Load Dataset

```
import pandas as pd
df = pd.read_csv('/content/drive/MyDrive/ML Projects/adult.csv')
df
```

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capi
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	
4	18	?	103497	Some-college	10	Never-married	?	Own-child	White	Female	
...	...	...	...	...	...	...	...	...	...	...	
48837	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	
48838	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	
48839	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	
48840	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	
48841	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	1

48842 rows × 15 columns

Data Preparation

Data Preprocessing

```
df.shape
```

```
(48842, 15)
```

```
df.columns
```

```
Index(['age', 'workclass', 'fnlwgt', 'education', 'educational-num',
       'marital-status', 'occupation', 'relationship', 'race', 'gender',
       'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
       'income'],
      dtype='object')
```

```
df.dtypes
```

```
age                 int64
workclass          object
fnlwgt              int64
education          object
educational-num     int64
marital-status     object
occupation         object
relationship       object
race               object
gender             object
capital-gain        int64
capital-loss        int64
hours-per-week      int64
native-country     object
income             object
dtype: object
```

```
df.nunique()
```

```
age                   74
workclass              9
fnlwgt             28523
education             16
educational-num       16
marital-status         7
occupation            15
relationship           6
race                   5
gender                 2
capital-gain         123
capital-loss          99
hours-per-week        96
native-country        42
income                 2
dtype: int64
```

```
df.isna().any()
```

```
age                False
workclass          False
fnlwgt             False
education          False
educational-num    False
marital-status     False
occupation         False
relationship       False
race               False
gender             False
capital-gain       False
capital-loss       False
hours-per-week     False
native-country     False
income             False
dtype: bool
```

```
df.describe()
```

	age	fnlwgt	educational-num	capital-gain	capital-loss	hours-per-week
count	48842.000000	4.884200e+04	48842.000000	48842.000000	48842.000000	48842.000000
mean	38.643585	1.896641e+05	10.078089	1079.067626	87.502314	40.422382
std	13.710510	1.056040e+05	2.570973	7452.019058	403.004552	12.391444
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.175505e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.781445e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.376420e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.490400e+06	16.000000	99999.000000	4356.000000	99.000000

```
df.info()
```

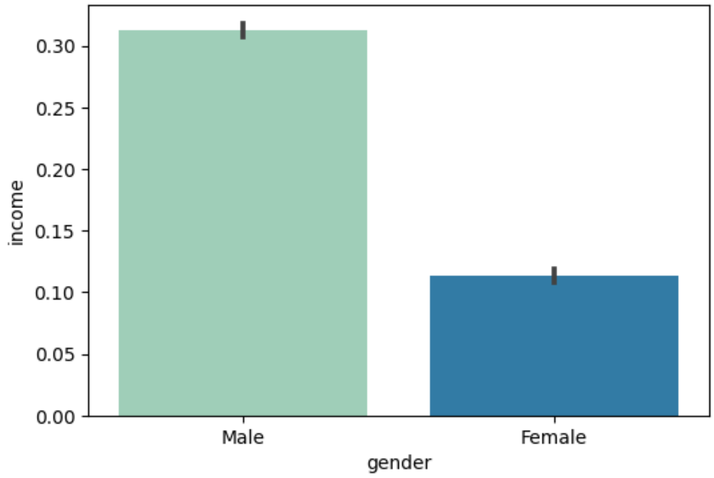
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   age              48842 non-null  int64 
 1   workclass        48842 non-null  object
 2   fnlwgt           48842 non-null  int64 
 3   education        48842 non-null  object
 4   educational-num  48842 non-null  int64 
 5   marital-status   48842 non-null  object
 6   occupation       48842 non-null  object
 7   relationship     48842 non-null  object
 8   race             48842 non-null  object
 9   gender           48842 non-null  object
 10  capital-gain     48842 non-null  int64 
 11  capital-loss     48842 non-null  int64 
 12  hours-per-week   48842 non-null  int64 
 13  native-country   48842 non-null  object
 14  income           48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

