Adult Census Income Prediction

USING MACHINE LEARNING AND DATA ANALYSIS

NAME: ADITYA ANIL PATADE
GUIDED BY: SAMEER WARSOLKAR

Overview of the Project

- **Objective: Predicting whether an individual earns more than $50,000 annually based on census data**
- The Adult Census Income Prediction Project aims to develop a machine learning model to predict whether an individual's income exceeds $50,000 per year based on demographic and employment attributes from the U.S. Census Bureau's Adult dataset. This project involves several key steps, including data preprocessing, exploratory data analysis, feature engineering, model selection, and evaluation.
- Various algorithms, such as Logistic Regression, KNeighbor Classifier, Decision Trees and Random Forests are tested and optimized using techniques like GridSearchCV to find the best hyperparameters. The final model's performance is evaluated using metrics like accuracy, precision, recall, and the F1-score. The goal is to provide insights into the factors influencing income levels and build a robust predictive model for practical applications in fields like economic planning and policy-making.

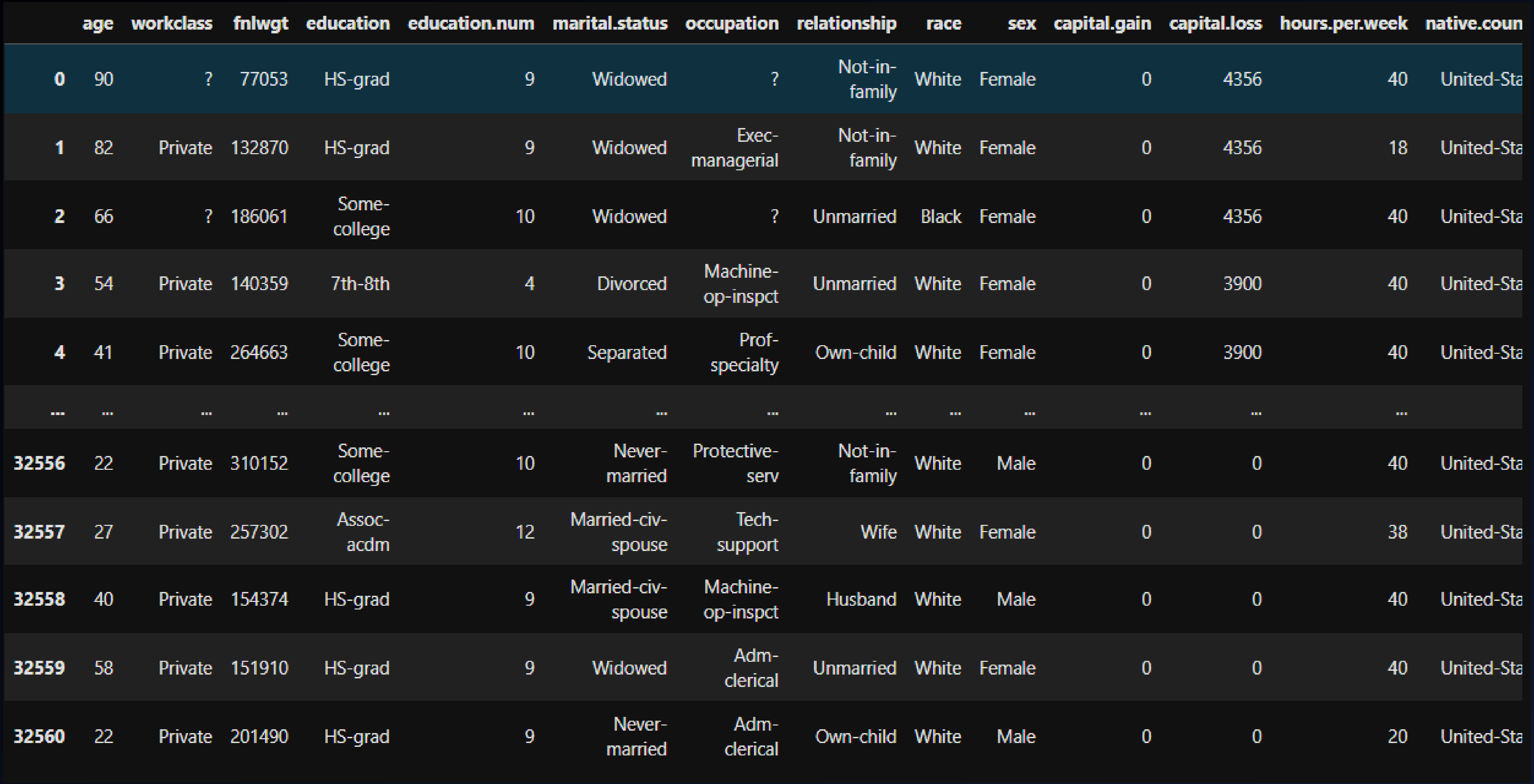
Dataset Overview

- Features: age, workclass, fnlwgt, education, education.num, marital.status, occupation, relationship, race, sex, capital.gain, capital.loss, hours.per.week, native.country
- Target: income (>50K or <=50K)

Attribute Information

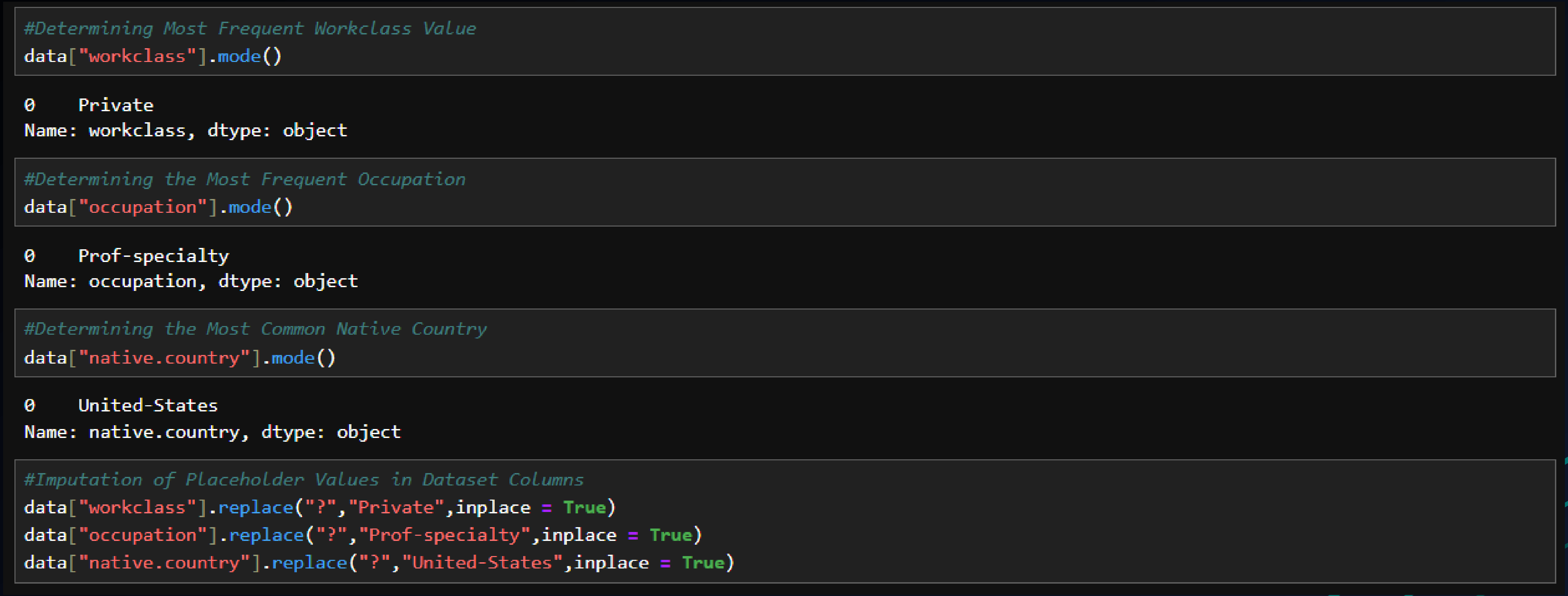
age: continuous – age of a Person
workclass: Where do a person works – categorical -Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
fnlwgt: This weight is assigned by the Current Population Survey (CPS). People with similar demographic characteristics should have similar weights since it is a feature aimed to allocate similar weights to people with similar demographic characteristics – continuous
education: Degree the person has – Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
education-num: no. of years a person studied – continuous.
marital.status: No of people - Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
occupation: people's occupation - Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
relationship: people's relationship- Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
sex: Gender - Female, Male.
capital.gain: Investment gain of the person other than salary – continuous
capital.loss: Loss from investments – continuous
hours.per.week: No. of hours a person works – continuous.
native-country: Individual person country.
income: >50K, <=50K (dependent variable, the salary is in Dollars per year)

Data Preprocessing



	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.coun
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Female	0	4356	40	United-Sta
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	4356	18	United-Sta
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female	0	4356	40	United-Sta
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	3900	40	United-Sta
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	3900	40	United-Sta
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
32556	22	Private	310152	Some-college	10	Never-married	Protective-serv	Not-in-family	White	Male	0	0	40	United-Sta
32557	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	United-Sta
32558	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	United-Sta
32559	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	United-Sta
32560	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	United-Sta

Handling Missing Values



```
#Determining Most Frequent Workclass Value
data["workclass"].mode()
```

```
0    Private
Name: workclass, dtype: object
```

```
#Determining the Most Frequent Occupation
data["occupation"].mode()
```

```
0    Prof-specialty
Name: occupation, dtype: object
```

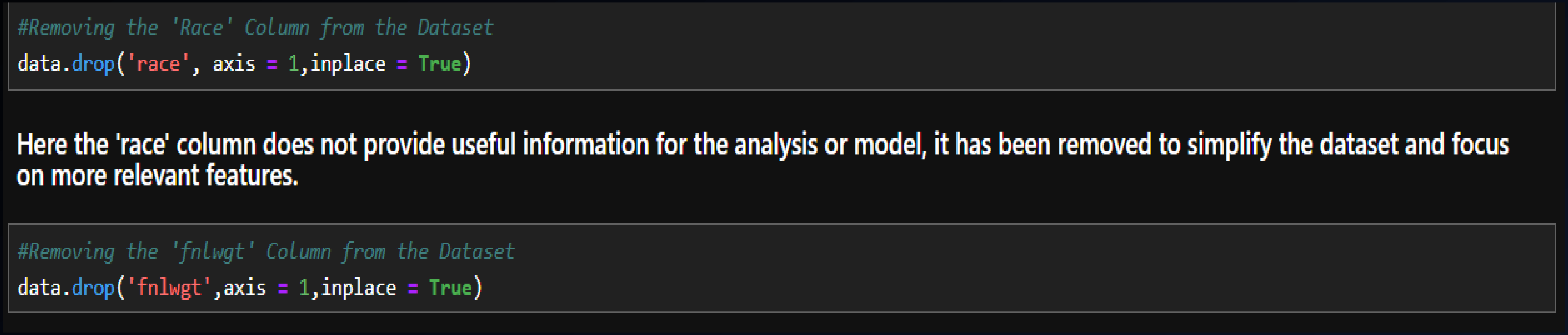
```
#Determining the Most Common Native Country
data["native.country"].mode()
```

```
0    United-States
Name: native.country, dtype: object
```

```
#Imputation of Placeholder Values in Dataset Columns
data["workclass"].replace("?","Private",inplace = True)
data["occupation"].replace("?","Prof-specialty",inplace = True)
data["native.country"].replace("?","United-States",inplace = True)
```

REPLACING ALL INSTANCES OF '?' IN THE "WORKCLASS","OCCUPATION","NATIVE.COUNTRY" COLUMN WITH "PRIVATE",`"PROF-SPECIALTY","UNITED-STATES" RESPECTIVELY WHICH WE HAVE OBTAINED FROM DETERMINING THE MOST FREQUENT VALUE BY USING MODE.THE MODE OF A SET OF VALUES IS THE VALUE THAT APPEARS MOST OFTEN.

Dropping Irrelevant Columns

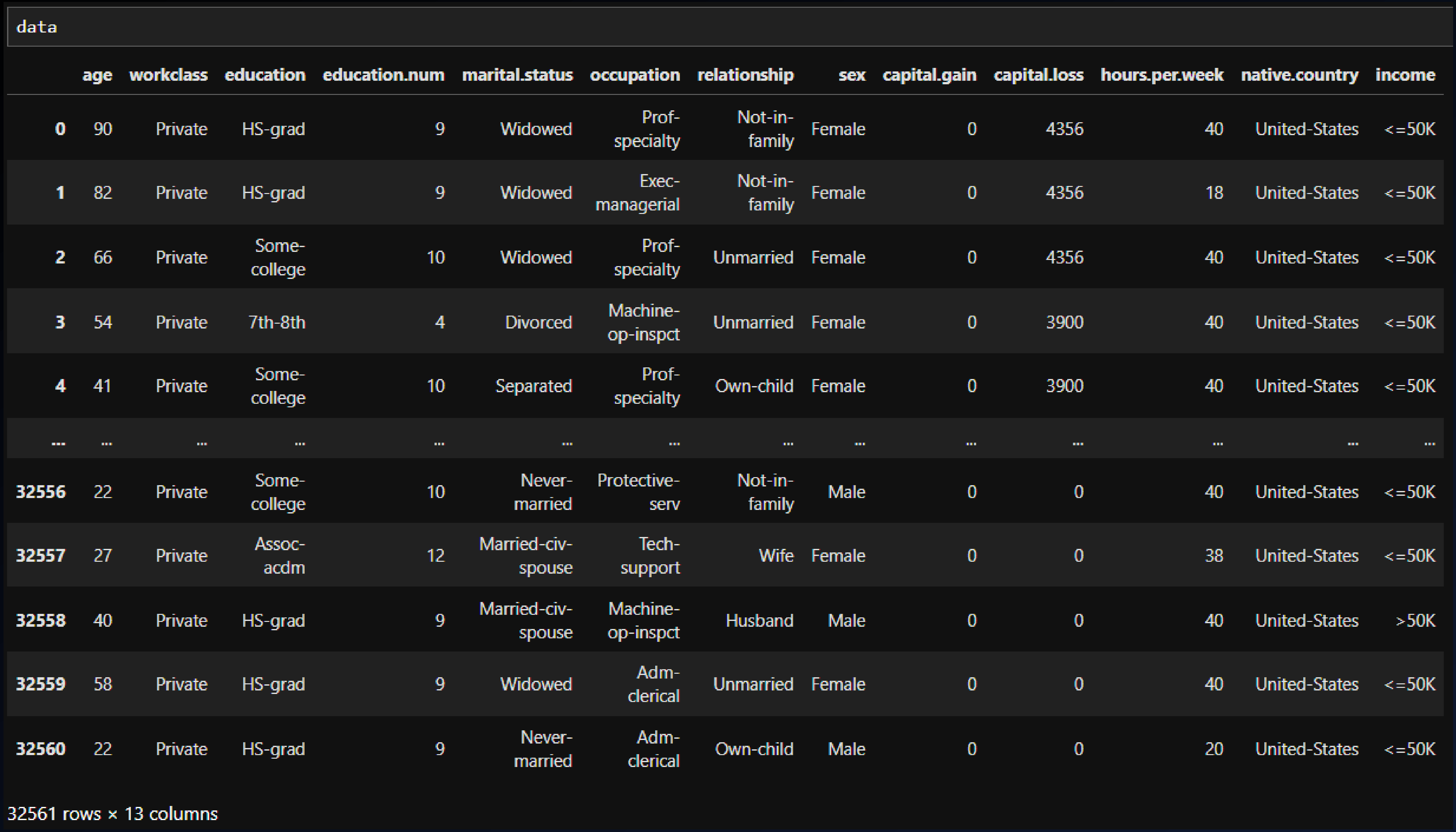


```
#Removing the 'Race' Column from the Dataset
data.drop('race', axis = 1,inplace = True)
```

Here the 'race' column does not provide useful information for the analysis or model, it has been removed to simplify the dataset and focus on more relevant features.

```
#Removing the 'fnlwgt' Column from the Dataset
data.drop('fnlwgt',axis = 1,inplace = True)
```

HERE THE FNLWGT COLUMN AND RACE COLUMN DOES NOT PROVIDE USEFUL INFORMATION FOR THE ANALYSIS OR MODEL, SO IT HAS BEEN REMOVED TO SIMPLIFY THE DATASET AND FOCUS ON MORE RELEVANT FEATURES.¶

