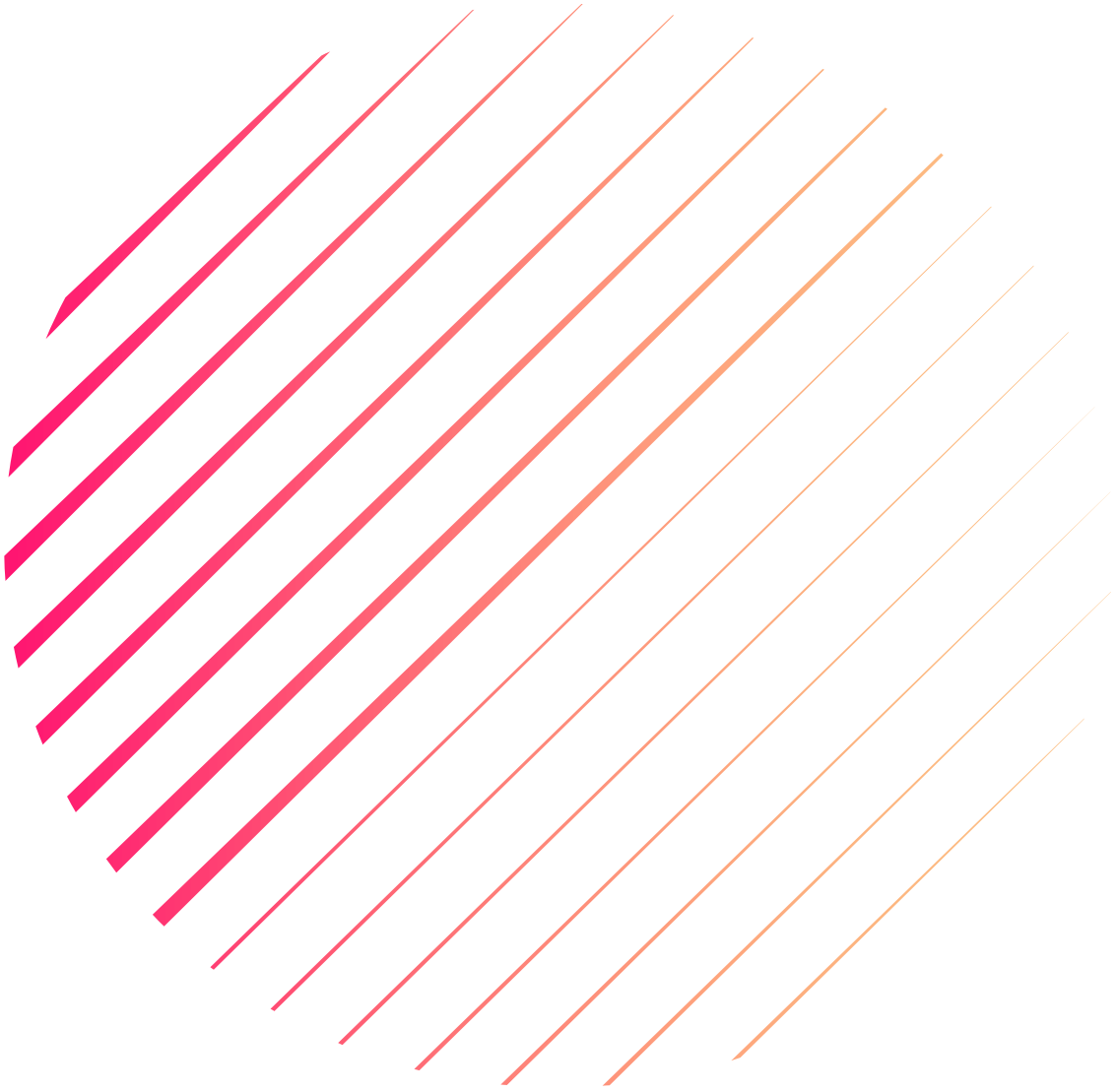
INCOME CLASSIFICATION



25th November 2023

Introduction to Data Science

Mukul Malik (*21ucs133*), Mukul Verma (*21ucs134*), Jeetaksh Gandhi(*21ucs098*), Soumyadeep (*21dcs010*)

Dr. Aloke Datta,
Dr. Subrat K. Dash,
Dr. Upendra Pratap Singh

DATASET: Census Income

We have chosen the "**Census Income**" dataset from the UC Irvine Machine Learning Repository.

We have to **predict** whether income exceeds $50K/yr based on census data. The subject area for this problem is Social Science. In terms of analytics, this is a **binary classification** problem.

Following is the dataset characterisitcs:

Dataset Characteristics	**Subject Area**	**Associated Tasks**
Multivariate	Social Science	Classification
Feature Type	**# Instances**	**# Features**
Integer	48842	14(**13 effective features**)

On the basis of the features, we want to classify if the person will have income less than $50k (target label $\mathbf{0}$) or greater than/equal to (target label $\mathbf{1}$).

Dataset Feature Description

There are **14 "*independent*" features** of each pattern. Following table describes these features!

Feature Name	Type	Decription	Value
Age	Integer	Represents the age of the individual.	Age (17-90)
Workclass	Categorical **(*)**	Represents the workclass	Private(0), Self-emp-not-inc(1), Self-emp-inc(2), Federal-gov(3), Local-gov(4), State-gov(5), Without-pay(6), Never-worked(7)
Fnlwgt	Integer	To adjust sampling.	
Education	Categorical **(*)** NOTE: *This was removed during dimensionality reduction.*	Represents the highest level of education achieved by the individual.	
Education-num	Integer	Represents the Education field categorical values mapped into integers.	Preschool(0), 1st-4th(1), 5th-6th(2), 7th-8th(3), 9th(4), 10th(5), 11th(6), 12th(7), HS-grad(8), Some-college(9), Ass-voc(10), Assoc-acdm(11), Bachelors(12), Masters(13), Prof-school(14), Doctorate(15)
Marital-status	Categorical **(*)**	Describes the marital status of the individual.	Married-civ-spouse(0), Divorced(1), Never-married(2), Separated(3),

			Widowed(4), Married-spouse-absent(5), Married-AF-spouse(6)
Occupation	Categorical **(*)**	Represents the type of occupation the individual is engaged in.	Tech-support(0), Craft-repair(1), Other-service(2), Sales(3), Exec-managerial(4), Prof-specialty(5), Handlers-cleaners(6), Machine-op-inspct(7), Adm-clerical(8), Farming-fishing(9), Transport-moving(10), Priv-house-serv(11), Protective-serv(12), Armed-Forces(13)
Relationship	Categorical **(*)**	Describes the relationship of the individual in the family.	Wife(0), Own-child(1), Husband(2), Not-in-family(3), Other-relative(4), Unmarried(5)
Race	Categorical **(*)**	Represents the race of the individual.	White(0), Asian-Pac-Islander(1), Amer-Indian-Eskimo(2), Other(3), Black(4)
Sex	Integer (Binary)	Indicates the gender of the individual.	Female(0), Male(1)
Capital-gain	Integer	Represents capital gains for the individual.	0 – 99999
Capital-loss	Integer	Represents capital losses for the individual.	0 - 4356
Hours-per-week	Integer	Represents the number of hours the individual works per week.	1 – 99
Native-Country	Categorical **(*)**	Represents the country of	United-States(0), Cambodia(1), England(2),

		origin of the individual.	Puerto-Rico(3), Canada(4), Germany(5), Outlying-US(Guam-USVI-etc)(6), India(7), Japan(8), Greece(9), South(10), China(11), Cuba(12), Iran(13), Honduras(14), Philippines(15), Italy(16), Poland(17), Jamaica(18), Vietnam(19), Mexico(20), Portugal(21), Ireland(22), France(23), Dominican-Republic(24), Laos(25), Ecuador(26), Taiwan(27), Haiti(28), Columbia(29), Hungary(30), Guatemala(31), Nicaragua(32), Scotland(33), Thailand(34), Yugoslavia(35), El-Salvador(36), Trinadad&Tobago(37), Peru(38), Hong(39), Holand-Netherlands(40)

(*) These values were originally in categorical format as mentioned in the values column. In the final dataset, we mapped all categorical data into integers. The (?) in the value column denotes the integer that the categorical value was mapped to during pre-processing. The program with name "**mapper.cpp**" was used to do this mapping.

The 14th column is **income0** which has binary value 0 (if income <= 50K) and 1 (if income > 50K). This is the **target value.**

DATA PRE-PROCESSING

The dataset initially consisted of **48,842** instances with each instance consisting pf 14 features out of which many were categorical as explained in the table above.

Our first goal was to perform *Dimensionality Reduction*. Here, we tried to identify redundant values. The education feature was duplicate. Column #4 (Education) had categorical values and Column #5 (Education-num) had those values just already mapped into integers. So, we removed the Education (#4) column. This reduced the feature space from 14 to 13.

Next goal was to map all remaining *categorical data* into *numerical data* to be able to apply a diveristy of models to it. The mapped integer values are mentioned in the table above. We wrote a C++ program to do the same called **mapper.cpp,** the details of which are provided in the program itself.

The final step was to drop incomplete patterns (in which one of the features was missing or had NULL value [?]). This is done using `np.dropna()` function in the Jupyter notebook shared.

After this, our instances got reduced to **45,222.** We consider this to be a low-medium sized dataset.