Data Visualization

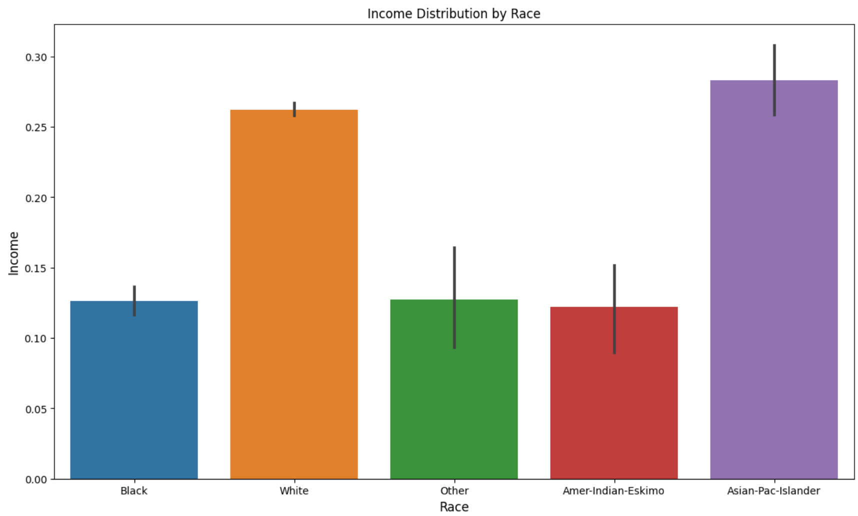
```
import matplotlib.pyplot as plt
import seaborn as sns


plt.figure(figsize=(6,4))
sns.barplot(x = 'gender' , y = 'income' , data = df , palette='YlGnBu')
```

```
<Axes: xlabel='gender', ylabel='income'>
```

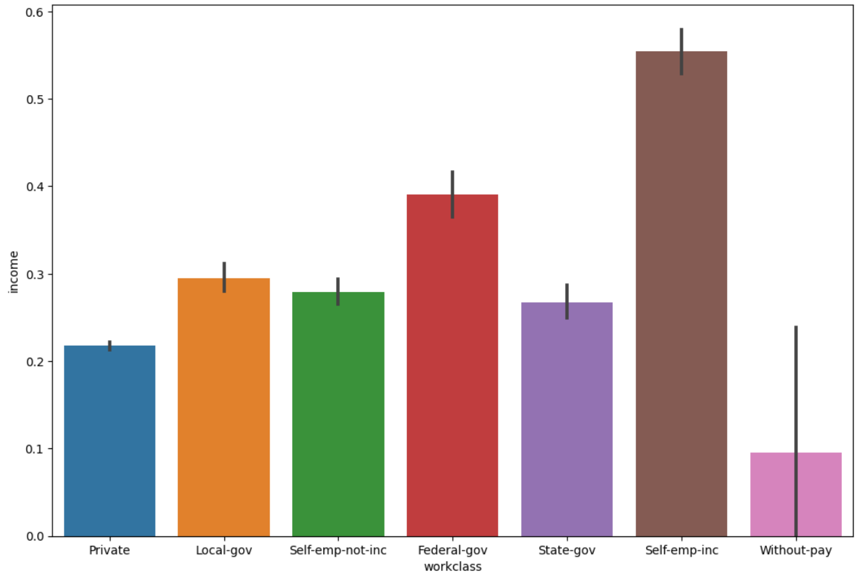



```
plt.figure(figsize=(14,8))
sns.barplot(x = 'race' , y = 'income' , data = df)
plt.xlabel('Race' , size='large')
plt.ylabel('Income' , size = 'large')
plt.title("Income Distribution by Race")
plt.show()
```

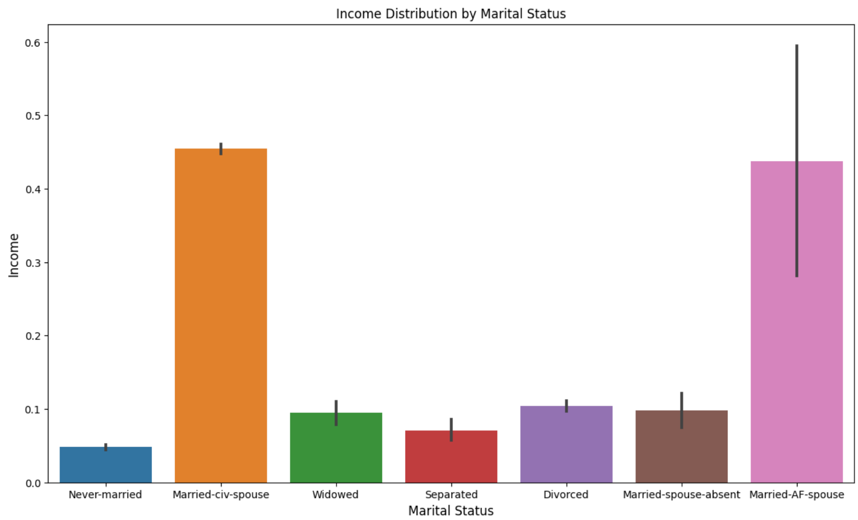



```
plt.figure(figsize=(12,8))
sns.barplot(x = "workclass" , y = "income" , data = df)
```

```
<Axes: xlabel='workclass', ylabel='income'>
```

