# Project Report

## DataScience Course/Graded Assessment 1

Prepared by: Georges Assaf
Date: Nov 2024

This document contains 2 Reports:

- Report 1: created based on the clarification sent by email
- Report 2: It was created before we received the clarifications, and it covers all tasks of the assignment and employs Linear Regression and many other classifiers with all the variables of the dataset and contains a description that covers all the steps of the project

# Report 1

Report 1 covers only the linear regression as per the clarification mail shared with us

## Introduction

This project involves working with a dataset derived from the 1994 US Census for analyzing income patterns. We will explore data relationships, handle missing values, and perform linear regression and other classifier to uncover insights about working hours per week for male and female The assignment requires submitting a Jupyter Notebook. The section in the notebook that covers this section is called Machine learning as requested in the clarification email

## Scope

The objective of this project is to perform multiple linear regression fits, each time adding an additional control variable, and observe the trend of weekly working hours between men and woman.

- In the first model, we will fit a linear regression using only the "sex" variable.
- In the second model, we will include an additional control variable, "education-num."
- In the third model, a third control variable, "income," will be incorporated.

### Insights from Original Dataset

as a basic analysis and to have an insight about the trend of working hours , I have plotted the average of working hours per week in the actual/original dataset and group them by sex.

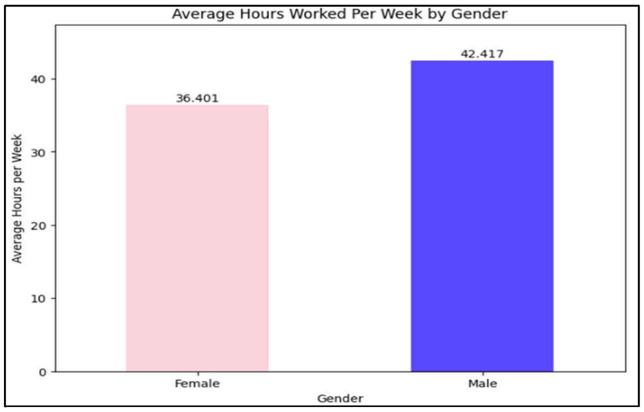


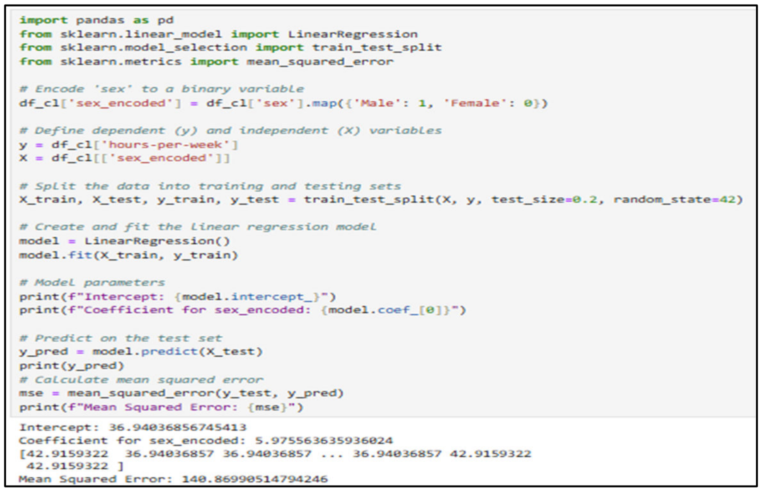
Figure 1-Avg hours per week in original data

This chart compare the average number of hours worked per week between male and female. The blue bar representing the average working hours per week for male is significantly higher than the pink bar representing females. The difference is approximately around 6 hours per week.

### Statistic Used

In linear regression several statistical metrics are used to evaluate the performance of the model. The most common one used are MSE, RMSE,etc. In this exercise, I will be using the MSE (Mean Squared Error) that measures the average squared difference between the actual and predicted values. Lower MSE indicates better model performance.

## Linear Regression with Sex



```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Encode 'sex' to a binary variable
df_cl['sex_encoded'] = df_cl['sex'].map({'Male': 1, 'Female': 0})

# Define dependent (y) and independent (X) variables
y = df_cl['hours-per-week']
X = df_cl[['sex_encoded']]

# 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)

# Create and fit the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Model parameters
print(f"Intercept: {model.intercept_}")
print(f"Coefficient for sex_encoded: {model.coef_[0]}")

# Predict on the test set
y_pred = model.predict(X_test)
print(y_pred)
# Calculate mean squared error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Intercept: 36.94036856745413
Coefficient for sex_encoded: 5.975563635936024
[42.9159322  36.94036857 36.94036857 ... 36.94036857 42.9159322
 42.9159322 ]
Mean Squared Error: 140.86990514794246
```

Figure 2-Linear Regression [sex]

Using Sex as independent variable, I ran this model and got the followings:
Intercept: and intercept of 36.9 represent a baseline of 36.9 hours per week
Coefficient: a coefficient for sex of 5.9 represents the difference in weekly working hours between the sex categories.
Mean Squared Error: This is average squared error. MSE= 140 indicates that there is some error in the model's prediction and it could be potentially improved