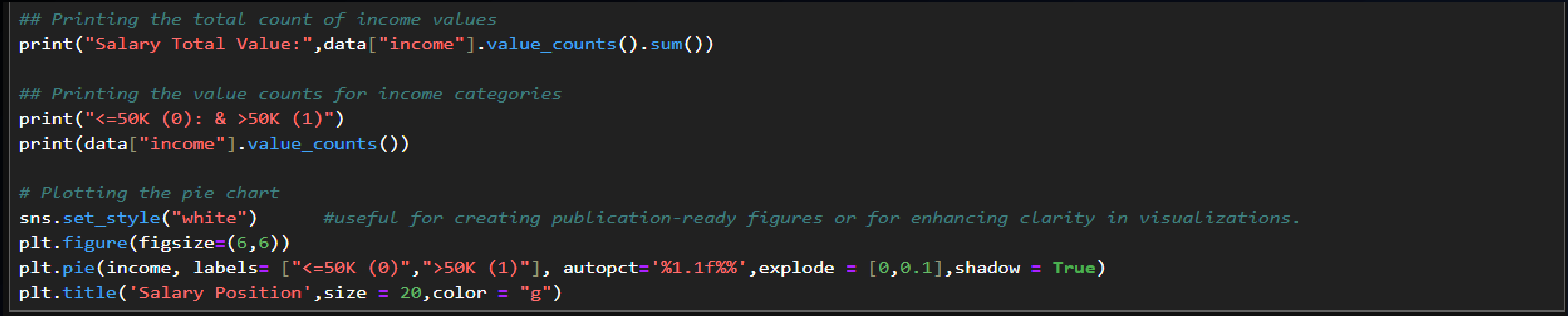


```
data
```

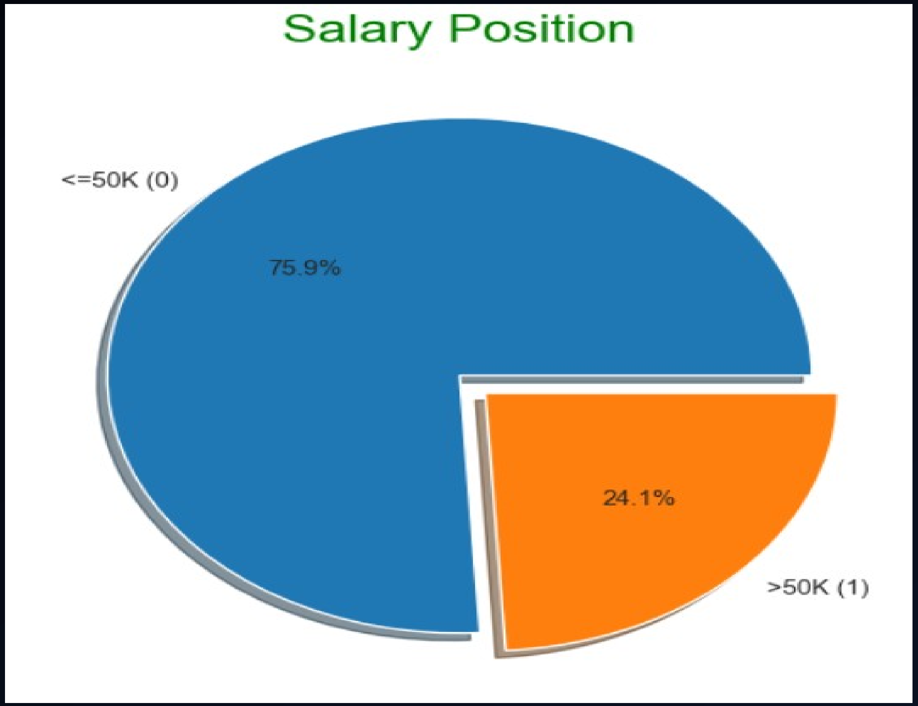
	age	workclass	education	education.num	marital.status	occupation	relationship	sex	capital.gain	capital.loss	hours.per.week	native.country	income
0	90	Private	HS-grad	9	Widowed	Prof-specialty	Not-in-family	Female	0	4356	40	United-States	<=50K
1	82	Private	HS-grad	9	Widowed	Exec-managerial	Not-in-family	Female	0	4356	18	United-States	<=50K
2	66	Private	Some-college	10	Widowed	Prof-specialty	Unmarried	Female	0	4356	40	United-States	<=50K
3	54	Private	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	Female	0	3900	40	United-States	<=50K
4	41	Private	Some-college	10	Separated	Prof-specialty	Own-child	Female	0	3900	40	United-States	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...
32556	22	Private	Some-college	10	Never-married	Protective-serv	Not-in-family	Male	0	0	40	United-States	<=50K
32557	27	Private	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	Female	0	0	38	United-States	<=50K
32558	40	Private	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	Male	0	0	40	United-States	>50K
32559	58	Private	HS-grad	9	Widowed	Adm-clerical	Unmarried	Female	0	0	40	United-States	<=50K
32560	22	Private	HS-grad	9	Never-married	Adm-clerical	Own-child	Male	0	0	20	United-States	<=50K

32561 rows × 13 columns

Data Visualization

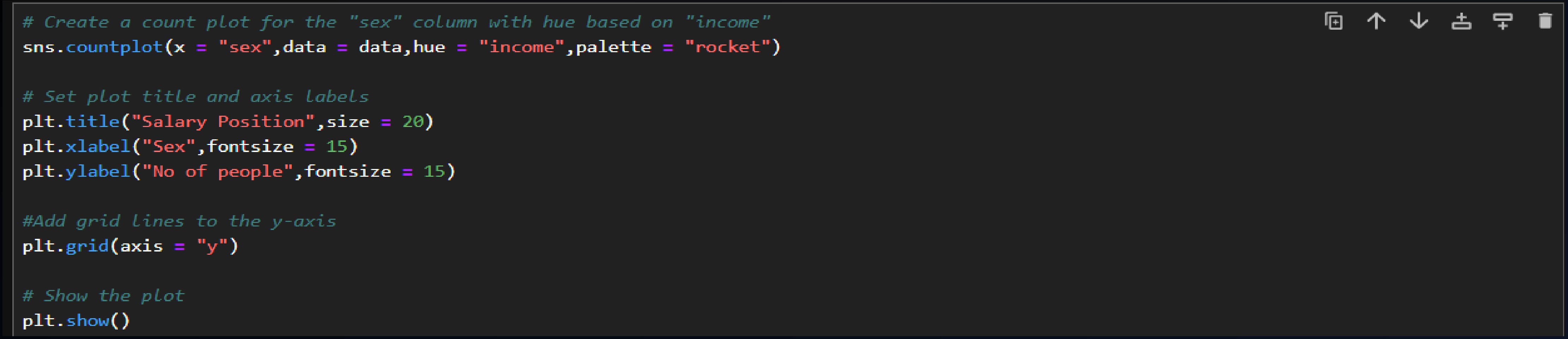


```
## Printing the total count of income values
print("Salary Total Value:",data["income"].value_counts().sum())

## Printing the value counts for income categories
print("<=50K (0): & >50K (1)")
print(data["income"].value_counts())

# Plotting the pie chart
sns.set_style("white")      #useful for creating publication-ready figures or for enhancing clarity in visualizations.
plt.figure(figsize=(6,6))
plt.pie(income, labels= ["<=50K (0)",">50K (1)"], autopct='%1.1f%%',explode = [0,0.1],shadow = True)
plt.title('Salary Position',size = 20,color = "g")
```

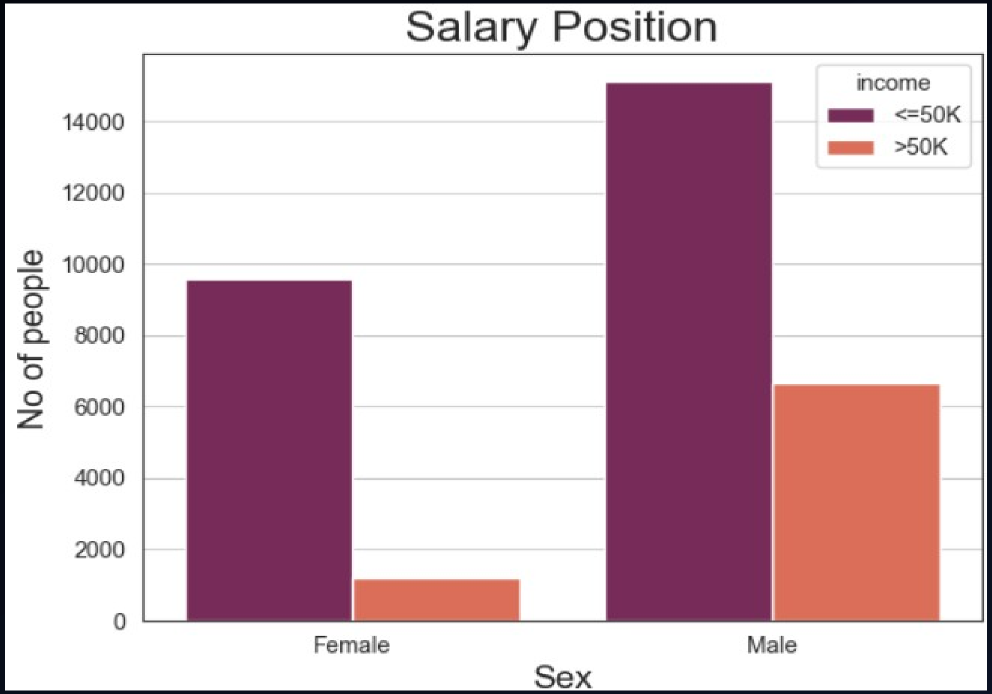


- Out Of total people ,24720 (75.9%) are less than 50K (0) and 7841(24.1%) are greater than 50K (1).From here we can infer that most people had salary or income less than 50K as compared to others.

BASED ON GENDER AND INCOME

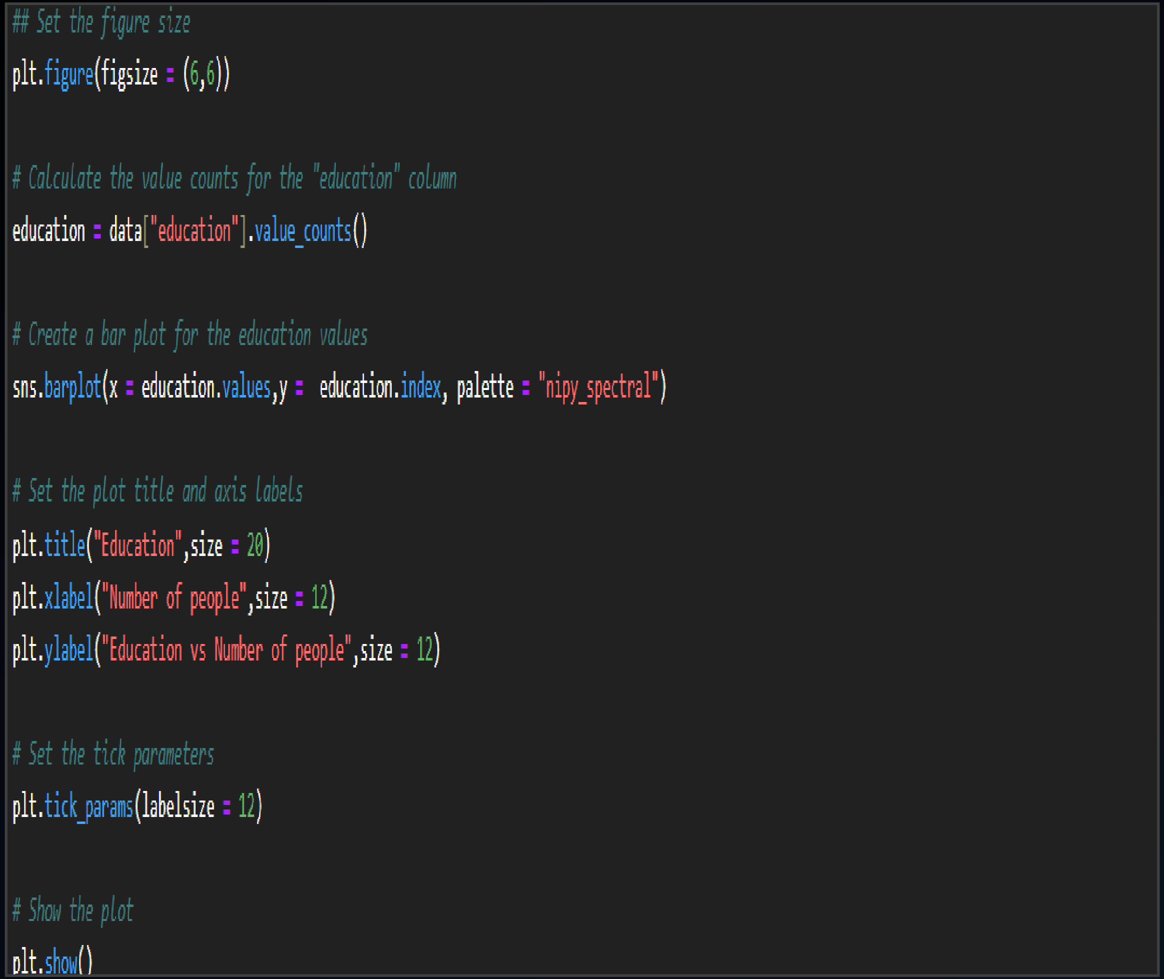


```
# Create a count plot for the "sex" column with hue based on "income"
sns.countplot(x = "sex",data = data,hue = "income",palette = "rocket")

# Set plot title and axis labels
plt.title("Salary Position",size = 20)
plt.xlabel("Sex",fontsize = 15)
plt.ylabel("No of people",fontsize = 15)

#Add grid lines to the y-axis
plt.grid(axis = "y")

# Show the plot
plt.show()
```

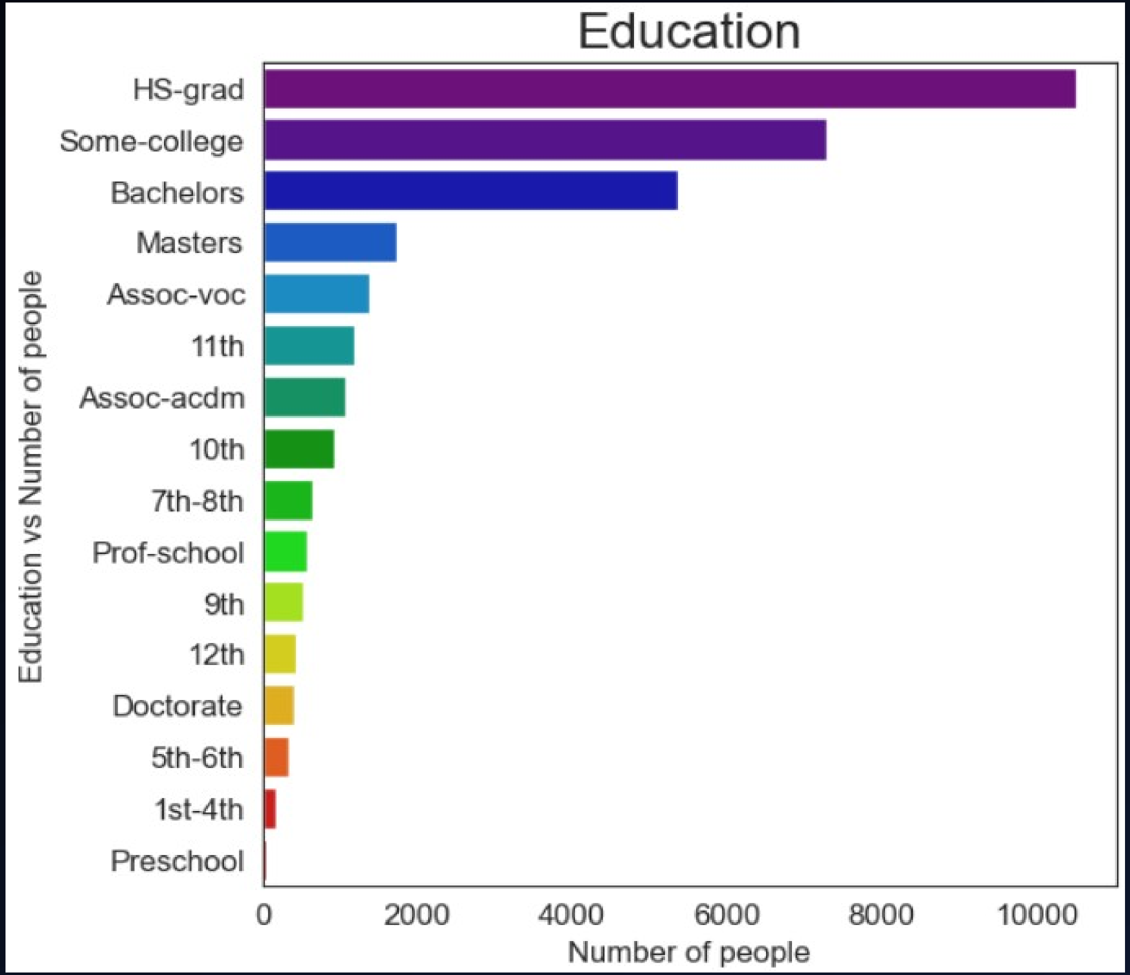


- The figure shows that Males in both income groups earn significantly more than the females respectively