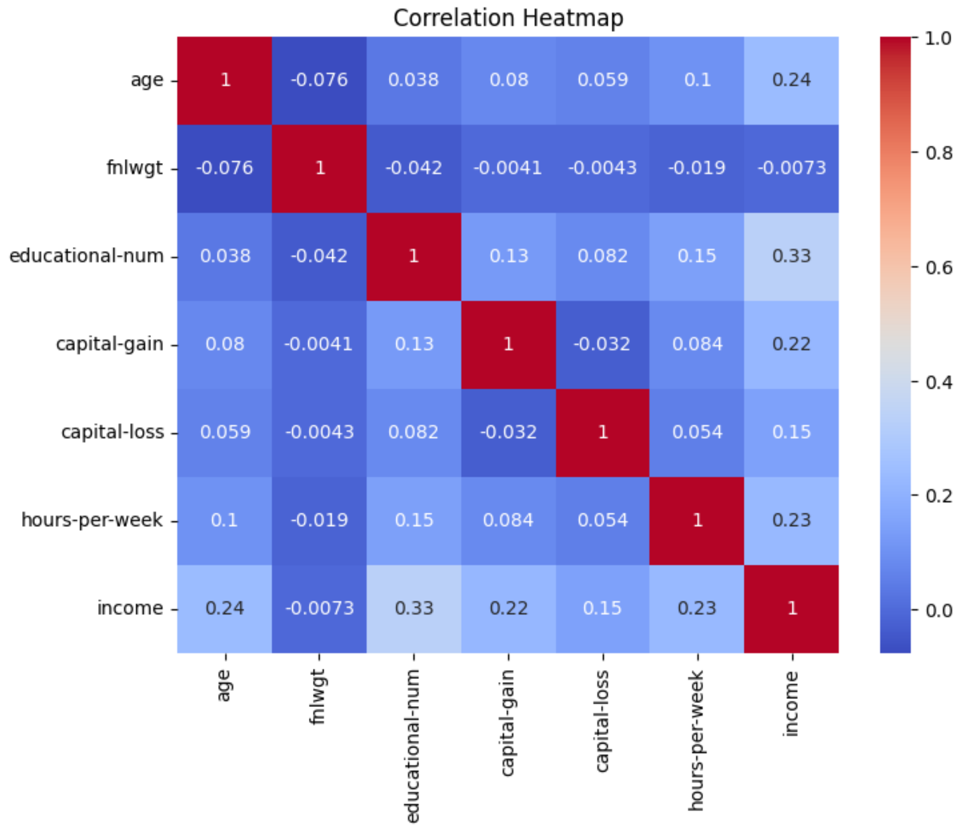



```
plt.figure(figsize=(14,8))
sns.barplot(x = 'marital-status' , y = 'income' , data = df)
plt.xlabel('Marital Status' , size='large')
plt.ylabel('Income' , size = 'large')
plt.title("Income Distribution by Marital Status")
plt.show()
```




```
plt.figure(figsize =(8,6))
sns.heatmap(df.corr() , annot = True , cmap = "coolwarm")
plt.title("Correlation Heatmap")
```

```
<ipython-input-112-86ab77d31fb3>:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ve
  sns.heatmap(df.corr() , annot = True , cmap = "coolwarm")
Text(0.5, 1.0, 'Correlation Heatmap')
```



Data Cleaning

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

```
age               0
workclass         0
fnlwgt            0
education         0
educational-num   0
marital-status    0
occupation        0
relationship      0
race              0
gender            0
capital-gain      0
capital-loss      0
hours-per-week    0
native-country    0
income            0
dtype: int64
```

```
df['workclass'].unique()
```

```
array(['Private', 'Local-gov', '?', 'Self-emp-not-inc', 'Federal-gov',
       'State-gov', 'Self-emp-inc', 'Without-pay', 'Never-worked'],
      dtype=object)
```

```
for x in df.index:
  if df.loc[x , 'workclass'] == '?' :
    df.drop(x , inplace=True)
df['workclass'].unique()
```

```
array(['Private', 'Local-gov', 'Self-emp-not-inc', 'Federal-gov',
       'State-gov', 'Self-emp-inc', 'Without-pay', 'Never-worked'],
      dtype=object)
```

```
df['occupation'].unique()
```

```
array(['Machine-op-inspct', 'Farming-fishing', 'Protective-serv',
       'Other-service', 'Prof-specialty', 'Craft-repair', 'Adm-clerical',
       'Exec-managerial', 'Tech-support', 'Sales', 'Priv-house-serv',
```

```
       'Transport-moving', 'Handlers-cleaners', 'Armed-Forces', '?'],
      dtype=object)
```

```
for x in df.index:
  if df.loc[x , 'occupation'] == '?' :
    df.drop(x , inplace=True)
```

```
df['native-country'].unique()
```

```
array(['United-States', '?', 'Peru', 'Guatemala', 'Mexico',
       'Dominican-Republic', 'Ireland', 'Germany', 'Philippines',
       'Thailand', 'Haiti', 'El-Salvador', 'Puerto-Rico', 'Vietnam',
       'South', 'Columbia', 'Japan', 'India', 'Cambodia', 'Poland',
       'Laos', 'England', 'Cuba', 'Taiwan', 'Italy', 'Canada', 'Portugal',
       'China', 'Nicaragua', 'Honduras', 'Iran', 'Scotland', 'Jamaica',
       'Ecuador', 'Yugoslavia', 'Hungary', 'Hong', 'Greece',
       'Trinadad&Tobago', 'Outlying-US(Guam-USVI-etc)', 'France',
       'Holand-Netherlands'], dtype=object)
```

```
for x in df.index:
  if df.loc[x , 'native-country'] == '?' :
    df.drop(x , inplace=True)
```

