Binary Classification: Adult Dataset

Report for the CEGE0004: Assignment

Kratos

Miron Kiselev
21034710
ucabise@ucl.ac.uk

Pratibha Patel
21152395
ucesatb@ucl.ac.uk

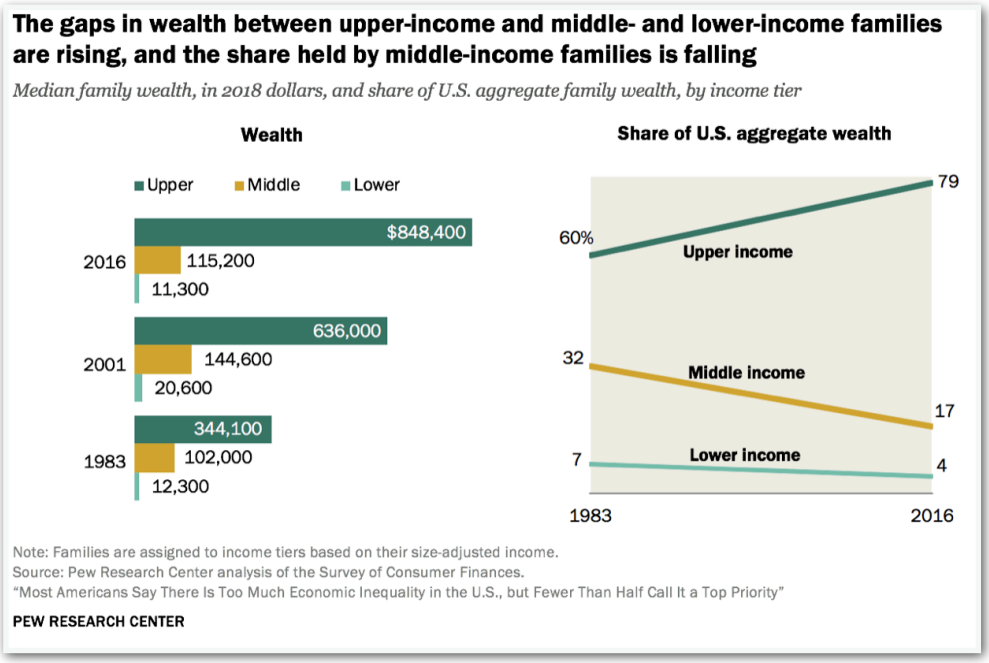
Zichun Wang
20115437
ucesz07@ucl.ac.uk

Chen Gong
21117614
ucescg5@ucl.ac.uk

Baoyan Han
21058730
ucesbh2@ucl.ac.uk

1. Introduction:

Background Context:



The income inequality in the United States of America has been on the rise, with the gap in wealth between upper-income and lower income families reaching its all time high in 2016. As such, it is important to understand the factors that affect the income of an individual in the US in order to explain the macro trends in America's wealth disparity. Income inequality has implications that are much wider than just the economic environment of a nation. For example life expectancy has risen from 1980 among the middle and upper class individuals in the USA, however it has been stagnant for those in the lower bracket (Bor, 2017). Furthermore, the difference in purchasing power between the low and high income demographics create further divides in the social structure of a nation's society (Nwakaroko, 2015). This study leverages the UCI Adult Dataset in order create a model that predicts whether an individual's income is greater than $50,000 per annum based on a variety of factors, such as education, hours worked, occupation and race. The living wage in the United States of America was $68,808 per year in 2019, as such the model can be used to predict whether an individual can afford basic necessities, such as healthcare, food and accommodation without the need of relying on external support (Nadeau, 2020). Therefore, the model can be used by local governments to identify individuals that may require financial assistance.

Measuring Performance:

There are four main ways to measure the performance of a learning model:

Accuracy - is the ratio of correctly predicted observations to the total number of observations.

Precision - is the ratio of correctly predicted positive observations to the total number of positive predictions made by the model. Thus this measure focuses on an algorithm's performance on classifying the positive class.

Recall - is the ratio of correctly predicted positive observations to total positive observations in the dataset, otherwise known as the True Positive Rate.

F1 Score - is a harmonic mean between precision and recall calculated by the following formula:

$$2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

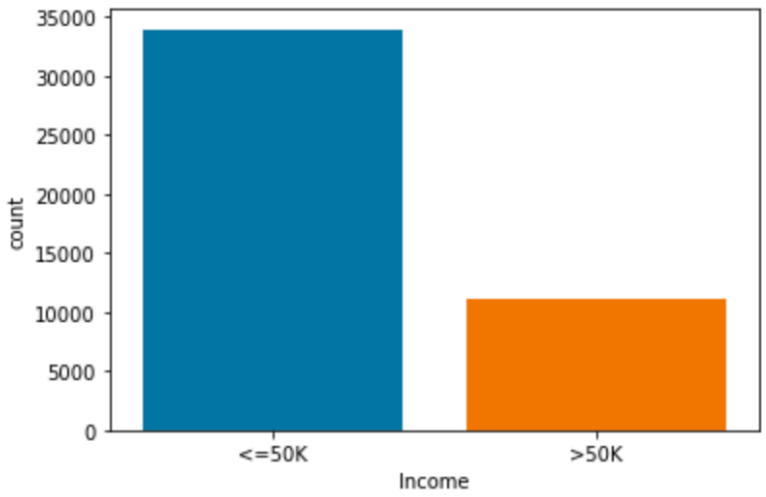
Both, the False Negative and the False Positive classifications are taken into account by the F1 score when the weighted average is calculated.

In order to better understand the skewness of the target variable (Income), the variable was plotted as a frequency bar chart using sns.countplot:

```
ax = sns.countplot(x="Income", data=df)
print(df.Income.value_counts())
```

This produced the following output:

```
<=50K   33973
>50K    11202
Name: Income, dtype: int64
```



As such, it is evident that the variable is left skewed with around 75% of cases having an income of <=$50k. The class distribution in the dataset is uneven meaning that accuracy is not the best performance metric to evaluate the learning models, as 75% accuracy can be achieved through classifying each observation in the majority class. Therefore, the F1 score was selected as the best measure of performance as it combines both, the precision and recall metrics. Furthermore due to the class distribution of the dataset, focus is placed on an algorithm's performance on classifying the minority class.

Furthermore, the dataset contains a column titled 'Final Weight' which indicates the number of individuals a particular observation represents. As some observations account for a larger number of individuals than others, it is important to account for such differences when looking at the classification report for a particular learning algorithm. Therefore, a custom classification function has been created in order to generate a classification report that accounts for the weight of each observation for the minority class:

```
def calculate_weighted_classification(final_weight, y_pred_income, y_actual):
    df = pd.DataFrame(final_weight)
    df["Predicted_Income"] = y_pred_income
    df["Actual_Income"] = y_actual
    true_positive = 0
    false_negatives = 0
    false_positive = 0
    for i in range(len(df)):
        actual_income = df.iloc[i]["Actual_Income"]
        predicted_income = df.iloc[i]["Predicted_Income"]
        final_weight = df.iloc[i]["Final Weight"]
        if actual_income == 1 and predicted_income == 1:
            true_positive += final_weight
        elif actual_income == 1 and predicted_income == 0:
            false_negatives += final_weight
        elif actual_income == 0 and predicted_income == 1:
            false_positive += final_weight
    precision = true_positive / (true_positive + false_positive)
    recall = true_positive / (true_positive + false_negatives)
    f1= (2 * precision * recall) / (precision + recall)
    return precision, recall, f1
```

Completing the task by hand:

Should the outline task be completed without the use of machine learning algorithms, it may be beneficial to use basic statistical tools like the measure of central tendency or dispersion. It may also be possible to use cross-tabulation in order to create a matrix using features that correlate the most with the target variable in order to make predictions on the earning class of a given observation. Finally, simple logistic regression can be performed among the features most correlated with the target variable.

2. Learning Tasks:

There are two types of learning tasks: classification and regression. As the output variable being predicted by the tasks is a discrete value, the task requires classification models. A general overview of a classification function is presented below:

$$y = f(x)$$

where, **y** is the output (target variable),
x is vector of features, so for any vector x, we will receive a desired output,
f(x) is which class/label associated to such vector.

In the discussed dataset, the target variable is 'Income', which shows whether an individual makes <= $50k (Class 0) or >$50k (Class 1) a year. The input variables are the remaining 13 features (such as Age, Education and Occupation…). We learn a target function(f) that best maps input variables(x) to an output variable(y).

3. Material:

Describing the Data:

The Adult Dataset consists of 15 variables. A detailed explanation for each variable can be seen on the table below:

Feature Name:	Description:	Type:	% of Missing Values	Distribution:	Target?
Age:	Age the individual (As the census is only filled in by adults, the minimum age is 18).	Continuous	0	Negative Binomial	No
Workclass:	The type of organisation that the individual is employed by.	Categorical	5.73	Categorical	No
Final Weight:	Number of individuals the observation represents	Continuous	0	Random	No
Education:	Highest level of education achieved	Categorical	0	Categorical	No
Education Number of Years:	Number of years spent in education	Continuous	0	R-cens Normal	No
Marital Status	Whether an individual is married or not.	Categorical	0	Categorical	No
Occupation:	Type of job that the individual does.	Categorical	5.75	Categorical	No
Relationship:	Role of the individual in their family.	Categorical	0	Categorical	No
Race:	Race of the individual.	Categorical	0	Categorical	No
Sex:	Sex of the individual.	Bivariate	0	Bernoulli	No
Capital Gain:	Profit incurred when an asset owned by the individual rises in value.	Continuous	0	Negative Binomial	No
Capital Loss:	Loss incurred when an asset owned by the individual drops in value.	Continuous	0	Negative Binomial	No
Hours per Week:	Number of hours the individual works per week.	Continuous	0	Normal	No
Native Country:	Country in which the individual was born in.	Categorical	1.75	Categorical	No
Income	Whether an individual makes less than or more than $50,000 per year.	Bivariate	0	Bernoulli	Yes

As the data in the dataset is recorded, rather than measured, it is highly unlikely to have any noise.

The missing values were removed via:

```
df.replace({"?": None}, inplace= True)
df.dropna(inplace = True)
```

After the missing values were removed, there were 45,222 observations remaining. Following, the duplicate observations were removed through:

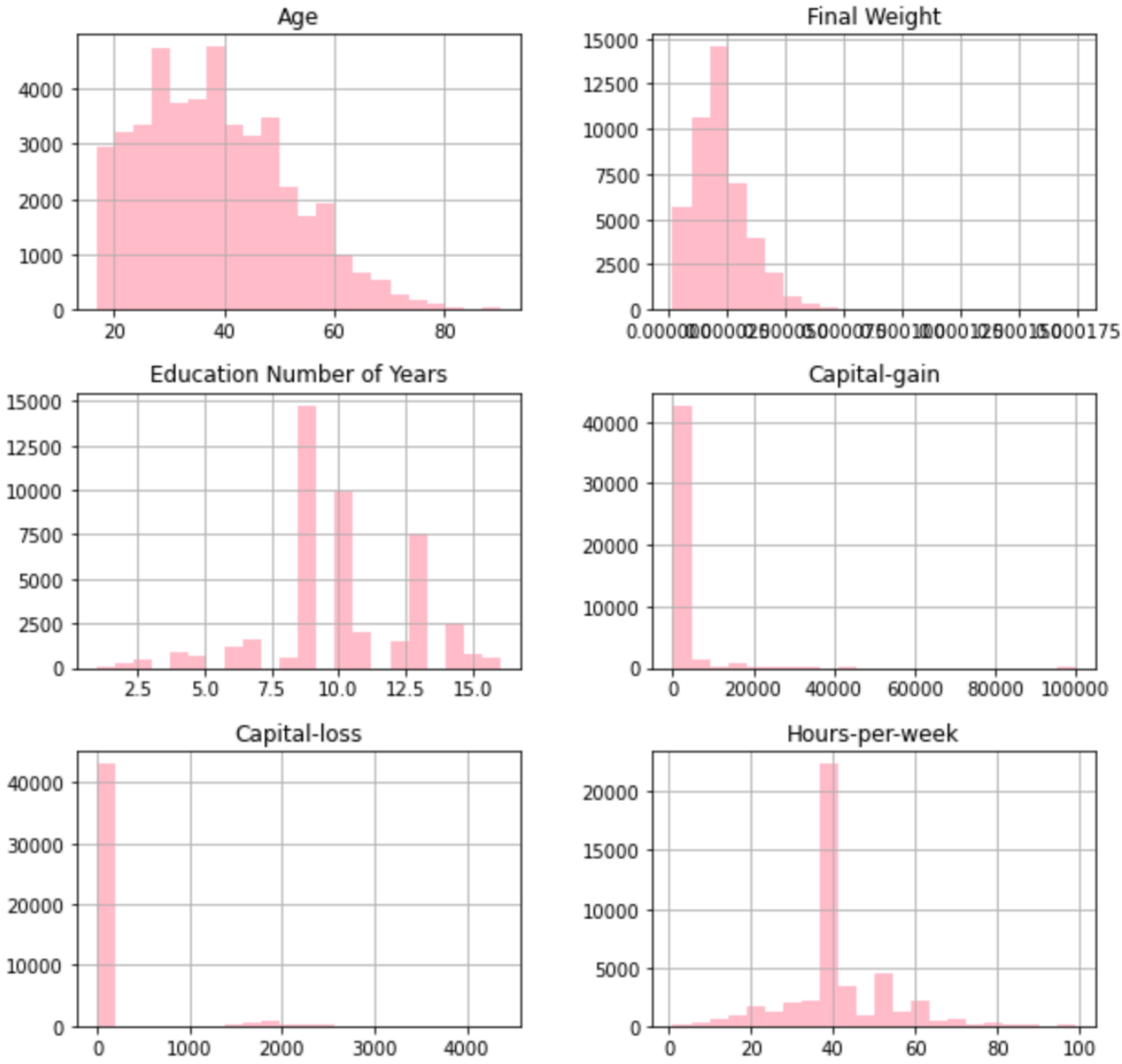
```
df = df.drop_duplicates()
```

As a result of removing 47 duplicate observations, there were 45,175 rows remaining in the dataset.

