Healthcare.

Course-end Project 2

DESCRIPTION

Problem Statement:

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

Project Task: Week 1

Data Exploration:

1. Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below, a value of zero does not make sense and thus indicates missing value:

• Glucose

• BloodPressure

• SkinThickness

• Insulin

• BMI

2. Visually explore these variables using histograms. Treat the missing values accordingly.
3. There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

Project Task: Week 2

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 map.

Project Task: Week 3

Data Modeling:

1. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.
2. Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm.

Project Task: Week 4

Data Modeling:

1. Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc. Please be descriptive to explain what values of these parameter you have used.

Data Reporting:

2. Create a dashboard in tableau by choosing appropriate chart types and metrics useful for the business. The dashboard must entail the following:

a. Pie chart to describe the diabetic or non-diabetic population

b. Scatter charts between relevant variables to analyze the relationships

c. Histogram or frequency charts to analyze the distribution of the data

d. Heatmap of correlation analysis among the relevant variables

e. Create bins of these age values: 20-25, 25-30, 30-35, etc. Analyze different variables for these age brackets using a bubble chart.

```
#connect To Drive

from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
#import required libraries

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

```
#import dataset

df=pd.read_csv('/content/drive/MyDrive/CAPSTONE PROJECT/Project 2/Healthcare - Diabetes/health care diabetes.csv')
```

```
#check shape of data

df.shape
```

```
(768, 9)
```

```
#check dataset

df.head()
```

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

```
#check for null values

df.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Project Task: Week 1

Data Exploration:

1. Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below, a value of zero does not make sense and thus indicates missing value:

- Glucose
- BloodPressure
- SkinThickness
- Insulin
- BMI

```
#make list of given column

col=['Glucose', 'BloodPressure', 'SkinThickness','Insulin','BMI']


#write a function to calculate number of zeros from column


def get_zeros_outcome_count(df,column_name):
    count = df[df[column_name] == 0].shape[0]
    print("Total No of zeros found in " + column_name + " : " + str(count))


#print zeros from each columns

for i in col:
  get_zeros_outcome_count(df,i)
```

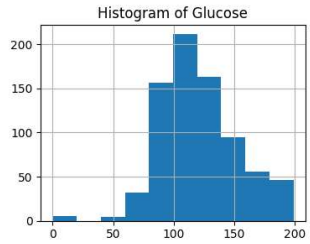
```
Total No of zeros found in Glucose : 5
Total No of zeros found in BloodPressure : 35
Total No of zeros found in SkinThickness : 227
Total No of zeros found in Insulin : 374
Total No of zeros found in BMI : 11
```

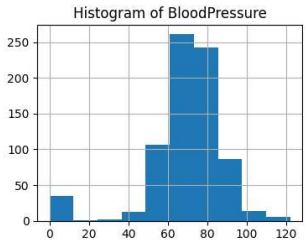
We can clearly see the number of zeros present in each column

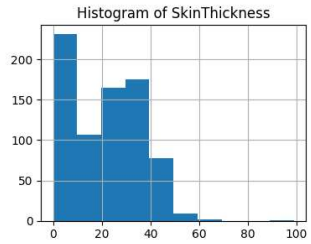
we will see graph without treating it an with treating it

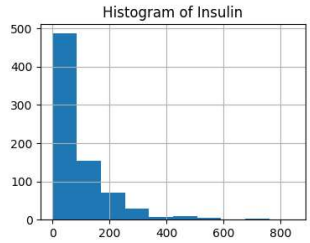
2. Visually explore these variables using histograms. Treat the missing values accordingly.

