = pd.read_csv('preprocessed_diabetes_data.csv')
```

```
# View top 10 rows of the Diabetes dataset
diabetes_data.head(10)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesP
0	6	148.0	72.000000	35.0	125.0	33.6	
1	1	85.0	66.000000	29.0	125.0	26.6	
2	8	183.0	64.000000	29.0	125.0	23.3	
3	1	89.0	66.000000	23.0	94.0	28.1	
4	0	137.0	40.000000	35.0	168.0	43.1	
5	5	116.0	74.000000	29.0	125.0	25.6	
6	3	78.0	50.000000	32.0	88.0	31.0	
7	10	115.0	72.405184	29.0	125.0	35.3	
8	2	197.0	70.000000	45.0	543.0	30.5	
9	8	125.0	96.000000	29.0	125.0	32.3	

Identification of variables and data types

```
diabetes_data.shape
```

```
(768, 9)
```

Dataset comprises of 768 observations and 9 fields.

The following features have been provided to help us predict whether a person is diabetic or not:

- **Pregnancies:** Number of times pregnant
- **Glucose:** Plasma glucose concentration over 2 hours in an oral glucose tolerance test. Less than 140 mg/dL is considered normal level of glucose.
- **BloodPressure:** Diastolic blood pressure (mm Hg). 120/80 is normal BP level for females above 18 years old.
- **SkinThickness:** Triceps skin fold thickness (mm)
- **Insulin:** 2-Hour serum insulin (mu U/ml). 16-166 mIU/L is considered the normal level of insulin.
- **BMI:** Body mass index (weight in kg/((height in m$)^2$))
- **DiabetesPedigreeFunction:** Diabetes pedigree function (a function which scores likelihood of diabetes based on family history)
- **Age:** Age (in years)
- **Outcome:** Class variable (0 if non-diabetic, 1 if diabetic)

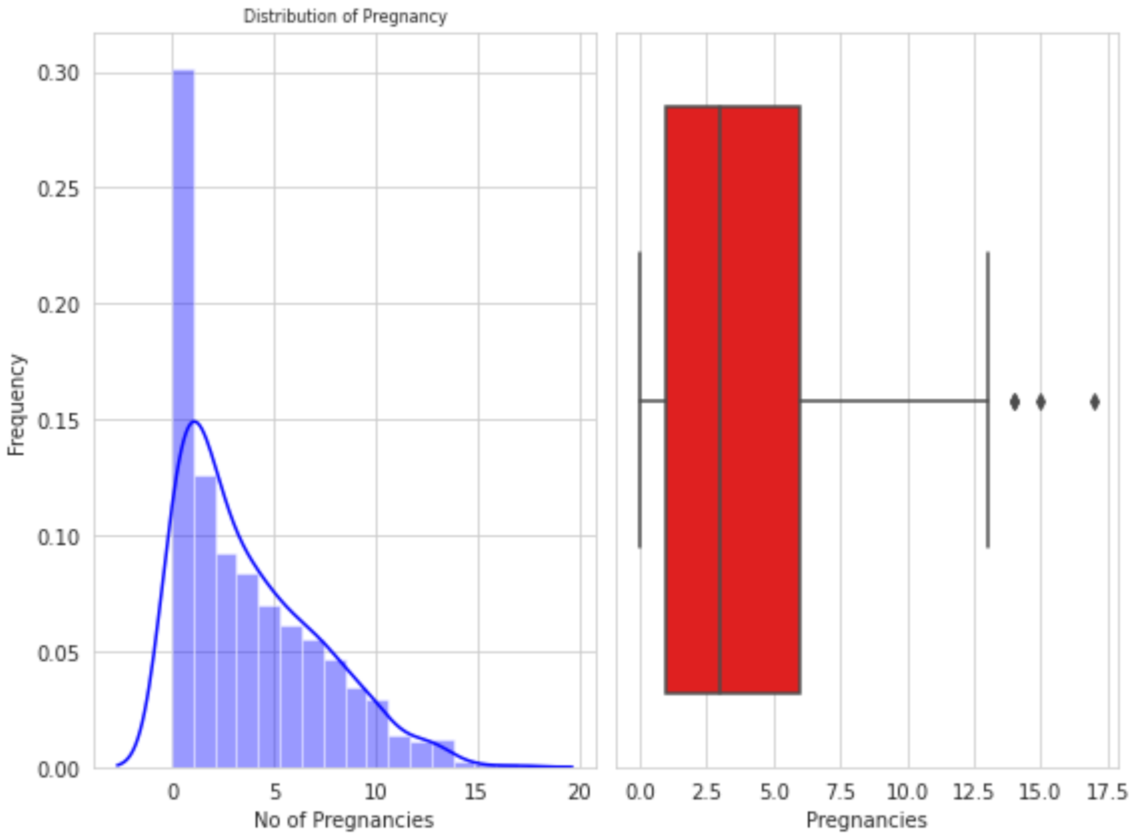
```
# Get the details of each column
diabetes_data.describe().T
```

	count	mean	std	min	25%	
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3
Glucose	768.0	121.686763	30.435949	44.000	99.75000	117
BloodPressure	768.0	72.405184	12.096346	24.000	64.00000	72
SkinThickness	768.0	29.108073	8.791221	7.000	25.00000	29
Insulin	768.0	140.671875	86.383060	14.000	121.50000	125
BMI	768.0	32.455208	6.875177	18.200	27.50000	32
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0
Age	768.0	33.240885	11.760232	21.000	24.00000	29
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0

Let us see distribution and also boxplot for outliers of feature "Pregnancies".

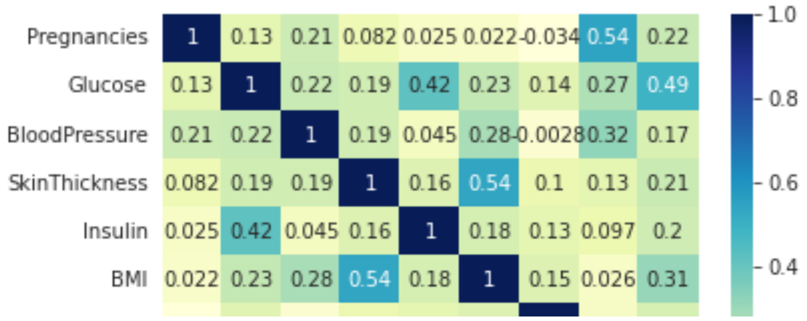
```
fig,axes = plt.subplots(nrows=1,ncols=2,figsize = (8,6))

plot00=sns.distplot(diabetes_data['Pregnancies'],ax=axes[0],color='b')
axes[0].set_title('Distribution of Pregnancy',fontdict={'fontsize':8})
axes[0].set_xlabel('No of Pregnancies')
axes[0].set_ylabel('Frequency')
plt.tight_layout()
```