Instance Modification during Pre-Processing

48,842 → 45,222

Feature Modification during Pre-Processing

14 Features → 13 Features

Exploratory Data Analysis

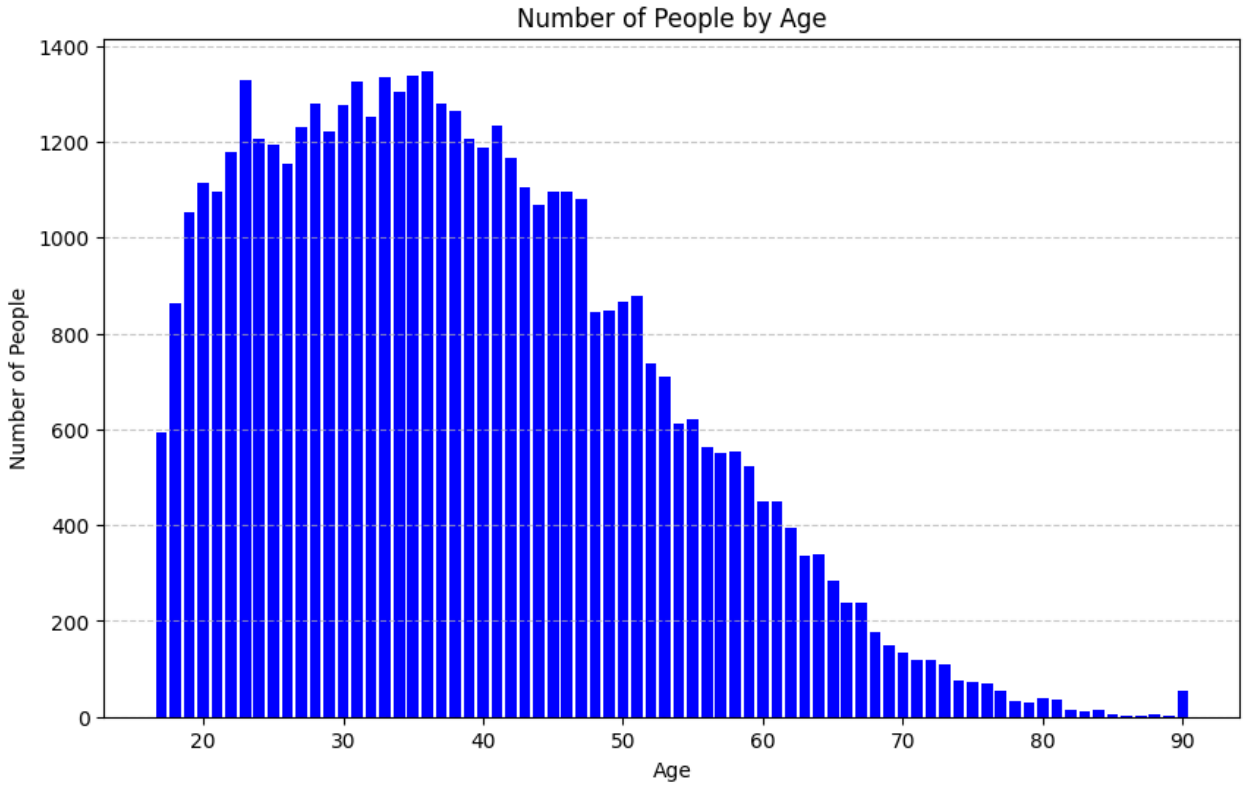
In this section, we try to uncover the features statistics using **descriptive statistics.**

First off, we calculated **minimum value, maximum value, mean, standard deviation** for each feature.

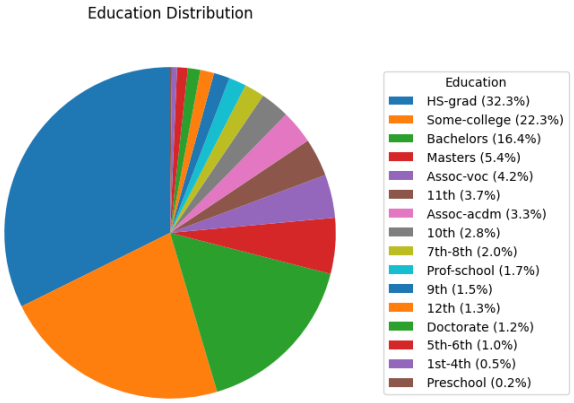
	age	WORK CLASS	fnlwgt	educational_num	MARITAL_STATUS	occupation	relationship	race	gender	capital_gain	capital_loss	hours_per_week	native_country
count	45222	45222	4.52E+04	45222	45222	45222	45222	45222	45222	45222	45222	45222	45222
mean	38.55	0.7422	1.90E+05	10.1185	1.056698	4.735483	2.39706	0.445	0.675	1101.4303	88.59542	40.93802	1.487174
std	13.22	1.4656	1.06E+05	2.55288	1.173525	2.981488	1.22672	1.198	0.4684	7506.4301	404.9561	12.00751	5.588405
min	17	0	1.35E+04	1	0	0	0	0	0	0	0	1	0
25%	28	0	1.17E+05	9	0	2	2	0	0	0	0	40	0
50%	37	0	1.78E+05	10	1	4	2	0	1	0	0	40	0
75%	47	1	2.38E+05	13	2	7	3	0	1	0	0	45	0
max	90	6	1.49E+06	16	6	13	5	4	1	99999	4356	99	40

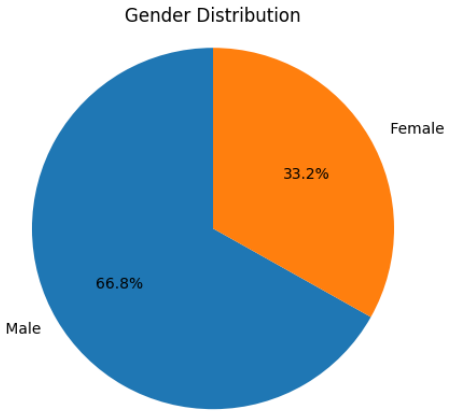
Some key stats to observe here are:

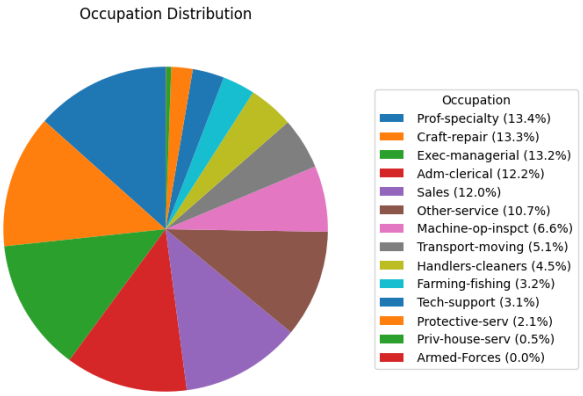
1. The mean age of the population is **35.88** with a standard deviation of **13.22**. The graph for the same is: (**Left skewed distribution**)

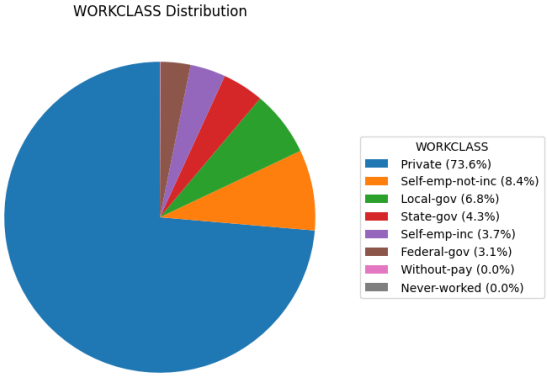


2. **71%** of the people either have some college experience or HS-grad or have a Bachelors degree.

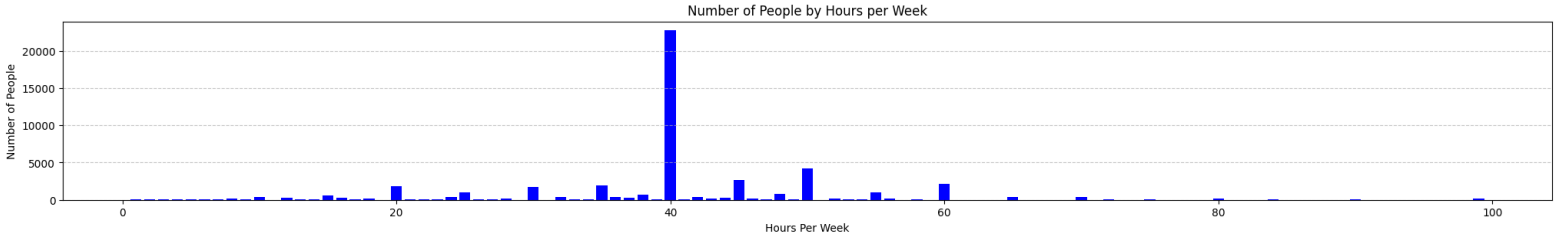


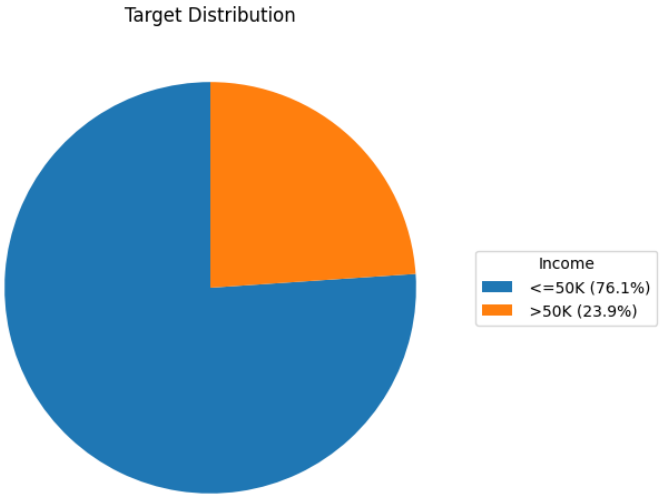






3. **73.6%** of the people are working in Private sector. Also, around 50% of the people work 40 hours a week.



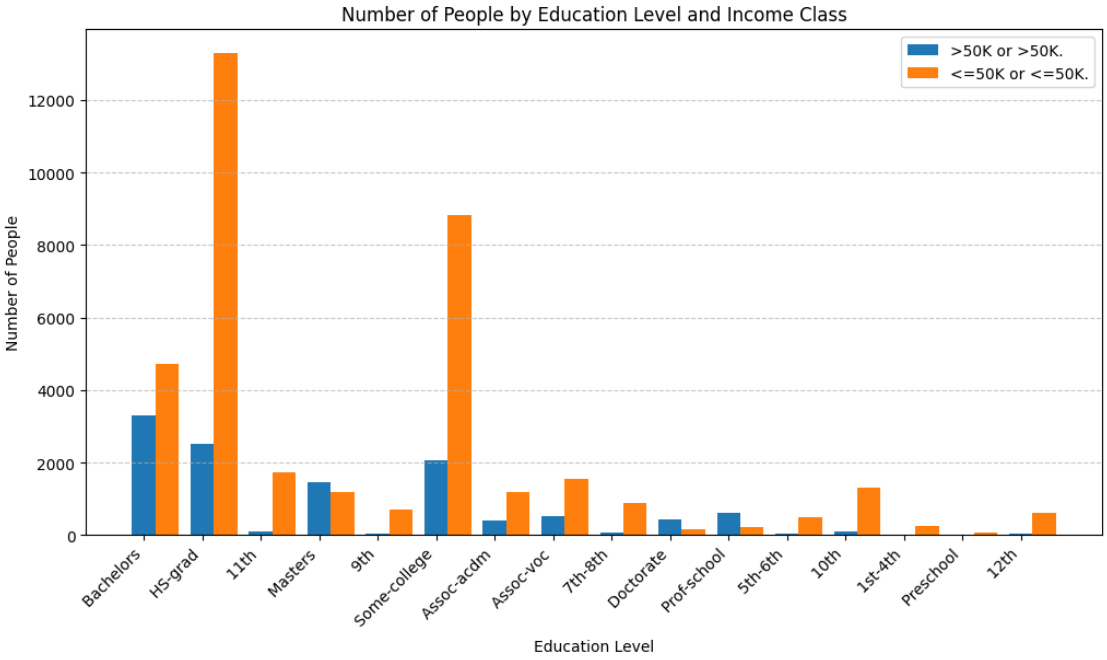


76.1% labels belong to class 0 (<=50K)
23.9% labels belong to class 1 (>50K)