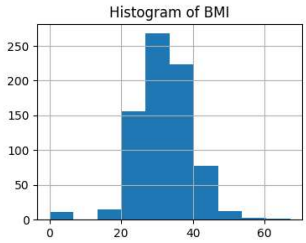
```
#plot a histogram without treating zeros

plt.figure(figsize=(15,7))
plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.4,
                    hspace=0.4)
plt.subplot(2,3,1)
plt.title('Histogram of Glucose')
df['Glucose'].hist()
plt.subplot(2,3,2)
plt.title('Histogram of BloodPressure')
df['BloodPressure'].hist()
plt.subplot(2,3,3)
plt.title('Histogram of SkinThickness')
df['SkinThickness'].hist()
plt.subplot(2,3,4)
plt.title('Histogram of Insulin')
df['Insulin'].hist()
plt.subplot(2,3,5)
plt.title('Histogram of BMI')
df['BMI'].hist()
plt.show()
```












```
#take all featured colum in new data
# we will take all columns except pregnacies because pregnancy can be '0'
# and we will replace '0' from all data as null

df_train=df.drop(['Pregnancies','Outcome'],axis=1)


#now we have feactured columns where the '0' values are present
#which has no meaning
#so to treat that we will first convert all zero values with Nan

df_train=df_train.replace(0,np.NAN)


# now check null values

df_train.isnull().sum()
```

```
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
dtype: int64
```

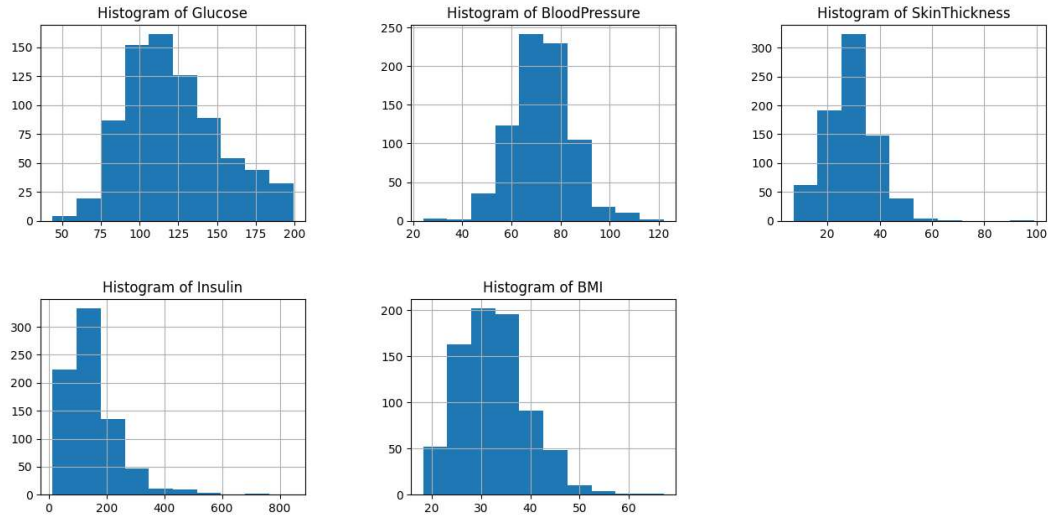
It is same as above number of zeros in columns

```
#now treat the null values using KNN method
#import required libraries

from sklearn.impute import KNNImputer


# Create a KNN imputer object with k=7
imputer = KNNImputer(n_neighbors=7)

# Perform KNN imputation on the dataset
df_train = pd.DataFrame(imputer.fit_transform(df_train), columns=df_train.columns)


#check again if null values treated or not

df_train.isnull().sum()
```

```
Glucose                   0
BloodPressure             0
SkinThickness             0
Insulin                   0
BMI                       0
```

```
DiabetesPedigreeFunction    0
Age                         0
dtype: int64
```

```
# now check graph again for treated values

plt.figure(figsize=(15,7))
plt.subplots_adjust(left=0.1,
····················bottom=0.1,
····················right=0.9,
····················top=0.9,
····················wspace=0.4,
····················hspace=0.4)
plt.subplot(2,3,1)
plt.title('Histogram of Glucose')
df_train['Glucose'].hist()
plt.subplot(2,3,2)
plt.title('Histogram of BloodPressure')
df_train['BloodPressure'].hist()
plt.subplot(2,3,3)
plt.title('Histogram of SkinThickness')
df_train['SkinThickness'].hist()
plt.subplot(2,3,4)
plt.title('Histogram of Insulin')
df_train['Insulin'].hist()
plt.subplot(2,3,5)
plt.title('Histogram of BMI')
df_train['BMI'].hist()
plt.show()
```