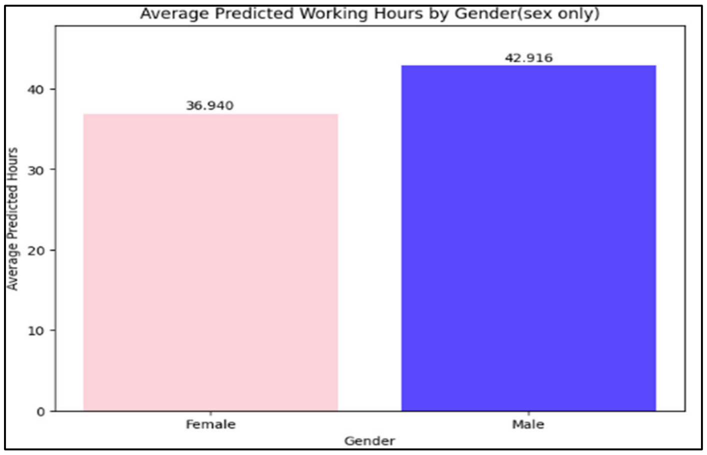
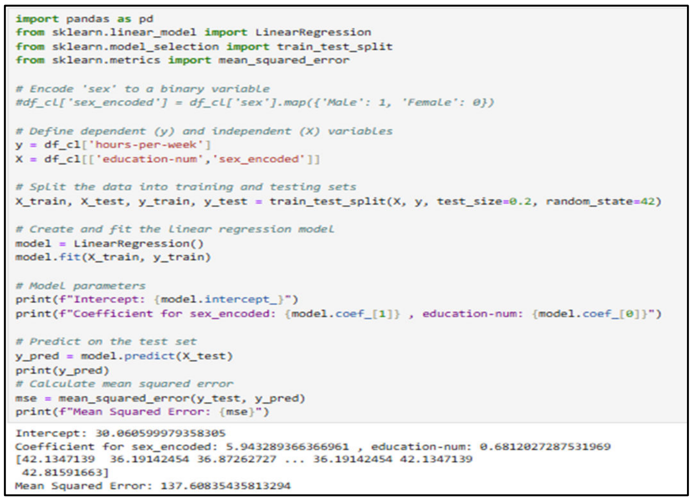


Figure 3-Predicted Avg Hours Per Week [sex]

This bar chart displays the predicted average weekly working hours, segmented by male and female. It is evident that the trend in the predicted data, generated by the model that includes sex as an independent variable, closely aligns with the trend observed in the original data.

## Linear Regression with Sex and education-num



```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Encode 'sex' to a binary variable
#df_cl['sex_encoded'] = df_cl['sex'].map({'Male': 1, 'Female': 0})

# Define dependent (y) and independent (X) variables
y = df_cl['hours-per-week']
X = df_cl[['education-num','sex_encoded']]

# 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)

# Create and fit the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Model parameters
print(f"Intercept: {model.intercept_}")
print(f"Coefficient for sex_encoded: {model.coef_[1]} , education-num: {model.coef_[0]}")

# Predict on the test set
y_pred = model.predict(X_test)
print(y_pred)
# Calculate mean squared error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Intercept: 30.060599979358305
Coefficient for sex_encoded: 5.943289366366961 , education-num: 0.6812027287531969
[42.1347139  36.19142454 36.87262727 ... 36.19142454 42.1347139
 42.81591663]
Mean Squared Error: 137.60835435813294
```

Figure 4-Linear Regression [education-num, sex]

Coefficient: The coefficient 0.68 for education-num indicates a relation between **education-num** and number of working hours. Individual with higher education levels if education tend to work slightly more hours per week. That verify why we have a slight decrease in MSE.
MSE: Adding a new control variable education-num seems to have an impact on reducing the error.between the average actual and predicted values of weekly working hours. Lower MSE indicates a better model.

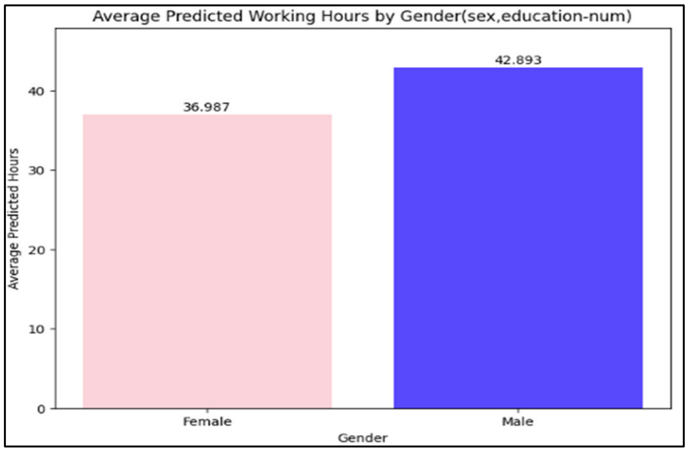
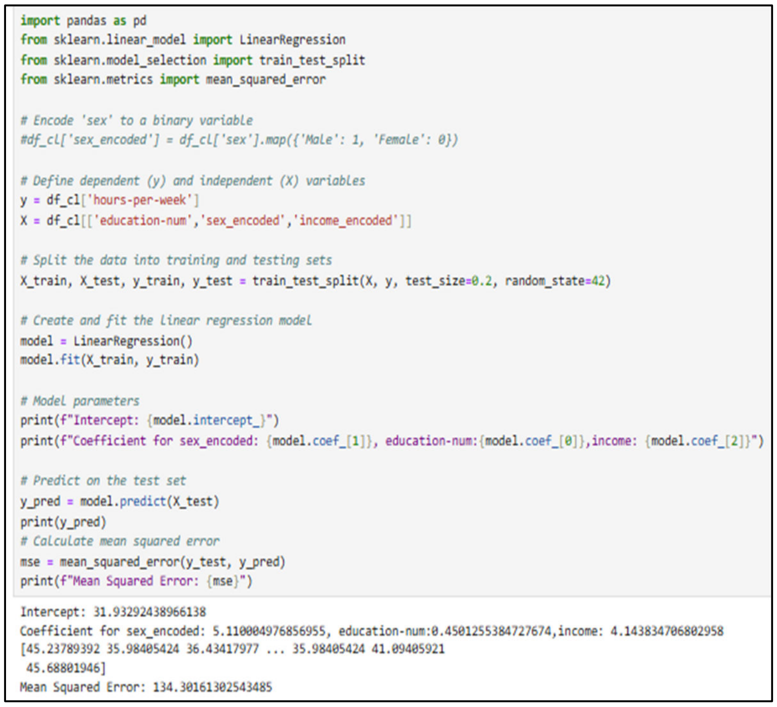


