Diabetes Prediction using Machine Learning

Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Importing dataset

In [2]:
```
dataset = pd.read_csv('diabetes.csv')
dataset
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Ag
0	6	148	72	35	0	33.6	0.627	5
1	1	85	66	29	0	26.6	0.351	3
2	8	183	64	0	0	23.3	0.672	3
3	1	89	66	23	94	28.1	0.167	2
4	0	137	40	35	168	43.1	2.288	3
...	...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.171	6
764	2	122	70	27	0	36.8	0.340	2
765	5	121	72	23	112	26.2	0.245	3
766	1	126	60	0	0	30.1	0.349	4
767	1	93	70	31	0	30.4	0.315	2

768 rows × 9 columns

Replacing the zero values with mean or median values of respective features

In [3]:
```
# Glucose
dataset["Glucose"] = dataset["Glucose"].replace(0, dataset["Glucose"].median())
# BloodPressure
dataset["BloodPressure"] = dataset["BloodPressure"].replace(0, dataset["BloodPressure"
# BMI
dataset["BMI"] = dataset["BMI"].replace(0, dataset["BMI"].mean())
```

```
# SkinThickness
dataset["SkinThickness"] = dataset["SkinThickness"].replace(0, dataset["SkinThickness"
# Insulin
dataset["Insulin"] = dataset["Insulin"].replace(0, dataset["Insulin"].mean())
```

In [4]:
```
dataset
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35.000000	79.799479	33.6	0.627
1	1	85	66	29.000000	79.799479	26.6	0.351
2	8	183	64	20.536458	79.799479	23.3	0.672
3	1	89	66	23.000000	94.000000	28.1	0.167
4	0	137	40	35.000000	168.000000	43.1	2.288
...	...	...	...	...	...	...	...
763	10	101	76	48.000000	180.000000	32.9	0.171
764	2	122	70	27.000000	79.799479	36.8	0.340
765	5	121	72	23.000000	112.000000	26.2	0.245
766	1	126	60	20.536458	79.799479	30.1	0.349
767	1	93	70	31.000000	79.799479	30.4	0.315