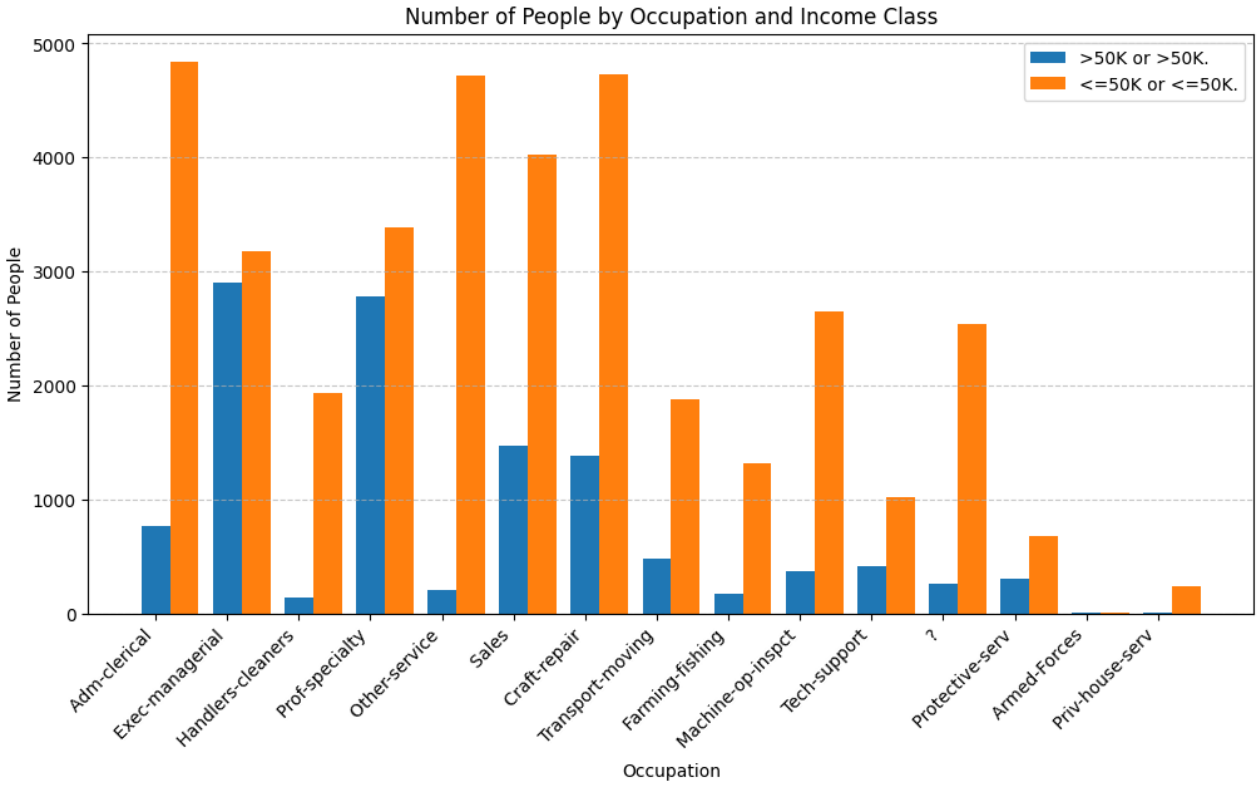
Since there are 13 “Independent” features, we will look at how they influence the classification individually.

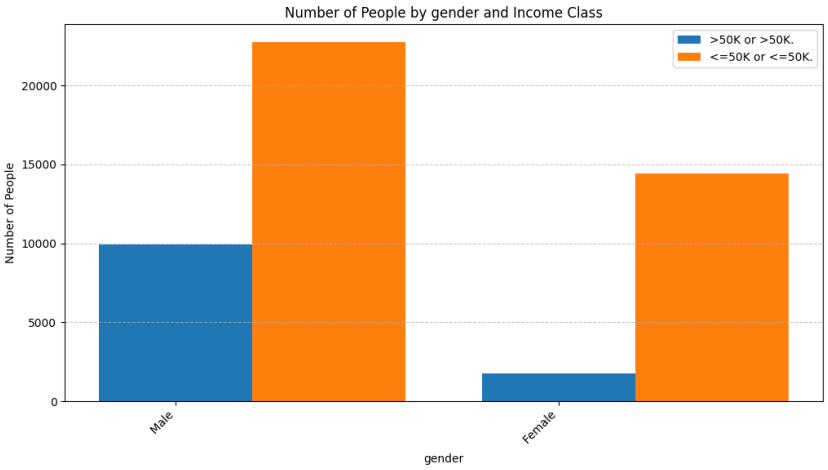
4. Most of the people who **earn** more than 50K $ are in the age range **33-53**. And those who make less than 50K are comparatively **younger.** If the person is younger than 25 years, it’s highly likely that he/she will earn less than 50K. This suggests effect of professional experience on income.

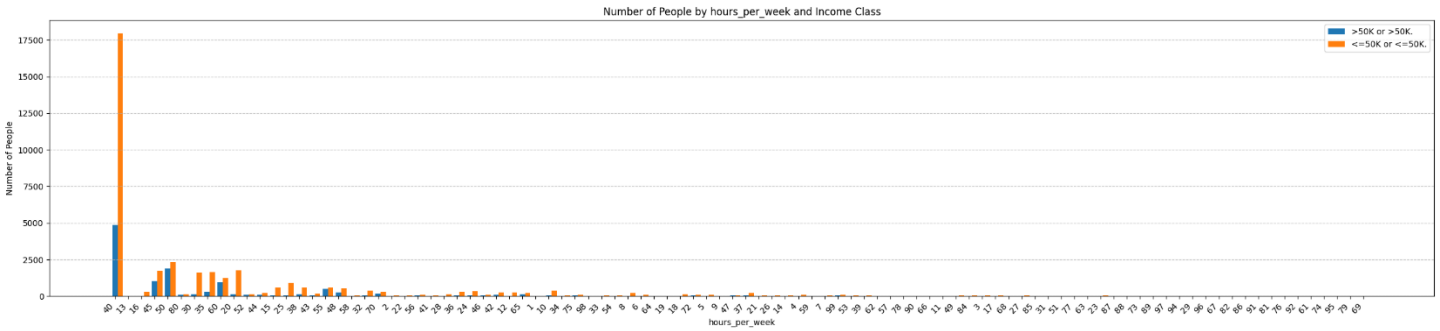
5. Having a bachelor’s/masters degree or some college experience or HS-Grad increases your chance of earning more than 50K. **If you have a masters degree, then you are <u>more likely to earn more than 50K instead of earning less than 50K!</u>**
   If you hold a Doctorate or have been a professor, then you’re HIGHLY likely to earn more than 50K.

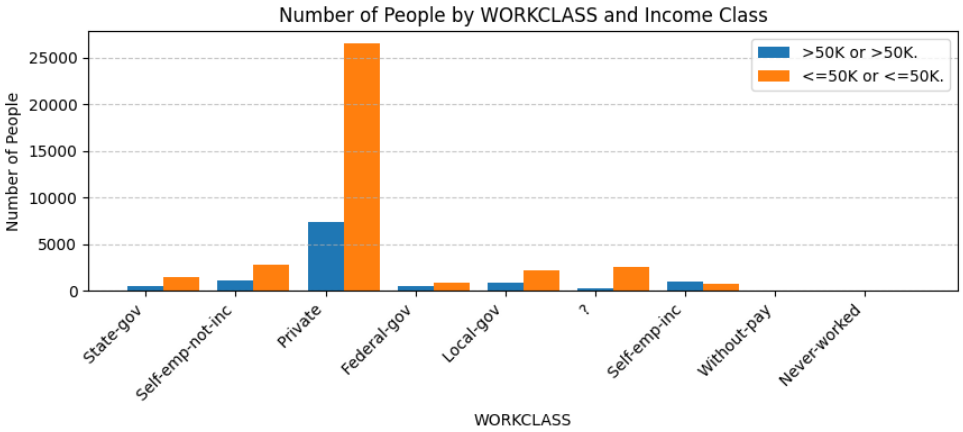


6. Being a professor, or an executive of management **highly** increases the probability of earning more than 50K.

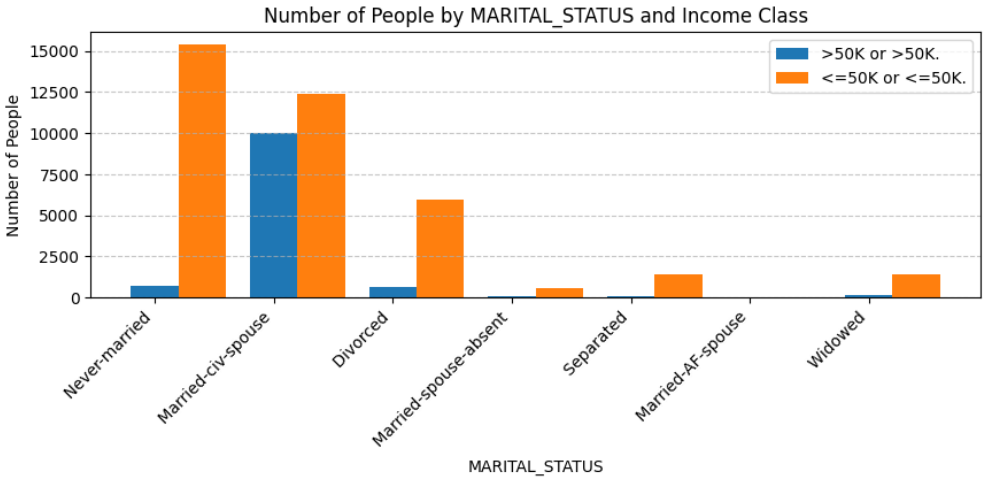








7. Being self-employed, or having your own business increases your chances of earing more than 50K!



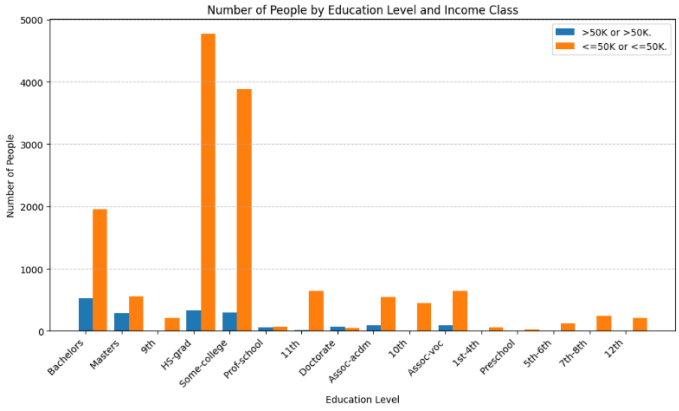
8. Being married indicates higher probability of earning more than 50K. This *could be due to more financial stability*. Also if you're married then your age will likely be 25+ and according to our age graph above, the probability of earning more than 50K increases in that age range.