Exploratory Data Analysis:

In order to best understand the distributions of the continuous variables were visualised in a histogram:

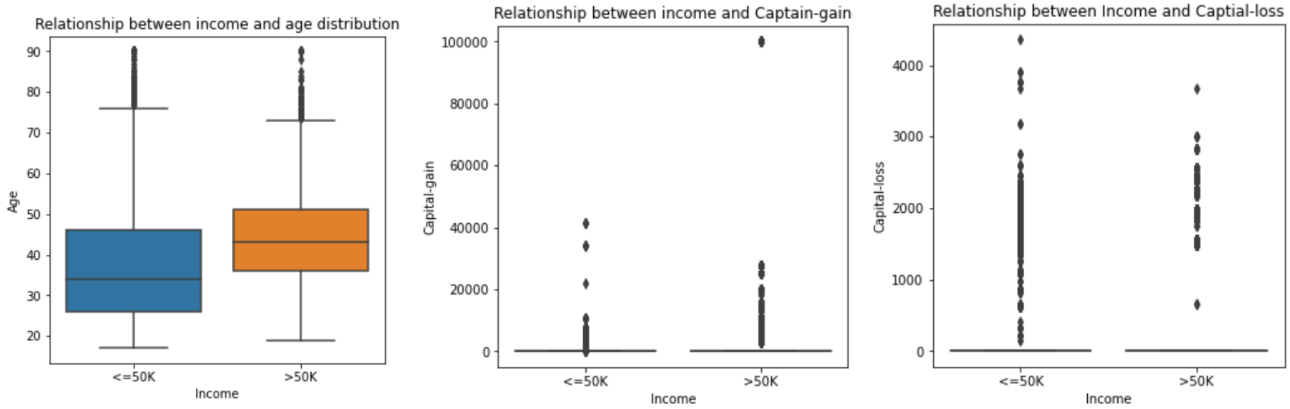
```
df.hist(figsize = (10,10), bins= 22, color = 'pink')
plt.show()
```



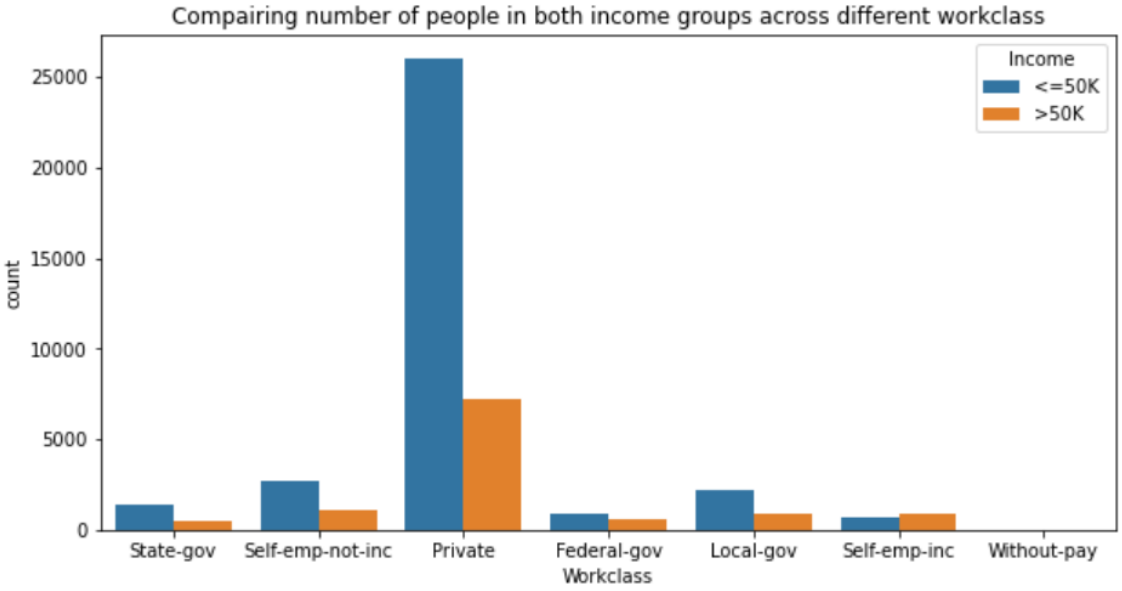
Following, the categorical variables have been visualised through bar charts (see Appendix One). As the dependent variable (Income) has already been visualised under the measuring performance section of this report, it has been omitted.

Bivariate Analysis:

In order to best understand the relationship between the variables, a bivariate analysis has been performed.

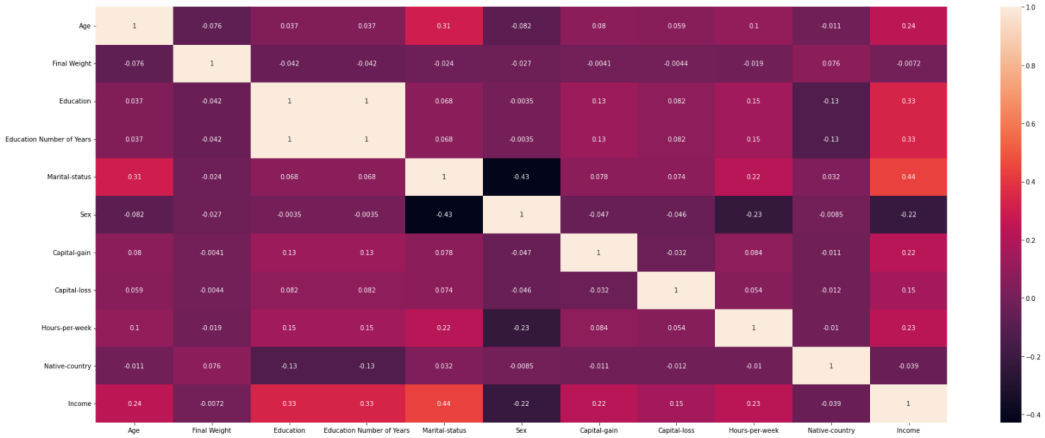


Income group <=50k has lower median age ~34 year than the Income group >50k which has median age of ~ 42 year as seen in the above figures. Most of the capital gains value is accumulated at 0 for both of the income groups. The box plot of the relationship between income and capital-loss is similar to that of the capital gain, where most of the values are concentrated on 0.



The above figure demonstrates that the percentage of individuals earning over $50k a year is relatively consistent among all the work classes. However, the private work sector had the highest absolute count of individuals earning over $50k.

Finally, a correlation matrix was produced in order to best understand the correlations between each variable in the dataset.



Common Data Transformations:

In order for the learning algorithms to understand the data from the dataset, it is necessary to convert categorical and bivariate features into numerical ones. The bivariate variables were transformed into their binary counterparts via the following:

```
sex_dict = {'Male': 0, 'Female': 1}
df["Sex"] = df["Sex"].map(sex_dict)
pd.to_numeric(df['Sex'])
#convert income
income_dict = {'<=50K' : 0, '>50K' : 1}
df['Income']= df["Income"].map(income_dict)
pd.to_numeric(df['Income'])
```

As the 'Native Country' feature was heavily dominated by the 'United States of America' class, it was also transformed into a binary variable through:

```
country_dict = {}
for country in df['Native-country'].values:
    if country == 'United-States':
        country_dict[country] = 0
    else:
        country_dict[country] = 1
df["Native-country"] = df["Native-country"].map(country_dict)
pd.to_numeric(df["Education"])
df.dtypes
```

Following, marriage has been converted into a binary variable, with 0 and 1 indicating married and unmarried observations respectively. Education was also converted into a numerical variable.

```
#converts marriage
marrige_dict = {'Divorced' : 0 ,'Married-AF-spouse' : 1,'Married-civ-spouse' : 1,'Married-spouse-
absent' : 1, 'Never-married' : 0,'Separated' : 0,'Widowed' : 0}
df["Marital-status"] = df["Marital-status"].map(marrige_dict)
pd.to_numeric(df['Income'])
education_dict = {'Bachelors' : 13, 'Prof-school' : 15, 'HS-grad' :9, '5th-6th' : 3, '12th' : 8,
'9th' : 5, 'Doctorate' : 16, '11th' : 7, 'Some-college' : 10, 'Assoc-voc' : 11, 'Preschool' : 1,
'1st-4th' : 2, 'Assoc-acdm' : 12, 'Masters' : 14, '7th-8th' : 4, '10th' : 6}
df['Education']= df["Education"].map(education_dict)
pd.to_numeric(df["Education"])
```

For the remaining features, reasonable groupings have been made in order to reduce the number of features in the dataset after Hot-One encoding. The occupation feature has been grouped in accordance with the ISCO international standards.

```
workclass_dict = {'Federal-gov' : 'Government', 'Local-gov' :"Government" , 'Private' : 'Private',
'Self-emp-inc': "Self-Employed",'Self-emp-not-inc': "Self-Employed", 'State-gov' : 'Government',
'Without-pay' : 'Other'}
df["Workclass"] = df["Workclass"].map(workclass_dict)
relationship_dict = {'Husband' : 'Husband', 'Not-in-family': 'Not-in-Family', 'Other-relative' :
'Other', 'Own-child': 'Child', 'Unmarried': 'Not-in-family', 'Wife' : 'Wife'}
df["Relationship"] = df["Relationship"].map(relationship_dict)
occupation_dict = {
    'Adm-clerical' : 'Level 3',
    'Exec-managerial' : 'Level 4',
    'Handlers-cleaners': 'Level 1',
    'Prof-specialty' :'Level 4',
    'Other-service' : 'Level 3',
    'Sales' : 'Level 3',
    'Transport-moving': 'Level 1',
    'Farming-fishing' : 'Level 2',
    'Machine-op-inspct' : 'Level 2',
    'Tech-support': 'Level 4',
    'Craft-repair' : 'Level 2',
    'Protective-serv': 'Level 2',
    'Armed-Forces' : 'Level 0',
    'Priv-house-serv' : "Level 1"}
df["Occupation"] = df["Occupation"].map(occupation_dict)
```

Following the groupings, a Hot-One encoder was performed in order to convert the grouped categorical features into their binary counterparts.

```
occupation_econoding = pd.get_dummies(df['Occupation'], prefix='Occupation:')
df = pd.concat([df,occupation_econoding],axis=1)
df = df.drop(columns='Occupation')
race_encoding = pd.get_dummies(df['Race'], prefix='Race:')
df = pd.concat([df,race_encoding],axis=1)
df = df.drop(columns='Race')
workclass_encoding = pd.get_dummies(df['Workclass'], prefix='Workclass:')
df = pd.concat([df,workclass_encoding],axis=1)
df = df.drop(columns='Workclass')
relationship_enconding = pd.get_dummies(df['Relationship'], prefix='Relationship:')
df = pd.concat([df,relationship_enconding],axis=1)
df = df.drop(columns='Relationship')
```

Finally, as the correlation matrix shows a 1:1 correlation between the Education Number of Years and Education attributes and the correlation between each of those features and the rest of the dataset is equivalent, it is reasonable to assume that the education number of years was not recorded and instead was estimated in a similar way to the one shown above. As such, Education number of years has been removed from the dataset.

Dataset Sampling:

In order to keep the datasets consistent across all learning notebooks, the dataset was split after the transformations have been implemented. 80% of the data would be used for training the algorithm, with the remaining 20% used for testing. As the number of observations is significantly lower in the minority class of the target variable, the dataset was stratified in accordance to it, to keep the distributions relatively consistent across the train and test splits. Validation was be achieved using k-fold validation on the testing set, and thus a validation set is not required. The split datasets were then saved to their respective CSV files such that they can be used by the learning algorithms.

```
train, test = train_test_split(df, test_size=0.2, stratify= df['Income'])
train.to_csv('train.csv', encoding='utf-8', index=False)
test.to_csv('test.csv', encoding='utf-8', index=False)
```

4. Technology:

Jupyter Notebooks was the main development tool for the project. The web-based interactive software allowed for execution of dependant python scripts with the output of each code block displayed directly below it. In addition, the software allows for rich text elements to be included, which was used to provide additional information about the functionality of important code blocks.

In addition to the default libraries contained in **Python 3.8**, **Conda** was used to manage all the required dependancies:

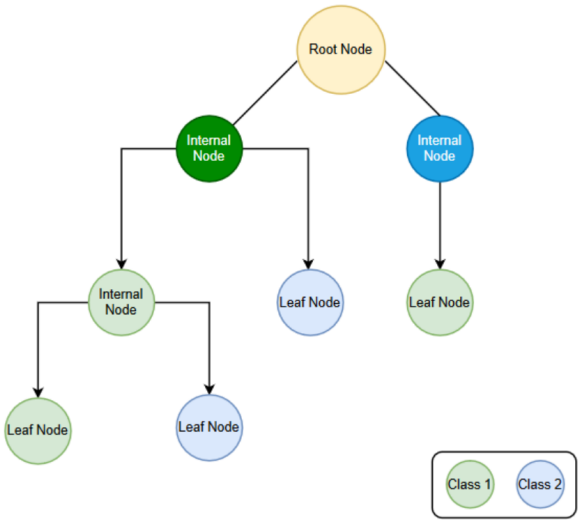
1. **Pandas & Numpy:** were used to read from the CSV documents provided, preprocess the data and export the relevant train/test splits into new CSVs.
2. **MatPlotLib:** was used for data visualisation during the exploratory data analysis stage. It was also used to demonstrate the confusion matrices for each learning algorithm.
3. **Seaborn:** was used for histogram and box plots visualisation during exploratory data analysis. It was also used to create the correlation matrix.
4. **JobLib & Pickle:** used to export the tuned models from each notebook and import the models into the model ensemble. This was implemented in order to avoid training the models twice.
5. **Scikit-Learn:** was used to create each learning model, tune their hyper-parameters (via GridSearch and RandomSearch), validate the models and display classification reports with their relevant metics. It was also used to standardise the data for the the MLP classifier.

