Data science capstone

HEALTHCARE

BY- MEET HARIYANI

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.

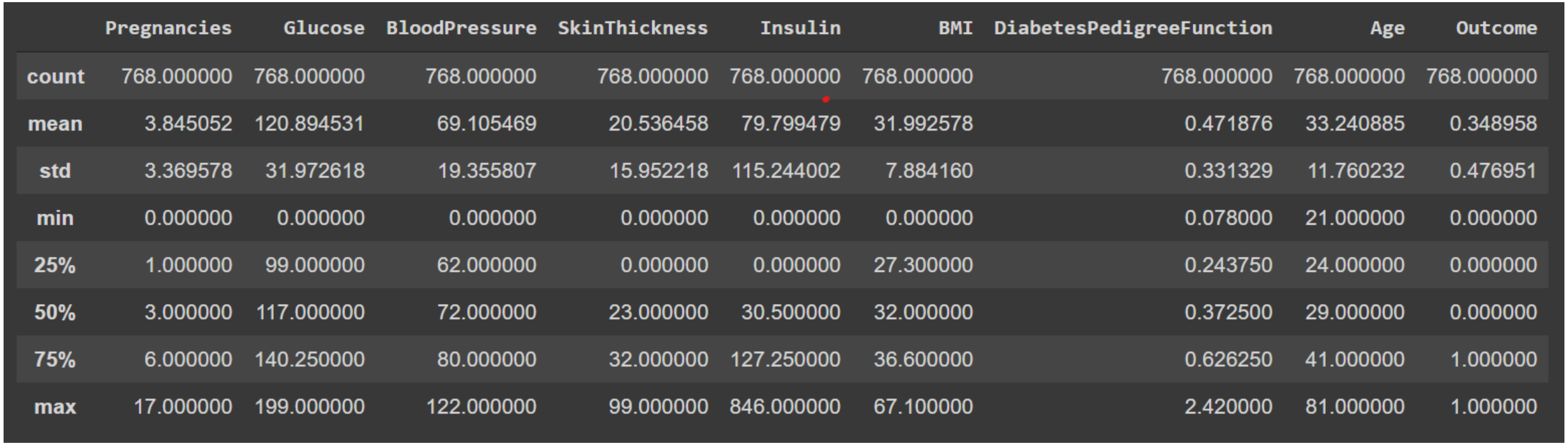
Dataset description

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

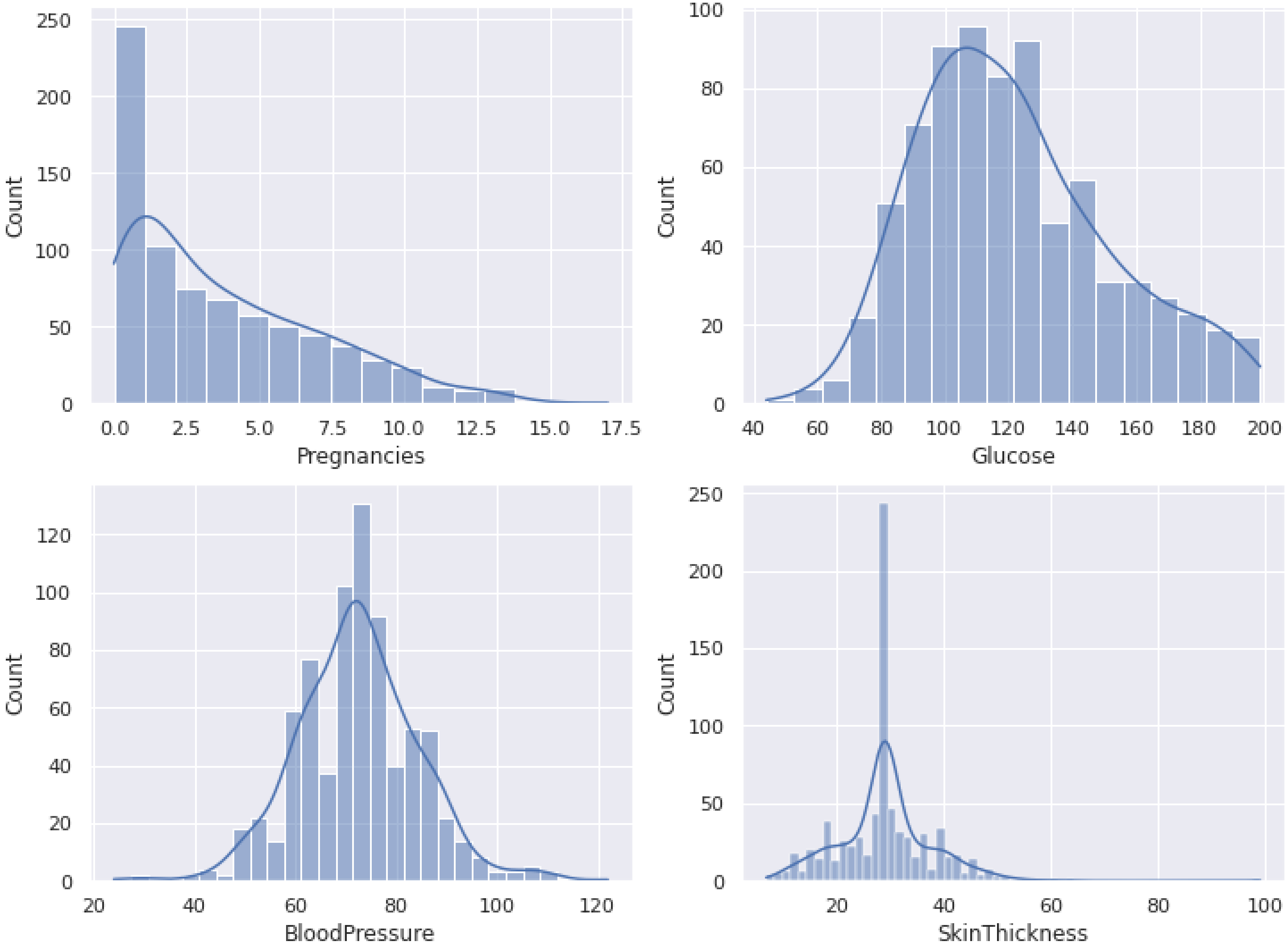
Project task: week-1

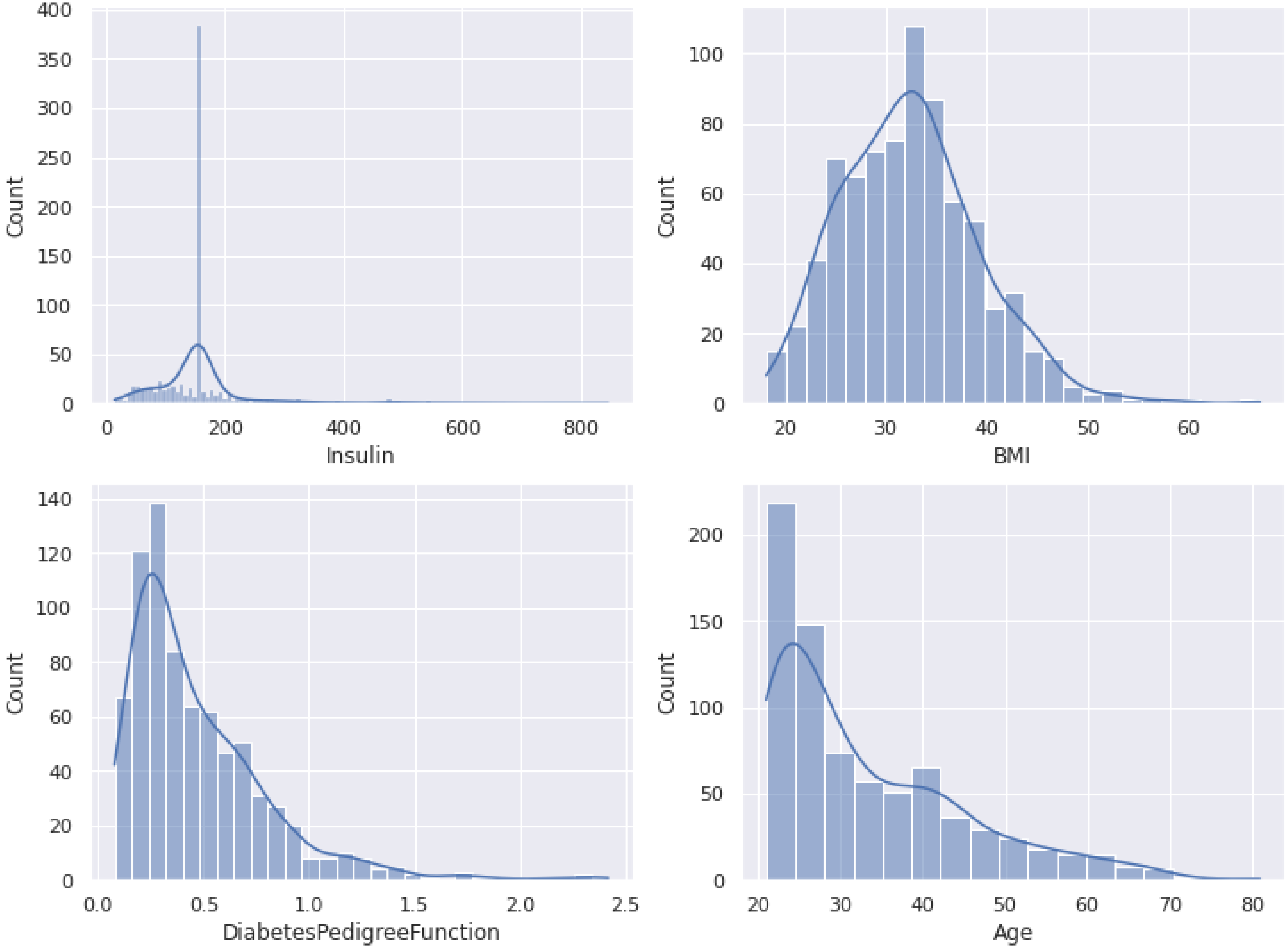
- Performed descriptive analysis on the dataset.