768 rows × 9 columns

Information about dataset

In [5]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    float64
 4   Insulin                   768 non-null    float64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(4), int64(5)
memory usage: 54.1 KB
```

In [6]:
```
dataset.isnull().sum()
```

Out[6]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Statistical Analysis of Data

In [7]:
```
dataset.describe()
```

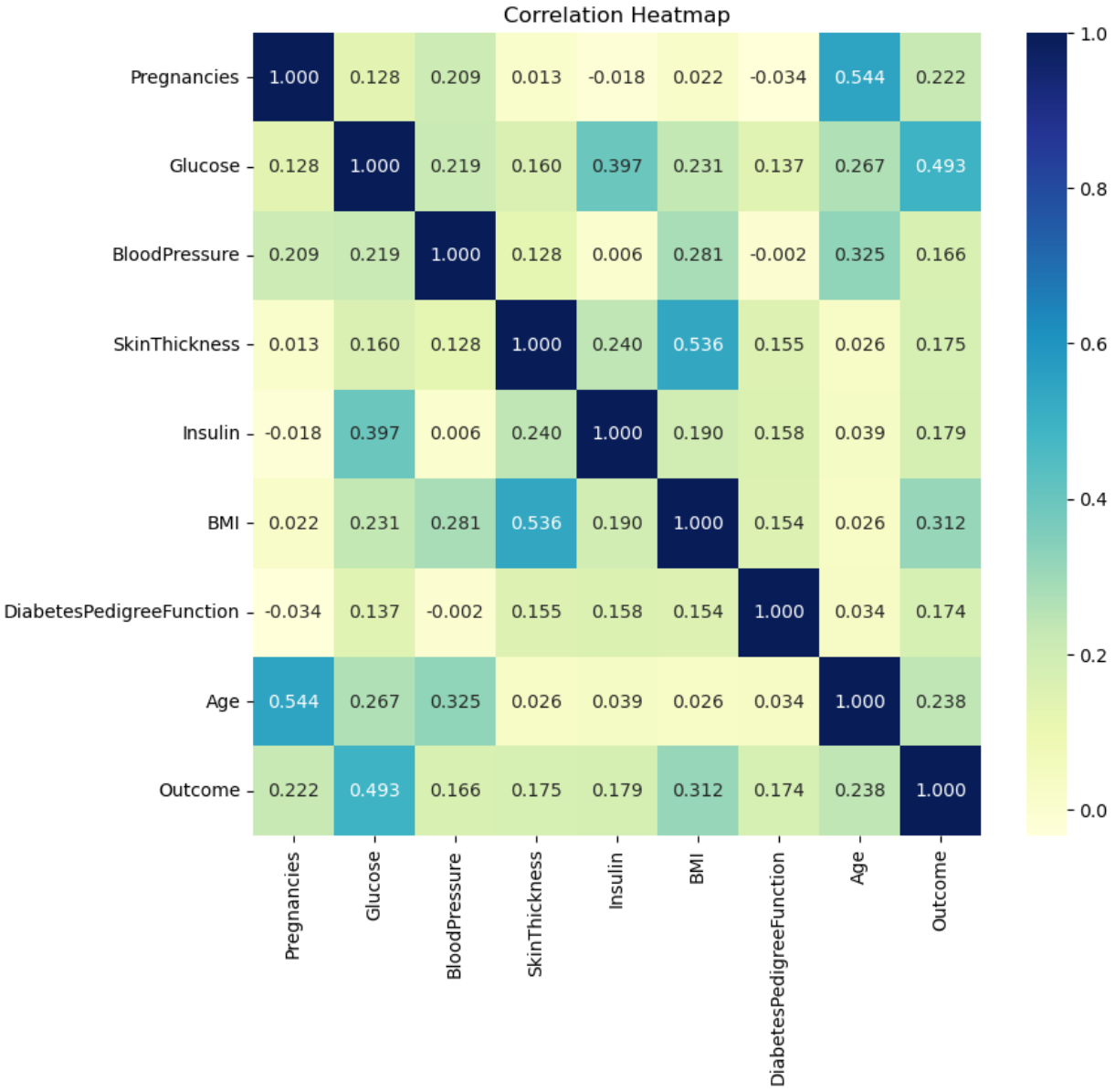
Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigr
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	121.656250	72.386719	26.606479	118.660163	32.450805	
std	3.369578	30.438286	12.096642	9.631241	93.080358	6.875374	
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	
25%	1.000000	99.750000	64.000000	20.536458	79.799479	27.500000	