- As we mentioned we didnt take pregnancies in our featured column

  because pregnancy can be zero

  But after treating missing values we can add pregnancy column

```
#taking pregnancies in one data

extracted_column=df[['Pregnancies']]
```

```
#join with train
#as we want in sequence we are using join ( joining pregnancies column at start )
#otherwise we can just copy data from main df


df_train=extracted_column.join(df_train)


#do the same for outcome column


extracted_column2=df[['Outcome']]


#as we need complete data we are joining outcome also
#as we want data in sequence we are joining outcome column at last

df_train=df_train.join(extracted_column2)


#checking feacture dataset

df_train.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148.0	72.0	35.0	180.857143	33.6	0.627	50.0
1	1	85.0	66.0	29.0	56.857143	26.6	0.351	31.0
2	8	183.0	64.0	27.0	171.000000	23.3	0.672	32.0
3	1	89.0	66.0	23.0	94.000000	28.1	0.167	21.0
4	0	137.0	40.0	35.0	168.000000	43.1	2.288	33.0

```
#save this treated complete dataset
#to use in tableau

df_train.to_csv('diabeties_treated.csv')
```

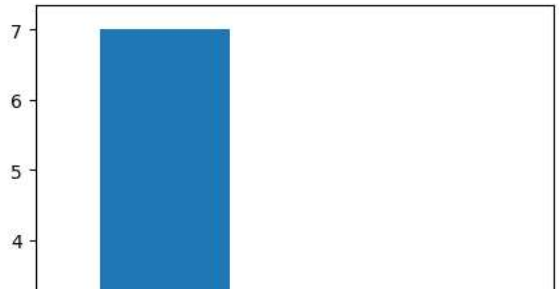
3. There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

```
#checking different type of dtypes present in dataset

count_dtype = df_train.dtypes.value_counts()


#displaying it

count_dtype
```

```
float64    7
int64      2
dtype: int64
```

```
#ploting the dtypes

plt.figure(figsize=(5,5))
count_dtype.plot(kind='bar')
```

```
<Axes: >
```



Project Task: Week 2

Data Exploration:

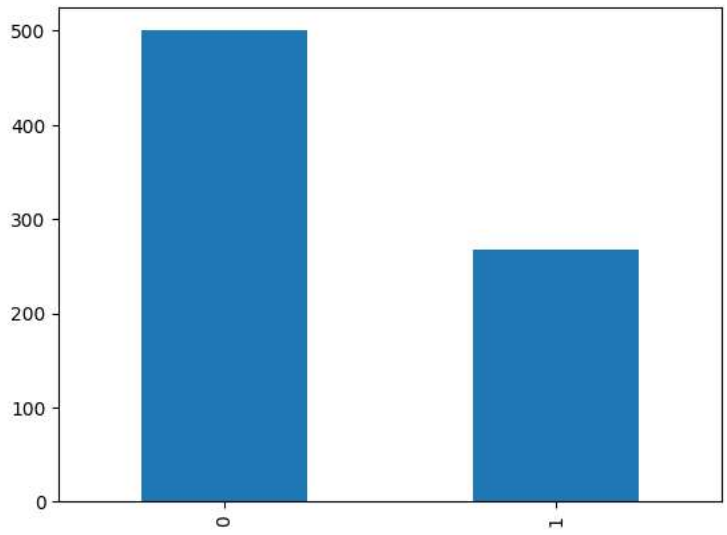
1. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.


```
#checking number of different outcomes

Outcome_values=df_train['Outcome'].value_counts()


Outcome_values
```

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

```
#ploting different outcome

Outcome_values.plot(kind='bar')
```

```
<Axes: >
```

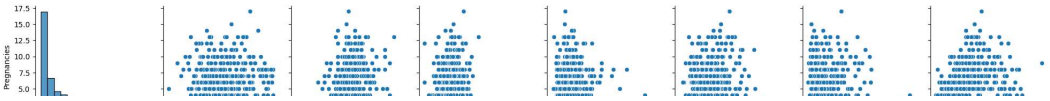


2. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.