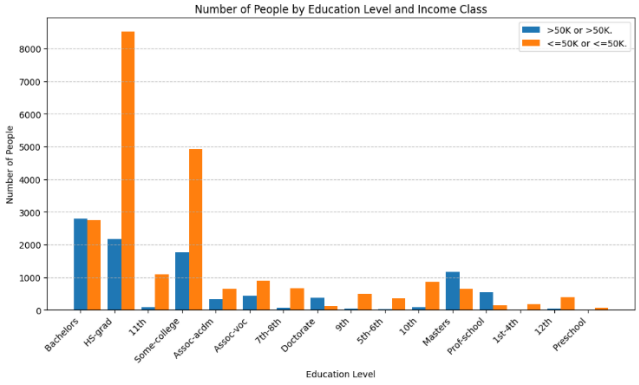
Gender/Society Bias

If you look at the Correlation Table on Page 14, then 'gender', 'education', and some other features are highly correlating to income. Let's now examine the affect of GENDER combined with some other features to see if there is bias towards Gender.

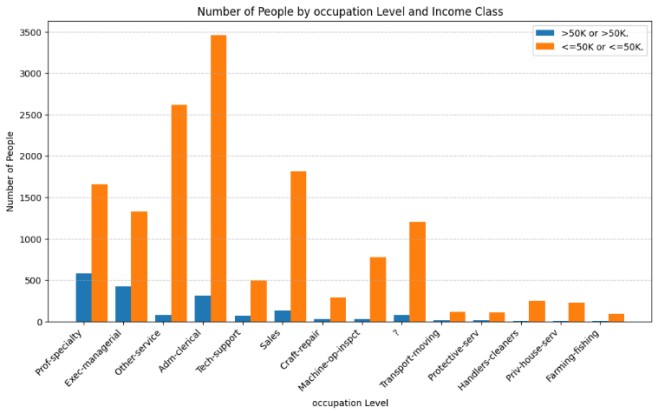
*The graphs on the **LEFT** only consider females and graphs on the **RIGHT** only consider male candidates.*

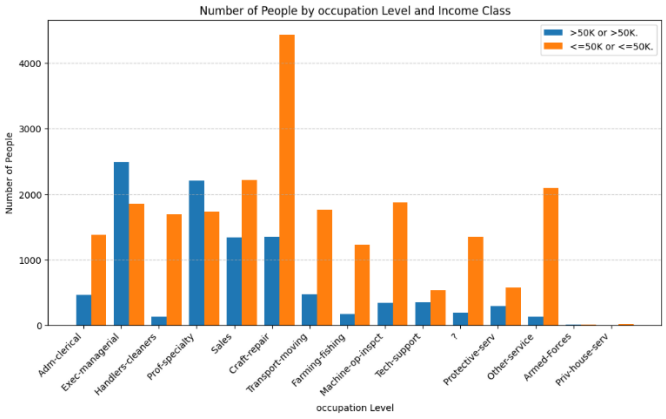
1. Considering same education levels, being a male increases probability of earning more than 50K.





2. Considering same occupation, being a male ***highly*** increases probability of earning more than 50K. Consider the case of "Executive of Management", the prior probability of a female for earning more than 50K is **29%** while for male is **57%** (calculated using Bar Lengths).



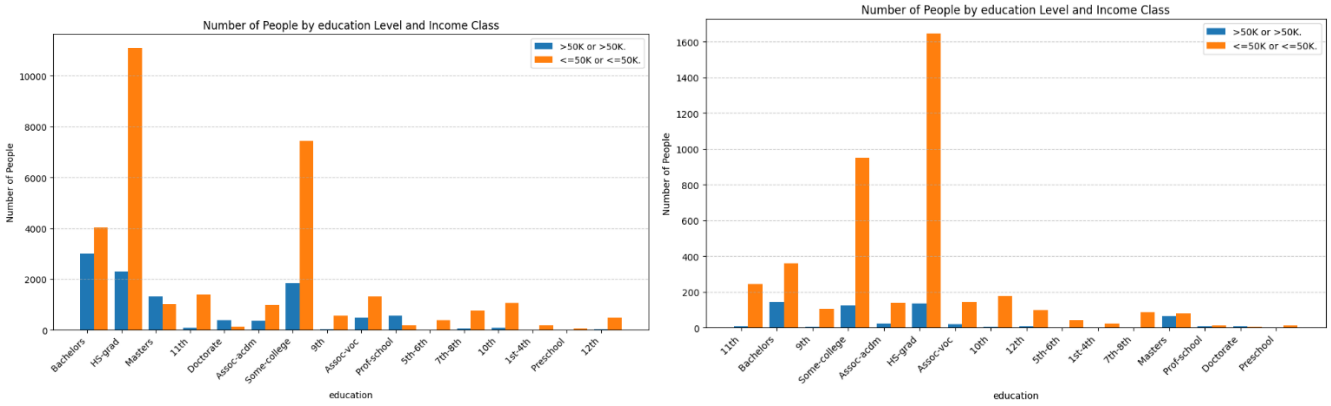


This biasness may be due to the societal biases present in the world at the time of collection of the dataset. It was collected in 1990s and thus, reflects the old statistics in a time when *gender biasness was highly prevalent.*

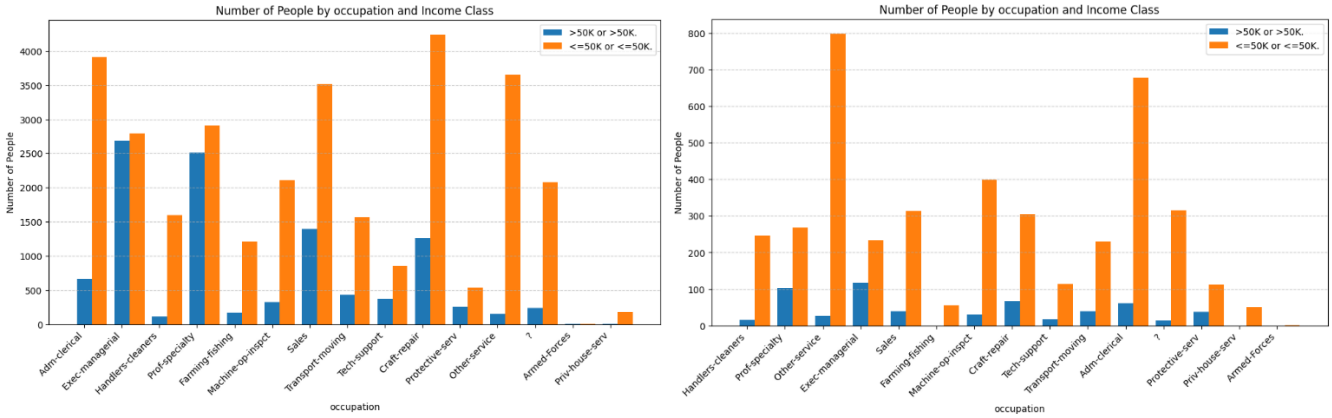
Let's now examine the affect of RACE combined with some other features to see if there is bias towards Race.

*The graphs on the **LEFT** only consider Whites and graphs on the **RIGHT** only consider Black candidates.*

1. Considering same education levels, race doesn't affect income much.



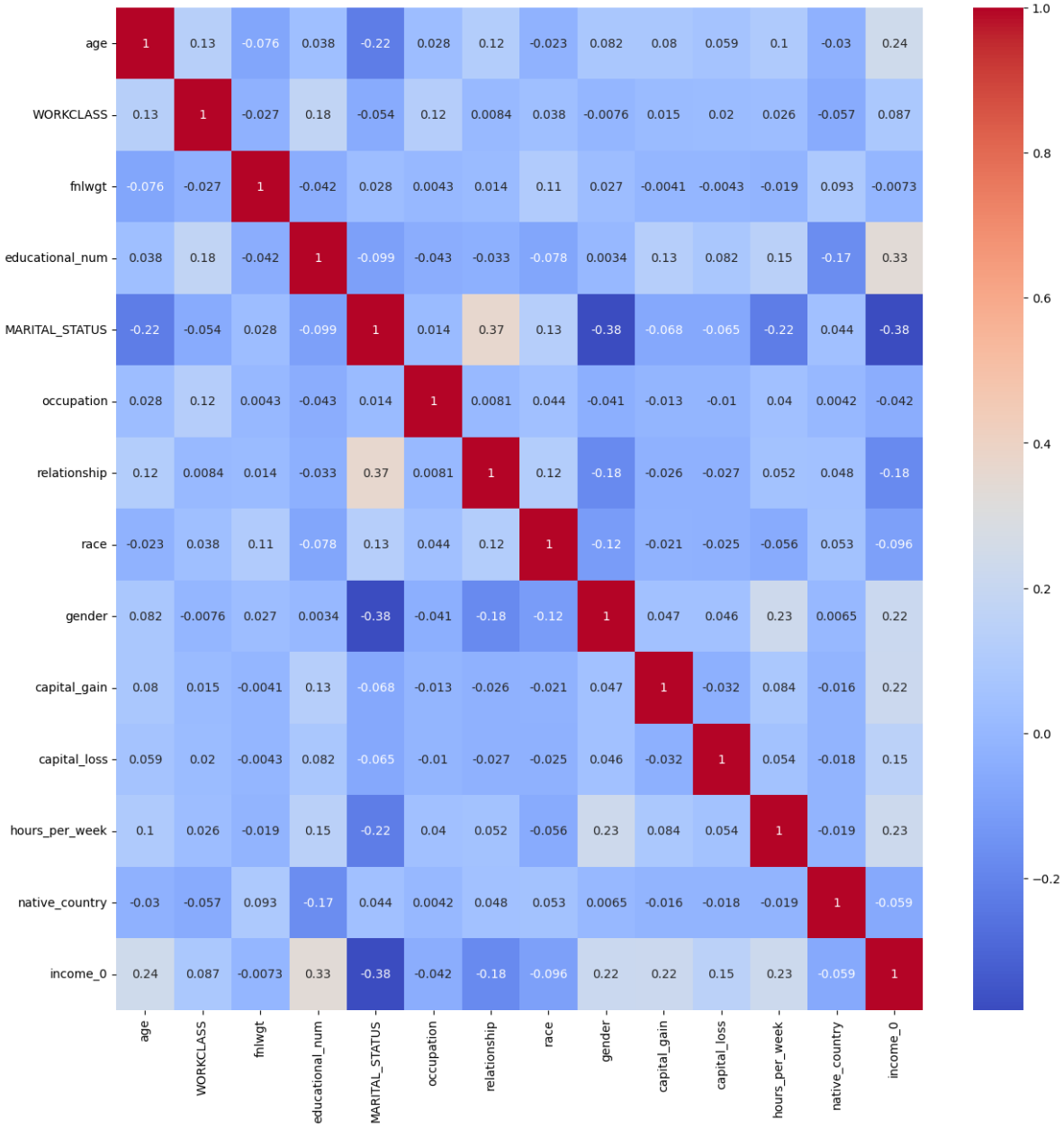
2. Considering same occupation levels, race ***highly*** affects the income.