BASED ON EDUCATION

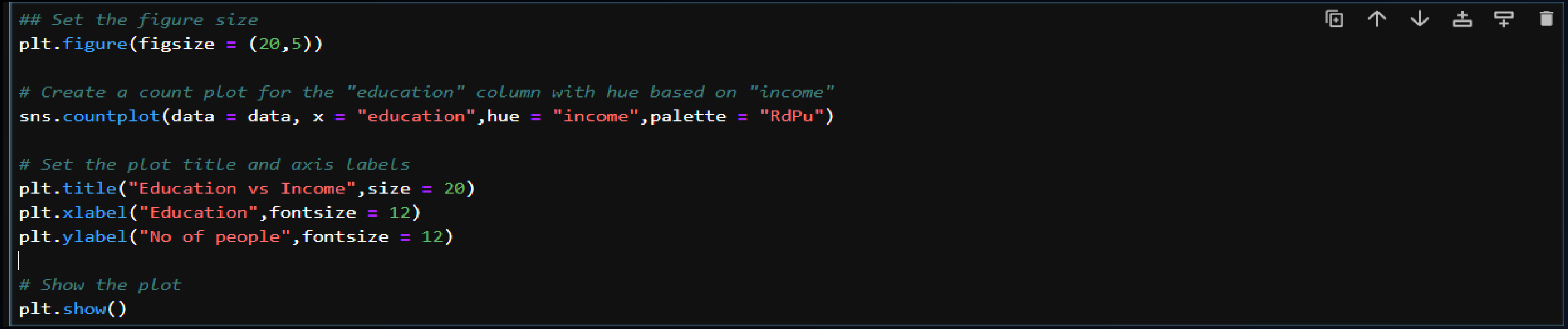


```
## Set the figure size
plt.figure(figsize = (6,6))

# Calculate the value counts for the "education" column
education = data["education"].value_counts()

# Create a bar plot for the education values
sns.barplot(x = education.values,y =  education.index, palette = "nipy_spectral")

# Set the plot title and axis labels
plt.title("Education",size = 20)
plt.xlabel("Number of people",size = 12)
plt.ylabel("Education vs Number of people",size = 12)

# Set the tick parameters
plt.tick_params(labelsize = 12)

# Show the plot
plt.show()
```

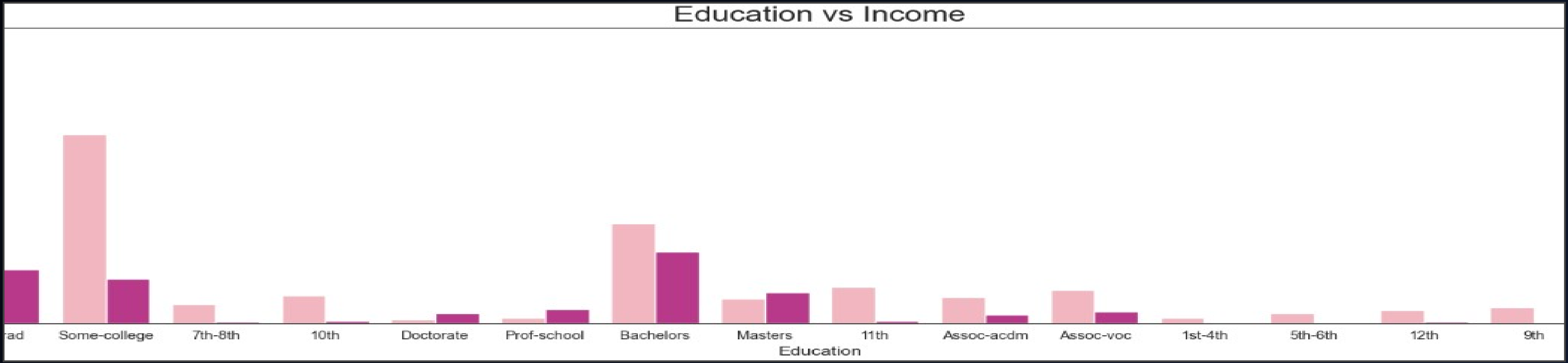


Here most of the people have completed their High School Graduate, Bachelors Degree and have completed there Masters.

BASED ON EDUCATION AND INCOME



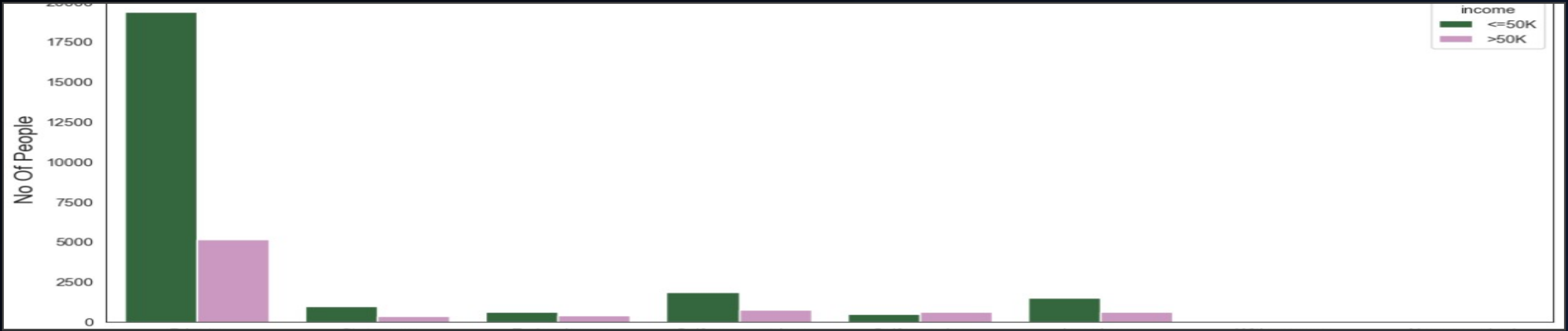
```
## Set the figure size
plt.figure(figsize = (20,5))

# Create a count plot for the "education" column with hue based on "income"
sns.countplot(data = data, x = "education",hue = "income",palette = "RdPu")

# Set the plot title and axis labels
plt.title("Education vs Income",size = 20)
plt.xlabel("Education",fontsize = 12)
plt.ylabel("No of people",fontsize = 12)

# Show the plot
plt.show()
```



From above plot we can conclude that majority of the work class are High school grads, bachelors degree holders and college grads. The people who has Masters, HS-grad, Doctorate, Bachelors, Prof-school tend to be earning more than 50K than the other degrees.

Based On Work class and Income

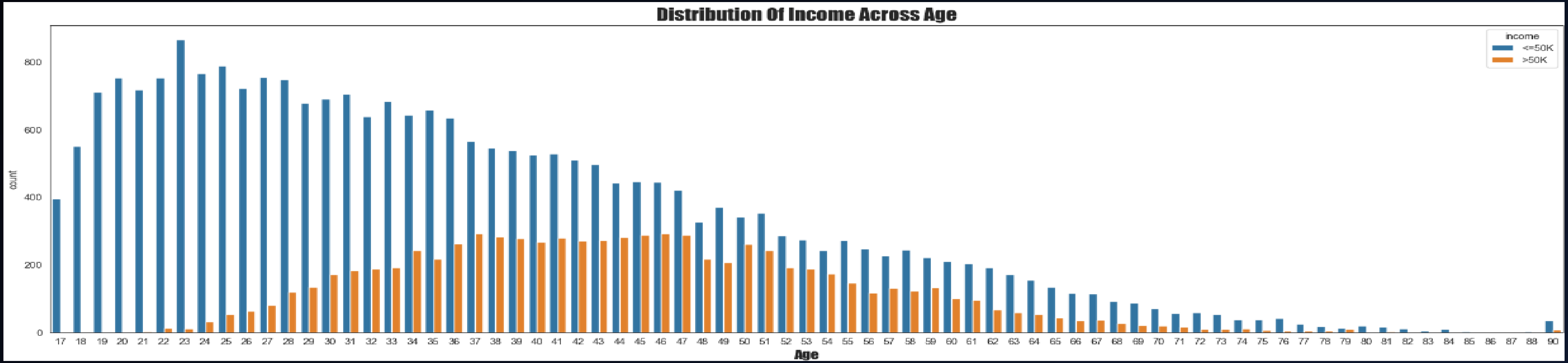
```
# Set the figure size
plt.figure(figsize = (15,6))

# Create a count plot for the "workclass" column with hue based on "income"
sns.countplot(data = data,x = "workclass",hue = "income",palette = "cubehelix")

# Set the plot title and axis labels
plt.title("Based On Workclass",size = 20)
plt.xlabel("Workclass",fontsize = 15)
plt.ylabel("No Of People",fontsize = 15)

# Show the plot
plt.show()
```



From above we can see that 70% of workclass fall under the private and Private sector has the highest number of people who has income greater than USD 50K.

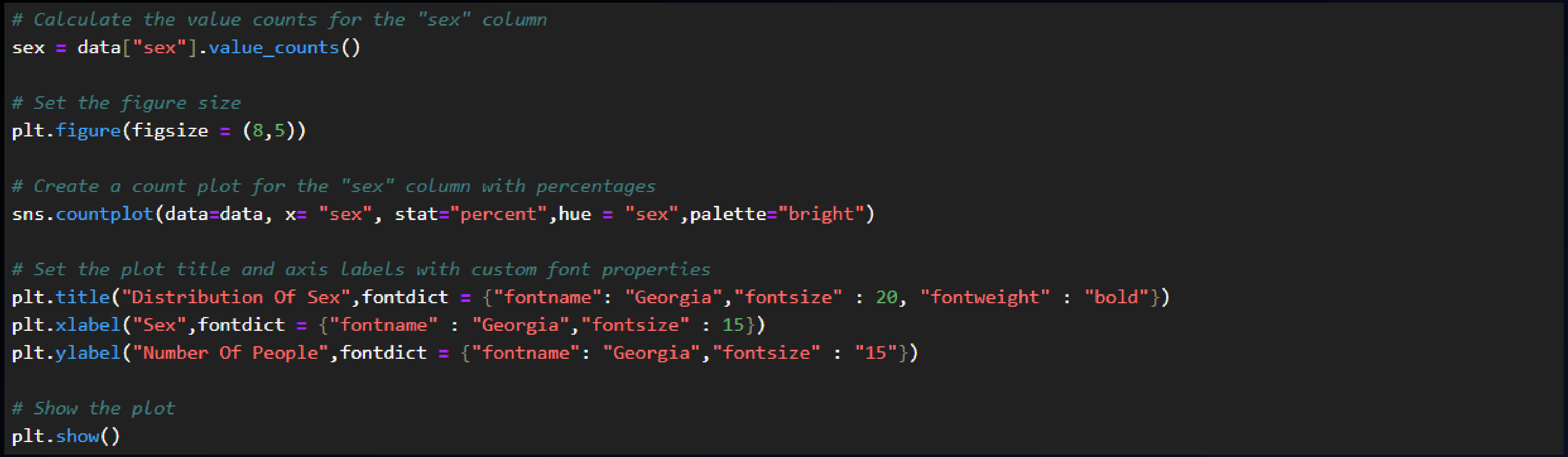
DISTRIBUTION OF INCOME ACROSS AGE

```
# Set the figure size
plt.figure(figsize = (25,7))

# Create a count plot for the "age" column with hue based on "income"
sns.countplot(x = "age",hue = "income",data = data)

# Set the plot title and axis labels with custom font properties
plt.title("Distribution Of Income Across Age",fontdict = {
    "fontname": "fantasy","fontsize": 20,"fontweight": "bold"})
plt.xlabel("Age",fontdict ={"fontname" : "fantasy","fontsize": 15})

# Show the plot
plt.show()
```

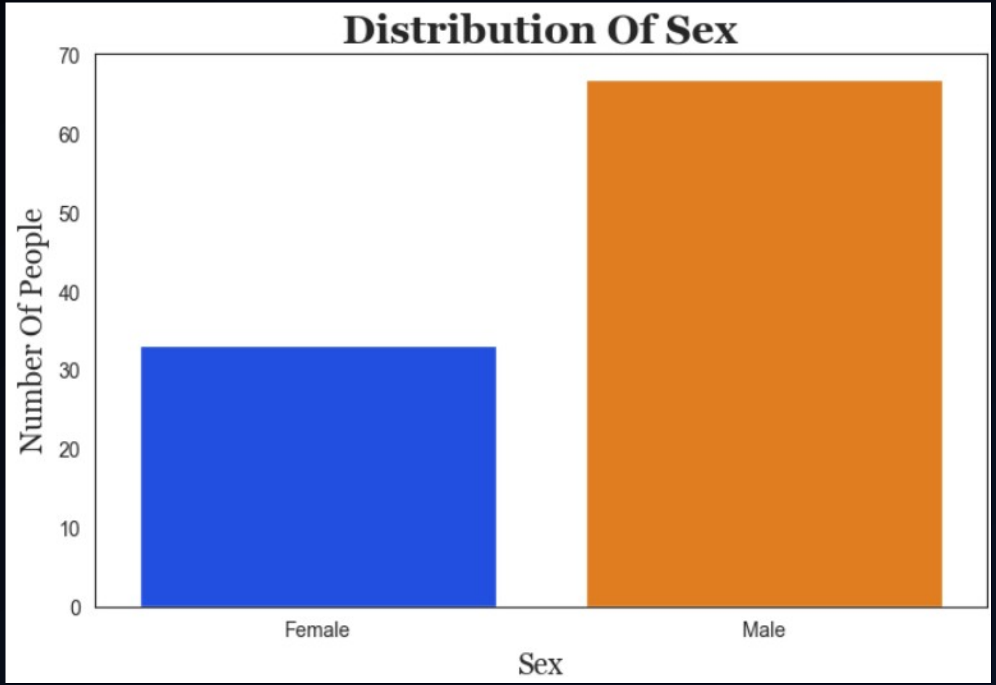


We infer that min salary of most of the population is in their 20's which is a valid as it is the starting time of a career. Here we can conclude that Age and salary are in direct proportion with each other. The data is left skewed as there are a very few people who work after 60's,therefore,the chances of salary being more than 50k is very less.

BASED ON GENDER

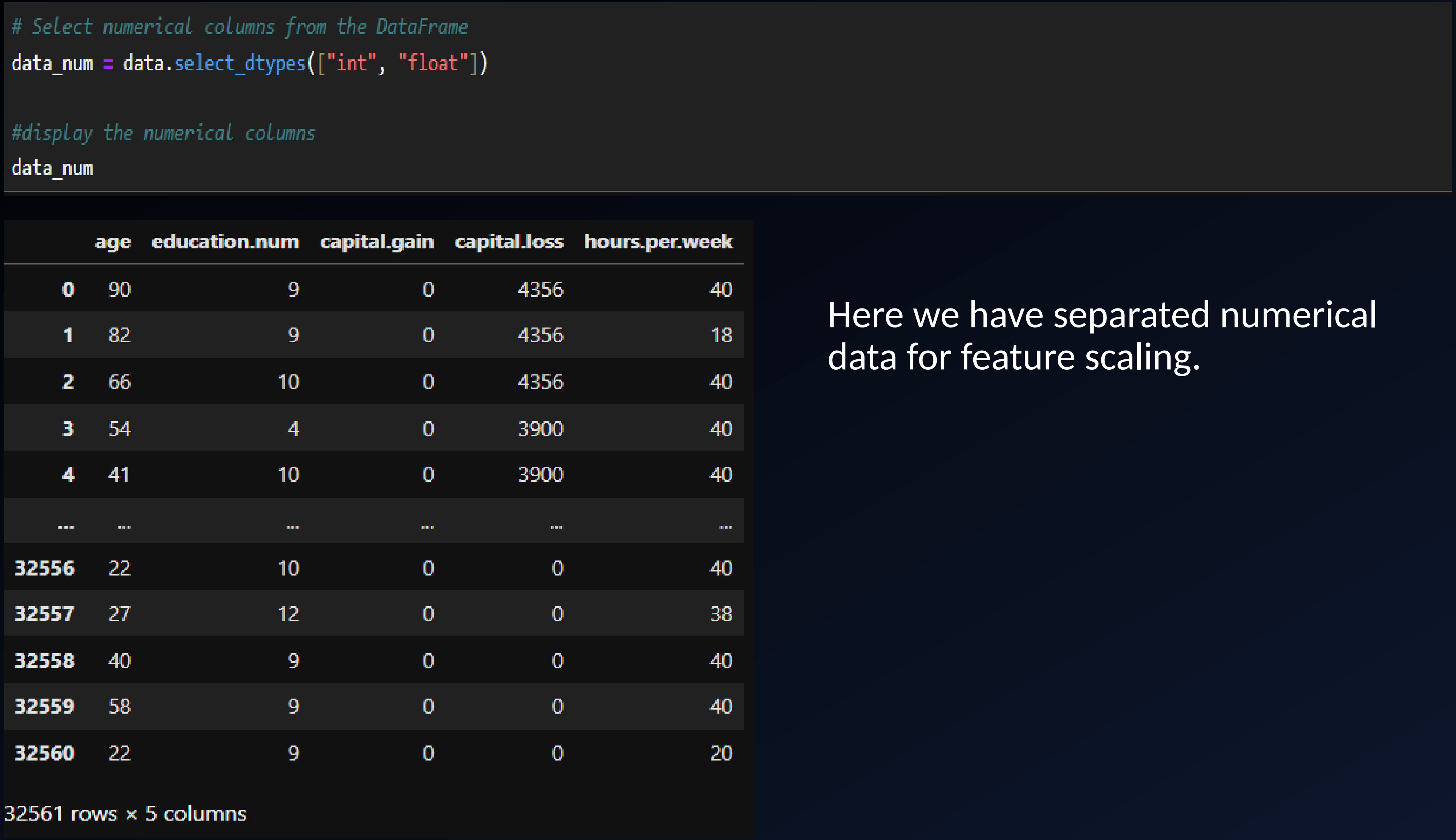


```python
# Calculate the value counts for the "sex" column
sex = data["sex"].value_counts()

# Set the figure size
plt.figure(figsize = (8,5))

# Create a count plot for the "sex" column with percentages
sns.countplot(data=data, x= "sex", stat="percent",hue = "sex",palette="bright")

# Set the plot title and axis labels with custom font properties
plt.title("Distribution Of Sex",fontdict = {"fontname": "Georgia","fontsize" : 20, "fontweight" : "bold"})
plt.xlabel("Sex",fontdict = {"fontname" : "Georgia","fontsize" : 15})
plt.ylabel("Number Of People",fontdict = {"fontname": "Georgia","fontsize" : "15"})

# Show the plot
plt.show()
```



We can see from the graph that 67% of employees are males while only 33% are females.