50%	3.000000	117.000000	72.000000	23.000000	79.799479	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

Correlation plot of independent variables

In [8]:
```
plt.figure(figsize = (9, 8))
sns.heatmap(dataset.corr(), annot = True, fmt = ".3f", cmap = "YlGnBu")
plt.title("Correlation Heatmap")
```

Out[8]:
```
Text(0.5, 1.0, 'Correlation Heatmap')
```



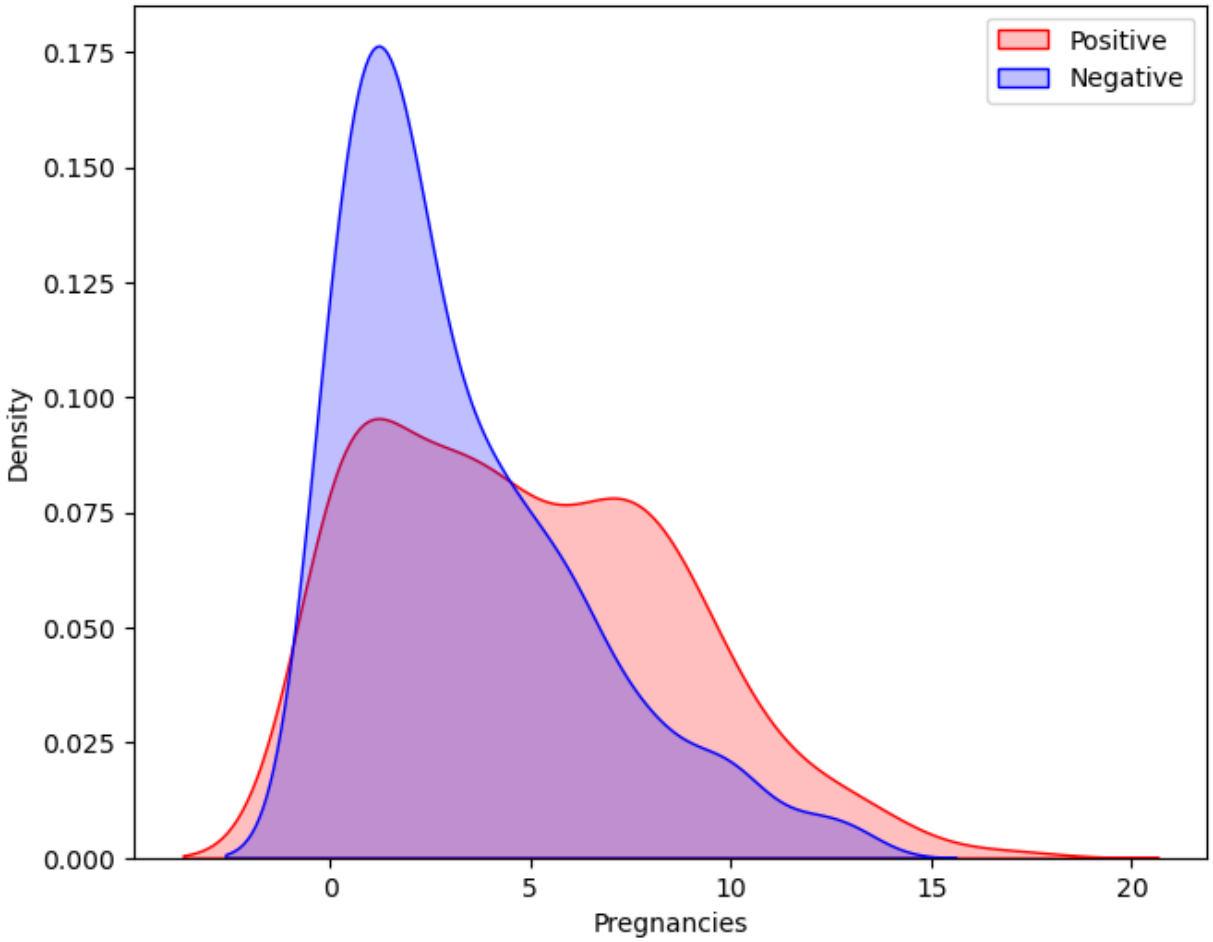
Performing EDA(Exploratory Data Analysis)

Explore Pregnancies and target variables

Plotting Density function graph of the pregnancies and target variables

In [9]:
```
plt.figure(figsize = (7.5, 6))
kde = sns.kdeplot(dataset["Pregnancies"][dataset["Outcome"]==1], color = 'Red', fill =
kde = sns.kdeplot(dataset["Pregnancies"][dataset["Outcome"]==0], color = 'Blue', fill
kde.set_xlabel("Pregnancies")
kde.set_ylabel("Density")
kde.legend(["Positive", "Negative"])
```

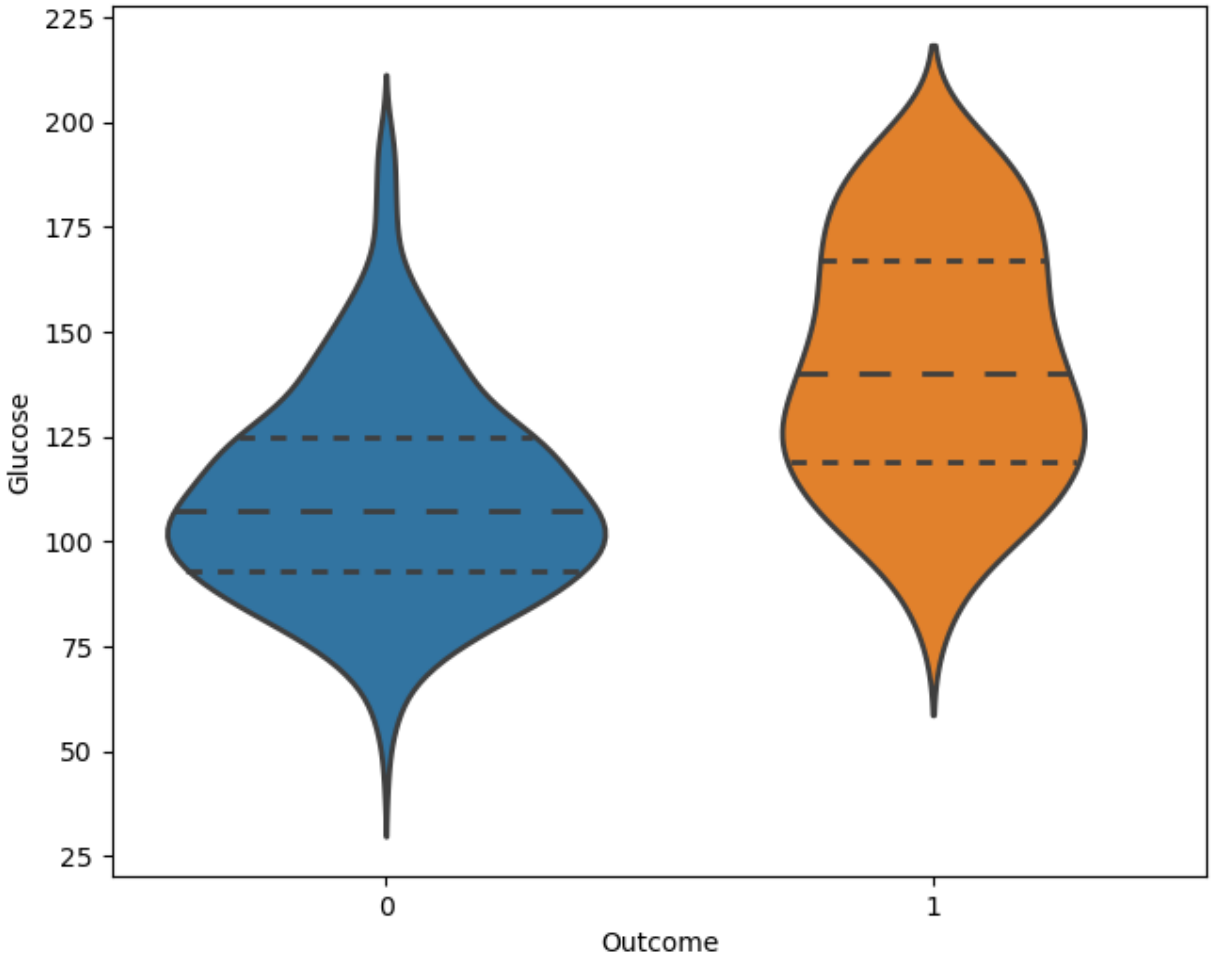
Out[9]:
```
