Course: Artificial Intelligence and Data Science

Module Leader: Mr. Sahan Priyanayana

CM2604 Machine Learning

Assignment Type:

Individual

Student Name: W.A.C.I.V Bandara

IIT ID-20222453

RGU ID-2311689

Table of content

Corpus preparation

The UCL repository is a website which contain huge selection of datasets for machine learning tasks. Adult data set was selected form UCL repository and this dataset contain 48842 data records and 15 attributes and indicate that given person Census Income is more than 50K or not. Data cleaning was one of the pre-processing steps that was carried out. A few of the steps included removing outliers, rejecting duplicate values, and looking for and removing null values.

Random Forest and Naïve Bayes are the algorithms that used for classify weather the salary exceeds 50K or not. After cleaning and preprocessing phase 70% of data is used for training while remaining amount used for testing purposes.

Preprocessing techniques

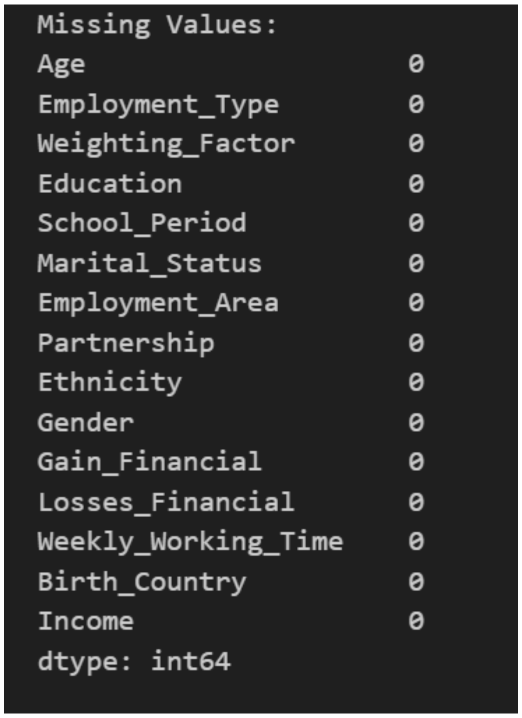
The preprocessing stage is crucial to the refinement of unprocessed data for machine learning models. To guarantee data quality and compatibility, they include activities like data cleansing, scaling, and encoding. By simplifying and emphasizing pertinent data, feature selection and dimensionality reduction expedite the training of models. Text preprocessing transforms textual data into numerical representations in order to make it ready for analysis. Class distribution biases are addressed via methods for handling unbalanced data, and efficient model evaluation is facilitated by data partitioning. These methods work together to provide the foundation for data preprocessing, which makes it possible to create reliable and accurate machine learning models.

Data Cleaning visualization and preprocessing

The process of finding and fixing mistakes, inconsistencies, and inaccuracies in a dataset such that it is reliable and of high quality for analysis or modelling is known as data cleaning. This includes dealing with duplicate or missing values, fixing mistakes in data entry, eliminating outliers, and standardizing formats, among other things. Through the resolution of these problems, data cleaning increases the dataset's integrity, lowers the possibility of inaccurate or biased outcomes, and boosts the efficiency of ensuing data analysis or machine learning operations.

Checking missing values in the dataset

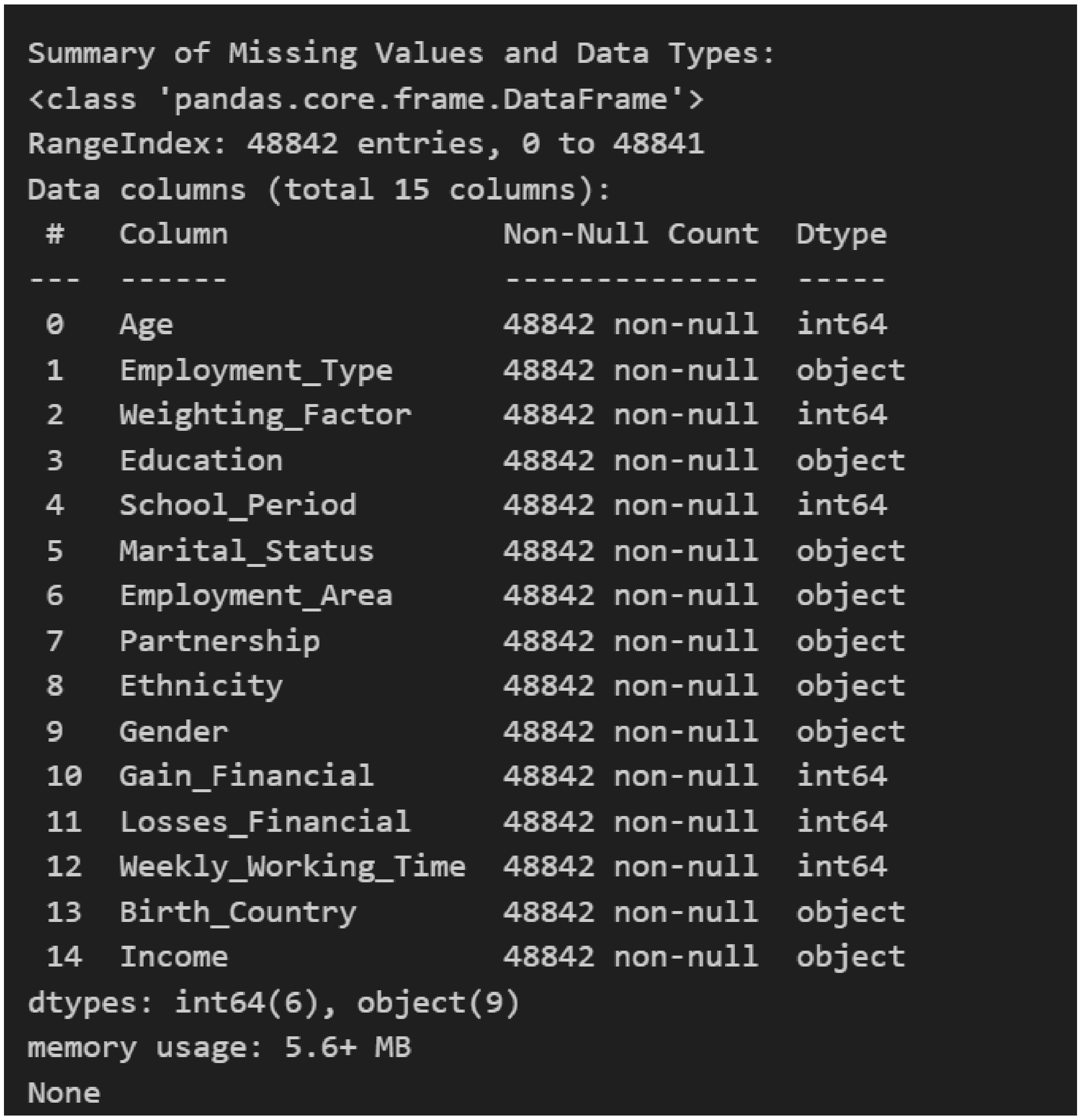
```
missing_values = df1.isnull()
# taking the count
missing_counts = missing_values.sum()
print("Missing Values:")
print(missing_counts)
```



```
Missing Values:
Age                   0
Employment_Type       0
Weighting_Factor      0
Education             0
School_Period         0
Marital_Status        0
Employment_Area       0
Partnership           0
Ethnicity             0
Gender                0
Gain_Financial        0
Losses_Financial      0
Weekly_Working_Time   0
Birth_Country         0
Income                0
dtype: int64
```

Printing missing values

```
print("\nSummary of Missing Values and Data Types:")
print(df1.info())
```



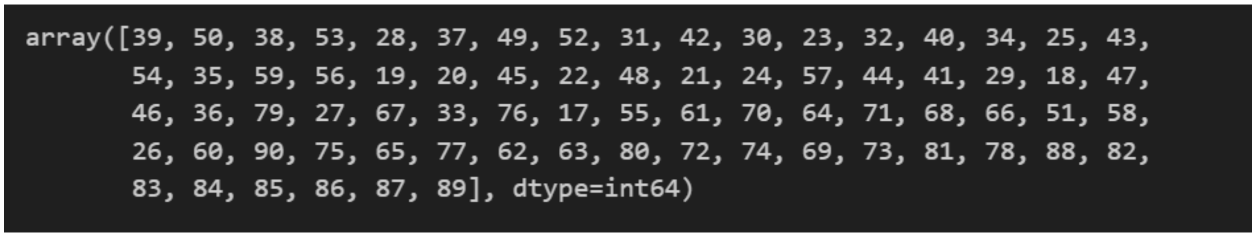
```
Summary of Missing Values and Data Types:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   Age                  48842 non-null  int64 
 1   Employment_Type      48842 non-null  object
 2   Weighting_Factor     48842 non-null  int64 
 3   Education            48842 non-null  object
 4   School_Period        48842 non-null  int64 
 5   Marital_Status       48842 non-null  object
 6   Employment_Area      48842 non-null  object
 7   Partnership          48842 non-null  object
 8   Ethnicity            48842 non-null  object
 9   Gender               48842 non-null  object
 10  Gain_Financial       48842 non-null  int64 
 11  Losses_Financial     48842 non-null  int64 
 12  Weekly_Working_Time  48842 non-null  int64 
 13  Birth_Country        48842 non-null  object
 14  Income               48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
None
```

Drop duplicate values

```
df1 = df1.drop_duplicates()
```

Handling age Variable

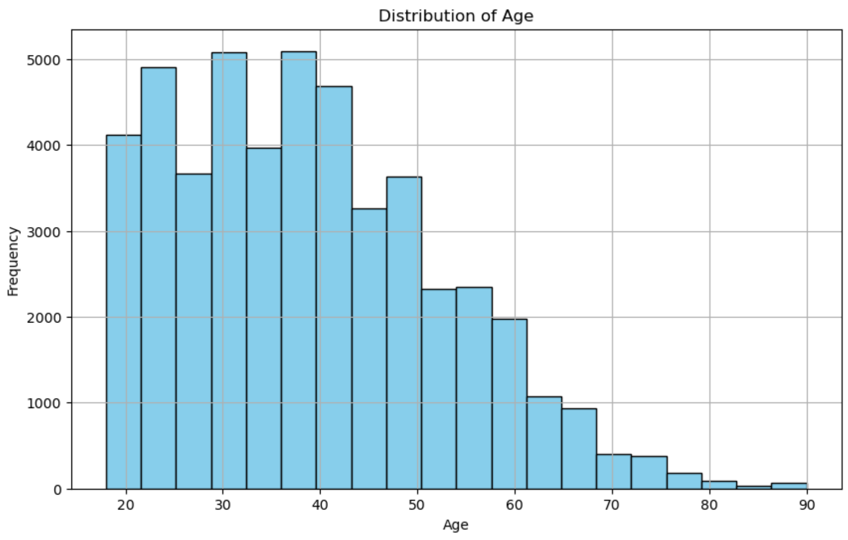
```
df1['Age'].unique()
# here i considerd people over 18 years are adults
# List of values to remove
values_to_remove = [17]

# Filtering to exclude rows with specified values
df1 = df1[~df1['Age'].isin(values_to_remove)]
```



```
array([39, 50, 38, 53, 28, 37, 49, 52, 31, 42, 30, 23, 32, 40, 34, 25, 43,
       54, 35, 59, 56, 19, 20, 45, 22, 48, 21, 24, 57, 44, 41, 29, 18, 47,
       46, 36, 79, 27, 67, 33, 76, 17, 55, 61, 70, 64, 71, 68, 66, 51, 58,
       26, 60, 90, 75, 65, 77, 62, 63, 80, 72, 74, 69, 73, 81, 78, 88, 82,
       83, 84, 85, 86, 87, 89], dtype=int64)
```

