**Census Income Project in Jupiter notebook**

Firstly I have open a new Jupiter notebook for the Census Income Python Project, and named it as Census Income. I started import pandas as pd, import numpy as np, import seaborn as sns,

import matplotlib.pyplot as plt, from sklearn.preprocessing ,

import StandardScaler from sklearn.model\_selection,

import train\_test\_split, GridSearchCV ,from sklearn.metrics,

import mean\_squared\_error, r2\_score

from sklearn import svm

from sklearn.metrics import accuracy\_score

import warnings

warnings.filterwarnings('ignore').

As we import these terms then we have to use a csv file for the DataFrame,

We use this csv file from the Github link which provided by our internship team [FlipRoboTechnologies] .

url=("https://raw.githubusercontent.com/FlipRoboTechnologies/ML\_-Datasets/main/Census%20Income/Census%20Income.csv")

df = pd.read\_csv(url)

df

after run these data we had shown a table that has rows and columns for Census Income in which 32560 entries in rows and 15 columns.

Then after we run df.info() in which we got Range index.

After this we run df.isnull().sum() in which we check that any columns have any null

Values.

Then we run df.columns which we have been shown the name of the title index,

Then we run df.columns.tolist() in which the title index names in a vertical form,

After this we run df.dtypes in which we know the details of the title index whether it

will be in int64 or object form.

Then we have to find the number of nulls in the table by run df.isnull().sum().sum().

After that we have to run sns.heatmap(df.isnull()) to a heat map for the details.

Then we want to know about the Ages in the table so we run df['Age'].unique(),

Then we run df['Age'].nunique() for the counts of no of ages in the table.

We want to the details of each title name in the table so run it for i in df.columns:

print (df[i].value\_counts())

print("\n").

we have to run it df.iloc[488,:],

after that we run this if len(df) > 488:

print(df.iloc[488, :])

else:

print("Index 488 is out of bounds for the DataFrame.").

then we run it categorical\_col=[]

for i in df.dtypes.index:

if df.dtypes[i]=="object":

categorical\_col.append(i)

print("Categorical columns:",categorical\_col)

print("\n")

numerical\_col=[]

for i in df.dtypes.index:

if df.dtypes[i]!="object":

numerical\_col.append(i)

print("Numerical Columns:",numerical\_col) for numerical and categorical

column, after this we have to run df.nunique().to\_frame("no. of unique values") for

the number of unique items in the table.

We want the head of the table so we run df.head(), then we run df.describe() for the details of counts , mean , std, min , max, 25%,50%,75%.

Thenafter we run DATA Visualization Univariate Analysis in which we know the

countplot by print

print(df['Age'].value\_counts())

ax=sns.countplot(x='Age',data=df)

plt.show() .

after that we run many countplot related to the title of the table that required and get the Data Visualization analysis for this.

After this we run plt.figure(figsize=(10,6),facecolor='white')

plotnumber=1

for col in numerical\_col:

if plotnumber <=4:

ax=plt.subplot(2,2,plotnumber)

sns.distplot(df[col],color="m")

plt.xlabel(col,fontsize=12)

plt.yticks(rotation=0,fontsize=10)

plotnumber+=1

plt.tight\_layout()

After this we would like to do UNIVARIATE Analysis.

So we have to run

import matplotlib.pyplot as plt

import seaborn as sns

plt.title("comparison between Age and Workclass")

sns.stripplot(x="Age",y="Workclass",data=df)

plt.show()

We show the stripplot, catplot, scatterplot and barplot for the title that we have in the table like ['Workclass', 'Education', 'Marital\_status', 'Occupation', 'Relationship', 'Race', 'Sex', 'Native\_country', 'Income','Age', 'Fnlwgt', 'Education\_num', 'Capital\_gain', 'Capital\_loss', 'Hours\_per\_week'].

After this we have the conclusion which is required.

("The Census Income dataset, also known as the 'Adult' dataset, is sourced from the UCI Machine Learning Repository. It contains 48,842 instances, "

"with 14 attributes such as age, education level, marital status, occupation, and race. The target variable is 'income', indicating whether an individual's "

"income is '>50K' or '<=50K'.

Understanding the structure and characteristics of this dataset is crucial for effective data preprocessing and model building.")

("Data preprocessing is a critical step in any machine learning project. It involves cleaning the dataset, handling missing values, encoding categorical "

"variables, and scaling numerical features. For the Census Income dataset, missing values are replaced with the mode of the respective columns. Categorical "

"features are encoded using One-Hot Encoding, and numerical features are scaled to ensure they contribute equally to the model.")