SEPARATING NUMERICAL & CATEGORICAL DATA



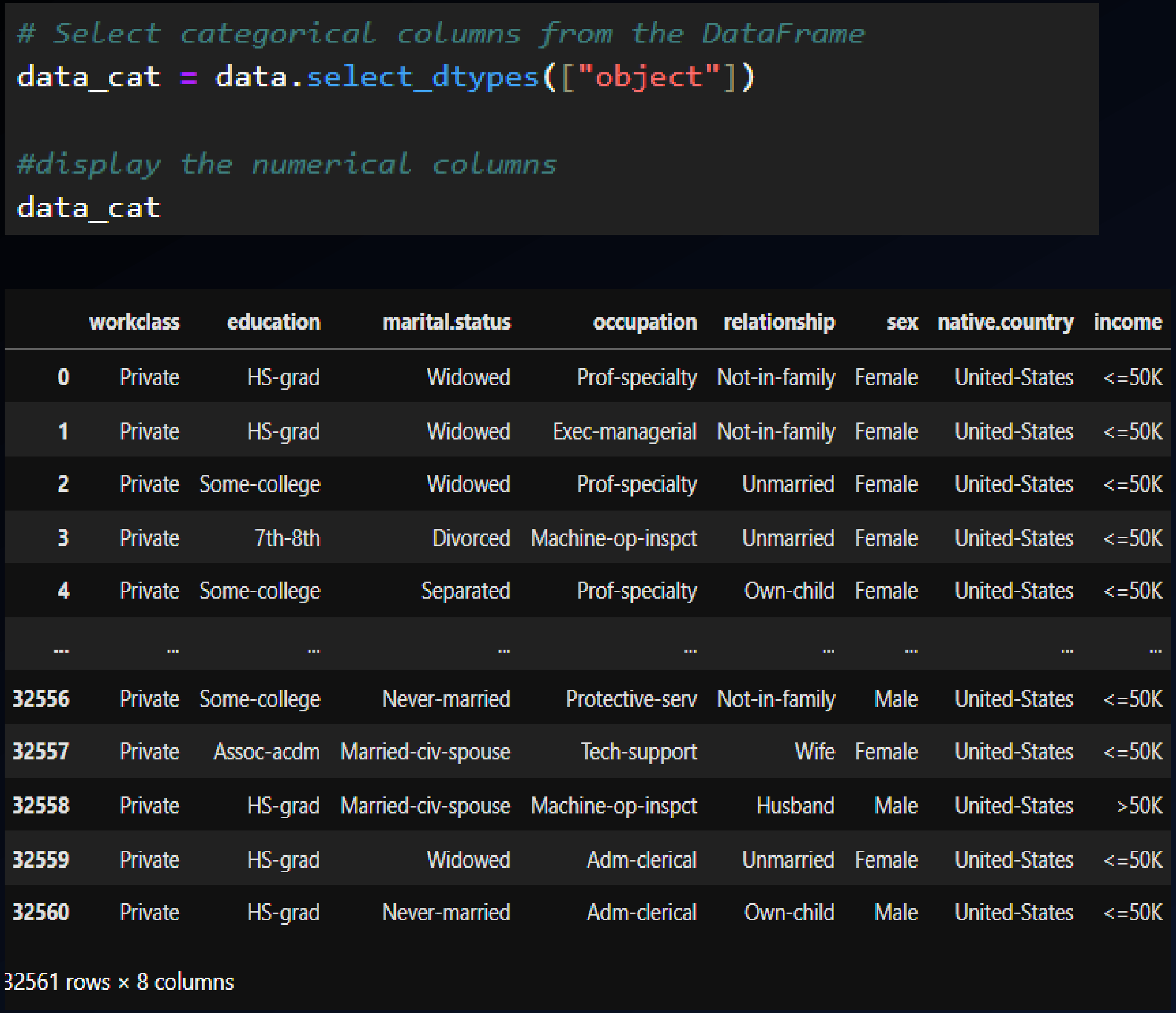
```
# Select numerical columns from the DataFrame
data_num = data.select_dtypes(["int", "float"])

#display the numerical columns
data_num
```

	age	education.num	capital.gain	capital.loss	hours.per.week
0	90	9	0	4356	40
1	82	9	0	4356	18
2	66	10	0	4356	40
3	54	4	0	3900	40
4	41	10	0	3900	40
...	...	...	...	...	...
32556	22	10	0	0	40
32557	27	12	0	0	38
32558	40	9	0	0	40
32559	58	9	0	0	40
32560	22	9	0	0	20

32561 rows × 5 columns

Here we have separated numerical data for feature scaling.

SEPARATING CATEGORICAL COLUMN



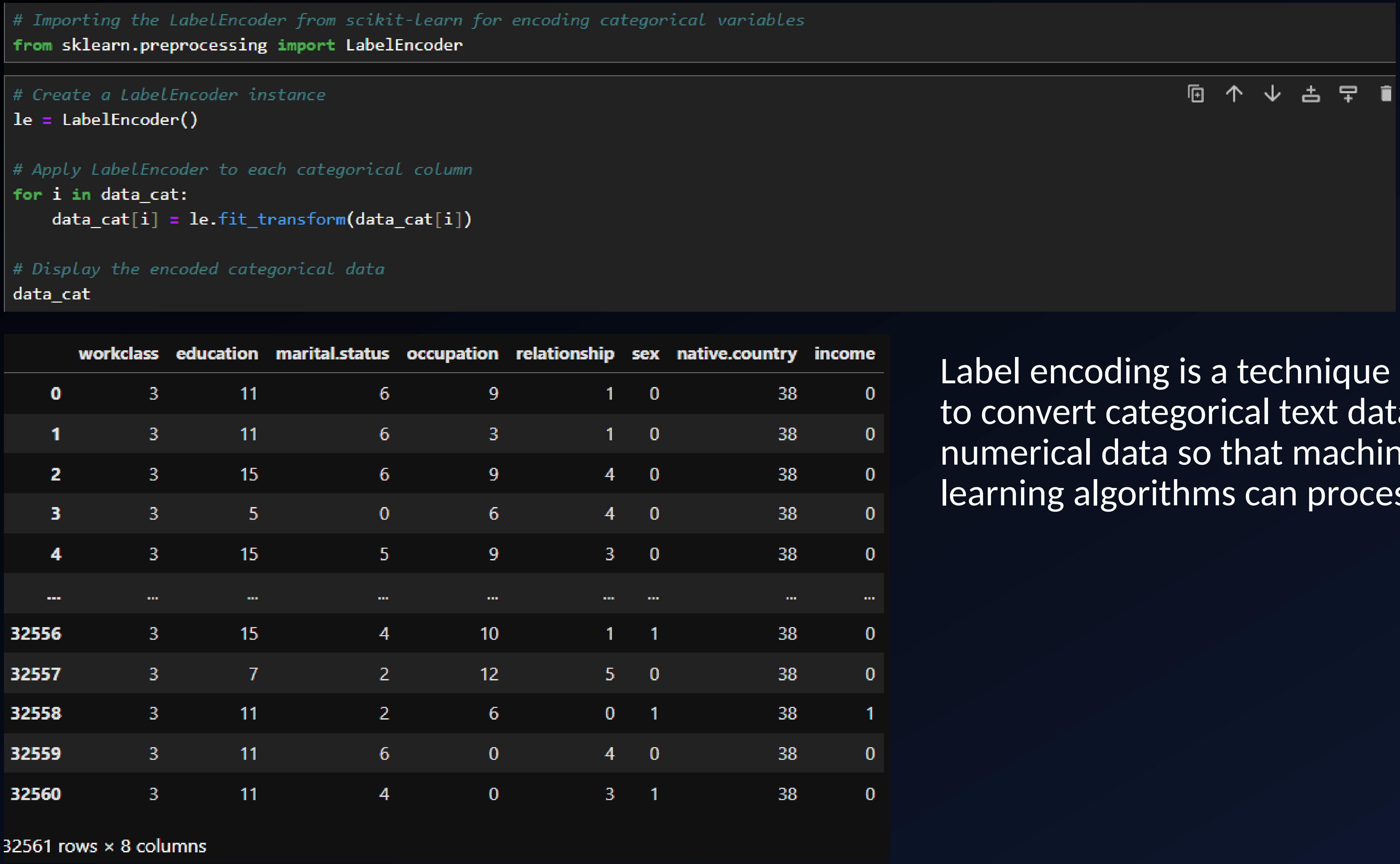
```python
# Select categorical columns from the DataFrame
data_cat = data.select_dtypes(["object"])

#display the numerical columns
data_cat
```

	workclass	education	marital.status	occupation	relationship	sex	native.country	income
0	Private	HS-grad	Widowed	Prof-specialty	Not-in-family	Female	United-States	<=50K
1	Private	HS-grad	Widowed	Exec-managerial	Not-in-family	Female	United-States	<=50K
2	Private	Some-college	Widowed	Prof-specialty	Unmarried	Female	United-States	<=50K
3	Private	7th-8th	Divorced	Machine-op-inspct	Unmarried	Female	United-States	<=50K
4	Private	Some-college	Separated	Prof-specialty	Own-child	Female	United-States	<=50K
...	...	...	...	...	...	...	...	...
32556	Private	Some-college	Never-married	Protective-serv	Not-in-family	Male	United-States	<=50K
32557	Private	Assoc-acdm	Married-civ-spouse	Tech-support	Wife	Female	United-States	<=50K
32558	Private	HS-grad	Married-civ-spouse	Machine-op-inspct	Husband	Male	United-States	>50K
32559	Private	HS-grad	Widowed	Adm-clerical	Unmarried	Female	United-States	<=50K
32560	Private	HS-grad	Never-married	Adm-clerical	Own-child	Male	United-States	<=50K

32561 rows × 8 columns

Here we have separated categorical column for label encoding.

LABEL ENCODING



```
# Importing the LabelEncoder from scikit-learn for encoding categorical variables
from sklearn.preprocessing import LabelEncoder
```

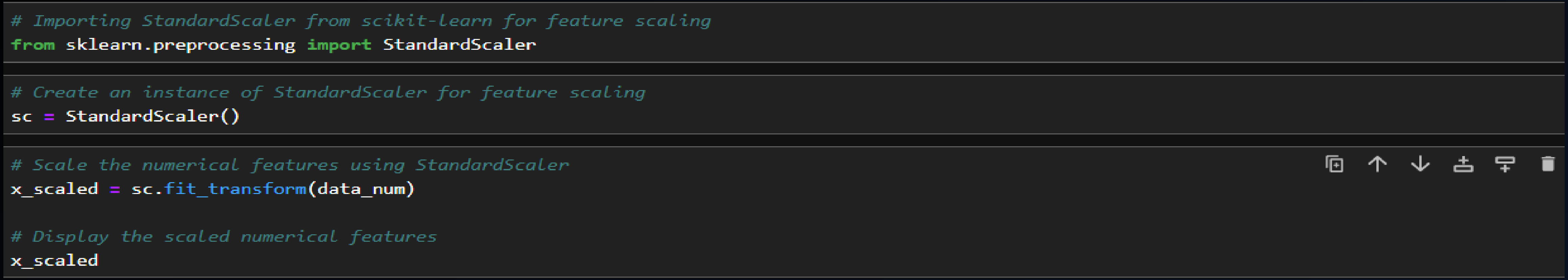
```
# Create a LabelEncoder instance
le = LabelEncoder()

# Apply LabelEncoder to each categorical column
for i in data_cat:
    data_cat[i] = le.fit_transform(data_cat[i])

# Display the encoded categorical data
data_cat
```

	workclass	education	marital.status	occupation	relationship	sex	native.country	income
0	3	11	6	9	1	0	38	0
1	3	11	6	3	1	0	38	0
2	3	15	6	9	4	0	38	0
3	3	5	0	6	4	0	38	0
4	3	15	5	9	3	0	38	0
...	...	...	...	...	...	...	...	...
32556	3	15	4	10	1	1	38	0
32557	3	7	2	12	5	0	38	0
32558	3	11	2	6	0	1	38	1
32559	3	11	6	0	4	0	38	0
32560	3	11	4	0	3	1	38	0

32561 rows × 8 columns

Label encoding is a technique used to convert categorical text data into numerical data so that machine learning algorithms can process it.