Figure 5-Predicted Avg Hours Per Week [education-num, sex]

After introducing the new control variable, education-num, the trend showing that males work more hours per week remains unchanged. However, there is a slight variation in the number of working hours observed.

## Linear Regression with Sex, education-num, and income



```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Encode 'sex' to a binary variable
#df_cl['sex_encoded'] = df_cl['sex'].map({'Male': 1, 'Female': 0})

# Define dependent (y) and independent (X) variables
y = df_cl['hours-per-week']
X = df_cl[['education-num','sex_encoded','income_encoded']]

# 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)

# Create and fit the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Model parameters
print(f"Intercept: {model.intercept_}")
print(f"Coefficient for sex_encoded: {model.coef_[1]}, education-num:{model.coef_[0]},income: {model.coef_[2]}")

# Predict on the test set
y_pred = model.predict(X_test)
print(y_pred)
# Calculate mean squared error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Intercept: 31.93292438966138
Coefficient for sex_encoded: 5.110004976856955, education-num:0.4501255384727674,income: 4.143834706802958
[45.23789392 35.98405424 36.43417977 ... 35.98405424 41.09405921
 45.68801946]
Mean Squared Error: 134.30161302543485
```

Figure 6-Linear Regression [education-num, sex, income]

MSE: Adding a third control variable seems to have an impact of reducing more the error. This can verified by looking at the coefficient of income (4.14) that means for each one-unit increase in income, the model predict 4.14 hours increase in weekly working hours. This indicates that people with higher income are expected to work more hours per week/

We can conclude that adding more variable is helping the model to capture more patterns and thus reducing the error.

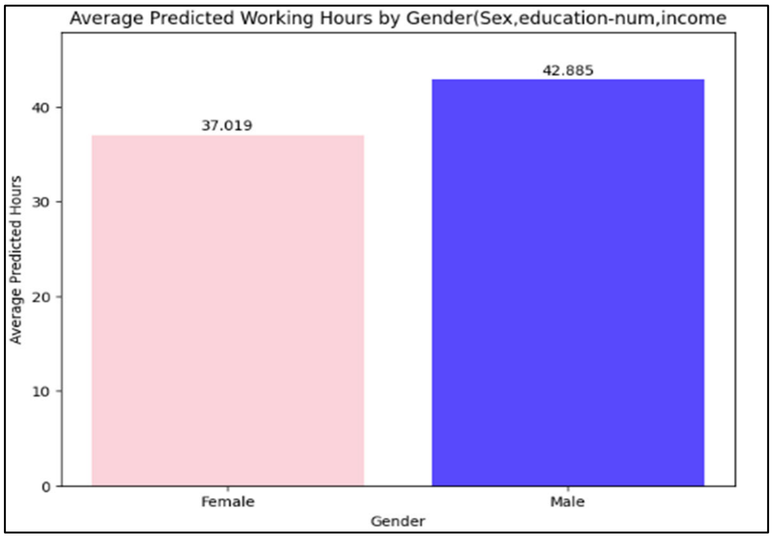


Figure 7-Predicted Avg Hours Per Week [education-num, sex, income]

Adding a third binary variable income to the model, does not have an impact on changing the trend of weekly working hours between male and female. However we noticed a small variation in the number of hours.

## Conclusion:

To improve the model, we should consider the followings:

- Adding more features
- Exploring other types of regression models or regularization techniques.

In the next report, I am analyze the entire dataset with income as the dependent variable, apply various models, and conclude with a comparison of their performance to identify the best model.

# Report 2

NB: this report covers all the steps available in the notebook that follows the steps of the assignment that are mainly grouped under 3 main groups:

- Data Exploration
- Correlation
- Linear Regression and other classifier models (XGBoost, Logistic Regression, RandomForest) where income is considered the dependent variable and all remaining variables are used

## Introduction

This project involves working with a dataset derived from the 1994 US Census for analyzing income patterns. We will explore data relationships, handle missing values, and perform linear regression and other classifier to uncover insights about income trends. The assignment requires submitting a Jupyter Notebook with analysis and a brief report summarizing the findings.