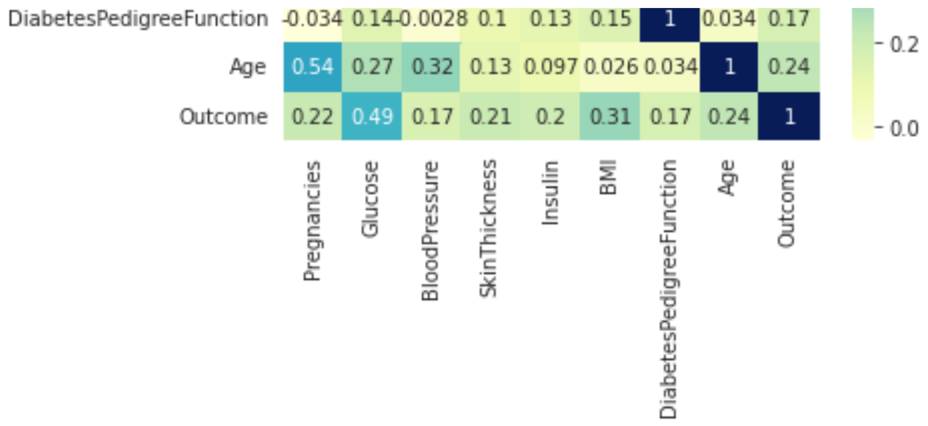
```
plot01=sns.boxplot('Pregnancies',data=diabetes_data,ax=axes[1],orient = 'v', color=
plt.tight_layout()
```



```
## Find out the "Correlation" between the different attributes present in the data.
## Also plot a heatmap (refer Seaborn documentation) for the correlation values obt
from scipy.stats import pearsonr

corr, _ = pearsonr(diabetes_data['Age'], diabetes_data['Pregnancies'])
print('Correlation: %.3f \n' % corr)

# plotting the heatmap
print('Heatmap for the correlation values:')
diabetes_data_plot = sns.heatmap(diabetes_data.corr(), cmap="YlGnBu", annot=True)
```

```
    Correlation: 0.544

    Heatmap for the correlation values:
```





```
diabetes_data.corr()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness
Pregnancies	1.000000	0.127911	0.208522	0.081770
Glucose	0.127911	1.000000	0.218367	0.192686
BloodPressure	0.208522	0.218367	1.000000	0.191853
SkinThickness	0.081770	0.192686	0.191853	1.000000
Insulin	0.025047	0.419064	0.045087	0.155610
BMI	0.021559	0.231128	0.281199	0.543205
DiabetesPedigreeFunction	-0.033523	0.137060	-0.002763	0.102188
Age	0.544341	0.266534	0.324595	0.126107
Outcome	0.221898	0.492928	0.166074	0.214873

Observations

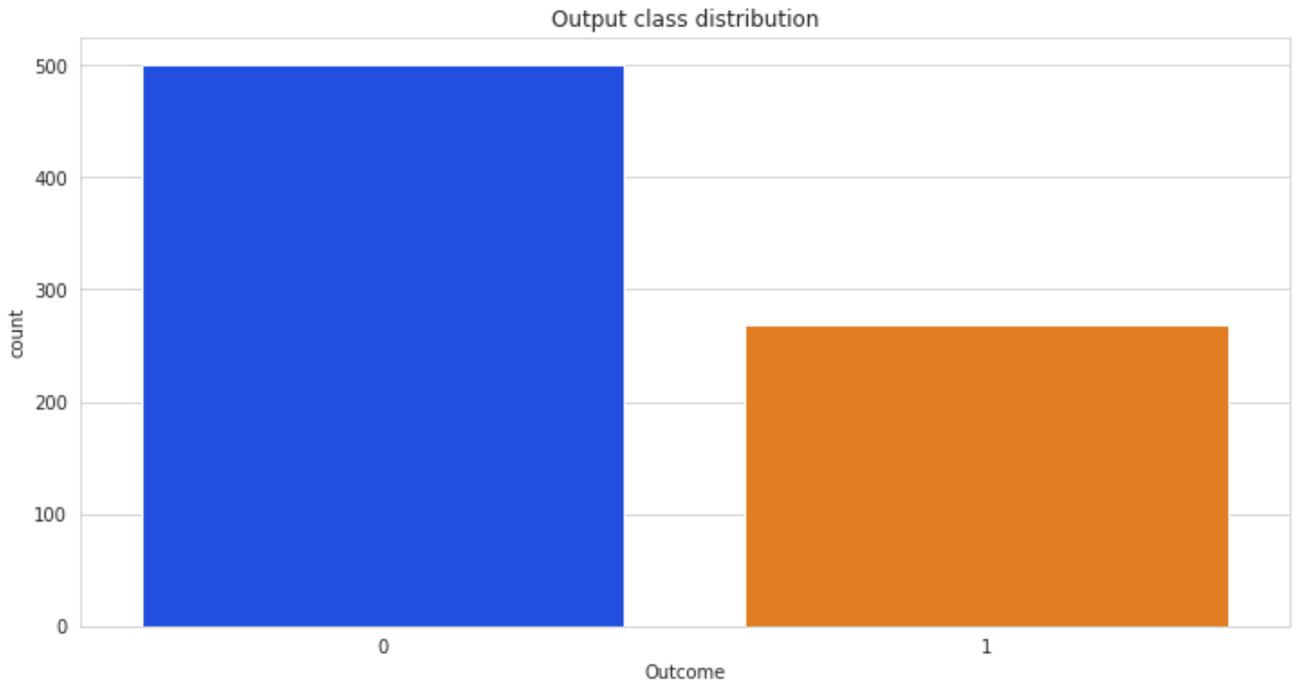
- From the correlation map you just obtained above, it seems that Insulin is highly correlated with Glucose, BMI and Age. It means that as the values of glucose, BMI and Age increase, the insulin is also increasing. It seems logical also that overweight and elderly people might have a higher level of insulin in their bodies.
- In the same way SkinThickness is highly correlated with BMI.

Checking if the data is balanced or imbalanced

We can produce a seaborn count plot to check if the output is dominated by one of the classes or not.

```
plt.figure(figsize=(12,6))
sns.countplot(x='Outcome',data=diabetes_data, palette='bright')
plt.title("Output class distribution")

