```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
df=pd.read_csv('/content/drive/MyDrive/diabetes.csv')
```

```
df.head()
```

	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

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.tail()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

```
df.sample(10)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
153	1	153	82	42	485	40.6	0.687	23	0
18	1	103	30	38	83	43.3	0.183	33	0
478	8	126	74	38	75	25.9	0.162	39	0
0	6	148	72	35	0	33.6	0.627	50	1
590	11	111	84	40	0	46.8	0.925	45	1
731	8	120	86	0	0	28.4	0.259	22	1
572	3	111	58	31	44	29.5	0.430	22	0
392	1	131	64	14	415	23.7	0.389	21	0
316	3	99	80	11	64	19.3	0.284	30	0
118	4	97	60	23	0	28.2	0.443	22	0

```
df.shape
```

```
(768, 9)
```

```
df.dtypes
```

	0
Pregnancies	int64
Glucose	int64
BloodPressure	int64
SkinThickness	int64
Insulin	int64
BMI	float64
DiabetesPedigreeFunction	float64
Age	int64
Outcome	int64

dtype: object

```
df.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
```

```
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	76
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	

```
#before drop the duplicates
df.shape
```

```
(768, 9)
```

```
df=df.drop_duplicates()
```

```
#after drop the duplicates
df.shape
```

```
(768, 9)
```

```
#checking of null values
df.isnull().sum()
```

	0
Pregnancies	0
Glucose	0
BloodPressure	0
SkinThickness	0
Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
Outcome	0

dtype: int64

```
df.columns
```

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

```
#checking no of zero values in the dataset
print('No. of Zero values in Glucose',df[df['Glucose']==0].shape[0])
```

```
No. of Zero values in Glucose 5
```

```
print('No. of Zero values in BloodPressure',df[df['BloodPressure']==0].shape[0])
```

```
No. of Zero values in BloodPressure 35
```

```
print('No. of Zero values in SkinThickness',df[df['SkinThickness']==0].shape[0])
```

```
No. of Zero values in SkinThickness 227
```

```
print('No. of Zero values in Insulin',df[df['Insulin']==0].shape[0])
```

```
No. of Zero values in Insulin 374
```

```
print('No. of Zero values in BMI',df[df['BMI']==0].shape[0])
```

```
No. of Zero values in BMI 11
```

```
#replacing zeros with mean of that columns
df['Glucose']=df['Glucose'].replace(0,df['Glucose'].mean())
print('No. of Zero values in Glucose',df[df['Glucose']==0].shape[0])
```

```
No. of Zero values in Glucose 0
```

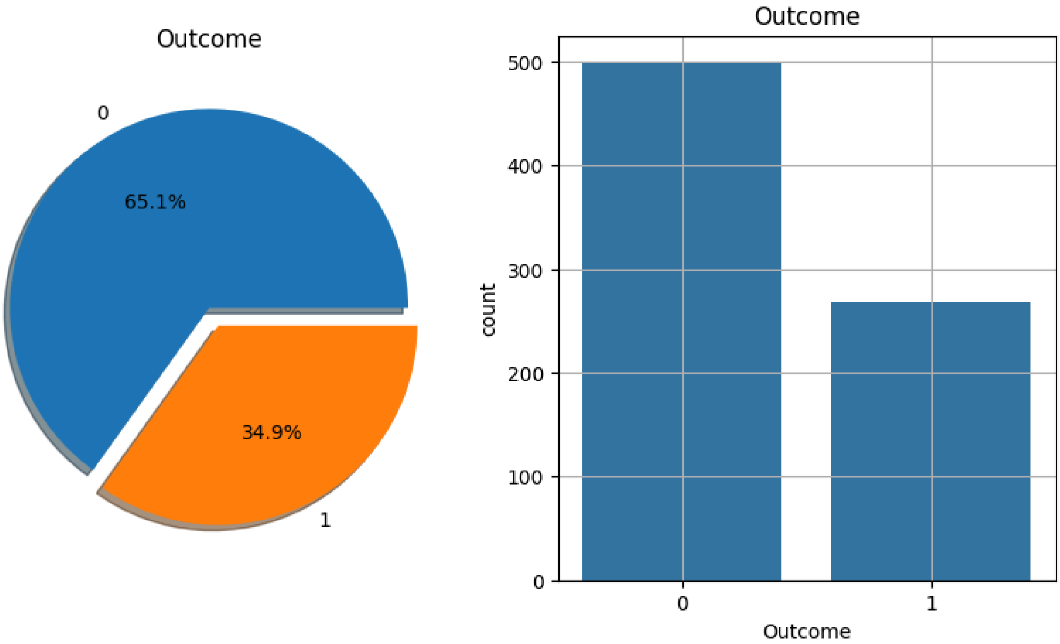
```
df['BloodPressure']=df['BloodPressure'].replace(0,df['BloodPressure'].mean())
df['SkinThickness']=df['SkinThickness'].replace(0,df['SkinThickness'].mean())
df['Insulin']=df['Insulin'].replace(0,df['Insulin'].mean())
df['BMI']=df['BMI'].replace(0,df['BMI'].mean())
```

```
df.describe()
```

