MANCHESTER METROPOLITAN UNIVERSITY

MSC DATA SCIENCE

6G7Z1018

INTRODUCTION TO DATA SCIENCE

COURSEWORK: - HIGH EARNERS

DANIEL RAY

18053479

14/12/18

CONTENTS

INTRODUCTION

As part of the Introduction to Data Science module we have been tasked with mining some data set in order to produce some new information from said dataset. In this report, I will articulate the journey in which I have taken; starting with exploring and preparing the data in order to accurately mine the data to produce a model that can be used to generate new information about the dataset.

THE DATA SET

I have been given the "High Earner" dataset, which consists of several features relating to attributes of a sample of people, including age, sex, job type, etc. Doing some quick analysis in python, we can see the shape of the data set is (32561, 13) meaning that there are 32,561 observations along with 13 columns. The last column is named "High Income?" which has only two values, yes or no. In data mining this is referred to as the target variable or response, meaning this is the variable in which we want to predict.

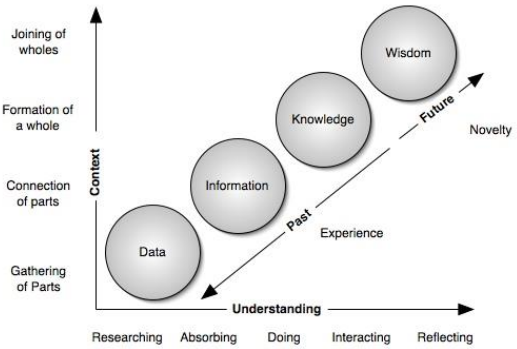
DATA, INFORMATION, KNOWLEDGE AND WISDOM HIERARCHY

Russell Ackoff popularised the notion that Data, Information, Knowledge and Wisdom are organised in a hierarchical structure. In his 1989 paper "From Data to Wisdom" he stated that:

- Data is symbols that represent properties of objects and events;
- Information is data that has been given meaning via relational connection;
- Knowledge is an appropriate collection of information with the intent of being useful and
- Wisdom is the process of judgement calling upon all previous levels of consciousness, such as morals and ethics, in order to synthesise new knowledge.

Ackoff theorised that data, information and knowledge are related to the past, meaning that they deal with what is already known or has been. Wisdom differs in the sense that it is related to the future; wisdom incorporates vision, design and consciousness in order to create something new. Ackoff went on to say that in order to achieve wisdom on some subject base, one must move through the previous stages in succession via the process of understanding.

Clark built on Ackoff's hierarchy, showing that not only was understanding the underlying relationship for transforming data into wisdom but context was too.



As you can see above, Clark associated verbs at intervals as understanding increased; going from researching, absorbing, doing, interacting and reflecting. He also showed how, as understanding increased so did context, going from gathering parts to connection those parts forming into a whole then joining wholes together.

There has been much discussion as to how accurate the hierarchy is a depicting reality, and I have to agree that the intricacies of formulating knowledge from data are more complex than just context

and understanding. However, as Data Scientist's we are familiar with how models work and the accuracy in which they depict reality. Subsequently, I would like to employ the hierarchy to formulate my aims and objectives of the assignment. I feel anchoring my thoughts to a model or framework allows me to be both concise and structured which will hopefully bring a rich narrative to the report.

AIMS & OBJECTIVES

The overarching aim of the assignment is to use the past to predict the future. More precisely, we want to increase context of the data to hopefully increase understanding, so much so that we are able to apply data mining techniques to the data to produce wisdom.

To achieve this, we first have to increase our understanding of the data. Using both numerical and visual summaries of the data we will be able to see trends and distributions of the data thus increasing context. Once we have a deeper understanding of the data, we can use the information generated to solve any issues within the dataset such as missing or erroneous data.

Now that we have been able to extract several pieces of information, we can connect them together to form knowledge from the data. Using this knowledge, we can select the correct data mining algorithms in order to produce wisdom. However, before said algorithms are applied to the data, we have to encode the data in such a way that maximises the performance of the data mining algorithms without losing the validity of the data.

