Introduction

This assignment presents a binary classification problem using the Adult Data Set from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Adult). The objective is to employ a Machine Learning classifier to predict (Y/N) the income of an individual, with certain features, exceeds $50,000 per annum. In addition, we are required to use 90% of the data set for training, and 10% for testing purposes.

The data set is described as having the following characteristics:

Data set characteristics: Multivariate

Number of instances: 48842

Number of attributes: 14

Attributes characteristics: Categorical & Integer

Target characteristics: Categorical

Missing values: Yes

These will be further expanded on in the data preprocessing section.

I have employed *Jupyter Notebook*, a web-based IDE, that allows interactive computing through the combination of code, equations, narrative text, and visualization. It is open-source and is actively used by organizations and individuals alike for data science projects. Implementation is achieved using Python programming language along with the popular ML library scikit-learn, which is built upon data processing and visualization libraries such as NumPy and matplotlib. It is simple yet effective as it allows the use of efficient tools for predictive data analysis. Additionally, other data storage, processing, and visualization libraries used i.e., pandas, seaborn etc.

The entire ML workflow has been adopted for this assignment: historical data, model building, model evaluation, model optimization, new data, & results. These are discussed in depth in the subsequent sections.

Methodology & Justification

I. Algorithm Description

I have employed the use of the *Random Forest Classifier*, which is a supervised, non-parametric, meta classifier that fits several decision trees on randomly selected various sub-samples of the dataset and averages to improve predictive accuracy.

To select a single ML classifier, I attempted to solve the task at hand using both the *K-Nearest Neighbor* and the R*andom Fore*st classifier. The results obtained showed similar trends, however there were obvious advantages with Random Forests:

- Employs several decision trees and takes the majority vote for the final prediction.
- Highly accurate and not prone to overfitting due to averaging.
- Takes random samples of the data, thus eliminating bias.

To fully understand how the Random Forest classifier works, let's briefly look at the Decision Tree classifier – which it employs. The decision tree algorithm breaks down the training dataset into branches based on attributes/features until it can completely predict the output/label. It uses information gain, which is a measure of how uncertainty in the target variable is reduced, given a set of independent variables. This is calculated using entropy, which is a metric for calculating uncertainty. This allows it to choose the features for the best split at each possible node.

Random Forests differentiates from decision trees since it uses random samples of the training data with different feature sets rather than just one. This is known as Bootstrap sampling. The data fed to the decision trees results in different outputs. A majority voting system is employed to obtain the final output. The figure below depicts this.

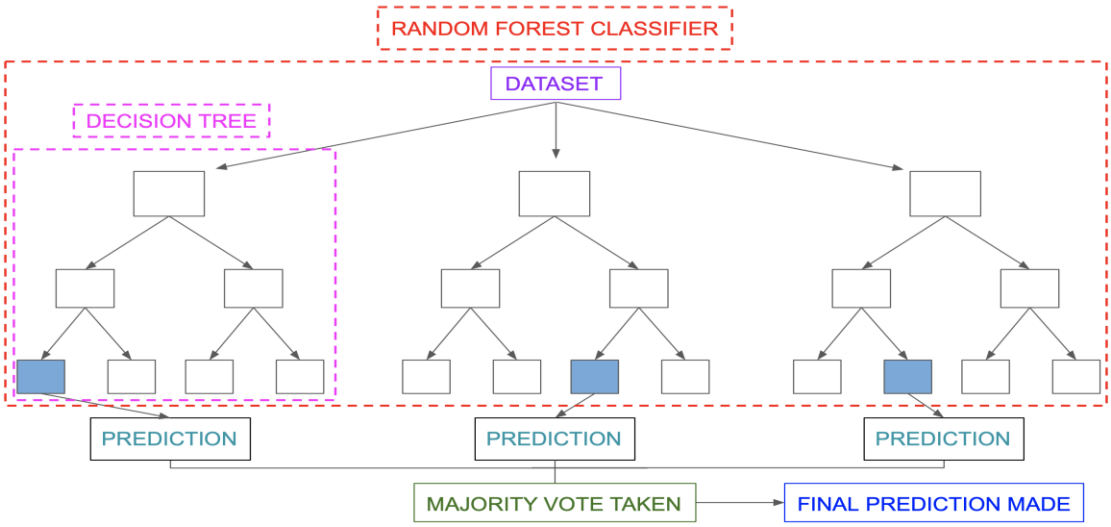


Image Source: Medium

II. Exploratory Data Analysis

Under this section, I analyzed the data set using various statistical and visualization functions.