print(diabetes_data['Outcome'].value_counts())
```

```
0    500
1    268
Name: Outcome, dtype: int64
```



Observations

A total of 768 women were registered in the database. 268 women had diabetes, while 500 women did not have diabetes.

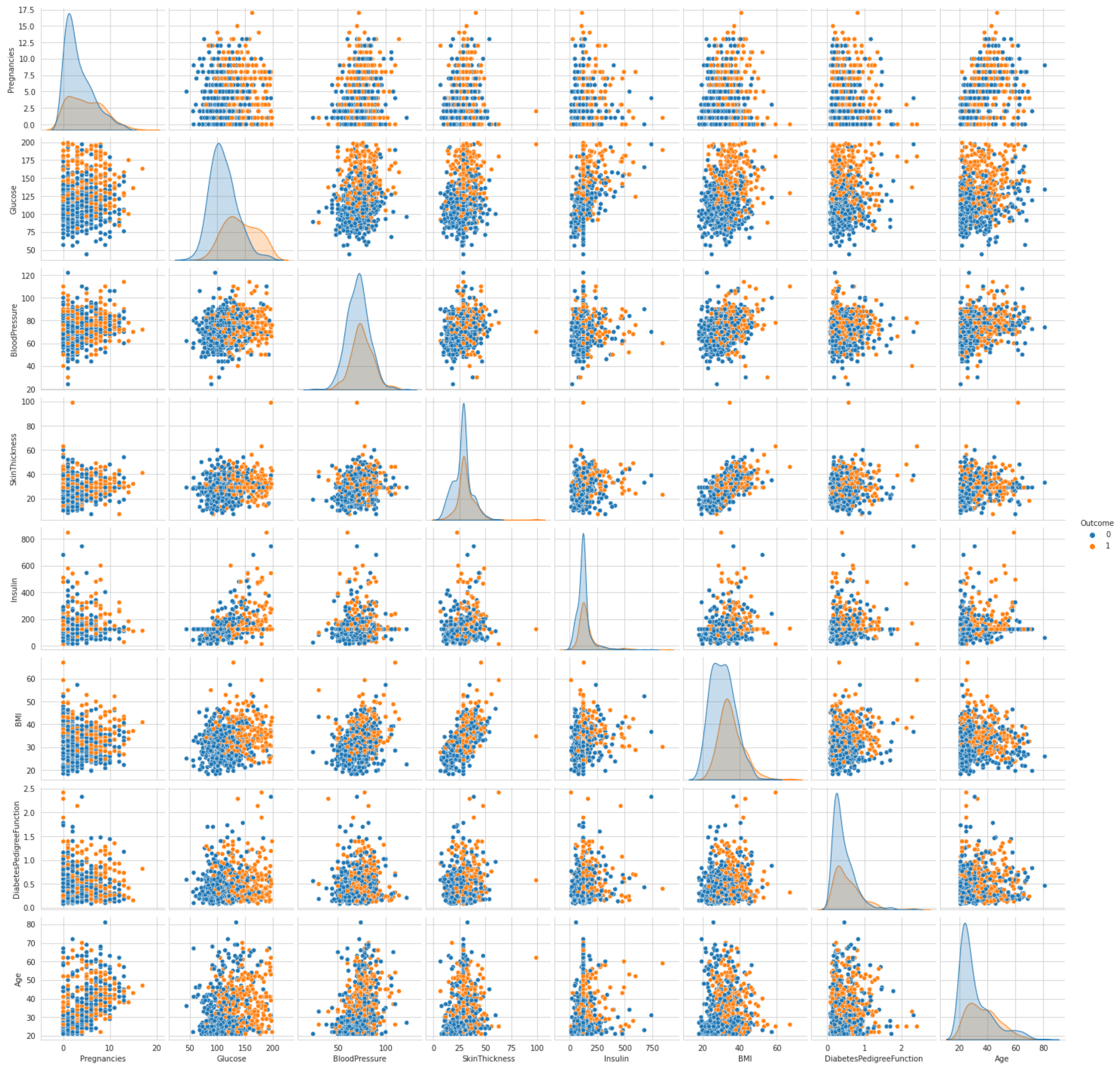
The above graph shows that the dataset is biased towards non-diabetic people. The number of non-diabetic people is almost twice the number of diabetic patients.

Scatter matrix of data

A pair-plot builds on two basic figures, the histogram and the scatter plot. The histogram on the diagonal allows us to see the distribution of a single variable while the scatter plots on the upper and lower triangles show the relationship (or lack thereof) between two variables.

```
##Display a pairplot using Seaborn for the diabetes dataset, with the 'outcome' as
```

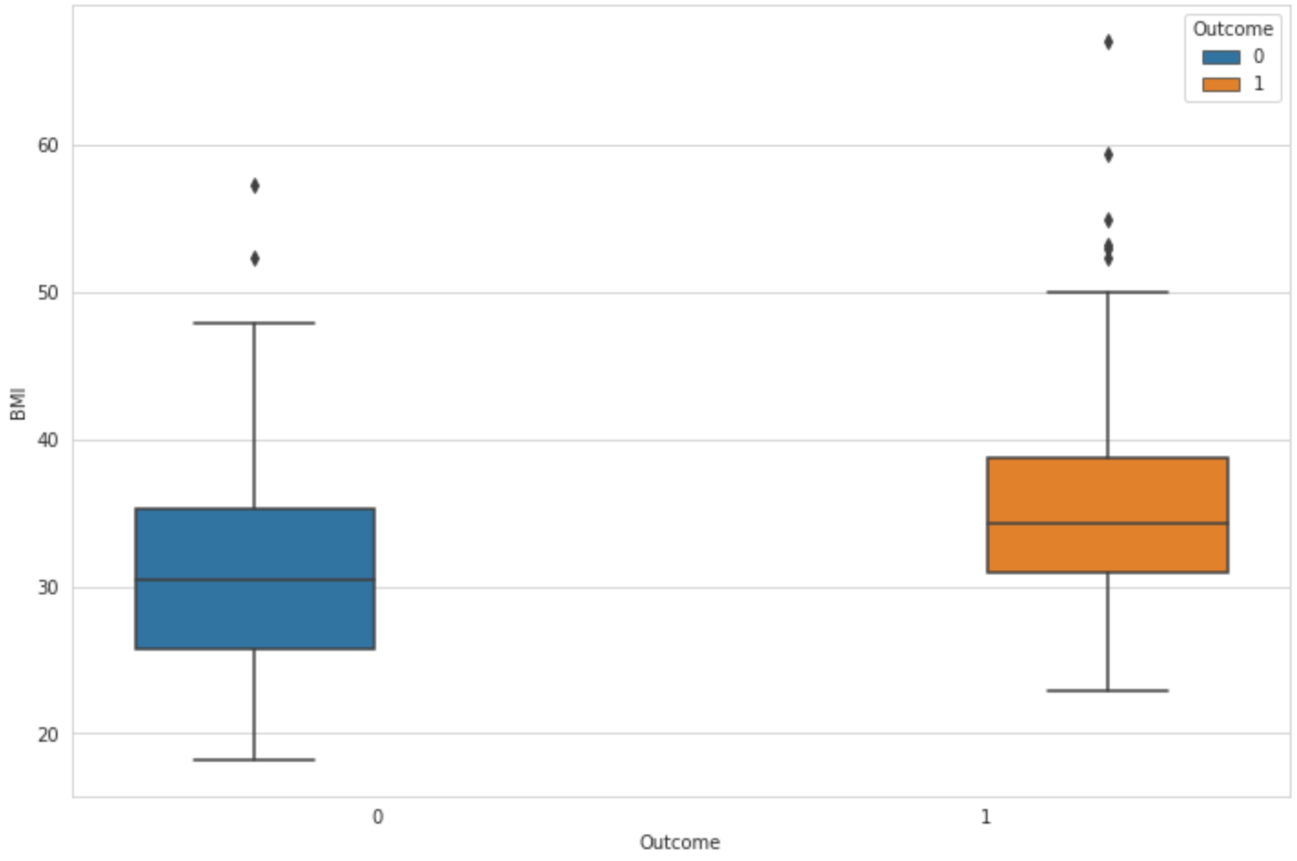
```
sns.pairplot(diabetes_data, hue = 'Outcome', data = diabetes_data)
plt.show()
```



BMI vs Outcome

```
plt.figure(figsize=(12,8))
sns.boxplot(x='Outcome', y='BMI',data=diabetes_data, hue='Outcome')
```