Finally, **Git** and **Github** were used for source code management and ensuring that each team-member had access to the latest version of the notebooks.

5. Decision Trees:

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The aim is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features (Scikit-Learn, 2016). There is no special pre-processing of the dataset required for the DT family, as DTs are able to take continuous, binary and categorical variables as inputs. However, the Scikit-Learn package does not currently allow for categorical inputs, thus variables were converted during the previously discussed pre-processing stage.

Structure of a DT: to classify an instance, we need to navigate the tree from the root node to the leaf nodes. Each step is dictated by the values we find in the instance.



Root node: the first node which trains the data set. It is located at the very top of the tree.

Internal Node*:* This is the point where subgroup is split to a new sub-group or leaf node. Also referred to as a decision node because this is where observations are split further based on the best attribute of the sub-group. It represents an attribute to be tested. An edge connects each internal node to either: another internal node, a root node and/or a Leaf Node.

Leaf Node*:* Final node from any internal node, this holds the decision i.e. represent the predicted class of the instance. It is located at the bottom of each branch of the tree.

Figure AAA: demonstrating the structure of a Decision Tree.
Source: https://medium.com/geekculture/decision-trees-with-cart-algorithm-7e179acee8ff

CART Algorithm: Classification and regression trees (CARTs) use recursive partitions to divide data into increasingly homogenous groups. Splits in CARTs are chosen based by finding the split that results in the smallest value of some node impurity measure.The split with the smallest node impurity is chosen. Then, the process is repeated at the new child nodes. If node impurity fails to decrease at the next step, the child node becomes a leaf node. By default, CART uses Gini index as splitting criteria. (Breiman, 1984)

Gini's Impurity Index: is measure of inequality. Its value ranges between 0 and 1, where a value of 0 indicates that the samples are perfectly homogeneous. On the other hand, a Gini coefficient with a value of 1 shows maximum inequality between the elements. It is sum of the square of the probabilities for each class and is calculated via:

$$Gini\,index = 1 - \sum_{i=1}^{n} p_i^2$$

Where i is the number of classes and p is the probability of the classes. An alternative measure of impurity that was considered during the tuning of the hyper-parameters is **Entropy,** which measures the variance within the dataset after each split. As such, the Information Gain for the model is measured by how much entropy has been removed from the system, thus showing how well a certain split is at classifying the data. The formula for entropy is presented below:

$$E = -\sum_{i}^{C} p_i \log_2 p_i$$

Where p_i is the probability of randomly picking an element of class i.

Implementation:

The model was first initially trained using the default parameters.

```
default_dt = DecisionTreeClassifier()
default_dt.fit(x_train, y_train)
```

The default model was then used to predict the target variable in the test dataset.

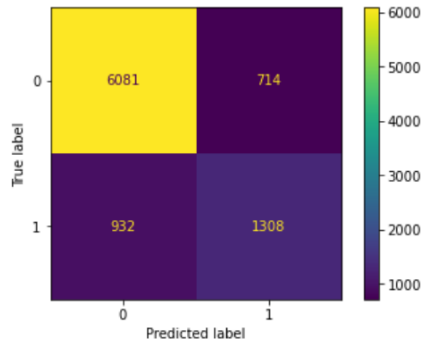
```
y_pred_default = default_dt.predict(x_test)
```

A confusion matrix and classification report were then produced to analyse the performance of the algorithm on the test dataset.

```
plot_confusion_matrix(default_dt, x_test, y_test)
plt.figure(figsize=(0.5,0.5))
plt.show()

print(classification_report(y_test,y_pred_default))
```

This produced the following output:



	precision	recall	f1-score	support
0	0.87	0.89	0.88	6795
1	0.65	0.59	0.61	2240
accuracy			0.82	9035
macro avg	0.76	0.74	0.75	9035
weighted avg	0.81	0.82	0.81	9035

The classification report show performance that is relatively low, especially with the classification of the minority dataset. The Confusion Matrix shows that the model correctly classified 6081 (True Negative) of out 6795 cases of individuals making <=$50 a year, which left 714 observations incorrectly classified (Type I error). Similarly, the minority class was correctly classified for 1308 observations, whilst 932 instances were incorrect (False Negative: Type II error). In addition, the precision for the minority class shows that only 65% of cases predicted to make over $50k (True Positive) were classified correctly, while recall for the class shows that only 59% of individuals actually making over $50K have been correctly classified. As such the F1 score of the minority class is relatively low and it is necessary to tune the hyper-parameters in order to improve the performance of the model and minimise overfitting.

Hyper-parameter Tuning and Cross Validation:

- **criterion**: defines the function to measure the quality of split. Sklearn includes the “Gini” criteria for Gini Index & “Entropy” for Information Gain. By default it is given a value of ‘Gini’.
- **splitter**: defines the strategy to choose the split at each node. It supports “best” value to choose the best split & “random” to choose the best random split. By default, it is “best”.
- **max_features** : defines the number of features considered when looking for the best split. It can take values of integers, floats, strings & None values.
- **max_depth** : represents the maximum depth of the tree. It may take any integer value or None. If the selected value is None, the nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. By default, a value of “None” is assumed.
- **min_samples_split**: informs the model of the minimum number of samples required to split an internal node. By default, it assumes a value of 2.
- **min_samples_leaf**: the minimum number of samples required for a node to be considered a leaf. By default, it is given a value of 1.
- **max_leaf_nodes**: defines the maximum number of possible leaf nodes. It may take any integer value or None. If its value is None the model may have an unlimited number of leaf nodes, until a different limiting condition is met. The default value for this parameter is “None”.
- **min_impurity_split**: defines the threshold at which the tree is prevented from growing. A node will split if its impurity is above the threshold, otherwise the node will become a leaf.

The hyper parameters have been tuned and cross-validated via:

```
f1 = make_scorer(f1_score , pos_label = 1, average='binary')
param_grid = {
    'max_depth': range(5,10),
    'min_samples_leaf': range(1,10),
    'min_samples_split': range(1,10),
    'criterion': ["entropy", "gini"],
    'splitter': ["best", "random"]
}
n_folds = 5
dtree = DecisionTreeClassifier()
grid_search = GridSearchCV(estimator = dtree, param_grid = param_grid,
                           cv = n_folds, verbose = 1, scoring=f1)
grid_search.fit(x_train,y_train)
```

GridSearchCV is used to iterates through predefined grid of HyperParameters to find the optimal parameters for the estimator (model) on the training set. The make_scorer function is defined in order to indicate that the algorithm should focus on maximising the F1 score for the minority class (class 1: >50k). The n_fold was given a value of 5 for the purposes of cross validation. min_sample_leaf and max_depth were tuned to control the size of the tree and prevent the model from overfitting to the training dataset.

The optimal parameters for the model were noted through:

```
print("Best F-1 Score: ", grid_search.best_score_)
print("It was achieved using the following hyper-parameters: ")
print(grid_search.best_params_)
```

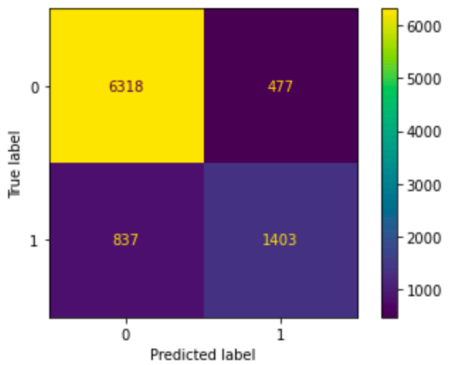
```
Best F-1 Score:  0.6722166324782753
It was achieved using the following hyper-parameters:
{'criterion': 'entropy', 'max_depth': 8, 'min_samples_leaf': 3, 'min_samples_split': 9, 'splitter': 'best'}
```

As such, a new model was created using the outlined hyper-parameters and trained on the training dataset.

```
tuned_model = DecisionTreeClassifier(criterion = "entropy",
                                     max_depth=8,
                                   min_samples_leaf=3,
                                   min_samples_split=9,
                                     splitter="best"
                                     )
tuned_model.fit(x_train, y_train)
```

The tuned model was then used to predict the target variable in the test dataset, from which a Confusion Matrix and Classification report have been produced.

```
y_pred_test = tuned_model.predict(x_test)
y_pred_test = tuned_model.predict(x_test)
print(classification_report(y_test, y_pred_test))
plot_confusion_matrix(tuned_model, x_test, y_test)
plt.figure(figsize=(0.5,0.5))
plt.show()
```



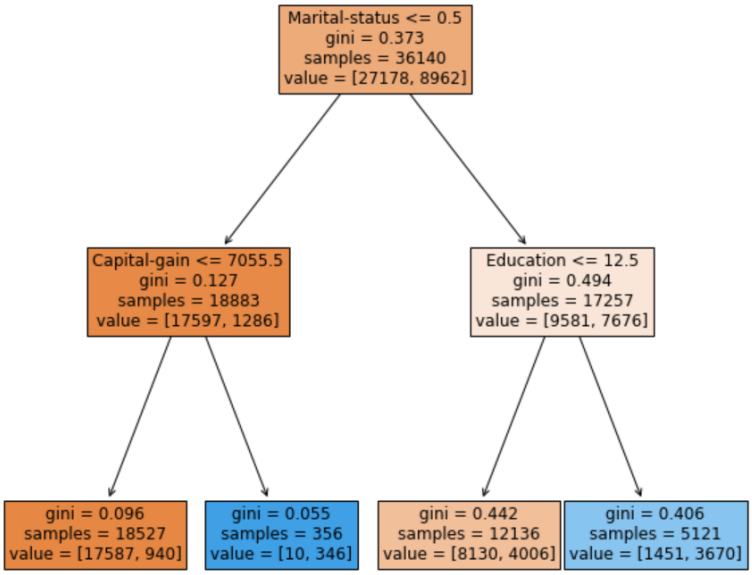
	precision	recall	f1-score	support
0	0.88	0.93	0.91	6795
1	0.75	0.63	0.68	2240
accuracy			0.85	9035
macro avg	0.81	0.78	0.79	9035
weighted avg	0.85	0.85	0.85	9035

The tuned model shows an improvement in the classification of the minority class, with the precision increasing from 65% to 75%. Similarly the F1 score for the minority class has improved from 0.61 to 0.68, in addition the F1 score also improved for the majority class (from 0.88 to 0.91). The confusion matrix shows that the model correctly classifies the majority class 6318 times (True Negative), whilst 477 observations are incorrectly classified (False Positive: Type I error). The minority class is classified correctly 1403 times (True

Positive) and incorrectly classifies the class for 837 observations, showing an improvement when compared to the default model. As the confusion matrix and classification report indicate, the tuned model performs better than the one with the default parameters showing evidence of overfitting of the default model to the training dataset.

Finally, the performance of the model was further evaluated using the previously outlined weighted report function, which produced the following results:

Weighted Precision: 0.762934310944565
Weighted Recall: 0.6255359907647139
Weighted F1 Score: 0.6874369145635915



For visualisation purposes, the decision tree above was created using a max_depth = 2 and is used for visualisation purposes demonstrating how the tree is split. The rest of the hyper-parameters are kept the same as in the tuned model, however should the max_depth = 10 as suggested by GridSearchCV(), the tree becomes too complex to visualise.

6. Instance Based Learning :

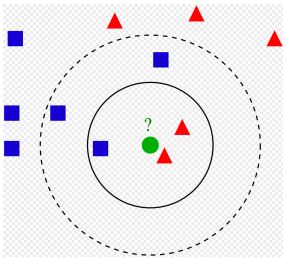
This family of algorithms are often referred to as ‘lazy’ as the computation is only performed once a new instance has been observed. The training data is stored in memory and the algorithms locate the nearest training examples and leverage them to estimate the class of a new instance. Instance Based Learning (IBL) algorithms are able to take discrete and continuous inputs to predict discrete/continuous target functions. The three most common algorithms in the IBL family are :

Collaborative Filtering: primarily used for recommendation systems, where recommendations are made through the analysis of historical data. Historic user preferences are leveraged in order to estimate the user preferences of individuals that are similar to them. As such, the algorithm assumes that individuals who have agreed in the past will continue to do so in the future.

Support Vector Machines: is a non-probabilistic binary classifier that determines the class of a new instance by viewing data through a p-dimentional vector space. Hyperplanes are used as decision boundaries to classify datapoints in a space, where as support vectors are data points that influence the orientation and positioning of a hyperplane. As such, data points on either side on a p-dimensional hyperplane are identified as belonging to different classes.

K-Nearest Neighbours: KNN predicts the class for a new observation through calculating the distance between all the training data points and as such determining the probability of the instance belonging to a class of K training data. The steps in classifying an unseen observation are as follows:

1. Choose the number of K neighbours (N).
2. Determine the distance of K number of neighbours.
3. Select the nearest N neighbours based on the calculation in step II and calculate the number of data points in each class.
4. The most common class out of the N closest neighbours is given to the new instance.



The above figure shows a visual representation of the KNN algorithm, where the number of neighbours is three and as such the three nearest neighbours to the new instance (circle) are selected. As the number of triangles (2) is greater than the number of squares (1), the new instance will be classified as a triangle.

There are three ways to calculate the distance for the KNN algorithm, the mathematical formulas for which are presented below.;

Manhattan Distance:

$$L^1(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sum_i |x_{1,i} - x_{2,i}| = ||x_1 - x_2||_1$$

Euclidean Distance:

$$L^2(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sqrt{\sum_i |x_{1,i} - x_{2,i}|^2} = ||x_1 - x_2||_2$$

Cosine Distance:

$$d_{cos}(\boldsymbol{x}_1, \boldsymbol{x}_2) = 1 - \frac{\boldsymbol{x}_1^{\mathrm{T}} \cdot \boldsymbol{x}_2}{||\boldsymbol{x}_1||||\boldsymbol{x}_2||} = 1 - \frac{\sum_i x_{1,i} x_{2,i}}{\sqrt{\sum_i x_{1,i}^2}\sqrt{\sum_i x_{2,i}^2}}$$

The K-Nearest Neighbours algorithm has been selected for the purposes of this report, due to the simplicity of its implementation through the sklearn package.

