Health Care

DESCRIPTION

NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatments for the most chronic, costly, and consequential diseases.

The dataset used in this project is originally from NIDDK. The objective is to predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Build a model to accurately predict whether the patients in the dataset have diabetes or not.

Dataset Description

The datasets consists of several medical predictor variables and one target variable (Outcome). Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and more.

Variables Description Pregnancies Number of times pregnant Glucose Plasma glucose concentration in an oral glucose tolerance test BloodPressure Diastolic blood pressure (mm Hg) SkinThickness Triceps skinfold thickness (mm) Insulin Two hour serum insulin BMI Body Mass Index DiabetesPedigreeFunction Diabetes pedigree function Age Age in years Outcome Class variable (either 0 or 1). 268 of 768 values are 1, and the others are 0

```
from io import IncrementalNewlineDecoder
##import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Saving...

```
from matplotlib import style
```

```
## import the data
diabetes= pd.read_csv("/content/health care diabetes.csv")
```

```
diabetes
```

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

768 rows × 9 columns

```
diabetes.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

Here 1 indicates the person is diabetes and 0 indicates the person is Non-diabetes.

```
##  columnname
diabetes.columns
```

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

```
## count of outcome column
diabetes.groupby('Outcome').size()
```

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

```
##checking null value
diabetes.isnull().any()
```

```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```



```
##info
diabetes.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    int64  
 4   Insulin                   768 non-null    int64  
 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(2), int64(7)
memory usage: 54.1 KB
```

A count of frequency (plot)describing the data types and the count of variables.

```
##glucose

diabetes['Glucose'].value_counts().head(10)
```

```
99     17
100    17
111    14
129    14
125    14
106    14
112    13
108    13
95     13
105    13
Name: Glucose, dtype: int64
```

```
diabetes['Glucose']
```

```
0      148
1       85
2      183
3       89
```

```
4      137
       ...
763    101
764    122
765    121
766    126
767     93
Name: Glucose, Length: 768, dtype: int64
```

```
##bloodpressure
diabetes['BloodPressure'].value_counts().head(10)
```

```
70    57
74    52
78    45
68    45
72    44
64    43
80    40
76    39
60    37
0     35
Name: BloodPressure, dtype: int64
```

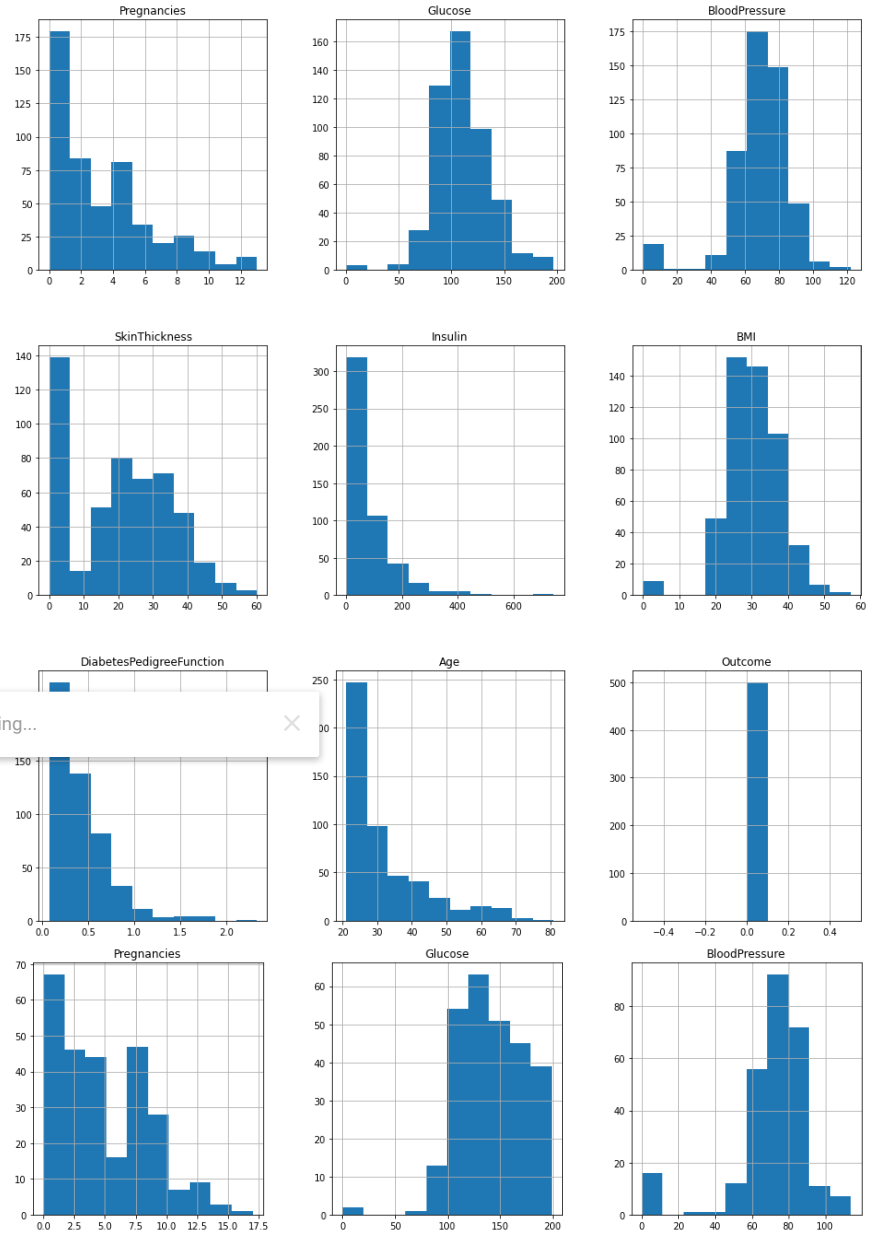
Double-click (or enter) to edit

the help of groupby and outcome we can create all column histogram