We have a total of 48842 instances with 14 features. Amongst them, 8 features are categorical in nature whilst 6 are continuous. We see all continuous features do not share the same scale, whilst categorical features range from binary to multiple valued. The figure below shows the characteristics of the data set.

```
In [4]: new_data.info()

<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   workclass       48842 non-null  object
 2   fnlwgt          48842 non-null  int64
 3   education       48842 non-null  object
 4   education-num   48842 non-null  int64
 5   marital-status  48842 non-null  object
 6   occupation      48842 non-null  object
 7   relationship    48842 non-null  object
 8   race            48842 non-null  object
 9   sex             48842 non-null  object
 10  capital-gain    48842 non-null  int64
 11  capital-loss    48842 non-null  int64
 12  hours-per-week  48842 non-null  int64
 13  native-country  48842 non-null  object
 14  income          48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

The value counts for the target label for each class is as follows:

<=50K	**37155**	**76.07%**
>50k	**11687**	**23.93%**

Our data set is unbalanced with a high majority of the instances belonging to individuals who earn less than or equal to $50K per annum. However, this is also a representation of the social (income) trend of numerous individuals that was captured as a result of the data collection activity and, is representative of future data/trends.

We look at some statistics of the two feature groups and visualize their relationship with the target column i.e., income.

```
In [9]: new_data.describe().T
```

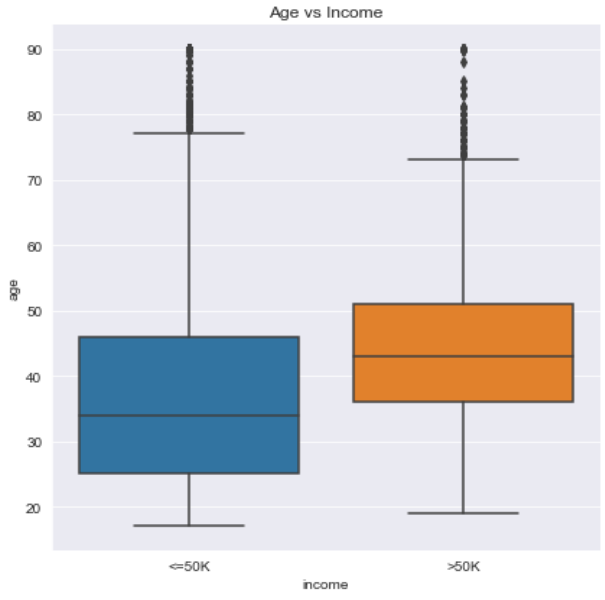
Out[9]:

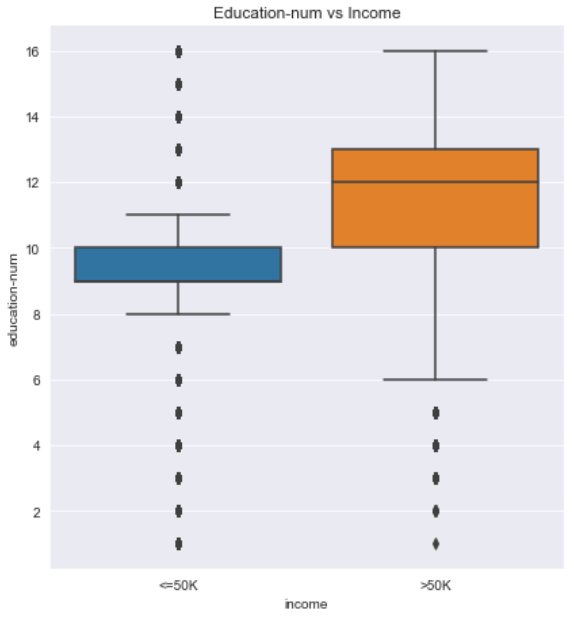
	count	mean	std	min	25%	50%	75%	max
age	48842.0	38.643585	13.710510	17.0	28.0	37.0	48.0	90.0
fnlwgt	48842.0	189664.134597	105604.025423	12285.0	117550.5	178144.5	237642.0	1490400.0
education-num	48842.0	10.078089	2.570973	1.0	9.0	10.0	12.0	16.0
capital-gain	48842.0	1079.067626	7452.019058	0.0	0.0	0.0	0.0	99999.0
capital-loss	48842.0	87.502314	403.004552	0.0	0.0	0.0	0.0	4356.0
hours-per-week	48842.0	40.422382	12.391444	1.0	40.0	40.0	45.0	99.0

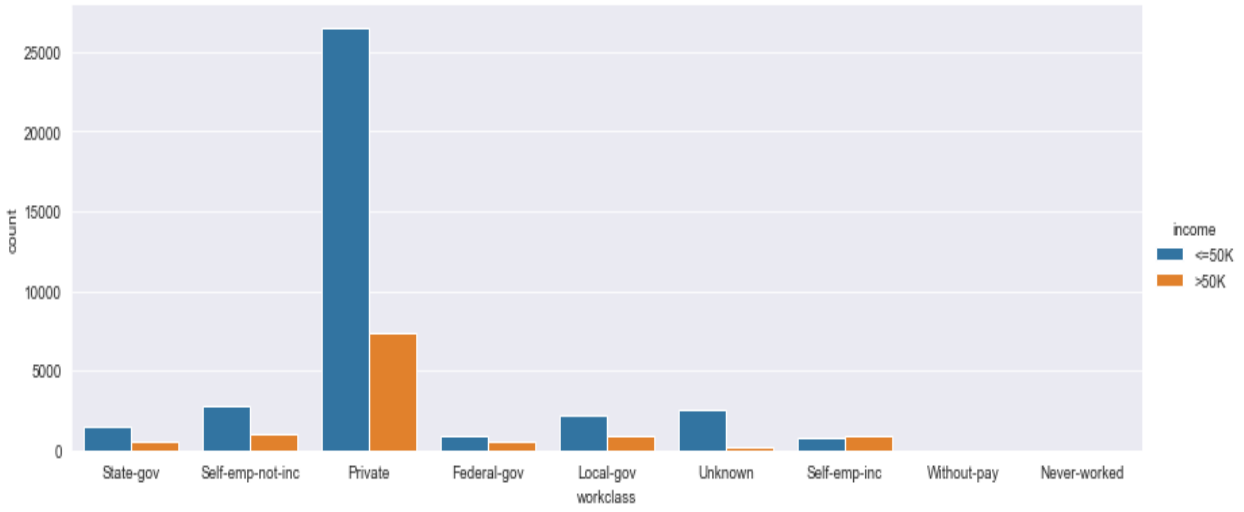
```
In [10]: new_data.describe(include="object").T
```

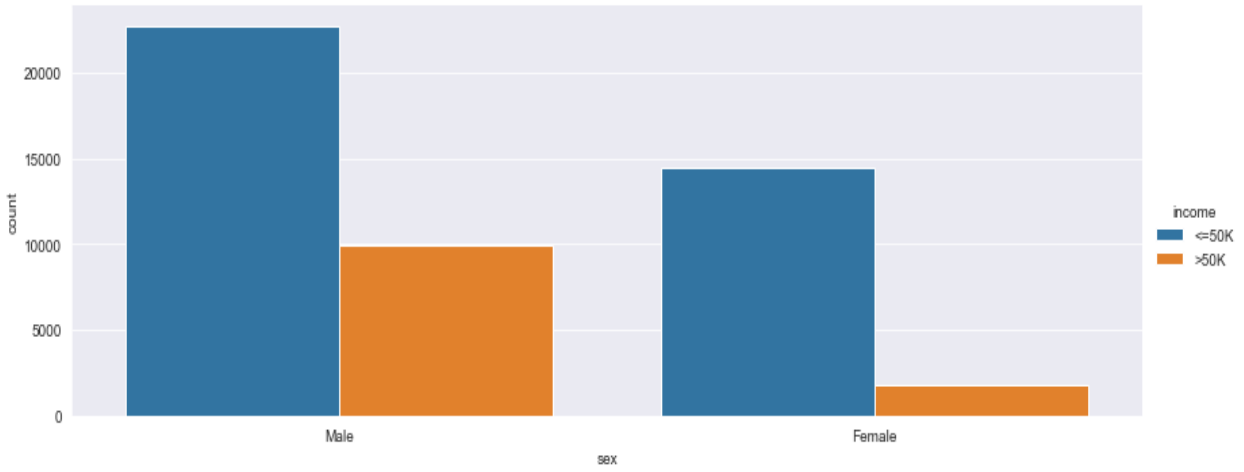
Out[10]:

	count	unique	top	freq
workclass	48842	9	Private	33906
education	48842	16	HS-grad	15784
marital-status	48842	7	Married-civ-spouse	22379
occupation	48842	15	Prof-specialty	6172
relationship	48842	6	Husband	19716
race	48842	5	White	41762
sex	48842	2	Male	32650
native-country	48842	42	United-States	43832
income	48842	2	<=50K	37155









After conducting data exploratory analysis, we see that all features are relevant to the target label and should be used for model construction purposes. To address biasness in features, we will look at data preprocessing further down the line.

III. Data Preprocessing

Data preprocessing is a crucial step towards ensuring the quality of data for extracting meaningful insights. It involves cleaning and organizing the data for building and training the machine learning model.

Initially, I merged the two data set files i.e., adult and adult.test to obtain the complete data set. This was stored as a comma separated file (csv). All relevant libraries which include pandas, NumPy, matplotlib, seaborn, Scikit learn etc., were imported for use.

i. Missing Values

Typically, data is raw, incomplete, and inconsistent. It needs to be tailored into the right format before it can be fed to our model. Our data set contains 3620 instances that have at least one cell value missing. This is a small yet noticeable figure. In addition, all missing data belongs to the categorical features and carries meaning. Thus, under the imputation strategy applied, I have replaced all such instances with a new category, 'Unknown'. This renders the data set as complete.

ii. Label Encoding

Using the *LabelEncoder* class from the sci-kit learn library, I have encoded the target variable classes into digits, where the new representation is as follows:

<=50K → 0

>50k → 1

This allows for ease when applying the classifier in terms of calculations etc. Although Random forests can handle categorical labels, this is a much more efficient way of handling. In addition, we can easily revert to the original label classes when required.

iii. Feature and Target Data Frame

To perform further preprocessing on the data set it is pertinent that we distinguish data by bifurcating it into two data frames (matrices), the feature set and the target set. This enables us to perform feature scaling and encoding as per our need.