Advantages of KNN:

- Easy to implement, as learning is done at the time of making predictions
- New data can be added seamlessly, as no training is required prior to making predictions

Drawbacks of KNN:

- May struggle to work with large datasets, as the distance between each neighbour increases, it is difficult for the algorithm to calculate the distance across multiple dimensions.
- Sensitive to missing values and outliers
- Requires feature scaling (Standardisation)

Pre Processing:

As the algorithm requires feature scaling in order to best measure the distances between each data point, the dataset has been standardised using StandardScaler() via the following:

```
stds = StandardScaler()
stds.fit(x_train)
x_train = stds.transform(x_train)
x_test = stds.transform(x_test)
```

Implementation:

The model was first initially trained using the default parameters.

```
default_knn = KNeighborsClassifier()
default_knn.fit(x_train, y_train)
```

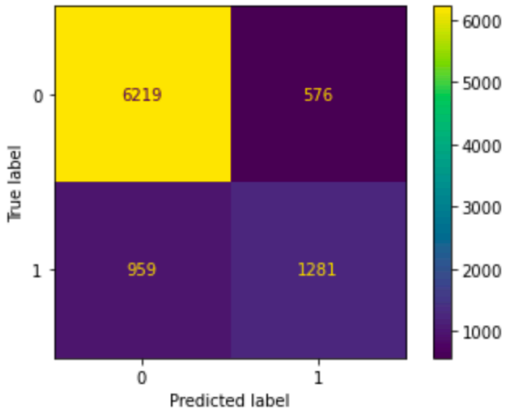
The default model was then used to predict the target variable in the test dataset.

```
y_pred_default = default_knn.predict(x_test)
```

A confusion matrix and classification report were then produced to analyse the performance of the algorithm on the test dataset.

```
plot_confusion_matrix(default_knn, x_test, y_test)
plt.show()
print(classification_report(y_test, y_pred_default))
```

This produced the following output:



	precision	recall	f1-score	support
0	0.87	0.92	0.89	6795
1	0.69	0.57	0.63	2240
accuracy			0.83	9035
macro avg	0.78	0.74	0.76	9035
weighted avg	0.82	0.83	0.82	9035

The Confusion Matrix shows that the model correctly classified 6219 (True Negative) of out 6795 cases of individuals making <=$50 a year, which left 576 observations incorrectly classified (Type I error). Similarly, the minority class was correctly classified for 1281 observations, whilst 959 classifications were incorrect (False Negative: Type II error). However, the precision for the minority class shows that only 69% of individuals making over $50k (True Positive) were predicted correctly, while recall for the class shows that only 63% of

cases predicted to make $50k by the model actually achieve this threshold. As such the F1 score of the minority class is rather low (0.68) and it is necessary to tune the hyper-parameters of the model in order to minimise overfitting to the training dataset.

Hyper Parameters:

Number of Neighbours: the number of k-neighbours that are considered when classifying a new instance. (Discussed above)

Weights: the weight function that is used for prediction.

Metric: the distance of measure that is used to calculate the nearest k-neighbours (the formulas for which are outlined above).

The hyper parameters have been tuned and cross-validated via:

```
f1 = make_scorer(f1_score, pos_label = 1, average = 'binary')
# create the parameter grid
param_grid = [{
    'weights': ["uniform", "distance"],
    'n_neighbors': list(range(10, 20, 5)),
    'metric': ['euclidean', 'manhattan', 'cosine'],
}]
n_folds = 5
# Instantiate the grid search model
knn_clf = KNeighborsClassifier()
grid_search = GridSearchCV(estimator = knn_clf, param_grid = param_grid,
                           cv = n_folds, verbose = 2, scoring = f1, n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(x_train, y_train)
```

GridSearchCV is used to iterates through predefined grid of HyperParameters to find the optimal parameters for the estimator (model) on the training set. The make_scorer function is defined in order to indicate that the algorithm should focus on maximising the f1 score for the minority class (class 1: >50k). The n_fold was given a value of 5 for the purposes of cross validation.

The optimal parameters for the model were noted through:

```
print("Best F-1 Score: ", grid_search.best_score_)
print("It was achieved using the following hyper-parameters: ")
print(grid_search.best_params_)
```

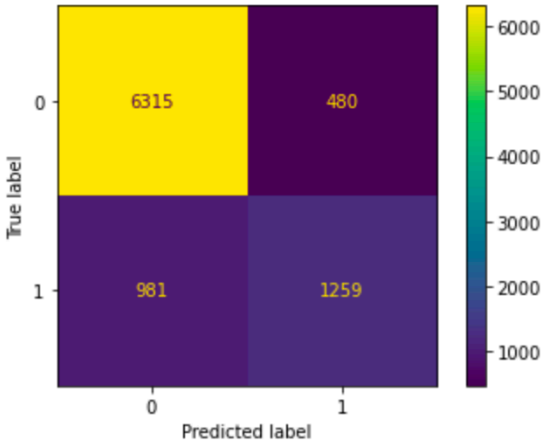
```
Best F-1 Score:  0.6773250270928044
It was achieved using the following hyper-parameters:
{'leaf_size': 25, 'metric': 'manhattan', 'n_neighbors': 15, 'weights': 'uniform'}
```

As such, a new model was created using the outlined hyper-parameters and trained on the training dataset.

```
tuned_model = KNeighborsClassifier(metric = 'manhattan',
                                   n_neighbors = 15,
                                   weights = 'uniform')
tuned_model.fit(x_train, y_train)
```

The tuned model was then used to predict the target variable in the test dataset, from which a Confusion Matrix and Classification report have been produced.

```
y_pred_test = tuned_model.predict(x_test)
print(classification_report(y_test, y_pred_test))
plot_confusion_matrix(tuned_model, x_test, y_test)
plt.show()
```



	precision	recall	f1-score	support
0	0.87	0.93	0.90	6795
1	0.72	0.56	0.63	2240
accuracy			0.84	9035
macro avg	0.79	0.75	0.76	9035
weighted avg	0.83	0.84	0.83	9035

The performance of the model post hyper parameter tuning is very similar to that of the default model. The precision for the minority class has improved slightly, from 0.69 to 0.72, similarly the Type I error has dropped from 576 observations to 480, yet the f1 score remained the same at 0.63. As such, it is reasonable to assume that there was little overfitting by the default model to the training dataset.

Finally, the performance of the model was further evaluated using the previously outlined weighted report function, which produced the following results:

Weighted Precision: 0.7593155509245307
Weighted Recall: 0.6204033061579024
Weighted F1 Score: 0.6828664778951304

7. Bayesian Learning:

Introduction:

The Bayes theorem makes can be leveraged to make predictions through conditional probability. Bayes rule (presented below) is used to calculate the probability of an event A occurring given that event B has occurred, where event B is termed as evidence.

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

As such, the models learn from experience and the probability of the hypothesis is updated as more events occur or more information becomes available. As the discussed task requires the classification of individuals based on their yearly income, a selection of Naive Bayesian Classifiers are implemented and compared in their performance based on their F-1 score of the minority class. NB classifiers are referred to as naive as they assume that the presence of a particular feature in a class is unrelated to the presence of any other feature and thus ignores the relationships between each feature.

Advantages of Naive Bayes Classifiers:

1. The algorithms are simple and computationally efficient, which makes them very fast at predicting the class on the test data set.
2. Performs better with small datasets when compared with other learning models (Al-Aidaroos et al, 2010).
3. The algorithm family performs better with categorical features, which is what the majority of the used dataset consists of.
4. Should the assumption of independence hold, NB classifiers perform better than models such as logistic regression (Sunil, 2017).

Disadvantages of Naive Bayes Classifiers:

1. Should the testing dataset have a feature that is not observed in the training dataset, smoothing will be required in order to make a prediction. This may hinder the performance of the algorithm.
2. The Bayes theorem of independence in features rarely holds true. In the reality it is very difficult to obtain a set of features that are truly independent. For example, in the discussed dataset it is very unlikely that that marital status and age are truly independent.

The dataset is consists of continuous, binary and discrete attributes. Thus, in order to create a NB model that performs best on the classification of the minority class, the following algorithms were trained and tested:

GaussianNB which assumes Gaussian (Normal) distribution of continuous features in a model.

MultinomialNB which assumes that a multinomial distribution to model feature occurrences.

ComplementNB: which leverages statistics of the complement of each class to calculate the weights of the model. It is best suited for imbalanced datasets.

BernoulliNB which assumes data follows the multivariate Bernoulli distribution.

CategorialNB which assumes each feature has its own categorical distribution.

Implementation:

Each of the above algorithms was implemented in the same way, by simply chaining the name of classifier when an instance of it was created. The code to train the algorithm is as follows:

```
model_gaussian = GaussianNB()
model_gaussian.fit(x_train, y_train)
```

The model is then used to predict the target attribute of the testing set via:

```
y_pred_gaussian = model_gaussian.predict(x_test)
```

A confusion matrix of the predicted test set was created for each algorithm through:

```
plot_confusion_matrix(model_gaussian, x_test, y_test)
plt.show()
```

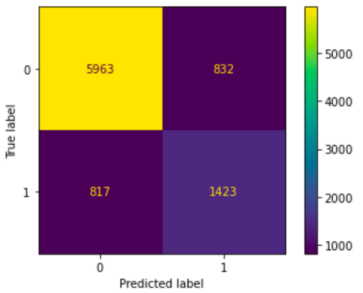
Finally, a classification report for each NB classifier was produced:

```
print(classification_report(y_test,y_pred_gaussian))
```

The output for each confusion matrix and corresponding classification report can be seen in FIGURE.

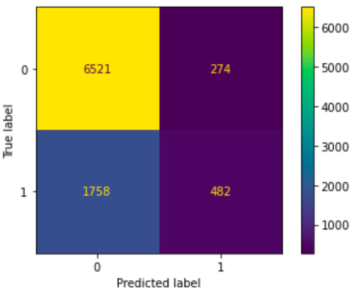
Confusion Matrix: *Classification Report:*

GaussianNB:



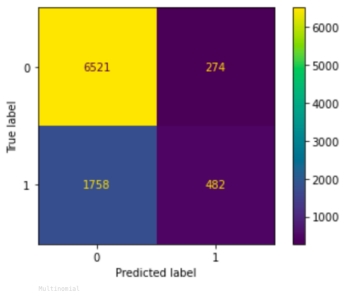
	precision	recall	f1-score	support
0	0.88	0.88	0.88	6795
1	0.63	0.64	0.63	2240
accuracy			0.82	9035
macro avg	0.76	0.76	0.76	9035
weighted avg	0.82	0.82	0.82	9035

MultinomialNB:



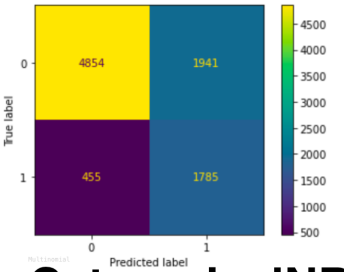
	precision	recall	f1-score	support
0	0.79	0.96	0.87	6795
1	0.64	0.22	0.32	2240
accuracy			0.78	9035
macro avg	0.71	0.59	0.59	9035
weighted avg	0.75	0.78	0.73	9035

ComplementNB:



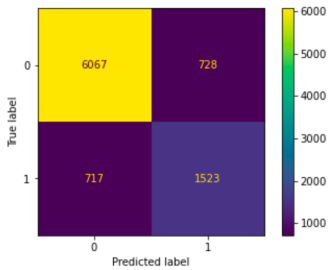
	precision	recall	f1-score	support
0	0.79	0.96	0.87	6795
1	0.64	0.22	0.32	2240
accuracy			0.78	9035
macro avg	0.71	0.59	0.59	9035
weighted avg	0.75	0.78	0.73	9035

BernouilliNB:



	precision	recall	f1-score	support
0	0.91	0.71	0.80	6795
1	0.48	0.80	0.60	2240
accuracy			0.73	9035
macro avg	0.70	0.76	0.70	9035
weighted avg	0.81	0.73	0.75	9035

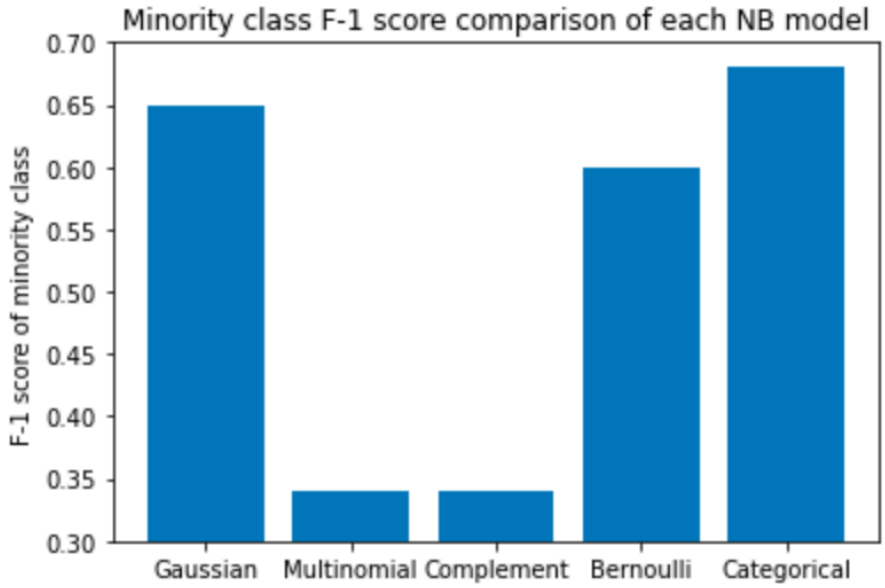
CategoricalNB:



	precision	recall	f1-score	support
0	0.89	0.89	0.89	6795
1	0.68	0.68	0.68	2240
accuracy			0.84	9035
macro avg	0.79	0.79	0.79	9035
weighted avg	0.84	0.84	0.84	9035

In order to determine the best classifier for the model, the f1 scores for the minority class of each algorithm were plotted as a bar chart using:

```
leaners = ['Gaussian','Multinomial','Complement','Bernoulli', 'Categorical']
f1_score = [0.65,0.34,0.34,0.6,0.68]
plt.bar(leaners, f1_score)
plt.title('Minority class F-1 score comparison of each NB model')
plt.ylabel('F-1 score of minority class')
plt.ylim(0.3,0.7)
plt.show()
```



The CategoricalNB algorithm had the best performance with classifying the minority class, with an F1 score of 0.68. As such, its performance was further evaluated using the previously outlined weighted report function, which produced the following results:

Weighted Precision: 0.6966371586663492
Weighted Recall: 0.6714933624648347
Weighted F1 Score: 0.6838342115254388

8. Neural Networks:

Neural Networks models were inspired by the neurones from the human body. There are several common types of neurone network models discussed below:

Single Layer Perceptron:

It is the most simple neural network as there is only one input layer and one output layer contained in the network. The input neurones are being weighted sum according to the weights in each edge and then an activation function is applied as a switch prior to the output layer. The limitation for such network is that it can only learn linearly separable problems through implementing OR, AND logic, however it cannot learn XOR logic where the truth table cannot be separated by a single line.

