The necessary library modules required for data analysis , data visualization and machine learning are imported.

In [1]:
```
import pandas as pd   #library for dataframe
import numpy as np    #library for vectorized computation
import os             #library for interacting with our OS
import seaborn as sns  # library for data visualization
import matplotlib.pyplot as plt
from pylab import rcParams
os.chdir(r"C:\Users\Ram Kumar R S\Downloads\SoSe 2020") # changing the path
%matplotlib inline
# magic function to display the images in the notebook
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score , precision_score, recall_score, roc_auc_score, roc_curve,cl
assification_report
from sklearn.inspection import permutation_importance
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.utils import resample
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline

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

In [2]:
```
# function to read the input dataset file
def load_dataset(file_name):
    """Arg: file_name- the name of the dataset to be loaded as dataframe
       Return: Dataframe of the datast"""
    return pd.read_csv(file_name)
```

In [3]:
```
dataset=load_dataset("diabetes.csv")  # function call and the object is pointed to the variable called
 "dataset"
```

In [4]:
```
dataset.head()  # head method displays first 5 rows
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [5]:
```
dataset.columns  # different columns in the dataset
```

Out[5]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

Dataset Exploration and Visualization

In [6]:
```
dataset.info()  # method to display the features along with number of values in it.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
Pregnancies                 768 non-null int64
Glucose                     768 non-null int64
BloodPressure               768 non-null int64
SkinThickness               768 non-null int64
Insulin                     768 non-null int64
BMI                         768 non-null float64
DiabetesPedigreeFunction    768 non-null float64
Age                         768 non-null int64
Outcome                     768 non-null int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [7]:
```
dataset.dtypes  # method to display only the data type of all the values present in the dataset
```

Out[7]:
```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

In [8]:
```
dataset.isna().any() # method to display for the presence of any null values, True means presence of nu
ll values,
                     # False means absence of null values
```