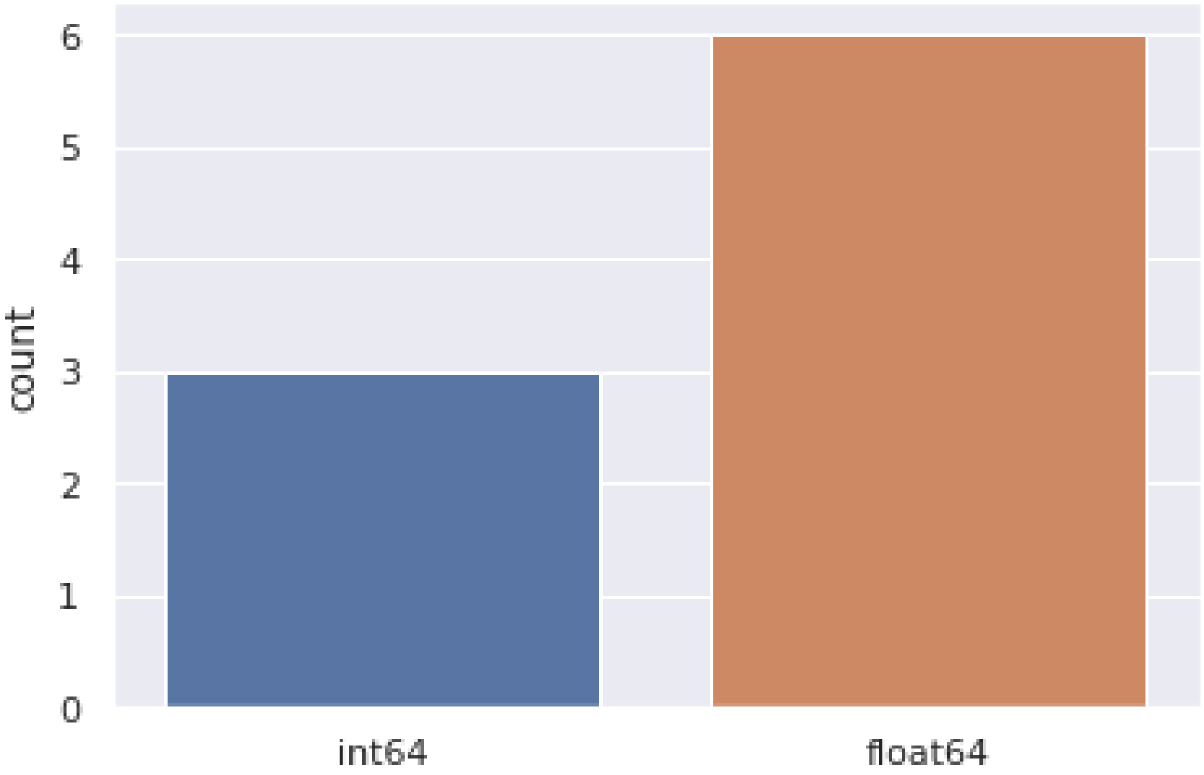
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

- Visually explored variables with histograms and replaced missing values with mean value of particular variable.



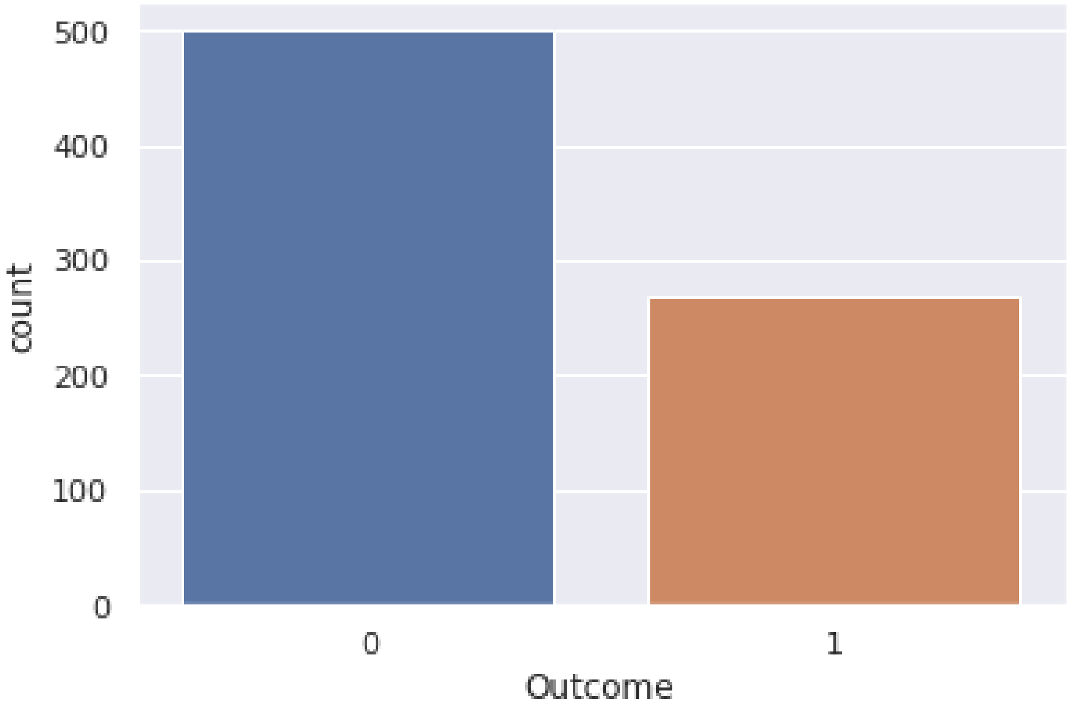

Count
400
350
300
250
200
150
100
50
0
0
200
400
600
800
Insulin
Count
100
80
60
40
20
0
20
30
40
50
60
BMI
Count
140
120
100
80
60
40
20
0
0.0
0.5
1.0
1.5
2.0
2.5
DiabetesPedigreeFunction
Count
200
150
100
50
0
20
30
40
50
60
70
80
Age

- Created a count (frequency) plot describing the data types and the count of variables.

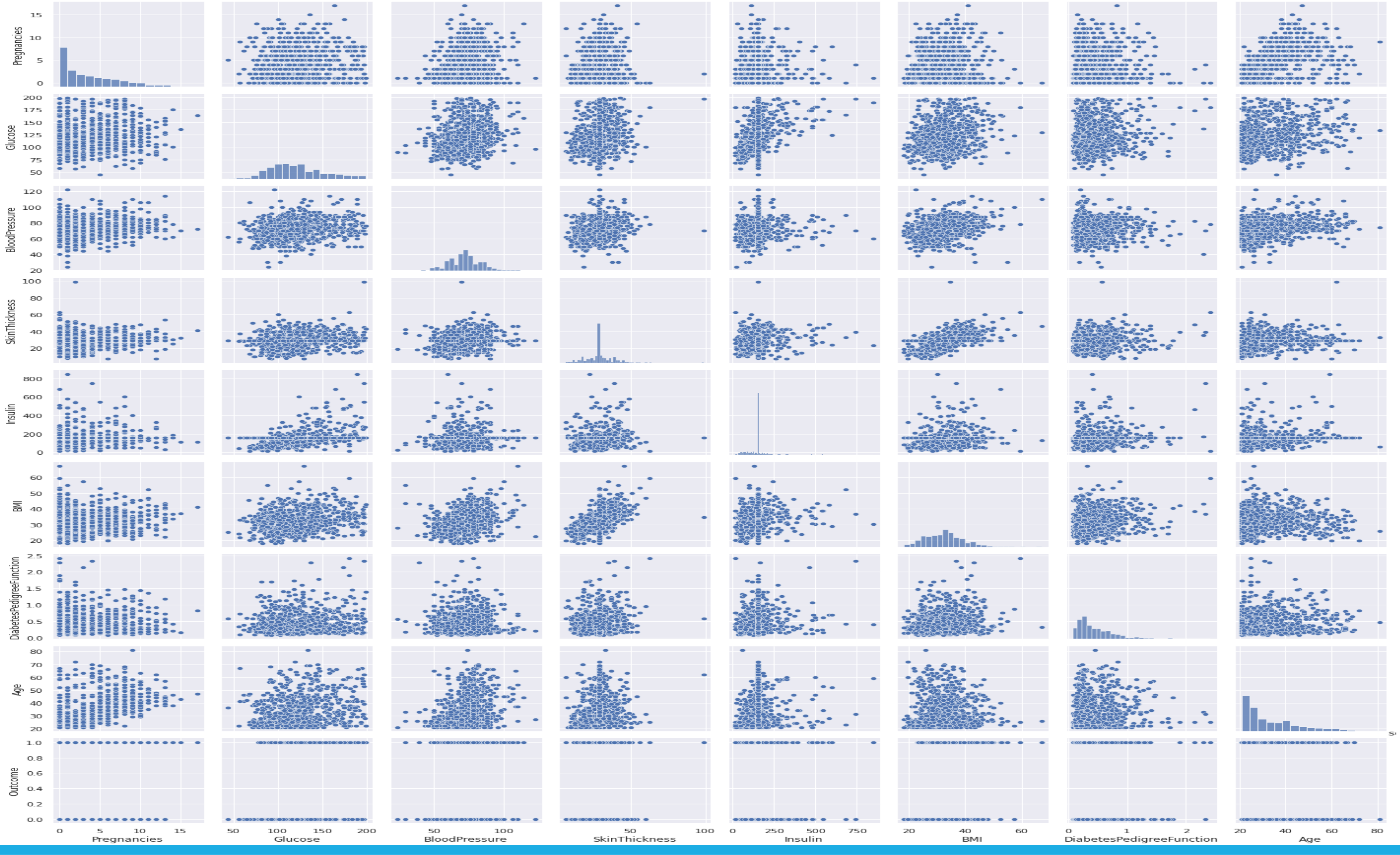


Project task: week-2

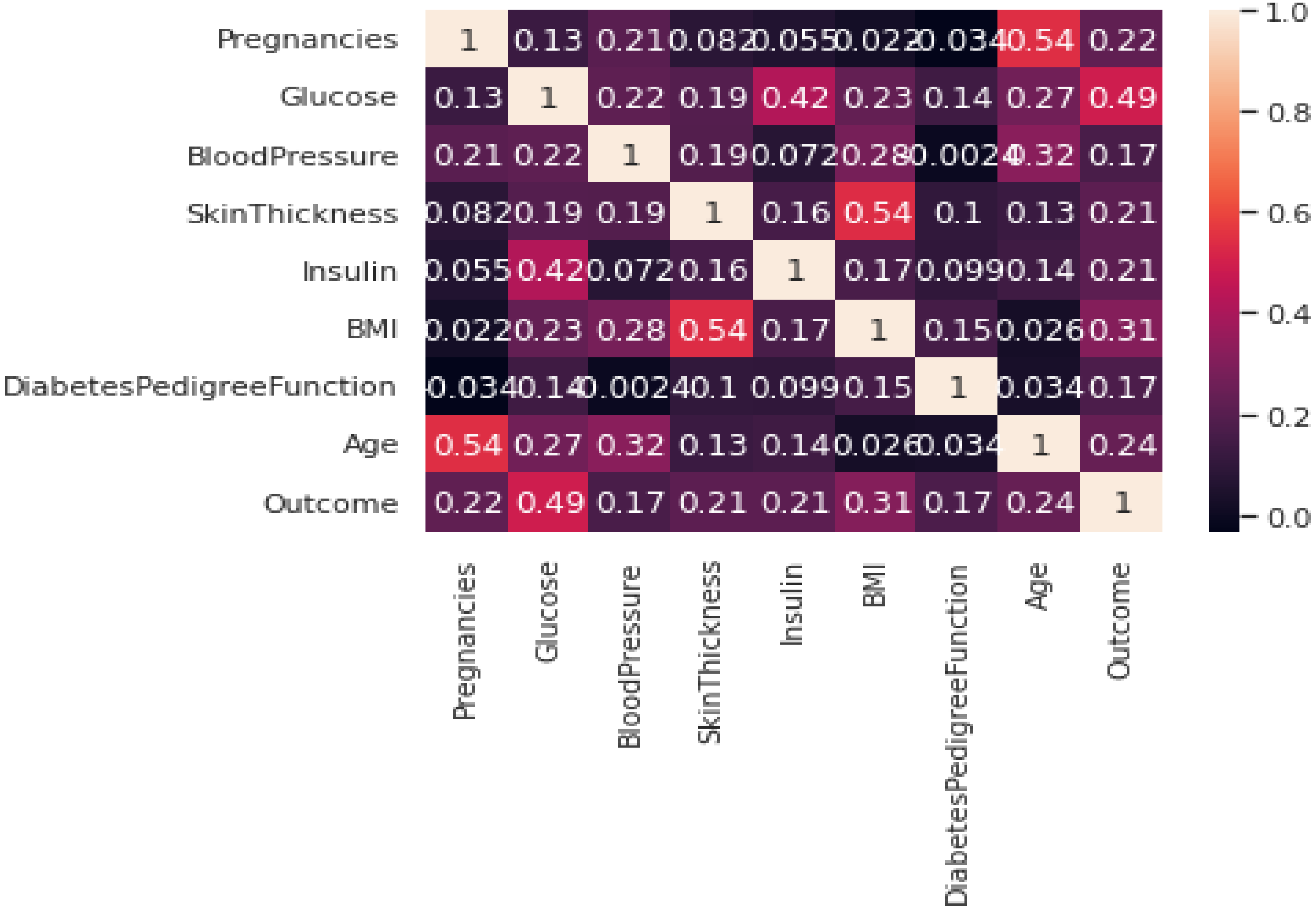
- Checked the balance of the data by plotting the count of outcomes by their value.