This biasness may be due to the societal biases present in the world at the time of collection of the dataset. It was collected in 1990s and thus, reflects the old statistics in a time when *race biasness was highly prevalent.*

Correlation Matrix

	income	
age	0.237040	(High impact)
WORKCLASS	0.086802	
fnlwgt	-0.007264	
education	0.332800	(High impact)
MARITAL_STATUS	-0.377580	(High impact)
occupation	-0.042349	
relationship	-0.181754	
race	-0.096050	
gender	0.215760	(High impact)
capital_gain	0.221034	(High impact)
capital_loss	0.148687	
hours_per_week	0.227199	(High impact)
native_country	-0.058785	



Some General Inferences

from POV of above statistics

1. Being married increases likelihood of earning more than 50K.
2. Having a *capital_gain* more than 0 increases the likelihood of earning more than 50K. (From correlation matrix, we observe a high positive correlation).
3. Occupation of "*Exec-managerial*", "*Prof-speciality*" highly increases probability of earning more than 50K. Occupations of "*Other-services*", "*Adm-clerical*" highly decreases probability of earning more than 50K.
4. Having work hours per week less than 40 influences probability of earning more than 50K to decrease.
5. Education levels of "*masters*", "*bachelors*", "*college*" increase likelihood of earning more than 50K.

Let's use some data from our dataset to see our inferences are useful.

age	WORKCLASS	fnlwgt	education	MARITAL_STATUS	occupation	relationship	race	gender	capital_gain	capital_loss	hours_per_week	native_country	income_0
37	Private	284582	Masters	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	United-States	<=50K
49	Private	160187	9th	Married-spouse-absent	Other-service	Not-in-family	Black	Female	0	0	16	Jamaica	<=50K
52	Self-emp-not-inc	209642	HS-grad	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	45	United-States	>50K
31	Private	45781	Masters	Never-married	Prof-specialty	Not-in-family	White	Female	14084	0	50	United-States	>50K
42	Private	159449	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	5178	0	40	United-States	>50K
37	Private	280464	Some-college	Married-civ-spouse	Exec-managerial	Husband	Black	Male	0	0	80	United-States	>50K
30	State-gov	141297	Bachelors	Married-civ-spouse	Prof-specialty	Husband	Asian-Pac-Islander	Male	0	0	40	India	>50K
23	Private	122272	Bachelors	Never-married	Adm-clerical	Own-child	White	Female	0	0	30	United-States	<=50K
32	Private	205019	Assoc-acdm	Never-married	Sales	Not-in-family	Black	Male	0	0	50	United-States	<=50K

See the 2nd data point, it has an education level of 9th standard and occupation of "Other-Service", thus it's income is likely to be less than 50K which is true!

See the 8th and 9th data points, they are not married and their Occupations and Education levels (9th) correspond to low earning categories. Thus, they are likely to earn below 50K which is true!

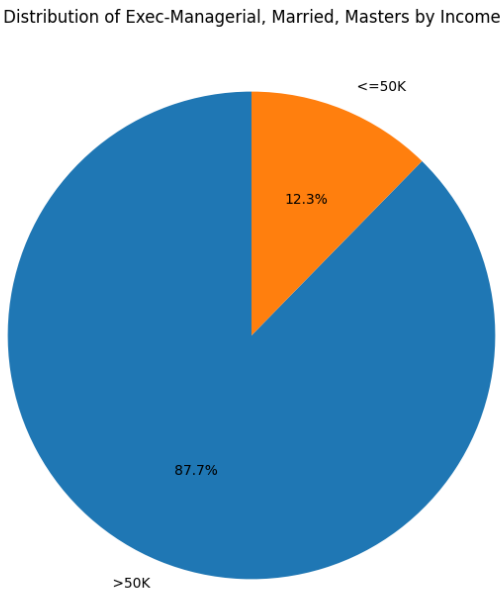
See the 4th data point, the person is not married **but** is a professor, has a Masters degree and a high capital gain, she (gender isn't as highly correlated with income) must earn more than 50K! That is true again!!

Consider the 6th data point, the person is married and black. He has some college experience but is an exec-managerial. And works 80 hours a week. Thus, he earns more than 50K.

Now **consider the first data point,** the person has an age of 37 (ideal for earning more than 50K), is married (highly correlated to higher income), has a master's degree (highly ideal for 50K above) and on top of that is an Exec-managerial. This increases her probability of earning more than 50K HIGHLY. **BUT she earns less than 50K. HOW DID THIS HAPPEN? *No classification can be 100% correct.*** *This data point could be an outlier.*

THE REASON FROM THE PROBABILITY THEORY

Considering the probability theory, we know that from the graphs above in Exploratory Data Analysis, that if a person has a Masters degree, is married and is an Exec-managerial, the prior probability of earning more than 50K is very high. But now look at this statistics from the entire dataset. It shows income distribution among people who match **all these 3 characteristics**:



Even still, there is some chance (12.3%), that the person may earn below 50K. Surely, the prior probability of this happening is very low but **it is not ZERO.** This may lead to accuracy loss which is the reason no classifcation algorithm may provide 100% accuracy on all datasets.

FUN FACT!

Our human classifier, based on our inferences that we derived, was able to correctly classify 8 of the 9 data points above (88.8% accuracy)! 😊

CODES:
All the codes written uptil here are contained within (with proper comments and labelling):

mapper.cpp → Code for pre-processing step 2
graph.ipynb → Codes for graphical sketching

Python Libraries used:

1. ***Numpy*** is used to make numerical computations and create numpy arrays,
2. ***Pandas*** is used to process data from CSV format dataset and create dataframes,
3. ***Matplotlib*** to plot the data points onto a graph,
4. ***Seaborn*** to statistically plot the data points onto the graph.

The Task of CLASSIFICATION

Now, we officially begin onto the task of classification using **machine learning algorithms**.

We will discuss why we need such an algorithm; how different algorithms differ in terms of usage, comparison of different algorithm's accuracy on this dataset, followed by conclusion.

Why **Classify using Machine Learning?**

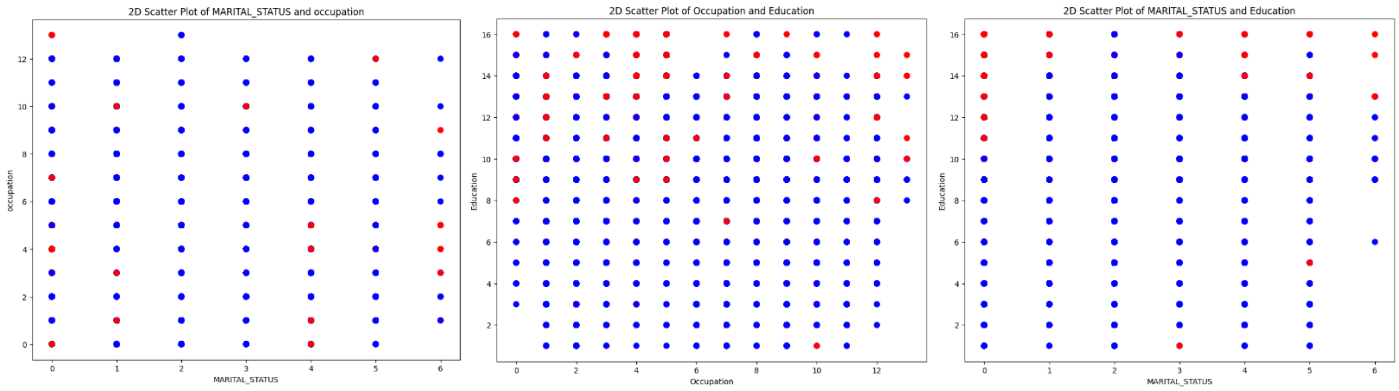
The dataset we're considering here is small as compared to the real world. Usually, the data is gigantic with huge feature space. It's not humanly possible to draw out good inferences. Even with our dataset, we can't fully observe (**on our own**) all its characteristics as it is pretty large.

As you saw in above inferences, we were forming some sort of decision boundaries for different features in our mind and plainly **guessing** the income. This is not very efficient and mathematically sound. We, on our own, can't learn all characteristics of the data. Machine Learning algorithms are however able to do this much more systematically, and with much accountability (why such a decision was taken? Machine learning algorithms will provide you insights into why this decision was taken mathematically).

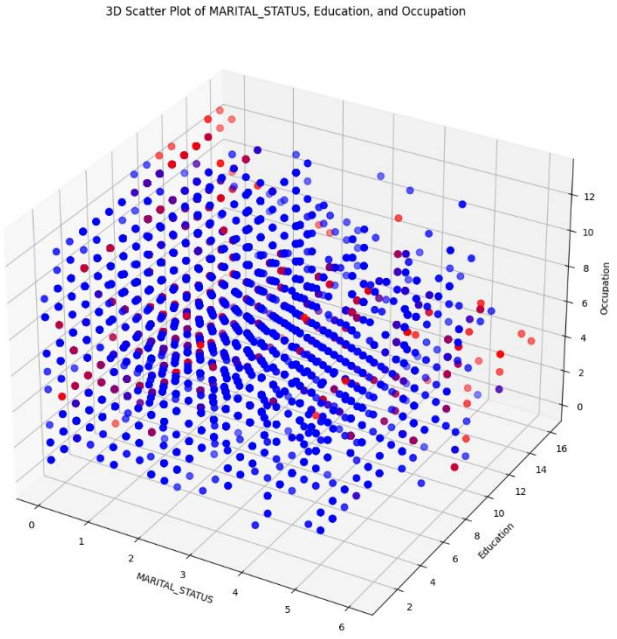
These algorithms may use different concepts like probability, optimization etc. to achieve this. We basically want to estimate functions/decision boundaries that are best able to discriminate between different labels with minimal error. The more data you feed into it, the more inferences the model can make and optimize itself. But then no model may classify all data with 100% accuracy as explained with the **probability concept** above. But different approaches may be able to better fit the different types of data. It is essential to know how the dataset is, its basic characteristics like **is it linearly separable? Etc.** before choosing an algorithm. An algorithm that works great on one type of data may not work as good on another. On this point, we'd like to mention a famous theorem! The **'*No Free Lunch Theorem*'** (NFLT) states that two optimization algorithms are equivalent when their performances are averaged over all possible learning problems.