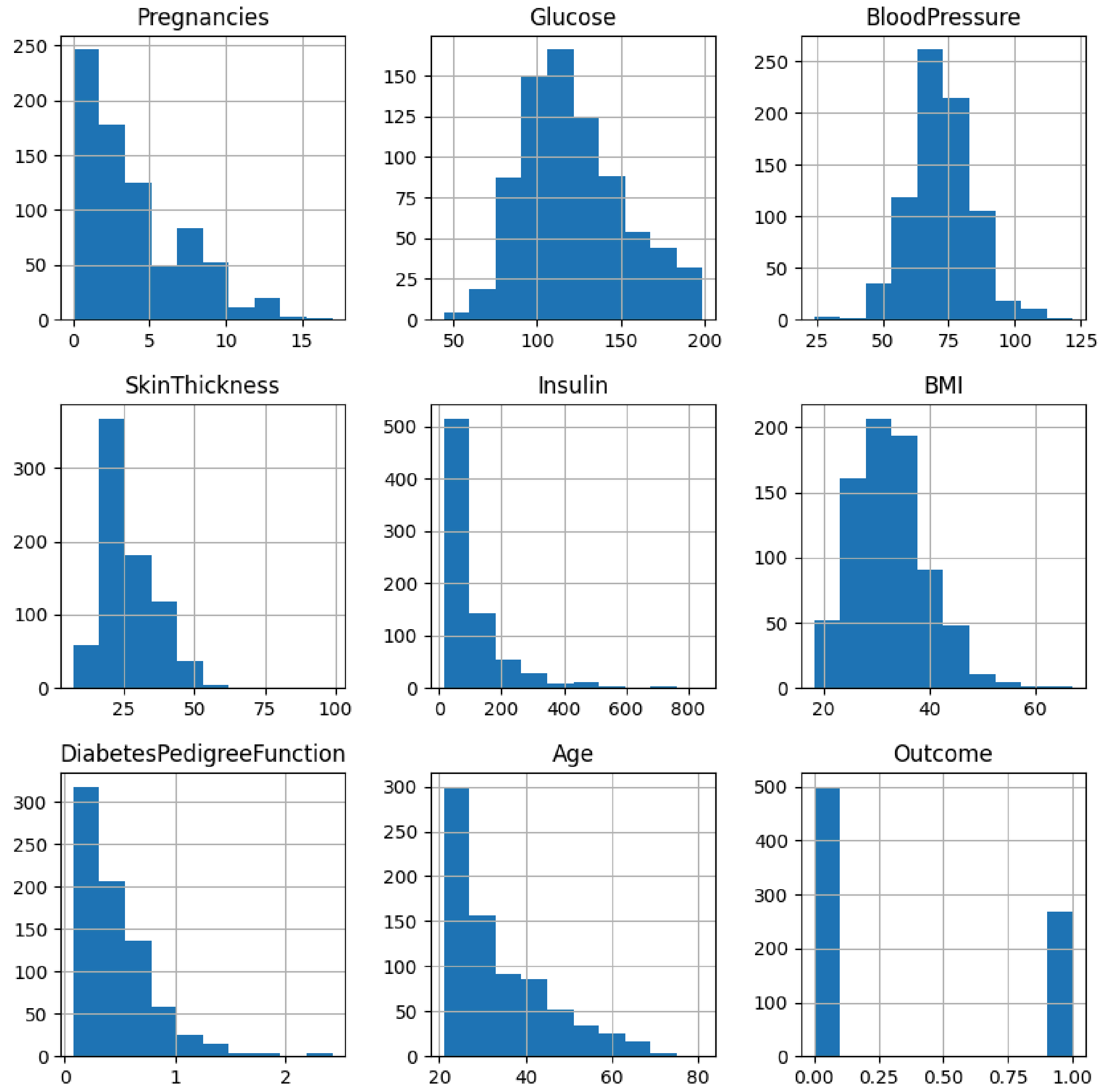
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	76
mean	3.845052	121.681605	72.254807	26.606479	118.660163	32.450805	0.471876	33.240885	
std	3.369578	30.436016	12.115932	9.631241	93.080358	6.875374	0.331329	11.760232	
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	
25%	1.000000	99.750000	64.000000	20.536458	79.799479	27.500000	0.243750	24.000000	
50%	3.000000	117.000000	72.000000	23.000000	79.799479	32.000000	0.372500	29.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	

```
#count plot
f,ax=plt.subplots(1,2,figsize=(10,5))
df['Outcome'].value_counts().plot.pie(explode=[0,0.1],autopct='%1.1f%%',ax=ax[0],shadow=True)
ax[0].set_title('Outcome')
ax[0].set_ylabel('')
sns.countplot(x='Outcome',data=df,ax=ax[1]) # Added x= to specify the column name
ax[1].set_title('Outcome')
N,P = df['Outcome'].value_counts()
print('Negative (0): ',N)
print('Positive (1): ',P)
plt.grid()
plt.show()
```

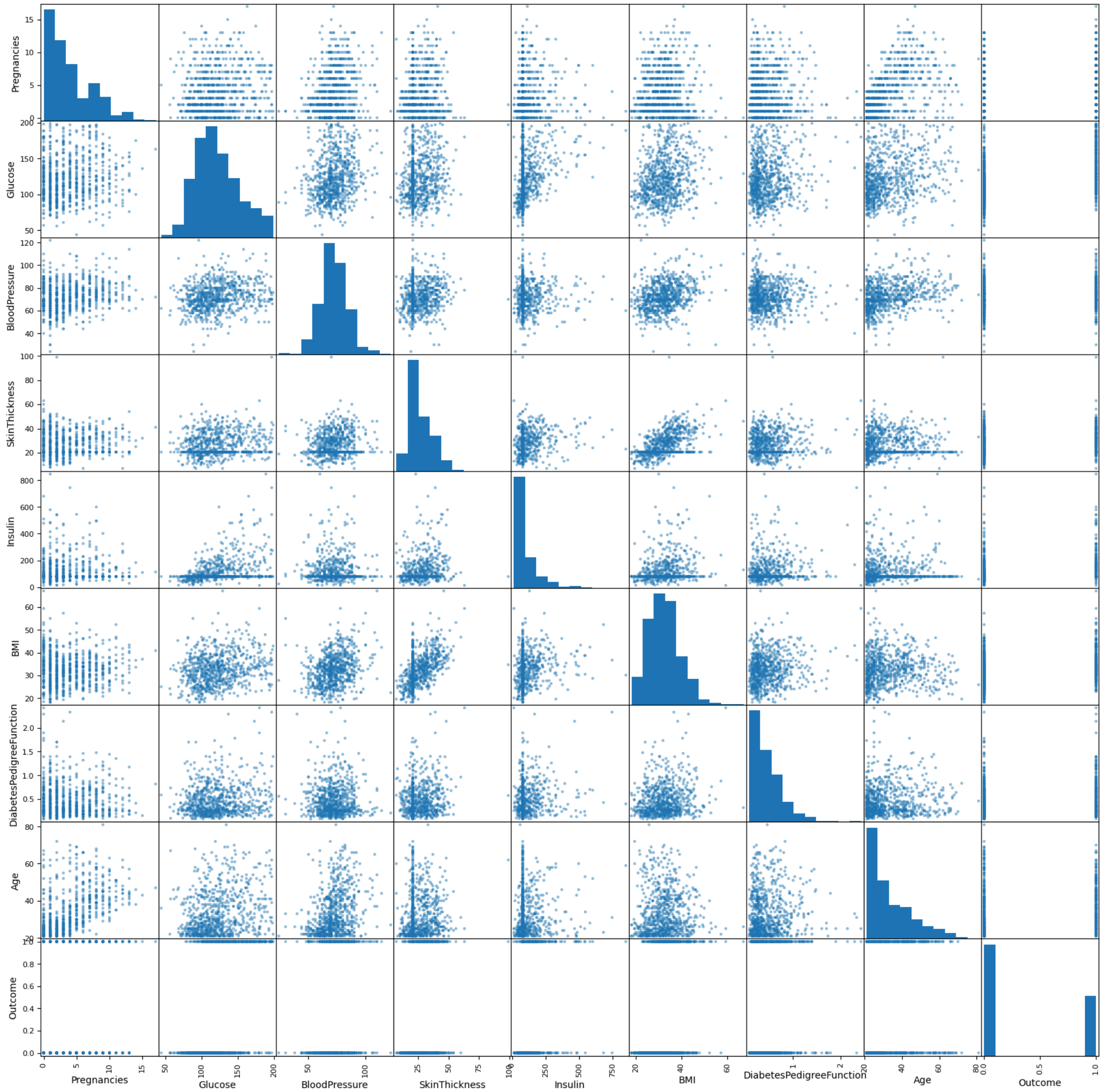
```
Negative (0):  500
Positive (1):  268
```