```
    <matplotlib.axes._subplots.AxesSubplot at 0x7f2b2c876590>
```



Observations

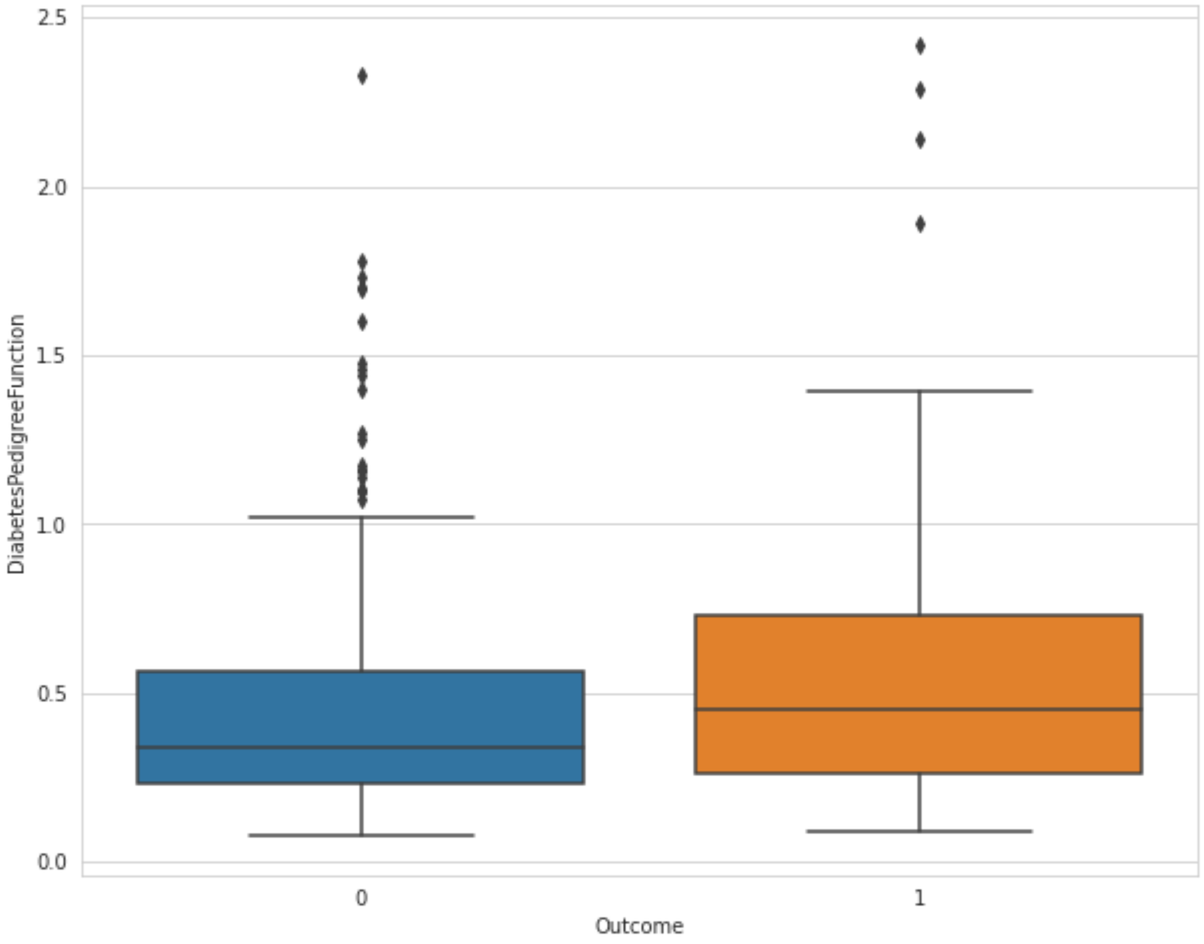
It is surprising that the median BMI does not significanty change as the number of pregnancies increases. Those who tested positive for diabetes had higher BMIs than those who did not. However,there is not a very large difference between the medians.

BMI might be higher for women who have had more numbers of pregnancies as well as for those who test positive for diabetes and that the relationship between the pedigree function and the test results will show that those who had a higher pedigree function tested positive and those who had a lower pedigree function tested negative.

Pedigree function vs Diabetes

```
##Display a boxplot between the Pedigree function and Diabetes.
plt.figure(figsize=(10,8))
sns.boxplot(x='Outcome', y='DiabetesPedigreeFunction',data=diabetes_data)
```

```
    <matplotlib.axes._subplots.AxesSubplot at 0x7f2b2c773510>
```



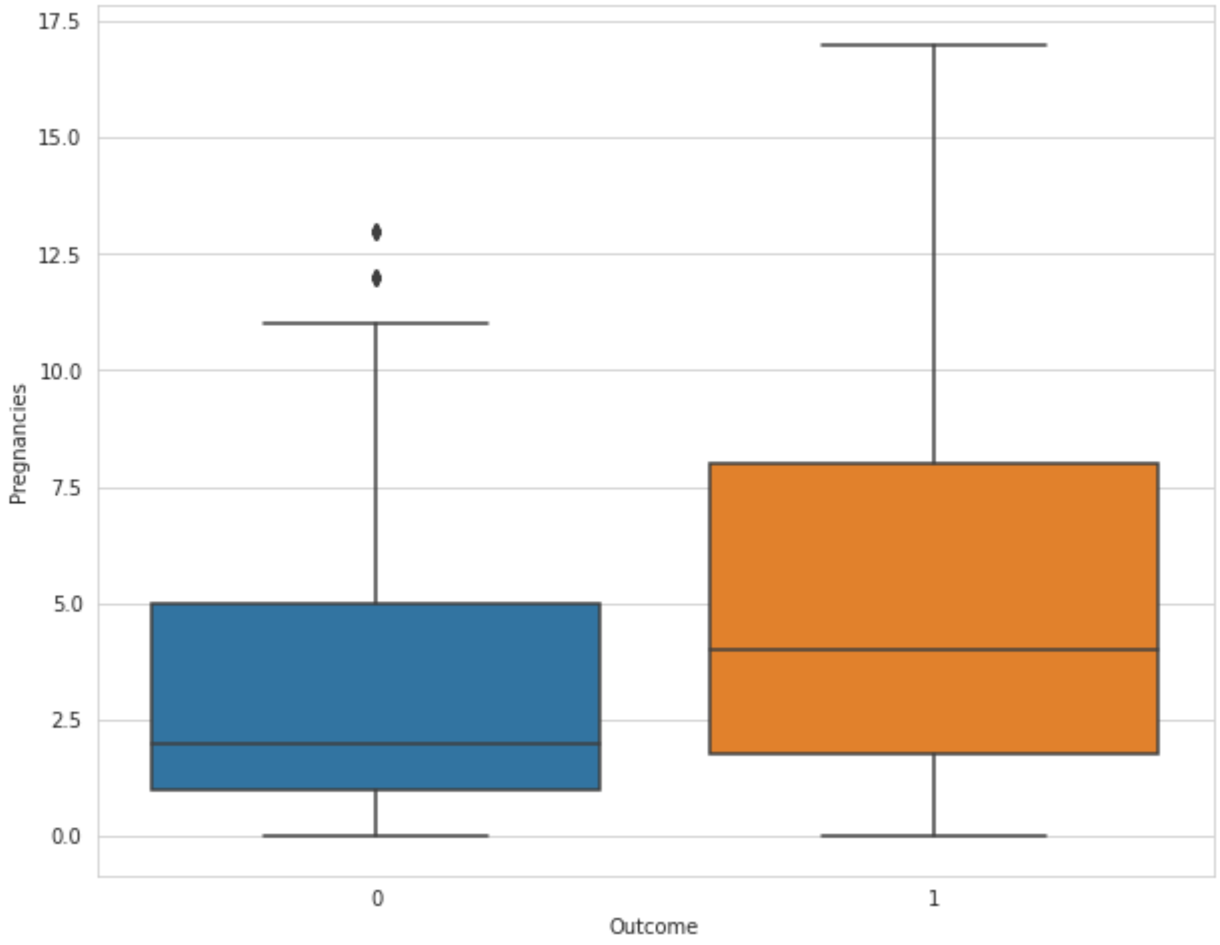
Observations

This graph more clearly shows the relationship between the pedigree function and the test results that the women got for diabetes. Since those who tested positive have a higher median and more high outliers, it is clear that the pedigree function does in fact, accurately help estimate the test results for diabetes. It shows that diabetes does follow genetics so those whose ancestors suffered from it have a higher risk of getting the disease themselves as well. Both test results show many outliers yet the outliers for those who tested negative seem to have lower pedigree functions than those who tested positive. This indicates that the genetic component is likely to contribute more to the emergence of diabetes in the Pima Indians and their offspring.

Pregnancy vs Diabetes