Data Separation

```
y = df['income']
x = df.drop(columns='income')
```

```
x
```

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	
5	34	Private	198693	10th	6	Never-married	Other-service	Not-in-family	White	Male	
...	...	...	...	...	...	...	...	...	...	...	
48837	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	
48838	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	
48839	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	
48840	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	
48841	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	

45222 rows × 14 columns

```
y
```

```
0         <=50K
1         <=50K
2          >50K
3          >50K
5         <=50K
          ...
48837     <=50K
48838      >50K
48839     <=50K
48840     <=50K
48841      >50K
Name: income, Length: 45222, dtype: object
```

Data Spliting

```
from sklearn.model_selection import train_test_split
x_train , x_test , y_train , y_test = train_test_split(x , y , test_size = 0.2 , random_state = 100)
```

```
x_train.shape
```

```
(36177, 14)
```

```
x_test.shape
```

```
(9045, 14)
```

Data Encoding

```
df.dtypes
```

```
age                 int64
workclass          object
fnlwgt              int64
education          object
educational-num     int64
marital-status     object
occupation         object
relationship       object
race               object
gender             object
capital-gain        int64
capital-loss        int64
hours-per-week      int64
native-country     object
income             object
dtype: object
```

```
df.columns
```

```
Index(['age', 'workclass', 'fnlwgt', 'education', 'educational-num',
       'marital-status', 'occupation', 'relationship', 'race', 'gender',
       'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
       'income'],
      dtype='object')
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['income'] = le.fit_transform(df['income'])
df['income']
```

```
0        0
1        0
2        1
3        1
5        0
        ..
48837    0
48838    1
48839    0
48840    0
48841    1
Name: income, Length: 45222, dtype: int64
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

x_train['workclass'] = le.fit_transform(x_train['workclass'])
x_test['workclass'] = le.transform(x_test['workclass'])

x_train['education'] = le.fit_transform(x_train['education'])
x_test['education'] = le.transform(x_test['education'])

x_train['marital-status'] = le.fit_transform(x_train['marital-status'])
x_test['marital-status'] = le.transform(x_test['marital-status'])

x_train['occupation'] = le.fit_transform(x_train['occupation'])
x_test['occupation'] = le.transform(x_test['occupation'])

x_train['relationship'] = le.fit_transform(x_train['relationship'])
x_test['relationship'] = le.transform(x_test['relationship'])

x_train['race'] = le.fit_transform(x_train['race'])
x_test['race'] = le.transform(x_test['race'])

x_train['gender'] = le.fit_transform(x_train['gender'])
x_test['gender'] = le.transform(x_test['gender'])

x_train['native-country'] = le.fit_transform(x_train['native-country'])
x_test['native-country'] = le.transform(x_test['native-country'])
```

Model phase

Logistic Regression model

Build the model

```
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression()
```

Train the model

```
lr_model.fit(x_train , y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

Make predicitons

```
y_lr_model_pred = lr_model.predict(x_test)
```

Evaluate the model

```
from sklearn.metrics import confusion_matrix
lr_results = confusion_matrix(y_test , y_lr_model_pred)
lr_results
```

```
array([[6448,  367],
       [1595,  635]])
```

```
def accuracy_percentage_from_confusion_matrix(confusion_matrix):
  true_positive , false_positive = confusion_matrix[0]
  false_negative , true_negative = confusion_matrix[1]
  accuracy = (true_positive + true_negative) / (true_positive + true_negative + false_positive + false_negative)