We are considering **Neural Network, Random Forrest Classifier, Logistic Regression, Support Vector Machines, kNN Classifier** algorithms as choices for this dataset as they are a part of this course.

Let's take a look at an interesting plot of our dataset using three most correlevant features!



Combining them, we obtain the following **3D scatter plot:**



Blue label → Less than 50K
Red label → More than/Equal to 50K

*[**To check what categories these numerical values map to, check the feature description table given on page 4**]*

Of course, there are other features but these 3 seem to be the most powerful ones. A good starting cue at choosing a machine learning algorithm to classify this, could be using this plot as a standard. **Which one can best separate the blue dots from red dots here?**

***Outlier Analysis*:** Judging by the plots, there definitely are outliers. But their number isn't very large and low-medium amount.

***Decision Boundary Analysis*:** We want to give higher importance to some features than others as evident from correlation matrix. And from the looks of it, an entirely linear plane possibly can't classify with good accuracy. We require some degree of non-linearity in the decision function/algorithm.

Now, let's take a look at our classifiers!

1. Support Vector Machines (SVMs):

SVMs are effective in high-dimensional spaces and are well-suited for tasks where the data is not linearly separable. SVMs form a decision function/hyperplane to separate classes. Non-linear kernels can add non-linearity to the decision boundary.

Strengths: Good for binary classification, works well in high-dimensional spaces.
Considerations: SVMs can be sensitive to noise. We need to specify how much regularization to do.

2. Random Forrest Classifier:

Random Forests are versatile. They are effective in dealing with high-dimensional data and are resistant to overfitting.

Strengths: Robust to overfitting, handles non-linearities well, provides **feature importance**.
Considerations: May not perform as well on very small datasets, can be computationally expensive for large datasets.

3. Neural Network:

Neural networks, especially deep learning models, are powerful for complex tasks like image recognition, natural language processing, and tasks with **large amounts of data**. They use **convex optimization**.

Strengths: Excellent at capturing complex patterns and relationships in data, can automatically learn hierarchical representations.
Considerations: **Requires large amounts of data for training**, **computationally intensive**, and may need significant computational resources.

4. Logistic Regression:

Logistic Regression is a simple and fast algorithm for binary classification tasks. It's useful when you need **probabilities** (sigmoid) for class assignments.

Strengths: Simple, interpretable, efficient for small datasets, provides probabilities.
Considerations: Limited to linear decision boundaries, may not perform well on highly non-linear data.

5. kNN Classifier:

kNN is useful when the decision boundary is not well-defined or when the relationships in the data are complex.

Strengths: Simple to understand, works well with small datasets, handles non-linearities.
Considerations: Can be computationally expensive for large datasets, sensitive to irrelevant or redundant features, may require careful preprocessing.

MODEL CHOICE

Given our requirements, *Random Forrest Classifier* seems to be a fine choice as we want to give higher importance to some features and RFCs can efficiently do that with versatility. Additionally, they can effectively handle the amount of outliers in our dataset (they are robust to outlier presence).

Surely, neural networks may outperform the RFC if trained for a large amount of time (~5-10 minutes/150 epochs) as they are extremely good at learning complex features. But for that same level of accuracy, the **Random Forrest Classifier** could achieve it within a few seconds (~30 seconds). Given the size of our dataset, a deep learning approach should not be used as it will be a waste of resources.

Other models could also be used like SVMs or Logistic Regression.

For further analysis, once we're done using our main RANDOM FORREST CLASSIFIER model, we will use all 5 models and compare their accuracies for curiousity and judging if our choice made sense as we thought.

Training and Testing Split

To train the Random Forrest classifier, or say any other models from our arsenal, we need to split our dataset into a training set and testing set. The patterns in the training set will actually contribute to feature learning whereas the testing set will test the model's accuracy as a generic test case (one that the model has not seen previously to judge how **generically** it is fitting the problem domain). In maching learning, there is always a trade-off of bias and variance (or say, underfitting and overfitting). Our aim will be to achieve a good **test set accuracy.**

Additionally, we're using a test split of 15% (7,000 data instances) and a training split of 85% (38,000 data instances). Having a greater training split allows for better learning. Of our pre-processed dataset, we're using first 38,000 rows for training and remaining 7,000 for testing.

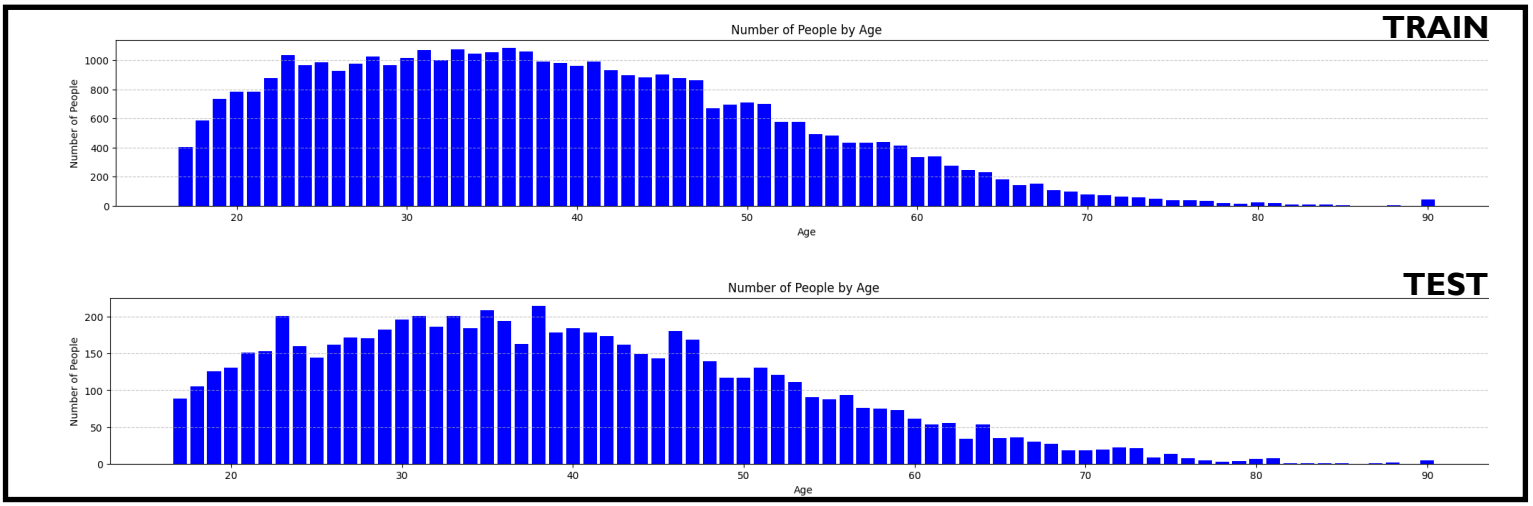
STRATIFIED SAMPLING

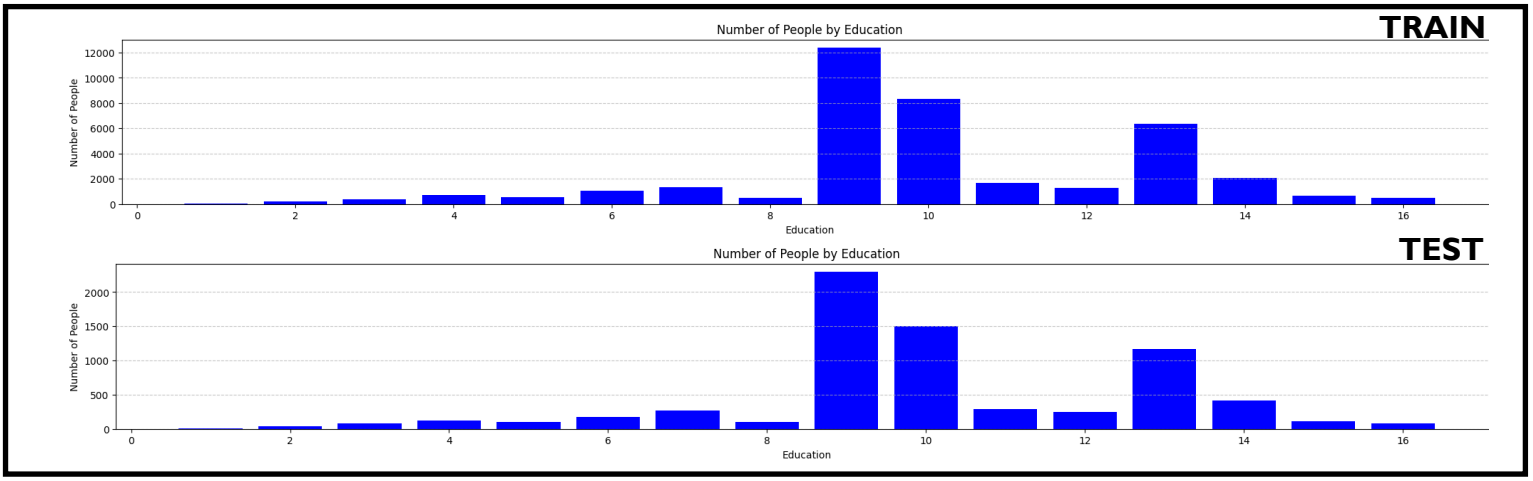
To build a robust and efficient split, its best to make sure the distribution of the original dataset remains intact in both the splits i.e., all features and classes are represented as heavily as they are in the original dataset, in the splits. The original balance should be maintained. This called "Stratified Sampling".