- Created scatter charts between the pair of variables to understand the relationships. Describe your findings.

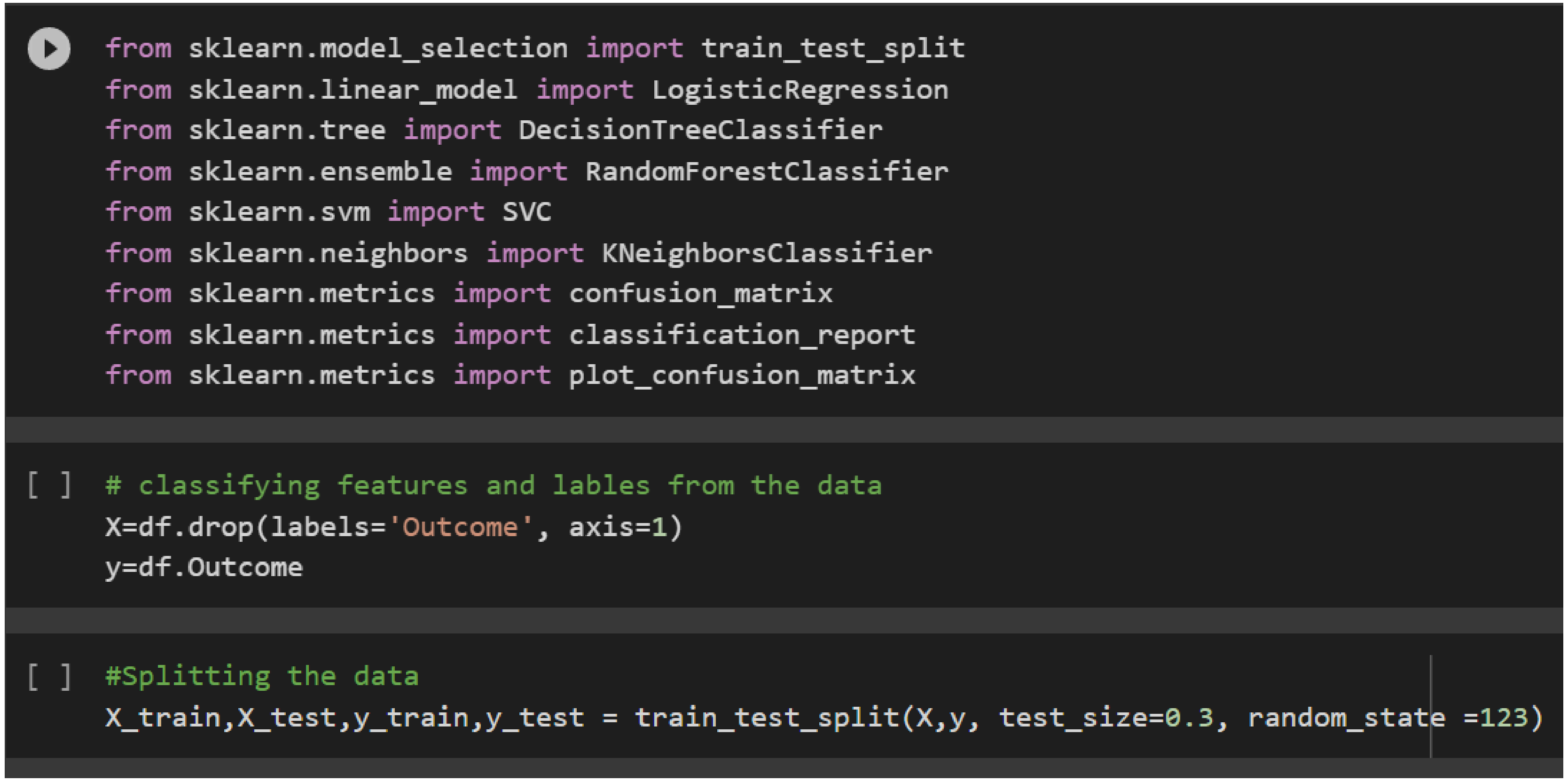


- Perform correlation analysis. Visually explore it using a heat map.



Project task: week-3

- model building – Importing libraries ,Preparing and splitting dataset for training and testing.

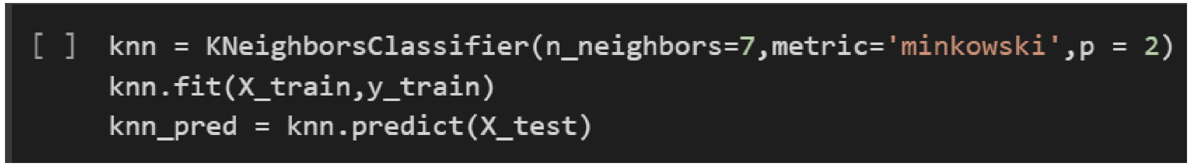


```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import plot_confusion_matrix
```

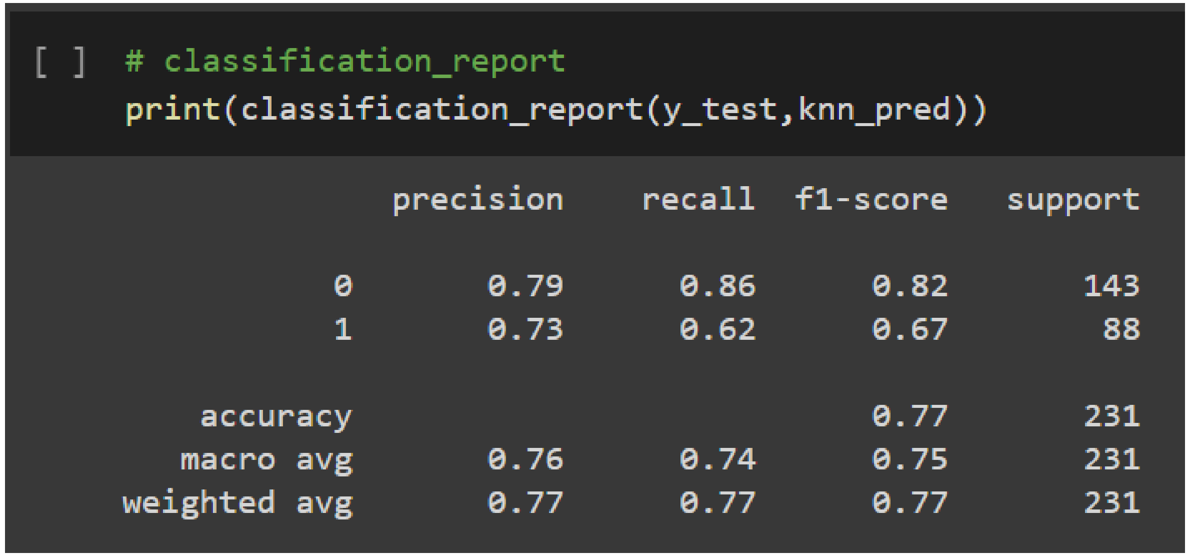
```
# classifying features and lables from the data
X=df.drop(labels='Outcome', axis=1)
y=df.Outcome
```

```
#Splitting the data
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.3, random_state =123)
```

- KNN Algorithm.

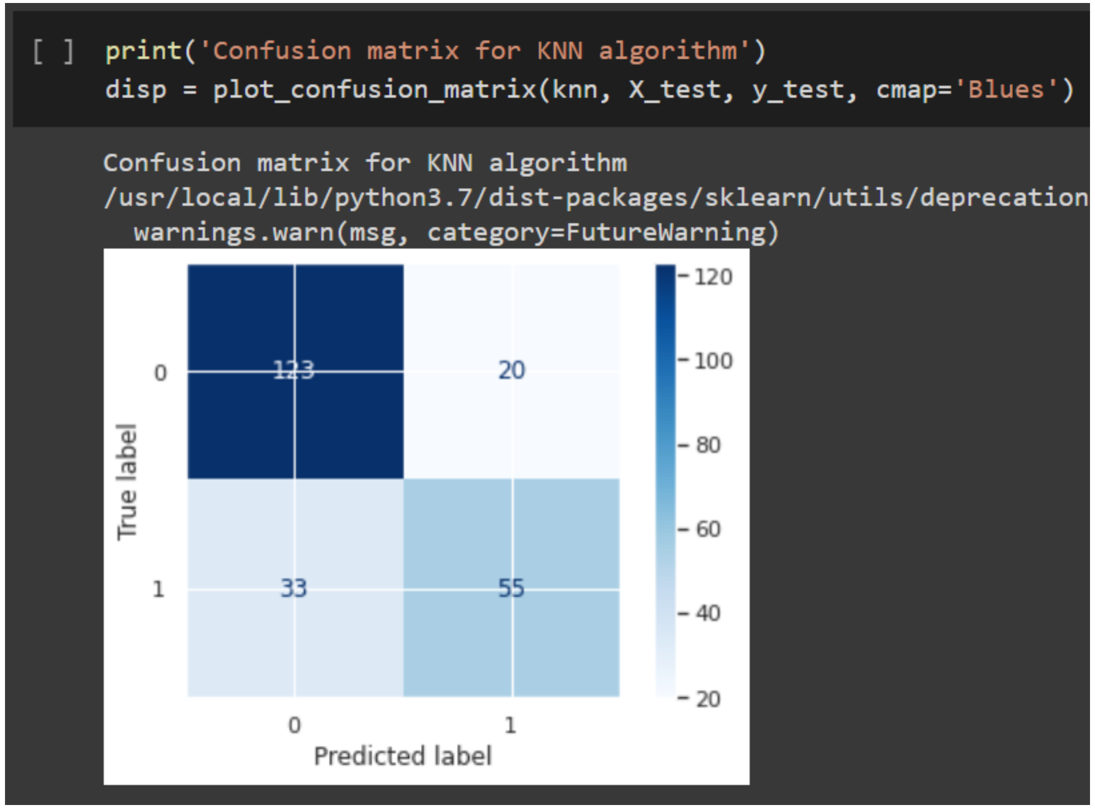


```
[ ]  knn = KNeighborsClassifier(n_neighbors=7,metric='minkowski',p = 2)
     knn.fit(X_train,y_train)
     knn_pred = knn.predict(X_test)
```



```
[ ]  # classification_report
     print(classification_report(y_test,knn_pred))
```