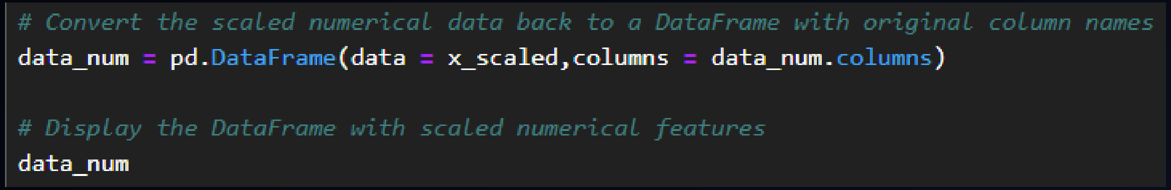
FEATURE SCALING



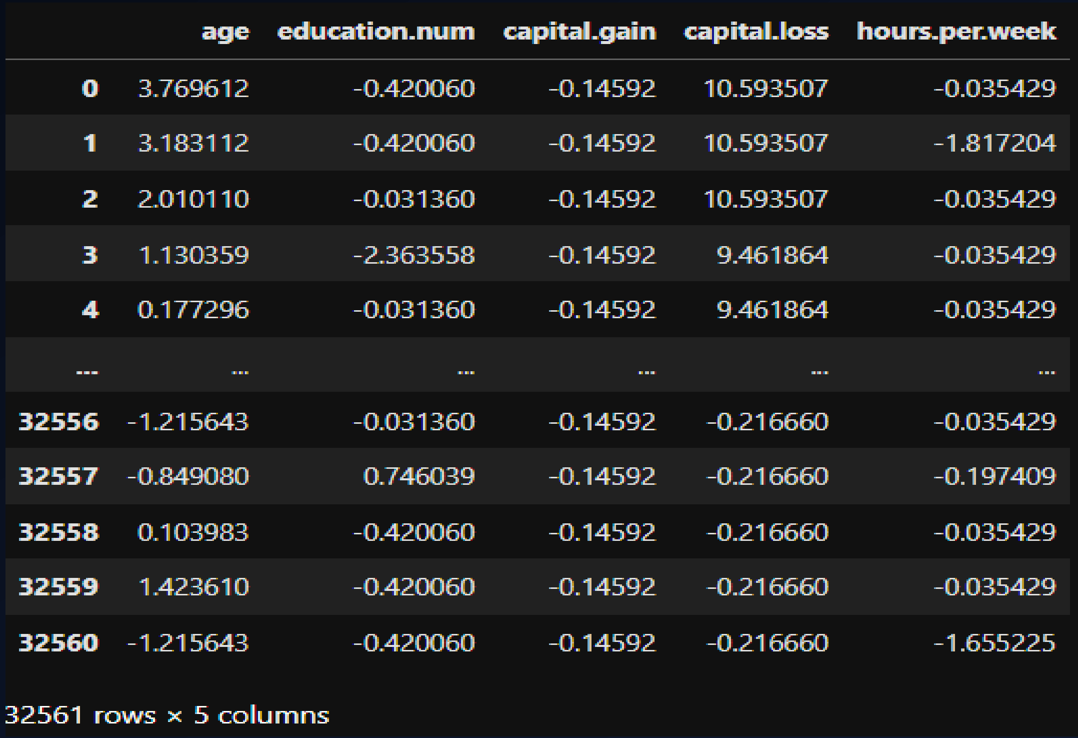
```
# Importing StandardScaler from scikit-learn for feature scaling
from sklearn.preprocessing import StandardScaler
```

```
# Create an instance of StandardScaler for feature scaling
sc = StandardScaler()
```

```
# Scale the numerical features using StandardScaler
x_scaled = sc.fit_transform(data_num)

# Display the scaled numerical features
x_scaled
```



```
# Convert the scaled numerical data back to a DataFrame with original column names
data_num = pd.DataFrame(data = x_scaled,columns = data_num.columns)

# Display the DataFrame with scaled numerical features
data_num
```

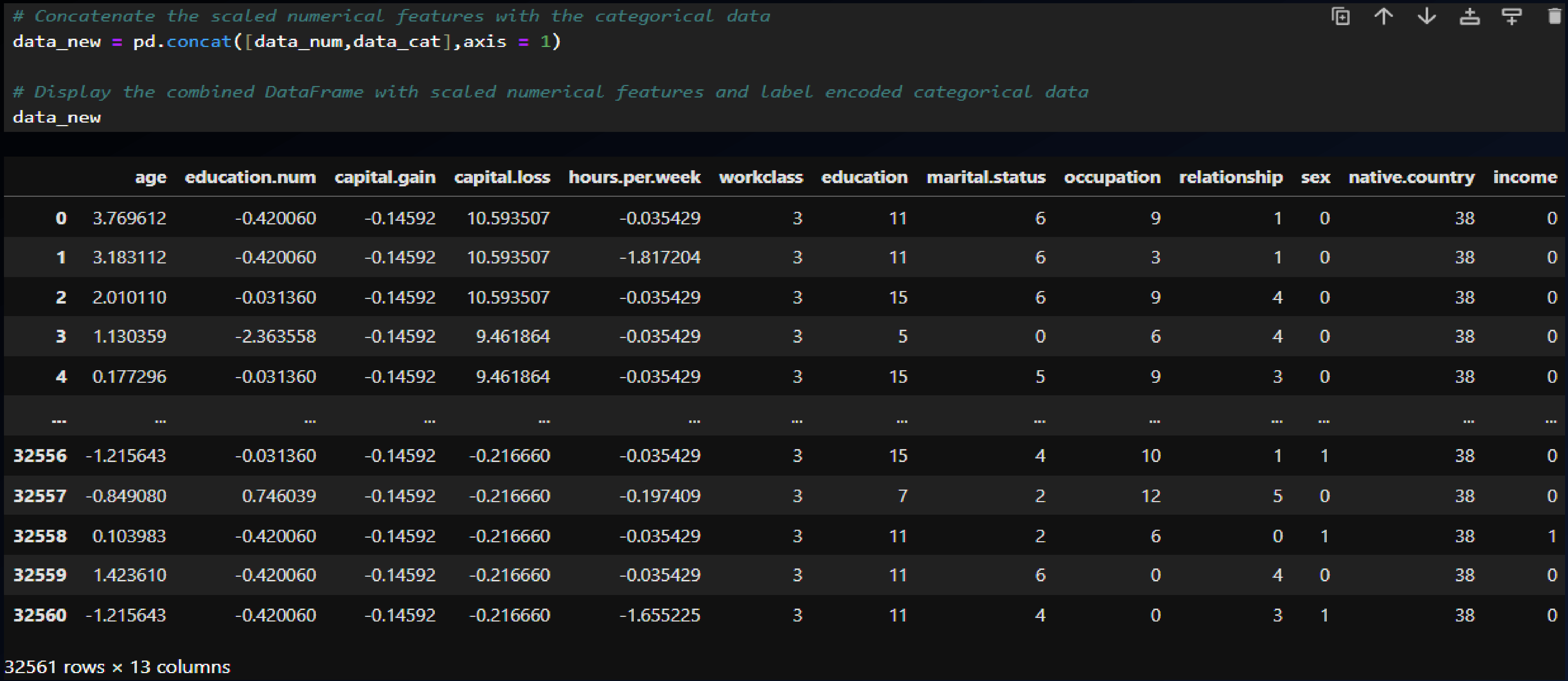


	age	education.num	capital.gain	capital.loss	hours.per.week
0	3.769612	-0.420060	-0.14592	10.593507	-0.035429
1	3.183112	-0.420060	-0.14592	10.593507	-1.817204
2	2.010110	-0.031360	-0.14592	10.593507	-0.035429
3	1.130359	-2.363558	-0.14592	9.461864	-0.035429
4	0.177296	-0.031360	-0.14592	9.461864	-0.035429
...	...	...	...	...	...
32556	-1.215643	-0.031360	-0.14592	-0.216660	-0.035429
32557	-0.849080	0.746039	-0.14592	-0.216660	-0.197409
32558	0.103983	-0.420060	-0.14592	-0.216660	-0.035429
32559	1.423610	-0.420060	-0.14592	-0.216660	-0.035429
32560	-1.215643	-0.420060	-0.14592	-0.216660	-1.655225

32561 rows × 5 columns

Feature scaling process ensures that the numerical data is standardized, which is a common preprocessing step in machine learning to improve model performance and convergence.

CONCATENATING NUMERICAL DATA AND CATEGORICAL DATA



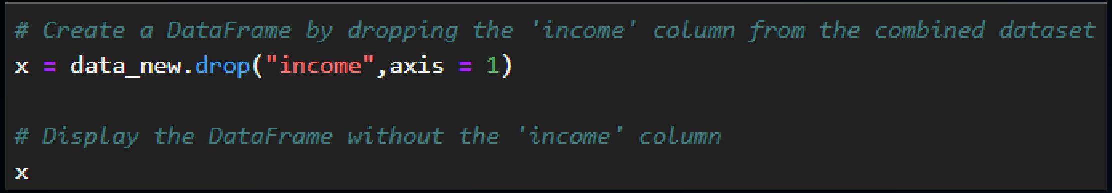
```python
# Concatenate the scaled numerical features with the categorical data
data_new = pd.concat([data_num,data_cat],axis = 1)

# Display the combined DataFrame with scaled numerical features and label encoded categorical data
data_new
```

	age	education.num	capital.gain	capital.loss	hours.per.week	workclass	education	marital.status	occupation	relationship	sex	native.country	income
0	3.769612	-0.420060	-0.14592	10.593507	-0.035429	3	11	6	9	1	0	38	0
1	3.183112	-0.420060	-0.14592	10.593507	-1.817204	3	11	6	3	1	0	38	0
2	2.010110	-0.031360	-0.14592	10.593507	-0.035429	3	15	6	9	4	0	38	0
3	1.130359	-2.363558	-0.14592	9.461864	-0.035429	3	5	0	6	4	0	38	0
4	0.177296	-0.031360	-0.14592	9.461864	-0.035429	3	15	5	9	3	0	38	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
32556	-1.215643	-0.031360	-0.14592	-0.216660	-0.035429	3	15	4	10	1	1	38	0
32557	-0.849080	0.746039	-0.14592	-0.216660	-0.197409	3	7	2	12	5	0	38	0
32558	0.103983	-0.420060	-0.14592	-0.216660	-0.035429	3	11	2	6	0	1	38	1
32559	1.423610	-0.420060	-0.14592	-0.216660	-0.035429	3	11	6	0	4	0	38	0
32560	-1.215643	-0.420060	-0.14592	-0.216660	-1.655225	3	11	4	0	3	1	38	0

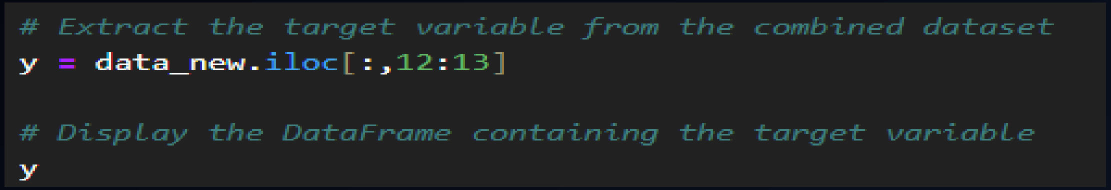
32561 rows × 13 columns

Here we have concatenated label encoded Categorical data and feature scaled numerical data. This process is useful when preparing data for further analysis or machine learning tasks, where both numerical and categorical features need to be included in the same

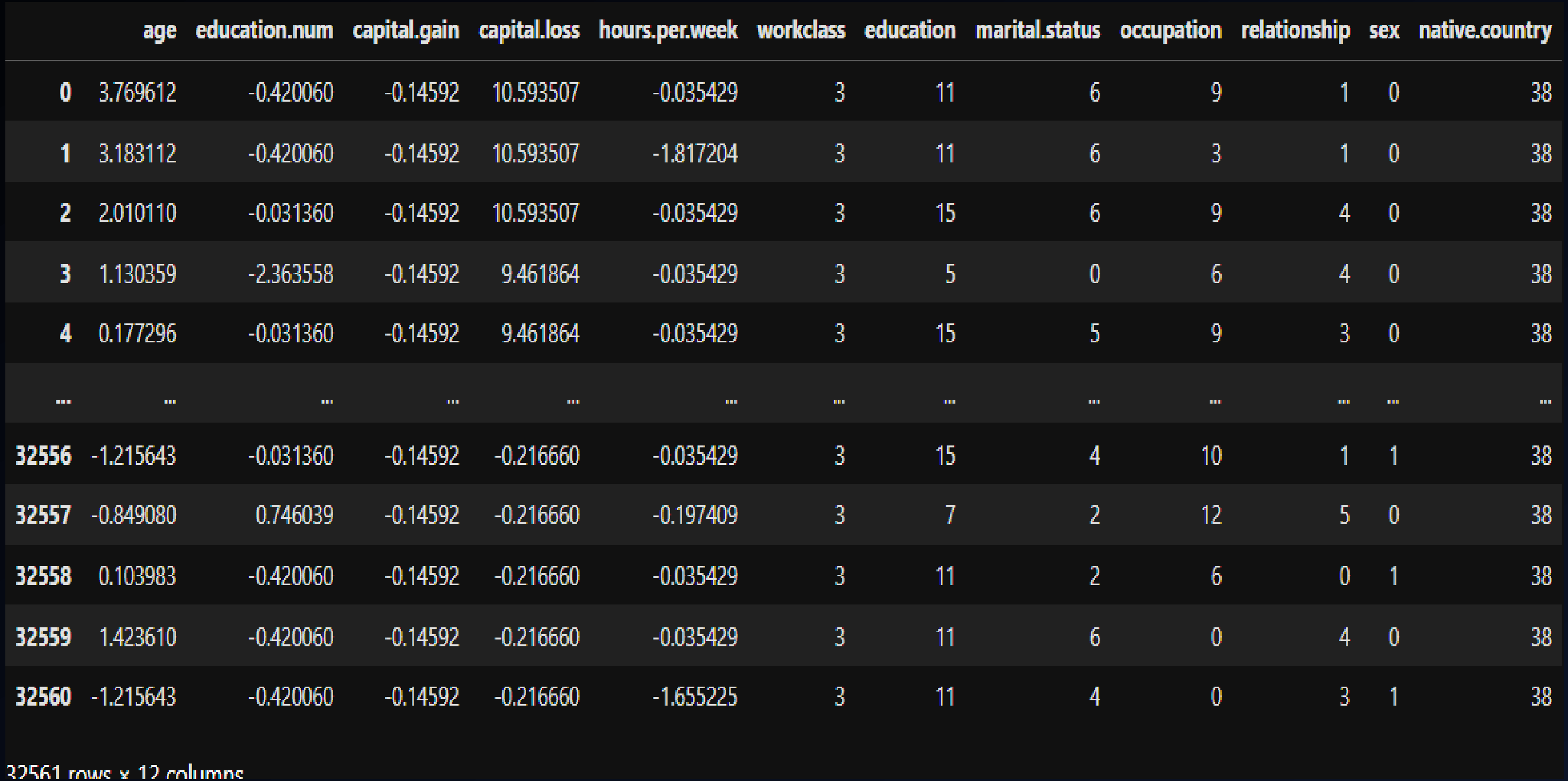
SPLITTING DATA INTO X (FEATURES) & Y (TAREGET



```python
# Create a DataFrame by dropping the 'income' column from the combined dataset
x = data_new.drop("income",axis = 1)

# Display the DataFrame without the 'income' column
x
```

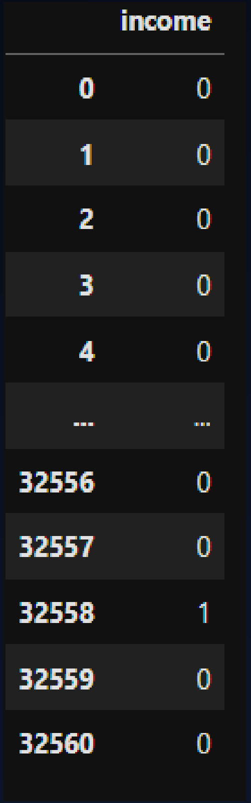


```python
# Extract the target variable from the combined dataset
y = data_new.iloc[:,12:13]

# Display the DataFrame containing the target variable
y
```



	age	education.num	capital.gain	capital.loss	hours.per.week	workclass	education	marital.status	occupation	relationship	sex	native.country
0	3.769612	-0.420060	-0.14592	10.593507	-0.035429	3	11	6	9	1	0	38
1	3.183112	-0.420060	-0.14592	10.593507	-1.817204	3	11	6	3	1	0	38
2	2.010110	-0.031360	-0.14592	10.593507	-0.035429	3	15	6	9	4	0	38
3	1.130359	-2.363558	-0.14592	9.461864	-0.035429	3	5	0	6	4	0	38
4	0.177296	-0.031360	-0.14592	9.461864	-0.035429	3	15	5	9	3	0	38
...	...	...	...	...	...	...	...	...	...	...	...	...
32556	-1.215643	-0.031360	-0.14592	-0.216660	-0.035429	3	15	4	10	1	1	38
32557	-0.849080	0.746039	-0.14592	-0.216660	-0.197409	3	7	2	12	5	0	38
32558	0.103983	-0.420060	-0.14592	-0.216660	-0.035429	3	11	2	6	0	1	38
32559	1.423610	-0.420060	-0.14592	-0.216660	-0.035429	3	11	6	0	4	0	38
32560	-1.215643	-0.420060	-0.14592	-0.216660	-1.655225	3	11	4	0	3	1	38

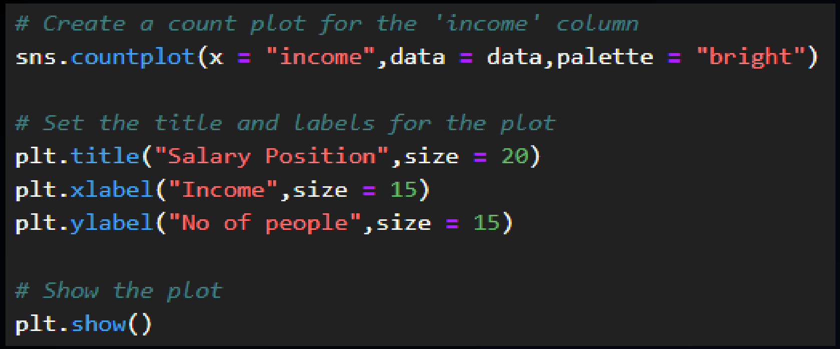
32561 rows x 12 columns



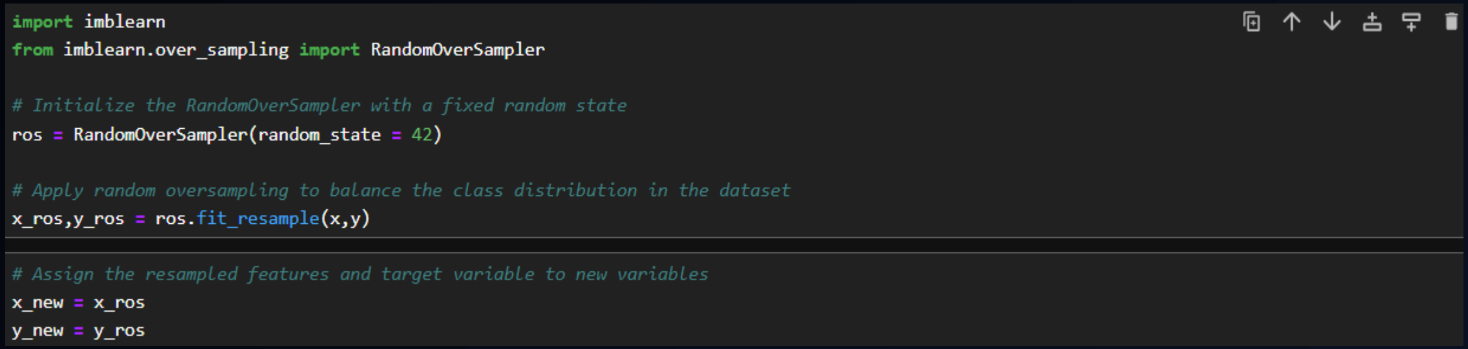
	income
0	0
1	0
2	0
3	0
4	0
...	...
32556	0
32557	0
32558	1
32559	0
32560	0