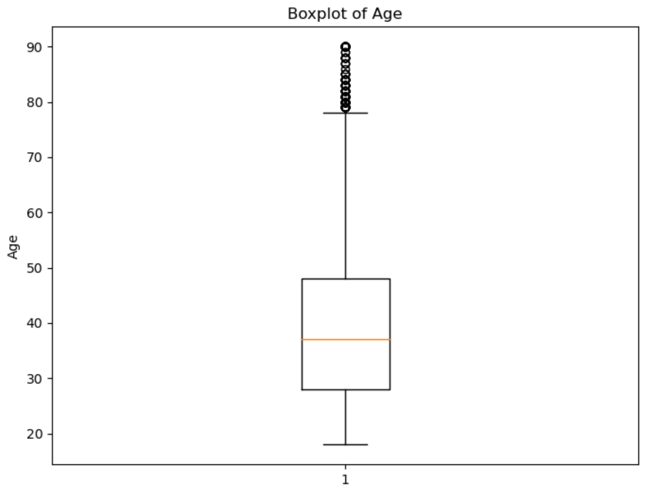
Visualizing age variable before handling outliers. Needed to check how the data is distributed with the age and good idea on data distribution is very important in handling outliers.

```
#Taking the distribution of age and then inspect how age is distributed with the frequency before handling outliers
plt.figure(figsize=(10, 6))
plt.hist(df1['Age'], bins=20, color='skyblue', edgecolor='black')
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



Made the boxplot using IQR method to handle outliers

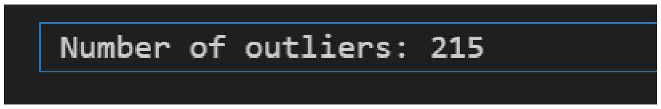
```
# Age
plt.figure(figsize=(8, 6))
plt.boxplot(df1['Age'])
plt.title('Boxplot of Age')
plt.ylabel('Age')
plt.show()
```



Calculated the number of outliers to take the most appropriate decision to handle outliers

```python
import numpy as np

# Calculate quartiles
Q1 = np.percentile(df1['Age'], 25)
Q3 = np.percentile(df1['Age'], 75)

# Calculate IQR
IQR = Q3 - Q1

# Define the outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers
outliers = df1[(df1['Age'] < lower_bound) | (df1['Age'] > upper_bound)]

# Count the number of outliers
num_outliers = len(outliers)

print("Number of outliers:", num_outliers)
```

Number of outliers

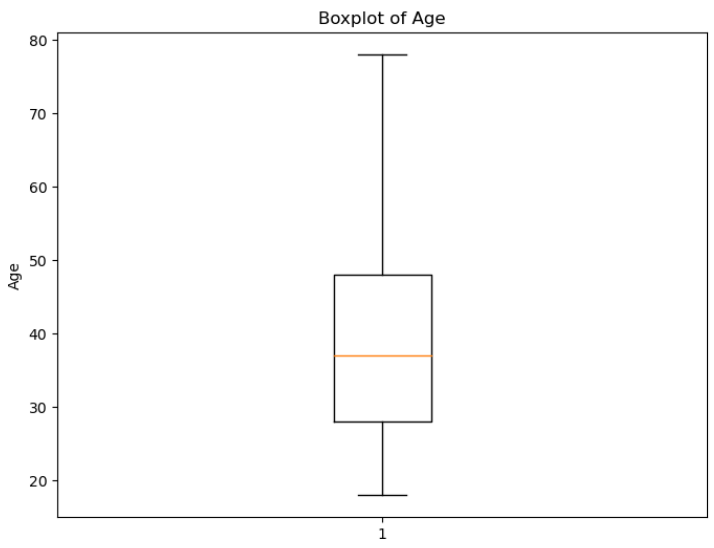


```
Number of outliers: 215
```

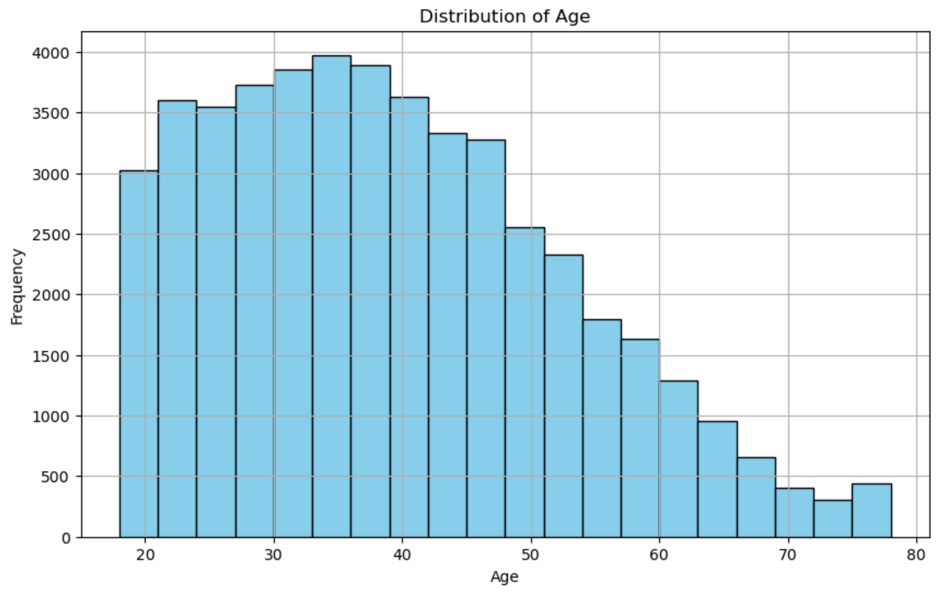
noticed that number of outliers are a little amount when comparing with total dataset and then decided to winsorize it to lower and upper bound of the boxplot.

```python
# Apply Winsorization to replace outliers
df1['Age'] = np.where(df1['Age'] < lower_bound, lower_bound, df1['Age'])
df1['Age'] = np.where(df1['Age'] > upper_bound, upper_bound, df1['Age'])
```

Visualization of boxplot after handling outliers



Distribution of age variable after handling the outliers

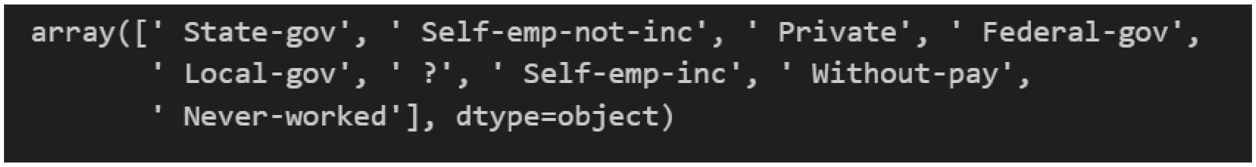


Handling employment type Variable

Found some data inconsistencies and removed those inconsistencies

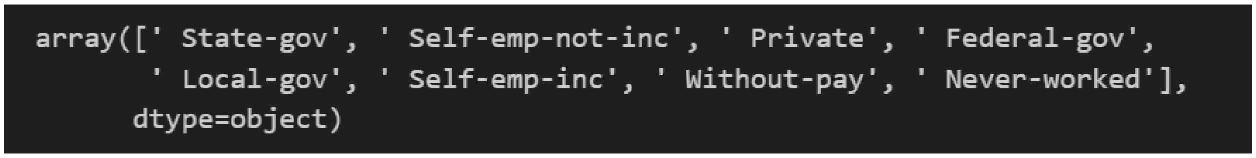
```
df1['Employment_Type'].unique()
# List of values to remove
values_to_remove = [' ?']