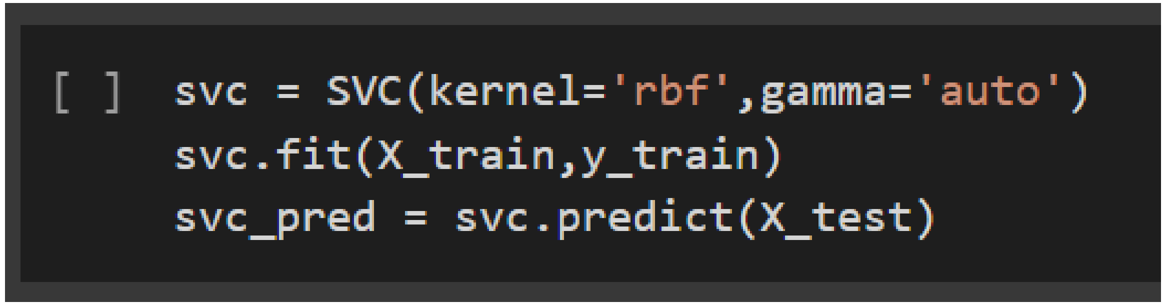
```
              precision    recall  f1-score   support

           0       0.79      0.86      0.82       143
           1       0.73      0.62      0.67        88

    accuracy                           0.77       231
   macro avg       0.76      0.74      0.75       231
weighted avg       0.77      0.77      0.77       231
```



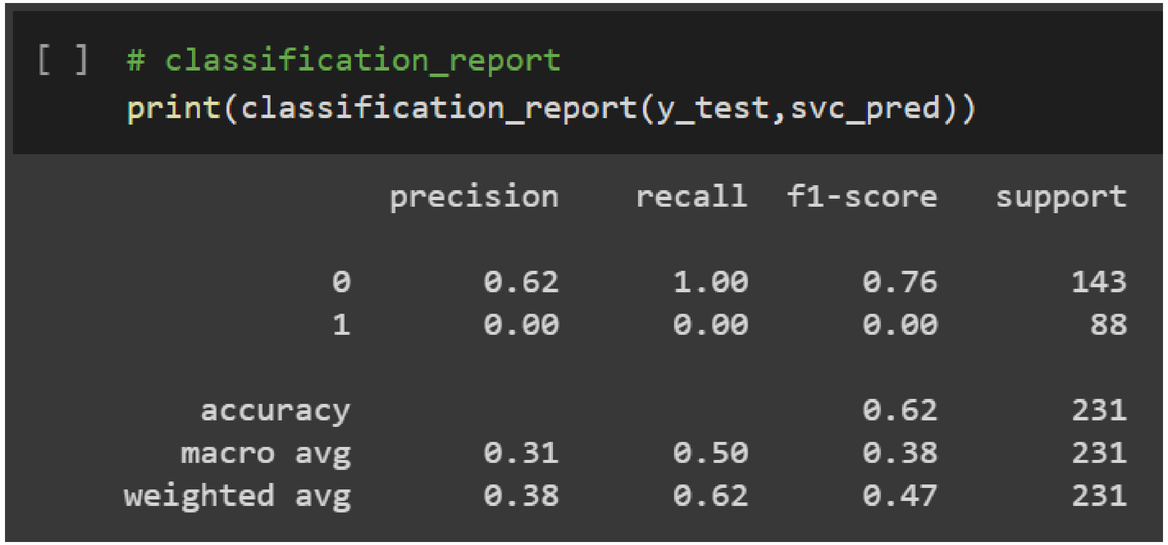
```
[ ]  print('Confusion matrix for KNN algorithm')
     disp = plot_confusion_matrix(knn, X_test, y_test, cmap='Blues')
```

```
Confusion matrix for KNN algorithm
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation
  warnings.warn(msg, category=FutureWarning)
```

- Support vector Classifier

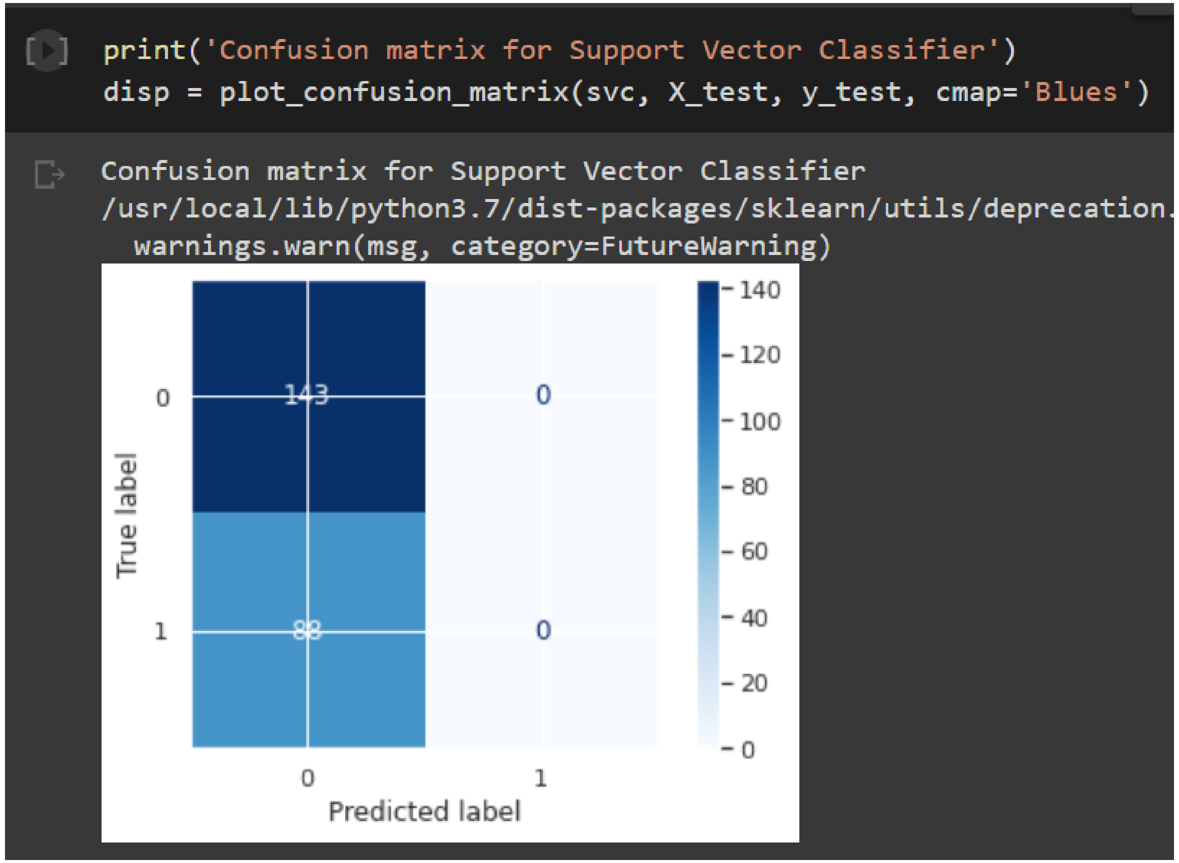


```
[ ]  svc = SVC(kernel='rbf',gamma='auto')
     svc.fit(X_train,y_train)
     svc_pred = svc.predict(X_test)
```



```
[ ]  # classification_report
     print(classification_report(y_test,svc_pred))
```

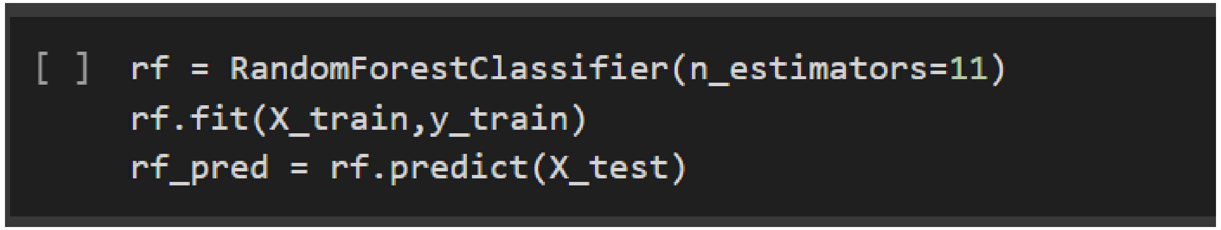
```
              precision    recall  f1-score   support

           0       0.62      1.00      0.76       143
           1       0.00      0.00      0.00        88

    accuracy                           0.62       231
   macro avg       0.31      0.50      0.38       231
weighted avg       0.38      0.62      0.47       231
```



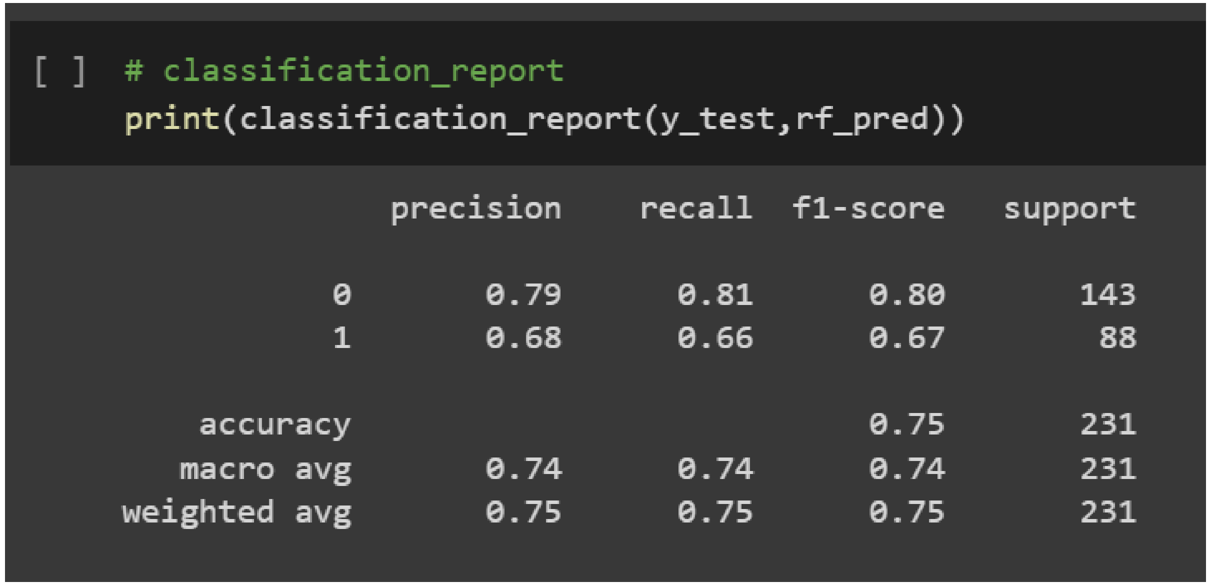
```
print('Confusion matrix for Support Vector Classifier')
disp = plot_confusion_matrix(svc, X_test, y_test, cmap='Blues')
```

```
Confusion matrix for Support Vector Classifier
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.
  warnings.warn(msg, category=FutureWarning)
```