## Data Description

### Dataset Overview

The dataset, extracted from the 1994 US Census, includes demographic and employment-related attributes to analyze income patterns and trends.

### Key Features

The dataset, derived from the 1994 US Census, contains a mix of categorical and numerical features representing demographic and employment information. Numerical features include **age**, **fnlwgt** (sampling weight), **education_num** (numerical education level), **capital_gain**, **capital_loss**, and **hours_per_week**. Categorical features capture aspects like **workclass**, **education**, **marital_status**, **occupation**, **relationship**, **race**, **sex**, and **native_country**.

### Target Variable

The target variable, **income**, is categorical and indicates whether an individual's income exceeds $50K annually.

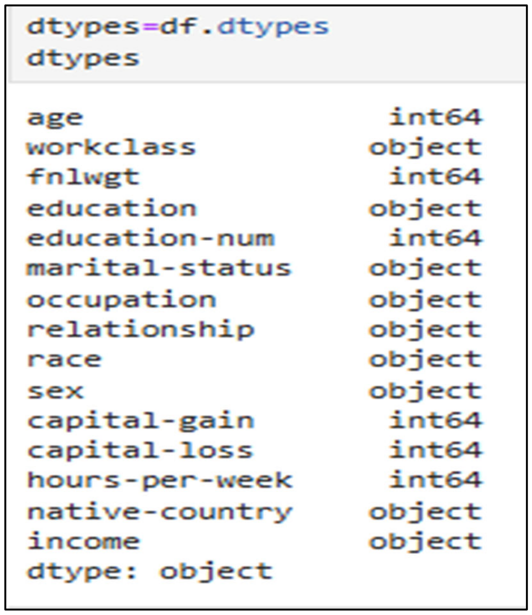
### Challenges

This dataset seems to have challenges, like outliers, missing values and inconstancies in the column values.

## Data exploration

### Column Data Types

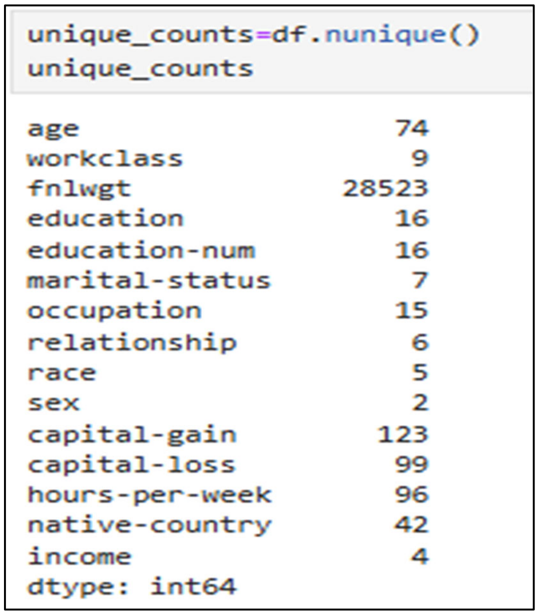
After importing the dataset from the library ucimlrepo, I have validated the column types comparing to the dataset column type with the text description sheet. The integer columns are easily comparable since the same type is on both.
However, for the object columns in the imported dataset, I had to run df.nunique() to have a look at the number of distinct values which can give an idea whether the column is categorical or not, all are matching.
For the binary columns in the dataset, I checked the distinct values and noticed that for **sex** has 2 distinct values while income is having 4 distinct values. To dig more, I had to print the distinct values of income and noticed that there is a typos error (>50K.  and <=50K.) which will updated later in the project to have a total of 2 distinct values.



```
dtypes=df.dtypes
dtypes
```

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

Figure 8-Column Type



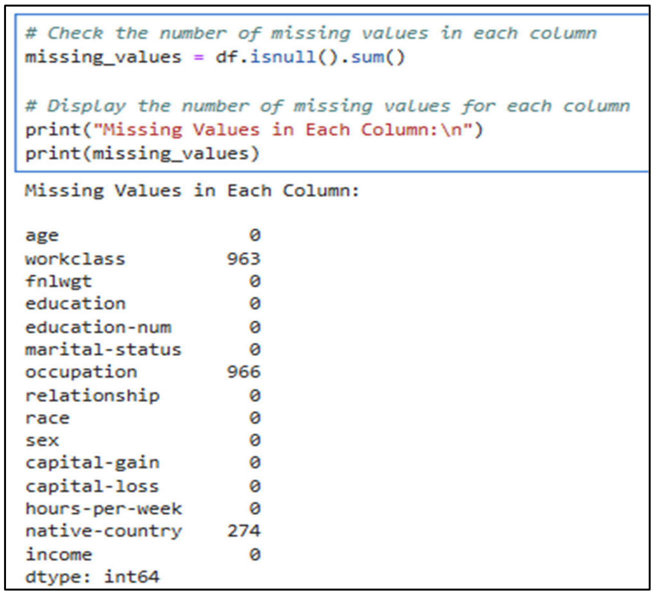
```
unique_counts=df.nunique()
unique_counts
```

```
age                  74
workclass             9
fnlwgt            28523
education            16
education-num        16
marital-status        7
occupation           15
relationship          6
race                  5
sex                   2
capital-gain        123
capital-loss         99
hours-per-week       96
native-country       42
income                4
dtype: int64
```

