Vivekanand Education Society's

Institute of Technology

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Department of Information Technology

AIDS - 2 Lab

Experiment - 10

Aim: **Supervised learning algorithm Random Forest**

Roll No.	70
Name	MAYURI SHRIDATTA YERANDE
Class	D20B
Subject	AIDS - 2
Grade:	

EXPERIMENT - 10

AIM: Supervised learning algorithm Random Forest

THEORY:

Supervised learning is a type of machine learning where an algorithm learns from labeled training data to make predictions or decisions without human intervention. It is called "supervised" because it involves a "teacher" who provides the algorithm with the correct answers during training, allowing the algorithm to learn the relationship between input data and output labels.

Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as bagging. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees. Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called Bootstrap. We need to approach the Random Forest regression technique like any other machine learning technique.

Advantages Random Forest Regression

- It is easy to use and less sensitive to the training data compared to the decision tree.
- It is more accurate than the decision tree algorithm.
- It is effective in handling large datasets that have many attributes.
- It can handle missing data, outliers, and noisy features.

Disadvantages Random Forest Regression

- The model can also be difficult to interpret.
- This algorithm may require some domain expertise to choose the appropriate parameters like the number of decision trees, the maximum depth of each tree, and the number of features to consider at each split.
- It is computationally expensive, especially for large datasets.
- It may suffer from overfitting if the model is too complex or the number of decision trees is too high.

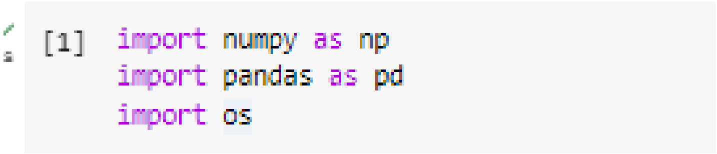
\

IMPLEMENTATION:

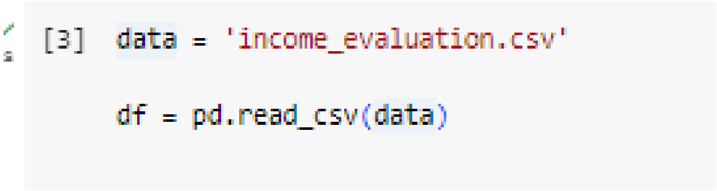
TO-DO: To make predictions where the prediction task is to determine whether a person makes over 50K a year. Implementing Random Forest Classification with Python and Scikit-Learn.

Dataset Link: https://www.kaggle.com/datasets/lodetomasi1995/income-classification

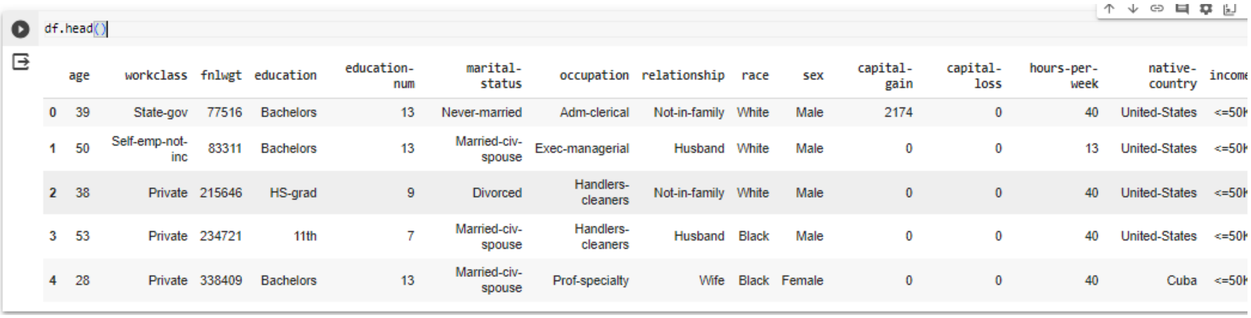
- **Import libraries and read the dataset**



```
[1] import numpy as np
    import pandas as pd
    import os
```