("Exploratory Data Analysis involves visualizing and summarizing the dataset to uncover patterns and insights. For the Census Income dataset, data "

"distributions are visualized using histograms and box plots. Correlation matrices are used to identify relationships between features. EDA helps in "

"understanding the data better and guides feature selection and engineering.")

("Various machine learning algorithms can be employed to predict income. Commonly used algorithms include Logistic Regression, Decision Trees, Random Forests, "

"and Support Vector Machines. Each algorithm has its strengths and weaknesses, and the choice of algorithm depends on the dataset characteristics and "

"project requirements. Models are trained on the preprocessed dataset, and their performance is evaluated using accuracy, precision, recall, and F1-score.")

("Class imbalance is a common issue in classification problems where the classes are not represented equally. The Census Income dataset has an imbalance, "

"with more instances of '<=50K' compared to '>50K'. Techniques such as SMOTE (Synthetic Minority Over-sampling Technique) are applied to balance the class "

"distribution by generating synthetic samples for the minority class.")

('Model Evaluation and Tuning')

("Evaluating the performance of machine learning models is crucial for ensuring their effectiveness. Performance metrics such as accuracy, precision, recall, "

"and F1-score are used to evaluate models. Cross-validation is employed to validate the model's performance on different subsets of the data. Hyperparameter "

"tuning is performed to optimize the model parameters and improve performance.)

('Deploying the Model')

("Once a machine learning model is trained and validated, it can be deployed for practical use. The model is saved using joblib or pickle, and a simple "

"prediction interface is built to allow users to input new data and receive income predictions. Deployment ensures the model can be used in real-world "

"applications, providing valuable insights and predictions.")

('Conclusion')

("The Census Income project demonstrates the practical application of machine learning in socio-economic data analysis. Through data preprocessing, "

"exploratory data analysis, model building, and deployment, we gain insights into factors influencing income levels. Future work can focus on improving "

"model accuracy by incorporating additional features and advanced algorithms. The project's findings can inform policy decisions and contribute to economic studies.")

Converting the target variable income to binary 0/1

We want an int64 term by using this to use in binary 0/1

df['Hours\_per\_week']=df['Hours\_per\_week'].apply(lambda x: 0 if x =='<=10' else 1)

df['Hours\_per\_week'].head()

Converting categorical variables to numerical variables using LabelEncoder()

We use LabelEncoder() to use this

from sklearn import preprocessing

categorical=['Workclass','Education','Marital\_status','Occupation','Relationship','Sex','Native\_country']

for feature in categorical:

le=preprocessing.LabelEncoder()

X\_train[feature]=le.fit\_transform(X\_train[feature])

X\_test[feature]=le.transform(X\_test[feature])

Then we have to find scale using StandardScaler()

We run this syntax

from sklearn.preprocessing import LabelEncoder

categorical\_cols = ['Workclass', 'Education', 'Marital\_status', 'Occupation', 'Relationship', 'Sex', 'Native\_country']

label\_encoder = LabelEncoder()

for col in categorical\_cols:

X\_train[col] = label\_encoder.fit\_transform(X\_train[col])

X\_test[col] = label\_encoder.transform(X\_test[col])

And run this syntax

scaler=StandardScaler()

X\_train=pd.DataFrame(scaler.fit\_transform(X\_train),columns=X.columns)

X\_test=pd.DataFrame(scaler.fit\_transform(X\_test),columns=X.columns)

After that this

X\_train.head()

After this we have use MODELTRAINING

Using Logistic Regression

X\_train\_encoded = pd.get\_dummies(X\_train)

X\_test\_encoded = pd.get\_dummies(X\_test)

logreg = LogisticRegression()

logreg.fit(X\_train\_encoded, y\_train)

y\_pred = logreg.predict(X\_test\_encoded)

from sklearn.metrics import accuracy\_score

accuracy = accuracy\_score(y\_test, y\_pred)

print('Logistic Regression accuracy score with all the features: {0:0.4f}'.format(accuracy))

we got the **Logistic Regression accuracy score with all the features: 0.7948**

with this

from sklearn.metrics import classification\_report

print(classification\_report(y\_test, y\_pred))

After this we have use **Evaluation metrics**

in which we run from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred, pos\_label=' >50K')

recall = recall\_score(y\_test, y\_pred, pos\_label=' >50K')

f1 = f1\_score(y\_test, y\_pred, pos\_label=' >50K')