Figure 9- Count of Unique Values

## Data Completeness and Validity

For data completeness, I checked only the presence of data by identifying **np.nan** values. However, this approach does not address the validity of the values. To ensure validity, I created a new dataframe summarizing the distinct values in each column, allowing for a visual inspection and validation.



```
# Check the number of missing values in each column
missing_values = df.isnull().sum()

# Display the number of missing values for each column
print("Missing Values in Each Column:\n")
print(missing_values)
```

```
Missing Values in Each Column:

age                 0
workclass         963
fnlwgt              0
education           0
education-num       0
marital-status      0
occupation        966
relationship        0
race                0
sex                 0
capital-gain        0
capital-loss        0
hours-per-week      0
native-country    274
income              0
dtype: int64
```

Figure 10-count of Null values

After quick validation of the dataset columns, I found out that the columns **workclass, occupation, and native-country** are having invalid values represented by ?

To continue with the project, the values represented by ? are casted with **np.nan** considering the values are missing.

## Data Distribution

To look at the data distribution, I started by generating the histogram for capital-gain and capital-loss

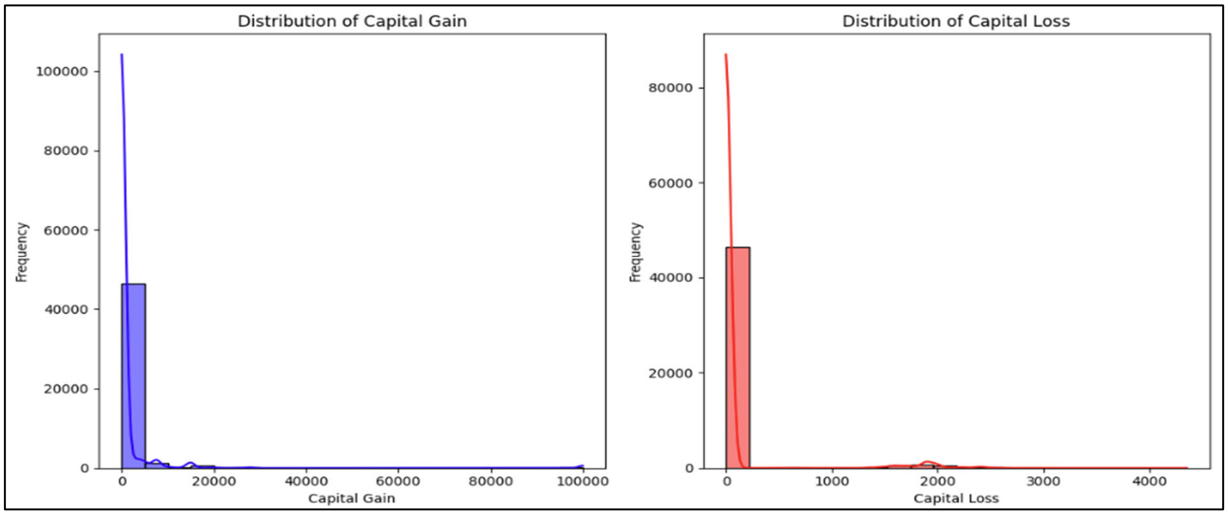
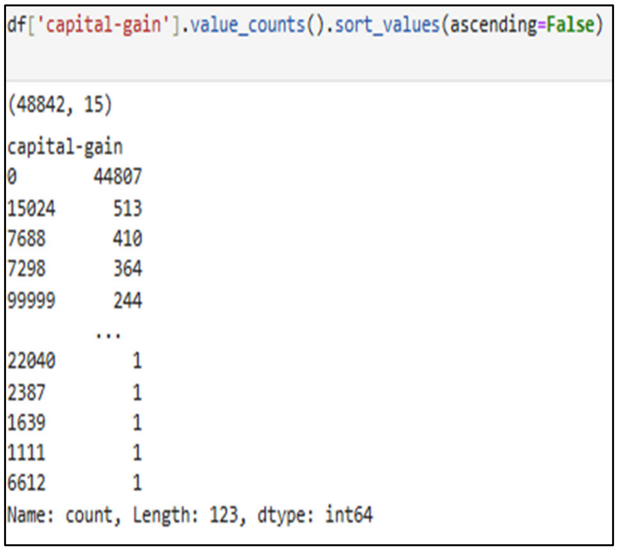