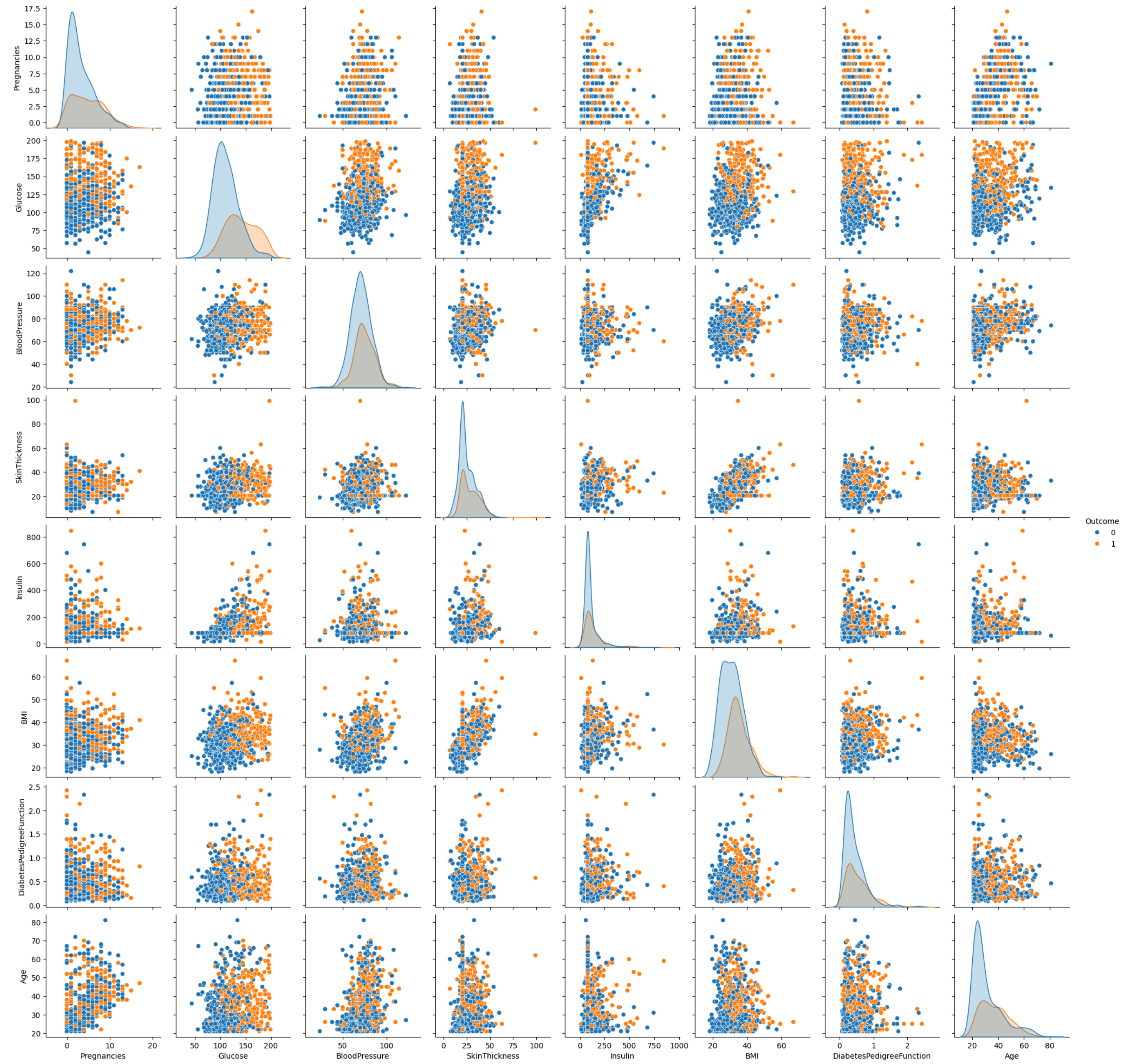

```
#Histogram of each feature
df.hist(bins=10,figsize=(10,10))
plt.show()
```



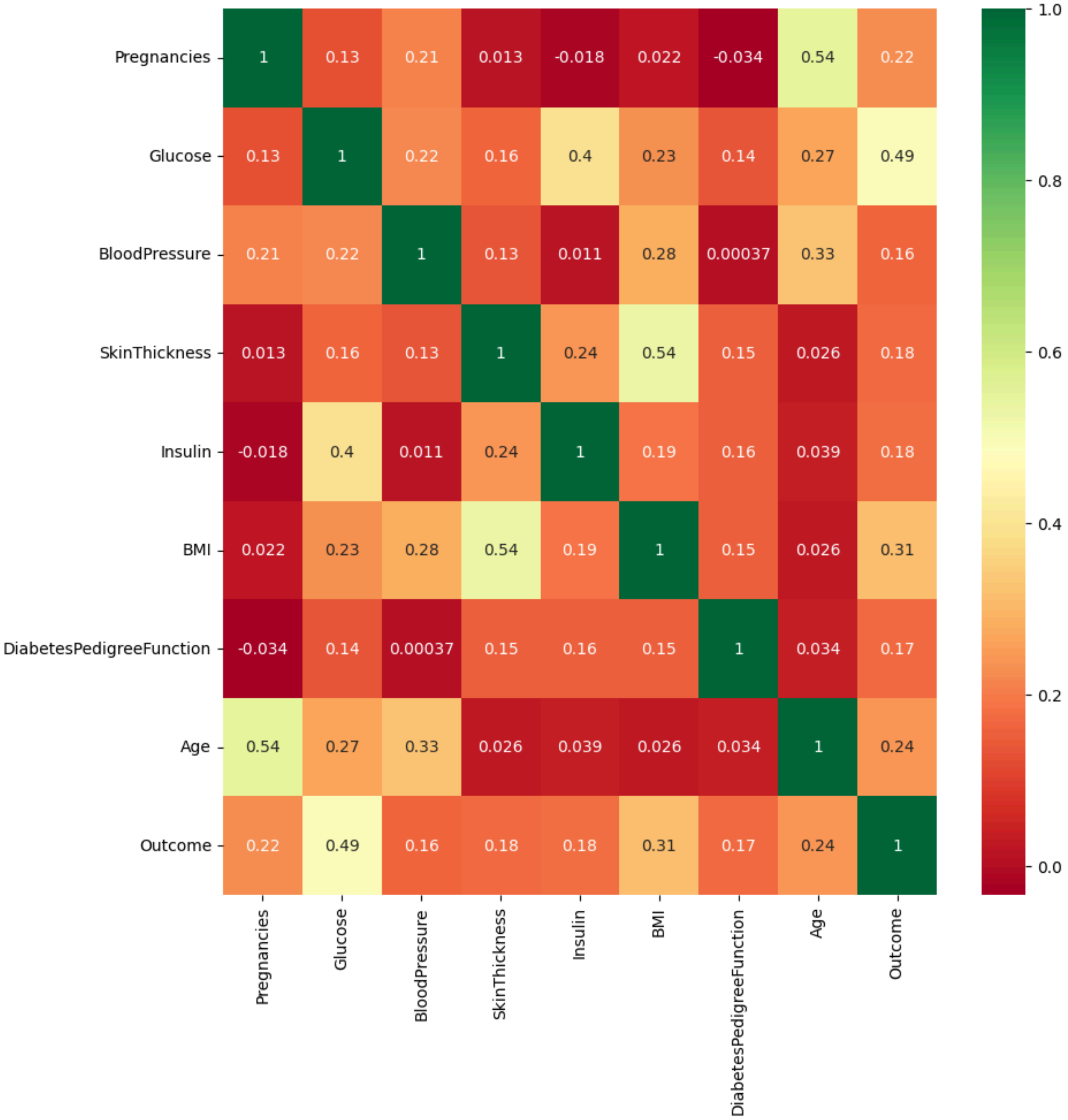

```
#Scatter plot matrix
from pandas.plotting import scatter_matrix
scatter_matrix(df, figsize=(20,20));
```