<matplotlib.legend.Legend at 0x250ae7df050>
```



Explore Glucose and target variables

In [10]:
```
plt.figure(figsize = (7.5, 6))
sns.violinplot(data = dataset, x = "Outcome", y = "Glucose", split = True, linewidth =
```

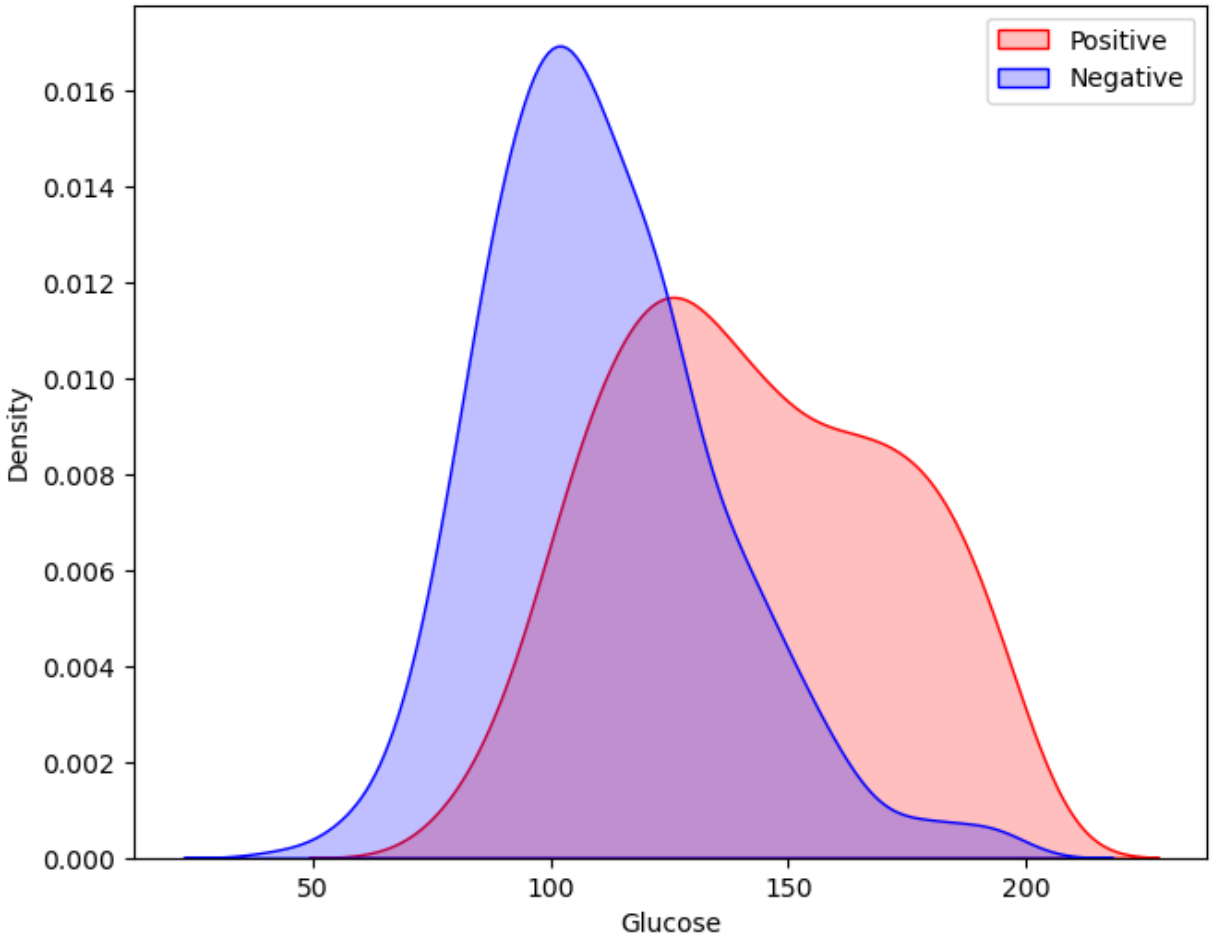
Out[10]:
```
<Axes: xlabel='Outcome', ylabel='Glucose'>
```



Plotting Density function graph of the Glucose and target variables

In [11]:
```
plt.figure(figsize = (7.5, 6))
kde = sns.kdeplot(dataset["Glucose"][dataset["Outcome"]==1], color = 'Red', fill = Tru
kde = sns.kdeplot(dataset["Glucose"][dataset["Outcome"]==0], color = 'Blue', fill = Tr
kde.set_xlabel("Glucose")
kde.set_ylabel("Density")
kde.legend(["Positive", "Negative"])
```

Out[11]:
```
<matplotlib.legend.Legend at 0x250ad860890>
```



Splitting the dependant and independant variables

In [12]:
```
X = dataset.drop(["Outcome"], axis = 1)
y = dataset["Outcome"]
```

In [13]:
```
X
```

Out[13]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35.000000	79.799479	33.6	0.627
1	1	85	66	29.000000	79.799479	26.6	0.351
2	8	183	64	20.536458	79.799479	23.3	0.672
3	1	89	66	23.000000	94.000000	28.1	0.167
4	0	137	40	35.000000	168.000000	43.1	2.288
...	...	...	...	...	...	...	...
763	10	101	76	48.000000	180.000000	32.9	0.171
764	2	122	70	27.000000	79.799479	36.8	0.340
765	5	121	72	23.000000	112.000000	26.2	0.245
766	1	126	60	20.536458	79.799479	30.1	0.349
767	1	93	70	31.000000	79.799479	30.4	0.315

768 rows × 8 columns

In [14]:
```
y
```

Out[14]:
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Splitting the dataset into training and testing datasets

In [15]:
```
from sklearn.model_selection import train_test_split
```

In [16]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_sta
```