Figure 11-Distribution of capital-gain and capital-loss

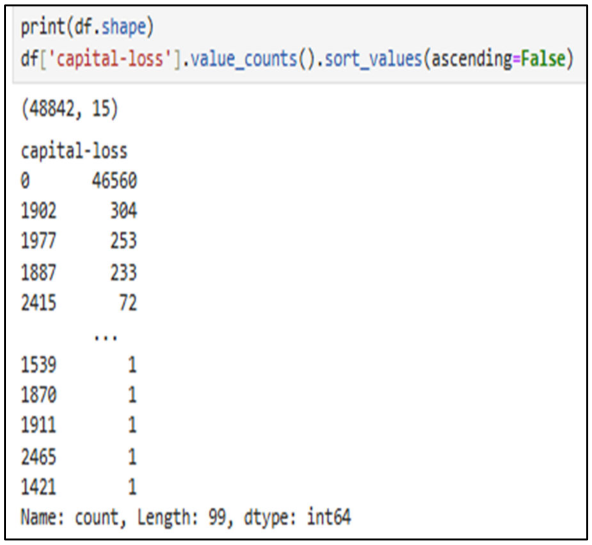
Capital-gain: the distribution of capital-gain is likely to be skewed with the majority of people reporting zero or very low capitals gains and a very small percentage having high capital-gain. This is confirmed by the value_counts which shows that 91.7% (44807 rows) are having 0 values and only 8.3% are having values distributed between low and high gain. Based on this observation.
I recommend transforming the capital-gain column into two categories: gain and no-gain
Capital-loss: The value_counts output for the capital_loss column shows that the majority of individuals have zero capital loss, with 46,560 rows showing a value of zero. A few other values, such as 1902, 1977, and 1887, appear more frequently but still have relatively low occurrences.



```
df['capital-gain'].value_counts().sort_values(ascending=False)

(48842, 15)
capital-gain
0        44807
15024      513
7688       410
7298       364
99999      244
         ...
22040        1
2387         1
1639         1
1111         1
6612         1
Name: count, Length: 123, dtype: int64
```

Figure 12-Value Count for capital-gain



```
print(df.shape)
df['capital-loss'].value_counts().sort_values(ascending=False)

(48842, 15)
capital-loss
0       46560
1902      304
1977      253
1887      233
2415       72
         ...
1539        1
1870        1
1911        1
2465        1
1421        1
Name: count, Length: 99, dtype: int64
```

Figure 13-Value Counts for capital-loss

Based on this observation,
I recommend transforming the capital-loss column into two categories: loss and no-loss.
In conclusion, both columns will be merged in one column called capital-gain-loss-classification as per below table:

| Gain | Loss | Capital-gain-loss-classfication |
|---|---|---|
| >0 | | Gain |
| | >0 | Loss |
| 0 | 0 | Break-even |

That conversion lead to the below distribution:

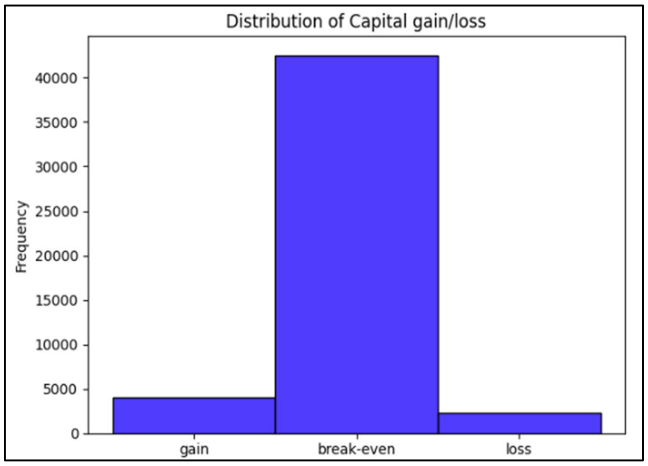
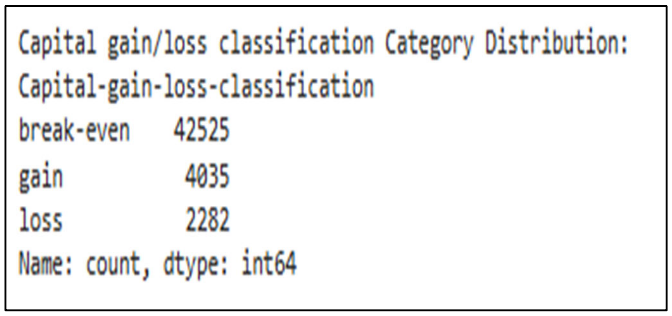


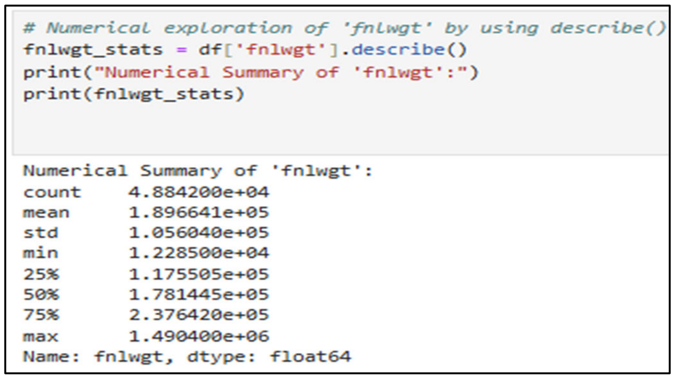
Figure 15-Distribution of capital-gain-loss



```
Capital gain/loss classification Category Distribution:
Capital-gain-loss-classification
break-even    42525
gain           4035
loss           2282
Name: count, dtype: int64
```