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




```
#Analyzing relationships between variables
#correlation analysis
import seaborn as sns
corrmat=df.corr()
top_corr_features=corrmat.index
plt.figure(figsize=(10,10))
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```




```
#splitting the data frame into X and Y
target_name='Outcome'
y=df[target_name]
x=df.drop(target_name,axis=1)
```

```
x.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	50
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	31
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	32
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	21
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	33

Next steps: Generate code with x View recommended plots New interactive sheet

```
y.head()
```

	Outcome
0	1
1	0
2	1
3	0
4	1

dtype: int64

```
#Applying Feature Scalling
#feature scalling techniques are of 4 types
#Standard Scaler
#Normalizer
#minmax scaler
#binarizer
#applying standard scaler
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(x)
SSX=scaler.transform(x)
```

```
#dividing data into trainging data which is 80% and testing data which is 20%
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(SSX,y,test_size=0.2,random_state=7)
```

```
x_train.shape,y_train.shape
```

```
((614, 8), (614,))
```

```
x_test.shape,y_test.shape
```

```
((154, 8), (154,))
```

```
#Buliding the algorithms
#LOGISTIC REGRESSION
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression(solver='liblinear',multi_class='ovr')
lr.fit(x_train,y_train)
```

```
LogisticRegression
LogisticRegression(multi_class='ovr', solver='liblinear')
```

```
#KNeighborsClassifer(KNN)
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()
knn.fit(x_train,y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

```
#Naive-Bayes Classifier
from sklearn.naive_bayes import GaussianNB
nb=GaussianNB()
nb.fit(x_train,y_train)
```

```
GaussianNB
GaussianNB()
```

```
#Support Vector Machine(SVM)
from sklearn.svm import SVC
sv=SVC()
sv.fit(x_train,y_train)
```

```
SVC
SVC()
```

```
#Decision tree
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dt.fit(x_train,y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

```
#Random Forest
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(criterion='entropy')
rf.fit(x_train,y_train)
```

```
RandomForestClassifier
RandomForestClassifier(criterion='entropy')
```

```
#Making Prediction
#making predictions by using logistic regression
lr_pred=lr.predict(x_test)
#making predictions by using KNN
knn_pred=knn.predict(x_test)
#making predictions by using Naive Bayes
nb_pred=nb.predict(x_test)
#making predictions by using SVM
sv_pred=sv.predict(x_test)
#making predictions by using Decision tree
dt_pred=dt.predict(x_test)
#making predictions by using Random Forest
rf_pred=rf.predict(x_test)
```

```
#Model Evaluation
#train score and test score of Logistic Regression
from sklearn.metrics import accuracy_score
print("Train Accuracy of Logistic Regression",lr.score(x_train,y_train)*100)
print("Accuracy test score of Logistic Regression",lr.score(x_test,y_test)*100)
print("Accuracy score of Logistic Regression",accuracy_score(y_test,lr_pred)*100)
```

```
Train Accuracy of Logistic Regression 77.36156351791531
Accuracy test score of Logistic Regression 77.27272727272727
Accuracy score of Logistic Regression 77.27272727272727
```

```
#train score and test score of KNN
print("Train Accuracy of KNN",knn.score(x_train,y_train)*100)
print("Accuracy test score of KNN",knn.score(x_test,y_test)*100)
print("Accuracy score of KNN",accuracy_score(y_test,knn_pred)*100)
```

```
Train Accuracy of KNN 81.10749185667753
Accuracy test score of KNN 74.67532467532467
Accuracy score of KNN 74.67532467532467
```

```
#train score and test score of Naive-Bayes
print("Train Accuracy of Naive Bayes",nb.score(x_train,y_train)*100)
print("Accuracy test score of Naive Bayes",nb.score(x_test,y_test)*100)
print("Accuracy score of Naive Bayes",accuracy_score(y_test,nb_pred)*100)

#train score and test score of SVM
print("Train Accuracy of SVM",sv.score(x_train,y_train)*100)
print("Accuracy test score of SVM",sv.score(x_test,y_test)*100)
print("Accuracy score of SVM",accuracy_score(y_test,sv_pred)*100)

#train score and test score of Decision Tree
print("Train Accuracy of Decision Tree",dt.score(x_train,y_train)*100)
print("Accuracy test score of Decision Tree",dt.score(x_test,y_test)*100)
print("Accuracy score of Decision Tree",accuracy_score(y_test,dt_pred)*100)

#train score and test score of RandomForest
print("Train Accuracy of Random Forest",rf.score(x_train,y_train)*100)
```

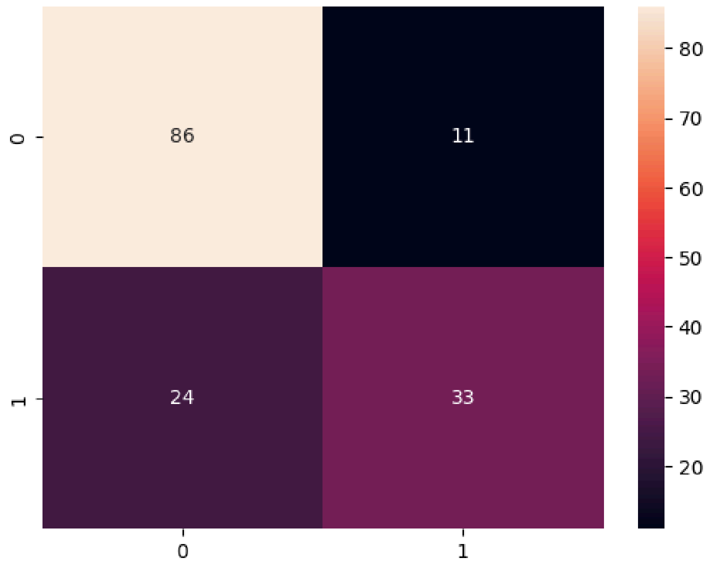
```
print("Accuracy test score of Random Forest",rf.score(x_test,y_test)*100)
print("Accuracy score of Random Forest",accuracy_score(y_test,rf_pred)*100)
```

```
Train Accuracy of Naive Bayes 74.2671009771987
Accuracy test score of Naive Bayes 74.02597402597402
Accuracy score of Naive Bayes 74.02597402597402
Train Accuracy of SVM 81.92182410423453
Accuracy test score of SVM 83.11688311688312
Accuracy score of SVM 83.11688311688312
Train Accuracy of Decision Tree 100.0
Accuracy test score of Decision Tree 76.62337662337663
Accuracy score of Decision Tree 76.62337662337663
Train Accuracy of Random Forest 100.0
Accuracy test score of Random Forest 81.81818181818183
Accuracy score of Random Forest 81.81818181818183
```

```
#Confusion matrix of logistic regression
from sklearn.metrics import classification_report,confusion_matrix
cm=confusion_matrix(y_test,lr_pred)
```

```
sns.heatmap(confusion_matrix(y_test,lr_pred),annot=True,fmt="d")
```

```
<Axes: >
```




```
print('Classification Report of Logistic Regression: \n',classification_report(y_test,lr_pred,digits=4))
```