  #convert the accuracy_score to a percentage
  lr_accuracy_result = accuracy * 100
  return lr_accuracy_result

confusion_matrix = ([6448,  367],
                    [1595,  635])

lr_accuracy_result = accuracy_percentage_from_confusion_matrix(confusion_matrix)
print(f'Accuracy: {lr_accuracy_result:.2f}%')
```

```
Accuracy: 78.31%
```

Confusion Matrix Visualization

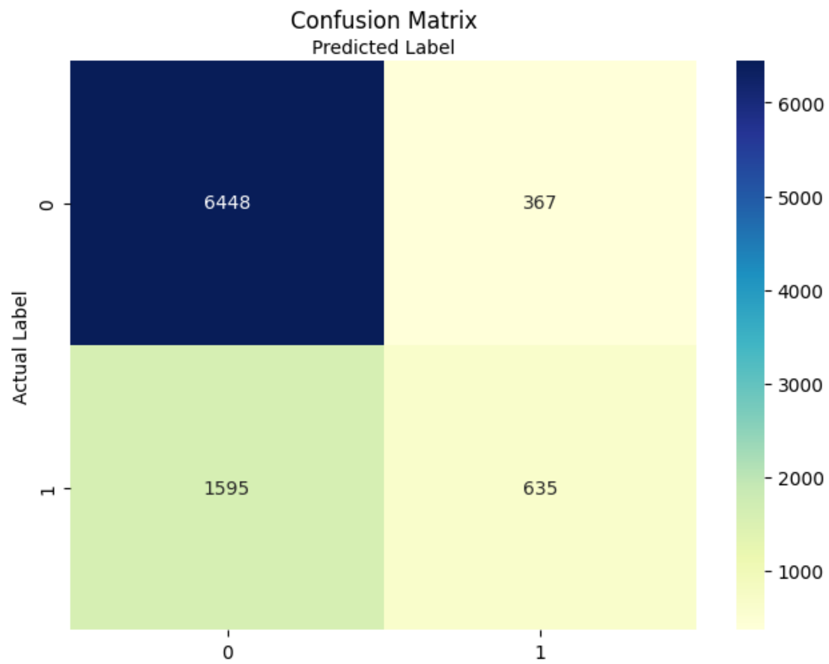
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

class_categories = [0,1]
fig , ax = plt.subplots()
tick_marks = np.arange(len(class_categories))
plt.xticks(tick_marks , class_categories)
plt.yticks(tick_marks , class_categories)

confusion_matrix = np.array([[6448,  367],
                            [1595,  635]])

#Creating the heatmap
sns.heatmap(pd.DataFrame(lr_results) , annot = True , cmap = "YlGnBu" , fmt='g')
ax.xaxis.set_label_position('top')
plt.tight_layout()
```

```
plt.title('Confusion Matrix')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.show()
```



Decision Tree model

```
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier()
```

```
dt_model.fit(x_train , y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
y_dt_model_pred = lr_model.predict(x_test)
```

```
from sklearn.metrics import accuracy_score
result1 = accuracy_score(y_test , y_dt_model_pred)
result1 = result1*100
dt_accuracy_result = pd.DataFrame(['Decision Tree' , result1]).transpose()
dt_accuracy_result.columns=['Model' , "Accuracy_percentage"]
dt_accuracy_result
```

	Model	Accuracy_percentage
0	Decision Tree	78.308458

Random Forest model

```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier()
```

```
rf_model.fit(x_train , y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
y_rf_model_pred = rf_model.predict(x_test)
```

```
from sklearn.metrics import accuracy_score
result2 = accuracy_score(y_test , y_rf_model_pred)
result2 = result2*100
```

```
rf_accuracy_result = pd.DataFrame(["Random Forest" , result2]).transpose()
rf_accuracy_result.columns=['Model' , 'Accuracy_percentage']
rf_accuracy_result
```

	Model	Accuracy_percentage
0	Random Forest	85.936982

Conclusion

Logistic Regression :

Logistic Regression is a linear model that works well for binary classification tasks like predicting income categories (>50k or <=50k). -The model has **78.31%** accuracy percentage

Decision Tree :

Decision Tree is a non-linear model that can capture complex relationships in the data. -The model has **78.30%** accuracy percentage which is similar to the logistic regression model

Random Forest :

Random Forest is an ensemble of Decision Trees that can improve predictive performance and reduce overfitting. It tends to provide better predictive accuracy compared to a single Decision Tree. Random Forest can capture complex interactions between features and is a robust choice for predicting income categories. -The model has **85.68%** accuracy percentage which is better than both decision tree and logistic regression models