Out[8]:
```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

In [9]:
```
dataset.isna().sum()  # method to display the number of missing values if it is present
                      # 0 indicates no missing values
```

Out[9]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [10]:
```
dataset.describe()  # method to display the various numerical properties of the column features.
```

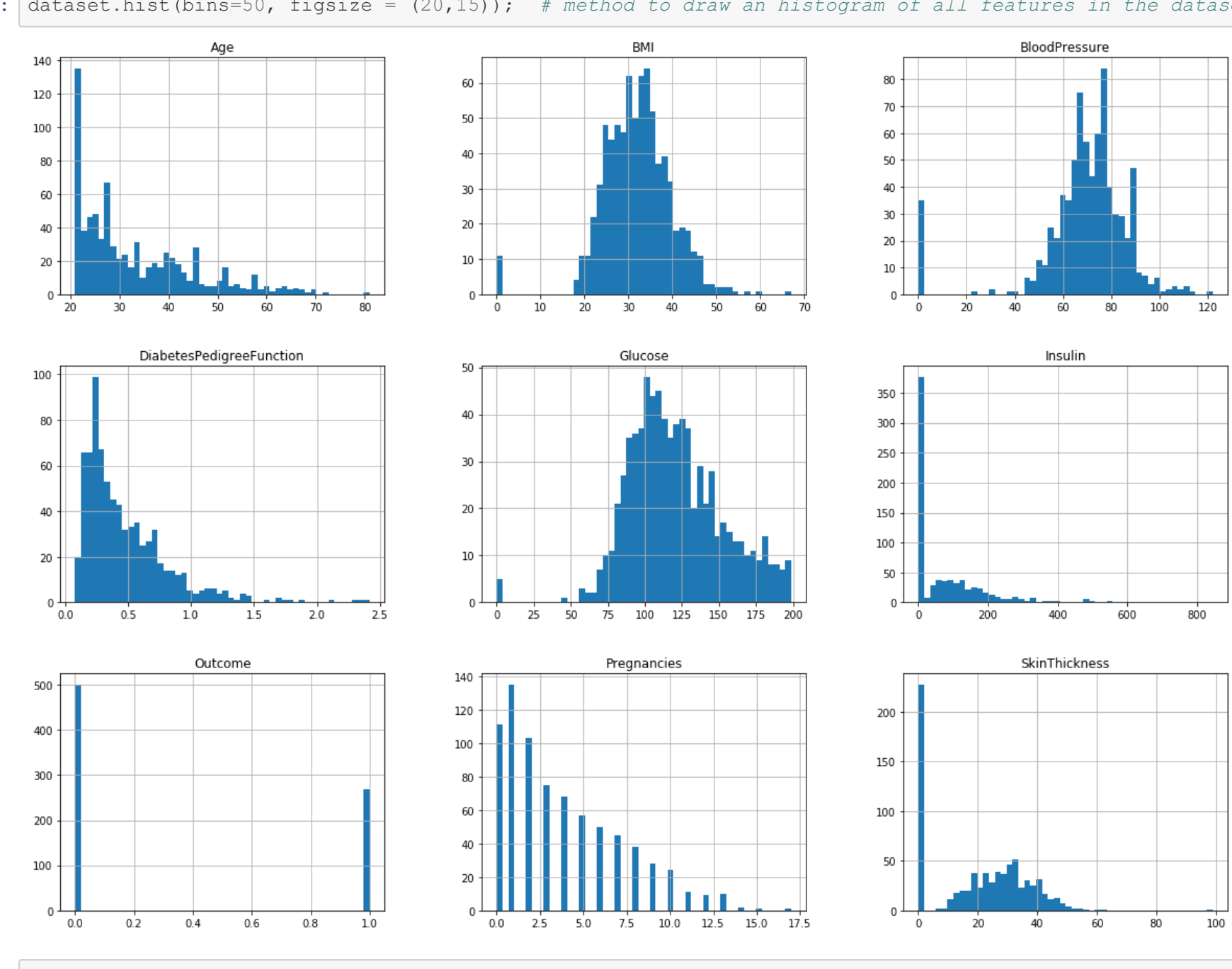
Out[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.00000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.34895
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.47695
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.00000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.00000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.00000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.00000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.00000

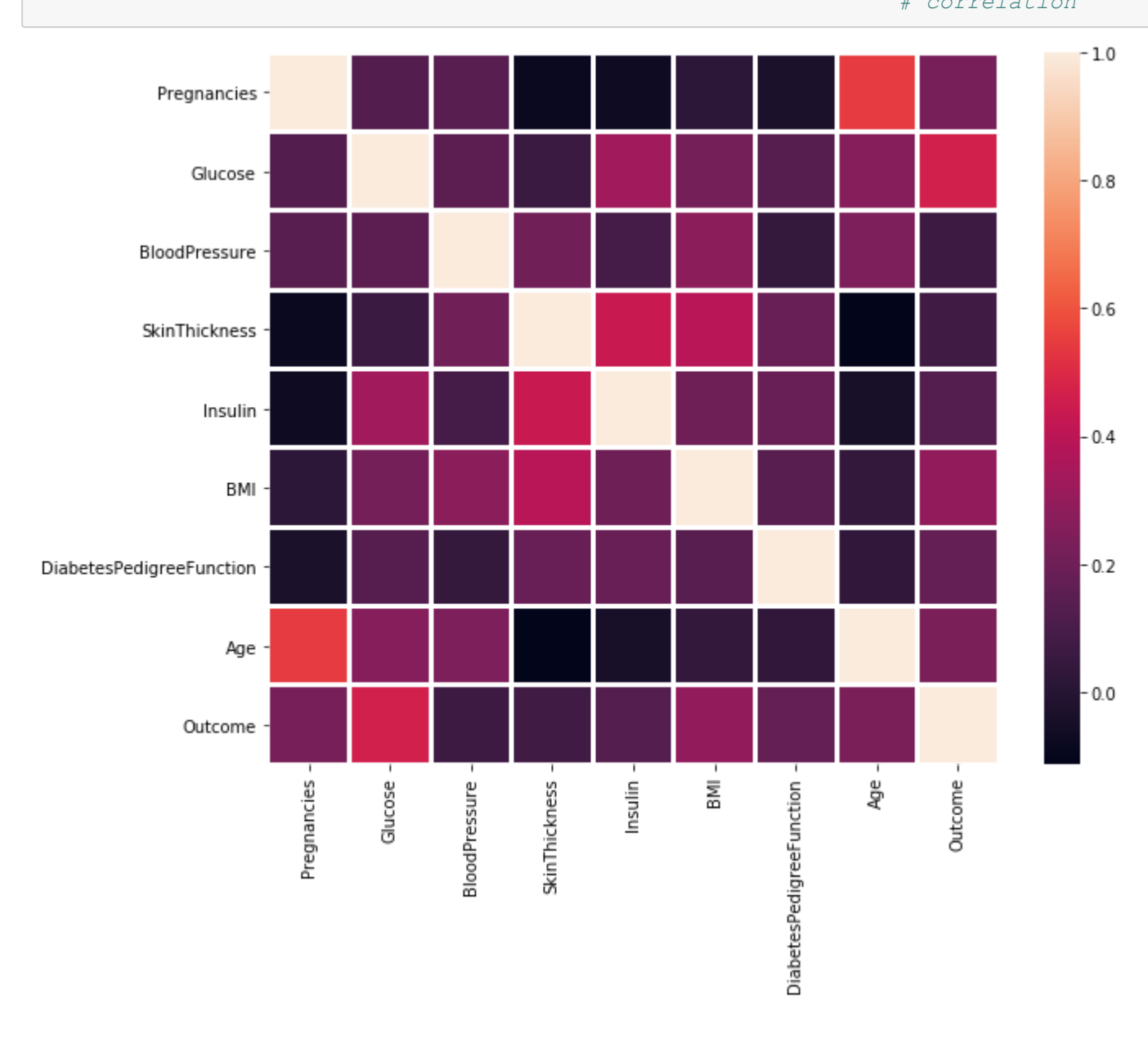
In [11]:
```
dataset.shape # shape of the dataset
```

Out[11]:
```
(768, 9)
```

In [12]:
```
dataset.hist(bins=50, figsize = (20,15));  # method to draw an histogram of all features in the dataset
```



In [13]:
```
plt.figure(figsize= (10, 8))
sns.heatmap(dataset.corr(),linecolor='white', linewidths=2);  # seaborn function to draw a heatmap base
d on feature
                                                               # correlation
```



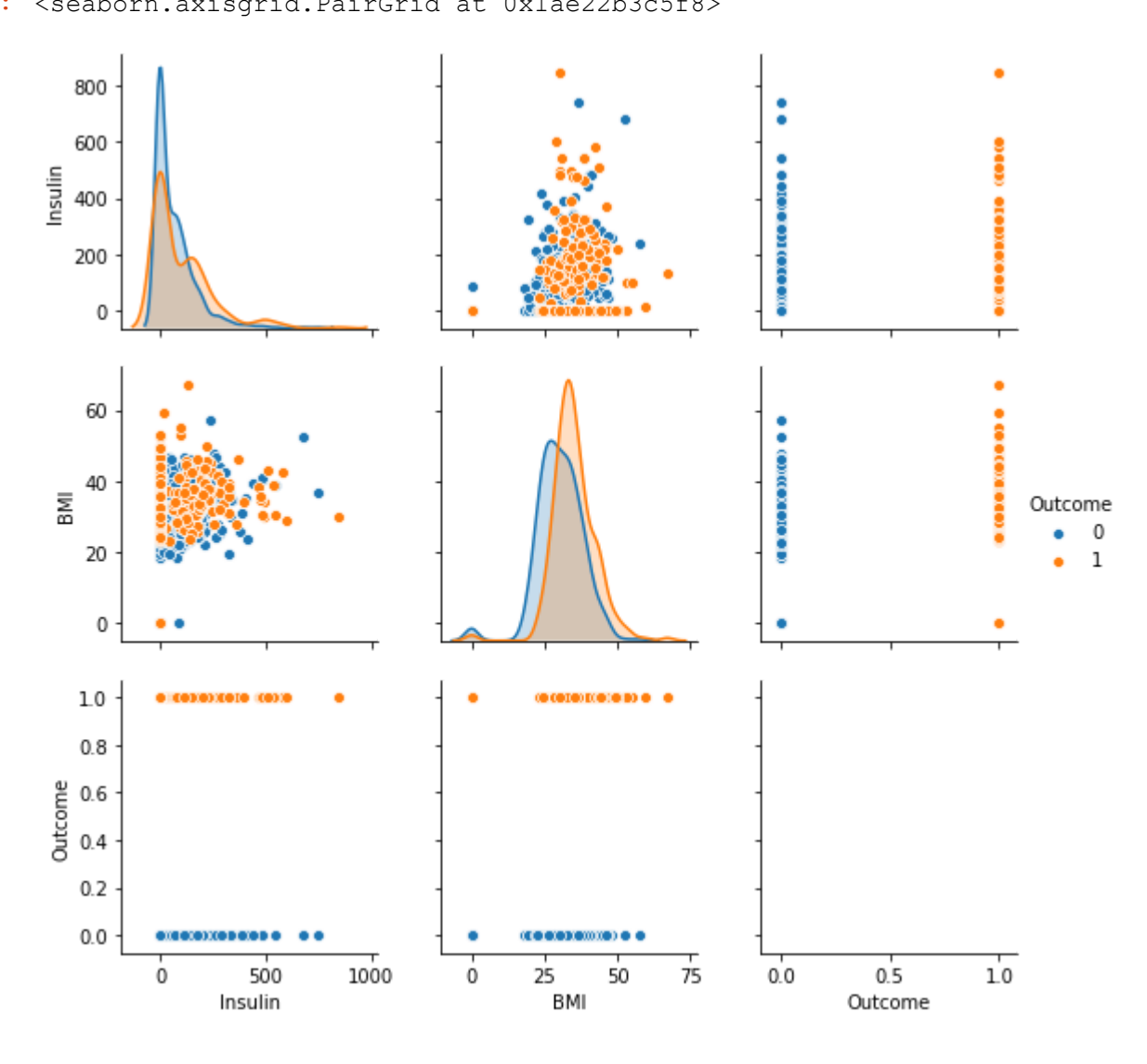
A pairs plot allows us to see both distribution of single variables and relationships between two variables.

In [14]:
```
sns.pairplot(dataset,hue='Outcome') # seaborn function to dislay all featues in pairplot
```

Out[14]:
```
<seaborn.axisgrid.PairGrid at 0x1ae229a9668>
```

In [15]:
```
sns.pairplot(dataset[['Insulin','BMI','Outcome']],hue='Outcome')
# only selected features
```

Out[15]:
```
<seaborn.axisgrid.PairGrid at 0x1ae22b3c5f8>
```



kNN Algorithm - Machine Learning

The Machine learning part consisting of segregating the input features into dependent variables and independent variables, training the model and testing.

In [16]:
```
# Create features and labels
features = dataset.drop(['Outcome'], axis=1)     # indepenedent variable
labels = dataset['Outcome']                       # dependent variable
```

In [17]:
```
# Create training and test set
features_train, features_test, labels_train, labels_test = \
train_test_split(features, labels, test_size=0.25,random_state=15)
```

In [18]:
```
# Initialize the algorithm
classifier = KNeighborsClassifier(n_neighbors=5)
```

In [19]:
```
# Fit data
classifier.fit(features_train, labels_train)
```

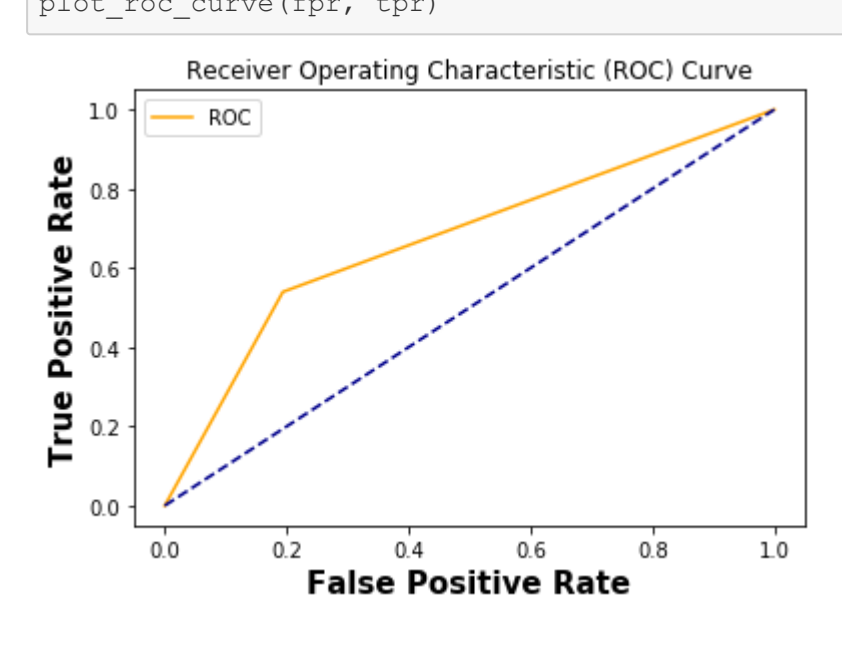
Out[19]:
```
KNeighborsClassifier()
```

In [20]:
```
#prediction
pred = classifier.predict(features_test)
```

In [21]:
```
accuracy = accuracy_score(labels_test, pred)  # accuracy
print('Accuracy: {:.2f}'.format(accuracy))