- Random Forest

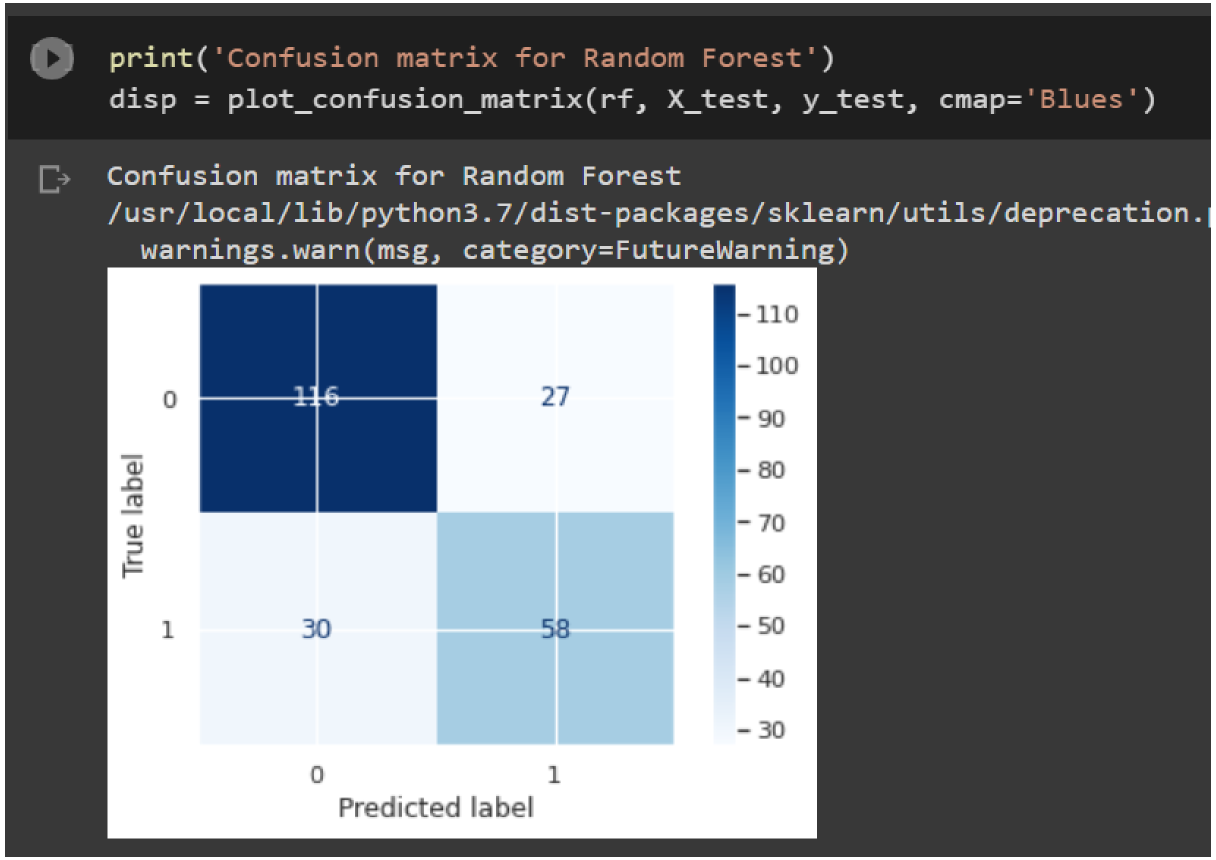


```
[ ]  rf = RandomForestClassifier(n_estimators=11)
     rf.fit(X_train,y_train)
     rf_pred = rf.predict(X_test)
```



```
[ ]  # classification_report
     print(classification_report(y_test,rf_pred))
```

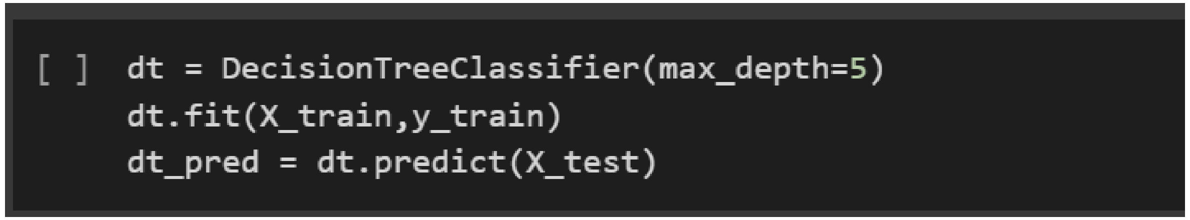
```
              precision    recall  f1-score   support

           0       0.79      0.81      0.80       143
           1       0.68      0.66      0.67        88

    accuracy                           0.75       231
   macro avg       0.74      0.74      0.74       231
weighted avg       0.75      0.75      0.75       231
```



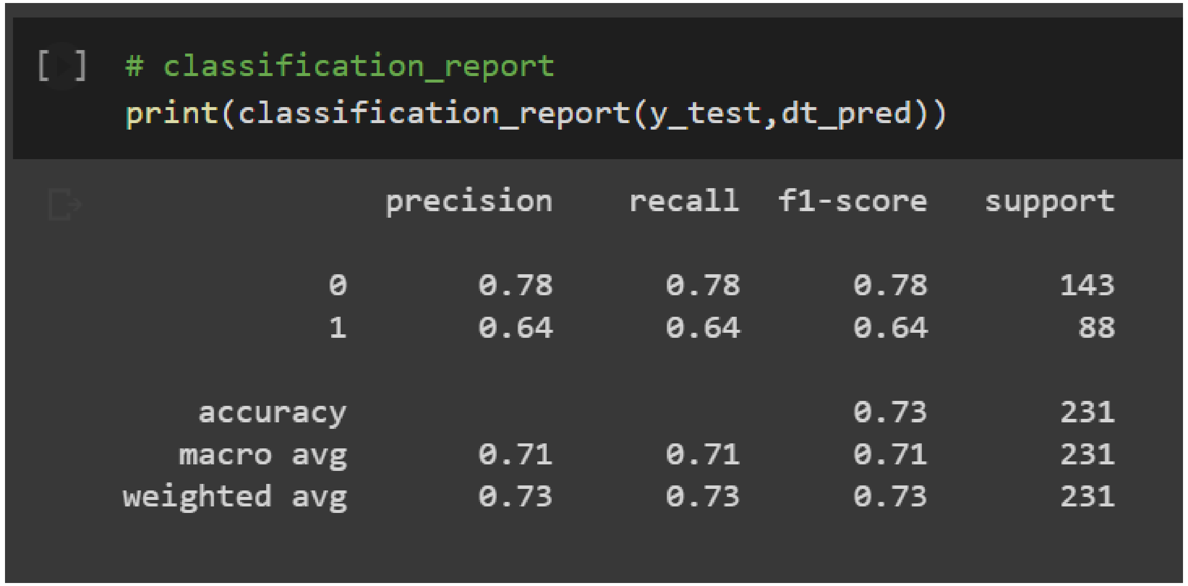
```
print('Confusion matrix for Random Forest')
disp = plot_confusion_matrix(rf, X_test, y_test, cmap='Blues')
```

```
Confusion matrix for Random Forest
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.
  warnings.warn(msg, category=FutureWarning)
```

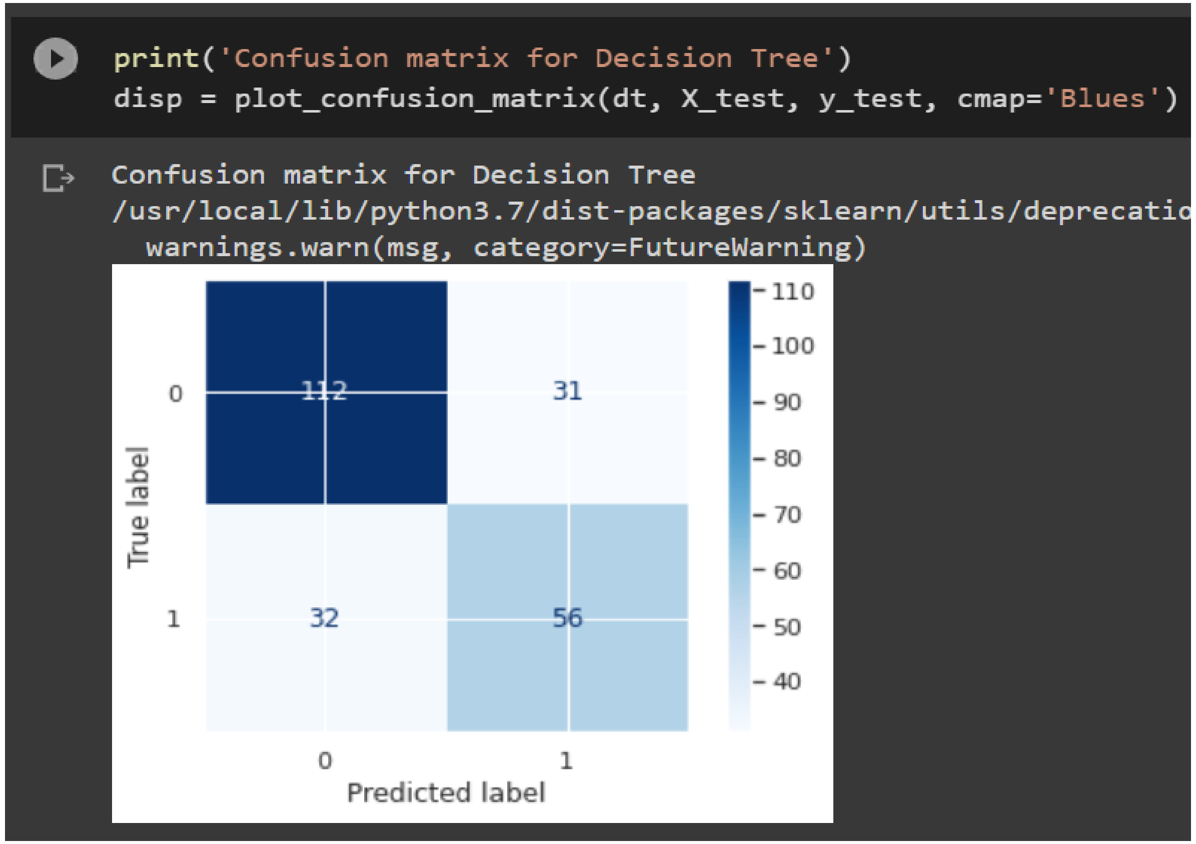
- Decision Tree Classifier



```
[ ]  dt = DecisionTreeClassifier(max_depth=5)
     dt.fit(X_train,y_train)
     dt_pred = dt.predict(X_test)
```

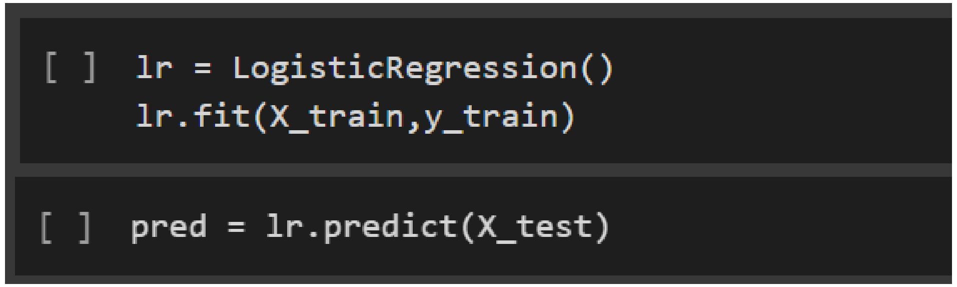


```
[ ]  # classification_report
     print(classification_report(y_test,dt_pred))

              precision    recall  f1-score   support

           0       0.78      0.78      0.78       143
           1       0.64      0.64      0.64        88

    accuracy                           0.73       231
   macro avg       0.71      0.71      0.71       231
weighted avg       0.73      0.73      0.73       231
```