```
## the function will draw histogram by data column nameand title
def plot_histogram(data_val,title_name):
    plt.hist(data_val,color="green")
    #plt.grid(axis='y', alpha=0.75)
    plt.title(title_name,fontsize=15)
    plt.show()
```

```
diabetes.groupby('Outcome').hist(figsize=(16, 18))
```

```
Outcome
0    [[AxesSubplot(0.125,0.670278;0.215278x0.209722...
1    [[AxesSubplot(0.125,0.670278;0.215278x0.209722...
dtype: object
```



```
#function to get total count of zeros and outcome details together
def get_zeros_outcome_count(data,column_name):
    count = data[data[column_name] == 0].shape[0]
    print("Total No of zeros found in " + column_name + " : " + str(count))
    print(data[data[column_name] == 0].groupby('Outcome')['Age'].count())
```

```
#Checking count of zeros in blood pressure
get_zeros_outcome_count(diabetes,'BloodPressure')
```

```
Total No of zeros found in BloodPressure : 35
Outcome
0    19
1    16
Name: Age, dtype: int64
```

```
##checking count of zeros in glucose
get_zeros_outcome_count(diabetes,'Glucose')
```

```
Total No of zeros found in Glucose : 5
Outcome
0    3
1    2
Name: Age, dtype: int64
```

```
##checking count of zeros in skinthickness
get_zeros_outcome_count(diabetes,'SkinThickness')
```

```
Total No of zeros found in SkinThickness : 227
Outcome
0    139
```

Saving...

```
##checking count of zeros in BMI
get_zeros_outcome_count(diabetes,'BMI')
```

```
Total No of zeros found in BMI : 11
Outcome
0    9
1    2
Name: Age, dtype: int64
```

```
##checking count of zeros in insulin
get_zeros_outcome_count(diabetes,'Insulin')
```

```
Total No of zeros found in Insulin : 374
Outcome
0    236
1    138
Name: Age, dtype: int64
```

After analysing above data we found lots of 0 in Insulin and SkinThickness and removing them or putting mean value will not good dataset. However, we can remove "BloodPressure", "BMI" and "Glucose" zeros row

```
diabetes_mod = diabetes[(diabetes.BloodPressure != 0) & (diabetes.BMI != 0) & (diabetes.Glucose != 0)]
print(diabetes_mod.shape)
```