precision = precision_score(labels_test, pred) # precision
print ('Precision: {:.2f}'.format(precision))


recall = recall_score(labels_test, pred)  # recall
print ('Recall: {:.2f}'.format(recall))


auroc = roc_auc_score(labels_test, pred)   # area under curve
print ('AUROC score: {:.2f}'.format(auroc))
```

```
Accuracy: 0.72
Precision: 0.58
Recall: 0.54
AUROC score: 0.67
```

In [22]:
```
fpr, tpr, thresholds = roc_curve(labels_test, pred)
# Defining a function to plot the AUROC-curve
def plot_roc_curve(fpr, tpr):
    plt.plot(fpr, tpr, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate',fontsize=15,fontweight='bold')
    plt.ylabel('True Positive Rate',fontsize=15,fontweight='bold')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
# Calling the function to finally plot the curve
plot_roc_curve(fpr, tpr)
```



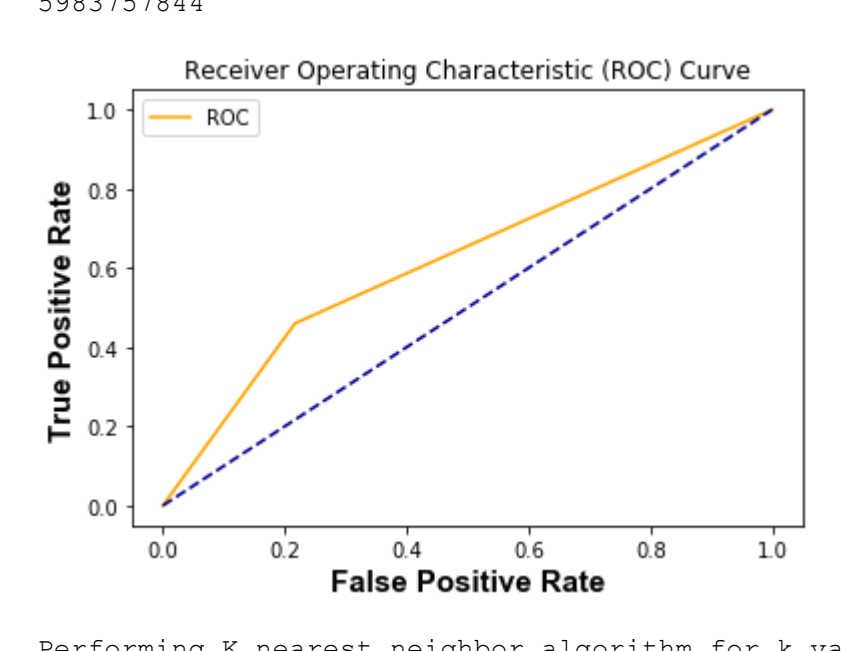
Elegant way of performing kNN algorithm using a function to collect all necessary functions.

function to perform kNN algorithm - Input : 1 k value

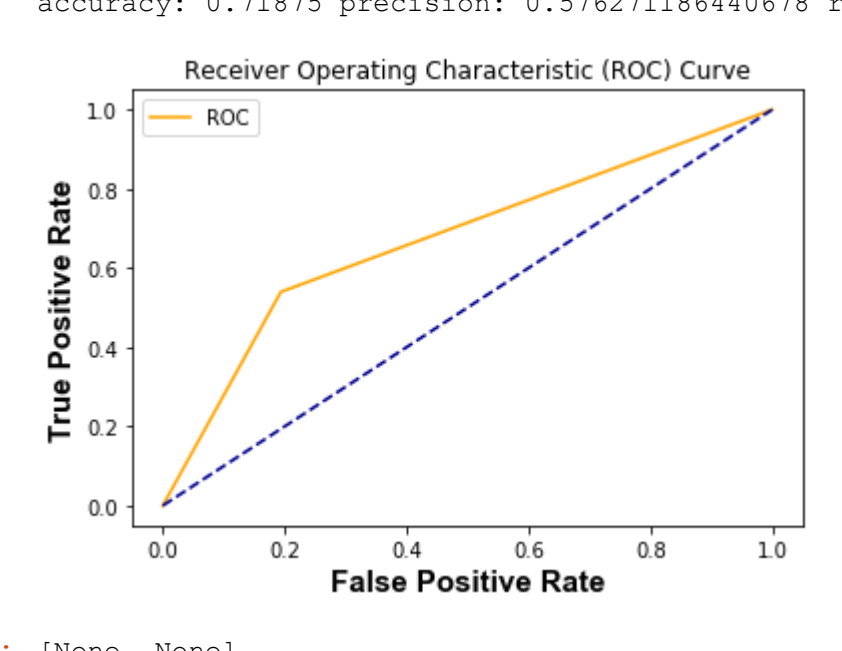
In [23]:
```
def performing_kNN(k_value): # for one k value
    """function to perform kNN algorithm for one input k value at a time
    Arg- k_value: k value to be used for the algorithm
    Returns: Different performance metrics with the AUC plot
    """
    print("Performing K nearest neighbor algorithm for k value :{}".format(k_value))
    classifier = KNeighborsClassifier(n_neighbors=k_value)
    classifier.fit(features_train, labels_train)
    pred = classifier.predict(features_test)
    accuracy = accuracy_score(labels_test, pred)
    precision = precision_score(labels_test, pred)
    recall = recall_score(labels_test, pred)
    auroc = roc_auc_score(labels_test, pred)
    print("accuracy:",accuracy,
          "precision:",precision,
           "recall:",recall,
         "auroc:",auroc)