Here we separated the data into features(x) and target(y) variable

SAMPLING

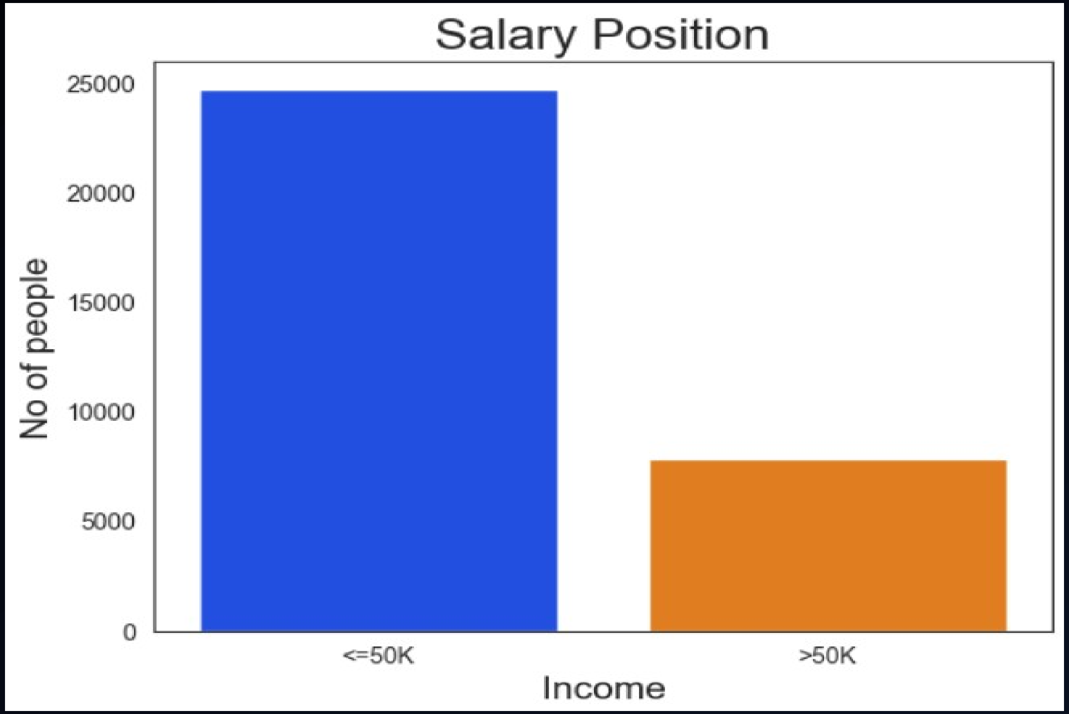


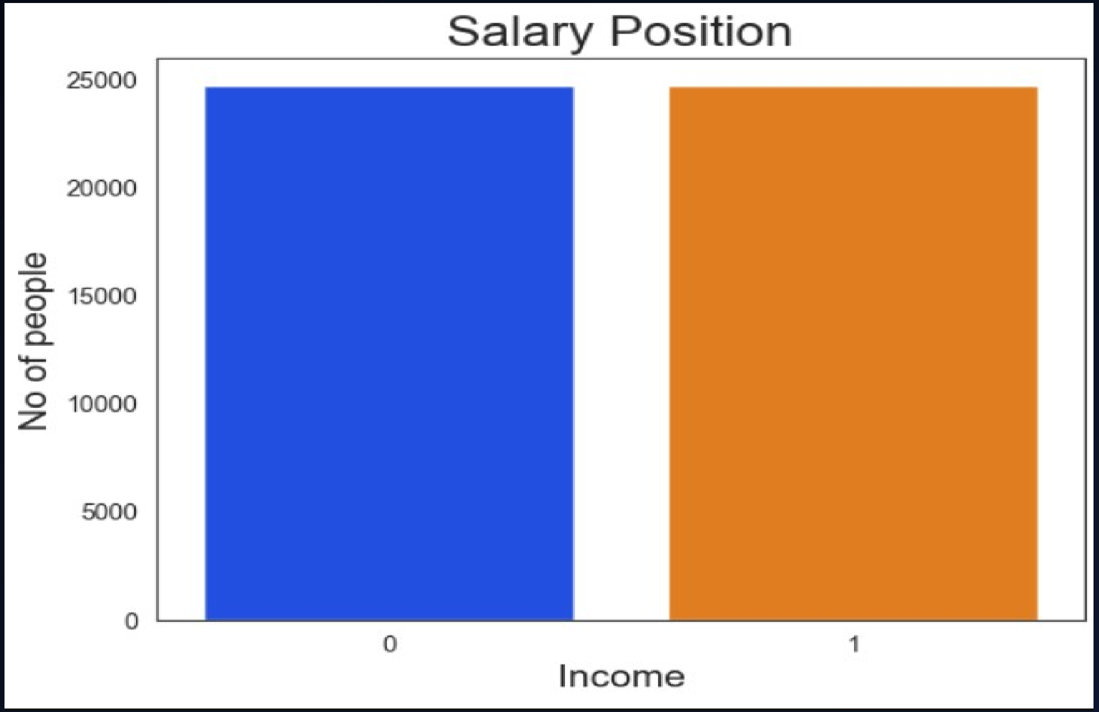
```python
# Create a count plot for the 'income' column
sns.countplot(x = "income",data = data,palette = "bright")

# Set the title and labels for the plot
plt.title("Salary Position",size = 20)
plt.xlabel("Income",size = 15)
plt.ylabel("No of people",size = 15)

# Show the plot
plt.show()
```



```python
import imblearn
from imblearn.over_sampling import RandomOverSampler

# Initialize the RandomOverSampler with a fixed random state
ros = RandomOverSampler(random_state = 42)

# Apply random oversampling to balance the class distribution in the dataset
x_ros,y_ros = ros.fit_resample(x,y)
```

```python
# Assign the resampled features and target variable to new variables
x_new = x_ros
y_new = y_ros
```

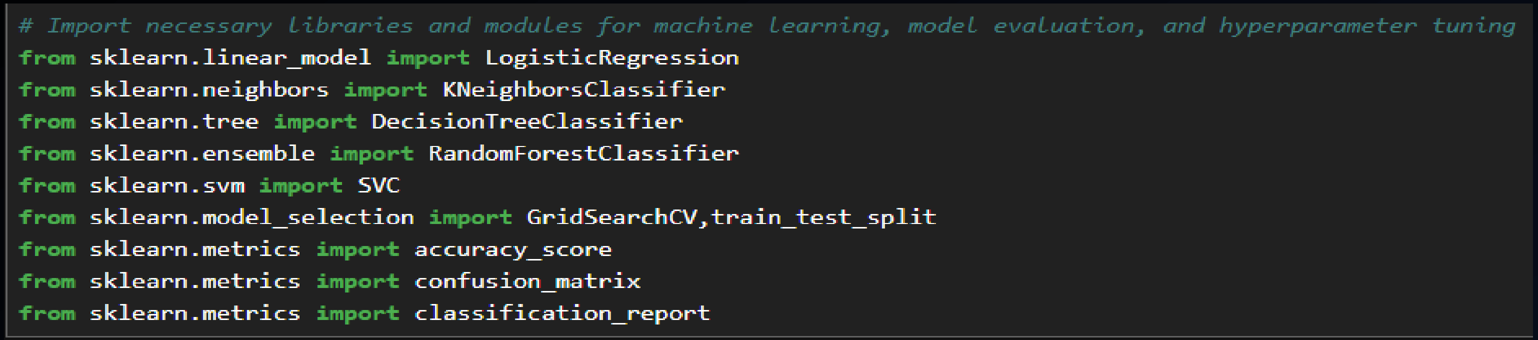




75.92% of the employees earn 50K USD or less while the remaining 24.08% earn above 50k USD.. Here we need to do up sampling to remove bias in training model as the data is imbalanced

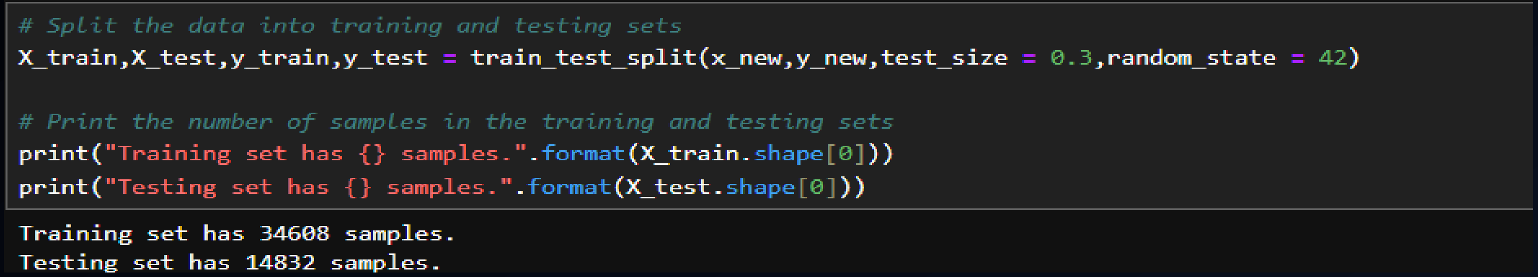
Above visualization helps to confirm that the oversampling process has balanced the distribution of income categories.
Here we have preferred Upsampling over downsampling because it helps to preserve all original data and improves the representation of the minority class without losing valuable information. However, the choice between upsampling and downsampling should be made based

MODEL SELECTION



```python
# Import necessary libraries and modules for machine learning, model evaluation, and hyperparameter tuning
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV,train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

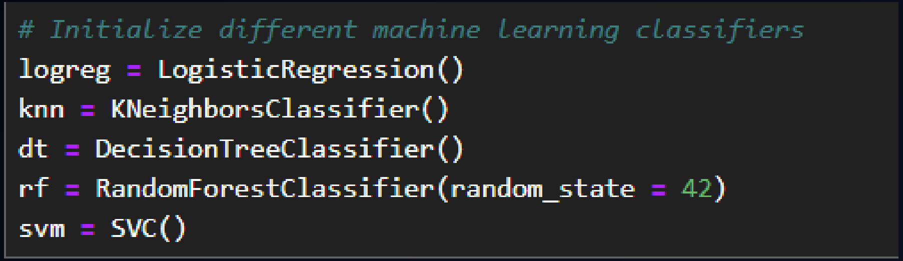
Here we have Imported necessary libraries and modules for machine learning, model evaluation, and hyperparameter tuning & Split the data into training and testing sets



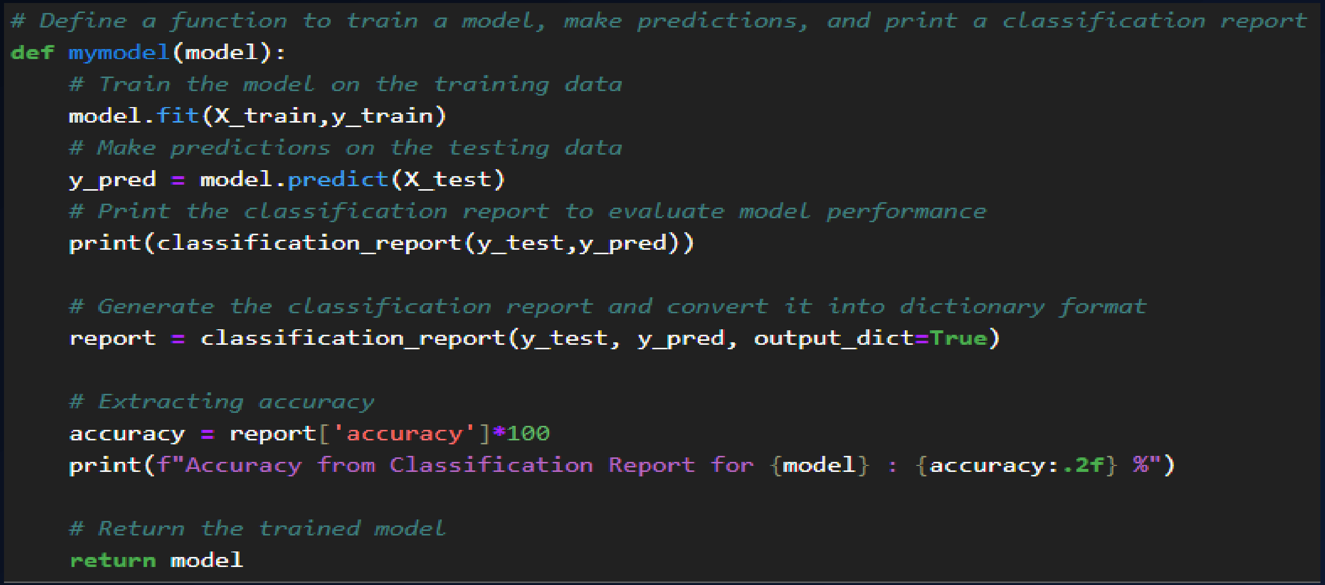
```python
# Split the data into training and testing sets
X_train,X_test,y_train,y_test = train_test_split(x_new,y_new,test_size = 0.3,random_state = 42)

# Print the number of samples in the training and testing sets
print("Training set has {} samples.".format(X_train.shape[0]))
print("Testing set has {} samples.".format(X_test.shape[0]))
```

```
Training set has 34608 samples.
Testing set has 14832 samples.
```

Further we have initialize different machine learning classifiers.



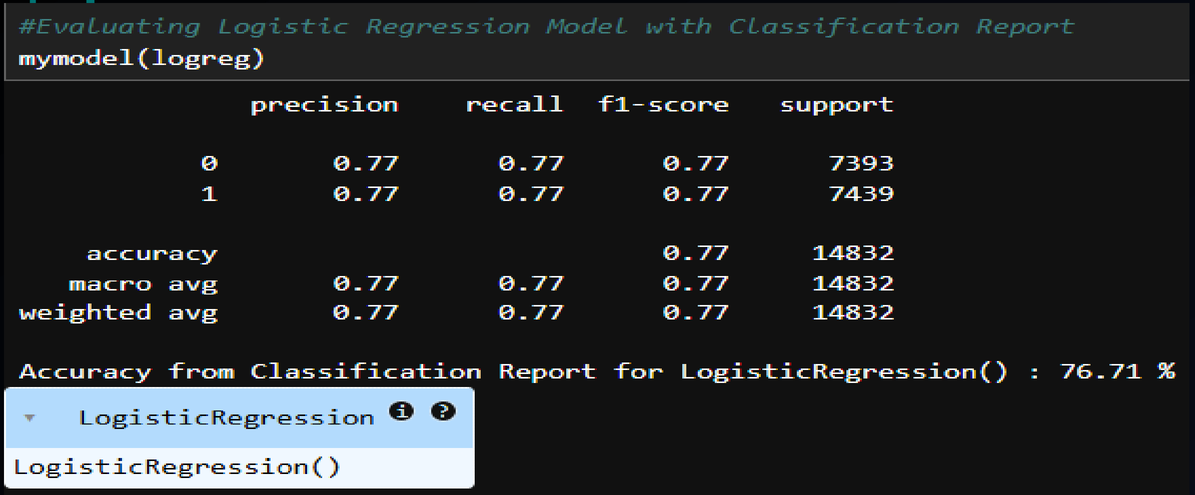
```python
# Initialize different machine learning classifiers
logreg = LogisticRegression()
knn = KNeighborsClassifier()
dt = DecisionTreeClassifier()
rf = RandomForestClassifier(random_state = 42)
svm = SVC()
```



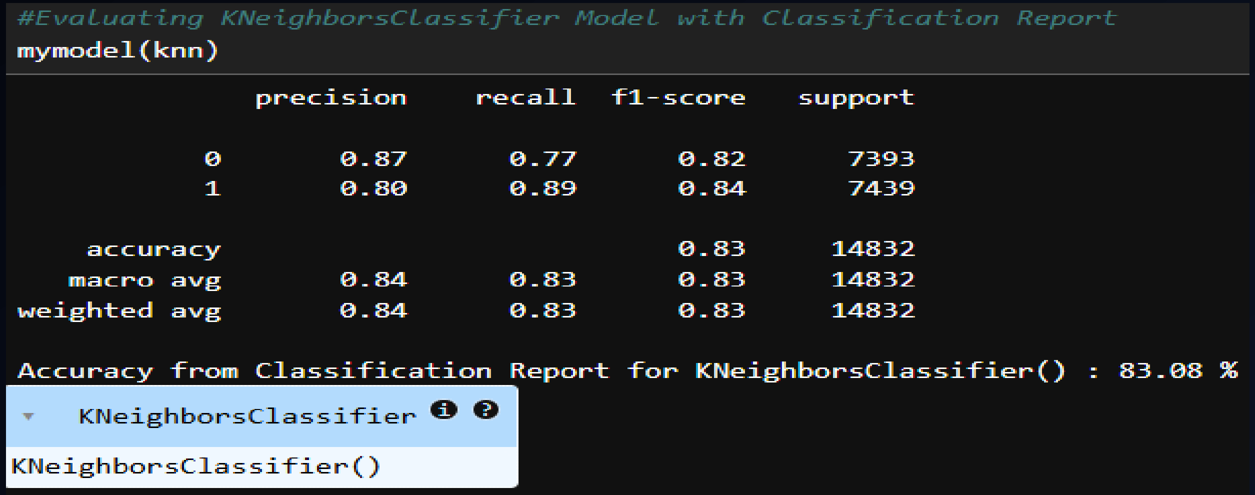
```python
# Define a function to train a model, make predictions, and print a classification report
def mymodel(model):
    # Train the model on the training data
    model.fit(X_train,y_train)
    # Make predictions on the testing data
    y_pred = model.predict(X_test)
    # Print the classification report to evaluate model performance
    print(classification_report(y_test,y_pred))