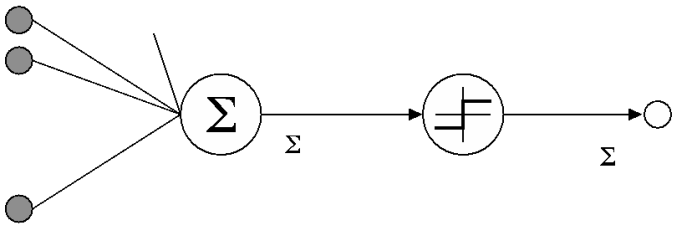


Figure demonstrating the structure of a Single Layer Perceptron.
Source: https://www.tutorialspoint.com/tensorflow/tensorflow_single_layer_perceptron.htm

Multi-Layer Perceptron:

The network is composed of one input layer, multiple hidden layers and one output layer. The activation function adds some non-linearity to the model and the large number of hidden layers and neurones adds complexity to the model. As such, theoretically, the multi-layer perceptron has the ability to model any function. The Multi-Layer Perception is a type of Artificial neural network where all neurones in the previous layer of the network are fully connected to the next layer in the network. It uses feedforward propagation for inputs, where they are weighted and fed into the activation function through every layer in the network and the weights of each edge are updated through back propagation by applying gradient descent algorithm on the loss function.

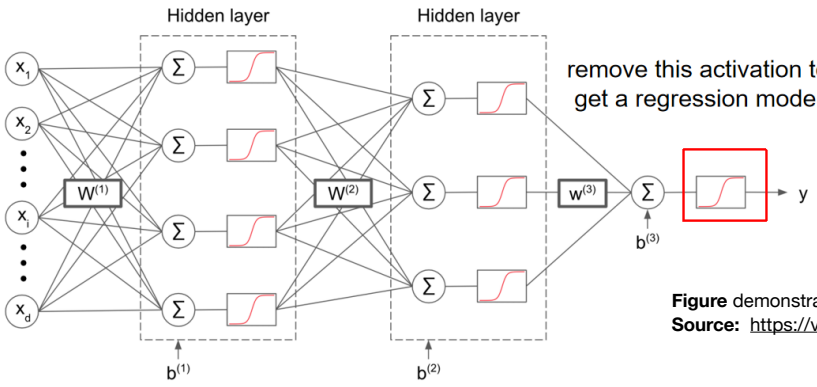


Figure demonstrating the structure of a Multi-Layer Perceptron.
Source: https://viso.ai/deep-learning/deep-neural-network-three-popular-types/

Recurrent neural network:

Whilst the Multi-Layer Perceptron can only predict the output for each independent input data and assumes the input data is not auto-correlated, the neurones in RNN will memorise not only the input of the current data but also previous data. This can be very useful for time-related data modelling.

Convolution neural network:

Convolution neural network includes some additional layers such as a convolution layer and pooling layer in order to perform feature selections prior to feeding the data into a fully-connected classification layer. This algorithm is widely used for for image recognition.

As the task requires a binary classification of a complex dataset, the **Multi-Layer Perceptron** (MLP) has been selected as the algorithm for the purposes of this report.

Pre-Processing Required:

If the relative importance of each input feature is unclear, it is important to scale the data in order to train the multi-layer perceptron neural network. Feature scaling typically describes two methods: normalisation and standardisation. For the purposes of this report, standardisation is used to speed up the convergence process of gradient descent and reduce the impact of the different scales for the input features on the model result (Yann A, 2012). This has been achieved through:

```
stds = StandardScaler()
stds.fit(x_train)
x_train = stds.transform(x_train)
x_test = stds.transform(x_test)
```

It is important to note, only the train dataset only the train set is used to find the mean and the variance, whilst the transformation is performed on both: the train and test sets.

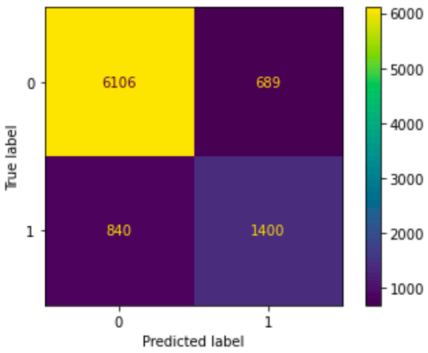
Implementation:

Initially, a rather complex MLP model was trained using rather complex network, then used to predict the target feature via:

```
complex_mlp = MLPClassifier(hidden_layer_sizes=(100,100,100,100),
activation="tanh", max_iter=400)
complex_mlp.fit(x_train, y_train)
y_pred_train = complex_mlp.predict(x_train)
```

From which, a confusion matrix and classification report was produced:

```
plot_confusion_matrix(complex_mlp, x_test, y_test)
plt.show()
print(classification_report(y_test, y_pred_test))
```



	precision	recall	f1-score	support
0	0.88	0.90	0.89	6795
1	0.67	0.62	0.65	2240
accuracy			0.83	9035
macro avg	0.77	0.76	0.77	9035
weighted avg	0.83	0.83	0.83	9035

The classification report show performance that is relatively low, especially with the classification of the minority dataset. The Confusion Matrix shows that the model correctly classified 6106 (True Negative) of out 6795 cases of individuals making <=$50 a year, which left 840 observations incorrectly classified (Type I error). Similarly, the minority class was correctly classified for 1400 observations, whilst 689 instances were incorrect (False Negative: Type II error). In addition, the precision for the minority class shows that only 67% of cases predicted to make over $50k (True Positive) were classified correctly, while recall for the class shows that only 62% of individuals actually making over $50K have been correctly classified. As such the F1 score of the minority class is relatively low (0.65) and it is necessary to tune the hyper-parameters in order to improve the performance of the model and ensure prevent overfitting.

Hyper-parameter tuning:

Number of Hidden Layers: For simple classification problems using the MLP, one hidden layer is believed to be sufficient, whilst two hidden layers may be selected when the data is discontinuous, which does not appear to be the case with our dataset (Gaurang, 2011). As such, only one hidden layer is considered in the grid search.

Number of Neurones in Each Hidden Layer: Determining the size of neurones is important when creating a MLP model, as it may cause under/over fitting of the model to the training dataset. Should the number of neurones in the hidden layer to be too low, the model may not be able to generate a complicated solution and could result in under-fitting and low performance. On the other hand, should the number of too many neurones can cause over-fitting of the model to the train dataset.

L2 Penalty: Regularisation in neural networks aims to avoid overfitting of the model. There are several techniques available to implement regularisation. For the purposes of this report, the “weight decay” (otherwise known as L2) regularisation method is used to reduce the value of all the weights by adding a regularisation term in the cost function via the following formula:

Where, C is the new cost function with L2 term and C0 is the cost function without L2. The value of L2 parameters are usually chosen on a logarithmic scale between 0 and 0.1 (Kuhn, 2018).

Learning Rate, Batch Size and Momentum: Stochastic gradient descent method is used to measure the gradient for the current iteration of the model and update the weights based on the loss function by using back propagation techniques. As the task requires a binary classification, a cross-entropy loss function is

$$C = C_0 + \frac{\alpha}{2n} \sum_w w^2$$

used. Batch size and momentum are both used to speed up the process of gradient descent algorithm and it is by default to be set as a mini-batch size of 200 and a momentum of 0.9.

Activation Function: For the binary classification task, the activation function for the output layer is chosen to be logistic. The following activation functions in the hidden layers were considered by RandomSearchCV: tanh, relu and logistic.

To find the optimal hyper parameters for the model, RandomSearchCV is used to find the iterate over a combination of hyper parameters of the MLP, which was validated through 5-fold cross validation. In addition, in order to maximise the performance at classification of the minority class, a custom scoring function was defined. This was achieved via:

```
param_grid = {
    "hidden_layer_sizes": [(16),(17),(18),(19),(20),(21),(22),(23),(24),(25)],
    "activation": ["tanh", "relu", "logistic"],
    "solver": ["sgd", "adam"],
    "alpha":[0.1, 0.01, 0.001, 0.0001],
    "learning_rate_init": np.arange(0.01, 0.1, 0.02)
}
f1 = make_scorer(f1_score , average='binary', pos_label = 1)
rs_cv = RandomizedSearchCV(mlp, param_grid, verbose=10, n_jobs=-1, cv=5, n_iter=200, scoring=f1)
rs_cv.fit(x_train,y_train)
```

```
print("Best F-1 Score: ", rs_cv.best_score_)
print("It was achieved using the following hyper-parameters: ")
print(rs_cv.best_params_)
```

```
Best F-1 Score:  0.6753326703454647
It was achieved using the following hyper-parameters:
{'solver': 'sgd', 'learning_rate_init': 0.08999999999999998, 'hidden_layer_sizes': 23, 'alpha': 0.1, 'activation': 'relu'}
```

As such, a new model was created using the outlined hyper-parameters and trained on the training dataset:

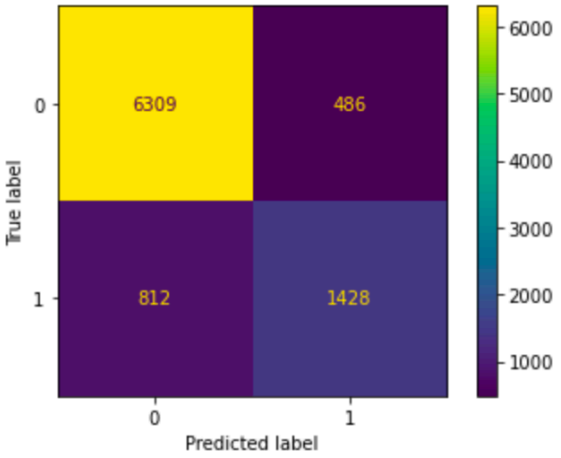
```
mlp_tuned = MLPClassifier(hidden_layer_sizes=(23)  ,solver='sgd', learning_rate_init=0.09,
alpha=0.1, activation='relu')
mlp_tuned.fit(x_train, y_train)
```

The tuned model was then used to predict the target feature via:

```
y_pred_train = mlp_tuned.predict(x_train)
```

Finally, a Confusion Matrix and Classification Report were created:

```
print(classification_report(y_test, y_pred_test))
plot_confusion_matrix(mlp_tuned, x_test, y_test)
plt.show()
```



	precision	recall	f1-score	support
0	0.89	0.93	0.91	6795
1	0.75	0.64	0.69	2240
accuracy			0.86	9035
macro avg	0.82	0.78	0.80	9035
weighted avg	0.85	0.86	0.85	9035

The tuned model shows an improvement in the classification of the minority class, with the precision increasing from 67% to 75%. Similarly the F1 score for the minority class has improved from 0.65 to 0.69, in addition the F1 score also improved for the majority class (from 0.89 to 0.91). The confusion matrix shows that the model correctly classifies the majority class 6309 times (True Negative), whilst 486 observations are incorrectly classified (False Positive: Type I error). The minority class is classified correctly 1428 times (True Positive) and incorrectly classifies the class for 812 observations, showing an improvement when compared to the complex model. As the confusion matrix and classification report indicate, the tuned model performs better than the one with the default parameters showing evidence of overfitting of the default model to the training dataset.