    fpr, tpr, thresholds = roc_curve(labels_test, pred)
    plt.plot(fpr, tpr, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate',fontsize=15,fontweight='bold',family='Arial')
    plt.ylabel('True Positive Rate',fontsize=15,fontweight='bold',family='Arial')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()

    #return accuracy,precision,recall,auroc
```

In [24]:
```
[performing_kNN(k) for k in [3,5]]
```

```
Performing K nearest neighbor algorithm for k value :3
accuracy: 0.6770833333333334 precision: 0.5087719298245614 recall: 0.4603174603174603 auroc: 0.621631
5983757844
```




```
Performing K nearest neighbor algorithm for k value :5
accuracy: 0.71875 precision: 0.576271186440678 recall: 0.5396825396825397 auroc: 0.6729420450350682
```



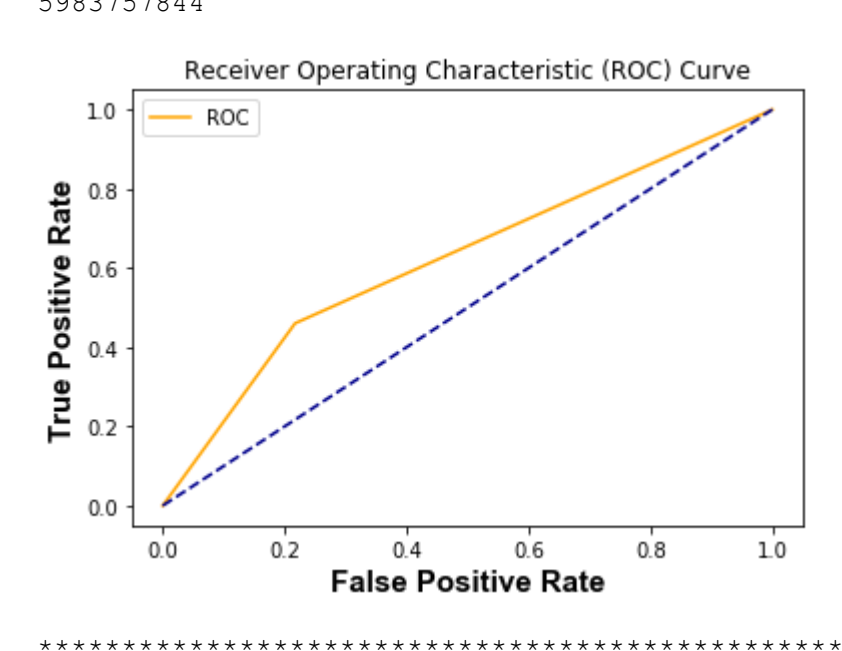
Out[24]:
```
[None, None]
```

function to perform kNN algorithm - Input : list of k values in a list

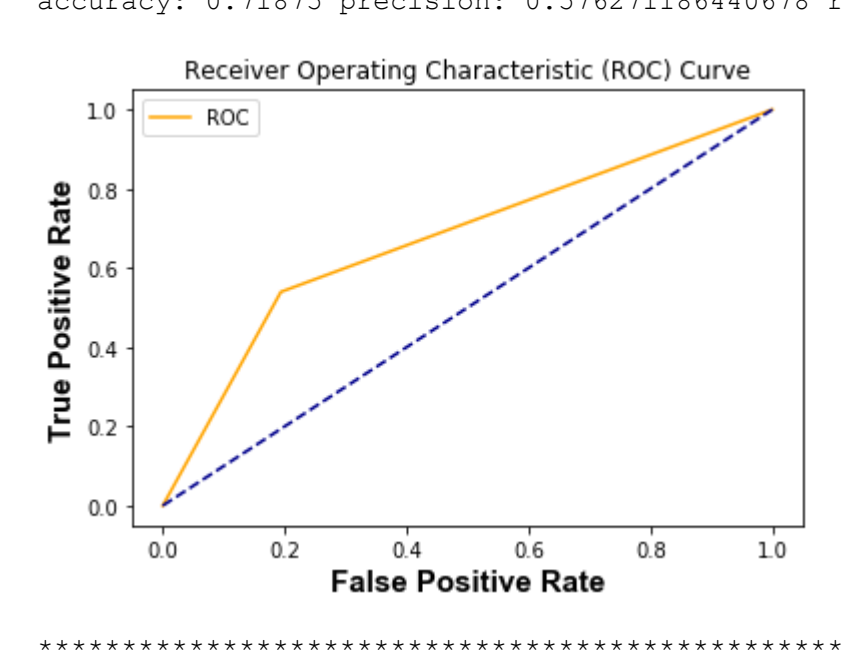
In [25]:
```
performance_metric_dict={}   # dict to hold the values of performance metrics
def performing_kNN_list(k_value_list):  # for a list of k values
    """function to perform kNN algorithm for a list of k as input values
    Arg- k_value_list: list with k values to be used for the algorithm
    Returns: Different performance metrics with the AUC plot
    """

    for k_value in k_value_list:
        print("Performing K nearest neighbor algorithm for k value :{}".format(k_value))
        classifier = KNeighborsClassifier(n_neighbors=k_value) # algorithm initialization
        classifier.fit(features_train, labels_train) # model training
        pred = classifier.predict(features_test) # model testing
        accuracy = accuracy_score(labels_test, pred) # accuracy
        precision = precision_score(labels_test, pred) # precision
        recall = recall_score(labels_test, pred)  # recall
        auroc = roc_auc_score(labels_test, pred)    # auroc
        print("accuracy:",accuracy,
              "precision:",precision,
               "recall:",recall,
             "auroc:",auroc)

        # dict filling step
        performance_metric_dict[k_value]=[accuracy,precision,recall,auroc]
        #key: k_value, value: its different performance metrics in a list format

        # plotting function
        fpr, tpr, thresholds = roc_curve(labels_test, pred)
        plt.plot(fpr, tpr, color='orange', label='ROC')
        plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
        plt.xlabel('False Positive Rate',fontsize=15,fontweight='bold',family='Arial')
        plt.ylabel('True Positive Rate',fontsize=15,fontweight='bold',family='Arial')
        plt.title('Receiver Operating Characteristic (ROC) Curve')
        plt.legend()
        plt.show()
        print("*"*100)