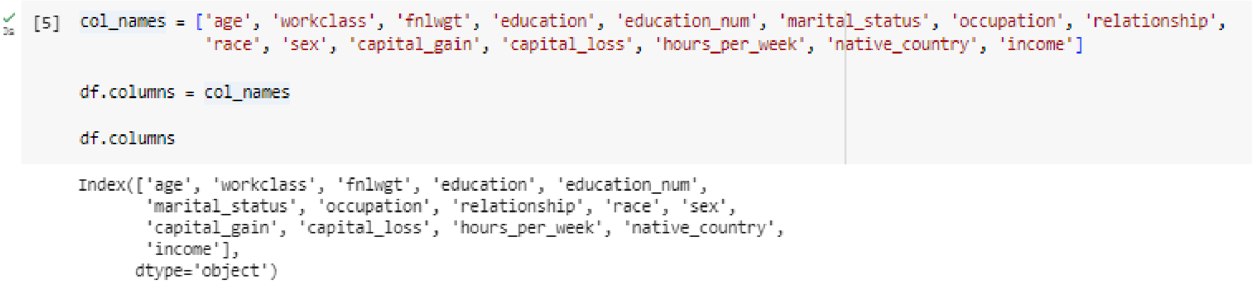


```
[3] data = 'income_evaluation.csv'

    df = pd.read_csv(data)
```



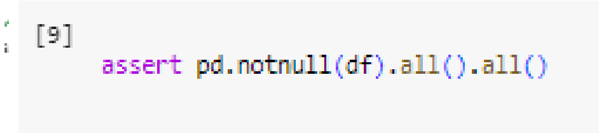
```
df.head()
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K



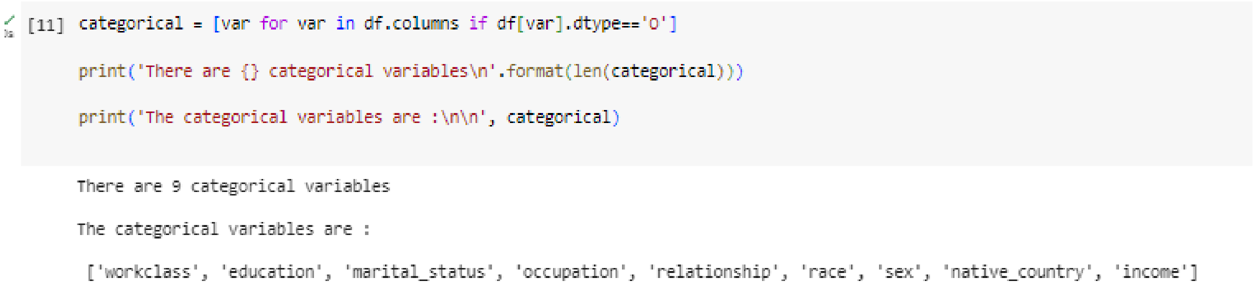
```
[5] col_names = ['age', 'workclass', 'fnlwgt', 'education', 'education_num', 'marital_status', 'occupation', 'relationship',
                 'race', 'sex', 'capital_gain', 'capital_loss', 'hours_per_week', 'native_country', 'income']