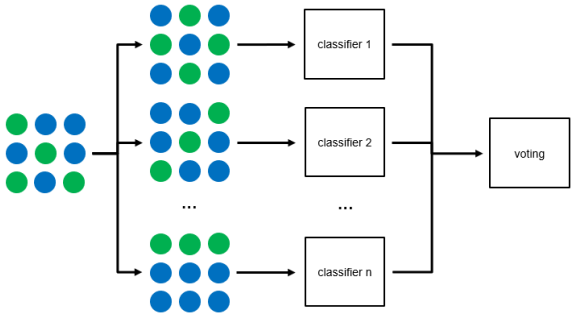
Finally, the performance of the model was further evaluated using the previously outlined weighted report function, which produced the following results:

Weighted Precision: 0.7579628641681798
Weighted Recall: 0.6417189072028622
Weighted F1 Score: 0.6950138393499392

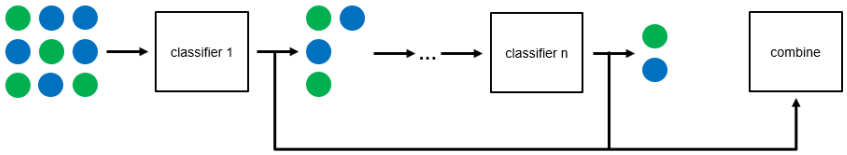
9. Model Ensemble:

A model ensemble combines the predictions of several learning algorithms to aggregate multiple predictions into one for the test dataset (unseen data). A single single model (base estimator) may over rely on on one or several features or is overly sensitive to particular feature scaling, meaning that its predictions are inaccurate. As such, the model ensemble creates a stronger model through the premise that even if a particular algorithm performs weakly, it is counterbalanced by the correct predictions made by the other models. Therefore a model ensemble was performed in order to create a more robust and reliable model. There are three main methods to create a model ensemble:

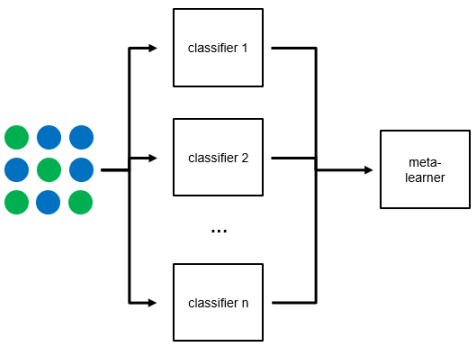
Bagging: is also known as a bootstrapping aggregation of multiple versions of a particular predicting model. The original training dataset is randomly separated with replacement (meaning a particular observation may appear multiple times in a a given subset) into multiple subsets of data (known as bags), following which the each of the subsets are used to train their respective models. As such multiple models are created and ensembled through a voting method. Bagging is typically performed on decision trees (through algorithms such as the Random Forest) in order to optimise the model fit by aggregating a number of decision trees that have been trained on different 'bags' of data.



Boosting: builds on the bagging model ensemble. Primarily, a subset (bag) of the training dataset is used for learning by the base model, the performance of which is then evaluated. Observations for which the base classier has made an incorrect prediction are then given more weight, which makes them more likely to appear in the next 'bag' of data and thus the following model places greater focus on such samples. The process is continued with each bag containing a weighted distribution of data based on the performance of the previous learner until the performance criteria has been met or the number of models limit has been reached.



Stacking: uses a combination of predictions made by base learners to make predictions, following which voting decides the predation made by the stacked model. The cross-verification method creates a robust model, which may improve the overall performance of the meta-model. However, the performance of a stacking model ensemble may be compromised if one, or several, base learners classify poorly.



There are two ways in which the stacking model ensemble votes on which prediction is given by the meta model. For each observation, 'hard voting' counts the number of classes given by the base learners and selects the most frequent class as the prediction, where as 'soft voting' classifies the input data based on the probabilities of predictions made by each classifier, giving more weight to classifiers with higher confidence (Kumar, 2020).

As the project requires the aggregation of the already trained based learners, the stacking model ensemble has been chosen as the model ensemble method for the purposes of this report. Furthermore, hard-voting was chosen as the voting method as each base learner classified each observation, rather than calculating the probability of each classification.

Implementation:

Initially, the trained models with tuned hyper-parameters for each learner have been imported. The CategoricalNB classifier was chosen as the algorithm from the Bayes family as it performed best at classifying the minority class.

```
#CategoricalNB
tuned_model_cnb = joblib.load('bayes.pkl')
# Decision Tree
tuned_model_dt = joblib.load('dt.pkl')
#MLP
with open('mlp.obj', 'rb') as f:
    tuned_model_mlp = pickle.load(f)
#KNN
tuned_model_knn = joblib.load('knn.pkl')
```

Following, the CategoricalNB and Decision Tree learners were used to make predictions on the test dataset:

```
y_pred_test_dt = tuned_model_dt.predict(x_test)
y_pred_test_cnb = tuned_model_cnb.predict(x_test)
```

As the Multi-Layer Perceptron and KNN classifiers required data normalisation (for reasons discussed in their respective sections), the test dataset was then normalised using a Standard Scaler. These learners were then used to make class predictions for the test dataset.

```
stds = StandardScaler()
stds.fit(x_train)
x_train = stds.transform(x_train)
x_test = stds.transform(x_test)
y_pred_test_mlp = tuned_model_mlp.predict(x_test)
y_pred_test_knn = tuned_model_knn.predict(x_test)
```

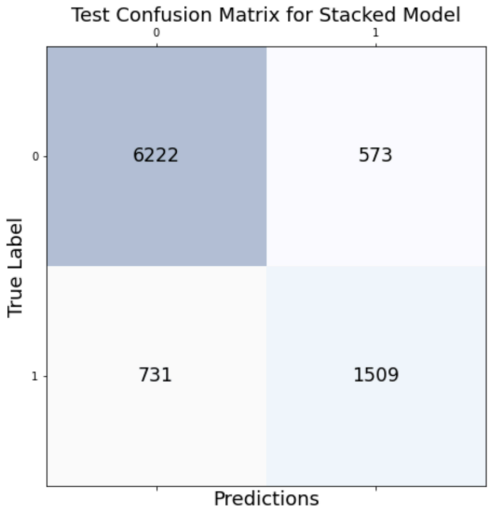
Following, the stacking model ensemble was conducted by selecting the most common predicted class among each of the four learners.

```
y_pred_test_stacking =[]#predicted result will be put into the list
for i in range(len(y_pred_test_dt)):
    final = []#totally 9035 final lists for adding predicted result (0 or 1) from dt, knn, cnb and
mlp
    final.append(y_pred_test_dt[i])#append each predicted result of decision tree
    final.append(y_pred_test_knn[i])#append each predicted result of k nearest neighbor
    final.append(y_pred_test_cnb[i])#append each predicted result of categoricalNB
    final.append(y_pred_test_mlp[i])#append each predicted result of neural networks
#getting the mode of predicted result of 4 algorithms, and add the mode value to be the predicted
result for hard voting
    if final.count(1) < final.count(0):
        y_pred_test_stacking.append(0)
    else:
        y_pred_test_stacking.append(1)
```

A classification report and confusion matrix for the stacked model was then created via:

```
y_pred_test_stacking = np.array(y_pred_test_stacking)
confusion_matrix = confusion_matrix(y_test,y_pred_test_stacking)
fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(confusion_matrix, cmap=plt.cm.Blues, alpha=0.3)
for i in range(confusion_matrix.shape[0]):
    for j in range(confusion_matrix.shape[1]):
        ax.text(x=j, y=i,s=confusion_matrix[i, j], va='center', ha='center', size='xx-large')
```

```
plt.xlabel('Predictions', fontsize=18)
plt.ylabel('True Label', fontsize=18)
plt.title('Test Confusion Matrix for Stacked Model', fontsize=18)
plt.show()
```



	precision	recall	f1-score	support
0	0.89	0.92	0.91	6795
1	0.72	0.67	0.70	2240
accuracy			0.86	9035
macro avg	0.81	0.79	0.80	9035
weighted avg	0.85	0.86	0.85	9035

The classification report shows relatively strong performance, especially with the classification of the majority dataset. The Confusion Matrix shows that the model correctly classified 6222 (True Negative) of out 6795 cases of individuals making <=$50 a year, which left 731 observations incorrectly classified (Type I error). Similarly, the minority class was correctly classified for 1509 observations, whilst 731 instances were incorrect (False Negative: Type II error). In addition, the precision for the minority class shows that 72% of cases predicted to make over $50k (True Positive) were classified correctly, while recall for the class shows that 67% of individuals actually making over $50K have been correctly classified. As such the F1 score for the minority class of the stacked model is 0.70, which is higher than than any of the individual base learners.

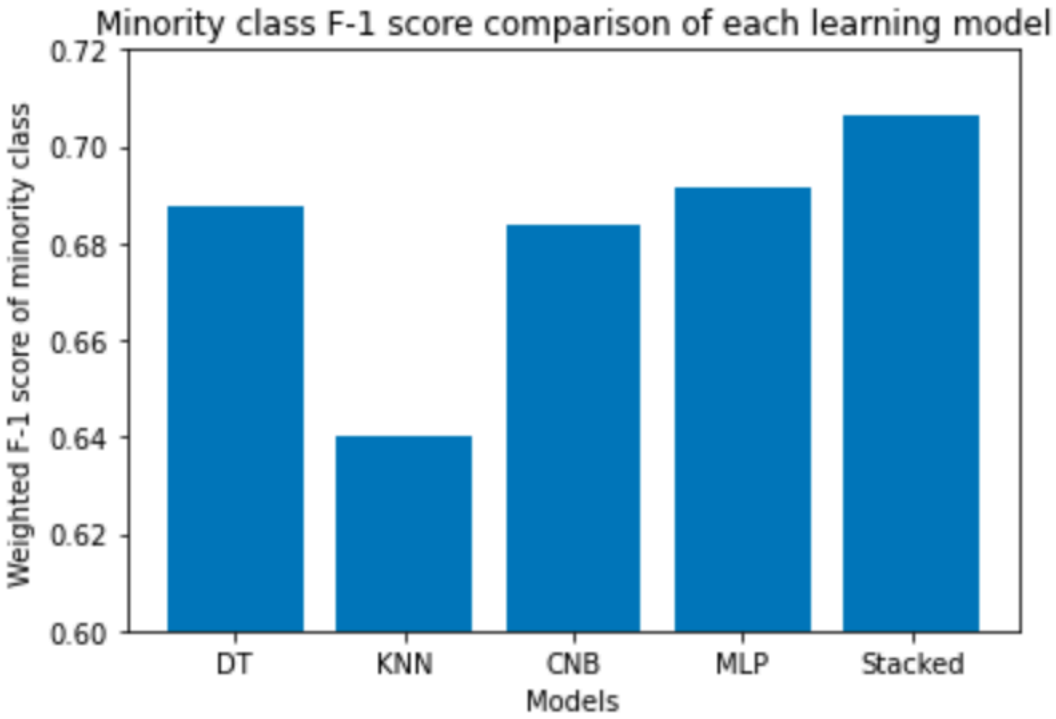
Finally, the performance of the model was further evaluated using the previously outlined weighted report function, which produced the following results:

Weighted Precision: 0.739504165579235
Weighted Recall: 0.6764855593441129
Weighted F1 Score: 0.7065925412929831

10. Conclusion

Overall, four learning algorithms (CART, KNN, MLP and CategoricalNB) have been trained and tested, following which they were ensembled into a stacked meta model. A bar chart of the weighted F1 score for each of these models has been created in order to visualise the comparison in their performance.

```
leaners = ['DT','KNN','CNB','MLP', 'Stacked']
weighted_f1_score = [f1_dt,f1_knn,f1_cnb,f1_mlp,f1_stacked]
plt.bar(leaners, weighted_f1_score)
plt.title('Minority class F-1 score comparison of each learning model')
plt.xlabel('Models')
plt.ylabel('Weighted F-1 score of minority class')
plt.ylim(0.6,0.72)
plt.show()
```



As shown by the figure above, the Stacked meta model has outperformed each base learner and thus is the most robust model produced for the purposes of this project. KNN had the poorest performance at classifying the target variable, potentially due to a large number of features in the dataset, which makes it difficult for the algorithm to calculate distances in a n-dimensional space. The remaining three learners performed relatively similar, with f1 scores of around 0.68. This may be an indicator that the pre-processing of the dataset was the limiting factor for the performance of the models. Should the task be investigated, it may be of value to alter the groupings performed and compare the performance of each algorithm after hot one encoding. It may also be beneficial to convert age into a categorical range variable in order to improve the performance of the CategoricalNB model.

It is important to note that this dataset has been collected over 15 years and as such, the value of $50k has grown between the first census collection and the last. As such, the number of individuals making over $50,000 is likely to be greater in the data collected most recently. Finally, it must be noted that individuals have appeared multiple times in the dataset, whether with exactly the same observations or otherwise, thus making the dataset uneven. Should further research be conducted it may be beneficial to analyse data exclusively from the most recent census in order to avoid the discussed biases.

11. Git Log

commit 721063844afffecc80bd3ac6a5ba9d023982e046
Author: miron2002 <mkiselev2002@icloud.com>
Date: Mon Apr 4 17:15:37 2022 +0100

    edited model ensemble

commit ed753d829ca0e8d9c3037b76d1376814ba00f7c2
Author: miron2002 <mkiselev2002@icloud.com>
Date: Sun Apr 3 21:17:32 2022 +0100

    fixing model ensemble

commit 0a77b04ebead7913d43cbd022670b91e0e338f5f
Author: miron2002 <mkiselev2002@icloud.com>
Date: Sun Apr 3 20:48:54 2022 +0100

    fix

commit f647230a57605b4850725b8098fab4c77d8f6e13
Author: miron2002 <mkiselev2002@icloud.com>
Date: Sun Apr 3 20:48:01 2022 +0100

    fixing knn and model ensemble from problems of standardisation

commit 33333852eacc1aebd9330061399e6db3f764e8e0
Author: Baoyan Han <ucesbh2@ucl.ac.uk>
Date: Sat Apr 2 22:35:49 2022 +0100

    change the hyper parameter and add the standardization to the test set

commit 57581164c9265c1acdae729e3222860d0b239269
Author: miron2002 <mkiselev2002@icloud.com>
Date: Sat Apr 2 20:46:19 2022 +0100

    fixing my mistake

commit 9b5f4519fb7a235fa2eeb9de6d4c94083462c890
Author: miron2002 <mkiselev2002@icloud.com>
Date: Sat Apr 2 20:42:51 2022 +0100

    added standardisation to model ensemble

commit 72740bd87ec017708f80f40ea07f0f88ebf04c61
Author: miron2002 <mkiselev2002@icloud.com>
Date: Sat Apr 2 20:34:45 2022 +0100

    standardised in knn

commit 887095fd61404fa1bdc579f542ec6ff74cc7f505
Author: ZichunWang98 <92408336+ZichunWang98@users.noreply.github.com>
Date: Thu Mar 31 16:52:31 2022 +0100

    Add files via upload

commit 79be160c9f2391ef90301b599fbdf2d1a689d929
Author: ZichunWang98 <92408336+ZichunWang98@users.noreply.github.com>
Date: Thu Mar 31 16:52:10 2022 +0100

    result update

commit a58517d03a2d18011cb5793c7ab2a83e6564fb7d
Author: ZichunWang98 <92408336+ZichunWang98@users.noreply.github.com>