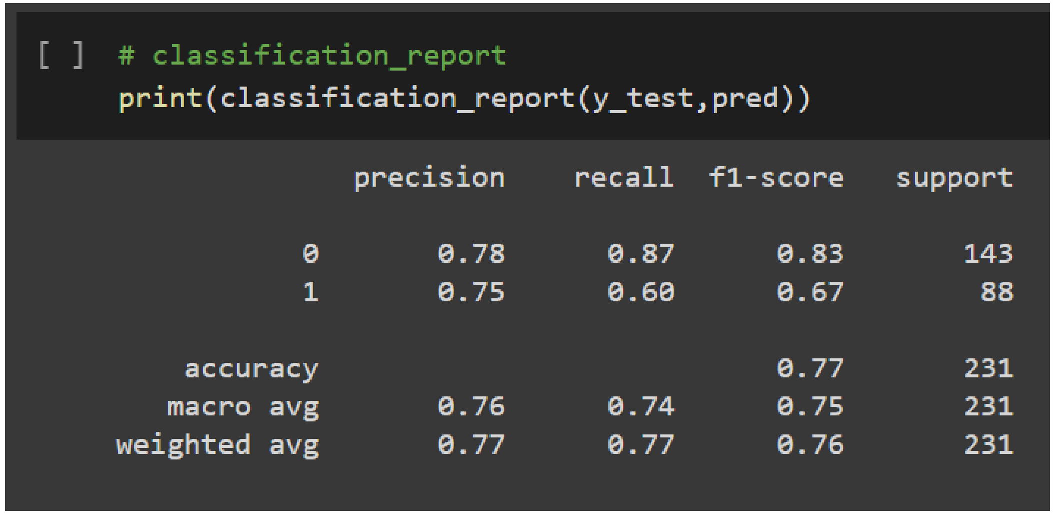
```
print('Confusion matrix for Decision Tree')
disp = plot_confusion_matrix(dt, X_test, y_test, cmap='Blues')

Confusion matrix for Decision Tree
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecatio
  warnings.warn(msg, category=FutureWarning)
```

- Logistic Regression



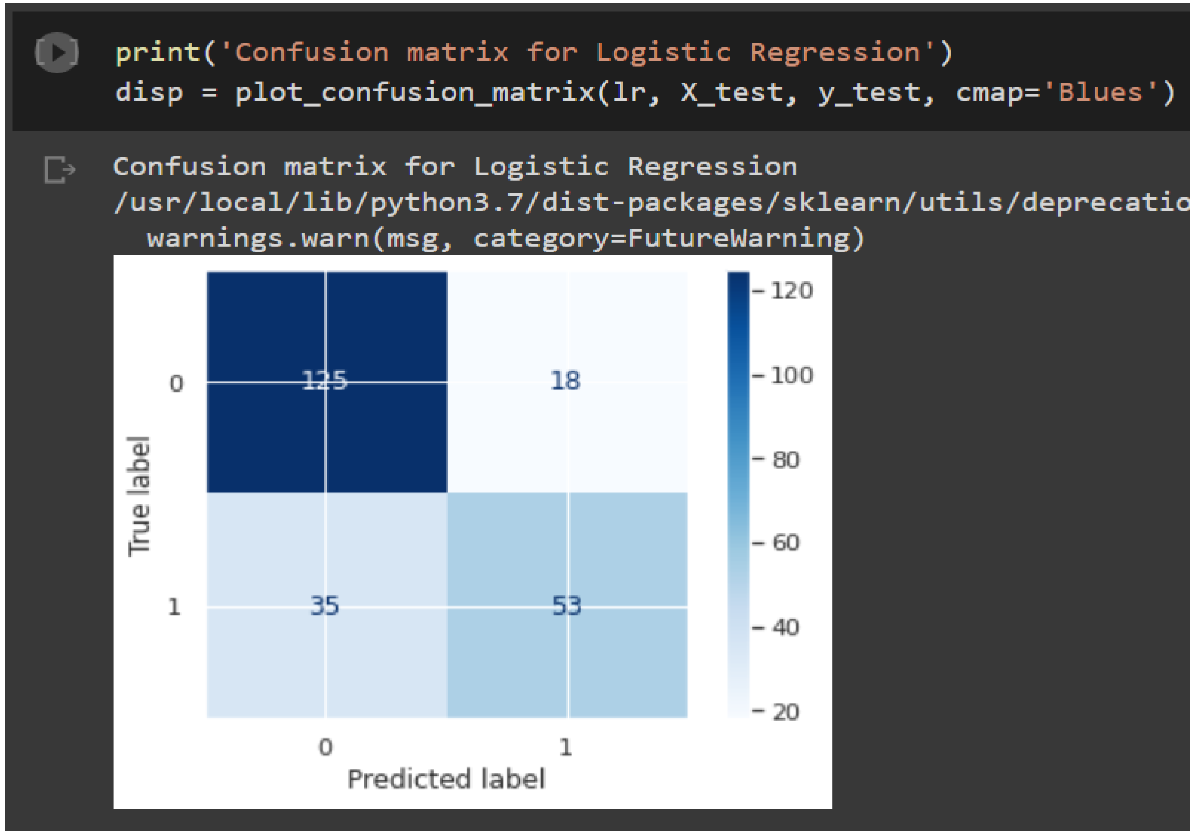
```
[ ]  lr = LogisticRegression()
     lr.fit(X_train,y_train)
```

```
[ ]  pred = lr.predict(X_test)
```



```
[ ]  # classification_report
     print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.87      0.83       143
           1       0.75      0.60      0.67        88

    accuracy                           0.77       231
   macro avg       0.76      0.74      0.75       231
weighted avg       0.77      0.77      0.76       231
```

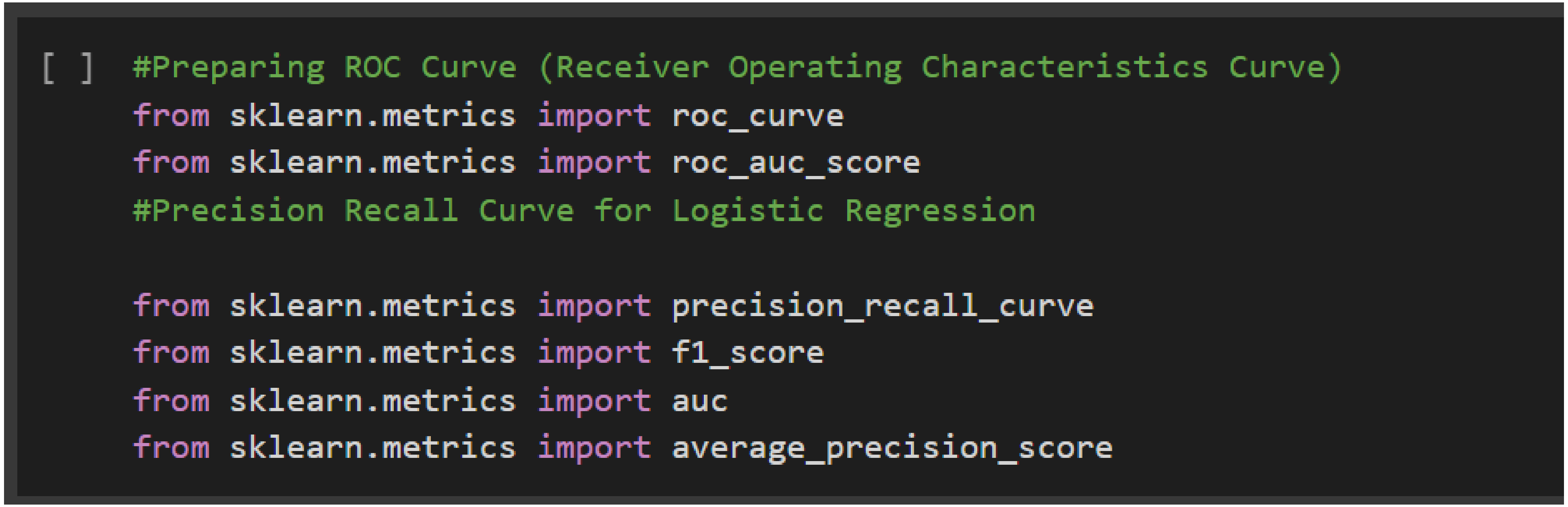


```
print('Confusion matrix for Logistic Regression')
disp = plot_confusion_matrix(lr, X_test, y_test, cmap='Blues')
```

```
Confusion matrix for Logistic Regression
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecatio
  warnings.warn(msg, category=FutureWarning)
```

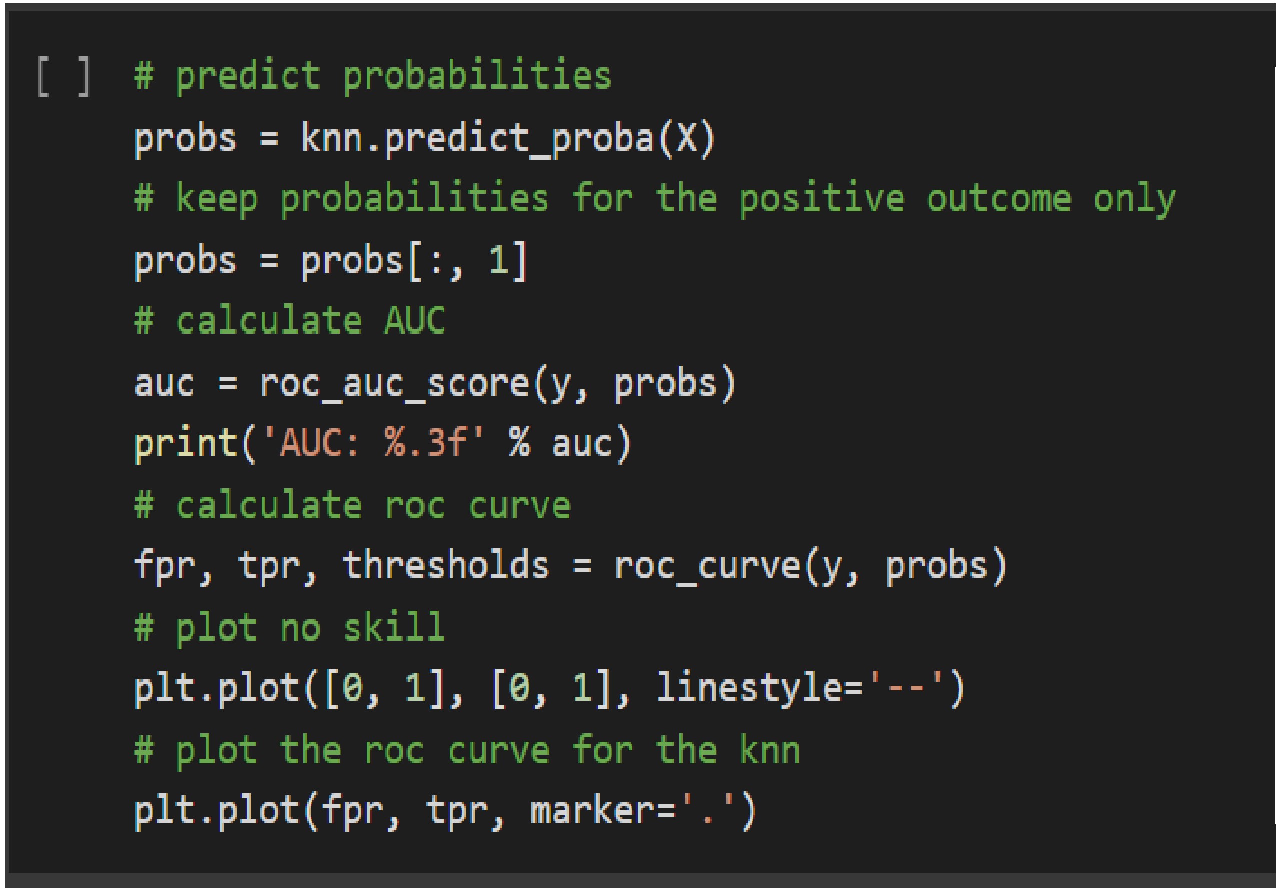
Project task: week-3

- Classification report by analyzing sensitivity, specificity, AUC (ROC curve).