```
##Display a boxplot between the number of Pregnancies and Diabetes.
plt.figure(figsize=(10,8))
sns.boxplot(x='Outcome', y='Pregnancies',data=diabetes_data)
```

```
    <matplotlib.axes._subplots.AxesSubplot at 0x7f2b2c7a5410>
```



Observations

The average number of pregnancies is higher in diabetic as compared to non-diabetic women.

Prevalence of Diabetes vs BMI

Let's try to find out the prevalence of diabetes and its relation to their BMI. Please note that the range of normal BMI is 18.5 to 25.

```
normalBMIData = diabetes_data[(diabetes_data['BMI'] >= 18.5) & (diabetes_data['BMI'
normalBMIData['Outcome'].value_counts()
```

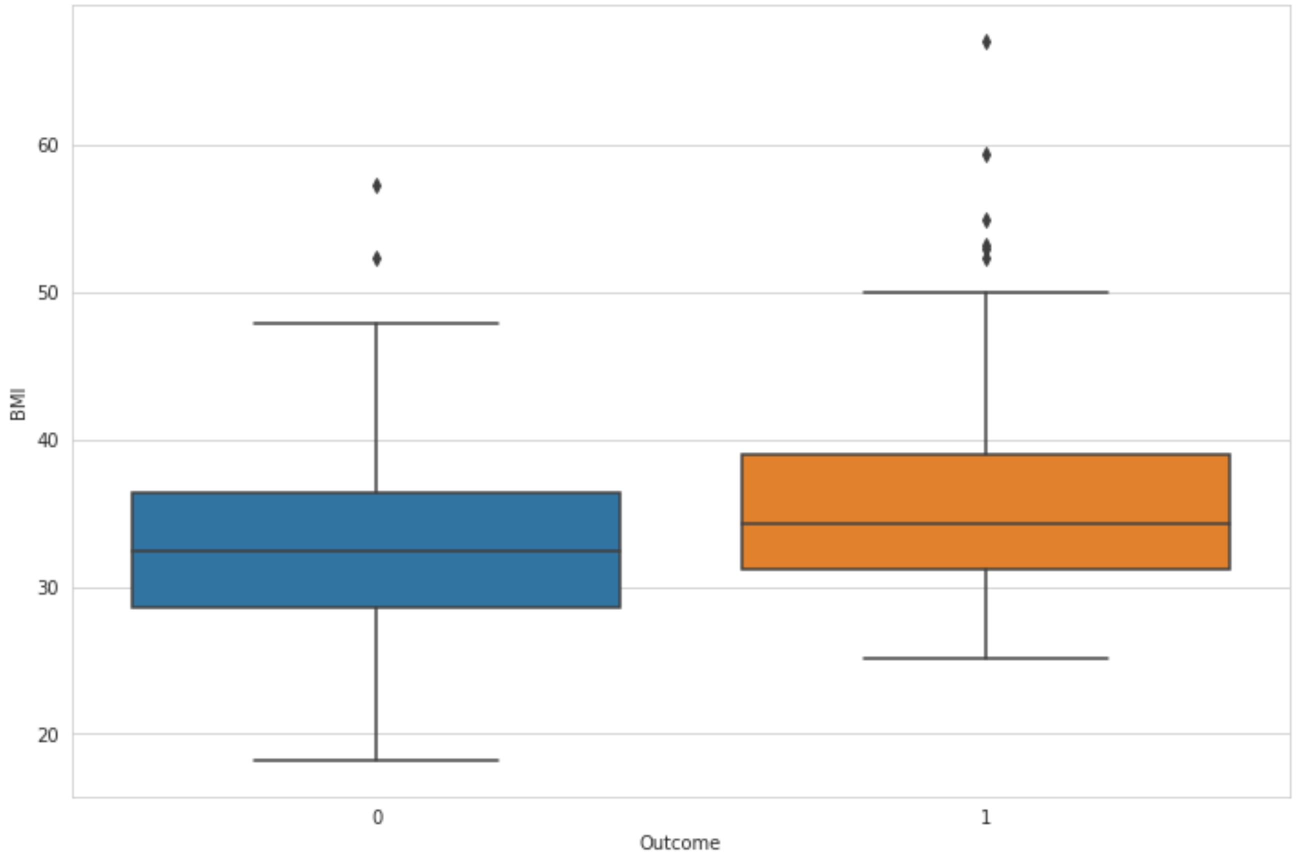
```
    0    101
    0    101
    1      7
    Name: Outcome, dtype: int64
```

```
notNormalBMIData = diabetes_data[(diabetes_data['BMI'] < 18.5) | (diabetes_data['BM
notNormalBMIData['Outcome'].value_counts()
```

```
    0    399
    1    261
    Name: Outcome, dtype: int64
```

```
plt.figure(figsize=(12,8))
sns.boxplot(x='Outcome', y='BMI',data=notNormalBMIData)
```