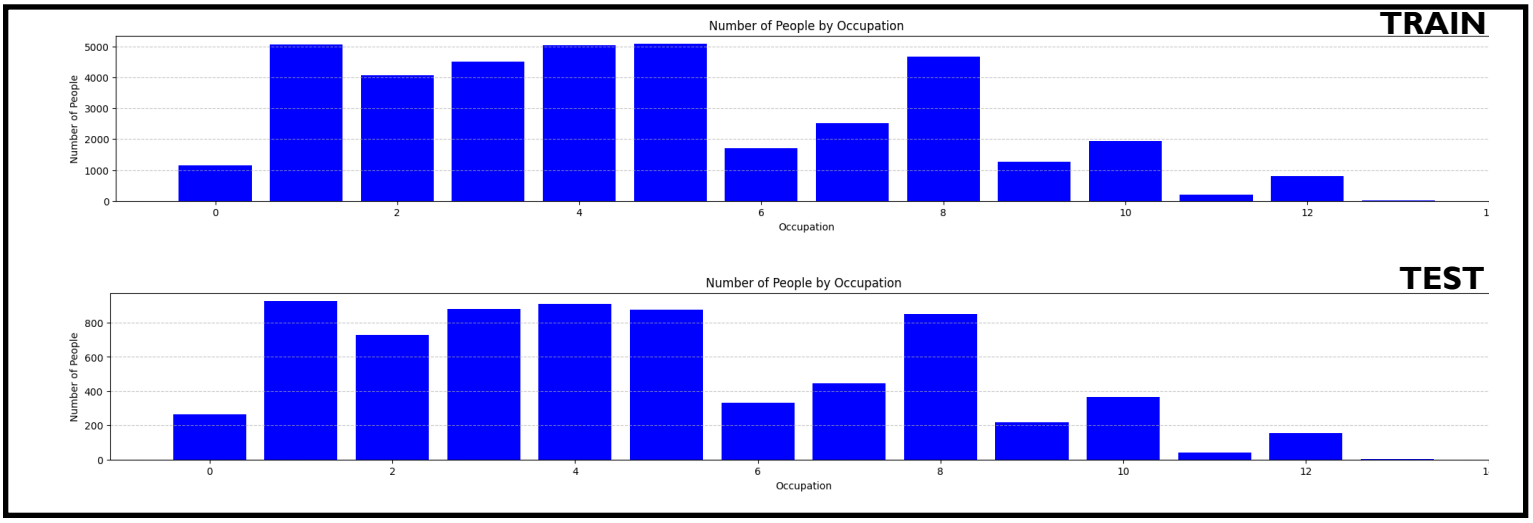
Originally, the dataset (after pre-processing) had 45,882 rows. And we wanted a train set of 38,000 rows and test set of 7,000 rows. So, we used to stratified sampling to do this.

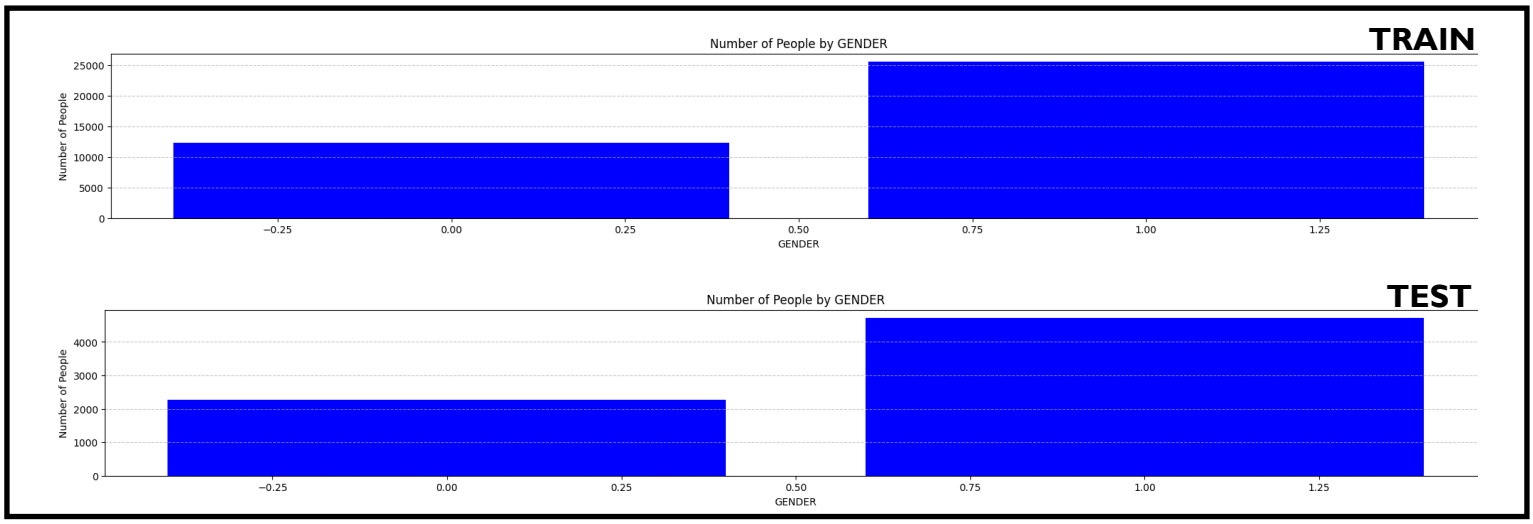
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split       # FOR splitting dataset into train and test.

# Load the training split and test split

data = pd.read_csv('final.csv')
data = data.dropna()     # Remove any invalid rows. This will produce at 45552 rows. PRE-PROCESSING FINAL STEP.

X = data.drop('income_0', axis=1)        # axis = 1 means drop this income column. Retain just features.
y = data['income_0']

# Performed stratified split into training (38000 rows) and test (7000 rows)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 38000,
test_size=7000, random_state=42, stratify=y)

# Concatenated X and y for both training and test sets
train_set = pd.concat([X_train, y_train], axis=1)  # 38,000 rows
test_set = pd.concat([X_test, y_test], axis=1)     # 7,000 rows
```

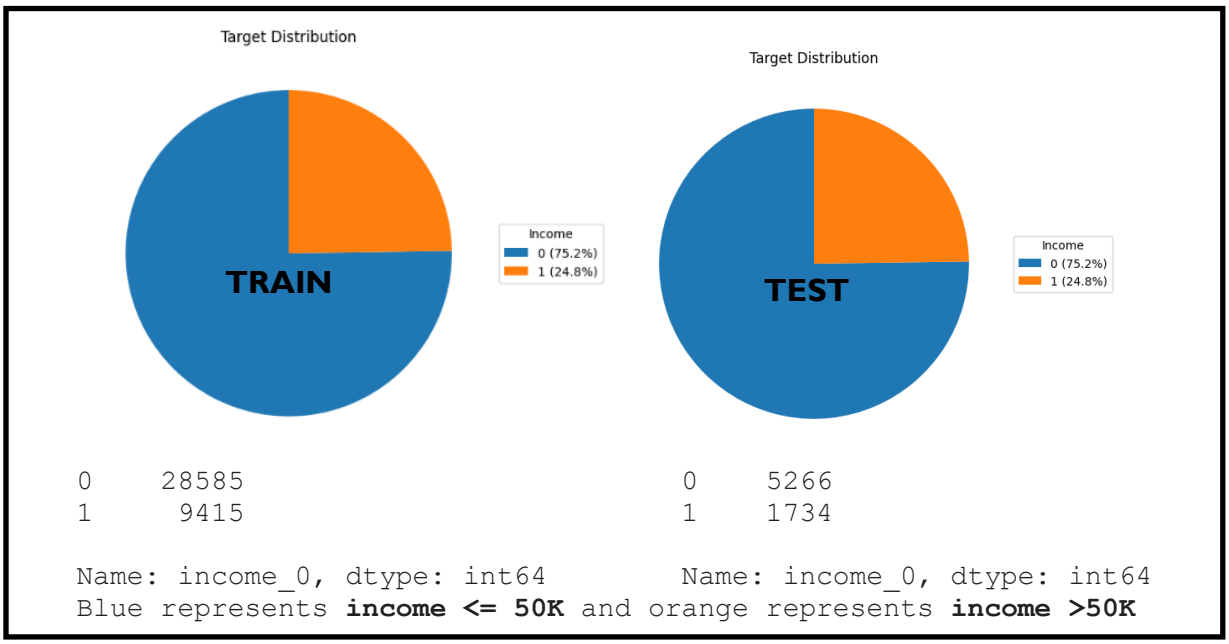
To check class balance/equivalent data distribution in the splits, we plotted graphs to check how different features are distributed across the splits.












```
0    28585                          0    5266
1     9415                          1    1734

Name: income_0, dtype: int64        Name: income_0, dtype: int64
Blue represents income <= 50K and orange represents income >50K
```

You can clearly see that the distribution among both the splits is almost equivalent!

The code for all these graphs is well documented and presented in "*Train/Test Split.ipynb*"

MODEL TRAINING

for **Random Forest Classifier**

Now that we have our training and test splits ready, we can finally train our model! For absolute code details, refer to "TRAINING.ipynb".

In our RF classifier, we're using 100 decision tree models and **entropy** criterion for decision ordering.

```
from sklearn.ensemble import RandomForestClassifier

# Created a RF classifier
clf = RandomForestClassifier(n_estimators = 100, random_state = 42, criterion = 'entropy')

# Training the model on the training dataset
# fit function is used to train the model using the training sets as parameters
# 'ravel' flatterns the y_train field
clf.fit(X_train, y_train.ravel())

# performing predictions on the test dataset
y_pred = clf.predict(X_test)
```

Then we obtain Accuracy metrics based on the predictions. *First, import metric tools from sklearn. We require accuracy, precision, recall and confusion matrix.*

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Testing set Accuracy: {accuracy*100}%')

# Precision
precision = precision_score(y_test, y_pred)
print(f'Testing set Precision: {precision*100}%')

# Recall
recall = recall_score(y_test, y_pred)
print(f'Testing set Recall: {recall*100}%')

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print(f'Testing set Confusion Matrix:\n{conf_matrix}')

# Performing predictions on the train dataset
x_pred = clf.predict(X_train)

# Compute accuracy on train set.
accuracy = accuracy_score(y_train, x_pred)
print(f'Training set Accuracy: {accuracy*100}%')

# Precision
precision = precision_score(y_train, x_pred)
print(f'Training set Precision: {precision*100}%')

# Recall
recall = recall_score(y_train, x_pred)
```

```
print(f'Training set Recall: {recall*100}%')

# Confusion Matrix
conf_matrix = confusion_matrix(y_train, x_pred)
print(f'Training set Confusion Matrix:\n{conf_matrix}')
```

Output:

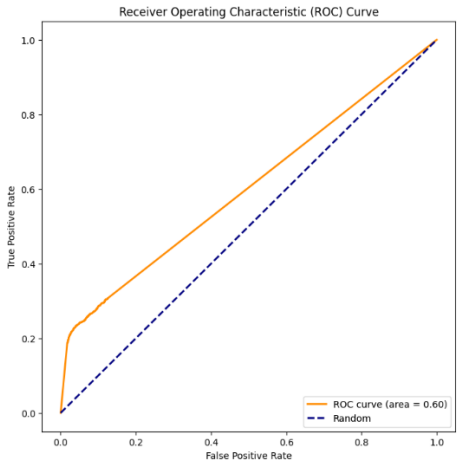
```
Testing set Accuracy:   85.14285714285714%
Testing set Precision:  74.14871438498957%
Testing set Recall:     61.4985590778098%
Testing set Confusion Matrix:
[[4893  372]
 [ 668 1067]]


Training set Accuracy:  99.98947368421052%
Training set Precision: 99.98937977909941%
Training set Recall:    99.96814610320662%
Training set Confusion Matrix:
[[28581     1]
 [    3  9415]]
```

The Random Forest classifier is extremely efficient in **fitting the data it was trained on with only 4 misclassifications in 38,000 predictions (99.98% accuracy)!**

On the test set as well, it shows an excellent accuracy of **85.14%**. It's also well precise in its predictions but it wasn't able to properly identify all true positives (low recall score). Overall, it performs with a decent accuracy.