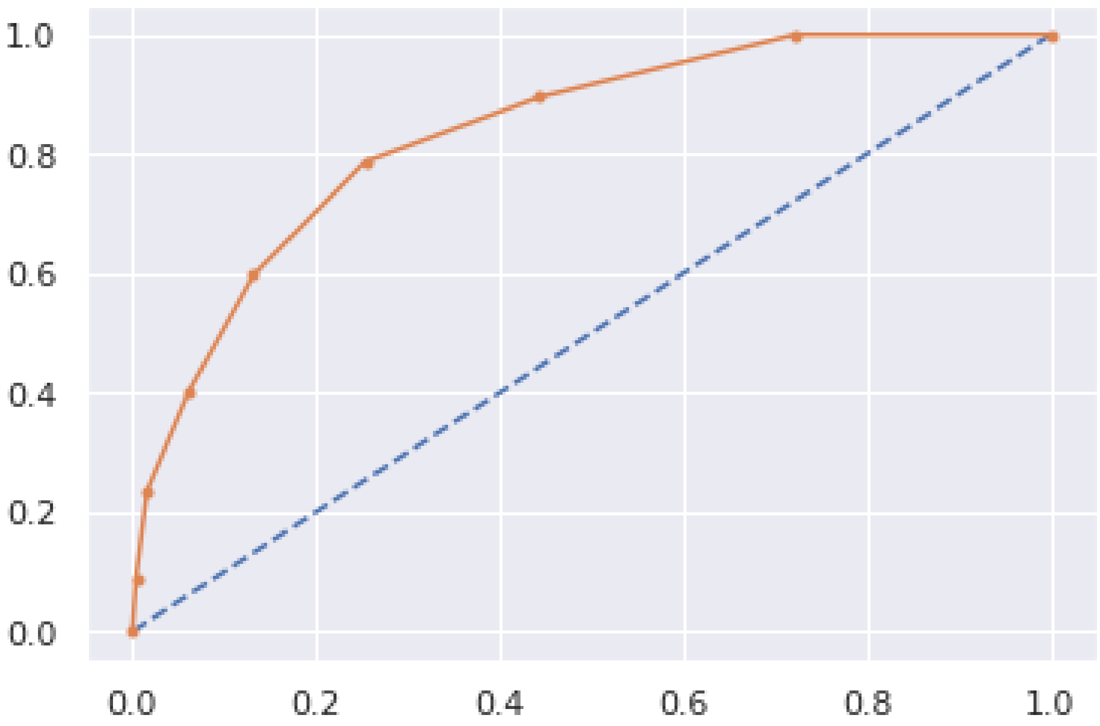


```
[ ]  #Preparing ROC Curve (Receiver Operating Characteristics Curve)
     from sklearn.metrics import roc_curve
     from sklearn.metrics import roc_auc_score
     #Precision Recall Curve for Logistic Regression

     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
```

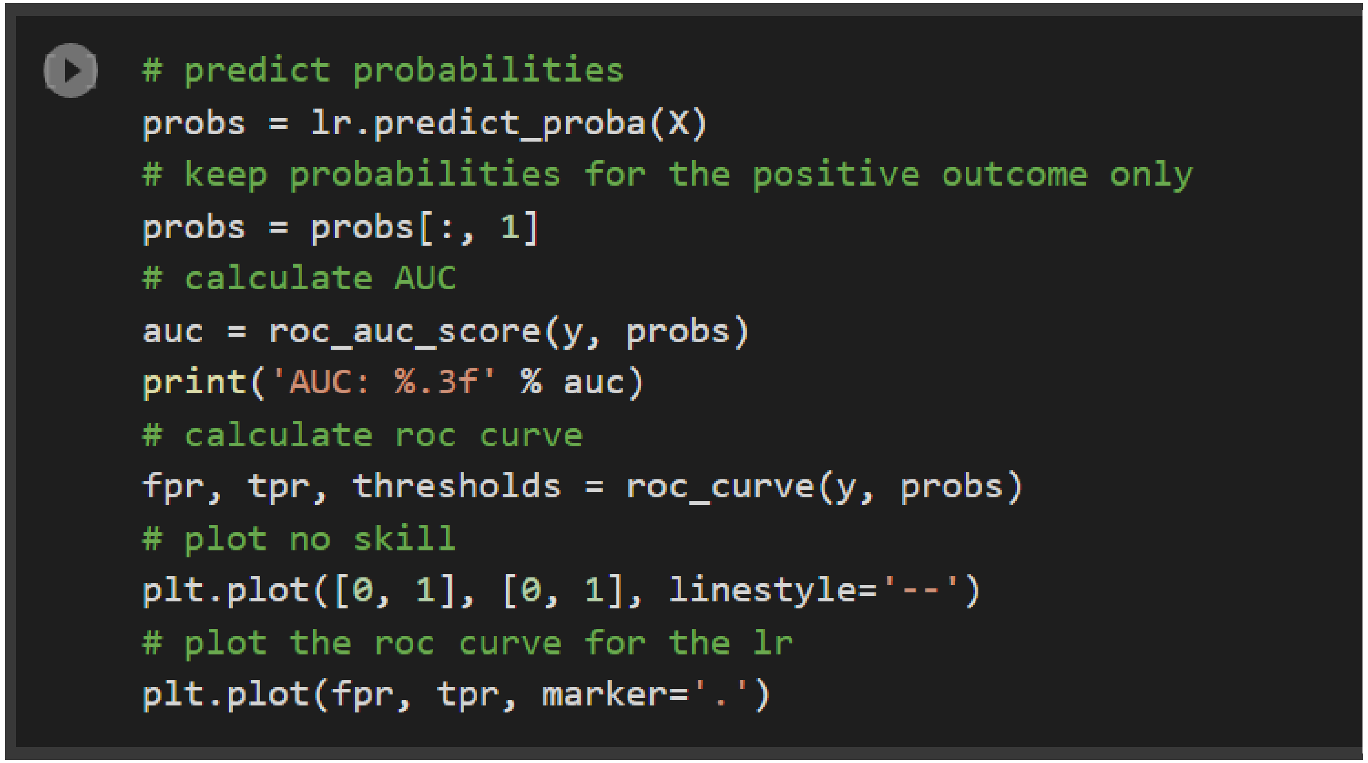
- KNN



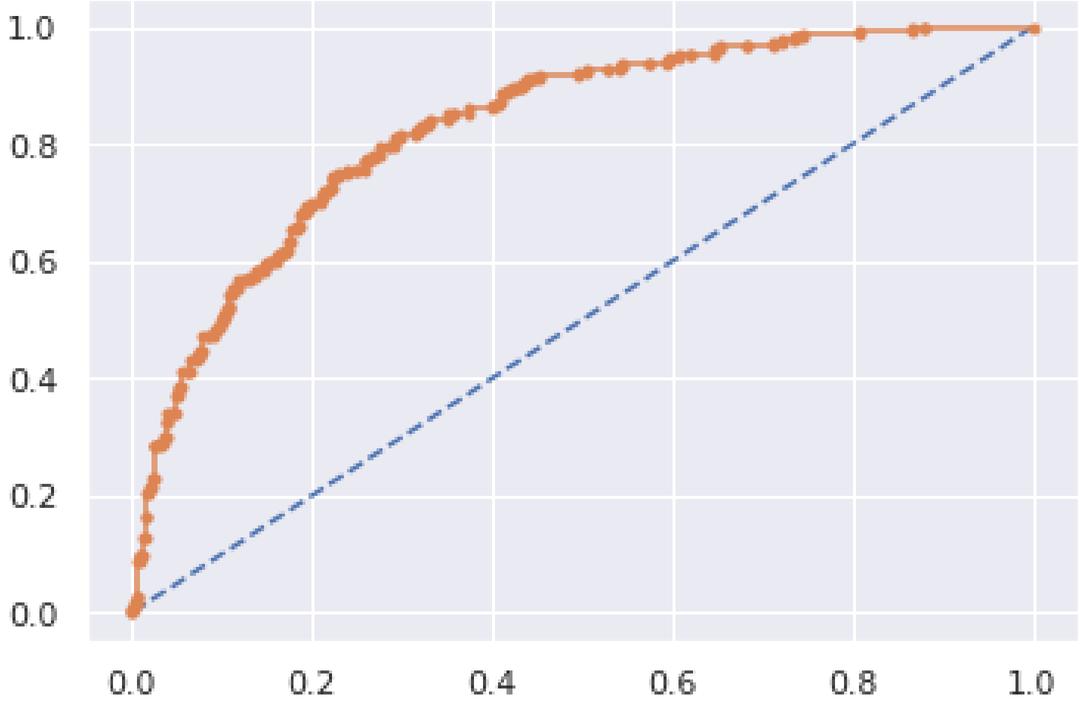
```
[ ]  # predict probabilities
     probs = knn.predict_proba(X)
     # keep probabilities for the positive outcome only
     probs = probs[:, 1]
     # calculate AUC
     auc = roc_auc_score(y, probs)
     print('AUC: %.3f' % auc)
     # calculate roc curve
     fpr, tpr, thresholds = roc_curve(y, probs)
     # plot no skill
     plt.plot([0, 1], [0, 1], linestyle='--')
     # plot the roc curve for the knn
     plt.plot(fpr, tpr, marker='.')
```