# Filter the DataFrame to exclude rows with specified values
df1 = df1[~df1['Employment_Type'].isin(values_to_remove)]
```

Filtered and removed data inconsistencies in the specific variable.



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

After handling inconsistencies

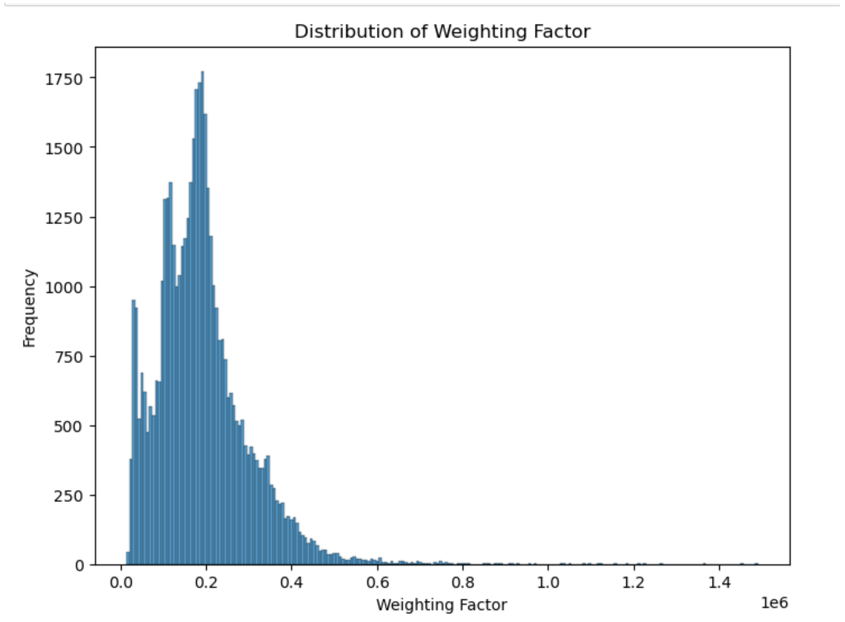


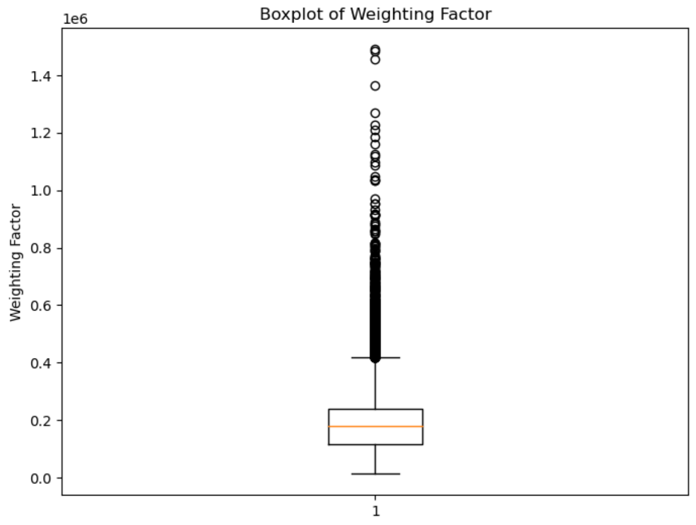
```
array([' State-gov', ' Self-emp-not-inc', ' Private', ' Federal-gov',
       ' Local-gov', ' Self-emp-inc', ' Without-pay', ' Never-worked'],
      dtype=object)
```

Handling weighting factor Variable

In this variable we cannot find proper description on meta data. But visualized the distribution of data and made the boxplot of outliers.

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

# Set the figure size
plt.figure(figsize=(8, 6))

# Plot a histogram of the 'Weighting_Factor' column
sns.histplot(df1['Weighting_Factor'], kde=False)

# Add labels and title
plt.xlabel('Weighting Factor')
plt.ylabel('Frequency')
plt.title('Distribution of Weighting Factor')

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





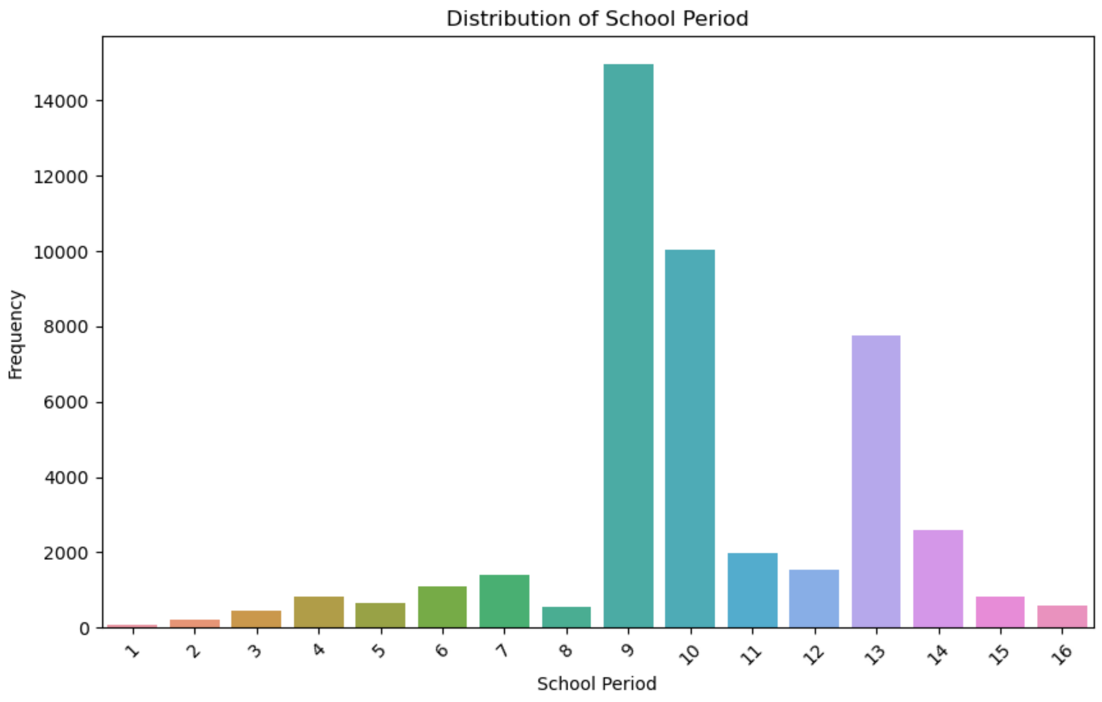
Handling Education and School period variables

Visualized and made a self-inspection on Education and School period.

```
df1['Education'].unique()
df1['School_Period'].unique()

import matplotlib.pyplot as plt
import seaborn as sns

# Set the figure size
plt.figure(figsize=(10, 6))

# Plot a countplot of the 'School_Period' column
sns.countplot(data=df1, x='School_Period')

# Add labels and title
plt.xlabel('School Period')
plt.ylabel('Frequency')
plt.title('Distribution of School Period')

# Rotate x-axis labels for better readability if needed
plt.xticks(rotation=45)

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

```
array([' Bachelors', ' HS-grad', ' 11th', ' Masters', ' 9th',
       ' Some-college', ' Assoc-acdm', ' Assoc-voc', ' 7th-8th',
       ' Doctorate', ' Prof-school', ' 5th-6th', ' 10th', ' Preschool',
       ' 12th', ' 1st-4th'], dtype=object)
```

```
array([13,  9,  7, 14,  5, 10, 12, 11,  4, 16, 15,  3,  6,  1,  8,  2],
      dtype=int64)
```

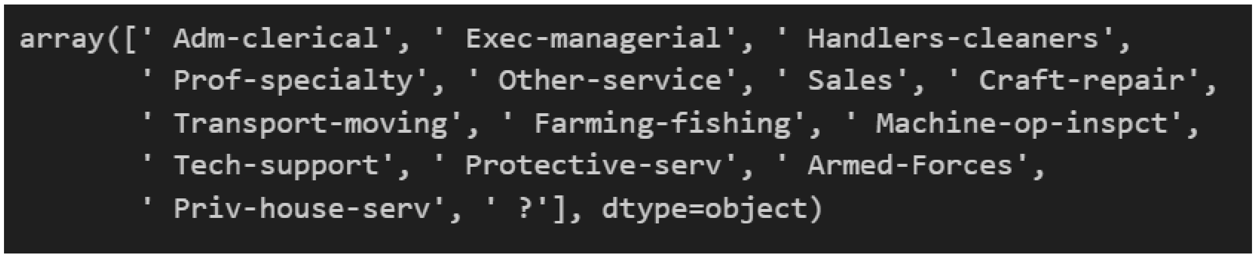


No issues in the distribution of this column and no data inconsistencies.

Handling Employment area column

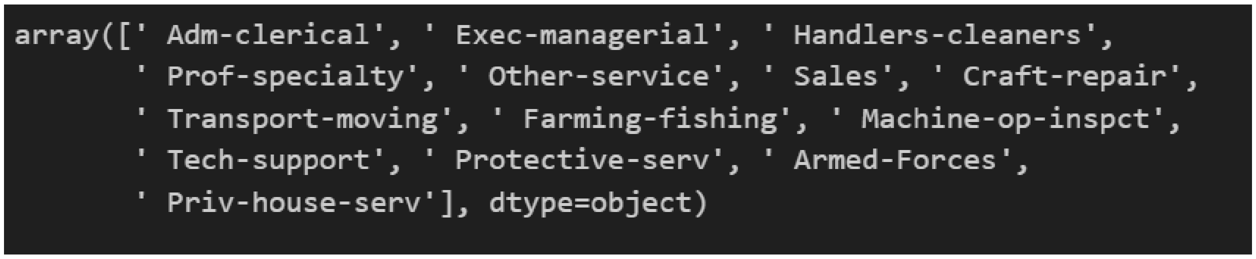
```
df1['Employment_Area'].unique()

# List of values to remove
values_to_remove = [' ?']

# Filter the DataFrame to exclude rows with specified values
df1 = df1[~df1['Employment_Area'].isin(values_to_remove)]
```

Before handling data inconsistencies



```
array([' Adm-clerical', ' Exec-managerial', ' Handlers-cleaners',
       ' Prof-specialty', ' Other-service', ' Sales', ' Craft-repair',
       ' Transport-moving', ' Farming-fishing', ' Machine-op-inspct',
       ' Tech-support', ' Protective-serv', ' Armed-Forces',
       ' Priv-house-serv', ' ?'], dtype=object)
```

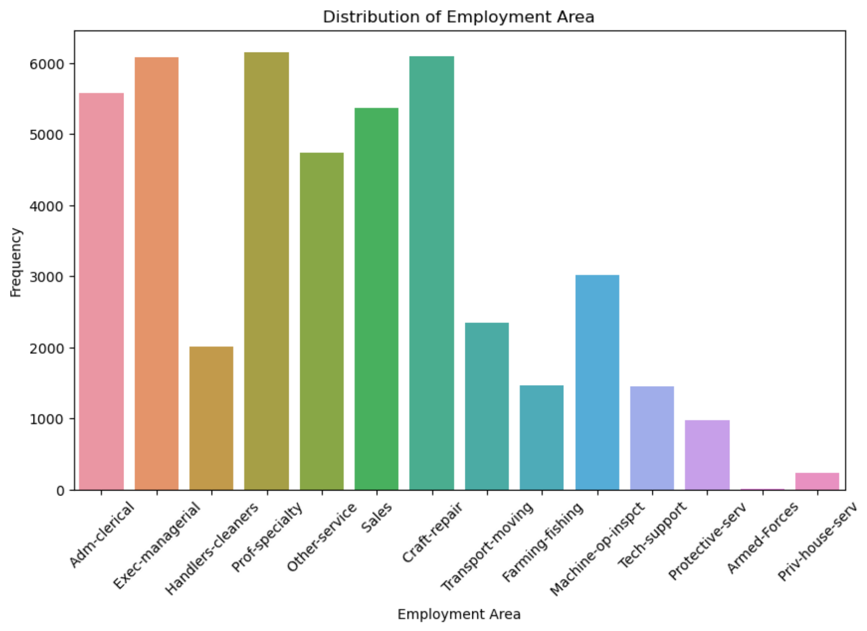
After handling data inconsistencies



```
array([' Adm-clerical', ' Exec-managerial', ' Handlers-cleaners',
       ' Prof-specialty', ' Other-service', ' Sales', ' Craft-repair',
       ' Transport-moving', ' Farming-fishing', ' Machine-op-inspct',
       ' Tech-support', ' Protective-serv', ' Armed-Forces',
       ' Priv-house-serv'], dtype=object)
```

Distribution of data in Employment area

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

# Set the figure size
plt.figure(figsize=(10, 6))

# Plot a countplot of the 'Employment_Area' column
sns.countplot(data=df1, x='Employment_Area')

# Add labels and title
plt.xlabel('Employment Area')
plt.ylabel('Frequency')
plt.title('Distribution of Employment Area')