    # Generate the classification report and convert it into dictionary format
    report = classification_report(y_test, y_pred, output_dict=True)

    # Extracting accuracy
    accuracy = report['accuracy']*100
    print(f"Accuracy from Classification Report for {model} : {accuracy:.2f} %")

    # Return the trained model
    return model
```

Here we have define a function to train a model, make predictions, and print a classification report

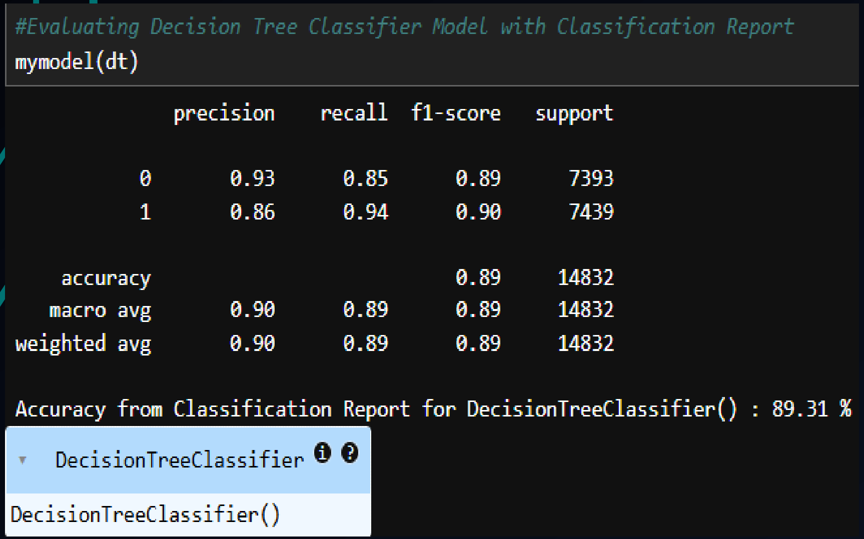
MODEL EVALUATION



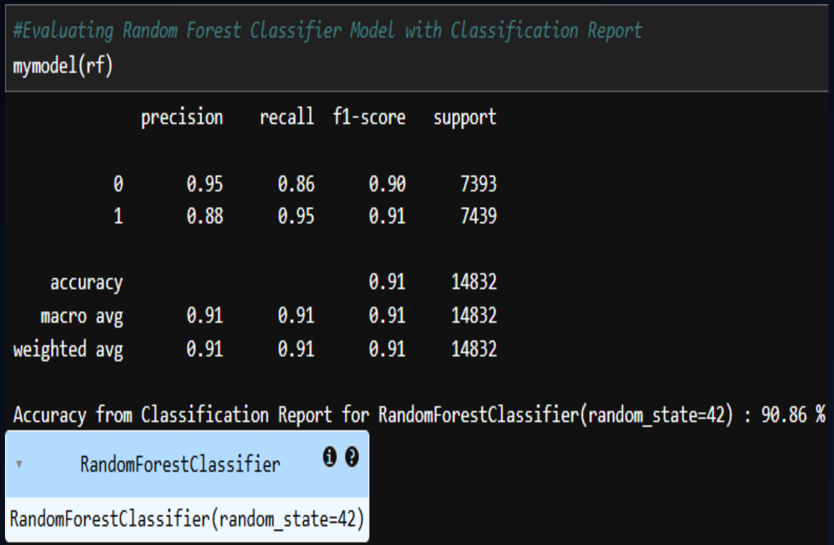
```
#Evaluating Logistic Regression Model with Classification Report
mymodel(logreg)

              precision    recall  f1-score   support

           0       0.77      0.77      0.77      7393
           1       0.77      0.77      0.77      7439

    accuracy                           0.77     14832
   macro avg       0.77      0.77      0.77     14832
weighted avg       0.77      0.77      0.77     14832

Accuracy from Classification Report for LogisticRegression() : 76.71 %

LogisticRegression
LogisticRegression()
```



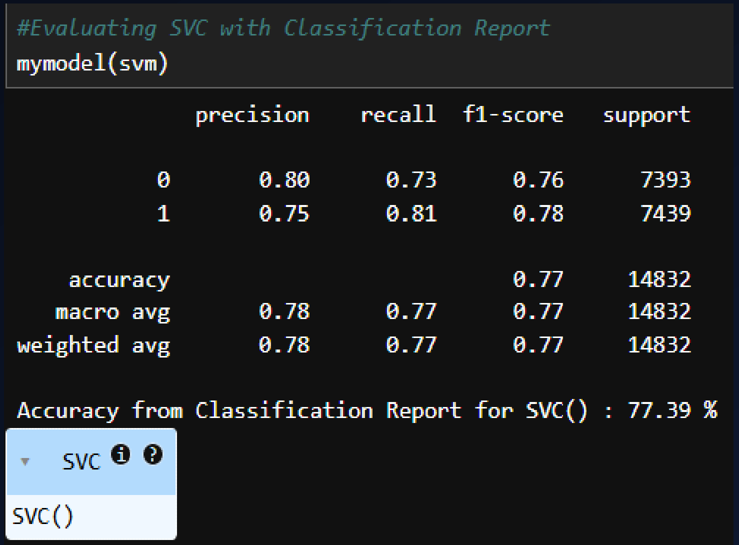
```
#Evaluating KNeighborsClassifier Model with Classification Report
mymodel(knn)

              precision    recall  f1-score   support

           0       0.87      0.77      0.82      7393
           1       0.80      0.89      0.84      7439

    accuracy                           0.83     14832
   macro avg       0.84      0.83      0.83     14832
weighted avg       0.84      0.83      0.83     14832

Accuracy from Classification Report for KNeighborsClassifier() : 83.08 %

KNeighborsClassifier
KNeighborsClassifier()
```



```
#Evaluating Decision Tree Classifier Model with Classification Report
mymodel(dt)

              precision    recall  f1-score   support

           0       0.93      0.85      0.89      7393
           1       0.86      0.94      0.90      7439

    accuracy                           0.89     14832
   macro avg       0.90      0.89      0.89     14832
weighted avg       0.90      0.89      0.89     14832

Accuracy from Classification Report for DecisionTreeClassifier() : 89.31 %

DecisionTreeClassifier
DecisionTreeClassifier()
```

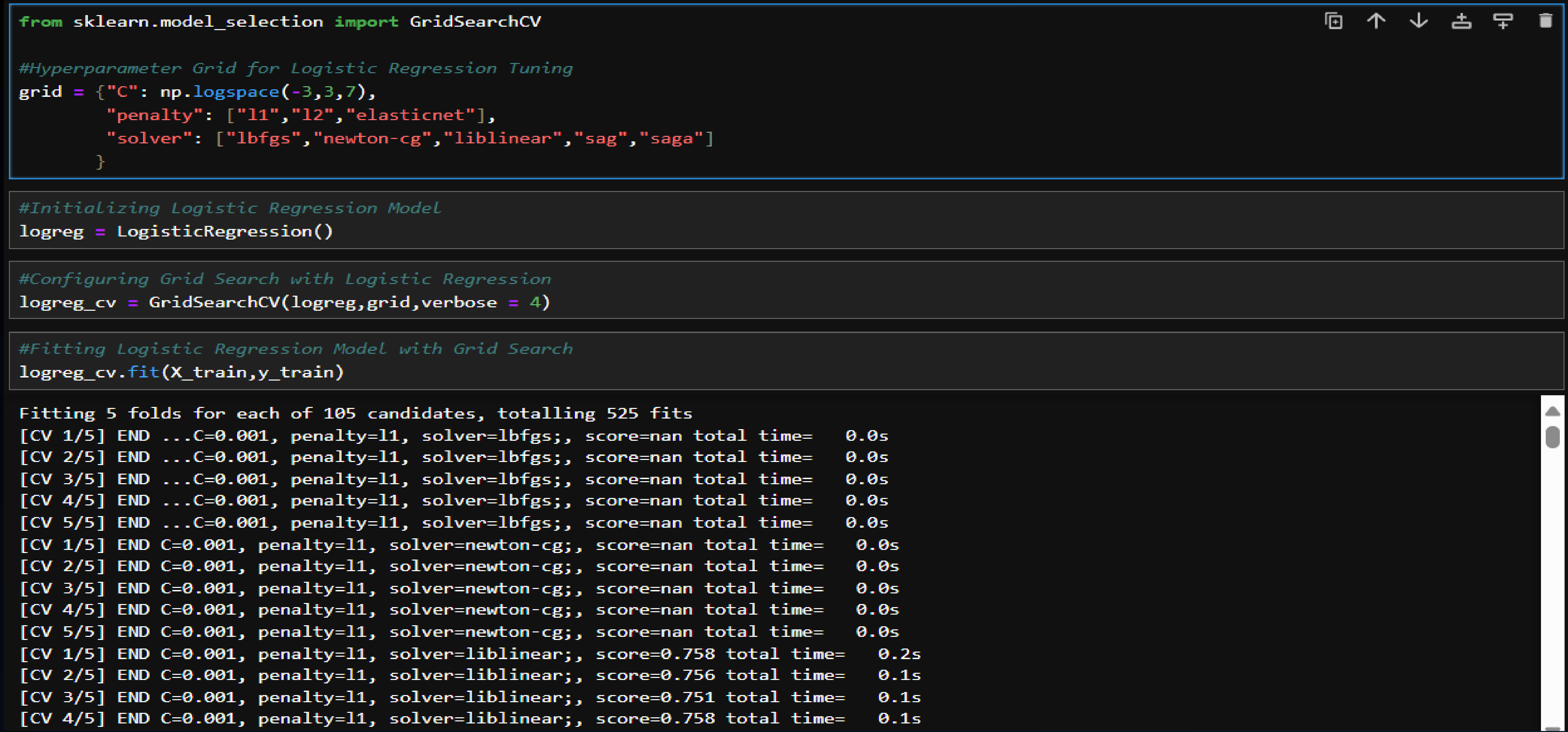


```
#Evaluating Random Forest Classifier Model with Classification Report
mymodel(rf)

              precision    recall  f1-score   support

           0       0.95      0.86      0.90      7393
           1       0.88      0.95      0.91      7439

    accuracy                           0.91     14832
   macro avg       0.91      0.91      0.91     14832
weighted avg       0.91      0.91      0.91     14832

Accuracy from Classification Report for RandomForestClassifier(random_state=42) : 90.86 %

RandomForestClassifier
RandomForestClassifier(random_state=42)
```



```
#Evaluating SVC with Classification Report
mymodel(svm)

              precision    recall  f1-score   support

           0       0.80      0.73      0.76      7393
           1       0.75      0.81      0.78      7439

    accuracy                           0.77     14832
   macro avg       0.78      0.77      0.77     14832
weighted avg       0.78      0.77      0.77     14832

Accuracy from Classification Report for SVC() : 77.39 %

SVC
SVC()
```

LOGISTIC REGRESSION ALGORITHM USING GRID SEARCH CV



```
from sklearn.model_selection import GridSearchCV

#Hyperparameter Grid for Logistic Regression Tuning
grid = {"C": np.logspace(-3,3,7),
        "penalty": ["l1","l2","elasticnet"],
        "solver": ["lbfgs","newton-cg","liblinear","sag","saga"]
       }
```

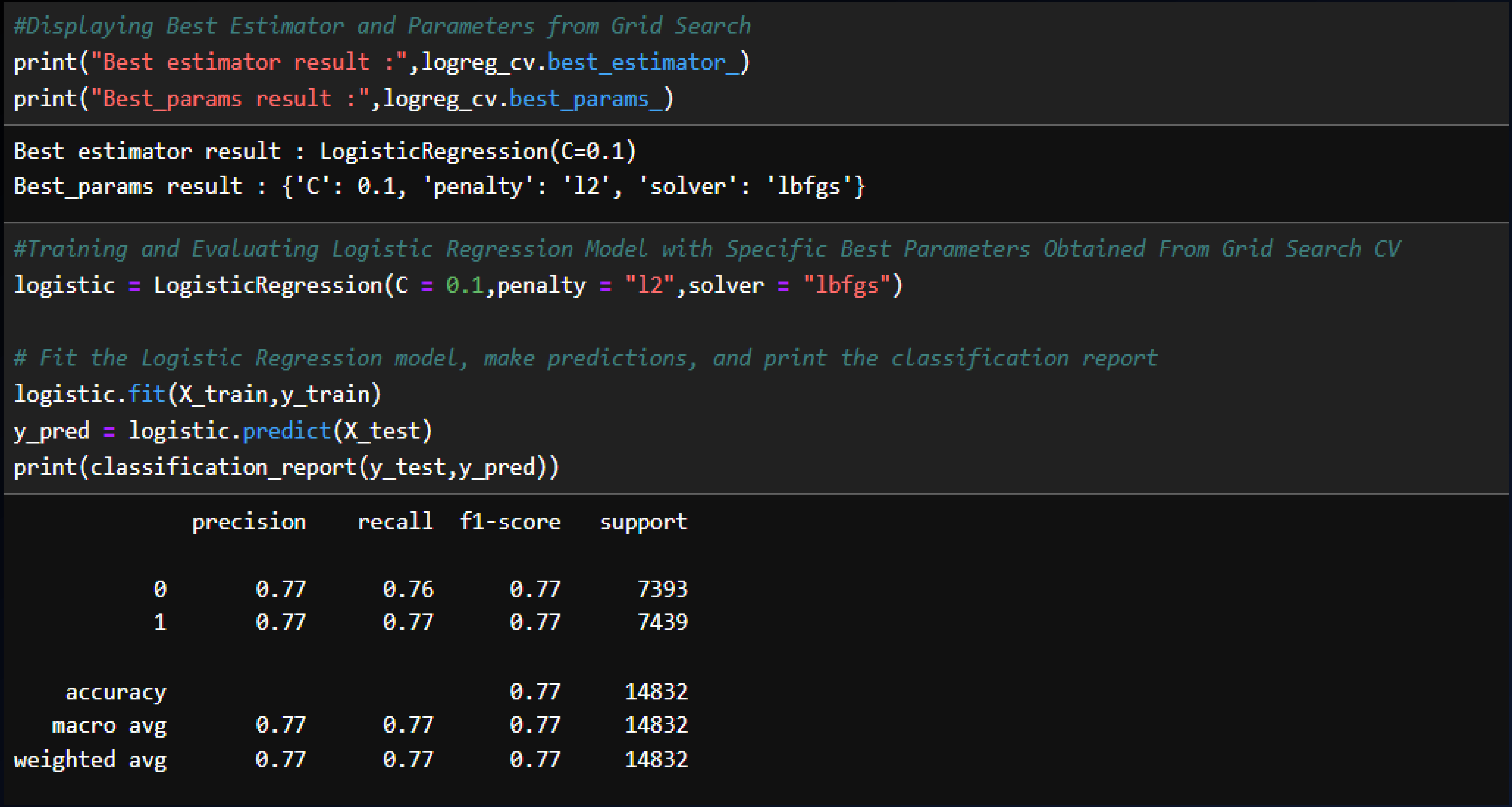
```
#Initializing Logistic Regression Model
logreg = LogisticRegression()
```