```
(724, 9)
```

```
## the stats of data after removing bloodpressure,bmi,glucose 0 rows
diabetes_mod.describe().transpose()
```

	count	mean	std	min	25%	50%	75%
Pregnancies	724.0	3.866022	3.362803	0.000	1.000	3.000	6.0000
Glucose	724.0	121.882597	30.750030	44.000	99.750	117.000	142.0000
BloodPressure	724.0	72.400552	12.379870	24.000	64.000	72.000	80.0000

Data Exploration

1. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.
2. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.
3. Perform correlation analysis. Visually explore it using a heat m ap.

Outcome	724.0	0.343923	0.475344	0.000	0.000	0.000	1.0000

```
#Lets create positive variable and store all 1 value Outcome data
Positive = diabetes_mod[diabetes_mod['Outcome']==1]
Positive.head(5)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	6	148	72	35	0	33.6	
2	8	183	64	0	0	23.3	
4	0	137	40	35	168	43.1	
6	3	78	50	32	88	31.0	
8	2	197	70	45	543	30.5	

Saving...

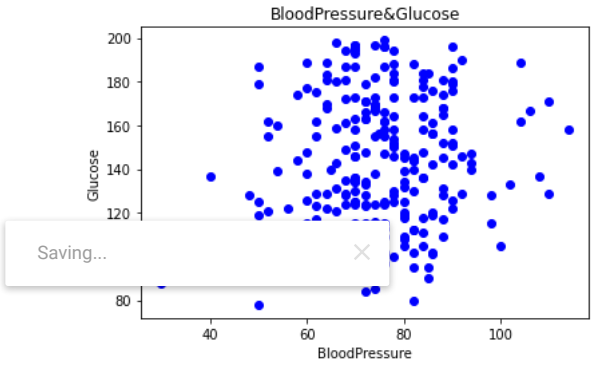
```
gsize=(14, 13),histtype='stepfilled',bins=20,color="blue",edgecolor="orange")
```

scatterplot for positive outcome

```
#function to create scatter plot
def create_scatter_plot(first_value,second_value,x_label,y_label,colour):
    plt.scatter(first_value,second_value, color=[colour])
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    title_name = x_label + '&' + y_label
    plt.title(title_name)
    plt.show()
```

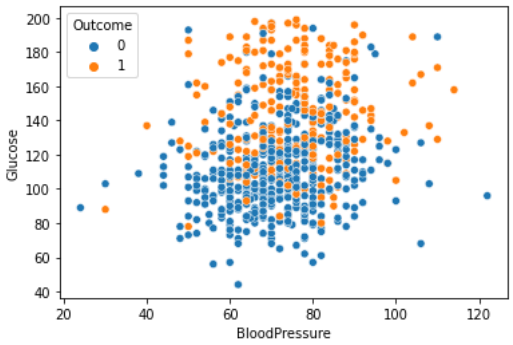
```
BloodPressure = Positive['BloodPressure']
Glucose = Positive['Glucose']
SkinThickness = Positive['SkinThickness']
Insulin = Positive['Insulin']
BMI = Positive['BMI']
```

```
create_scatter_plot(Positive['BloodPressure'],Positive['Glucose'],'BloodPressure','Glucose','blue')
```