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

           0     0.7818    0.8866    0.8309        97
           1     0.7500    0.5789    0.6535        57

    accuracy                         0.7727       154
   macro avg     0.7659    0.7328    0.7422       154
weighted avg     0.7700    0.7727    0.7652       154
```

```
TN=cm[0,0]
FP=cm[0,1]
FN=cm[1,0]
TP=cm[1,1]
```

```
TN,FP,FN,TP
```

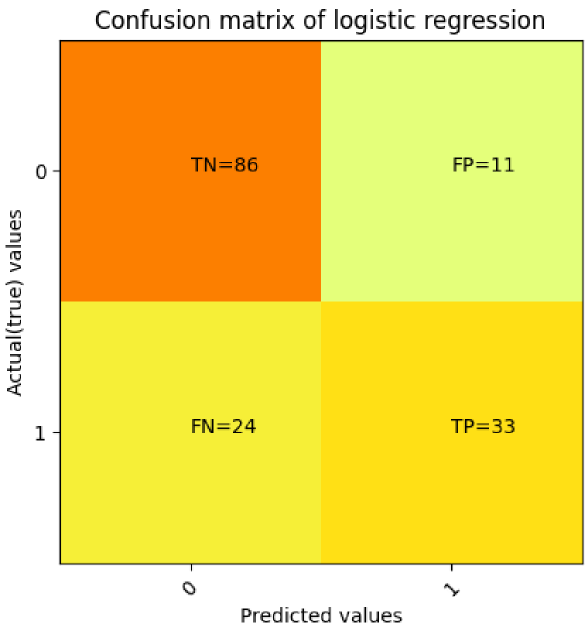
```
(86, 11, 24, 33)
```

```
#MAKING CONFUSION MATRIX OF LOGISTIC REGRESSION
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import accuracy_score,roc_auc_score,roc_curve
cm=confusion_matrix(y_test,lr_pred)

print('TN - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
print('FN - False Negative {}'.format(cm[1,0]))
```

```
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}' .format(np.divide(np.sum([cm[0,0],cm[1,1]]),np.sum(cm))*100))
print('Misclassification Rate: {}' .format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm))*100))
```

```
TN - True Negative 86
FP - False Positive 11
FN - False Negative 24
TP - True Positive 33
Accuracy Rate: 77.27272727272727
Misclassification Rate: 22.727272727272727
```

```
import matplotlib.pyplot as plt
plt.clf()
plt.imshow(cm,interpolation='nearest',cmap=plt.cm.Wistia)
classNames=['0','1']
plt.title('Confusion matrix of logistic regression')
plt.ylabel('Actual(true) values')
plt.xlabel('Predicted values')
tick_marks=np.arange(len(classNames))
plt.xticks(tick_marks,classNames,rotation=45)
plt.yticks(tick_marks,classNames)
s=[['TN','FP'],['FN','TP']]
for i in range(2):
    for j in range(2):
        plt.text(j,i,str(s[i][j])+"="+str(cm[i][j]))
plt.show()
```




```
pd.crosstab(y_test,lr_pred,margins=False)
```

col_0	0	1
Outcome		
0	86	11
1	24	33

```
pd.crosstab(y_test,lr_pred,margins=True)
```

col_0	0	1	All
Outcome			
0	86	11	97
1	24	33	57
All	110	44	154

```
pd.crosstab(y_test,lr_pred,rownames=['Actual values'], colnames=['Predicted values'],margins=True)
```

Predicted values	0	1	All
Actual values			
0	86	11	97
1	24	33	57
All	110	44	154

```
#Precision
TP,FP
```

```
(33, 11)
```

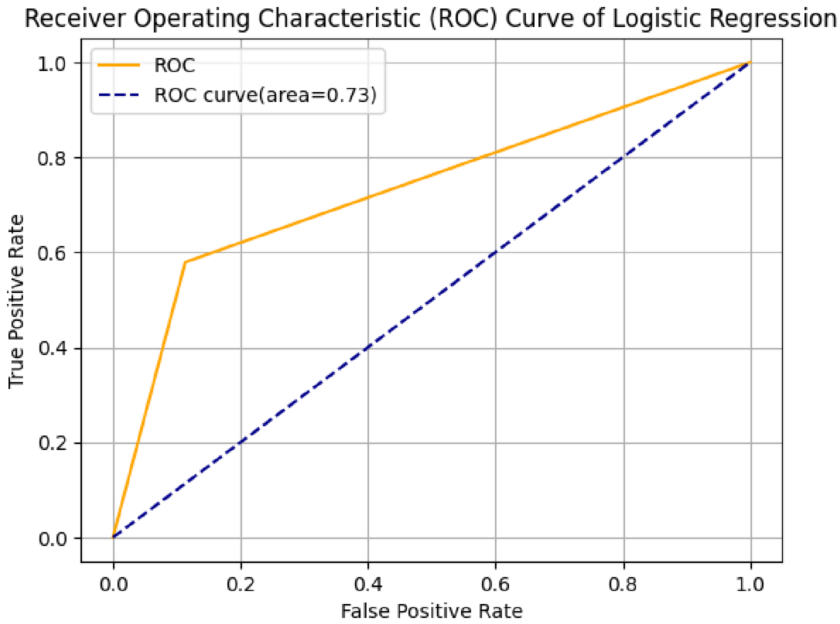
```
Precision=TP/(TP+FP)
Precision
```

```
0.75
```

```
#print precision score
precision_score=TP/float(TP+FP)*100
print('Precision score: {0:0.4f}' .format(precision_score))

from sklearn.metrics import precision_score
print("precision score is:", precision_score(y_test,lr_pred)*100)

#F1 score
from sklearn.metrics import f1_score
print('f1_score:',f1_score(y_test,lr_pred)*100)
f1_score: 65.34653465346535
#false positive rate(fpr)
FPR=FP/float(FP+TN)*100
print('False Positive Rate : {0:0.4f}' .format(FPR))

#Specificity
specificity=TN/(TN+FP)*100
print('Specificity : {0:0.4f}' .format(specificity))

#ROC Curve & ROC AUC
#Area under curve
auc= roc_auc_score(y_test,lr_pred)
print("ROC AUC_SCORE of Logistic Regression is",auc)
```

```
Precision score: 75.0000
precision score is: 75.0
f1_score: 65.34653465346535
False Positive Rate : 11.3402
Specificity : 88.6598
ROC AUC_SCORE of Logistic Regression is 0.7327726532826913
```

```
fpr,tpr,thresholds=roc_curve(y_test,lr_pred)
plt.plot(fpr,tpr,color='orange',label='ROC')
plt.plot([0,1],[0,1],color='darkblue',linestyle='--',label='ROC curve(area=%0.2f)'%auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve of Logistic Regression')
plt.legend()
plt.grid()
plt.show()
```




```
#confusion matrix of KNN
#MAKING CONFUSION MATRIX OF KNN
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import accuracy_score,roc_auc_score,roc_curve
cm=confusion_matrix(y_test,knn_pred)

print('TN - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}' .format(np.divide(np.sum([cm[0,0],cm[1,1]]),np.sum(cm))*100))
print('Misclassification Rate: {}' .format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm))*100))
```