    #return accuracy,precision,recall,auroc
```

In [26]:
```
performing_kNN_list([3,5,7,9,15,40])
```

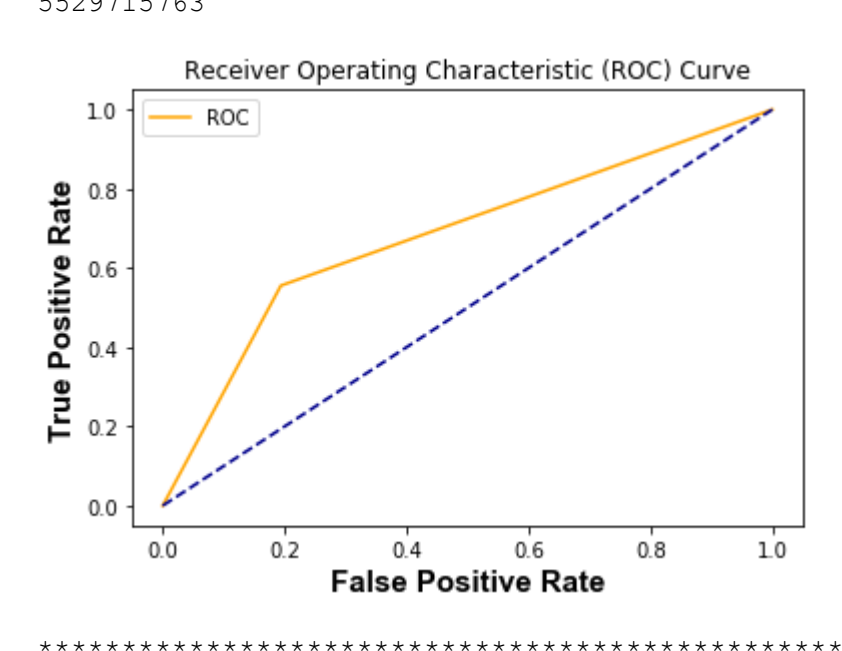
```
Performing K nearest neighbor algorithm for k value :3
accuracy: 0.6770833333333334 precision: 0.5087719298245614 recall: 0.4603174603174603 auroc: 0.621631
5983757844
```



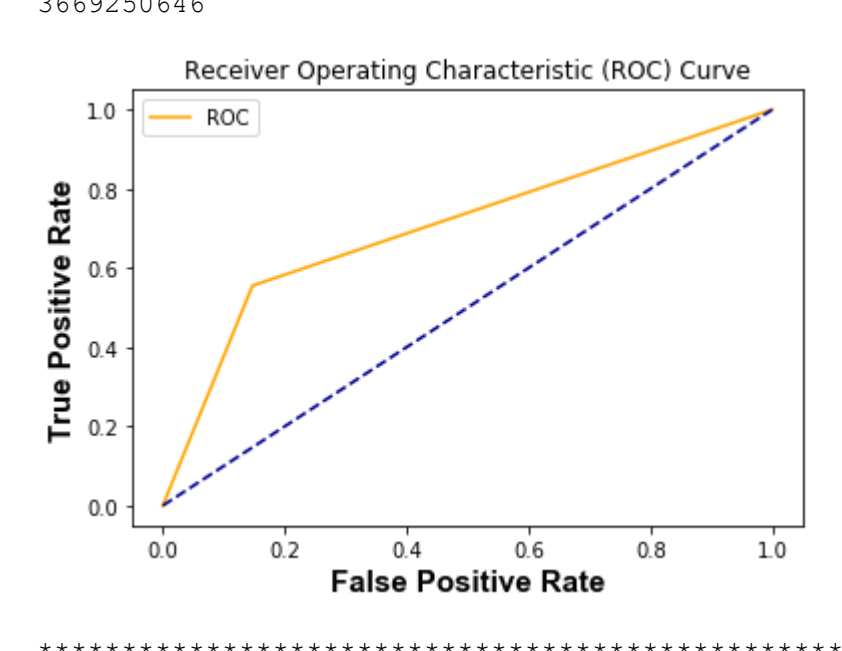

```
****************************************************************************************************
Performing K nearest neighbor algorithm for k value :5
accuracy: 0.71875 precision: 0.576271186440678 recall: 0.5396825396825397 auroc: 0.6729420450350682
```



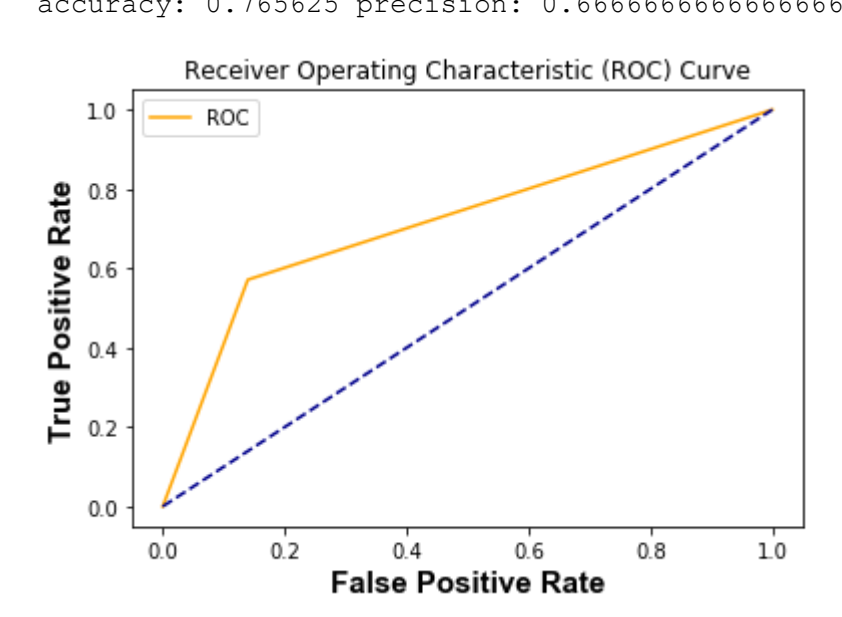

```
****************************************************************************************************
Performing K nearest neighbor algorithm for k value :7
accuracy: 0.7239583333333334 precision: 0.5833333333333334 recall: 0.5555555555555556 auroc: 0.680878
5529715763
```



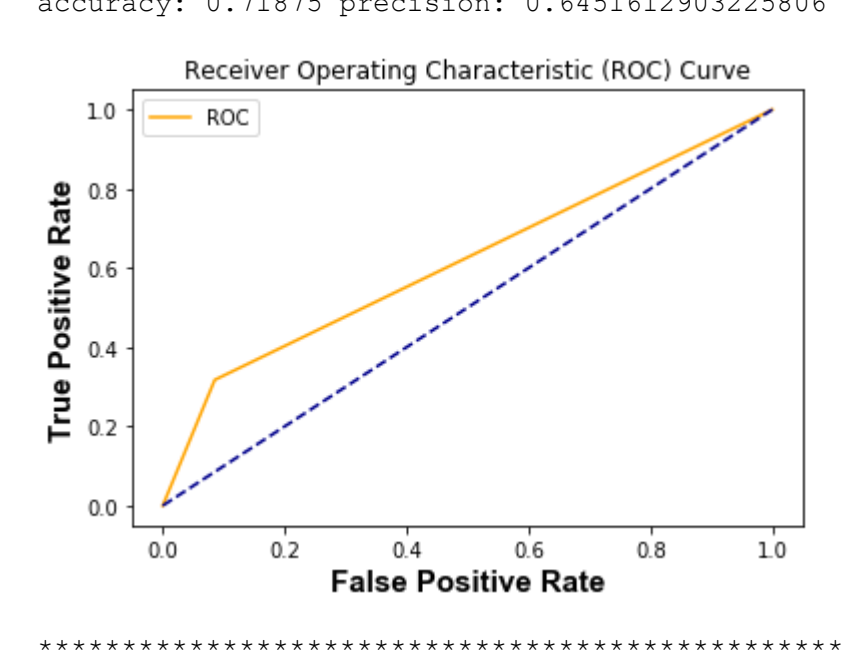