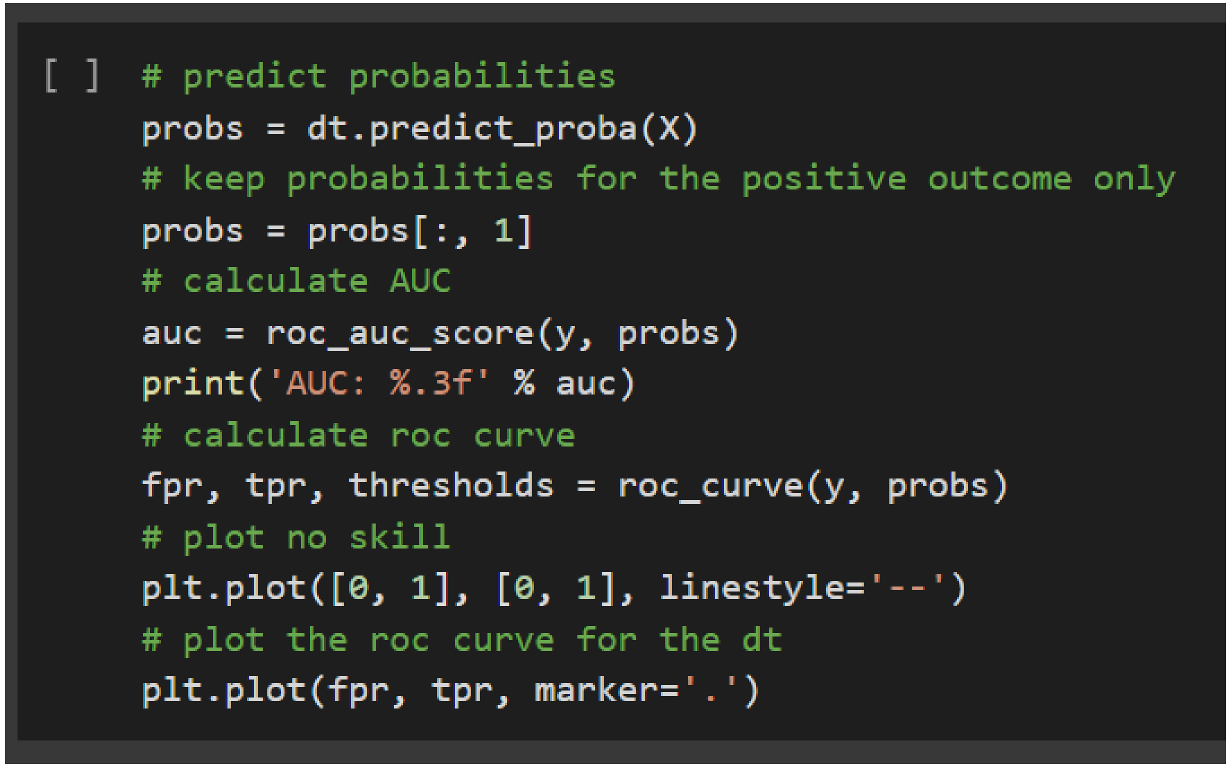
- Logistic Regression



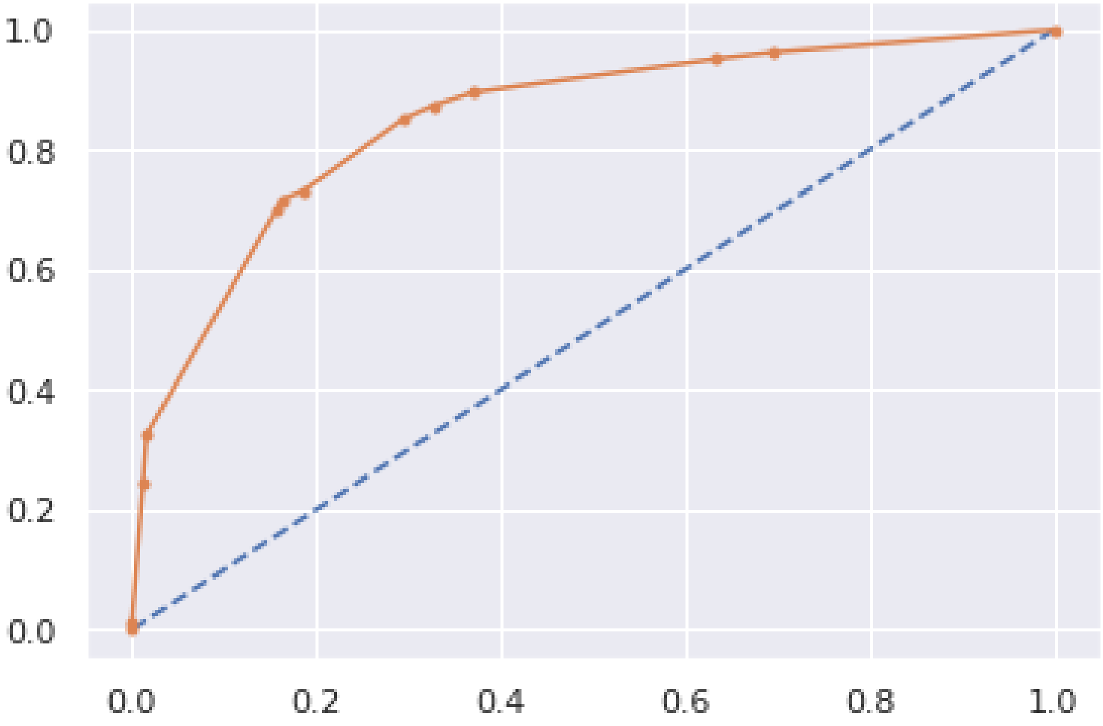
```
# predict probabilities
probs = lr.predict_proba(X)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y, probs)
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the lr
plt.plot(fpr, tpr, marker='.')
```



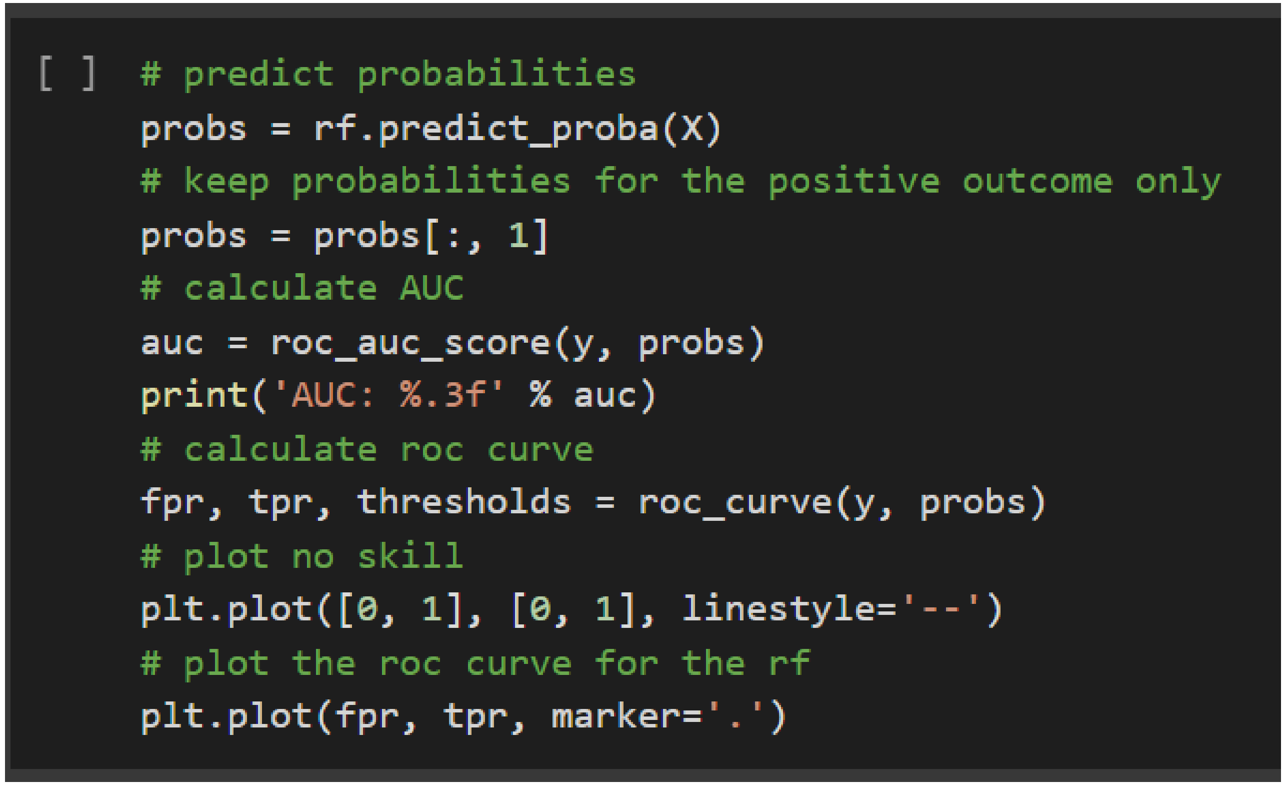
- Decision Tree Classifier



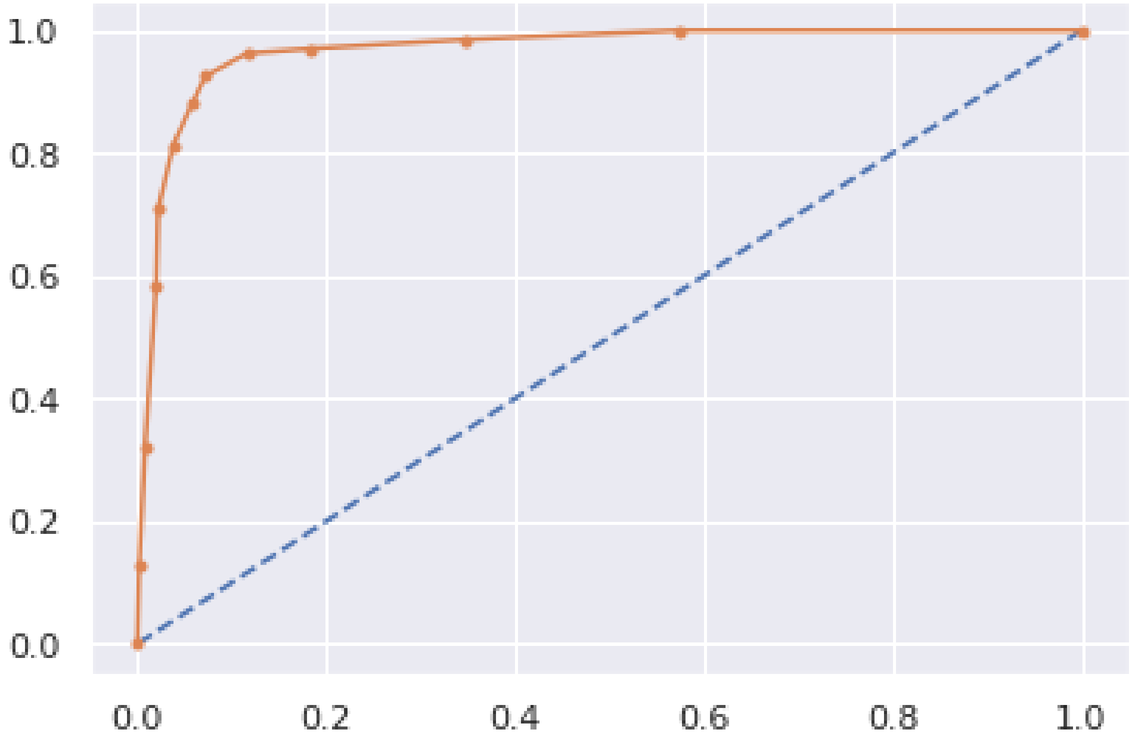
```
# predict probabilities
probs = dt.predict_proba(X)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y, probs)
print('AUC: %.3f' % auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y, probs)
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the dt
plt.plot(fpr, tpr, marker='.')
```



- Random Forest Classifier



```
[ ]  # predict probabilities
     probs = rf.predict_proba(X)
     # keep probabilities for the positive outcome only
     probs = probs[:, 1]
     # calculate AUC
     auc = roc_auc_score(y, probs)
     print('AUC: %.3f' % auc)
     # calculate roc curve
     fpr, tpr, thresholds = roc_curve(y, probs)
     # plot no skill
     plt.plot([0, 1], [0, 1], linestyle='--')
     # plot the roc curve for the rf
     plt.plot(fpr, tpr, marker='.')
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0