    df.columns = col_names

    df.columns

    Index(['age', 'workclass', 'fnlwgt', 'education', 'education_num',
           'marital_status', 'occupation', 'relationship', 'race', 'sex',
           'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
           'income'],
          dtype='object')
```

- **Checking if any null values are present or not**



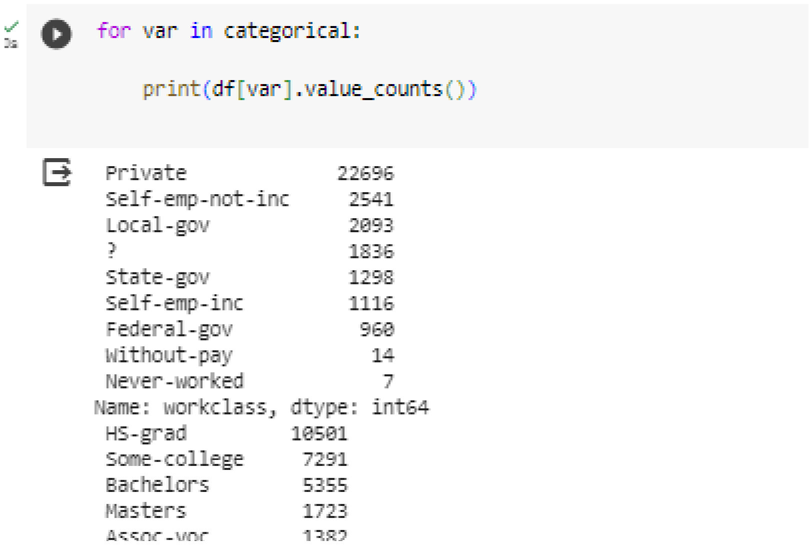
```
[9]
  assert pd.notnull(df).all().all()
```

- **Extracting the categorical data**



```
[11] categorical = [var for var in df.columns if df[var].dtype=='O']

     print('There are {} categorical variables\n'.format(len(categorical)))

     print('The categorical variables are :\n\n', categorical)
```

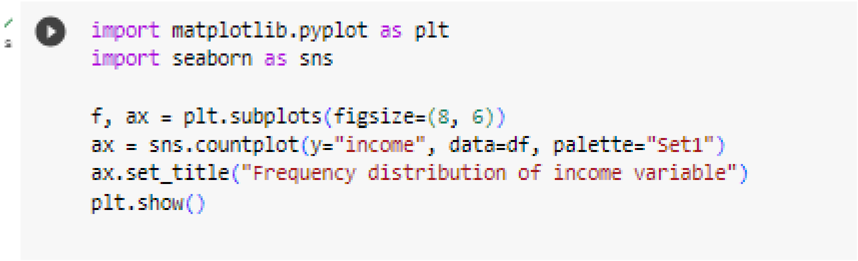
```
There are 9 categorical variables

The categorical variables are :

 ['workclass', 'education', 'marital_status', 'occupation', 'relationship', 'race', 'sex', 'native_country', 'income']
```



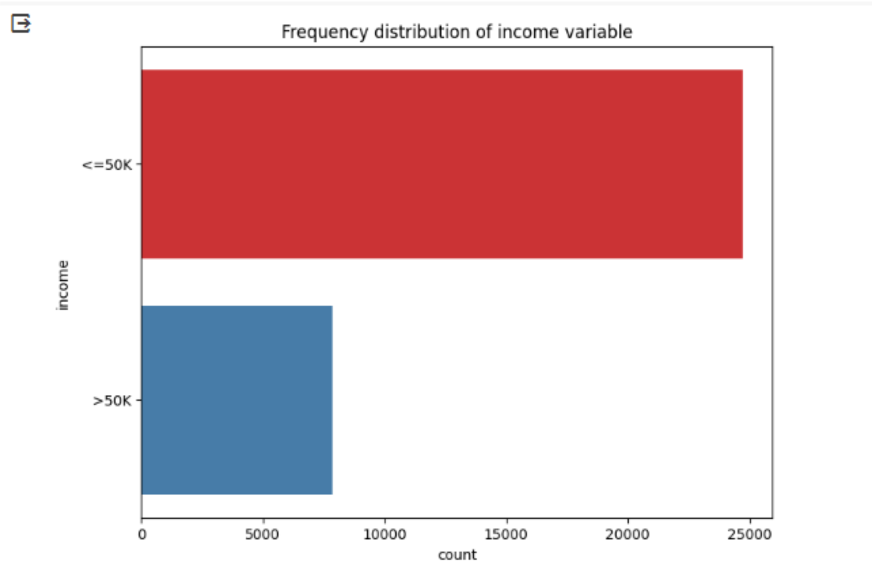
```
for var in categorical:

    print(df[var].value_counts())
```

```
 Private             22696
 Self-emp-not-inc     2541
 Local-gov            2093
 ?                    1836
 State-gov            1298
 Self-emp-inc         1116
 Federal-gov           960
 Without-pay            14
 Never-worked            7
Name: workclass, dtype: int64
 HS-grad         10501
 Some-college     7291
 Bachelors        5355
 Masters          1723
 Assoc-voc        1382
```

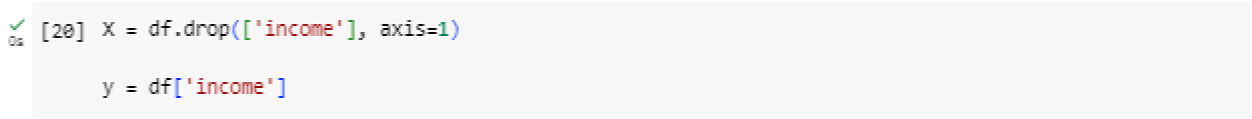
- **Plotting the frequency distribution graph**



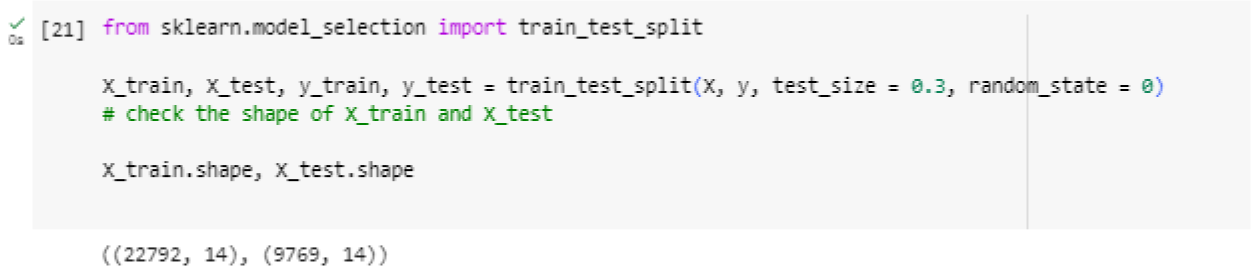
```
import matplotlib.pyplot as plt
import seaborn as sns

f, ax = plt.subplots(figsize=(8, 6))
ax = sns.countplot(y="income", data=df, palette="Set1")
ax.set_title("Frequency distribution of income variable")
plt.show()
```



- **Splitting the dataset into test and train**

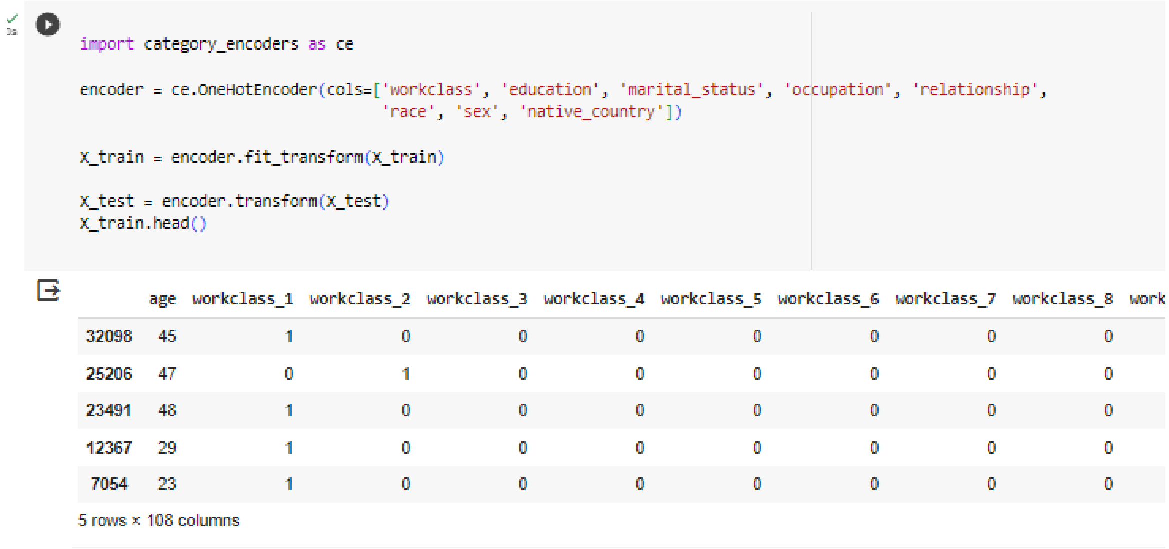