```
TN - True Negative 82
FP - False Positive 15
FN - False Negative 24
TP - True Positive 33
Accuracy Rate: 74.67532467532467
Misclassification Rate: 25.324675324675322
```

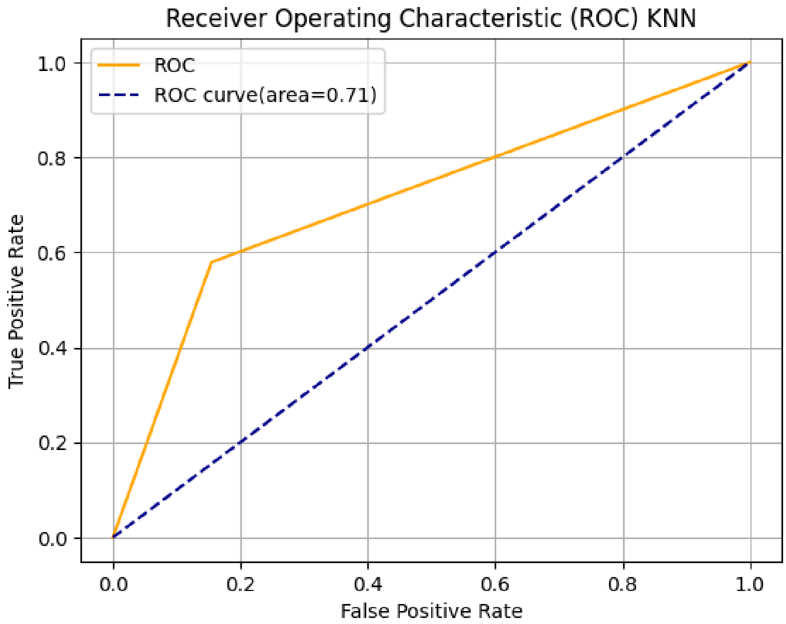
```
#classification report of KNN
print('Classification Report of KNN: \n', classification_report(y_test,knn_pred,digits=5))
```

```
Classification Report of KNN:
              precision    recall  f1-score   support

           0    0.77358   0.84536   0.80788        97
           1    0.68750   0.57895   0.62857        57

    accuracy                        0.74675       154
   macro avg    0.73054   0.71215   0.71823       154
weighted avg    0.74172   0.74675   0.74151       154
```

```
#Area under curve of KNN
auc=roc_auc_score(y_test,knn_pred)
print("ROC AUC SCORE of KNN is",auc)
```

```
ROC AUC SCORE of KNN is 0.7121540965816603
```

```
fpr,tpr,thresholds=roc_curve(y_test,knn_pred)
plt.plot(fpr,tpr,color='orange',label='ROC')
plt.plot([0,1],[0,1],color='darkblue',linestyle='--',label='ROC curve(area=%0.2f)'%auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) KNN')
plt.legend()
plt.grid()
plt.show()
```

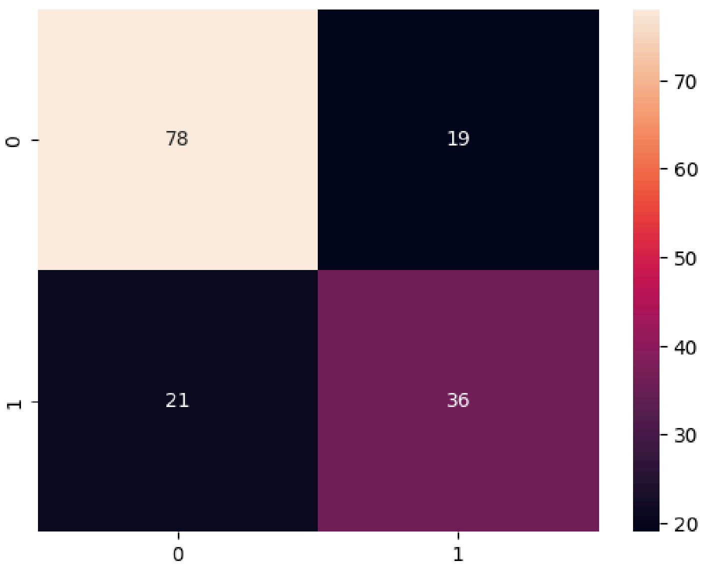



```
#confusion matrix of Naive Bayes
#MAKING CONFUSION MATRIX OF Naive Bayes
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import accuracy_score,roc_auc_score,roc_curve
cm=confusion_matrix(y_test,nb_pred)

print('TN - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}' .format(np.divide(np.sum([cm[0,0],cm[1,1]]),np.sum(cm))*100))
print('Misclassification Rate: {}' .format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm))*100))
```

```
TN - True Negative 78
FP - False Positive 19
FN - False Negative 21
TP - True Positive 36
Accuracy Rate: 74.02597402597402
Misclassification Rate: 25.97402597402597
```

```
sns.heatmap(confusion_matrix(y_test,nb_pred),annot=True,fmt="d")
```

```
<Axes: >
```




```
#classification report of Naive Bayes
print('Classification Report of Naive Bayes: \n', classification_report(y_test,nb_pred,digits=5))
```

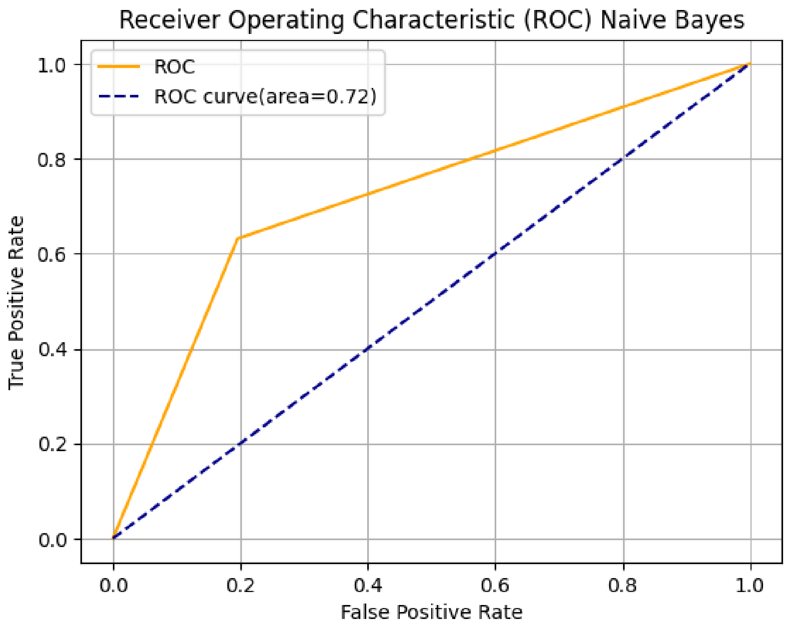
```
Classification Report of Naive Bayes:
              precision    recall  f1-score   support
```

```
           0     0.78788   0.80412   0.79592        97
           1     0.65455   0.63158   0.64286        57

    accuracy                         0.74026       154
   macro avg     0.72121   0.71785   0.71939       154
weighted avg     0.73853   0.74026   0.73927       154
```

```
#Area under curve of Naive Bayes
auc=roc_auc_score(y_test,nb_pred)
print("ROC AUC SCORE of Naive Bayes is",auc)
```