```
# seprate the data feature and target

df_train_feature=df_train.drop(columns='Outcome')
df_train_target=df_train['Outcome']
```

```
#ploting pairplot for featured data

plt.figure(figsize=(5,5))

sns.pairplot(data=df_train_feature)
```

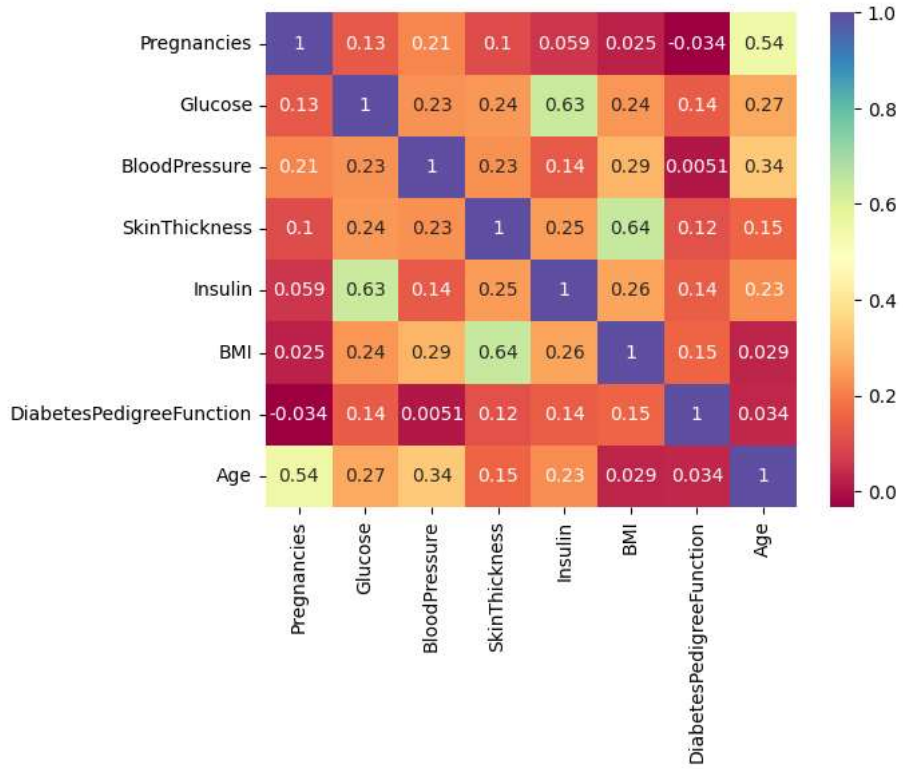
```
<seaborn.axisgrid.PairGrid at 0x7f3e6f225f70>
<Figure size 500x500 with 0 Axes>
```



3. Perform correlation analysis. Visually explore it using a heat map.

```
#plotting heatmap for featured data

sns.heatmap(df_train_feature.corr(),annot=True,cmap='Spectral')
```

```
<Axes: >
```

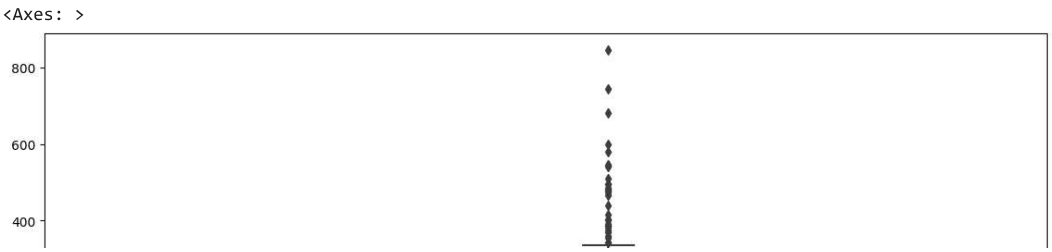


Project Task: Week 3

Data Modeling:

1. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.

```
plt.figure(figsize=(14,5))


sns.boxplot(df_train_feature)
```



Here we can clearly see there are so many outliers in the dataset

2. Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm.