```
[20] X = df.drop(['income'], axis=1)

     y = df['income']
```



```
[21] 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 = 0)
     # check the shape of X_train and X_test

     X_train.shape, X_test.shape

     ((22792, 14), (9769, 14))
```

- **Performing hot encoding**



```
import category_encoders as ce

encoder = ce.OneHotEncoder(cols=['workclass', 'education', 'marital_status', 'occupation', 'relationship',
                                 'race', 'sex', 'native_country'])

X_train = encoder.fit_transform(X_train)

X_test = encoder.transform(X_test)
X_train.head()
```

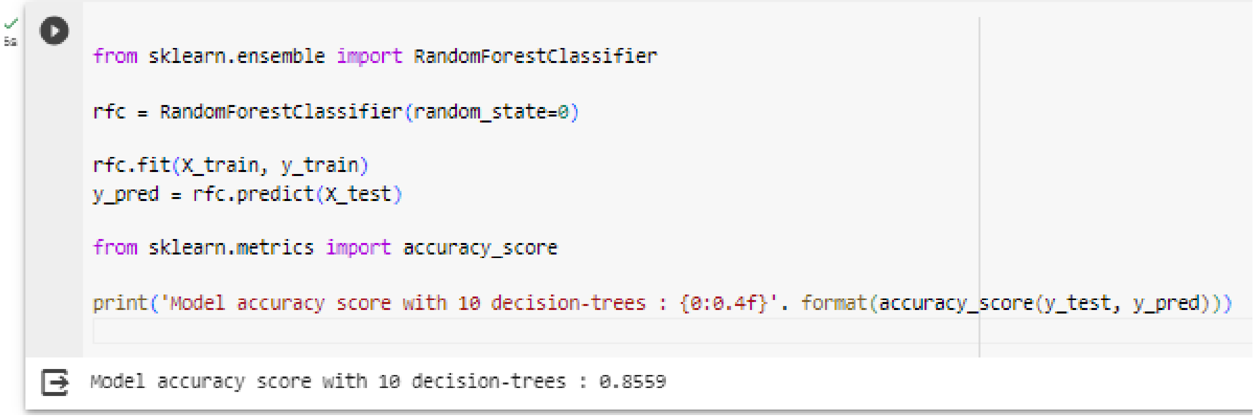
	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	workclass_7	workclass_8	work
32098	45	1	0	0	0	0	0	0	0	
25206	47	0	1	0	0	0	0	0	0	
23491	48	1	0	0	0	0	0	0	0	
12367	29	1	0	0	0	0	0	0	0	
7054	23	1	0	0	0	0	0	0	0	

5 rows × 108 columns

- **This code scales your training and test data using RobustScaler to ensure consistent feature scales, which is important for certain machine learning algorithms.**