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1 Score:", f1)

as a result this term is showing

Accuracy: 0.7948402948402948

Precision: 0.6537698412698413

Recall: 0.2847882454624028

F1 Score: 0.3967489464178206

After this we run this syntax

X\_test\_encoded = pd.get\_dummies(X\_test)

missing\_cols = set(X\_train\_encoded.columns) - set(X\_test\_encoded.columns)

for col in missing\_cols:

X\_test\_encoded[col] = 0

X\_test\_encoded = X\_test\_encoded[X\_train\_encoded.columns]

y\_prob = logreg.predict\_proba(X\_test\_encoded)[:, 1]

**ROC Curve**

We run this syntax

from sklearn.preprocessing import LabelBinarizer

from sklearn.metrics import roc\_curve

label\_binarizer = LabelBinarizer()

y\_test\_binary = label\_binarizer.fit\_transform(y\_test)

fpr, tpr, thresholds = roc\_curve(y\_test\_binary, y\_prob)

import matplotlib.pyplot as plt

plt.plot(fpr, tpr)

plt.plot([0, 1], [0, 1], linestyle="--")

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.show()

**Decision Tree Classifier**

We use this code

from sklearn.pipeline import Pipeline

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import ColumnTransformer

numeric\_cols = ['age', 'education\_num', 'capital\_gain', 'capital\_loss', 'hours\_per\_week']

categorical\_cols = ['workclass', 'education', 'marital\_status', 'occupation', 'relationship', 'race', 'sex', 'native\_country']

numeric\_transformer = SimpleImputer(strategy='mean')

categorical\_transformer = Pipeline([

('imputer', SimpleImputer(strategy='most\_frequent')),

('onehot', OneHotEncoder(handle\_unknown='ignore'))

])

preprocessor = ColumnTransformer(

transformers=[

('num', numeric\_transformer, numeric\_cols),

('cat', categorical\_transformer, categorical\_cols)

])

pipeline = Pipeline(steps=[('preprocessor', preprocessor)])

then we use this code

print(classification\_report(y\_test,y\_pred))

**Evaluation Metrics**

We run this code

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred, pos\_label=' >50K')

recall = recall\_score(y\_test, y\_pred, pos\_label=' >50K')

f1 = f1\_score(y\_test, y\_pred, pos\_label=' >50K')

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1 Score:", f1)

result is given by this code is

Accuracy: 0.7948402948402948

Precision: 0.6537698412698413

Recall: 0.2847882454624028

F1 Score: 0.3967489464178206

**k-Nearest neighbors**

we run this syntax

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=5)

knn.fit(X\_train, y\_train)

y\_pred=knn.predict(X\_test)

accuracy=accuracy\_score(y\_test,y\_pred)

print('knn accuracy:',accuracy)

with the result

knn accuracy: 0.7722153972153972

we run this

print(classification\_report(y\_test,y\_pred)).

**EVALUATION METRICS**

We run this code

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred, pos\_label=' >50K')

recall = recall\_score(y\_test, y\_pred, pos\_label=' >50K')

f1 = f1\_score(y\_test, y\_pred, pos\_label=' >50K')

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1 Score:", f1)

as a result

Accuracy: 0.7722153972153972

Precision: 0.5336356764928194

Recall: 0.3050993949870354

F1 Score: 0.3882320593896068

After this we run the syntax

from sklearn.preprocessing import LabelBinarizer

from sklearn.metrics import roc\_curve

label\_binarizer = LabelBinarizer()

y\_test\_binary = label\_binarizer.fit\_transform(y\_test)

fpr, tpr, thresholds = roc\_curve(y\_test\_binary, y\_prob)

import matplotlib.pyplot as plt

plt.plot(fpr, tpr)

plt.plot([0, 1], [0, 1], linestyle="--")

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.show()

**SUPPORT VECTOR MACHINE**

We run this code

from sklearn.svm import SVC

svm=SVC()

svm.fit(X\_train,y\_train)

y\_pred=svm.predict(X\_test)

accuracy=accuracy\_score(y\_test,y\_pred)

print('Support Vector Machine accuracy:',accuracy)

with the result

Support Vector Machine accuracy: 0.7965806715806716

**Random Forest Classifier**

We run this code

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf.fit(X\_train, y\_train)

y\_pred = rf.predict(X\_test)

accuracy=accuracy\_score(y\_test,y\_pred)

print('Random Forest accuracy:',accuracy)

As a result

Random Forest accuracy: 0.850839475839475

At last the final conclusion for the Models is

**Random Forest Model performed the best after analyzing the Census Income dataset and training various machine learning models with an accuracy of 85%**