```
ROC AUC SCORE of Naive Bayes is 0.7178513293543136
```

```
fpr,tpr,thresholds=roc_curve(y_test,nb_pred)
plt.plot(fpr,tpr,color='orange',label='ROC')
plt.plot([0,1],[0,1],color='darkblue',linestyle='--',label='ROC curve(area=%0.2f)'%auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Naive Bayes')
plt.legend()
plt.grid()
plt.show()
```

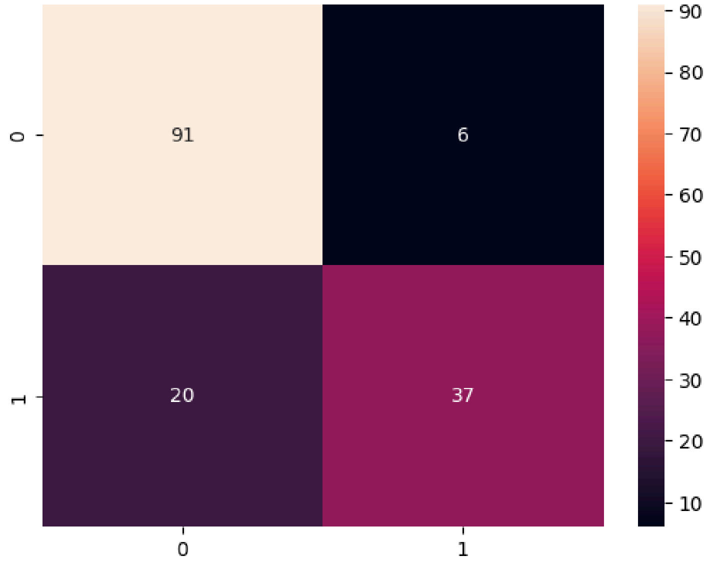



```
#confusion matrix of Support Vector Machine
#MAKING CONFUSION MATRIX OF Support Vector Machine
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import accuracy_score,roc_auc_score,roc_curve
cm=confusion_matrix(y_test,sv_pred)

print('TN - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}' .format(np.divide(np.sum([cm[0,0],cm[1,1]]),np.sum(cm))*100))
print('Misclassification Rate: {}' .format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm))*100))
```

```
TN - True Negative 91
FP - False Positive 6
FN - False Negative 20
TP - True Positive 37
Accuracy Rate: 83.11688311688312
Misclassification Rate: 16.883116883116884
```

```
sns.heatmap(confusion_matrix(y_test,sv_pred),annot=True,fmt="d")
```

```
<Axes: >
```



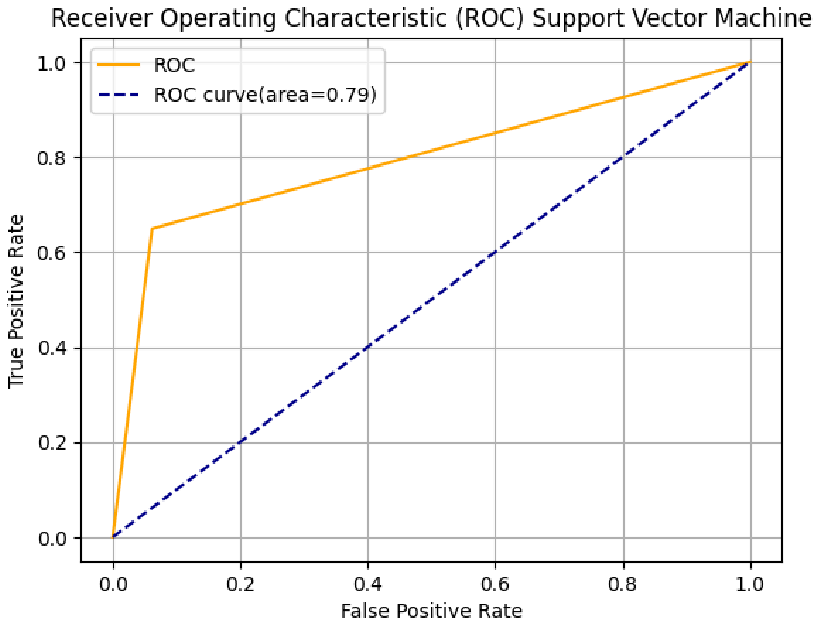

```
#classification report of Support Vector Machine
print('Classification Report of Support Vector Machine: \n', classification_report(y_test,sv_pred,digits=5))
```

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

           0    0.81982   0.93814   0.87500        97
           1    0.86047   0.64912   0.74000        57

    accuracy                        0.83117       154
   macro avg    0.84014   0.79363   0.80750       154
weighted avg    0.83486   0.83117   0.82503       154
```

```
#Area under curve of Support Vector Machine
auc=roc_auc_score(y_test,sv_pred)
print("ROC AUC SCORE of Support Vector Machine is",auc)
```

```
ROC AUC SCORE of Support Vector Machine is 0.7936335684572255
```

```
fpr,tpr,thresholds=roc_curve(y_test,sv_pred)
plt.plot(fpr,tpr,color='orange',label='ROC')
plt.plot([0,1],[0,1],color='darkblue',linestyle='--',label='ROC curve(area=%0.2f)'%auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Support Vector Machine')
plt.legend()
plt.grid()
plt.show()
```

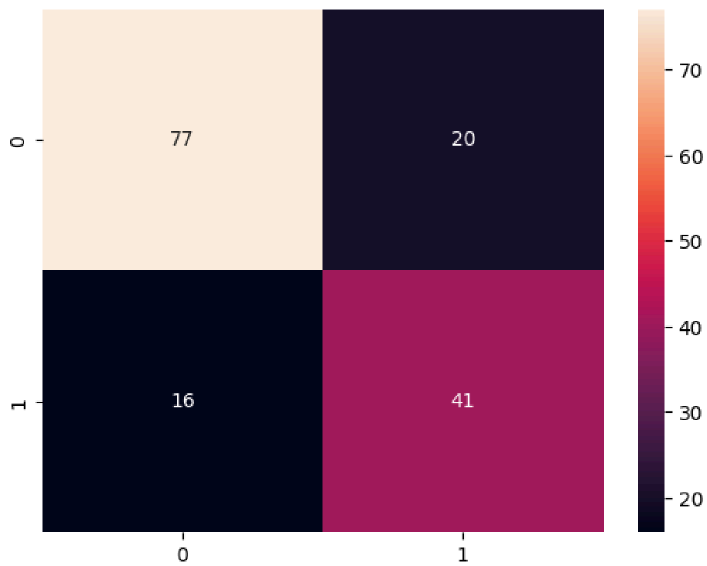



```
#confusion matrix of Decision tree
#MAKING CONFUSION MATRIX OF Decision tree
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import accuracy_score,roc_auc_score,roc_curve
cm=confusion_matrix(y_test,dt_pred)

print('TN - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}' .format(np.divide(np.sum([cm[0,0],cm[1,1]]),np.sum(cm))*100))
print('Misclassification Rate: {}' .format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm))*100))
```

```
TN - True Negative 77
FP - False Positive 20
FN - False Negative 16
TP - True Positive 41
Accuracy Rate: 76.62337662337663
Misclassification Rate: 23.376623376623375
```

```
sns.heatmap(confusion_matrix(y_test,dt_pred),annot=True,fmt="d")
```

```
<Axes: >
```




```
#classification report of Decision tree
print('Classification Report of Decision tree: \n', classification_report(y_test,dt_pred,digits=5))
```