Date: Thu Mar 31 16:51:55 2022 +0100

    result update

commit e38e44c6cb589f339927ded69f2b8048646d58e1
Author: miron2002 <mkiselev2002@icloud.com>
Date: Thu Mar 31 16:00:25 2022 +0100

    adding more labels

commit ba54b2748c8d8fdc374d5a040cc35254d26c6b36
Author: pratibha1708 <ucesatb@ucl.ac.uk>
Date: Wed Mar 30 23:45:44 2022 +0100

    improved the hyper-parameters

commit 28cf16ee9e6328460024a67fd8e8b4edebecc600
Author: Baoyan Han <ucesbh2@ucl.ac.uk>
Date: Sun Mar 27 23:27:16 2022 +0100

    delete training set report

commit e999c5ac50779849553245479d4159c82a2c36ba
Author: Baoyan Han <ucesbh2@ucl.ac.uk>
Date: Sun Mar 27 23:26:01 2022 +0100

    delete the training set report

commit a3de581acf0a67641b5eccc52a50375850cb9c12
Author: pratibha1708 <ucesatb@ucl.ac.uk>
Date: Sun Mar 27 23:22:56 2022 +0100

    changed parameters and exported file

commit af660d291edf2bd29663d5a2f0fb69218907c503
Author: ZichunWang98 <92408336+ZichunWang98@users.noreply.github.com>
Date: Sun Mar 27 20:59:48 2022 +0100

    change on decision tree part

commit 4787380bb3d6e5cc933d4a4e2330ded8a331a09b
Author: pratibha1708 <ucesatb@ucl.ac.uk>
Date: Sun Mar 27 20:17:43 2022 +0100

    improved the hpyer-parameters

commit 79e7e07e67caa2f8d0702eb203b6c3f2b6512992
Author: ucescg5 <93675678+ucescg5@users.noreply.github.com>
Date: Sun Mar 27 16:28:15 2022 +0100

    Update neuralNetwork.ipynb

commit 9447af76a1f12e6cecf5b95e80e7d12adfb8cd9d
Author: pratibha1708 <ucesatb@ucl.ac.uk>
Date: Sun Mar 27 15:19:49 2022 +0100

    added headings to the graphs in eda

commit a247e51662cb0bc3f4a3f72a4aa59447998b7b8b
Author: miron2002 <mkiselev2002@icloud.com>
Date: Sun Mar 27 00:37:45 2022 +0000

     adding a graph to bayes

commit acecb2b26495262285989246ee24611b190ed432

Author: pratibha1708 <ucesatb@ucl.ac.uk>
Date: Sat Mar 26 19:32:46 2022 +0000

    Final editing to the notebook

commit 92eae4562ab6ff3e5e1c0922156d2080e42c7173
Author: pratibha1708 <ucesatb@ucl.ac.uk>
Date: Sat Mar 26 19:28:24 2022 +0000

    final editing of the overall notebook

commit 3626da30acb07299fac26b60bcff28a20cff6e53
Author: miron2002 <mkiselev2002@icloud.com>
Date: Sat Mar 26 16:18:02 2022 +0000

    minor changes

commit bc4c14dee85e93576a6a8cb53dfa8cd1f3b79416
Author: miron2002 <mkiselev2002@icloud.com>
Date: Sat Mar 26 16:06:42 2022 +0000

    small changes

commit 63c1ab863b1f668a8c86223a2854e7b018f98fca
Author: miron2002 <mkiselev2002@icloud.com>
Date: Sat Mar 26 15:12:49 2022 +0000

    removing unused imports

commit de154a5fbf05123004b4bfae0521360ed150b2d9
Author: miron2002 <mkiselev2002@icloud.com>
Date: Sat Mar 26 15:06:29 2022 +0000

    additonal changes

commit 9455772cf15e157a26d4af156cc67a6a31a0dcaf
Author: miron2002 <mkiselev2002@icloud.com>
Date: Sat Mar 26 15:06:04 2022 +0000

    finalised model ensemble

commit ff3846bf922a1bbe901e2c7225d94d945ebd8ba9
Author: pratibha1708 <ucesatb@ucl.ac.uk>
Date: Sat Mar 26 14:08:08 2022 +0000

     added bivariate analysis for EDA using boxplots

commit 590af6a568b9e8dc774c65340b8e276da02114f7
Author: ZichunWang98 <92408336+ZichunWang98@users.noreply.github.com>
Date: Sat Mar 26 11:22:55 2022 +0000

    modified model ensemble for visualisation

commit a801055439af671b94f418db21aa530f6b57b168
Author: ZichunWang98 <92408336+ZichunWang98@users.noreply.github.com>
Date: Sat Mar 26 03:19:02 2022 +0000

    edited manual hard voting model ensemble

commit 8081949db4c0338d928e863de590a55767ca82b0
Author: miron2002 <mkiselev2002@icloud.com>
Date: Fri Mar 25 17:40:59 2022 +0000

    exporting models and fixing knn

commit 94f77c0ac38e397696ab067fcac1380fa4936f5c
Author: pratibha1708 <ucesatb@ucl.ac.uk>
Date: Fri Mar 25 17:21:22 2022 +0000

added code to export file for stacking(model essemble)

commit 0b6dc7093280e3ea648f721b144098de2610d271
Author: ucescg5 <93675678+ucescg5@users.noreply.github.com>
Date: Fri Mar 25 16:54:34 2022 +0000

Add files via upload

commit 2523bfd3c46d4993835650f56e389f3557ea018e
Author: Gong Chen <ucescg5@ucl.ac.uk>
Date: Fri Mar 25 15:52:56 2022 +0000

adding export mlp object to file

commit 7a3a933c9bdb531def78dffab935591faa9d77d3
Author: ucescg5 <93675678+ucescg5@users.noreply.github.com>
Date: Fri Mar 25 14:35:54 2022 +0000

Rename nueral.ipynb to neuralNetwork.ipynb

commit bb0ebae588522f82a34f26822f66fcdd1bad935e
Author: Gong Chen <ucescg5@ucl.ac.uk>
Date: Fri Mar 25 14:34:33 2022 +0000

expand parameter grid

commit bc337b851c4f42314a034e3802f226ef7b1d60f2
Author: miron2002 <mkiselev2002@icloud.com>
Date: Thu Mar 24 20:33:24 2022 +0000

added to ensemble

commit a7a028d3335c4d338660ff391576190e2bb57fd9
Author: ZichunWang98 <92408336+ZichunWang98@users.noreply.github.com>
Date: Thu Mar 24 20:05:30 2022 +0000

modified model ensemble

commit 86d7151f43ecf849b6e75d6d1601b751bf6343e7
Author: pratibha1708 <ucesatb@ucl.ac.uk>
Date: Thu Mar 24 19:47:58 2022 +0000

prioritise minor class in gridsearch and used final weights

commit 85b4d33fed29fe701878073b4a87344cf2309eb6
Author: ucesbh2 <93603102+honboyin@users.noreply.github.com>
Date: Thu Mar 24 19:22:17 2022 +0000

kNN

commit cdc358e35b8242d1db4b9a9d8aa6fb32f7dbdbb0
Author: Gong Chen <ucescg5@ucl.ac.uk>
Date: Thu Mar 24 19:19:26 2022 +0000

modification on neural network

commit 64fd5e093c3511097c23d78b04f0a2401ca378c1
Author: miron2002 <mkiselev2002@icloud.com>
Date: Thu Mar 24 19:12:33 2022 +0000

cleaning mess

commit 69b85fead05d726da27634ea1abd93d1d3789f30
Author: ucesbh2 <93603102+honboyin@users.noreply.github.com>
Date: Thu Mar 24 18:50:31 2022 +0000

    kNN

commit f14f82683fde0c1f3d7da8edf335ed55387f0df3
Author: ucesbh2 <93603102+honboyin@users.noreply.github.com>
Date: Thu Mar 24 18:50:07 2022 +0000

    kNN

commit 5dfd86a97731358fb1a1167e45608641ad83d2dd
Author: ucesbh2 <93603102+honboyin@users.noreply.github.com>
Date: Thu Mar 24 18:47:41 2022 +0000

    kNN

commit 2ecc23671bd4bafaee5389f1f045b5b2717696a1
Author: ZichunWang98 <92408336+ZichunWang98@users.noreply.github.com>
Date: Thu Mar 24 15:43:24 2022 +0000

    updated stacking model

    comparing stacking with and without MLPClassifier

commit 0cf3b0eaa004ac1f8b0b2d1da77ab498099d9afd
Author: ZichunWang98 <92408336+ZichunWang98@users.noreply.github.com>
Date: Thu Mar 24 15:06:09 2022 +0000

    Add files via upload

    model ensemble included bayes

commit e574c55298bae0179bd821f1c8d56ce1250668ce
Merge: 4216d45 42ffa37
Author: mkiselev2002 <71271836+mkiselev2002@users.noreply.github.com>
Date: Mon Mar 21 22:03:30 2022 +0000

    Merge pull request #2 from CEGE0004/miron

    finished bayes

commit 42ffa377b67288232465342a2977c0962f009588
Author: miron2002 <mkiselev2002@icloud.com>
Date: Mon Mar 21 21:52:51 2022 +0000

    finished bayes

commit 4216d455e9c8d04969cb3eda7bd3effe1f1213f6
Author: ZichunWang98 <92408336+ZichunWang98@users.noreply.github.com>
Date: Mon Mar 21 21:04:25 2022 +0000

    Add files via upload

    model ensemble added final weights

commit f5748ee93e482bf2b80924228c52e322b3577579
Author: ucesbh2 <93603102+honboyin@users.noreply.github.com>
Date: Mon Mar 21 21:01:34 2022 +0000

    kNN updated with final weight added

commit 01af6c5460df3d91eed3a6740ccc573791adfdd9
Author: miron2002 <mkiselev2002@icloud.com>
Date: Mon Mar 21 18:33:10 2022 +0000

    including function to calculate final weight

commit 1a026e72977a9b3d4bc3b4af3e272d8f99f49b73
Author: ucesbh2 <93603102+honboyin@users.noreply.github.com>
Date: Sat Mar 19 16:32:27 2022 +0000

    kNN updated with confusion matrix changed

commit 4416702db1924518b93d90b2d80e7a9e709fd1b3
Author: miron2002 <mkiselev2002@icloud.com>
Date: Sat Mar 19 16:17:38 2022 +0000

    Edited Confusion Matrix

commit e458aa2930b6be07399c0fda9ebe8c5fb7107164
Author: ZichunWang98 <92408336+ZichunWang98@users.noreply.github.com>
Date: Sat Mar 19 13:07:11 2022 +0000

    Delete model_ensemble1103.ipynb

commit b40594dcc00bd9444d10644d048313448b7f22e8
Author: ZichunWang98 <92408336+ZichunWang98@users.noreply.github.com>
Date: Sat Mar 19 13:06:40 2022 +0000

    model ensemble 0319

commit e10123028ff41077255c7d8151830ccaa7c1553e
Author: Gong Chen <ucescg5@ucl.ac.uk>
Date: Fri Mar 18 18:07:58 2022 +0000

    modify on neural

commit 27d4733436f4687bc812d283dc153a3d449e9a89
Author: Gong Chen <ucescg5@ucl.ac.uk>
Date: Fri Mar 18 17:38:50 2022 +0000

    modify neural

commit 371a5bcfd09708782fb03d8c46f63b22a23d4986
Author: Gong Chen <ucescg5@ucl.ac.uk>
Date: Fri Mar 18 14:50:27 2022 +0000

    updated neual network

commit 99254130fcb3bf7234bb23c1c51d9ec8864d59ee
Author: ucesbh2 <93603102+honboyin@users.noreply.github.com>
Date: Thu Mar 17 09:01:51 2022 +0000

    kNN

    kNN updated

commit 8fd8c5edb234a295195831ea445bcac415607b90
Author: miron2002 <mkiselev2002@icloud.com>
Date: Tue Mar 15 22:31:33 2022 +0000

    Added bayes learning

commit 23be14bbb6c9b4713745a7c00cccbc51dd4cc8ab
Author: miron2002 <mkiselev2002@icloud.com>

Date: Tue Mar 15 18:29:22 2022 +0000

    blueprint for other notebooks in decision tree notebook

commit 171e7dd8a9962b9afb5aa4cdd5c3a284b07d0f52
Merge: c3fd1ad 4565dab
Author: miron2002 <mkiselev2002@icloud.com>
Date: Tue Mar 15 16:32:56 2022 +0000

    preprocessing

commit c3fd1ad65370f95f89c5bf274527beae6f2d810f
Author: miron2002 <mkiselev2002@icloud.com>
Date: Tue Mar 15 16:29:53 2022 +0000

    preprocessing

commit 4565dab45b4ab7e97cdc3de585f4194626777021
Author: ucesbh2 <93603102+honboyin@users.noreply.github.com>
Date: Mon Mar 14 16:32:55 2022 +0000

    instance-based part

    instance-based part uploaded

commit 1a3ca0d2f96009cc4b64750cc719f4bf5ae93337
Merge: 9c17228 3139e65
Author: mkiselev2002 <71271836+mkiselev2002@users.noreply.github.com>
Date: Mon Mar 14 16:05:36 2022 +0000

    Merge pull request #1 from CEGE0004/DT

    Decision Tree Notebook

commit 9c172284ddcff752c08affadea8cb2859aaa452c
Author: miron2002 <mkiselev2002@icloud.com>
Date: Mon Mar 14 16:00:26 2022 +0000

    neural networks change

commit 58104490d20e35f83984dee81de2242466994193
Author: Gong Chen <ucescg5@ucl.ac.uk>
Date: Mon Mar 14 14:55:07 2022 +0000

    neual netwrok

commit 5315e1abd1d084e7f4f793bc801ce1118c88db71
Author: ZichunWang98 <92408336+ZichunWang98@users.noreply.github.com>
Date: Mon Mar 14 14:41:58 2022 +0000

    Add files via upload

commit 513dbde7fef8c04b4f3e18d60ad0980e86be39fd
Author: miron2002 <mkiselev2002@icloud.com>
Date: Mon Mar 14 14:38:41 2022 +0000

    init

commit 3139e65336246412e5a09a12157ab25ccdc4e9d9
Author: Pratibha Patel <92760098+pratibha1708@users.noreply.github.com>
Date: Fri Mar 11 17:51:02 2022 +0000

    Decision Tree Notebook

commit 07485ba8d913692fb3bcf07f880058ae127a73f1
Author: github-classroom[bot] <66690702+github-classroom[bot]@users.noreply.github.com>
Date: Mon Jan 24 14:16:38 2022 +0000

    Initial commit

12. Bibliography:

1.A. Ajanki (2007) ‘Example of k-nearest neighbour classification’. Available at: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#/media/File:KnnClassification.svg

2.K. M. Al-Aidaroos, A. A. Bakar and Z. Othman, "Naïve bayes variants in classification learning," *2010 International Conference on Information Retrieval & Knowledge Management (CAMP)*, 2010, pp. 276-281, doi: 10.1109/INFRKM.2010.5466902.