In [17]:
```
X_train
```

Out[17]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
464	10	115	98	20.536458	79.799479	24.0	1.022
223	7	142	60	33.000000	190.000000	28.8	0.687
393	4	116	72	12.000000	87.000000	22.1	0.463
766	1	126	60	20.536458	79.799479	30.1	0.349
570	3	78	70	20.536458	79.799479	32.5	0.270
...	...	...	...	...	...	...	...
71	5	139	64	35.000000	140.000000	28.6	0.411
106	1	96	122	20.536458	79.799479	22.4	0.207
270	10	101	86	37.000000	79.799479	45.6	1.136
435	0	141	72	20.536458	79.799479	42.4	0.205
102	0	125	96	20.536458	79.799479	22.5	0.262

514 rows × 8 columns

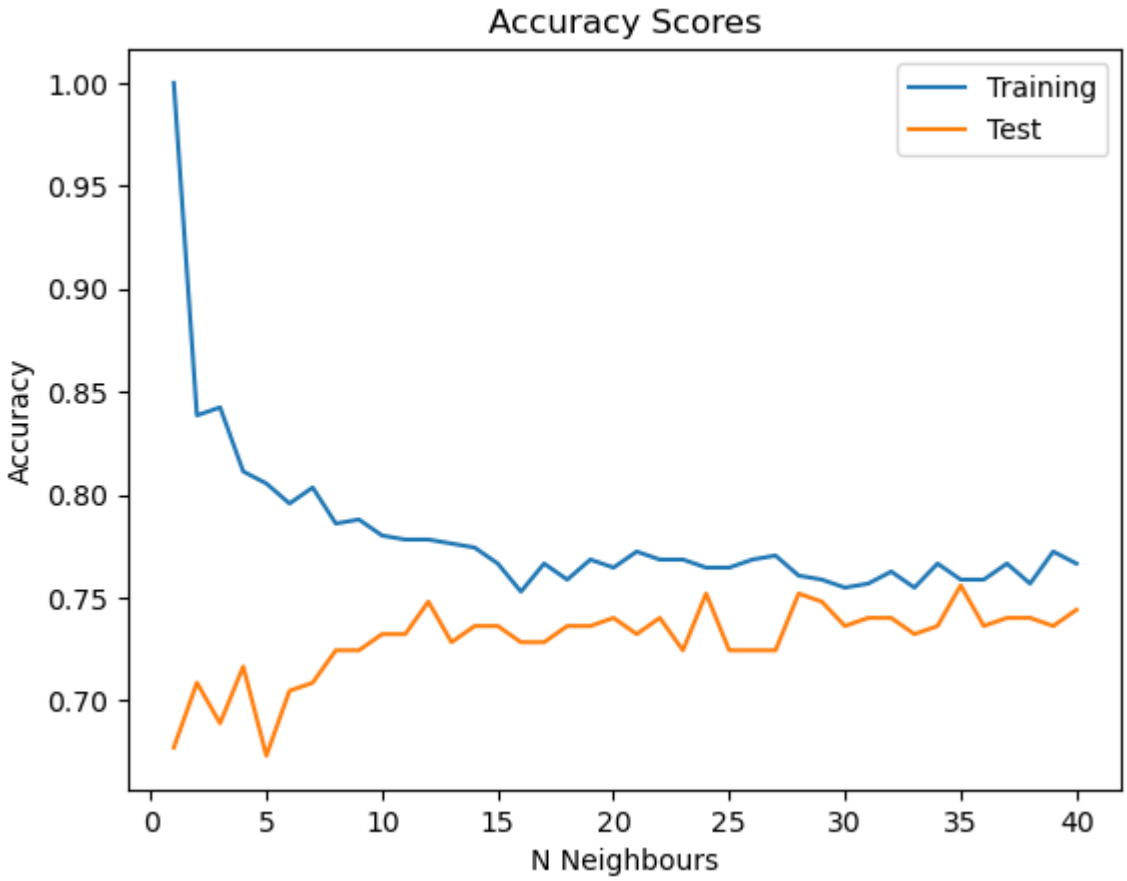
KNN Model

In [18]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [19]:
```
training_accuracy = []
test_accuracy = []
for n_neighbors in range(1,41):
    knn = KNeighborsClassifier(n_neighbors = n_neighbors)
    knn.fit(X_train, y_train)

    # Checking accuracy score
    training_accuracy.append(knn.score(X_train, y_train))
    test_accuracy.append(knn.score(X_test, y_test))
```

In [20]:
```
plt.plot(range(1,41), training_accuracy, label = "Training")
plt.plot(range(1,41), test_accuracy, label = "Test")
plt.title("Accuracy Scores")
plt.xlabel("N Neighbours")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Maximum Accuracy is when N Neighbors = 12

In [21]:
```
knn = KNeighborsClassifier(n_neighbors = 12)
knn.fit(X_train, y_train)
# Checking accuracy score
print(knn.score(X_train, y_train), ": Training Accuracy")
print(knn.score(X_test, y_test), ": Testing Accuracy")
```

```
0.7782101167315175 : Training Accuracy
0.7480314960629921 : Testing Accuracy
```

Decision Tree Model