```
#import all the model Libraries to compare

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


#create X and y for Feature and target

X=df_train_feature
y=df_train_target


#split the dataset into train and test dataset

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=7)


# model 1 - Logestic Regression

print('Model Number 1 : Logestic Regression')
log_reg=LogisticRegression(max_iter=500)
model_logReg=log_reg.fit(X_train,y_train)
pred_logReg=log_reg.predict(X_test)
print("Accuracy of model Logistic Regression :", accuracy_score(y_test, pred_logReg))
print('\n')
```

```
Model Number 1 : Logestic Regression
Accuracy of model Logistic Regression : 0.7532467532467533
```

```
# model 2 - Decision Tree

print('Model Number 2 : Decision Tree')
dec_tree = DecisionTreeClassifier()
model_DT=dec_tree.fit(X_train,y_train)
pred_DT=dec_tree.predict(X_test)
print("Accuracy of model Decision Tree :", accuracy_score(y_test, pred_DT))
print('\n')
```

```
Model Number 2 : Decision Tree
Accuracy of model Decision Tree : 0.7186147186147186
```

```
# model 3 - Random Forest Classifier

print('Model Number 3 : Random Forest Classifier')
RFC = RandomForestClassifier()
model_RFC=RFC.fit(X_train,y_train)
```

```
pred_RFC=RFC.predict(X_test)
print("Accuracy of model Random Forest Classifier :", accuracy_score(y_test, pred_RFC))
print('\n')
```

```
Model Number 3 : Random Forest Classifier
Accuracy of model Random Forest Classifier : 0.7489177489177489
```

```
# model 4 - Support Vector Machine

print('Model Number 4 : Support Vector Machine')
svc_modl = SVC()
model_svc=svc_modl.fit(X_train,y_train)
pred_svc=svc_modl.predict(X_test)
print("Accuracy of model Support Vector Machine :", accuracy_score(y_test, pred_svc))
print('\n')
```

```
Model Number 4 : Support Vector Machine
Accuracy of model Support Vector Machine : 0.7532467532467533
```

```
# model 5 - K-Nearest neighbour

print('Model Number 5 : K-Nearest neighbour')
KNN = KNeighborsClassifier()
model_knn=KNN.fit(X_train,y_train)
pred_knn=KNN.predict(X_test)
print("Accuracy of model K-Nearest neighbour :", accuracy_score(y_test, pred_knn))
print('\n')
```

```
Model Number 5 : K-Nearest neighbour
Accuracy of model K-Nearest neighbour : 0.7359307359307359
```

```
# print accuracy of every model to compare
print('\n')
print("Accuracy of model Logistic Regression :", accuracy_score(y_test, pred_logReg))
print("Accuracy of model Decision Tree :", accuracy_score(y_test, pred_DT))
print("Accuracy of model Random Forest Classifier :", accuracy_score(y_test, pred_RFC))
print("Accuracy of model Support Vector Machine :", accuracy_score(y_test, pred_svc))
print("Accuracy of model K-Nearest neighbour :", accuracy_score(y_test, pred_knn))
print('\n')
```

```
Accuracy of model Logistic Regression : 0.7532467532467533
Accuracy of model Decision Tree : 0.7186147186147186
Accuracy of model Random Forest Classifier : 0.7489177489177489
Accuracy of model Support Vector Machine : 0.7532467532467533
Accuracy of model K-Nearest neighbour : 0.7359307359307359
```

Project Task: Week 4

Data Modeling:

1. Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc. Please be descriptive to explain what values of these parameter you have used.

```
#import reuired libraries


from sklearn.metrics import classification_report, confusion_matrix, roc_curve, roc_auc_score



#print classification report for every model

print("Logistic Regression Classification Report:\n",classification_report(y_test,pred_logReg))
print('\n')
```