```
x = new_data.drop(['income'], axis = 1)
y = new_data['income']
```

The commands above have been employed in this case.

iv. Feature Scaling

Standardization is a form of feature scaling which is used when you have features with different units and scale. It essentially wraps the values around the mean = 0 with standard deviation = 1. This is done to effectively remove bias which can arise due to features having different scales and thus impacting the overall prediction. It does not contain the variable within a range.
Normalization is another feature scaling technique that normalizes values within a common scale without distorting differences in their range. The overall goal remains the same as before, reducing the influence of a variable/feature on the target variable due to the difference in scale. It scales the features to a range of [0,1] or [-1,1].
I have made use of standardization. The reason for this is that it is not affected greatly by outliers in the dataset as opposed to normalization, which is adversely affected by it. As we have seen already, our numerical features contain a plethora of outliers, specifically 'capital-gain' & 'capital-loss'.

Standardization applies the following formula to each numerical feature for scaling:

$$X_{new} = \frac{X - mean}{std}$$

Using the *StandardScaler* class from the sci-kit learn library, I have standardized 6 continuous features/attributes that the feature data frame possesses.

v. One-Hot Encoding

One-Hot Encoding implements categorical data encoding, where it converts categorical data into numerical one. It takes the categorical feature and splits it into multiple columns based on the number of categories for that feature. If a category is represented by that column, it receives a 1. Otherwise, it receives a 0.
One-Hot Encoding does not assume ordinality in the feature thus preserves the relationship between categories. This essentially differentiates it from Label Encoding, which does. Since many machine learning algorithms assume data is numeric and some require it as well, we need to encode categorical data.

The figure below depicts how one-hot encoding transforms categorical data.

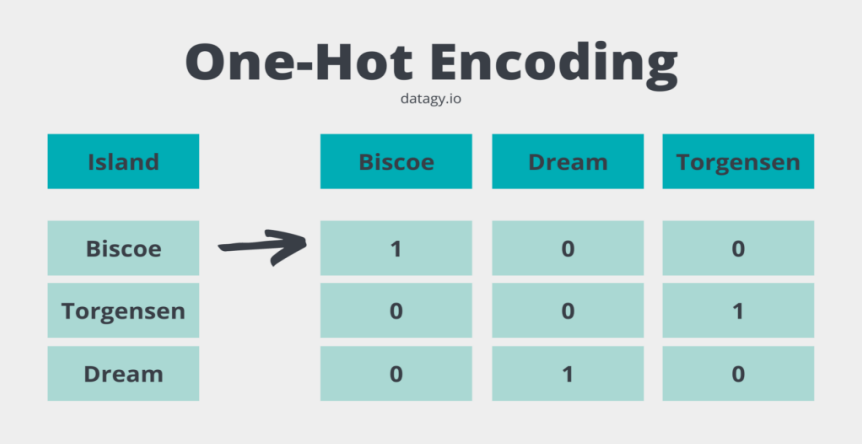


Image Source: datagy.io

Using the *OneHotEncoder* class from the sci-kit learn library, I have encoded 8 categorical attributes that the feature data frame possesses.

Both standardization and one-hot encoding were achieved simultaneously using the *ColumnTransformer* class of the sci-kit learn library. What it does essentially – is that it creates an object which allows us to apply different transformation of various columns of the data frame.

In addition, I have also made use of the *make_pipeline* class, which allows you apply a list of transformers i.e., scaling, encoding etc. and apply a final estimator. The purpose is to assemble multiple steps into a single stream.

Both classes were employed in parallel to streamline the data preprocessing steps.

vi. Training and Test Split

After preprocessing data into the right format, I split the data into training and test data set. For this purpose, I have employed *train_test_split* class.

X_train, X_test, y_train, y_test = train_test_split (X, y, random_state=123, test_size=0.1)

We have four data frames; *X_train* and *y_train*, used for training and validation of the model. *X_test* is used to predict labels and compare with the actual values in *y_test*. The training set is 90% of the data whilst test set is 10%. This is in line with the assignment instructions.

IV. Model Building

Under this section, first and foremost, I implement the Random Forest classifier while looping it several times. This is done to enable selection of optimal parameter values that can impact the training accuracy.