# Rotate x-axis labels for better readability if needed
plt.xticks(rotation=45)

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

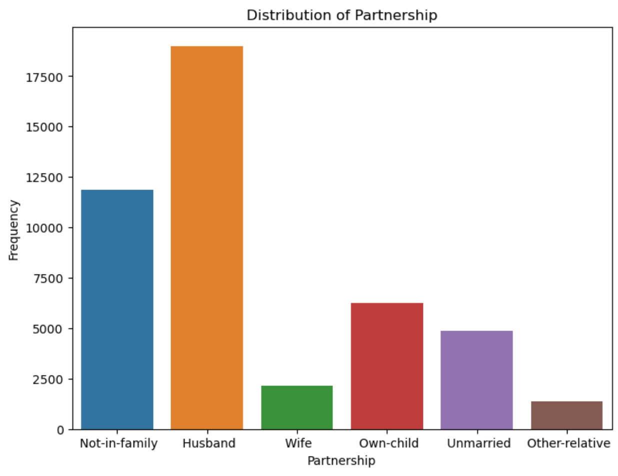


Handling partnership variable

There are no missing or data inconsistencies in this variable but visualized the distribution of data and unique values.

```
df1['Partnership'].unique()

import matplotlib.pyplot as plt
import seaborn as sns

# Set the figure size
plt.figure(figsize=(8, 6))

# Plot a countplot of the 'Partnership' column
sns.countplot(data=df1, x='Partnership')

# Add labels and title
plt.xlabel('Partnership')
plt.ylabel('Frequency')
plt.title('Distribution of Partnership')

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

```
array([' Not-in-family', ' Husband', ' Wife', ' Own-child', ' Unmarried',
       ' Other-relative'], dtype=object)
```



Handling Ethnicity variable

```
df1['Ethnicity'].unique()
# List of values to remove
values_to_remove = [' Other']

# Filter the DataFrame to exclude rows with specified values
df1 = df1[~df1['Ethnicity'].isin(values_to_remove)]

# Set the figure size
plt.figure(figsize=(12, 6))

# Plot a countplot of the 'Ethnicity' column
sns.countplot(data=df1, x='Ethnicity')

# Add labels and title
plt.xlabel('Ethnicity')
plt.ylabel('Frequency')
plt.title('Distribution of Ethnicity')

# Rotate x-axis labels for better readability if needed
plt.xticks(rotation=45)

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

Before handling

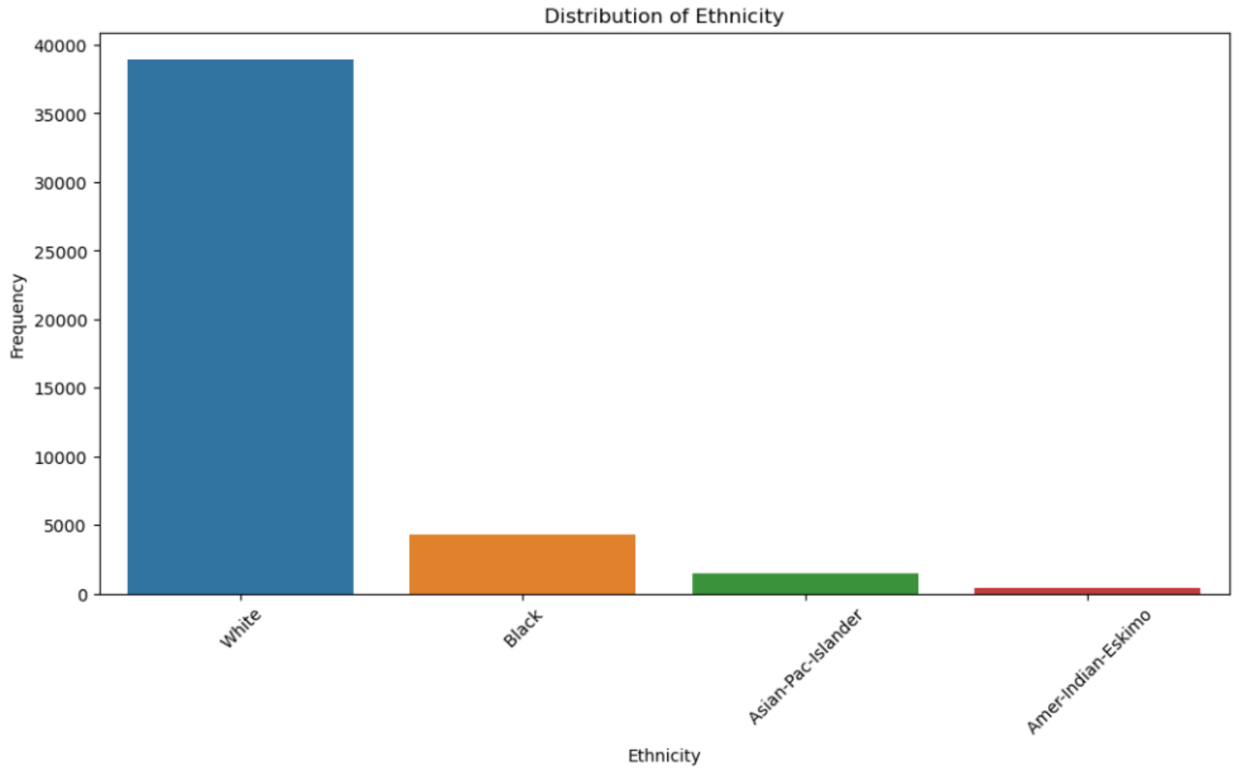
```
array([' White', ' Black', ' Asian-Pac-Islander', ' Amer-Indian-Eskimo',
       ' Other'], dtype=object)
```

There is an unidentified value called "Other" and removed that variable.

After handling

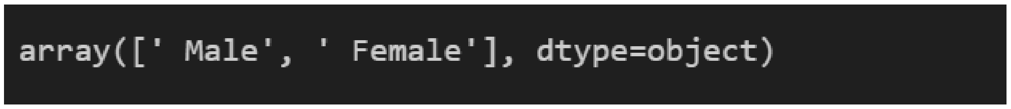
```
array([' White', ' Black', ' Asian-Pac-Islander', ' Amer-Indian-Eskimo'],
      dtype=object)
```

Distribution of data



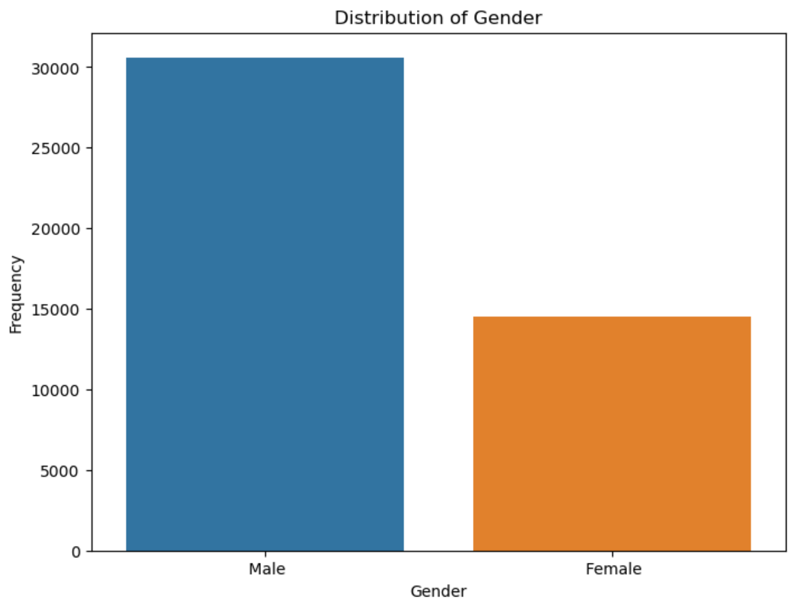
Handling Gender variable

```
df1['Gender'].unique()

# Set the figure size
plt.figure(figsize=(8, 6))

# Plot a countplot of the 'Gender' column
sns.countplot(data=df1, x='Gender')

# Add labels and title
plt.xlabel('Gender')
plt.ylabel('Frequency')
plt.title('Distribution of Gender')

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



```
array([' Male', ' Female'], dtype=object)
```

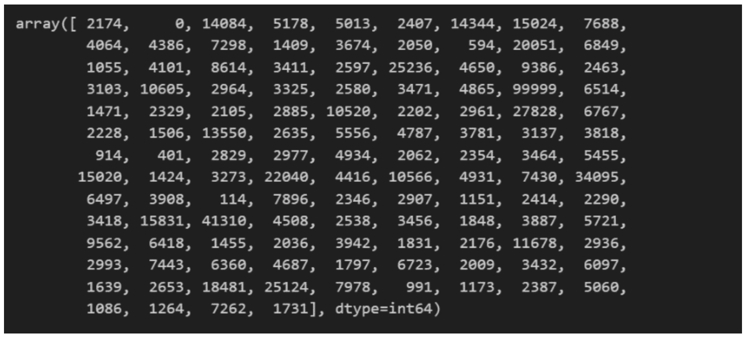
Distribution



Handling Gain financial variable

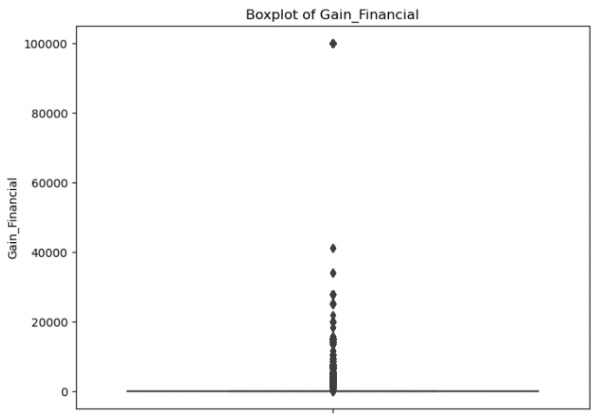
```
df1['Gain_Financial'].unique()

#Boxplot
plt.figure(figsize=(8, 6))
sns.boxplot(data=df1, y='Gain_Financial')
plt.title('Boxplot of Gain_Financial')
plt.show