In [22]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(random_state = 0)
dt.fit(X_train, y_train)
# Checking accuracy score
print(dt.score(X_train, y_train), ": Training Accuracy")
print(dt.score(X_test, y_test), ": Testing Accuracy")
```

```
1.0 : Training Accuracy
0.6811023622047244 : Testing Accuracy
```

In [23]:
```
from sklearn.tree import DecisionTreeClassifier
dt1 = DecisionTreeClassifier(random_state = 0, max_depth = 6)
dt1.fit(X_train, y_train)
# Checking accuracy score
print(dt1.score(X_train, y_train), ": Training Accuracy")
print(dt1.score(X_test, y_test), ": Testing Accuracy")
```

```
0.857976653696498 : Training Accuracy
0.7401574803149606 : Testing Accuracy
```

MLP Classifier

In [24]:
```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(random_state = 42)
mlp.fit(X_train, y_train)
# Checking accuracy score
print(mlp.score(X_train, y_train), ": Training Accuracy")
print(mlp.score(X_test, y_test), ": Testing Accuracy")
```

```
0.7509727626459144 : Training Accuracy
0.6811023622047244 : Testing Accuracy
```

MLP Classifiers with Standard Scaling

In [25]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.fit_transform(X_test)
```

In [26]:
```
mlp1 = MLPClassifier(random_state = 0)
mlp1.fit(X_train_scaled, y_train)
# Checking accuracy score
print(mlp1.score(X_train_scaled, y_train), ": Training Accuracy")
print(mlp1.score(X_test_scaled, y_test), ": Testing Accuracy")
```

```
0.8326848249027238 : Training Accuracy
0.7322834645669292 : Testing Accuracy
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
```

MLP Classifier Model with Standard Scaling gives the best accuracy.

Along with Decision Tree Model with Maximum depth as 6.