Figure 14-Numerical Distribution of capital-gain-loss-classification

Another distribution to look at is for the **fnlwgt** column

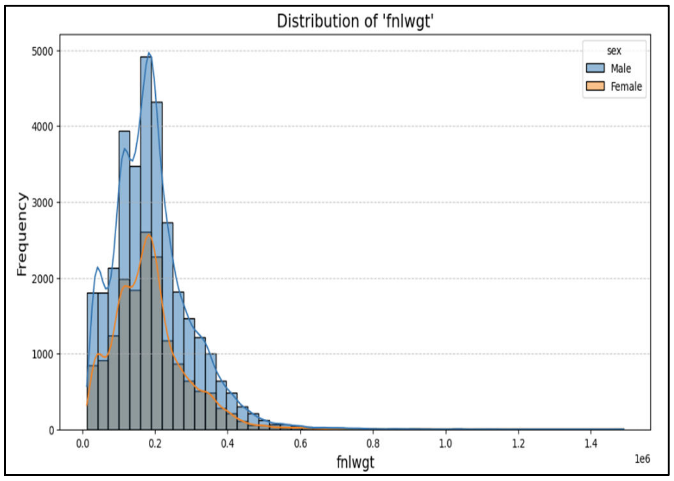


```
# Numerical exploration of 'fnlwgt' by using describe()
fnlwgt_stats = df['fnlwgt'].describe()
print("Numerical Summary of 'fnlwgt':")
print(fnlwgt_stats)
```

```
Numerical Summary of 'fnlwgt':
count    4.884200e+04
mean     1.896641e+05
std      1.056040e+05
min      1.228500e+04
25%      1.175505e+05
50%      1.781445e+05
75%      2.376420e+05
max      1.490400e+06
Name: fnlwgt, dtype: float64
```

by looking at the overall numerical distribution, we can notice that the mean value is around 189664 and the median or 50th percentile is around 178144 which indicates a slight skew in the distribution.
In addition, comparing the min=12285 and the max=1490400 indicates the existence of potential outliers.

Figure 16-Numerical Distribution for fnlwgt



When looking at the histogram, the distributions between men and woman are similar in shape, but there might be slight differences in density at certain ranges.
The **fnlwgt** variable has extreme values that do not represent realistic population weights, these should be excluded (set to NaN). Otherwise, they could disproportionately influence analyses. I will use the interquartile range (IQR) to set the outliers to missing (NaN).

Figure 17-Distribution of fnlwgt

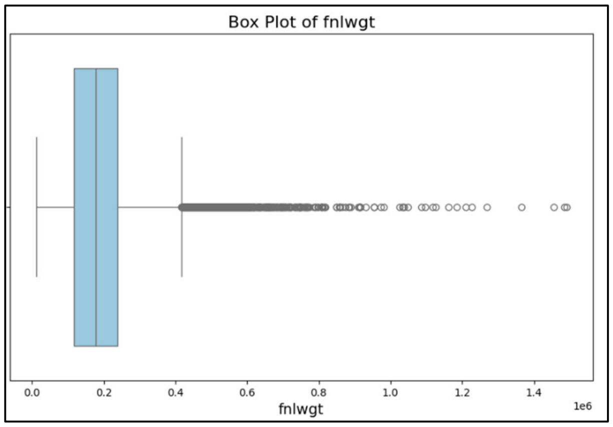


Figure 18-Boxplot-fnlwgt- before replacing outliers

This plot shows the outliers available under the columns fnlwgt. These outlies has been replace with NaN values.
In next plot the outliers has been replaced and its confirmed with df.isnull().sum() how the number of missing values has been increased.

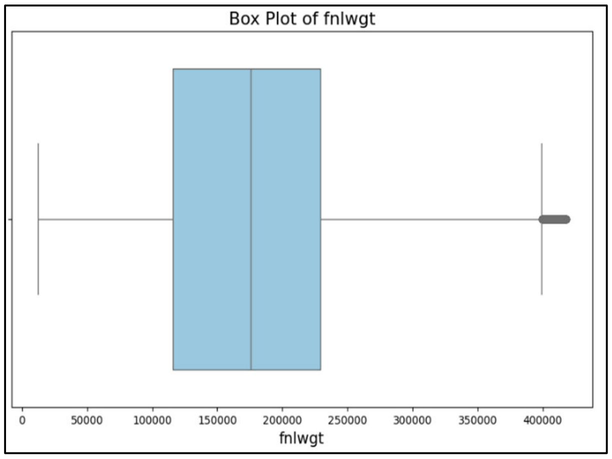
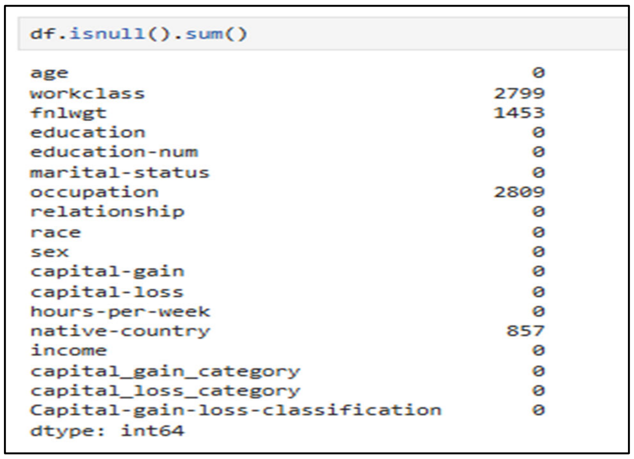