Some of the model hyperparameters considered and the values used for the purpose of this assignment:

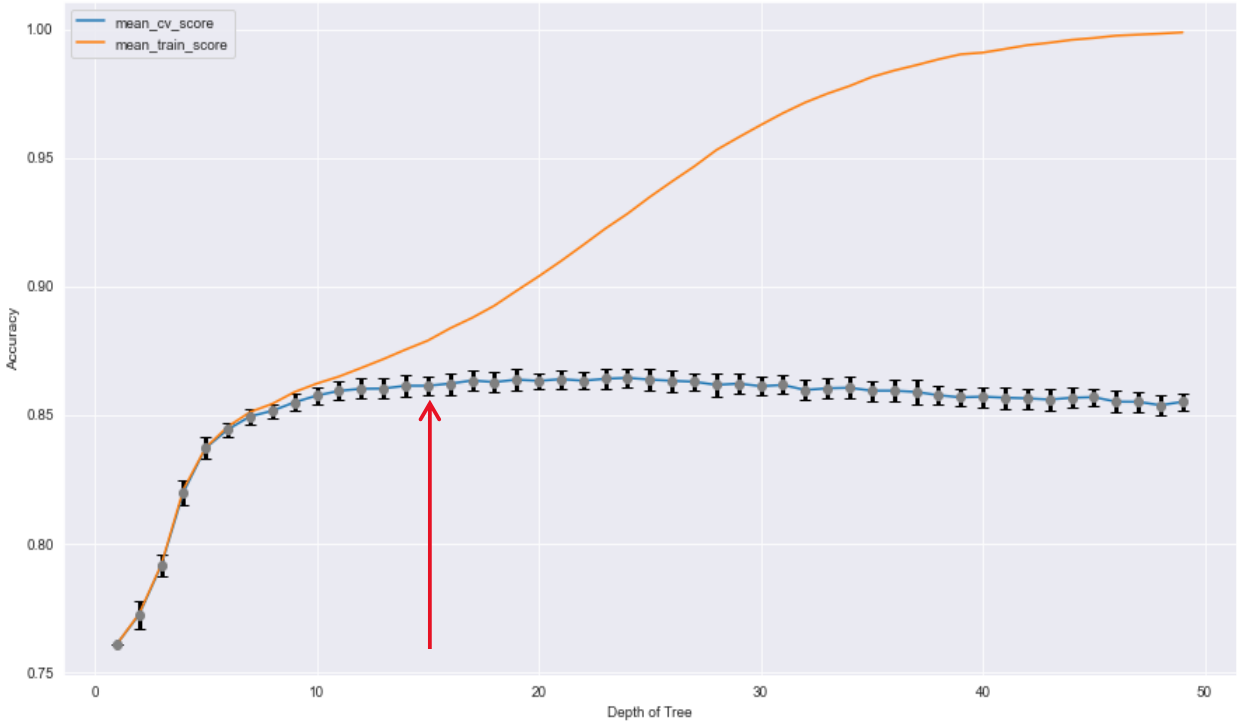
- **n_estimators:** The number of decision trees in the forest. I have used the default value of 100. This is a reasonable number of trees and works well to completely capture the essence of the data set. However, we can also work with a smaller number but need to ensure that we do not underfit on the training set. On the downside, with 100 trees, the classifier takes a lot of time to train.
- **criterion:** "entropy". Entropy is a measure of uncertainty and is used to calculate the information gain. This parameter is tree specific.
- **max_depth**: "int". The maximum depth of the tree. If none is specified, all nodes are expanded until they are completely pure. This can lead to forest overfitting; thus, I implemented a loop along with cross- validation to find the optimal value.
- **max_features**: "sqrt". Size of the random subset of features to consider when splitting a node. For regression, it is often better to use all features, however with classification, it is recommended in literature to use sqrt(n_features), where n_features are the total number of features.

I have employed k-fold cross validation where k = 10. Cross-validation is a statistical method for determining the skill of a machine learning model on a limited data set to gauge its performance on unseen data. It is intuitively simple to grasp and results in less biased estimate of the model's skill.

The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
    a. Take the group as a hold out or test data set
    b. Take the remaining groups as a training data set
    c. Fit a model on the training set and evaluate it on the test set
    d. Retain the evaluation score and discard the model
4. Summarize the skill of the model using the sample of model evaluation scores

To summarize, I have applied 10-fold cross-validation along with a 'For' loop (range=50) to train the classifier and evaluate its performance. The results obtained were visualized through statistical analysis of performance metrics such as cross-validation accuracy, training accuracy, and standard deviation of cross-validation accuracy. This enabled the selection of the optimal value for the model hyperparameter (max_depth) and observing overall performance of the classifier on the training data.



The graph above shows the mean cross-validation accuracy over 50 runs and the associated standard deviation depicted by the black bars at each point. This is in turn compared with the orange curve, which represents the mean training accuracy of the classifier.

As we can see the classifier underfits initially but then peaks in terms of accuracy. The performance of the cross-validation accuracy then reaches a steady state whilst the training accuracy sharply increases indicating that the model is overfitting.

When choosing the best value for the hyperparameter, we need to consider two things: high classification accuracy and least standard deviation. For this reason, based on the graphical representation, I have used max_depth = 15, meaning that each tree will only expand to 15 levels/branches deep.

V. Model Evaluation

Now the classifier's performance is tested on unseen test data with the hyper parameter values obtained during the training phase. We will now look at how the model performs using a set of metrics.

I. Predictive Accuracy

Accuracy is one of the metrics employed for evaluating classification models. It is represented as:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

In terms of binary classification, which is the case here, we have another definition:

$$\frac{TP + TN}{TP + TN + FP + FN}$$

Where TP = True Positive, TN = True Negative, FP = False Positive, FN = False Negative