```
****************************************************************************************************
Performing K nearest neighbor algorithm for k value :9
accuracy: 0.7552083333333334 precision: 0.6481481481481481 recall: 0.5555555555555556 auroc: 0.704134
3669250646
```




```
****************************************************************************************************
Performing K nearest neighbor algorithm for k value :15
accuracy: 0.765625 precision: 0.6666666666666666 recall: 0.5714285714285714 auroc: 0.7159468438538206
```




```
****************************************************************************************************
Performing K nearest neighbor algorithm for k value :40
accuracy: 0.71875 precision: 0.6451612903225806 recall: 0.31746031746031744 auroc: 0.61609449981543
```




```
****************************************************************************************************
```

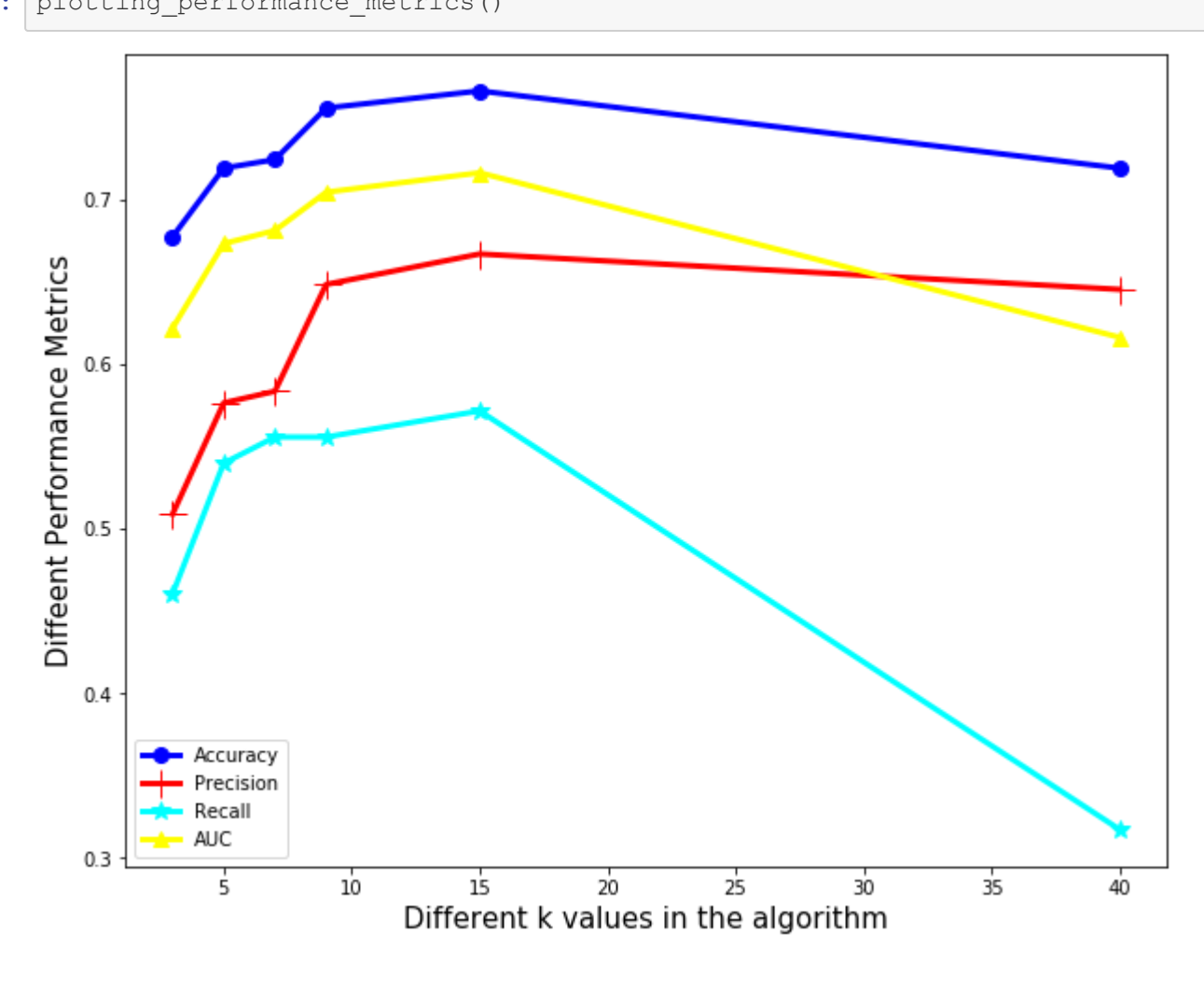
The `performance_metric_dict` holds the k values as keys and different evaluation metrics enclosed in a list as their values.

In [27]:
```
performance_metric_dict
```

Out[27]:
```
{3: [0.6770833333333334,
  0.5087719298245614,
  0.4603174603174603,
  0.6216315983757844],
 5: [0.71875, 0.576271186440678, 0.5396825396825397, 0.6729420450350682],
 7: [0.7239583333333334,
  0.5833333333333334,
  0.5555555555555556,
  0.6808785529715763],
 9: [0.7552083333333334,
  0.6481481481481481,
  0.5555555555555556,
  0.7041343669250646],
 15: [0.765625, 0.6666666666666666, 0.5714285714285714, 0.7159468438538206],
 40: [0.71875, 0.6451612903225806, 0.31746031746031744, 0.61609449981543]}
```

In [28]:
```
# plotting different performance metrics in a line plot
def plotting_performance_metrics():

    plt.figure(figsize= (10, 8))
    plt.plot(list(performance_metric_dict.keys()),[i[0] for i in performance_metric_dict.values()],"bo-
",color='blue',\
             linewidth=3,markersize=8,label='Accuracy')

    plt.plot(list(performance_metric_dict.keys()),[i[1] for i in performance_metric_dict.values()],"+-"
,color='red',\
             linewidth=3,markersize=15,label='Precision')

    plt.plot(list(performance_metric_dict.keys()),[i[2] for i in performance_metric_dict.values()],"*-"
,color='cyan',\
             linewidth=3,markersize=10,label='Recall')

    plt.plot(list(performance_metric_dict.keys()),[i[3] for i in performance_metric_dict.values()],"^-"
,color='yellow',\
             linewidth=3,markersize=8,label='AUC')
    plt.legend()
    plt.xlabel('Different k values in the algorithm',fontsize=15)

