INTRODUCTION

Algorithms:

- Logistic Regression: A linear model for binary classification.
- Decision Trees: Tree-like models that make decisions based on features.
- K-Nearest Neighbors (KNN): A non-parametric method that classifies based on the majority class of its nearest neighbors.

LINEAR REGRESSION

In linear regression with two variables (X and Y), there are two regression lines: y on x (to predict Y from X) and x on y (to predict X from Y).

Linear_Regression.py > ...

```python
from sklearn.linear_model import LinearRegression
X = [[1],[2],[3],[4],[5]]
y = [40,50,65,75,90]
model = LinearRegression()
model.fit(X, y)
hours = float(input('Enter study hours: '))

prdicted_marks = model.predict([[hours]])
print(f'Based on your study hour:{hours}, you may score around : {prdicted_marks}')
```

LOGISTIC REGRESSION

the probability of a binary outcome (like Yes/No) is related to predictor variables via a linear equation, and second, this probability is then transformed by a sigmoid (or logit) function to produce an S-shaped curve that models the relationship and outputs a probability between 0 and 1.

Logistic Regression.py > ...

```python
from sklearn.linear_model import LogisticRegression
X = [[1],[2],[3],[4],[5]]
y = [0,0,1,1,1]

model = LogisticRegression()
model.fit(X,y)
hours = float(input('enter study hours:'))
prdicted_value = model.predict([[hours]])[0]
if prdicted_value == 0:
    print(f'Based on your study hour:{hours}, you may not pass the exam')
else:
    print(f'Based on your study hour:{hours}, you may pass the exam')
```

DECISION TREE

A branch connecting a decision node to a subsequent node (either another decision node or a leaf node).

Decision Tree.py > ... ~/Desktop/RAW DATA/untitled folder/Logistic Regression.py

```python
from sklearn.tree import DecisionTreeClassifier
X = [
    [7,2],
    [8,3],
    [9,8],
    [10,9]
]

y = [0,0,1,1]
model = DecisionTreeClassifier()
model.fit(X,y)
size = float(input('Enter size of fruit:'))
shade = float(input('enter color shade 1-10;'))
value = model.predict([[size,shade]])[0]
if value == 0:
    print(  f'Based on size:{size} and shade:{shade}, the fruit is apple')
else:
    print(f'Based on size:{size} and shade:{shade}, the fruit is orange')
```

K-NN

The KNN (K-Nearest Neighbors) algorithm classifies new data points by comparing them to known neighbors, assigning a new data point to the class that has the most "votes" among its 'K' closest training data points

K_NN.py > ...

```python
from sklearn.neighbors import KNeighborsClassifier

X = [
    [180,7],
    [200,7.5],
    [250,8],
    [300,8.5],
    [350,9],
    [360,9.5]
]
y = [0,0,0,1,1,1]
model = KNeighborsClassifier(n_neighbors =3)
model.fit(X,y)
weight = float(input('Enter weight of fruit:'))
size = float(input('enter size of fruit:'))
value = model.predict([[weight,size]])[0]
if value == 0:
    print(f'Based on weight:{weight} and size:{size}, the fruit is apple')
else:
    print(f'Based on weight:{weight} and size:{size}, the fruit is orange')
```

APR-SCORE

Accuracy is overall correctness, precision is the correctness of positive predictions, and recall is the model's ability to find all positive instances in the data

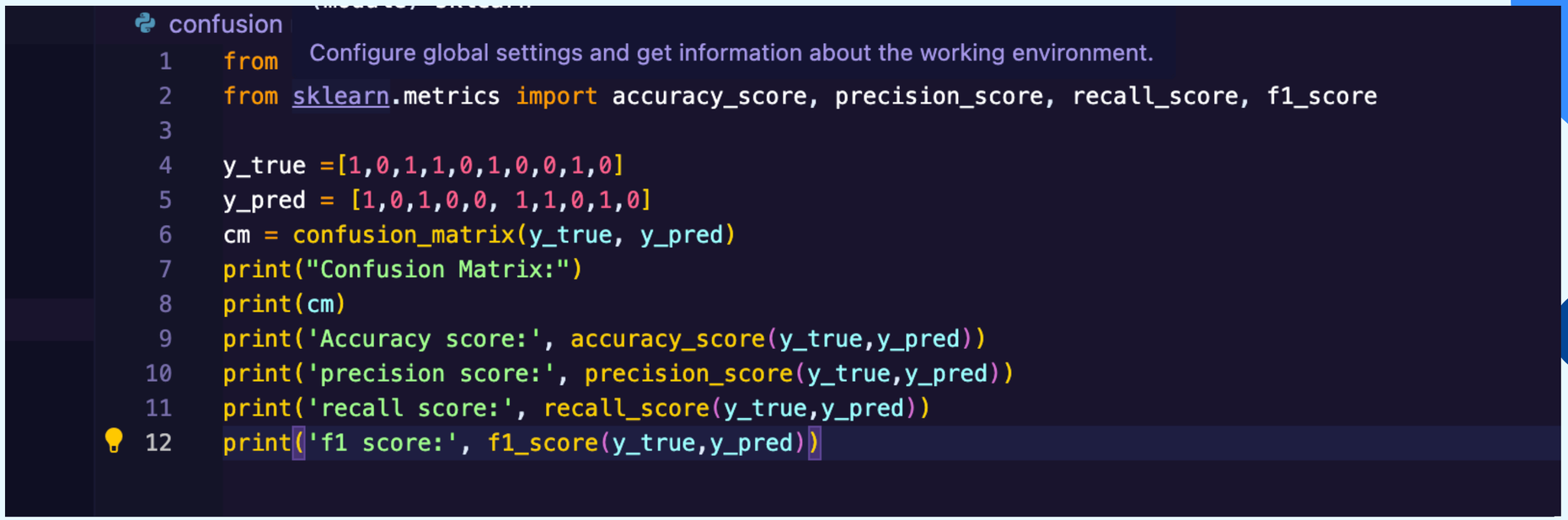
APR F1.py > ...

```python
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score

y_true = [1,0,1,1,0,1,0]
y_pred =[1,0,1,0,0,1,1,]
print('Accuracy score:', accuracy_score(y_true,y_pred))
print('precision score:', precision_score(y_true,y_pred))
print('recall score:', recall_score(y_true,y_pred))
print('f1 score:', f1_score(y_true,y_pred))
```

CONFUSION MATRIX

A confusion matrix is a table that evaluates the performance of a classification algorithm. The matrix uses target values to compare with machine learning (ML) predicted values. Each row in the matrix represents instances in a predicted class and each column represents instances in an actual class or vice versa



MAE, MSE, RMSE

Mean Absolute Error (MAE), which measures the average magnitude of the errors.
Mean Squared Error (MSE), which measures the average squared magnitude of the errors.
Root Mean Squared Error (RMSE), which measures the square root of the average squared magnitude of the errors.

MAE MSE RMSE.py > ...

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

real_score = [90,60,80,100]

predicted_score = [85,70,70,95]
mae = mean_absolute_error(real_score, predicted_score)
mse = mean_squared_error(real_score, predicted_score)
rmse = np.sqrt(mse)
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

THANK YOU