```
In [48]: print(classification_report(y_test, predictions))
```

	precision	recall	f1-score	support
0	0.88	0.96	0.91	3713
1	0.80	0.57	0.67	1172
accuracy			0.86	4885
macro avg	0.84	0.76	0.79	4885
weighted avg	0.86	0.86	0.85	4885

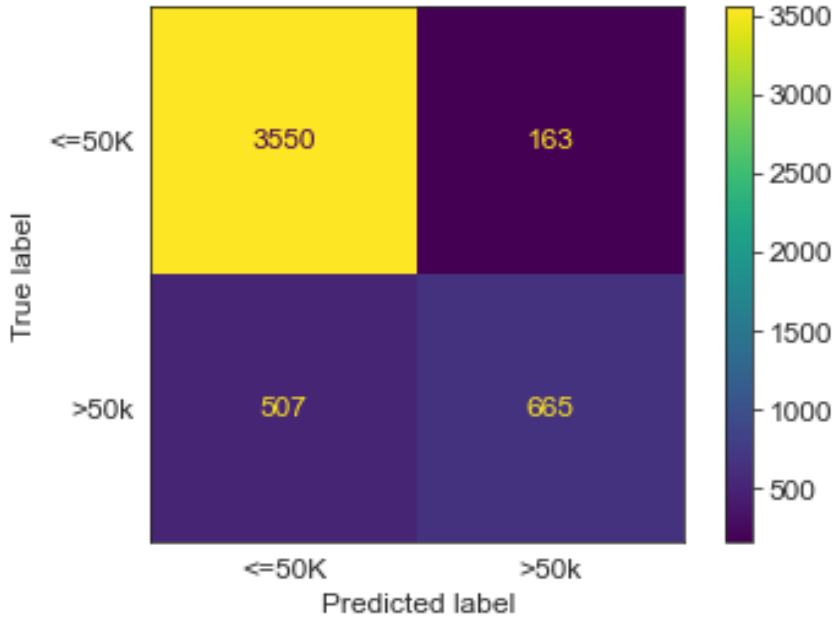
My model outputs an overall predictive accuracy of **86.28%.** Other metrics i.e., recall and f1-score also have similar output.

II. Confusion Matrix

It is a table that visualizes the model's performance. It shows all the possible labels and how the model can correctly or incorrectly predict them. There are four quadrants in the matrix and our explained as follows:

- **True Positive (TP):** Positively labeled instance that the model correctly predicts.
- **False Positive (FP):** Negatively labeled instance that the model incorrectly predicts as positive.
- **False Negative (FN):** Positively labeled instance that the model incorrectly predicts as positive.
- **True Negative (TN):** Negatively labeled instance that the model correctly predicts.

For us, in terms of the model's predictive accuracy, TP and TN hold great importance.



We can describe the confusion matrix as follows:

- TN: 3550 (72.67%)
- TP: 665 (13.61%)
- FN: 507 (10.34%)
- FP: 163 (3.34%)
- Total instances: 4885

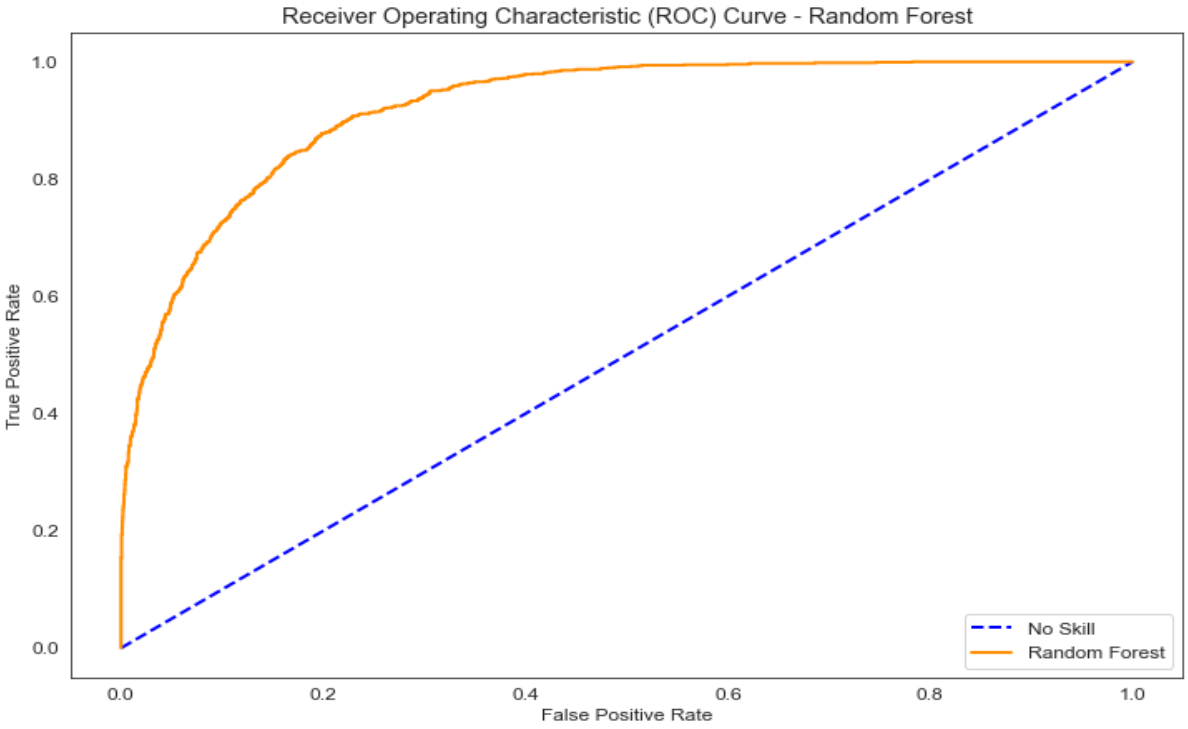
III. Receiver Operating Characteristic (ROC) Curve

ROC curve is another metric for evaluating the performance of a classifier. Typically used for binary classification but can be extended to multi-class classification. It features TP rate on the y-axis and FP rate on the x-axis. Additionally, it uses the prediction probability for each instance and a threshold for success to mark down points on the curve. There are multiple sub-metrics that hold great importance:

- The steepness of the ROC curve. Values near to the top left quadrant of the graph mean high TP rate and low FP, which is ideal.
- The area under the curve, known as the AUC. The larger it is, the better. However, our model cannot output unrealistically high results.