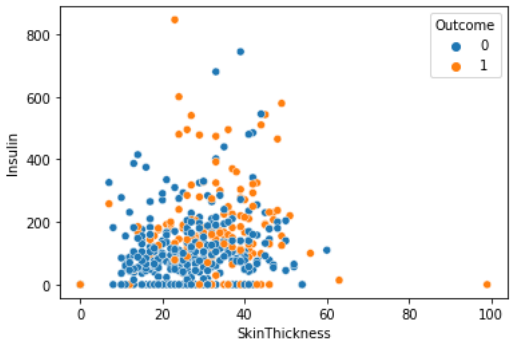
As I can compare postive & negative scatter plot with sns scatter plot all the value is matching, so now I will create common scatter plot for both outcome.

```
g =sns.scatterplot(x= "BloodPressure" ,y= "Glucose",
              hue="Outcome",
              data=diabetes_mod);
```




```
B=sns.scatterplot(x="BMI",y="Insulin",
                  hue="Outcome",data=diabetes_mod);
```


```
s=sns.scatterplot(x="SkinThickness",y="Insulin",hue="Outcome",data=diabetes_mod);
```




```
##correlation matrix
diabetes_mod.corr()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin
Pregnancies	1.000000	0.134915	0.209668	-0.095683	-0.080059
Glucose	0.134915	1.000000	0.223331	0.074381	0.337896
BloodPressure	0.209668	0.223331	1.000000	0.011777	-0.046856
SkinThickness	-0.095683	0.074381	0.011777	1.000000	0.420874
Insulin	-0.080059	0.337896	-0.046856	0.420874	1.000000
BMI	0.012342	0.223276	0.287403	0.401528	0.191831
DiabetesPedigreeFunction	-0.025996	0.136630	-0.000075	0.176253	0.182656
Age	0.557066	0.263560	0.324897	-0.128908	-0.049412
Outcome	0.224417	0.488384	0.166703	0.092030	0.145488

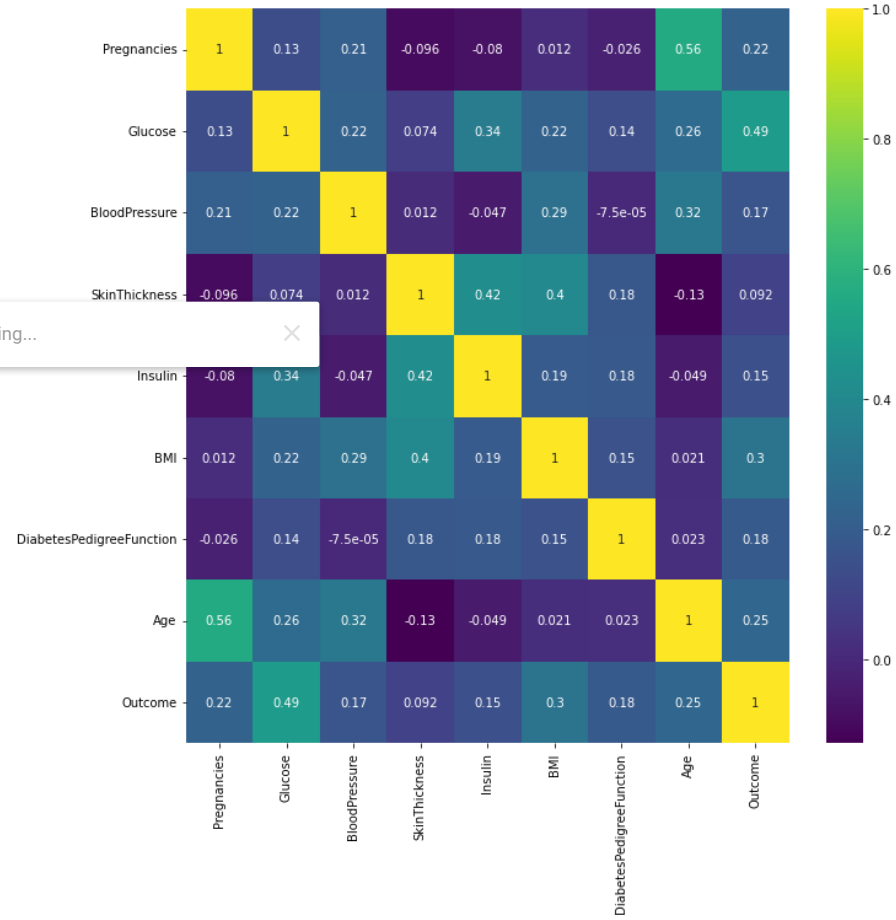
HEATMAP

```
##correlation heatmap
plt.subplots(figsize=(10,10))
sns.heatmap(diabetes_mod.corr())
```

```
<AxesSubplot:>
```


```
plt.subplots(figsize=(11,11))
sns.heatmap(diabetes_mod.corr(),annot=True,cmap='viridis')
```

```
<AxesSubplot:>
```



Logistic Regression and model building

```
feature_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
X = diabetes_mod[feature_names]
y = diabetes_mod.Outcome
```

```
X.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	

```
## train test split model
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y, test_size=0.3,random_state=12)
```

4	0	137	40	35	168	43.1

To Create Model

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

## import warning filter
from warnings import simplefilter
## ignore all future warnings
                                                    pry=FutureWarning)
```

Saving...