ROC curve:

```
from sklearn.metrics import roc_curve, auc

fpr, tpr, _ = roc_curve(y_test, y_pred)

# Calculate the area under the curve (AUC)
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



Now, let's compare other models!

Logistic Regression:

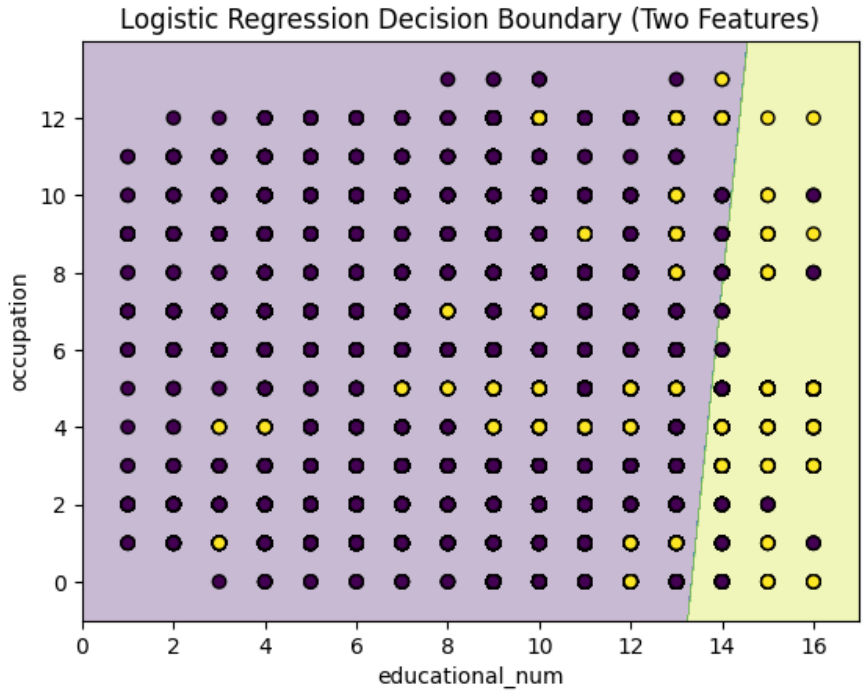
```
from sklearn.linear_model import LogisticRegression       # IMPORT LOGISTIC REGRESSION CLASSIFIER.

# Create an instance of the Logistic Regression classifier with specified parameters
logreg_classifier = LogisticRegression(random_state=42)

# Train the Logistic Regression classifier on the training data
logreg_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred_logreg = logreg_classifier.predict(X_test)
x_pred_logreg = logreg_classifier.predict(X_train)

# Evaluate the performance
accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
precision_logreg = precision_score(y_test, y_pred_logreg)
recall_logreg = recall_score(y_test, y_pred_logreg)
conf_matrix_logreg = confusion_matrix(y_test, y_pred_logreg)

# Print the results
print(f'Testing Logistic Regression Accuracy: {accuracy_logreg}')
print(f'Testing Logistic Regression Precision: {precision_logreg}')
print(f'Testing Logistic Regression Recall: {recall_logreg}')
print(f'Testing Logistic Regression Confusion Matrix:\n{conf_matrix_logreg}')

# Evaluate the performance
accuracy_logreg = accuracy_score(y_train, x_pred_logreg)
precision_logreg = precision_score(y_train, x_pred_logreg)
recall_logreg = recall_score(y_train, x_pred_logreg)
conf_matrix_logreg = confusion_matrix(y_train, x_pred_logreg)

# Print the results
print(f'Training Logistic Regression Accuracy: {accuracy_logreg}')
print(f'Training Logistic Regression Precision: {precision_logreg}')
print(f'Training Logistic Regression Recall: {recall_logreg}')
print(f'Training Logistic Regression Confusion Matrix:\n{conf_matrix_logreg}')
```

Output:

```
Testing Logistic Regression Accuracy:   0.7908571428571428
Testing Logistic Regression Precision:  0.7087827426810478
Testing Logistic Regression Recall:     0.26512968299711814
Testing Logistic Regression Confusion Matrix:
[[5076  189]
 [1275  460]]
Training Logistic Regression Accuracy:  0.7913157894736842
Training Logistic Regression Precision: 0.7108843537414966
Training Logistic Regression Recall:    0.2662985771926099
Training Logistic Regression Confusion Matrix:
[[27562  1020]
 [ 6910  2508]]
```

Logistic regression is a linear classifier thus will perform poorly. Following is the decision boundary line obtained between **education** and **occupation**:

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

features_to_visualize = ['educational_num', 'occupation']
X_train_subset = X_train[features_to_visualize]

# Plot the decision boundary. Finds the minimum and maximum values for the first and second features in your training
dataset. The - 1 and + 1 are used to add some padding to the plot.
x_min, x_max = X_train_subset.iloc[:, 0].min() - 1, X_train_subset.iloc[:, 0].max() + 1
y_min, y_max = X_train_subset.iloc[:, 1].min() - 1, X_train_subset.iloc[:, 1].max() + 1

# The np.meshgrid function creates a meshgrid from the specified ranges. It generates a grid of points for both x and y
axes, where each point represents a potential input to the classifier.
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))

# The np.c_ function concatenates the flattened xx and yy arrays into a single array. This array is then passed to the
predict method of the logistic regression classifier (logreg_classifier), which assigns a predicted class (0 or 1) to
each point in the meshgrid.
Z = logreg_classifier.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot decision boundary
plt.contourf(xx, yy, Z, alpha=0.3, cmap='viridis')

# Scatter plot of the training data
plt.scatter(X_train_subset.iloc[:, 0], X_train_subset.iloc[:, 1], c=y_train, cmap='viridis', edgecolors='k')
plt.title('Logistic Regression Decision Boundary (Two Features)')
plt.xlabel(features_to_visualize[0])
plt.ylabel(features_to_visualize[1])
plt.show()
```



Support Vector Machines

We're using the RBF kernel for non-linearity with regularization parameter as 1.0

```
from sklearn.svm import SVC

# Create an instance of the SVM classifier
svm_classifier = SVC(C=1.0, kernel='rbf', random_state=42)

# Train the SVM classifier on the training data
svm_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred_svm = svm_classifier.predict(X_test)

# Make predictions on the train data
x_pred_svm = svm_classifier.predict(X_train)

# Accuracy
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f'Testing set Accuracy: {accuracy_svm*100}%')

# Precision
precision_svm = precision_score(y_test, y_pred_svm)
print(f'Testing set Precision: {precision_svm*100}%')

# Recall
recall_svm = recall_score(y_test, y_pred_svm)
print(f'Testing set Recall: {recall_svm*100}%')

# Confusion Matrix
conf_matrix_svm = confusion_matrix(y_test, y_pred_svm)
print(f'Testing set Confusion Matrix:\n{conf_matrix_svm}')


# Compute accuracy on train set.
accuracy_svm = accuracy_score(y_train, x_pred_svm)
print(f'Training set Accuracy: {accuracy_svm*100}%')

# Precision
precision_svm = precision_score(y_train, x_pred_svm)
print(f'Training set Precision: {precision_svm*100}%')

# Recall
recall_svm = recall_score(y_train, x_pred_svm)
print(f'Training set Recall: {recall_svm*100}%')

# Confusion Matrix
conf_matrix_svm = confusion_matrix(y_train, x_pred_svm)
print(f'Training set Confusion Matrix:\n{conf_matrix_svm}')
```

Output:

```
Testing set Accuracy: 79.0%
Testing set Precision: 94.61279461279462%
Testing set Recall: 16.195965417867438%
Testing set Confusion Matrix:
[[5249   16]
 [1454  281]]
Training set Accuracy: 79.12105263157895%
Training set Precision: 96.49122807017544%
Training set Recall: 16.351667020598853%
Training set Confusion Matrix:
[[28526    56]
 [ 7878  1540]]
```

SVMs produce very strong precision but lack accuracy on both test and train sets.

kNN Classifier

```
from sklearn.neighbors import KNeighborsClassifier

K = []
training = []
test = []
scores = {}

for k in range(2, 21):      # Try for different values of k.
    clf = KNeighborsClassifier(n_neighbors = k)
    clf.fit(X_train, y_train.ravel())

    training_score = clf.score(X_train, y_train.ravel())
    test_score = clf.score(X_test, y_test.ravel())
    K.append(k)

    training.append(training_score)
    test.append(test_score)
    scores[k] = [training_score, test_score]

for keys, values in scores.items():
    print(keys, ':', values)
```

Output:

```
2 : [0.8719210526315789, 0.7791428571428571] (Best observed overall accuracy)
3 : [0.863578947368421, 0.7591428571428571]
4 : [0.8352105263157895, 0.7841428571428571]
5 : [0.8307894736842105, 0.7701428571428571]
6 : [0.820078947368421, 0.7858571428571428]
7 : [0.8186315789473684, 0.7787142857142857]
8 : [0.8120263157894737, 0.7884285714285715]
9 : [0.8114210526315789, 0.7838571428571428]
10 : [0.8077631578947368, 0.7888571428571428]
11 : [0.8067368421052632, 0.7854285714285715]
12 : [0.8041052631578948, 0.7907142857142857]
13 : [0.8049736842105263, 0.7892857142857143]
14 : [0.8031052631578948, 0.7911428571428571]
15 : [0.8035526315789474, 0.7882857142857143]
16 : [0.8013947368421053, 0.791]
17 : [0.8018947368421052, 0.7905714285714286]
18 : [0.799921052631579, 0.7915714285714286]
19 : [0.8003684210526316, 0.7917142857142857]
20 : [0.7996052631578947, 0.792]
```

Thus, our choice for using the Random Forest Classifier was good! It had highest test and train accuracy of 85% and 99.98% respectivelty!

Thank you for reading!

This took a **tremendous** amount of effort to make and bring together. We hope you liked it!