    plt.ylabel("Diffeent Performance Metrics",fontsize=15);
```

In [29]:
```
plotting_performance_metrics()
```



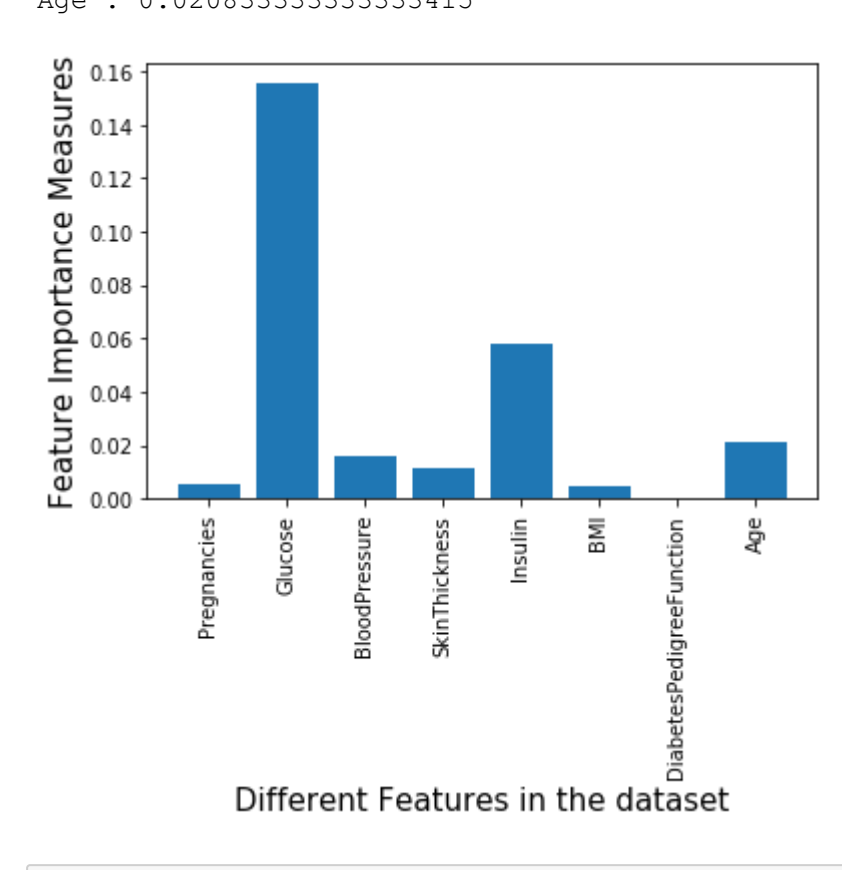
On observing this line plot, the accuracy, precision, recall, AUC was high in `k value = 15`. Therefore, the best k is 15.

In [30]:
```
#!pip install --upgrade scikit-learn
```

Feature Importance using kNN classifier

In [31]:
```
results = permutation_importance(classifier, features_train, labels_train, scoring='accuracy',random_st
ate=15)
# get importance
importance = results.importances_mean

# summarize feature importance
feature_importance_dict={}
print("Diffent features with their importances:")
print()
for i in range(len(importance)):
    print(features_train.columns[i],":",importance[i])
    feature_importance_dict[features_train.columns[i]]=importance[i]

# plot feature importance
plt.bar(features_train.columns,importance)
plt.xticks(rotation=90)
plt.xlabel("Different Features in the dataset",fontsize=15)
plt.ylabel("Feature Importance Measures",fontsize=15)
plt.show()


#https://machinelearningmastery.com/calculate-feature-importance-with-python/
#https://scikit-learn.org/stable/modules/generated/sklearn.inspection.permutation_importance.html
```

```
Diffent features with their importances:

Pregnancies : 0.005208333333333337
Glucose : 0.1555555555555556
BloodPressure : 0.015972222222222256
SkinThickness : 0.011458333333333337
Insulin : 0.05833333333333339
BMI : 0.004861111111111161
DiabetesPedigreeFunction : 0.0003472222222222321
Age : 0.020833333333333415
```



In [32]:
```
#{k: v for k, v in sorted(feature_importance_dict.items(), key=lambda item: item[1],reverse=True)}
print("The names of the features with decreasing strength of feature importance are as follows:")
[k for k, v in sorted(feature_importance_dict.items(), key=lambda item: item[1],reverse=True)]
```

```
The names of the features with decreasing strength of feature importance are as follows:
```

Out[32]:
```
['Glucose',
 'Insulin',
 'Age',
 'BloodPressure',
 'SkinThickness',
 'Pregnancies',
 'BMI',
 'DiabetesPedigreeFunction']
```

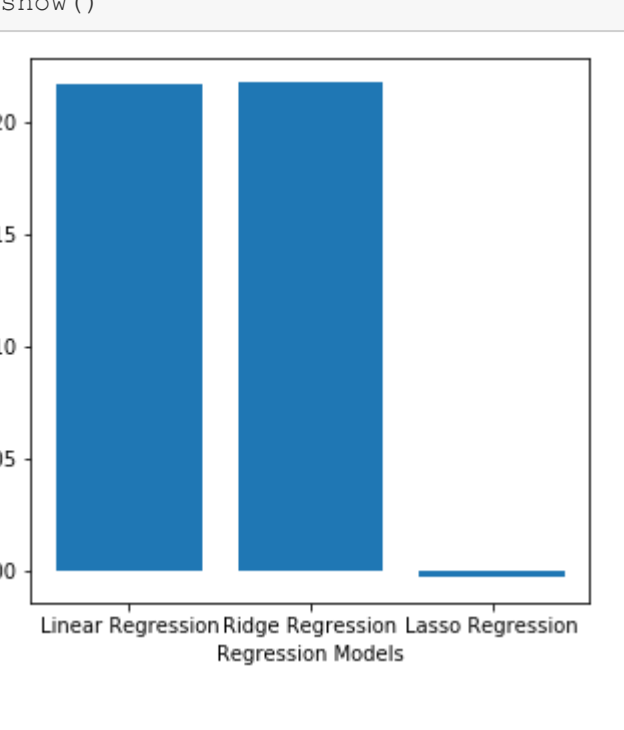
Scaling

Standardization

First, Standardization step was performed.

In [33]:
```
x_sc= StandardScaler()
features_train_scaled = x_sc.fit_transform(features_train)
features_test_scaled = x_sc.transform(features_test)
```

```
In [80]: plt.bar(models, scores)
plt.xlabel('Regression Models')
plt.ylabel('Score')
plt.show()
```



The penalized regression had the accuracy of just 0.20 which is extremely poor in performance.

Ensemble Tree Algorithm

```
In [81]: from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=2)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(features_train_scaled, labels_train)

y_pred=clf.predict(features_test_scaled)
```

```
In [82]: random_forest_classifier_accuracy=accuracy_score(labels_test, y_pred)
random_forest_classifier_accuracy
```