```
## logestic regression model
#LR Model
model_LR = LogisticRegression(solver='liblinear')
model_LR.fit(X_train,y_train)
```

```
LogisticRegression(solver='liblinear')
```

```
##LR model score and accuracy score

print("LogisticRegression Score :{}".format(model_LR.score(X_train,y_train)))
y_pred = model_LR.predict(X_test)
scores = (accuracy_score(y_test, y_pred))
print("LogisticRegression Accuracy Score :{}".format(scores))
```

```
LogisticRegression Score :0.7707509881422925
LogisticRegression Accuracy Score :0.7477064220183486
```

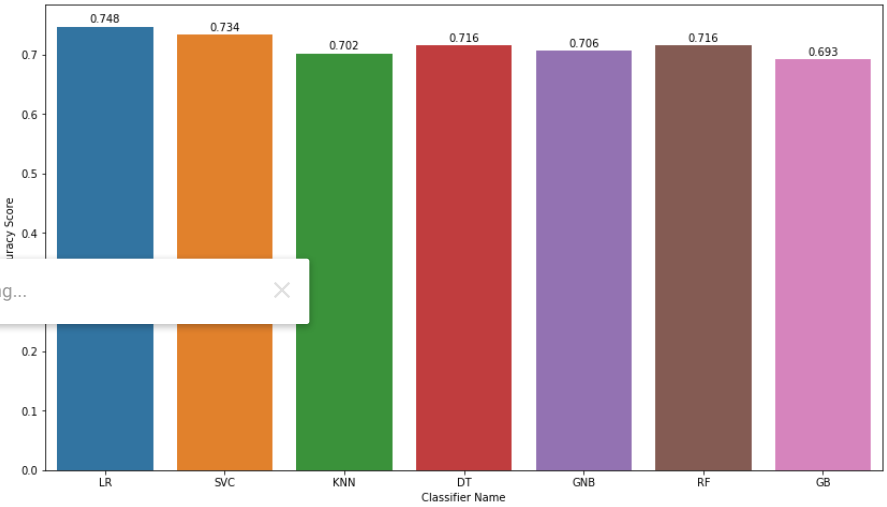
```
accuracyScores = []
modelScores = []
models = []
names = []
#Store algorithm into array to get score and accuracy
models.append(('LR', LogisticRegression(solver='liblinear')))
models.append(('SVC', SVC()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('DT', DecisionTreeClassifier()))
models.append(('GNB', GaussianNB()))
models.append(('RF', RandomForestClassifier()))
models.append(('GB', GradientBoostingClassifier()))
```

```
##fit each model in a loop and calculate the accuracy of the respective model using the “accuracy_score”
for name, model in models:
    model.fit(X_train, y_train)
    modelScores.append(model.score(X_train,y_train))
    y_pred = model.predict(X_test)
    accuracyScores.append(accuracy_score(y_test, y_pred))
    names.append(name)
```

```
tr_split_data = pd.DataFrame({'Name': names, 'Score': modelScores,'Accuracy Score': accuracyScores})
print(tr_split_data)
```

```
   Name     Score  Accuracy Score
0   LR  0.770751        0.747706
1  SVC  0.772727        0.733945
2  KNN  0.804348        0.701835
3   DT  1.000000        0.715596
4  GNB  0.772727        0.706422
5   RF  1.000000        0.715596
6   GB  0.948617        0.692661
```

```
##graphs
plt.subplots(figsize=(14,8))
axis = sns.barplot(x = 'Name', y = 'Accuracy Score', data = tr_split_data)
axis.set(xlabel='Classifier Name', ylabel='Accuracy Score')
for p in axis.patches:
    height = p.get_height()
    axis.text(p.get_x() + p.get_width()/2, height + 0.007, '{:1.3f}'.format(height), ha="center")

plt.show()
```




```
## check confusion matrix
#y is label value & X is feature value
cm = confusion_matrix(y,model_LR.predict(X))
cm
```

```
array([[425,  50],
       [121, 128]])
```

```
print(classification_report(y,model_LR.predict(X)))
```

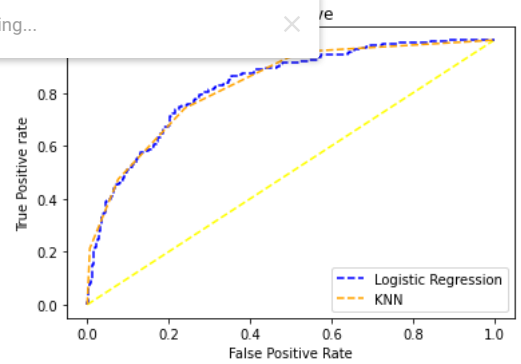
```
              precision    recall  f1-score   support