Figure 19-Boxplot-after replacing outliers



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

age                                    0
workclass                           2799
fnlwgt                              1453
education                              0
education-num                          0
marital-status                         0
occupation                          2809
relationship                           0
race                                   0
sex                                    0
capital-gain                           0
capital-loss                           0
hours-per-week                         0
native-country                       857
income                                 0
capital_gain_category                  0
capital_loss_category                  0
Capital-gain-loss-classification       0
dtype: int64
```

Figure 20-null values after replacing outliers

### Correlation

## Machine Learning Part

Since we are classifying income as either more than 50K or less, this is a classification problem. In this section, I have used several models, including Linear Regression with a threshold to convert it into a class, Logistic Regression, XGBoost Classifier, and Random Forest Classifier.

### Data Cleanning

The data cleaning consist of removing outliers and removing or imputing null values. In this exercise I have removed the outliers using the IQR method and removed the rows with null values. Another data cleaning is to remove the typos error available in the values of the income column

## Feature Engineering

At this stage, I have removed the variables (capital-gain, capital-loss ,capital_gain_category, capital_loss_category ) since all these column are now represented by the column capital-gain-loss-classification

Another step I took was encoding all the non-numerical variables. I experimented with two encoding methods: first, I used LabelEncoder(), and then I tried the get_dummies encoding algorithm..

I have tested models using both scaled and unscaled data to assess the impact of feature scaling on model performance. Scaling is important because many machine learning algorithms are very sensitive to the magnitude of input features. Withou scaling, features with larger values can dominate the learning process. I applied standard scaling to bring all features to a similar scale and observed any improvement in prediction accuracy. This comparison helped me to evaluate whether scaling enhances model performance or not.

## Splitting Data

I splitted the dataset into training and testing sets with test_size=0.2 that indicates the proportion of dataset that will be allocated to the test set. In this case, 20% will be allocated for testing and 80% will be allocated for training. Another parameter is random_state=42 that ensure the split is reproducible.

## Training & Prediction

I have done several training and prediction using different models.
For XGBoost and RandomForestClassifier, I have done some hyperparameters fine-tuning using gridseachcv and RandomizedSearchCV to find the best parameters. I summarized the accuracy of each combination of encoder used, model employed, data scaling and Accuracy

| # | Encoder | Model | Scaled/not Saled | Accuracy |
|---|---|---|---|---|
| 1 | Label Encoder | Logistic Regression | Not Scaled | 0.78 |
| 2 | Get_dummies | Logisitic Regression | Not Scaled | 0.83 |
| 3 | Label Encoder | Linear Regression(threshold 0.5) | Not Scaled | 0.8 |
| 4 | Get Dummies | Linear Regression (threshold 0.5) | Not Scaled | 0.83 |
| 5 | Get Dummies | RandomForestClassifier(n_esimator=131) | Not Scaled | 83.08 |
| 6 | Get Dummies | XGBoost(n_estimator=100) | Not Scaled | 83.92 |
| 7 | Get Dummies | XGBoost(n_estimator=31) | Not Scaled | 84.12 |
| 8 | Label Encoder | Linear Regresion | Scaled | 0.8 |
| 9 | Label Encoder | Logistic Regression | Scaled | 0.8 |
| 10 | Label Encoder | RandomForestClassifier(n_esimator=141) | Scaled | 82.94 |
| 11 | Label Encoder | XGBoost(n_estimator=41) | Scaled | 84.16 |
| 12 | Get Dummies | Linear Regresion | Scaled | 83 |
| 13 | Get Dummies | Logistic Regression | Scaled | 83.5 |
| 14 | Get Dummies | RandomForestClassifier(n_esimator=111) | Scaled | 82.93 |
| 15 | Get Dummies | XGBoost(n_estimator=31) | Scaled | 84.12 |
| 16 | Get Dummies | RandomForestClassifier() | Scaled | 83.68 |

|  |  |  |  |  |
|---|---|---|---|---|
|  |  | fine tuned with best param:<br>{'n_estimators': 600,<br>'min_samples_split': 2,<br>'min_samples_leaf': 2,<br>'max_features': 'sqrt',<br>'max_depth': 50,<br>'bootstrap': True} |  |  |
| 17 | GetDummies | XGBoost<br>Fine tuned with best param<br>Best Parameters: {'colsample_bytree': 1.0, 'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 200, 'subsample': 1.0} | Scaled | 84 |

## General Observations

Looking at  rows 1,2,9 and 13 from the table, we can conclude that accuracy with get dummies encoding outperforms label encoder especially with the Logistic Regression for both scaled(80% to 83.5%) and unscaled data (78% to 83%)

Looking at the rows 10, 14, and 16, we can conclude that the RandomForestClassifier shows varying accuracy based on parameter tuning, with the fine-tuned model reaching 83.68% with scaled data and 83.08% unscaled.

We can conclude that Get Dummies encoding is generally more effective than Label Encoder, particularly with XGBoost and Random Forest, both of which benefit more from this encoding. Scaling does not dramatically improve the models' performance but can offer slight improvements in specific cases, such as with XGBoost.