```
    <matplotlib.axes._subplots.AxesSubplot at 0x7f2b2c773650>
```



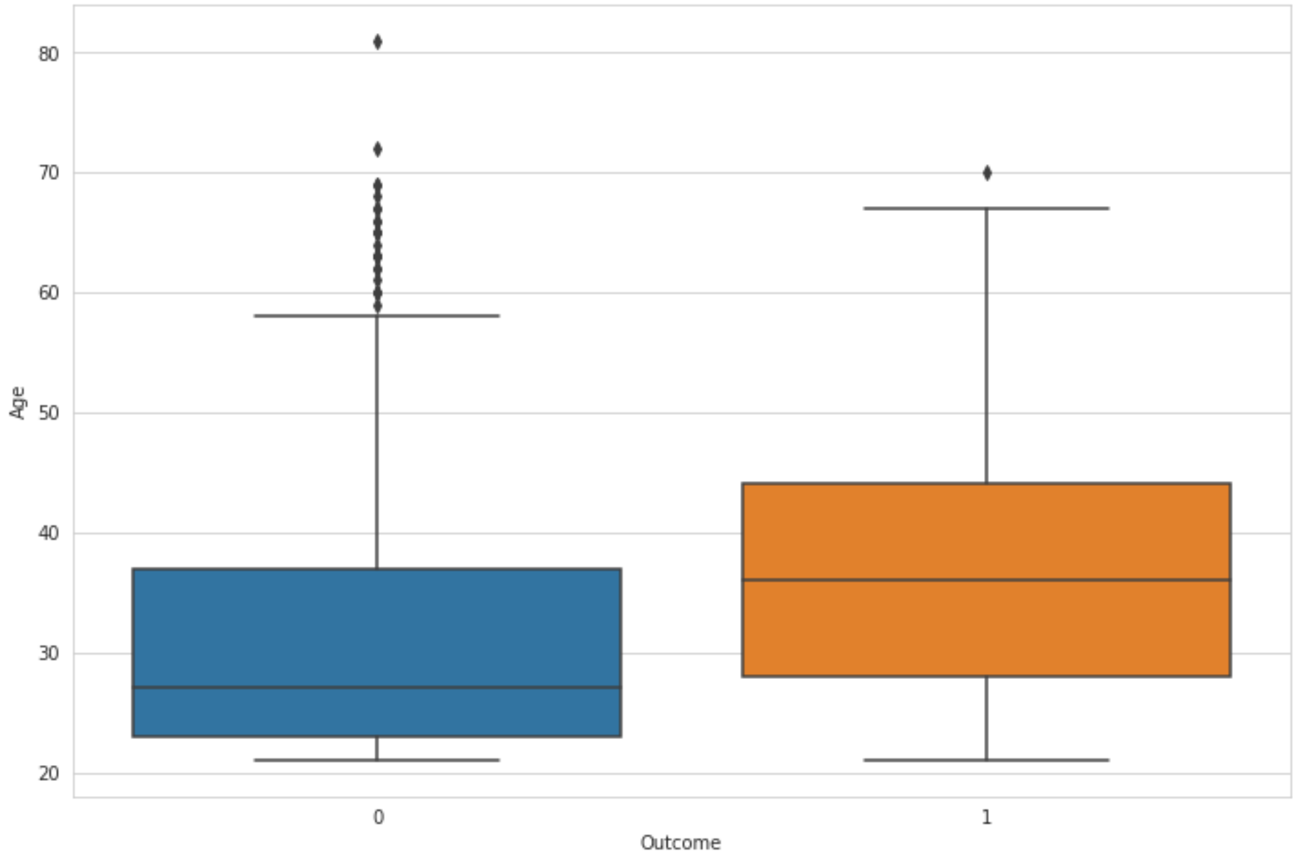
Observations

The Body Mass Index (BMI) shows a significant association with the occurrence of diabetes. The interquartile range for the women who tested positive reaches a higher BMI than the IQR for those who tested negative. Therefore, women could have higher BMIs and not be outliers if they tested positive as opposed to negative, showing that more women who tested positive did, in fact, have higher BMIs than those who tested negative.

Age vs Diabetes

```
## Display a boxplot between Age and Diabetes.
plt.figure(figsize=(12,8))
sns.boxplot(x='Outcome', y='Age',data=diabetes_data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b2c691c50>
```



Observations

A significant relation can be seen between the age distribution and occurrence of diabetes. Women at age group > 31 years were at higher risk of getting diabetes in comparison to the younger age group.

The Importance of Standardizing Data

```
unchanged_data = diabetes_data.drop('Outcome',axis=1)
```

```
unchanged_data
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete
0	6	148.0	72.0	35.0	125.0	33.6	
1	1	85.0	66.0	29.0	125.0	26.6	
2	8	183.0	64.0	29.0	125.0	23.3	
3	1	89.0	66.0	23.0	94.0	28.1	
4	0	137.0	40.0	35.0	168.0	43.1	
...	...	...	...	...	...	...	
763	10	101.0	76.0	48.0	180.0	32.9	
764	2	122.0	70.0	27.0	125.0	36.8	
765	5	121.0	72.0	23.0	112.0	26.2	
766	1	126.0	60.0	29.0	125.0	30.1	
767	1	93.0	70.0	31.0	125.0	30.4	

768 rows × 8 columns

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
```

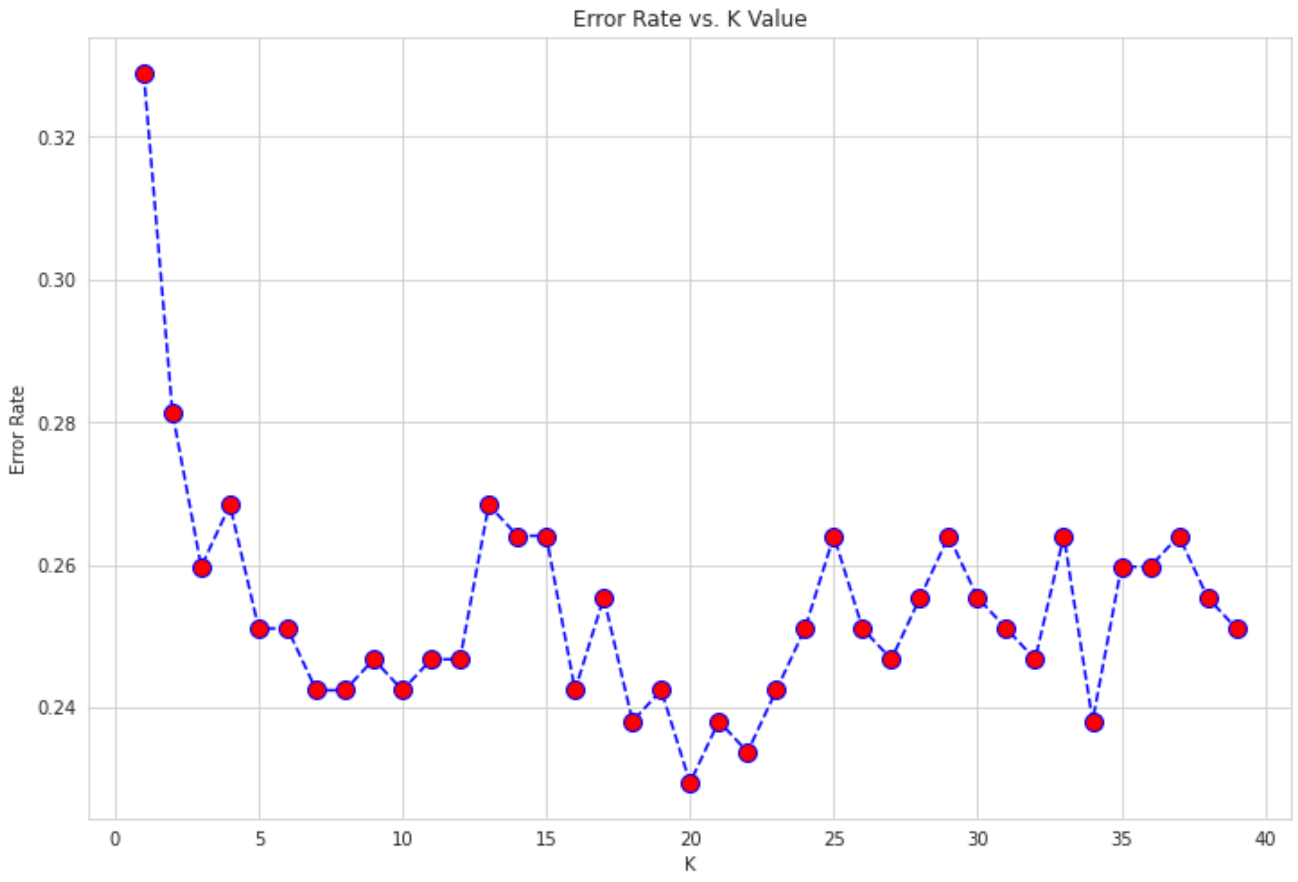
Choosing a K Value

Creating a for loop that trains various KNN models with different k values, then keep track of the error_rate for each of these models with a list.