```
#Configuring Grid Search with Logistic Regression
logreg_cv = GridSearchCV(logreg,grid,verbose = 4)
```

```
#Fitting Logistic Regression Model with Grid Search
logreg_cv.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 105 candidates, totalling 525 fits
[CV 1/5] END ...C=0.001, penalty=l1, solver=lbfgs;, score=nan total time=   0.0s
[CV 2/5] END ...C=0.001, penalty=l1, solver=lbfgs;, score=nan total time=   0.0s
[CV 3/5] END ...C=0.001, penalty=l1, solver=lbfgs;, score=nan total time=   0.0s
[CV 4/5] END ...C=0.001, penalty=l1, solver=lbfgs;, score=nan total time=   0.0s
[CV 5/5] END ...C=0.001, penalty=l1, solver=lbfgs;, score=nan total time=   0.0s
[CV 1/5] END C=0.001, penalty=l1, solver=newton-cg;, score=nan total time=   0.0s
[CV 2/5] END C=0.001, penalty=l1, solver=newton-cg;, score=nan total time=   0.0s
[CV 3/5] END C=0.001, penalty=l1, solver=newton-cg;, score=nan total time=   0.0s
[CV 4/5] END C=0.001, penalty=l1, solver=newton-cg;, score=nan total time=   0.0s
[CV 5/5] END C=0.001, penalty=l1, solver=newton-cg;, score=nan total time=   0.0s
[CV 1/5] END C=0.001, penalty=l1, solver=liblinear;, score=0.758 total time=   0.2s
[CV 2/5] END C=0.001, penalty=l1, solver=liblinear;, score=0.756 total time=   0.1s
[CV 3/5] END C=0.001, penalty=l1, solver=liblinear;, score=0.751 total time=   0.1s
[CV 4/5] END C=0.001, penalty=l1, solver=liblinear;, score=0.758 total time=   0.1s
```

Grid Search Cross-Validation (Grid Search CV) is a technique used in machine learning to find the best hyperparameters for a given model by exhaustively searching through a specified subset of the hyperparameter space.



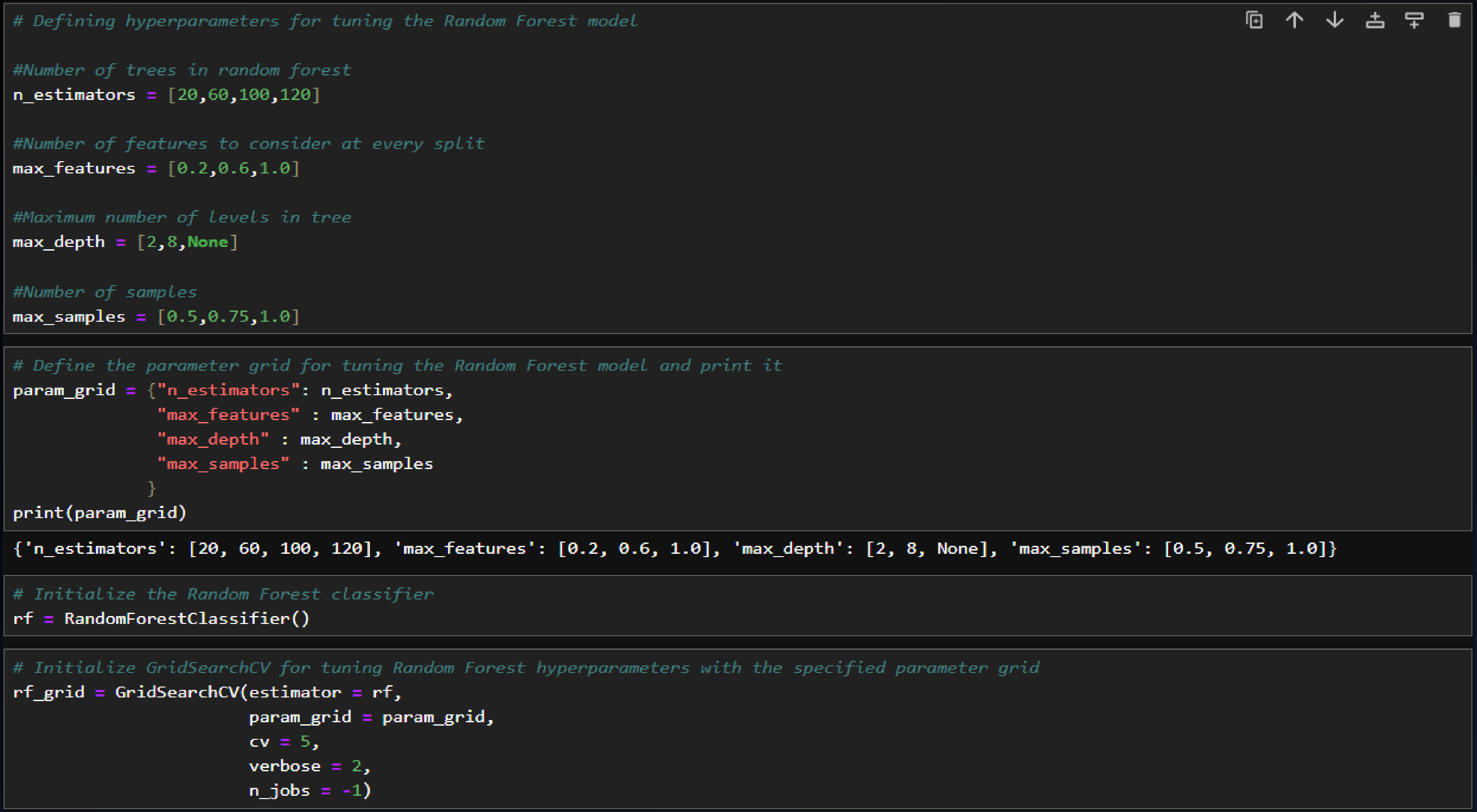
```
#Displaying Best Estimator and Parameters from Grid Search
print("Best estimator result :",logreg_cv.best_estimator_)
print("Best_params result :",logreg_cv.best_params_)
```

```
Best estimator result : LogisticRegression(C=0.1)
Best_params result : {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
```

```
#Training and Evaluating Logistic Regression Model with Specific Best Parameters Obtained From Grid Search CV
logistic = LogisticRegression(C = 0.1,penalty = "l2",solver = "lbfgs")

# Fit the Logistic Regression model, make predictions, and print the classification report
logistic.fit(X_train,y_train)
y_pred = logistic.predict(X_test)
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.77      0.76      0.77      7393
           1       0.77      0.77      0.77      7439

    accuracy                           0.77     14832
   macro avg       0.77      0.77      0.77     14832
weighted avg       0.77      0.77      0.77     14832
```

Random Forest Algorithm using Grid Search CV



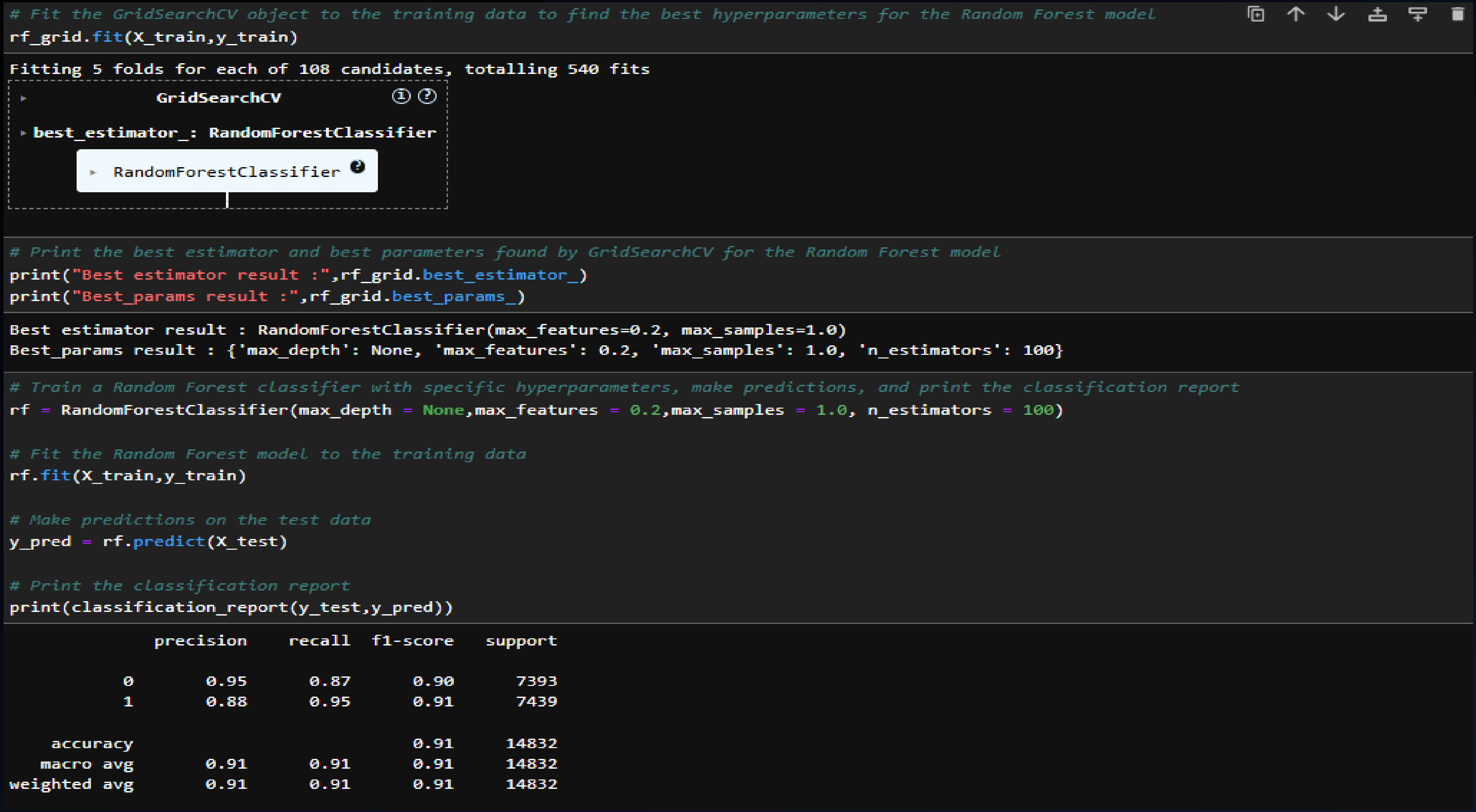
```
# Defining hyperparameters for tuning the Random Forest model

#Number of trees in random forest
n_estimators = [20,60,100,120]

#Number of features to consider at every split
max_features = [0.2,0.6,1.0]

#Maximum number of levels in tree
max_depth = [2,8,None]

#Number of samples
max_samples = [0.5,0.75,1.0]
```

```
# Define the parameter grid for tuning the Random Forest model and print it
param_grid = {"n_estimators": n_estimators,
              "max_features" : max_features,
              "max_depth" : max_depth,
              "max_samples" : max_samples
             }
print(param_grid)
```

```
{'n_estimators': [20, 60, 100, 120], 'max_features': [0.2, 0.6, 1.0], 'max_depth': [2, 8, None], 'max_samples': [0.5, 0.75, 1.0]}
```

```
# Initialize the Random Forest classifier
rf = RandomForestClassifier()
```

```
# Initialize GridSearchCV for tuning Random Forest hyperparameters with the specified parameter grid
rf_grid = GridSearchCV(estimator = rf,
                       param_grid = param_grid,
                       cv = 5,
                       verbose = 2,
                       n_jobs = -1)
```



```
# Fit the GridSearchCV object to the training data to find the best hyperparameters for the Random Forest model
rf_grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
```

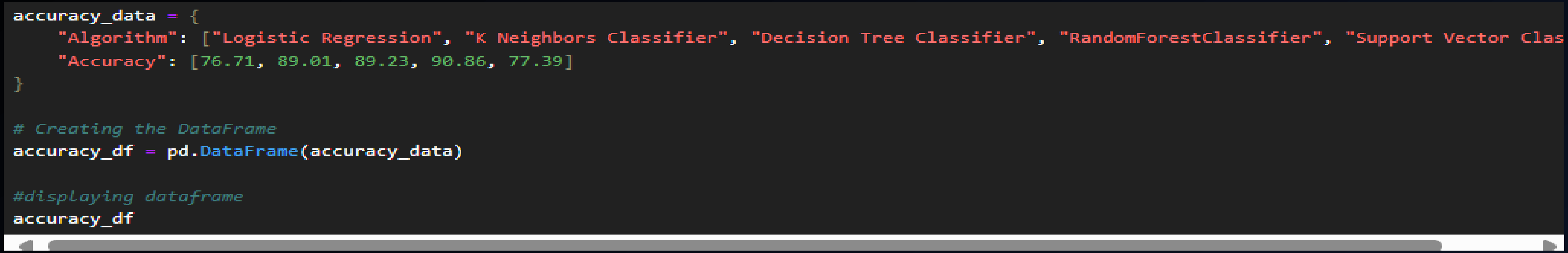
```
# Print the best estimator and best parameters found by GridSearchCV for the Random Forest model
print("Best estimator result :",rf_grid.best_estimator_)
print("Best_params result :",rf_grid.best_params_)
```

```
Best estimator result : RandomForestClassifier(max_features=0.2, max_samples=1.0)
Best_params result : {'max_depth': None, 'max_features': 0.2, 'max_samples': 1.0, 'n_estimators': 100}
```

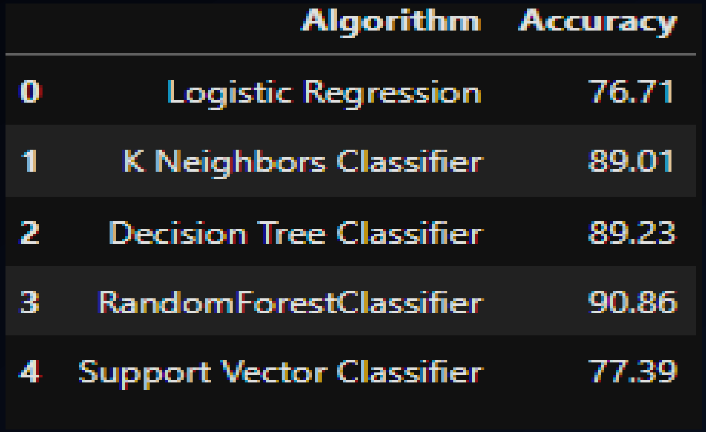
```
# Train a Random Forest classifier with specific hyperparameters, make predictions, and print the classification report
rf = RandomForestClassifier(max_depth = None,max_features = 0.2,max_samples = 1.0, n_estimators = 100)

# Fit the Random Forest model to the training data
rf.fit(X_train,y_train)

# Make predictions on the test data
y_pred = rf.predict(X_test)

# Print the classification report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.87      0.90      7393
           1       0.88      0.95      0.91      7439

    accuracy                           0.91     14832
   macro avg       0.91      0.91      0.91     14832
weighted avg       0.91      0.91      0.91     14832
```

Results



```
accuracy_data = {
    "Algorithm": ["Logistic Regression", "K Neighbors Classifier", "Decision Tree Classifier", "RandomForestClassifier", "Support Vector Clas
    "Accuracy": [76.71, 89.01, 89.23, 90.86, 77.39]
}

# Creating the DataFrame
accuracy_df = pd.DataFrame(accuracy_data)

#displaying dataframe
accuracy_df
```



	Algorithm	Accuracy
0	Logistic Regression	76.71
1	K Neighbors Classifier	89.01
2	Decision Tree Classifier	89.23
3	RandomForestClassifier	90.86
4	Support Vector Classifier	77.39

- From here, we can conclude that Random Forest Classifier has the highest accuracy with a score of 90.86%.
- This indicates that the Random Forest algorithm performs the best among the ones listed for the given dataset.
- Logistic Regression has the lowest accuracy with a score of 76.71%. This suggests that it is the least effective model in this comparison for this dataset.

CONCLUSION

- **We can conclude that Random Forest Classifier has the highest accuracy with a score of 90.86%. This indicates that the Random Forest algorithm performs the best among the ones listed for the given dataset.**
- **The Random Forest Classifier is the most accurate model in this comparison, achieving the highest accuracy, while Logistic Regression has the lowest accuracy. The other classifiers fall in between, with the K Neighbors and Decision Tree models showing strong performance but not quite matching the Random Forest's accuracy**

THANK YOU