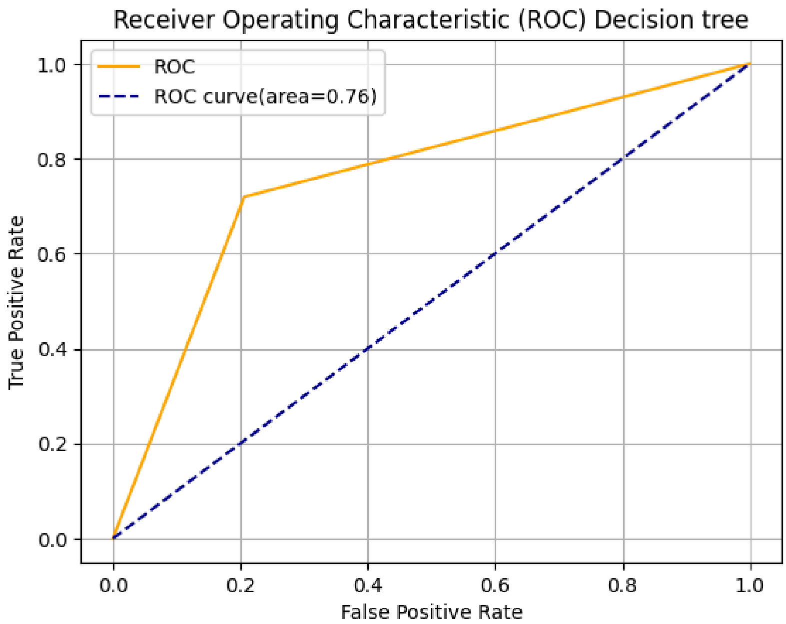
```
Classification Report of Decision tree:
              precision    recall  f1-score   support
```

```
              0     0.82796   0.79381   0.81053        97
              1     0.67213   0.71930   0.69492        57

       accuracy                         0.76623       154
      macro avg     0.75004   0.75656   0.75272       154
   weighted avg     0.77028   0.76623   0.76774       154
```

```
#Area under curve of Decision tree
auc=roc_auc_score(y_test,dt_pred)
print("ROC AUC SCORE of Decision tree is",auc)
```

```
ROC AUC SCORE of Decision tree is 0.7565563393018631
```

```
fpr,tpr,thresholds=roc_curve(y_test,dt_pred)
plt.plot(fpr,tpr,color='orange',label='ROC')
plt.plot([0,1],[0,1],color='darkblue',linestyle='--',label='ROC curve(area=%0.2f)'%auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Decision tree')
plt.legend()
plt.grid()
plt.show()
```

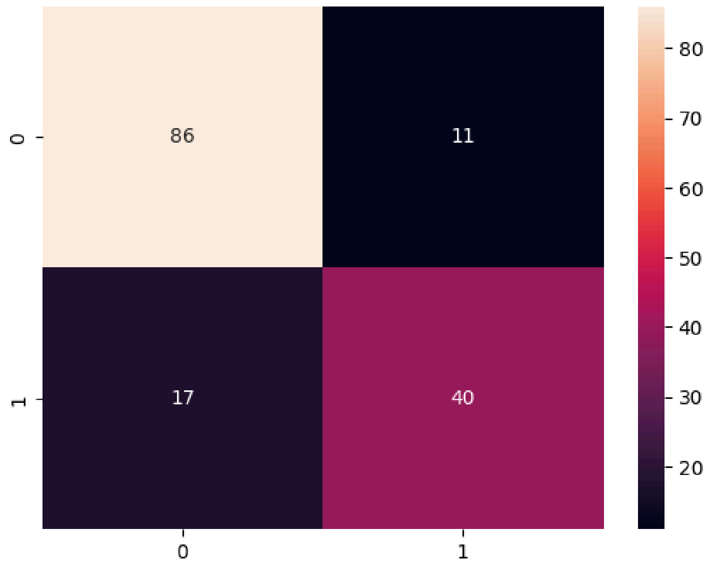



```
#confusion matrix of Random forest
#MAKING CONFUSION MATRIX OF Random forest
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import accuracy_score,roc_auc_score,roc_curve
cm=confusion_matrix(y_test,rf_pred)

print('TN - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}' .format(np.divide(np.sum([cm[0,0],cm[1,1]]),np.sum(cm))*100))
print('Misclassification Rate: {}' .format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm))*100))
```

```
TN - True Negative 86
FP - False Positive 11
FN - False Negative 17
TP - True Positive 40
Accuracy Rate: 81.81818181818183
Misclassification Rate: 18.181818181818183
```

```
sns.heatmap(confusion_matrix(y_test,rf_pred),annot=True,fmt="d")
```

```
<Axes: >
```




```
#classification report of Random forest
print('Classification Report of Random forest: \n', classification_report(y_test,rf_pred,digits=5))
```

```
Classification Report of Random forest: 
               precision    recall  f1-score   support

           0    0.83495   0.88660   0.86000        97
           1    0.78431   0.70175   0.74074        57

    accuracy                        0.81818       154
   macro avg    0.80963   0.79418   0.80037       154
weighted avg    0.81621   0.81818   0.81586       154
```

```
#Area under curve of Random forest
auc=roc_auc_score(y_test,rf_pred)
print("ROC AUC SCORE of Random forest is",auc)
```

```
ROC AUC SCORE of Random forest is 0.7941761620546209
```

```
fpr,tpr,thresholds=roc_curve(y_test,rf_pred)
plt.plot(fpr,tpr,color='orange',label='ROC')
plt.plot([0,1],[0,1],color='darkblue',linestyle='--',label='ROC curve(area=%0.2f)'%auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Random forest')
plt.legend()
plt.grid()
plt.show()
```

Receiver Operating Characteristic (ROC) Random forest

```
pip install streamlit
```

```
Collecting streamlit
  Downloading streamlit-1.39.0-py2.py3-none-any.whl.metadata (8.5 kB)
Requirement already satisfied: altair<6,>=4.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (4.2.2)
Requirement already satisfied: blinker<2,>=1.0.0 in /usr/lib/python3/dist-packages (from streamlit) (1.4)
Requirement already satisfied: cachetools<6,>=4.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (5.5.0)
Requirement already satisfied: click<9,>=7.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (8.1.7)
Requirement already satisfied: numpy<3,>=1.20 in /usr/local/lib/python3.10/dist-packages (from streamlit) (1.26.4)
Requirement already satisfied: packaging<25,>=20 in /usr/local/lib/python3.10/dist-packages (from streamlit) (24.1)
Requirement already satisfied: pandas<3,>=1.4.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (2.2.2)
Requirement already satisfied: pillow<11,>=7.1.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (10.4.0)
Requirement already satisfied: protobuf<6,>=3.20 in /usr/local/lib/python3.10/dist-packages (from streamlit) (3.20.3)
Requirement already satisfied: pyarrow>=7.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (16.1.0)
Requirement already satisfied: requests<3,>=2.27 in /usr/local/lib/python3.10/dist-packages (from streamlit) (2.32.3)
Requirement already satisfied: rich<14,>=10.14.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (13.8.1)
```