# selecting and remove outliers
df1 = df1[df1['Gain_Financial'] < 99999]

# after handling
plt.figure(figsize=(8, 6))
sns.boxplot(data=df1, y='Gain_Financial')
plt.title('Boxplot of Gain_Financial')
plt.show
```

Distribution of the array

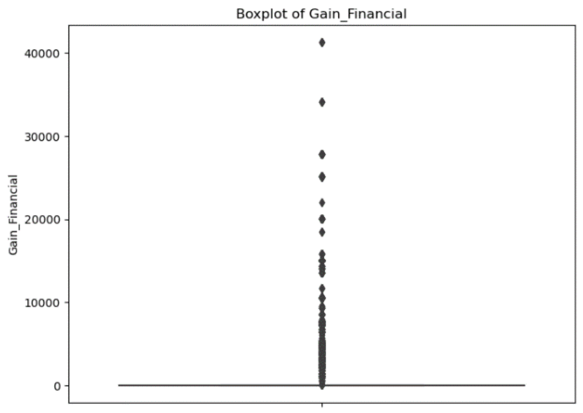


```
array([ 2174,     0, 14084,  5178,  5013,  2407, 14344, 15024,  7688,
        4064,  4386,  7298,  1409,  3674,  2050,   594, 20051,  6849,
        1055,  4101,  8614,  3411,  2597, 25236,  4650,  9386,  2463,
        3103, 10605,  2964,  3325,  2580,  3471,  4865, 99999,  6514,
        1471,  2329,  2105,  2885, 10520,  2202,  2961, 27828,  6767,
        2228,  1506, 13550,  2635,  5556,  4787,  3781,  3137,  3818,
         914,   401,  2829,  2977,  4934,  2062,  2354,  3464,  5455,
       15020,  1424,  3273, 22040,  4416, 10566,  4931,  7430, 34095,
        6497,  3908,   114,  7896,  2346,  2907,  1151,  2414,  2290,
        3418, 15831, 41310,  4508,  2538,  3456,  1848,  3887,  5721,
        9562,  6418,  1455,  2036,  3942,  1831,  2176, 11678,  2936,
        2993,  7443,  6360,  4687,  1797,  6723,  2009,  3432,  6097,
        1639,  2653, 18481, 25124,  7978,   991,  1173,  2387,  5060,
        1086,  1264,  7262,  1731], dtype=int64)
```

Boxplot before handling the outliers.



Boxplot after handling outliers



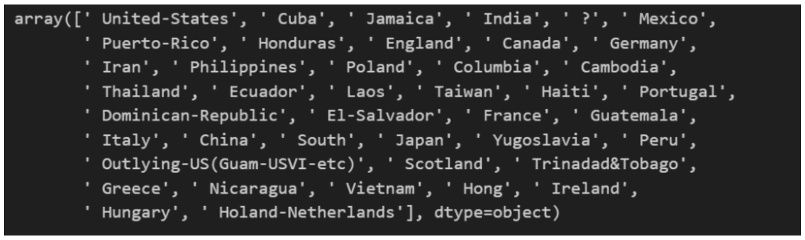
Handling Birth Country variable

```
df1['Birth_Country'].unique()
```

```
# List of values to remove
values_to_remove = [' ?']

# Filter the DataFrame to exclude rows with specified values
df1 = df1[~df1['Birth_Country'].isin(values_to_remove)]
```

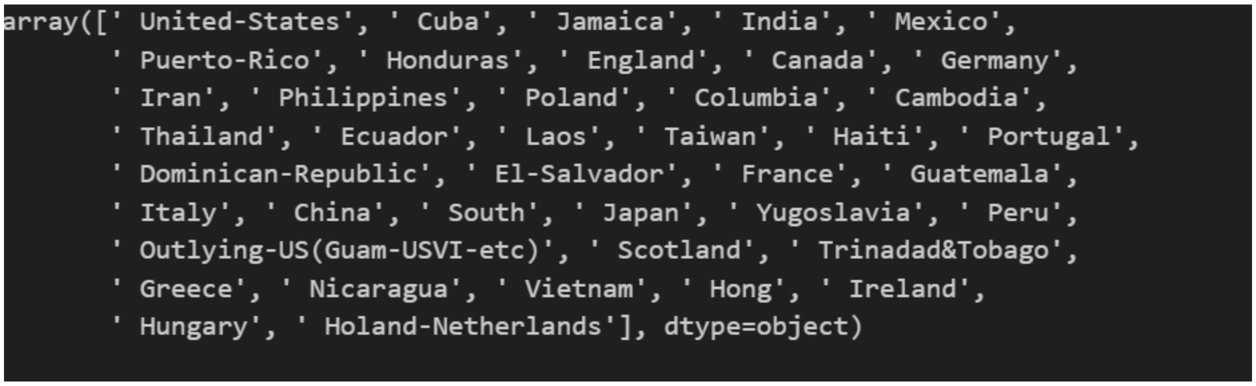
```
import matplotlib.pyplot as plt
import seaborn as sns

# Set the figure size
plt.figure(figsize=(12, 6))

# Plot a countplot of the 'Birth_Country' column
sns.countplot(data=df1, x='Birth_Country')

# Add labels and title
plt.xlabel('Birth Country')
plt.ylabel('Frequency')
plt.title('Distribution of Birth Country')

# Rotate x-axis labels for better readability if needed
plt.xticks(rotation=90)

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

Before handling missing values

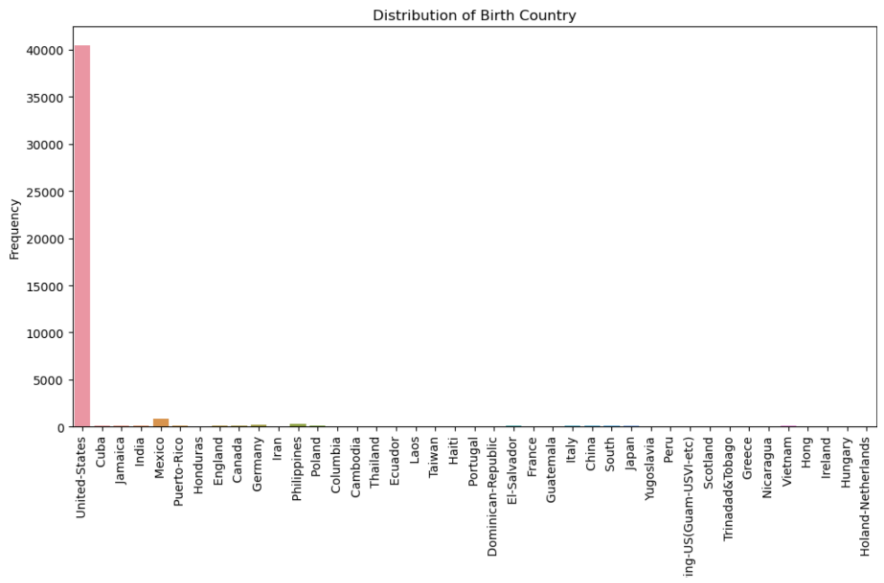


```
array([' United-States', ' Cuba', ' Jamaica', ' India', ' ?', ' Mexico',
       ' Puerto-Rico', ' Honduras', ' England', ' Canada', ' Germany',
       ' Iran', ' Philippines', ' Poland', ' Columbia', ' Cambodia',
       ' Thailand', ' Ecuador', ' Laos', ' Taiwan', ' Haiti', ' Portugal',
       '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

After handling missing values



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



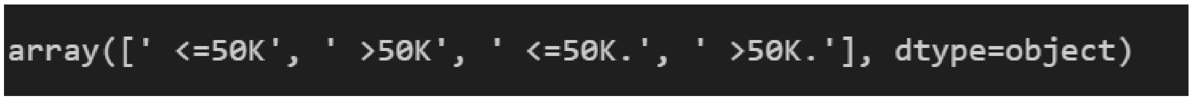
Handling Income Column

```
df1['Income'].unique()
```

```
import matplotlib.pyplot as plt

# Set the figure size
plt.figure(figsize=(10, 6))

# Plot a histogram of the 'Income' column
plt.hist(df1['Income'], bins=20, color='skyblue', edgecolor='black')

# Add labels and title
plt.xlabel('Income')
plt.ylabel('Frequency')
plt.title('Distribution of Income')

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

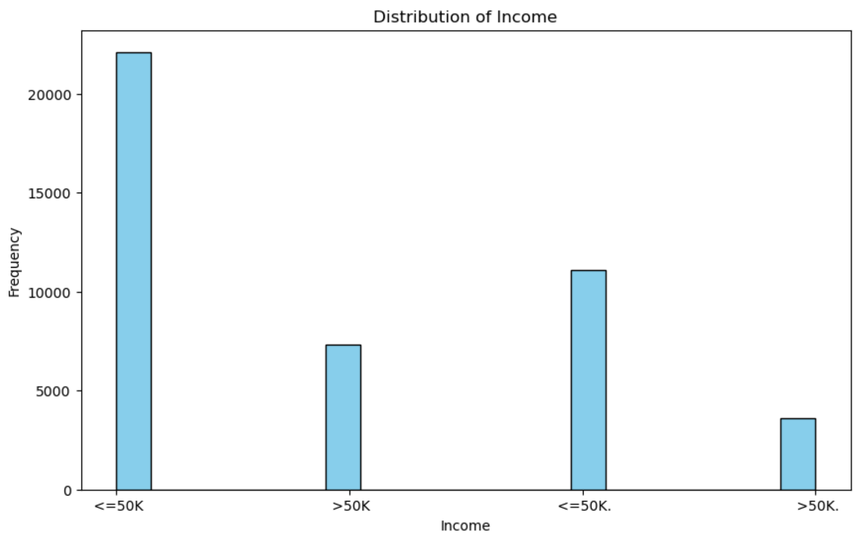
```
# Replace values in the 'Income' column
df1['Income'].replace({' <=50K.': ' <=50K', ' >50K.': ' >50K'}, inplace=True)
```

```
import matplotlib.pyplot as plt

# Set the figure size
plt.figure(figsize=(10, 6))

# Plot a histogram of the 'Income' column
plt.hist(df1['Income'], bins=20, color='skyblue', edgecolor='black')

# Add labels and title
plt.xlabel('Income')
plt.ylabel('Frequency')
plt.title('Distribution of Income')

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

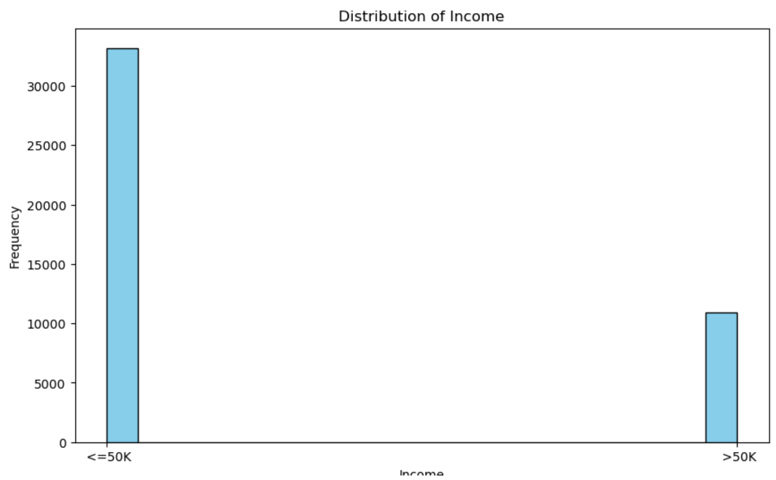