```
cols = X_train.columns
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
X_train = pd.DataFrame(X_train, columns=[cols])
X_test = pd.DataFrame(X_test, columns=[cols])
```

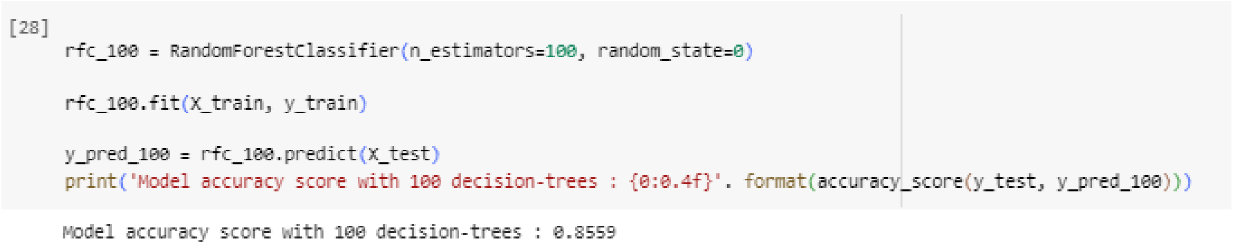
- **We now have X_train dataset ready to be fed into the Random Forest classifier**
- **We check the accuracy of the model with 10 decision trees**



```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(random_state=0)

rfc.fit(X_train, y_train)
y_pred = rfc.predict(X_test)

from sklearn.metrics import accuracy_score

print('Model accuracy score with 10 decision-trees : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with 10 decision-trees : 0.8559
```

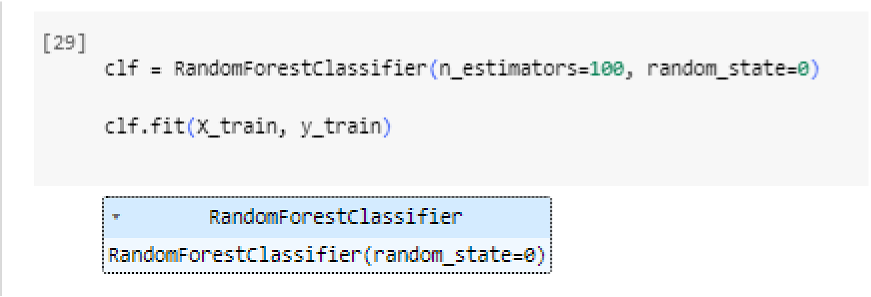
- **Checking the accuracy of the model with 100 decision trees**



```
[28]
rfc_100 = RandomForestClassifier(n_estimators=100, random_state=0)

rfc_100.fit(X_train, y_train)

y_pred_100 = rfc_100.predict(X_test)
print('Model accuracy score with 100 decision-trees : {0:0.4f}'. format(accuracy_score(y_test, y_pred_100)))
```

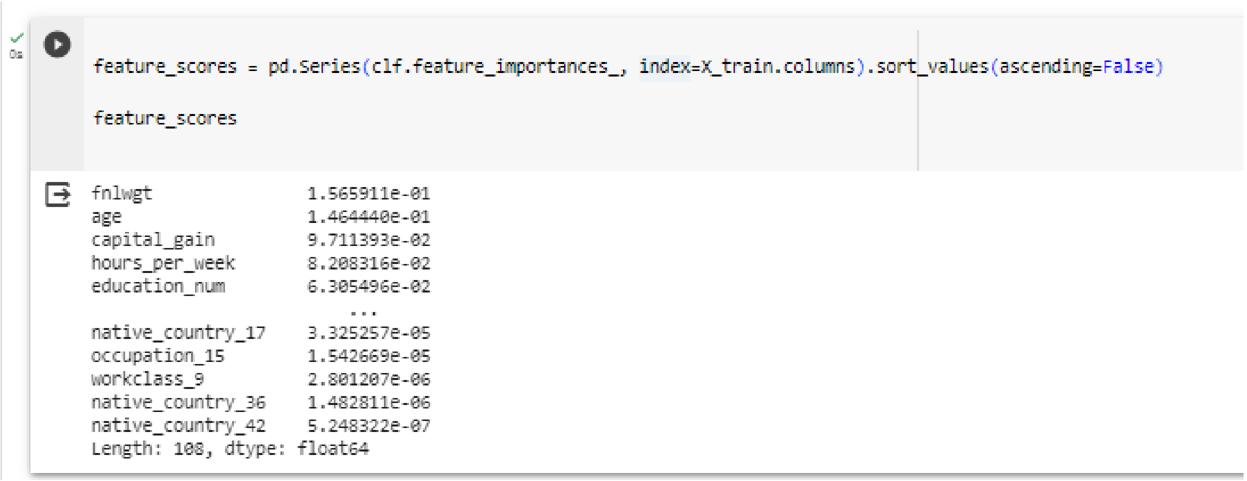
```
Model accuracy score with 100 decision-trees : 0.8559
```

- **Random forest classifier**