```
print("Decision tree Classification Report:\n",classification_report(y_test,pred_DT))
print('\n')
print("Random Forest Classification Report:\n",classification_report(y_test,pred_RFC))
print('\n')
print("Support Vector Machine Classification Report:\n",classification_report(y_test,pred_svc))
print('\n')
print("K_Nearest Neighbour Classification Report:\n",classification_report(y_test,pred_knn))
```

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.85      0.81       147
           1       0.69      0.58      0.63        84

    accuracy                           0.75       231
   macro avg       0.74      0.72      0.72       231
weighted avg       0.75      0.75      0.75       231



Decision tree Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.72      0.77       147
           1       0.59      0.71      0.65        84

    accuracy                           0.72       231
   macro avg       0.70      0.72      0.71       231
weighted avg       0.73      0.72      0.72       231



Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.83      0.81       147
           1       0.67      0.61      0.64        84

    accuracy                           0.75       231
   macro avg       0.73      0.72      0.72       231
weighted avg       0.74      0.75      0.75       231



Support Vector Machine Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.90      0.82       147
           1       0.74      0.50      0.60        84

    accuracy                           0.75       231
   macro avg       0.75      0.70      0.71       231
weighted avg       0.75      0.75      0.74       231



K_Nearest Neighbour Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.78      0.79       147
           1       0.63      0.65      0.64        84

    accuracy                           0.74       231
   macro avg       0.72      0.72      0.72       231
weighted avg       0.74      0.74      0.74       231
```

```
# create confusion matrix for all the models

conf_logReg = confusion_matrix(y_test, pred_logReg)
conf_DT = confusion_matrix(y_test, pred_DT)
conf_RFC = confusion_matrix(y_test, pred_RFC)
conf_svc = confusion_matrix(y_test, pred_svc)
conf_knn = confusion_matrix(y_test, pred_knn)


#print all confusion matrix
print('\n')
print("Logistic Regression Confusion Matrix:\n\n", conf_logReg)
print('\n')
print("Decision tree Confusion Matrix:\n\n", conf_DT)
print('\n')
```

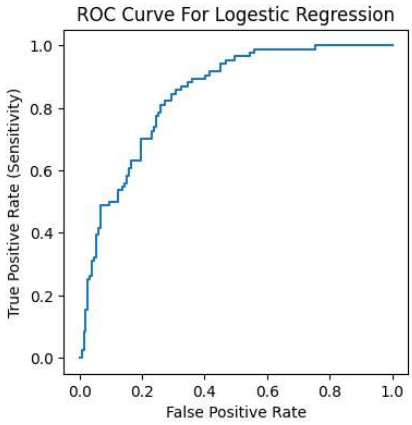
```
print("Random Forest Confusion Matrix:\n\n", conf_RFC)
print('\n')
print("Support Vector Machine Confusion Matrix:\n\n", conf_svc)
print('\n')
print("K-Nearest Neighbour Confusion Matrix:\n\n", conf_knn)
print('\n')
```

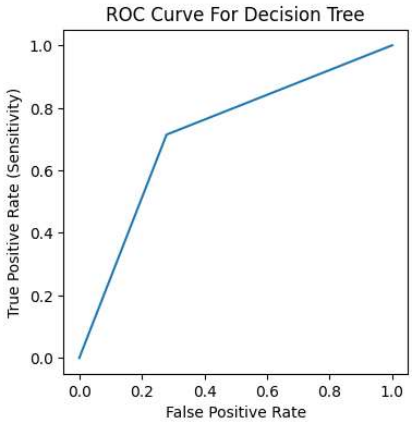
```
Logistic Regression Confusion Matrix:

 [[125  22]
 [ 35  49]]


Decision tree Confusion Matrix:

 [[106  41]
 [ 24  60]]


Random Forest Confusion Matrix:

 [[122  25]
 [ 33  51]]


Support Vector Machine Confusion Matrix:

 [[132  15]
 [ 42  42]]


K-Nearest Neighbour Confusion Matrix:

 [[115  32]
 [ 29  55]]
```