```
array([' <=50K', ' >50K', ' <=50K.', ' >50K.'], dtype=object)
```

Before handling inconsistencies



After handling inconsistencies

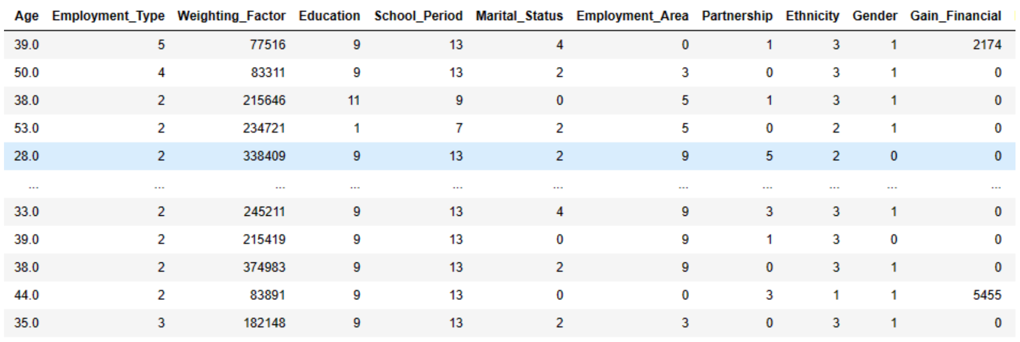


Feature Encoding

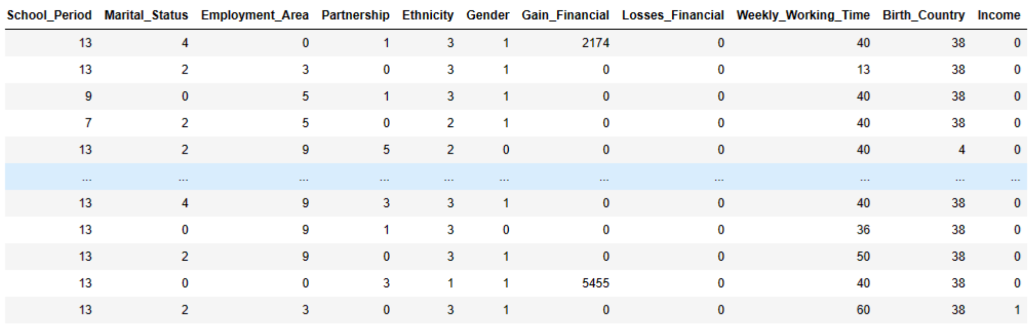
Used label encoder for encoding

```
from sklearn.preprocessing import LabelEncoder

# Assuming df1 is your DataFrame containing categorical features
categorical_columns = ['Marital_Status', 'Employment_Area','Employment_Type',
'Education', 'Partnership', 'Ethnicity', 'Gender', 'Birth_Country', 'Income']  #
Specify the names of categorical columns

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Iterate over each categorical column and encode its values
for col in categorical_columns:
    df1[col] = label_encoder.fit_transform(df1[col])
```

After encoding



Age	Employment_Type	Weighting_Factor	Education	School_Period	Marital_Status	Employment_Area	Partnership	Ethnicity	Gender	Gain_Financial
39.0	5	77516	9	13	4	0	1	3	1	2174
50.0	4	83311	9	13	2	3	0	3	1	0
38.0	2	215646	11	9	0	5	1	3	1	0
53.0	2	234721	1	7	2	5	0	2	1	0
28.0	2	338409	9	13	2	9	5	2	0	0
...	...	...	...	...	...	...	...	...	...	...
33.0	2	245211	9	13	4	9	3	3	1	0
39.0	2	215419	9	13	0	9	1	3	0	0
38.0	2	374983	9	13	2	9	0	3	1	0
44.0	2	83891	9	13	0	0	3	1	1	5455
35.0	3	182148	9	13	2	3	0	3	1	0



School_Period	Marital_Status	Employment_Area	Partnership	Ethnicity	Gender	Gain_Financial	Losses_Financial	Weekly_Working_Time	Birth_Country	Income
13	4	0	1	3	1	2174	0	40	38	0
13	2	3	0	3	1	0	0	13	38	0
9	0	5	1	3	1	0	0	40	38	0
7	2	5	0	2	1	0	0	40	38	0
13	2	9	5	2	0	0	0	40	4	0
...	...	...	...	...	...	...	...	...	...	...
13	4	9	3	3	1	0	0	40	38	0
13	0	9	1	3	0	0	0	36	38	0
13	2	9	0	3	1	0	0	50	38	0
13	0	0	3	1	1	5455	0	40	38	0
13	2	3	0	3	1	0	0	60	38	1

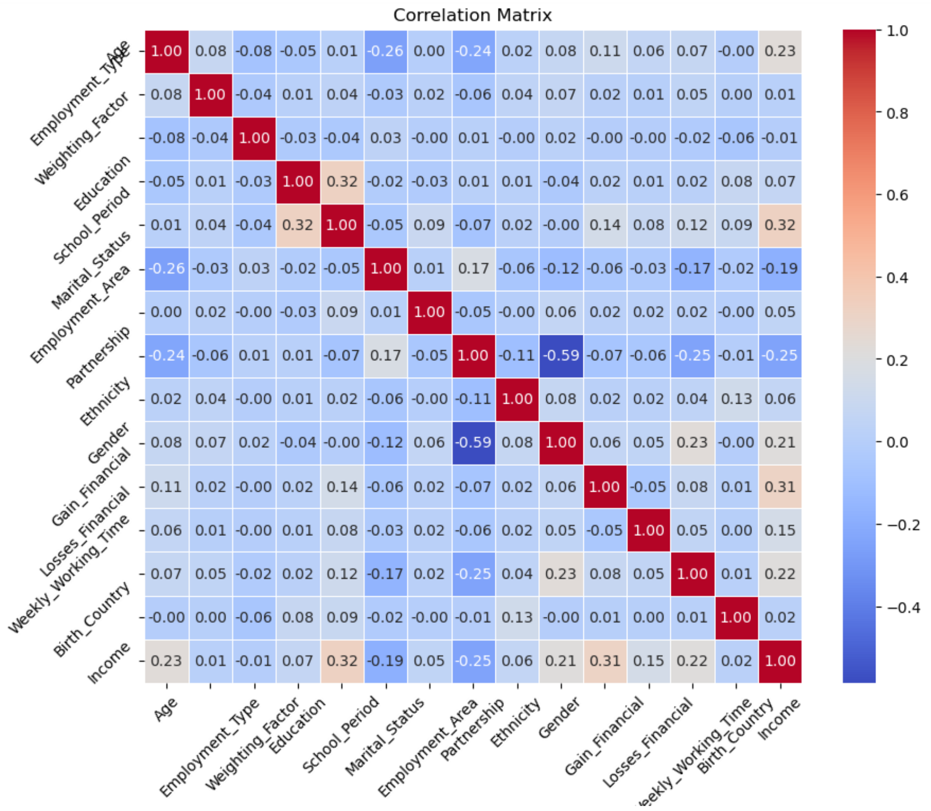
Feature engineering

For selecting best features for training the model used correlation matrix instead of PCA (Principal component analysis). Correlation matrix is the best way to find the most correlated features for the target variable when there are a smaller number of attributes in the dataset.

```
import pandas as pd

correlation_matrix = df1.corr()['Income'].sort_values(ascending=False)

print(correlation_matrix)
```

```
Income                 1.000000
School_Period          0.322467
Gain_Financial         0.306678
Age                    0.228470
Weekly_Working_Time    0.216589
Gender                 0.214485
Losses_Financial       0.151441
Education              0.065742
Ethnicity              0.060593
Employment_Area        0.050757
Birth_Country          0.017166
Employment_Type        0.010318
Weighting_Factor      -0.007856
Marital_Status        -0.186869
Partnership           -0.247844
Name: Income, dtype: float64
```

```
correlation_matrix = df1.corr()

# Set up the matplotlib figure
plt.figure(figsize=(10, 8))

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",
linewidths=0.5)

# Set the title of the plot
plt.title('Correlation Matrix')

# Rotate the tick labels for better readability
plt.xticks(rotation=45)
plt.yticks(rotation=45)

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



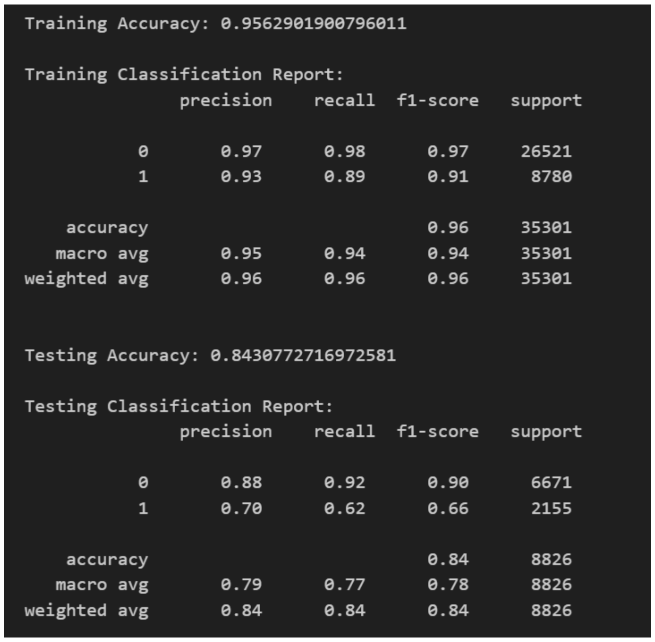
Selecting best features from the correlation matrix and dividing into training and test data

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Split the data into features (X) and target variable (y)
selected_features = ['School_Period', 'Age', 'Weekly_Working_Time',
'Gain_Financial',
                     'Gender', 'Losses_Financial', 'Education', 'Ethnicity',
                     'Employment_Area', 'Marital_Status', 'Partnership']
X = df1[selected_features]
y = df1['Income']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Model Training and Testing using Random forest classifier