3.A. Kumar (2020) ‘Hard vs Soft Voting Classifier Python Example’ Data Analytics. Available at: https://vitalflux.com/hard-vs-soft-voting-classifier-python-example/

4. Bor, J., Cohen, G. and Galea, S., 2017. Population health in an era of rising income inequality: USA, 1980–2015. *The Lancet*, 389(10077), pp.1475-1490.

5.Deepankar (2021) ‘Decision Tree with CART Algorithm’. Available at: https://medium.com/geekculture/decision-trees-with-cart-algorithm-7e179acee8ff

6.F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and É. Duchesnay (2011) ‘Scikit-learn: Machine Learning in Python’, JMLR 12, pp. 2825-2830. Available at : https://scikit-learn.org/stable/about.html#citing-scikit-learn

7.Joshua S. Richman (2011) ‘Chapter Thirteen - Multivariate Neighborhood Sample Entropy: A Method for Data Reduction and Prediction of Complex Data’ *Methods in Enzymology, Volume 487, pp.397-408. Available at: https://doi.org/10.1016/B978-0-12-381270-4.00013-5*

8.Panchal, Gaurang, et al (2011) ‘Behaviour analysis of multilayer perceptrons with multiple hidden neurons and hidden layers." *International Journal of Computer Theory and Engineering 3.2, pp.332-337* -------From neural network

9.LeCun, Yann A., et al (2012) ‘Efficient backprop Springer’ *Neural networks: Tricks of the trade*, *9-48*

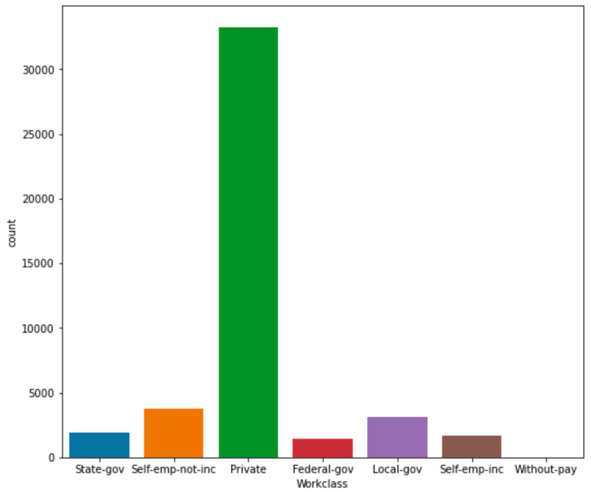
10.Max Kuhn (2013, 2018) Applied Predictive Modelling 1st ed, Corr, 2nd printing Edition

11.Nwakaroko, 2015, https://www.oecd.org/els/soc/cope-divide-europe-2017-background-report.pdf

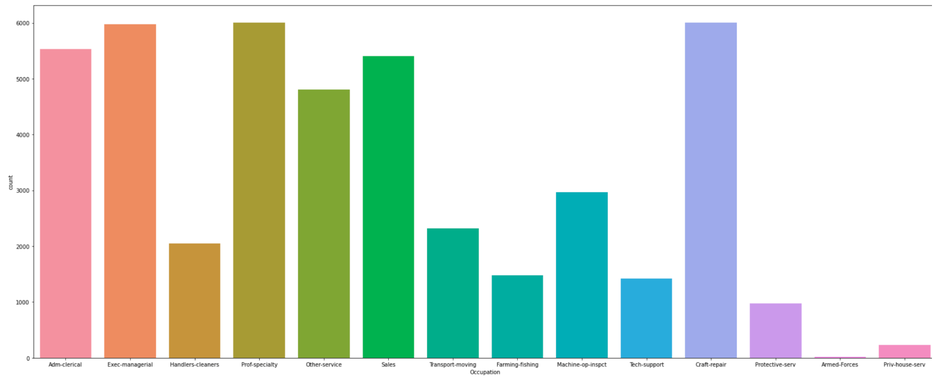
12. Suntil, 2017. *Learn Naive Bayes Algorithm | Naive Bayes Classifier Examples*. [online] Analytics Vidhya. Available at: <https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/> [Accessed 4 April 2022].

13.R. Zhang, Q. Liu, C. Gui, J. Wei, and H. Ma (2014) ‘Collaborative Filtering for Recommender Systems’ *2014 Second International Conference on Advanced Cloud and Big Data*, *pp. 301-308*. *Available at: https://ieeexplore.ieee.org/document/7176109*

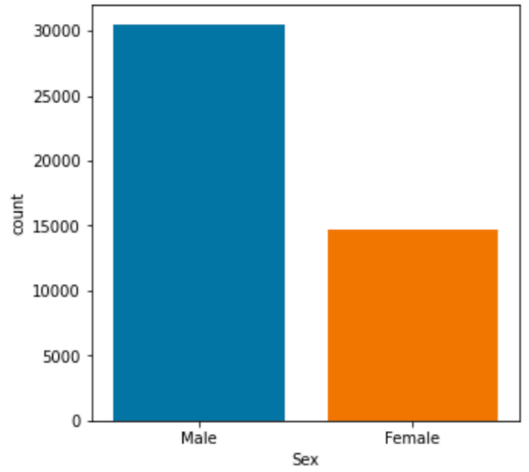
Appendix 1:



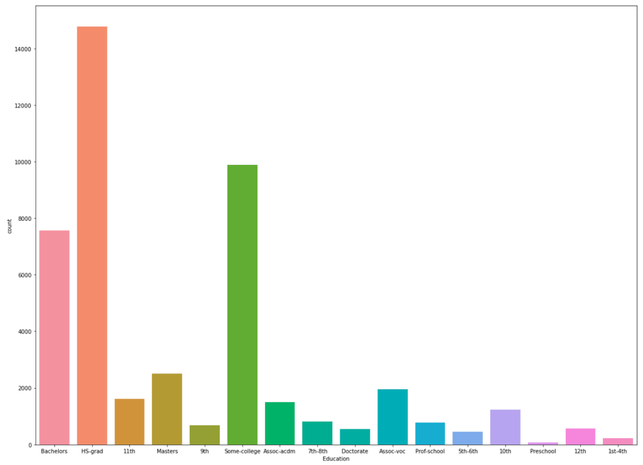
WorkClass: Private Employment is by far the most common.



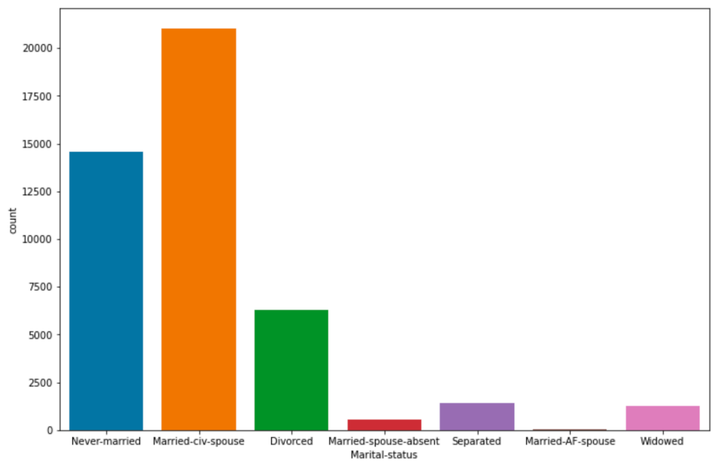
Occupation: is rather evenly distributed with Armed-Forces being the smallest class.



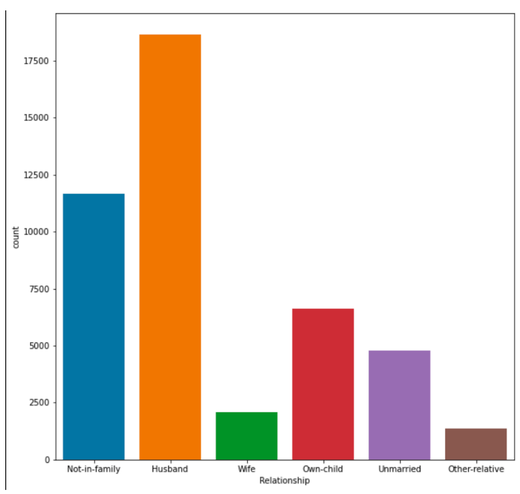
Sex: the male class was twice as common as the female class.



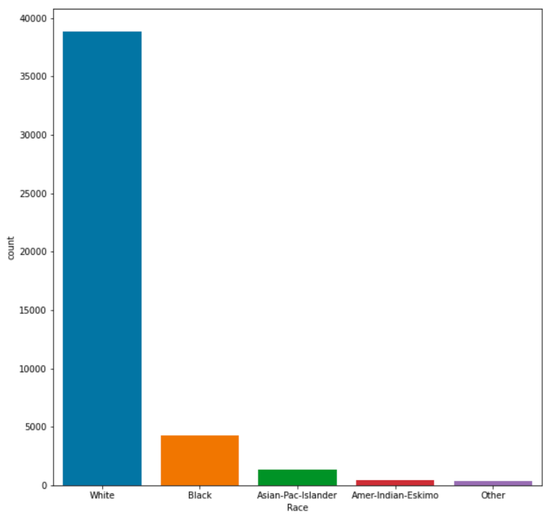
Education: High School and Some College are most common classes.



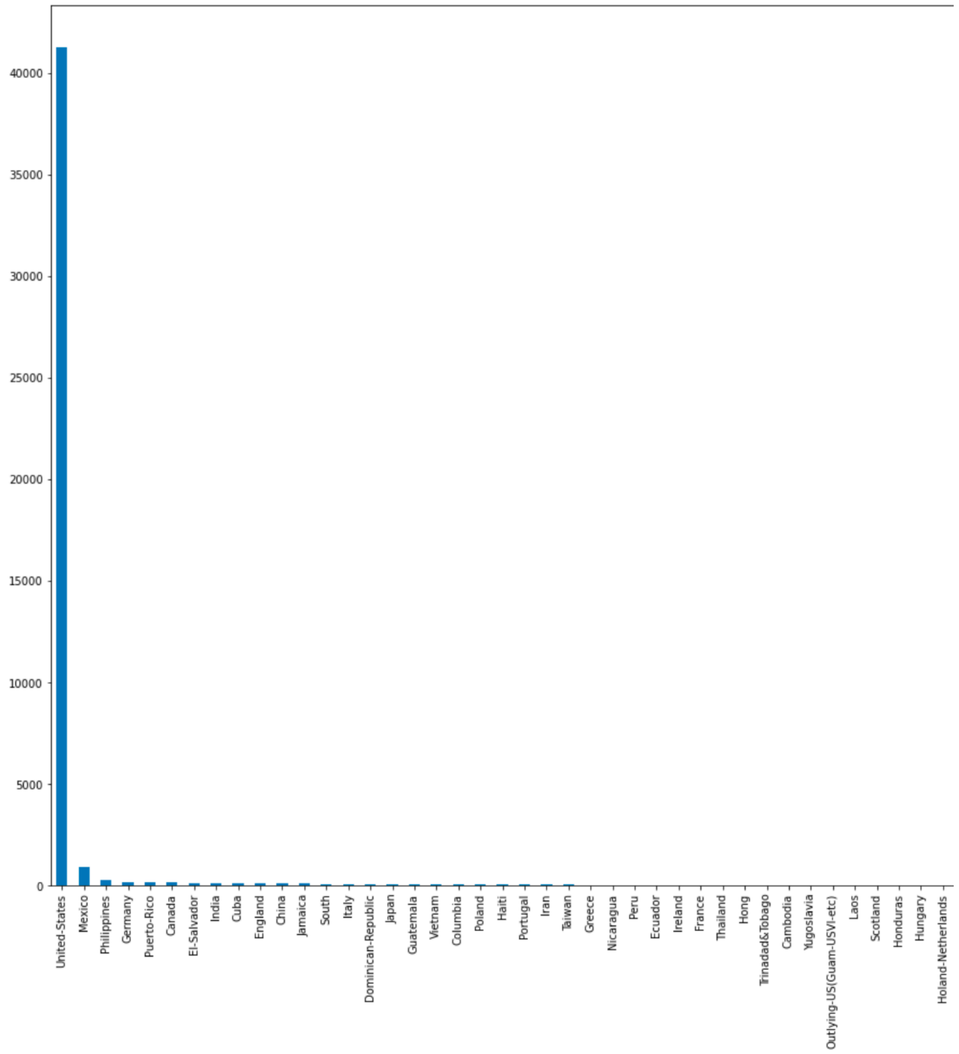
Marital Status: Never-married and Married are the most common classes.



Relationship: Wife class is significantly less common than the Husband class.



Race: White was is the most common class.



Native Country: United States of America was by far the most common class in the dataset.