There are two types of models. Skillful models that assign high probabilities to random positive instances than negative ones. This is represented by curve that is steep towards the top-left of the graphical space. A no-skill classifier is one that would not be able to differentiate between classes and would predict a constant class in all cases. This is represented on the graph with a diagonal line and has an AUC of 0.5.

Below is the ROC curve for my classifier when compared to a no-skill classifier.



The AUC for the Random Forest is **0.921** – a very good statistic.

Based on the results obtained from the various performance evaluation metrics that have been employed to gauge the random forest – the outcome/verdict is that the classifier does a great job. Furthermore, it produces similar results when evaluated on the unseen test data when compared to training evaluation. The classifier avoids overfitting and reduces bias drastically using cross-validation and averaging the output of the decisions trees.

However, if we look at the classification report, we see that classifier can correctly predict label 0 i.e., <=50k, but is not that accurate when it comes to predicting label 1 i.e., >50k. This can be attributed to an imbalance in the number of instances that represent label 1, and so the model is unable to generalize as well as it should on the training set.

VI. Conclusion

To conclude, this assignment presented a comprehensive case study of the economic distribution of individuals in terms of their income in various states in the United States. It presents a binary classification problem which is dependent on multiple independent features, both categorical and continuous.

The Random Forest classifier was able to perform well on the given data set after performing preprocessing. The evaluation metrics used depicted near-optimal results. The model was implemented following the entire machine learning workflow. However, there is room for improvement which can be incorporated into the model to equip it better for new (unseen) data. This can primarily be ensured by improving the quality of data which is reflective of future data and effectively training the model to capture the relationship between the feature space and the target variable.