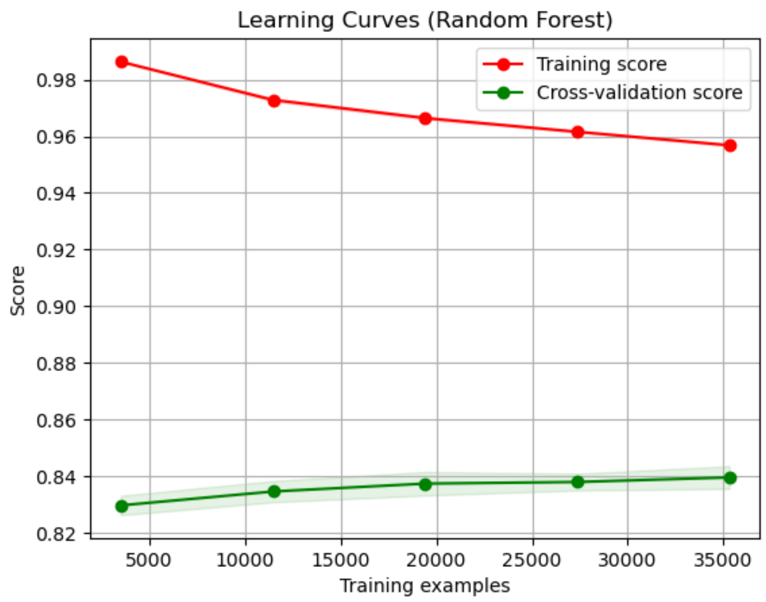
```
# Initialize the RandomForestClassifier
clf = RandomForestClassifier(random_state=42)

# Train the classifier
clf.fit(X_train, y_train)

# Predict on the training set
y_train_pred = clf.predict(X_train)

# Predict on the test set
y_test_pred = clf.predict(X_test)

# Evaluate the classifier for training set
train_accuracy = accuracy_score(y_train, y_train_pred)
train_class_report = classification_report(y_train, y_train_pred)

# Evaluate the classifier for testing set
test_accuracy = accuracy_score(y_test, y_test_pred)
test_class_report = classification_report(y_test, y_test_pred)
```

Results of random forest algorithm



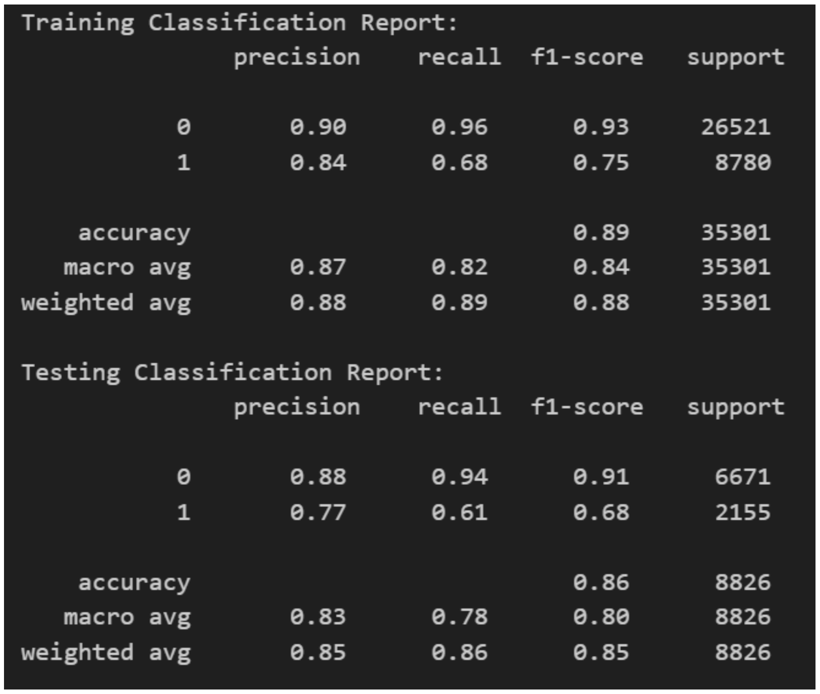
```
Training Accuracy: 0.9562901900796011

Training Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.98      0.97     26521
           1       0.93      0.89      0.91      8780

    accuracy                           0.96     35301
   macro avg       0.95      0.94      0.94     35301
weighted avg       0.96      0.96      0.96     35301


Testing Accuracy: 0.8430772716972581

Testing Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.92      0.90      6671
           1       0.70      0.62      0.66      2155

    accuracy                           0.84      8826
   macro avg       0.79      0.77      0.78      8826
weighted avg       0.84      0.84      0.84      8826
```

Classification reports are ok but there is a clear gap between training and testing data. For visualize that I used training and test curves.



Then used Hyperparameter tunning for tunning process and I used GridSearchCv method for this

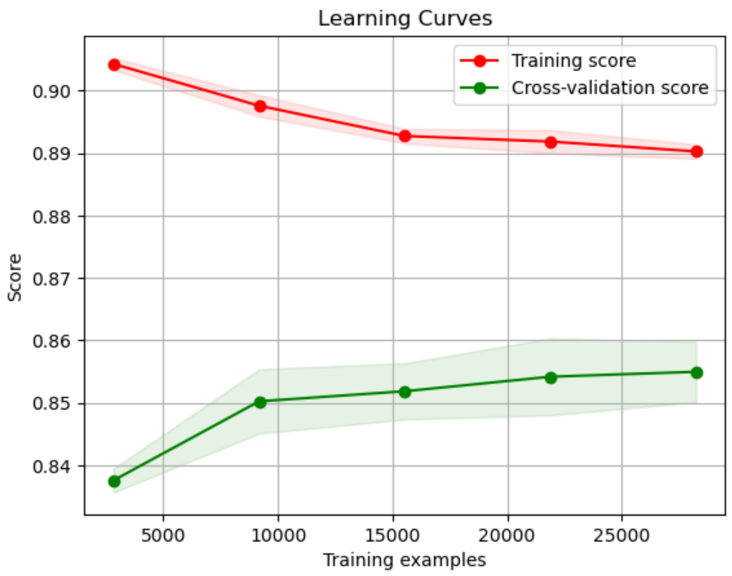
```
from sklearn.model_selection import GridSearchCV

# Define the hyperparameters grid
param_grid = {
    'n_estimators': [50, 100, 200],  # Number of trees in the forest
    'max_depth': [None, 10, 20],  # Maximum depth of the tree
    'min_samples_split': [2, 5, 10],  # Minimum samples required to split a node
    'min_samples_leaf': [1, 2, 4]  # Minimum samples required at each leaf node
}

# Initialize the RandomForestClassifier
clf = RandomForestClassifier(random_state=42)

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, cv=5,
scoring='accuracy')

# Perform Grid Search to find the best parameters
grid_search.fit(X_train, y_train)

# Get the best parameters and best estimator
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

# Train the best estimator
best_estimator.fit(X_train, y_train)

# Predict on the test set using the best estimator
y_pred_test_tuned = best_estimator.predict(X_test)

# Evaluate the best estimator on the testing set
test_accuracy_tuned = accuracy_score(y_test, y_pred_test_tuned)

print("Best Parameters:", best_params)
print("Best Training Accuracy after Tuning:", best_estimator.score(X_train,
y_train))
print("Testing Accuracy after Tuning:", test_accuracy_tuned)
```



```
Training Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.96      0.93     26521
           1       0.84      0.68      0.75      8780

    accuracy                           0.89     35301
   macro avg       0.87      0.82      0.84     35301
weighted avg       0.88      0.89      0.88     35301

Testing Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.94      0.91      6671
           1       0.77      0.61      0.68      2155

    accuracy                           0.86      8826
   macro avg       0.83      0.78      0.80      8826
weighted avg       0.85      0.86      0.85      8826
```

After evaluation learning curves

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve

# Define a function to plot learning curves
def plot_learning_curves(estimator, title, X, y, ylim=None, cv=None, n_jobs=None,
train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(estimator, X, y,
cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt

# Plot learning curves
plot_learning_curves(best_estimator, "Learning Curves", X_train, y_train, cv=5,
n_jobs=-1)
plt.show()
```



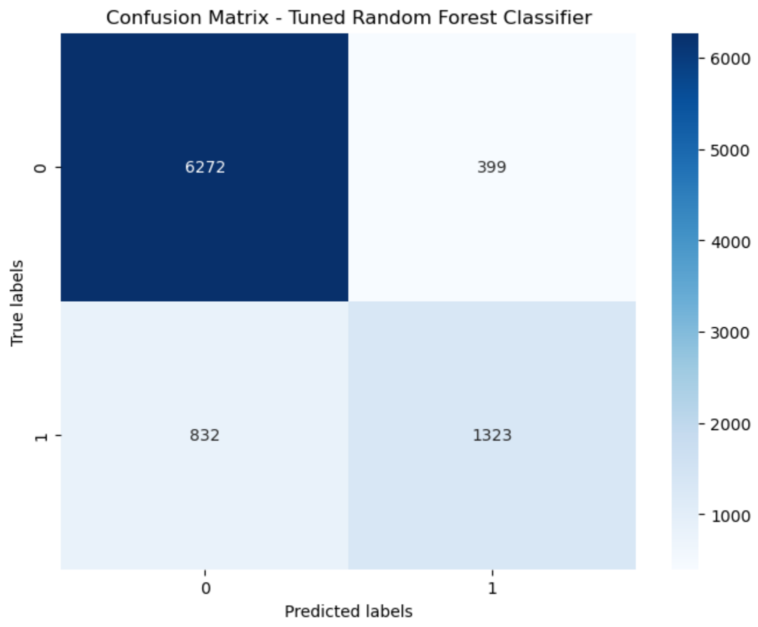
Applied cross validation to validate further

```
from sklearn.model_selection import cross_val_score

# Initialize the RandomForestClassifier with the best parameters
clf_cv = RandomForestClassifier(**best_params, random_state=42)