```
def plot_KNN_error_rate(xdata,ydata):
  error_rate = []
  test_scores = []
  train_scores = []

  ## [REQUIRED] Split the data into train and test sets in a 70:30 ratio (70% trair
  X_train, X_test, y_train, y_test = train_test_split(xdata, ydata, test_size=0.3,

  for i in range(1,40):
      ## [REQUIRED] Complete the code in the next three lines
```

```
        knn = KNeighborsClassifier(n_neighbors=i)
        knn.fit(X_train, y_train)
        pred_i = knn.predict(X_test)

        error_rate.append(np.mean(pred_i != y_test))
        train_scores.append(knn.score(X_train,y_train))
        test_scores.append(knn.score(X_test,y_test))

    plt.figure(figsize=(12,8))
    plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
            markerfacecolor='red', markersize=10)
    plt.title('Error Rate vs. K Value')
    plt.xlabel('K')
    plt.ylabel('Error Rate')
    print()
    ## score that comes from testing on the same datapoints that were used for traini
    max_train_score = max(train_scores)
    train_scores_ind = [i for i, v in enumerate(train_scores) if v == max_train_score
    print('Max train score {} % and k = {}'.format(max_train_score*100,list(map(lambd
    print()
    ## score that comes from testing on the datapoints that were split in the beginni
    max_test_score = max(test_scores)
    test_scores_ind = [i for i, v in enumerate(test_scores) if v == max_test_score]
    print('Max test score {} % and k = {}'.format(max_test_score*100,list(map(lambda

    return test_scores



  def plot_KNN_error_rate(xdata,ydata):
    error_rate = []
    test_scores = []
    train_scores = []

    X_train, X_test, y_train, y_test = train_test_split(xdata, ydata, test_size=0.3,

    for i in range(1,40):
        knn = KNeighborsClassifier(n_neighbors=i)
        knn.fit(X_train, y_train)
        pred_i = knn.predict(X_test)

        error_rate.append(np.mean(pred_i != y_test))
        train_scores.append(knn.score(X_train,y_train))
        test_scores.append(knn.score(X_test,y_test))

    plt.figure(figsize=(12,8))
    plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
            markerfacecolor='red', markersize=10)
    plt.title('Error Rate vs. K Value')
    plt.xlabel('K')
    plt.ylabel('Error Rate')
    print()
    ## score that comes from testing on the same datapoints that were used for traini
    max_train_score = max(train_scores)
    train_scores_ind = [i for i, v in enumerate(train_scores) if v == max_train_score
    print('Max train score {} % and k = {}'.format(max_train_score*100,list(map(lambd
    print()
    ## score that comes from testing on the datapoints that were split in the beginni
    max_test_score = max(test_scores)
    test_scores_ind = [i for i, v in enumerate(test_scores) if v == max_test_score]
    print('Max test score {} % and k = {}'.format(max_test_score*100,list(map(lambda

    return test_scores


unchanged_test_scores = plot_KNN_error_rate(unchanged_data,diabetes_data['Outcome']
```

```
Max train score 100.0 % and k = [1]

Max test score 77.05627705627705 % and k = [20]
```



Standardize the Variables

Standardization (also called z-score normalization) is the process of putting different variables on the same scale. Standardization transforms your data such that the resulting distribution has a mean of 0 and a standard deviation of 1.

$$Z = \frac{X - \mu}{\sigma}$$

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
scaler.fit(diabetes_data.drop('Outcome',axis=1))
```

```
    StandardScaler()
```

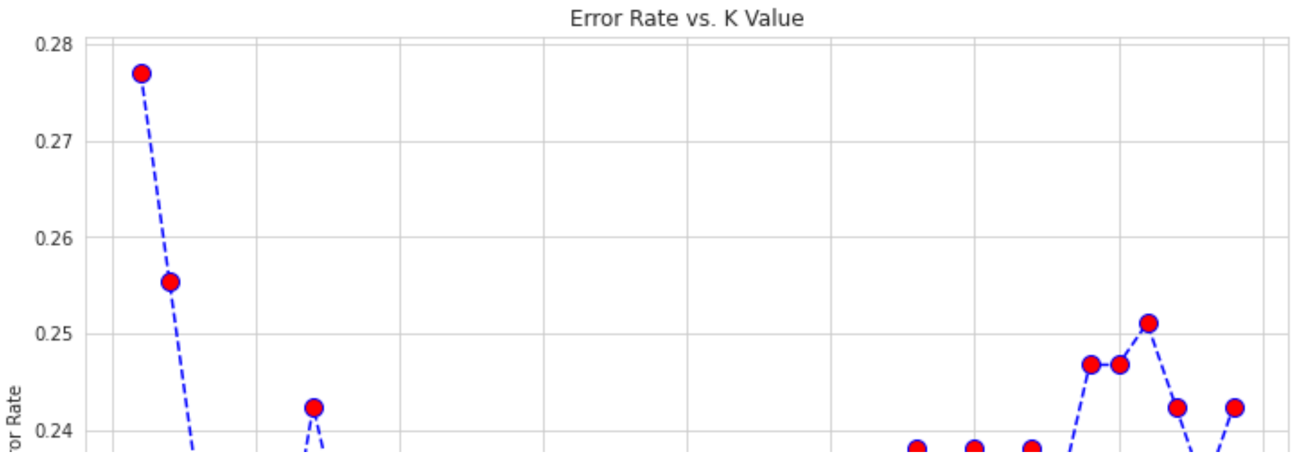
```
scaled_data = scaler.transform(diabetes_data.drop('Outcome',axis=1))
```