           0       0.78      0.89      0.83       475
           1       0.72      0.51      0.60       249

    accuracy                           0.76       724
   macro avg       0.75      0.70      0.72       724
weighted avg       0.76      0.76      0.75       724
```

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
```

```
#Preparing ROC Curve (Receiver Operating Characteristics Curve) - LR, KNN
# predict probabilities for LR
probs_LR = model_LR.predict_proba(X)
# predict probabilities for KNN - where models[2] is KNN
model_KNN = KNeighborsClassifier(n_neighbors=4)
```

```
model_KNN.fit(X_train, y_train)
probs_KNN = model_KNN.predict_proba(X)

# Sklearn has a very potent method roc_curve() which computes the ROC for your classifier in a matter of seconds! It returns the FPR, TPR, ar
fpr, tpr, thresholds = roc_curve(y, probs_LR[:, 1],pos_label=1)
fpr1, tpr1, thresholds1 = roc_curve(y, probs_KNN[:, 1],pos_label=1)

# roc curve for tpr = fpr
random_probs = [0 for i in range(len(y))]
p_fpr, p_tpr, _ = roc_curve(y, random_probs, pos_label=1)

# plot no skill
plt.plot(p_fpr, p_tpr, linestyle='--',color='yellow')
plt.plot(fpr, tpr, linestyle='--',color='blue', label='Logistic Regression')
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='KNN')

# plot the roc curve for the model
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')
#plt.plot(fpr, tpr, marker='.')
plt.legend(loc='best')
plt.show();
# keep probabilities for the positive outcome only
#The AUC score can be computed using the roc_auc_score() method of sklearn: calculate AUC
auc_LR = roc_auc_score(y, probs_LR[:, 1])
auc_KNN = roc_auc_score(y, probs_KNN[:, 1])
print('AUC LR: %.5f' % auc_LR, 'AUC KNN: %.5f' % auc_KNN)
```

Saving...




```
AUC LR: 0.83068 AUC KNN: 0.83140
```

```
def generate_graph(recall, precision,name):
    # plot no skill
    # plot the precision-recall curve for the model
    plt.figure()
    plt.subplots(figsize=(10,4))
    plt.plot([0, 1], [0.5, 0.5], linestyle='--',label='No Skill')
    plt.plot(recall, precision, marker='.',label=name)
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title(name)
    plt.legend(loc='best')
    plt.show()


#Store algorithm into array to get score and accuracy
p_r_Models = []
p_r_Models.append(('LR', LogisticRegression(solver='liblinear')))
p_r_Models.append(('KNN', KNeighborsClassifier()))
p_r_Models.append(('DT', DecisionTreeClassifier()))
p_r_Models.append(('GNB', GaussianNB()))
p_r_Models.append(('RF', RandomForestClassifier()))
p_r_Models.append(('GB', GradientBoostingClassifier()))
#Precision Recall Curve for All classifier
for name, model in p_r_Models:
    from sklearn.metrics import precision_recall_curve
    from sklearn.metrics import f1_score
    from sklearn.metrics import auc
    from sklearn.metrics import average_precision_score
```

```
print("\n====================---------- Precision Recall Curve for {} ----------====================\n".format(name))
model.fit(X_train, y_train)
# predict probabilities
probs = model.predict_proba(X)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# predict class values
yhat = model.predict(X)
# calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y, probs)
# calculate F1 score, # calculate precision-recall AUC
f1, auc = f1_score(y, yhat), auc(recall, precision)
# calculate average precision score
ap = average_precision_score(y, probs)
generate_graph(recall, precision,name)
print(str(name) + " calculated value : " + 'F1 Score =%.3f, Area Under the Curve=%.3f, Average Precision=%.3f\n' % (f1, auc, ap))
print("The above precision-recall curve plot is showing the precision/recall for each threshold for a {} model (yellow) compared to a no
```