```
[29]
clf = RandomForestClassifier(n_estimators=100, random_state=0)

clf.fit(X_train, y_train)
```

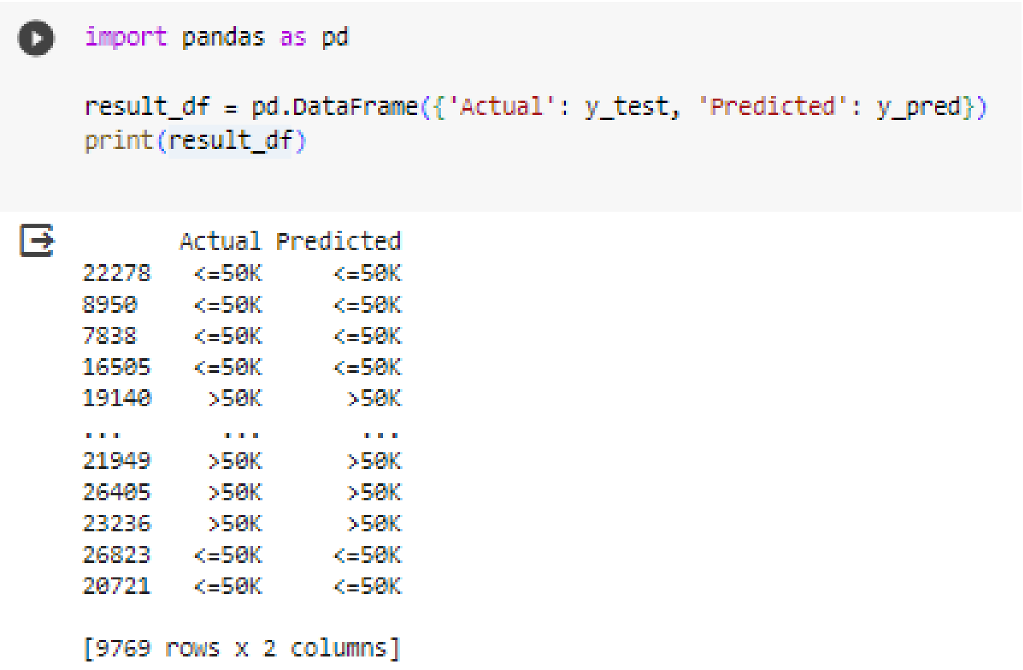
```
RandomForestClassifier
RandomForestClassifier(random_state=0)
```



```
feature_scores = pd.Series(clf.feature_importances_, index=X_train.columns).sort_values(ascending=False)