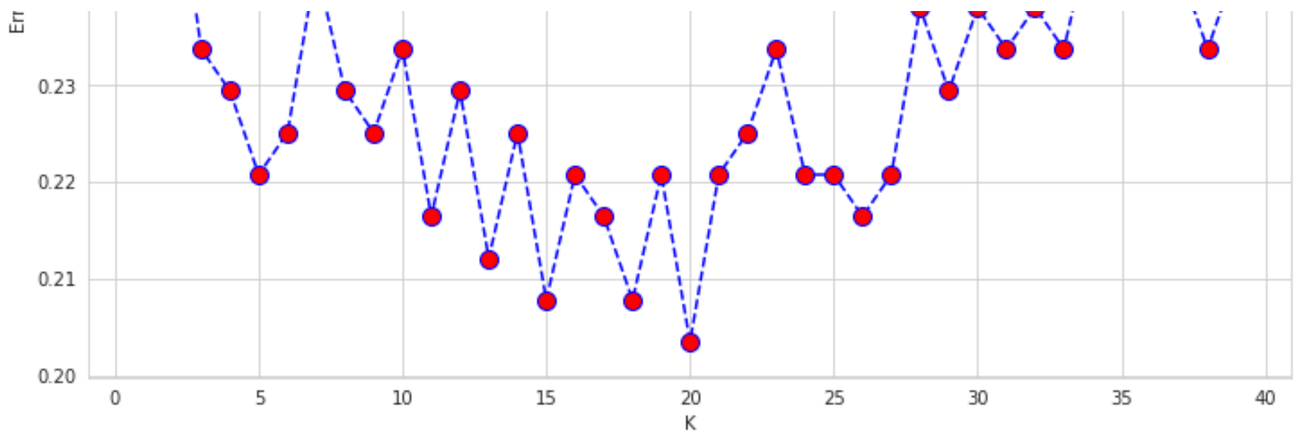
```
df_feat = pd.DataFrame(scaled_data,columns=diabetes_data.columns[:-1])
df_feat.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
0	0.639947	0.865108	-0.033518	0.670643	-0.181541	0.166619
1	-0.844885	-1.206162	-0.529859	-0.012301	-0.181541	-0.852200
2	1.233880	2.015813	-0.695306	-0.012301	-0.181541	-1.332500
3	-0.844885	-1.074652	-0.529859	-0.695245	-0.540642	-0.633881
4	-1.141852	0.503458	-2.680669	0.670643	0.316566	1.549303

```
scaled_test_scores = plot_KNN_error_rate(scaled_data,diabetes_data['Outcome'])
```

```
    Max train score 100.0 % and k = [1]

    Max test score 79.65367965367966 % and k = [20]
```

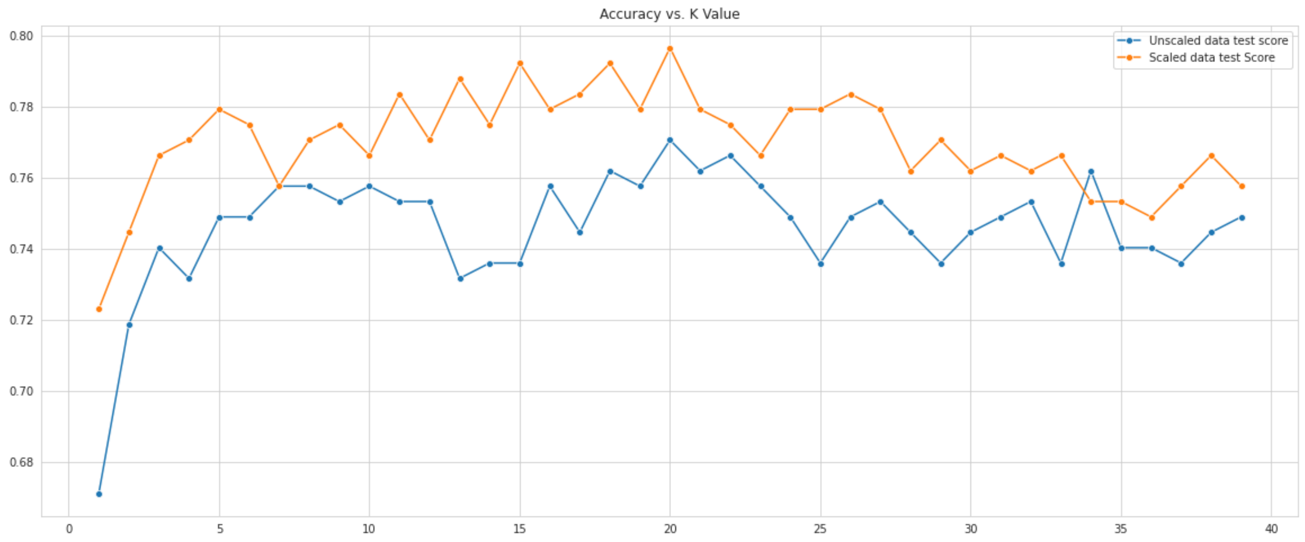




Comparing Accuracy before and after Standardization

```
plt.figure(figsize=(20,8))
plt.title('Accuracy vs. K Value')
sns.lineplot(range(1,40),unchanged_test_scores,marker='o',label='Unscaled data test
sns.lineplot(range(1,40),scaled_test_scores,marker='o',label='Scaled data test Scor
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b282c3a90>
```



```
##Referring to MinMax Scaler provided in scikit-learn.
## Use MinMax scaling on the dataset, and see the performance of KNN on this minma
from numpy import asarray
from sklearn.preprocessing import MinMaxScaler

comp_data = asarray([[100, 0.001],
        [8, 0.05],
        [50, 0.005],
        [88, 0.07],
        [4, 0.1]])
print(comp_data, '\n \n')
scaler = MinMaxScaler()
scaled = scaler.fit_transform(diabetes_data)
print(scaled)
```

```
[[1.0e+02 1.0e-03]
 [8.0e+00 5.0e-02]
 [5.0e+01 5.0e-03]
 [8.8e+01 7.0e-02]
 [4.0e+00 1.0e-01]]


[[0.35294118 0.67096774 0.48979592 ... 0.23441503 0.48333333 1.        ]
 [0.05882353 0.26451613 0.42857143 ... 0.11656704 0.16666667 0.        ]
 [0.47058824 0.89677419 0.40816327 ... 0.25362938 0.18333333 1.        ]
 ...
 [0.29411765 0.49677419 0.48979592 ... 0.07130658 0.15       0.        ]
 [0.05882353 0.52903226 0.36734694 ... 0.11571307 0.43333333 1.        ]
 [0.05882353 0.31612903 0.46938776 ... 0.10119556 0.03333333 0.        ]]
```

```
##Using K-Fold cross validation on all the above classification experiments and pre

array = diabetes_data.values

X = array[:,0:8]
Y = array[:,8]
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold

num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed, shuffle=True)
model = LogisticRegression()
results = cross_val_score(model, X, Y, cv=kfold)
print(("Accuracy: %.3f%% (%.3f%%)") % (results.mean()*100.0, results.std()*100.0))
```

```
Accuracy: 76.435% (5.033%)
```

Conclusion

From the data analysis we carried out, it seems that there is some form of an association between BMI, number of pregnancies, pedigree function, and the test results for diabetes.

As for the classification tasks, the standardized data yields much better results than the unscaled data over most of the K-values considered, thus indicating the importance of standardizing data in Machine Learning problems.