# Perform cross-validation
cv_scores = cross_val_score(clf_cv, X, y, cv=5)  # 5-fold cross-validation

# Calculate mean cross-validation accuracy
mean_cv_accuracy = cv_scores.mean()

print("Mean Cross-Validation Accuracy:", mean_cv_accuracy)
```

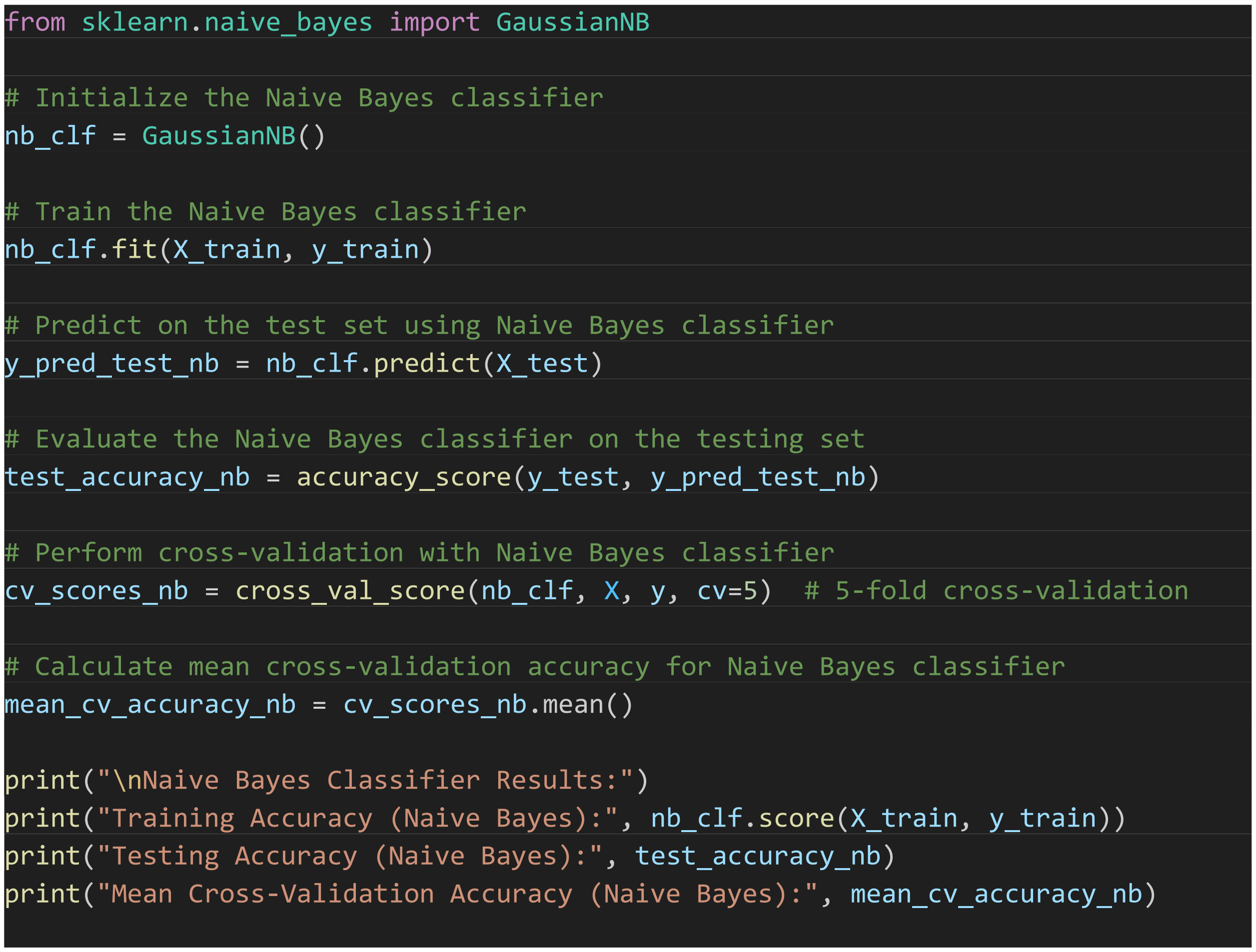
Result

```
Mean Cross-Validation Accuracy: 0.8577061540426849
```

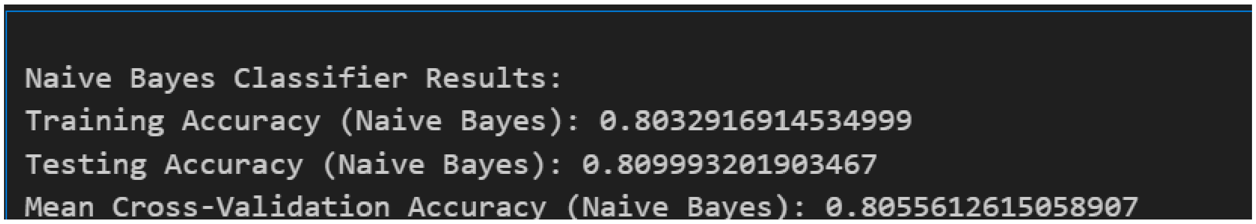
Confusion matrix


Confusion Matrix - Tuned Random Forest Classifier
6272
399
832
1323
True labels
Predicted labels
0
1
0
1
6000
5000
4000
3000
2000
1000

Model Training and Testing using Naïve Bayes algorithm

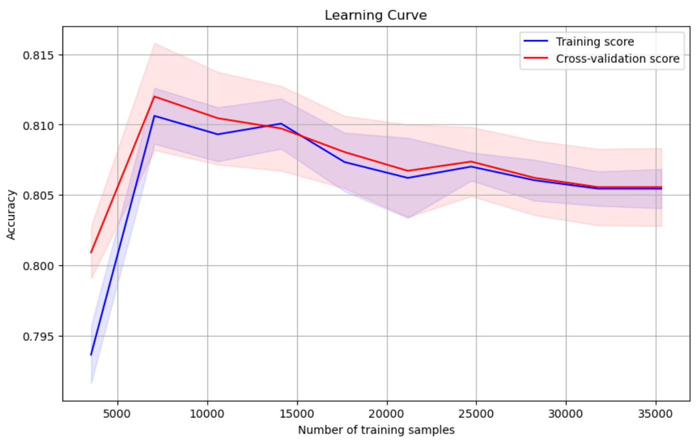


```python
from sklearn.naive_bayes import GaussianNB

# Initialize the Naive Bayes classifier
nb_clf = GaussianNB()

# Train the Naive Bayes classifier
nb_clf.fit(X_train, y_train)

# Predict on the test set using Naive Bayes classifier
y_pred_test_nb = nb_clf.predict(X_test)

# Evaluate the Naive Bayes classifier on the testing set
test_accuracy_nb = accuracy_score(y_test, y_pred_test_nb)

# Perform cross-validation with Naive Bayes classifier
cv_scores_nb = cross_val_score(nb_clf, X, y, cv=5)  # 5-fold cross-validation

# Calculate mean cross-validation accuracy for Naive Bayes classifier
mean_cv_accuracy_nb = cv_scores_nb.mean()

print("\nNaive Bayes Classifier Results:")
print("Training Accuracy (Naive Bayes):", nb_clf.score(X_train, y_train))
print("Testing Accuracy (Naive Bayes):", test_accuracy_nb)
print("Mean Cross-Validation Accuracy (Naive Bayes):", mean_cv_accuracy_nb)
```



```
Naive Bayes Classifier Results:
Training Accuracy (Naive Bayes): 0.8032916914534999
Testing Accuracy (Naive Bayes): 0.809993201903467
Mean Cross-Validation Accuracy (Naive Bayes): 0.8055612615058907
```

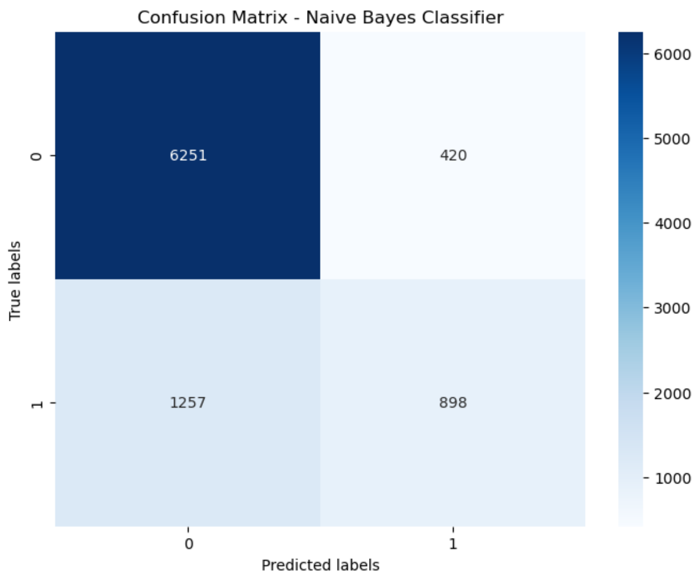
Training and testing curves



Evaluation results

```
Classification Report (Naive Bayes Classifier):
              precision    recall  f1-score   support

           0       0.83      0.94      0.88      6671
           1       0.68      0.42      0.52      2155

    accuracy                           0.81      8826
   macro avg       0.76      0.68      0.70      8826
weighted avg       0.80      0.81      0.79      8826
```

Confusion matrix



Solution methodology

Random forest and Naïve Bayes algorithms are used for classify salary. Summary of two algorithms that used were given below.

Random forest - This is a popular and adaptable ensemble learning technique for tasks involving regression and classification. It's a decision tree extension in which several decision trees are trained using various training data subsets and then combined to provide predictions.

- Multiple subsets are created by bootstrapping, which involves sampling random subsets of the training data with replacement.
- There is a decision tree constructed for every data subset. Nevertheless, a random subset of features is chosen for splitting at each node of the tree rather than all of the features. The trees are better decorated because of this unpredictability.
- To arrive at the final forecast, predictions from individual trees are aggregated by voting (for classification) or averaging (for regression).

Random forests are renowned for their high accuracy, scalability to huge datasets, and resilience against overfitting. When working with multi-feature, high-dimensional data, they are especially useful.

Naïve Bayes – This algorithm relies on the premise of predictor independence and is a probabilistic classification method that applies the Bayes theorem.

- Using the Bayes theorem, it determines the likelihood of each class given a collection of input features.
- The "naive" assumption of Naive Bayes simplifies probability calculations by assuming that the existence of a given characteristic in a class is independent of the existence of any other feature (i.e., features are analyzed independently).
- The class with the highest probability is then chosen to be the input data's predicted class.

In order to estimate the required parameters, naive Bayes classifiers require very minimal training data and are quick and simple to deploy. However, feature independence may not always hold true, which in some circumstances may result in less than ideal performance.

Evaluation Criteria

For income prediction use case used accuracy, precision, recall, and F1- scores as evaluation metrices and shown here, along with the explanations for their application.

Accuracy - measure, which indicates the percentage of correctly predicted occurrences among all instances in a dataset, is used to evaluate the performance of a machine learning model.

$$\boxed{Accuracy = \frac{TP + TN}{TP + TN + FP + FN}}$$

Precision - metric that evaluates the actual positive forecasts' accuracy

$$\boxed{Precison = \frac{TP}{TP + FP}}$$

Recall - The proportion of cases that were accurately classified.

$$\boxed{Recall = \frac{TP}{TP + FN}}$$

F1 Score - A model accuracy indicator that combines precision and recall into a single number.

$$\boxed{F1\ score = \frac{2 \times (precision \times recall)}{precision + recall}}$$

Evaluation results

	Random forest algorithm	Naïve Bayes algorithm
Train and test split	Train – 80% Test – 20%	Train – 80% Test – 20%
Test data accuracy	86%	80%
Train data accuracy	89%	80%
Precision	0.88	0.83
Recall	0.94	0.94
F1 Score	0.91	0.88

GitHub URL

https://github.com/Inupa6677/Machine-Learning-CW