```
Out[82]: 0.7291666666666666
```

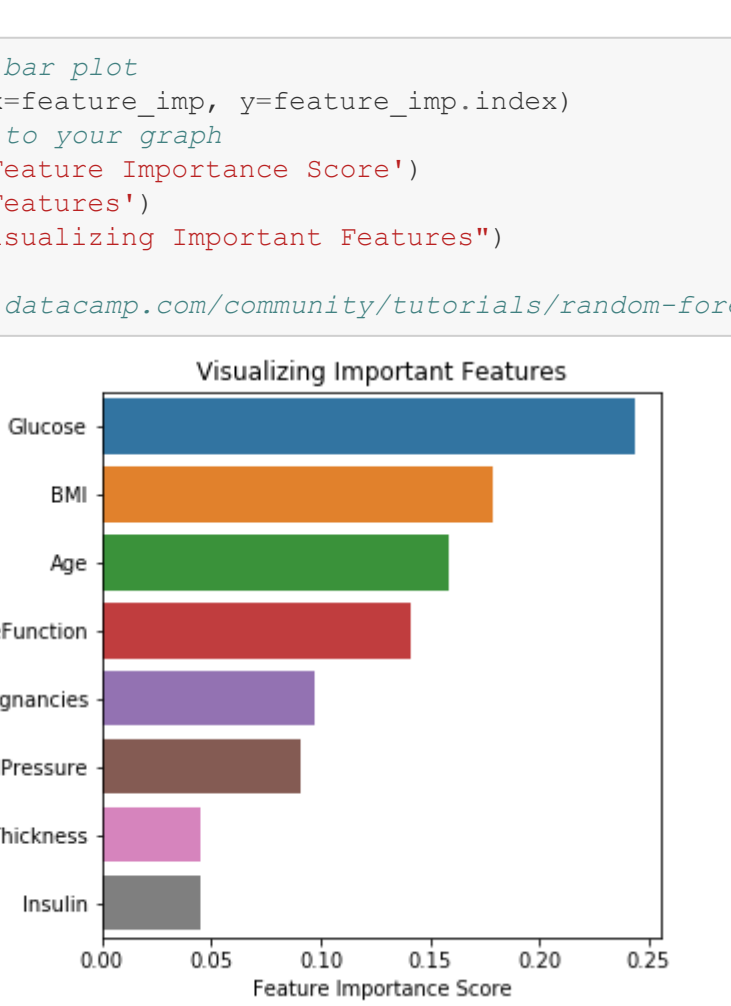
feature importance by Random tree classifier

```
In [83]: feature_imp = pd.Series(clf.feature_importances_,index=dataset.columns[:-1]).sort_values(ascending=Fals
e)
```

```
In [84]: feature_imp
```

```
Out[84]: Glucose                     0.243339
BMI                         0.178864
Age                         0.158452
DiabetesPedigreeFunction    0.141230
Pregnancies                 0.097641
BloodPressure               0.090816
SkinThickness               0.044935
Insulin                     0.044722
dtype: float64
```

```
In [85]: # Creating a bar plot
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.show()
#https://www.datacamp.com/community/tutorials/random-forests-classifier-python
```



from the plot, the Glucose, BMI, Age were the most prominent features and Insulin had less significance.

Discussion about Features Importance Ranking

Top 3 features obtained are:

1. The `kNN classifier` feature importance ranking suggest that Glucose, Insulin, Age
2. The `Logistic Regression` feature importance ranking suggest that Glucose, BMI, Diabetes Pedigree function
3. The `Random tree classifier` feature importance ranking suggest that Glucose, BMI, Age

All the 3 models predicted `Glucose` to be the most prominent feature. The scientific literature `Risk Factors for Type 2 Diabetes Mellitus, Published in  The Journal of cardiovascular nursing` suggests that Family history of diabetes [pedigree] , Age is a contributing factor as predicted by Logistic regression and random tree classifier respectively. The article also points that BMI as an important cause for diabetes and predicted by logistic regression.

Summary - Finding Best Model

```
In [86]: standardized_knn[5][0]      # kNN accuray of k=5
```

```
Out[86]: 0.765625
```

```
In [87]: random_forest_classifier_accuracy  # random forest acccuracy
```

```
Out[87]: 0.7291666666666666
```

```
In [88]: logistic_regression_accuracy # logistic regression accuracy
```

```
Out[88]: 0.7447916666666666
```

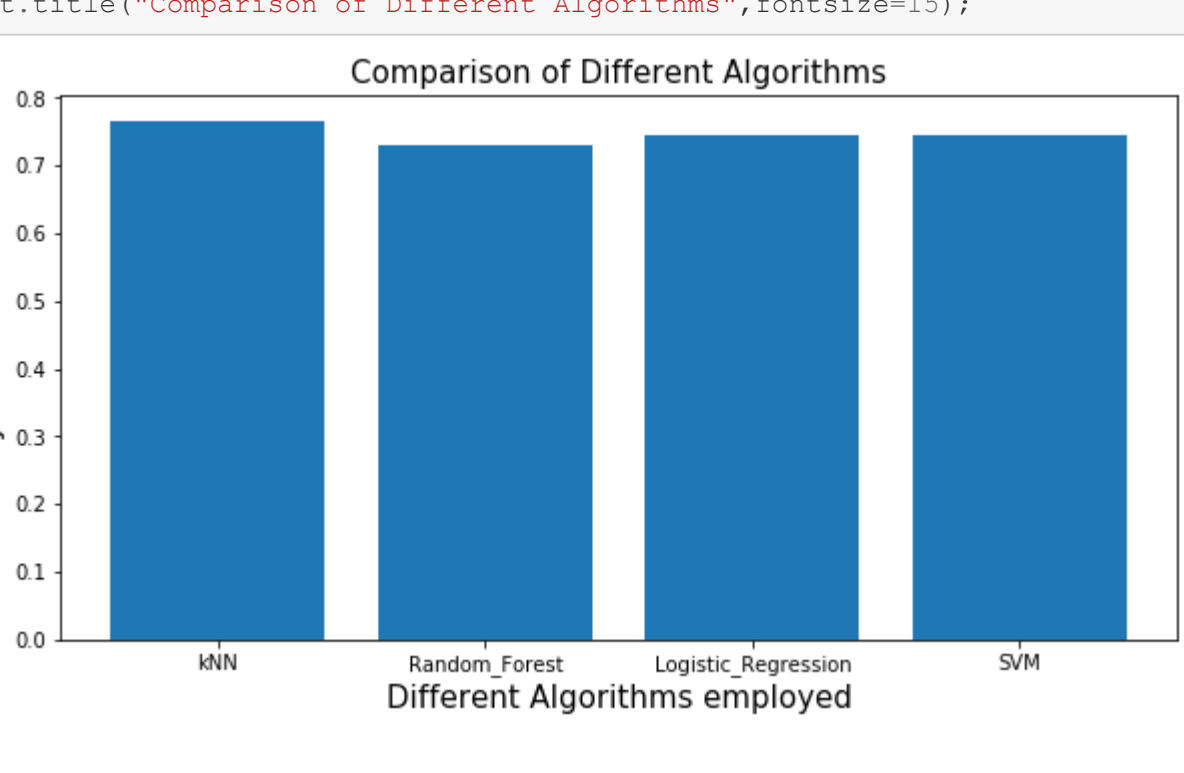
```
In [89]: svm_svc_accuracy    # svm best model accuracy
```

```
Out[89]: 0.7447916666666666
```

```
In [90]: accuracy_list=[standardized_knn[5][0],random_forest_classifier_accuracy,logistic_regression_accuracy,sv
m_svc_accuracy]
algo_list=['kNN','Random_Forest','Logistic_Regression','SVM']
```

Plots to visualize the results

```
In [91]: rcParams['figure.figsize'] = 10,5

plt.bar(algo_list,accuracy_list)
plt.xlabel("Different Algorithms employed", fontsize=15)
plt.ylabel("Accuracy Score observed",fontsize=15)
plt.title("Comparison of Different Algorithms",fontsize=15);
```



Conclusion:

```
    The kNN, Random forest, Logistic Regreesion, SVM seemed to peform in a similar way. The Penal
ized Regression   resulted in a worst performance metric.
```

For the diabetics prediction dataset, the kNN algorithm seemed to perform better in comparison to the other machine learning algorithms.