feature_scores
```

```
fnlwgt               1.565911e-01
age                  1.464440e-01
capital_gain         9.711393e-02
hours_per_week       8.208316e-02
education_num        6.305496e-02
                         ...
native_country_17    3.325257e-05
occupation_15        1.542669e-05
workclass_9          2.801207e-06
native_country_36    1.482811e-06
native_country_42    5.248322e-07
Length: 108, dtype: float64
```

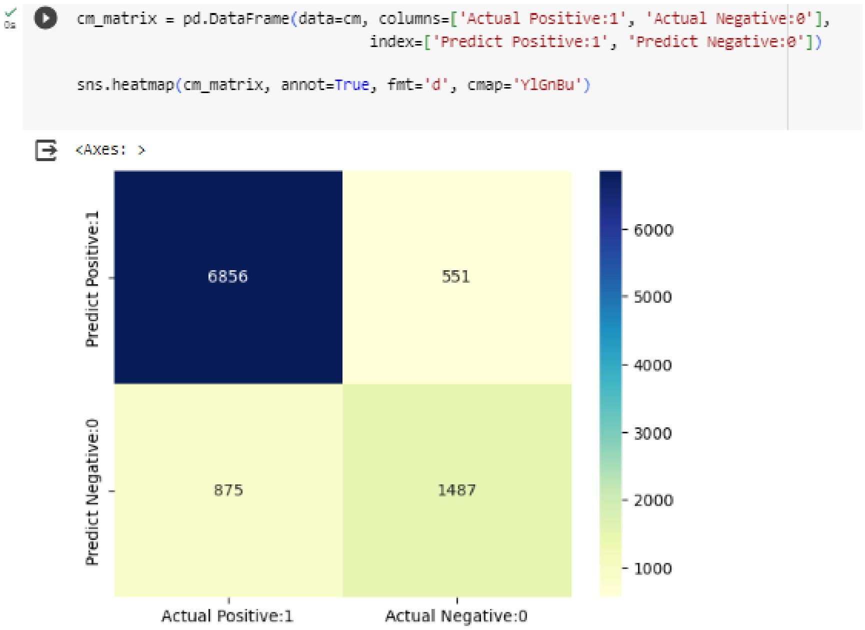
- **Predicted vs Actual Output**



```
import pandas as pd

result_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(result_df)
```

```
       Actual Predicted
22278   <=50K     <=50K
8950    <=50K     <=50K
7838    <=50K     <=50K
16505   <=50K     <=50K
19140    >50K      >50K
...       ...       ...
21949    >50K      >50K
26405    >50K      >50K
23236    >50K      >50K
26823   <=50K     <=50K
20721   <=50K     <=50K

[9769 rows x 2 columns]
```

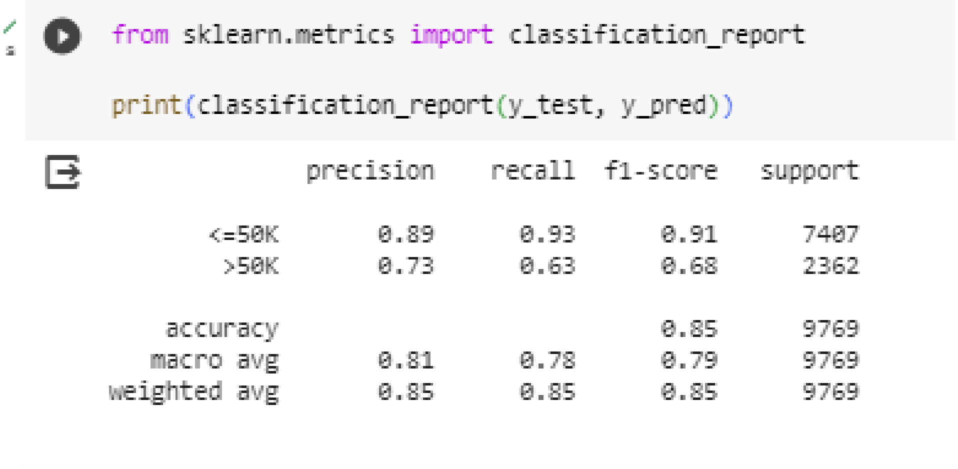
- **Confusion Matrix of the model**



```
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                                 index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

```
<Axes: >
```

- **Calculating precision, recall, f1 score and support**



```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

       <=50K       0.89      0.93      0.91      7407
        >50K       0.73      0.63      0.68      2362

    accuracy                           0.85      9769
   macro avg       0.81      0.78      0.79      9769
weighted avg       0.85      0.85      0.85      9769
```

CONCLUSION: Therefore, Random Forest is used for predicting results in machine learning. It is a powerful ensemble learning method that can be used for both classification and regression tasks. Random Forest combines the predictions of multiple decision trees to produce more accurate and robust predictions. Thus we successfully implemented random forest for income classification.