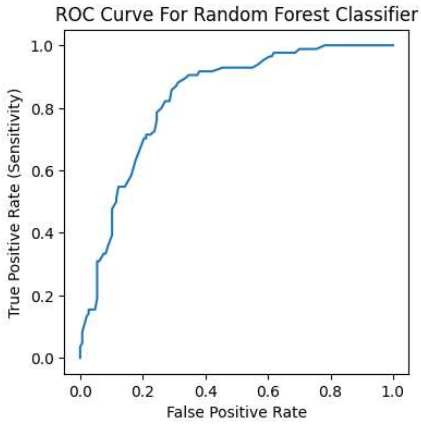
```
# Generate an ROC curve for each model


plt.figure(figsize=(15,15))
plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.4,
                    hspace=0.4)



fpr, tpr, thresholds = roc_curve(y_test, log_reg.decision_function(X_test))
plt.subplot(3,3,1)
plt.plot(fpr, tpr, label="ROC Curve For Logestic Regression ")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate (Sensitivity)")
plt.title("ROC Curve For Logestic Regression")


plt.subplot(3,3,2)
fpr, tpr, thresholds = roc_curve(y_test, dec_tree.predict_proba(X_test)[:,1])
plt.plot(fpr, tpr, label="ROC Curve_Decision Tree")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate (Sensitivity)")
plt.title("ROC Curve For Decision Tree")



plt.subplot(3,3,3)
fpr, tpr, thresholds = roc_curve(y_test, RFC.predict_proba(X_test)[:,1])
plt.plot(fpr, tpr, label="ROC Curve_RFC")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate (Sensitivity)")
plt.title("ROC Curve For Random Forest Classifier")
```

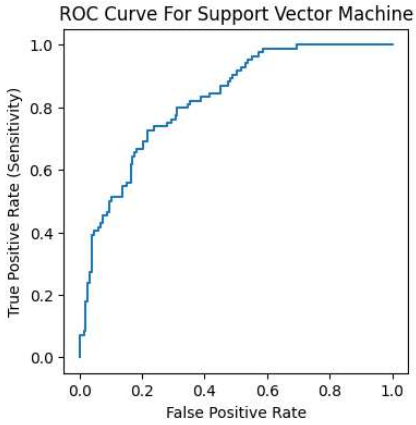
```
plt.subplot(3,3,4)
fpr, tpr, thresholds = roc_curve(y_test, svc_modl.decision_function(X_test))
plt.plot(fpr, tpr, label="ROC Curve_svc_modl")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate (Sensitivity)")
plt.title("ROC Curve For Support Vector Machine")
```

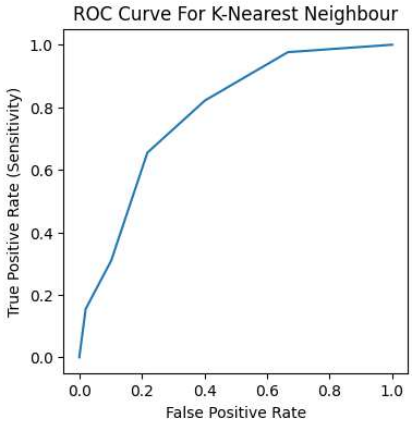
```
plt.subplot(3,3,5)
fpr, tpr, thresholds = roc_curve(y_test, KNN.predict_proba(X_test)[:,1])
plt.plot(fpr, tpr, label="ROC Curve_KNN")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate (Sensitivity)")
plt.title("ROC Curve For K-Nearest Neighbour")
plt.show()
```












```
# calculate AUC score for all models

print('\n')
auc_Log_reg = roc_auc_score(y_test, log_reg.decision_function(X_test))
print("Logistic Regression AUC Score:", auc_Log_reg)
auc_DT = roc_auc_score(y_test, dec_tree.predict_proba(X_test)[:,1])
print("Decision Tree AUC Score:", auc_DT)
auc_RFC = roc_auc_score(y_test, RFC.predict_proba(X_test)[:,1])
print("Random Forest AUC Score:", auc_RFC)
auc_svc = roc_auc_score(y_test, svc_modl.decision_function(X_test))
print("Support Vector Mahine AUC Score:", auc_svc)
auc_knn = roc_auc_score(y_test, KNN.predict_proba(X_test)[:,1])
print("K-Nearest neighbour AUC Score:", auc_knn)
```

```
print('\n')
```

```
Logistic Regression AUC Score: 0.8413508260447036
Decision Tree AUC Score: 0.7176870748299321
Random Forest AUC Score: 0.8296080336896663
Support Vector Mahine AUC Score: 0.823291221250405
K-Nearest neighbour AUC Score: 0.7796809199870425
```

So we can conclude that After Compairing All models

We found Logestic Regression is Best Fit For Given Dataset

✓ 0s completed at 11:58 AM