As a side note, I would like to say that the definition of wisdom takes into account all previous levels of consciousness, which a computer or algorithm doesn't possess. Therefore, the data mining algorithms themselves don't produce wisdom but the data scientist who interprets the results does.

DATA PREPERATION & EXPLORATION

In order to gain a deeper understanding of the data set, we have to first ensure the data is complete and accurate. This will be done by eliminating any missing or erroneous data from the data set. Once we are confident that the data is complete, we will extract information from the data using various statistical concepts in order to formulate knowledge from the data.

DATA PREPARATION

Using the pandas library, we can see what columns are in the data set:

```
In [108]: #what columns do we have?
          census_df.info()

          <class 'pandas.core.frame.DataFrame'>
          RangeIndex: 32561 entries, 0 to 32560
          Data columns (total 13 columns):
          age                     32561 non-null int64
          employer type           32561 non-null object
          educational level       32561 non-null int64
          marital status          32561 non-null object
          job type                32561 non-null object
          relationship status     32561 non-null object
          ethnicity               32561 non-null object
          sex                     32561 non-null object
          capital gains           32561 non-null int64
          capital losses          32561 non-null int64
          hours worked per week   32561 non-null int64
          country of birth        32561 non-null object
          high income?            32561 non-null object
          dtypes: int64(5), object(8)
```

As stated before, there are 13 columns with 32,561 entries. The columns include age, employer type, educational level, marital status job type, relationship status, ethnicity, sex, capital gains, capital losses, hours worked per week, country of birth and high income?

The first step of this journey is to ensure that there is no missing data. The method which I employed to discover such data was to use the pandas function '.unique()' which returns all the unique values of that column. Using this function on each column will allow us to see if there are any missing values.

The first column which produced some missing data was the employer type, doing a 'value_counts()' on the '?' entry we can see that there are 1,836 entries with missing data. If we return a data frame of the whole dataset only concerning the '?' entries for employer type, we can see if there is any data related to employer type to see if we can resolve the missing data.

```
census_df.loc[census_df['employer type']=='?']
```

	age	employer type	educational level	marital status	job type	relationship status	ethnicity	sex	capital gains	ca lo
2	50	?	9	Married-civ-spouse	?	Husband	Black	Male	0	
35	70	?	13	Married-civ-spouse	?	Husband	Asian-Pac-Islander	Male	0	
86	21	?	10	Never-married	?	Own-child	White	Male	0	
94	68	?	9	Married-civ-spouse	?	Husband	White	Male	0	
144	20	?	9	Married-civ-spouse	?	Wife	White	Female	0	
151	50	?	7	Married-civ-spouse	?	Own-child	Black	Female	0	
187	49	?	9	Married-civ-spouse	?	Husband	White	Male	0	
197	22	?	9	Divorced	?	Other-relative	Black	Female	0	
200	69	?	11	Widowed	?	Not-in-family	White	Female	0	
204	68	?	16	Married-civ-spouse	?	Husband	White	Male	0	
227	18	?	7	Never-married	?	Own-child	White	Male	0	
232	64	?	10	Widowed	?	Not-in-family	White	Male	0	

Looking at the results we can see that both job type and employer type are missing. To double check I used the numpy 'where' function to create a new column called remove that has a value of 'y' if both the employer and job types values were missing. I did this as I could see no other way of inferring the employer type without the job type. I therefore removed all entries with 'y' in the remove column (1,836 entries) and subsequently removed the remove column.

The next column I found with missing data was job type, again I produced a data frame with all the missing values to see if I could infer the values from other values in the data set.

```
census_df.loc[census_df['job type']=='?']
```

	age	employer type	educational level	marital status	job type	relationship status	ethnicity	sex	capital gains	ca lo
1054	18	Never-worked	10	Never-married	?	Own-child	White	Male	0	
2740	23	Never-worked	4	Divorced	?	Not-in-family	White	Male	0	
3192	17	Never-worked	6	Never-married	?	Own-child	White	Male	0	
8469	30	Never-worked	9	Married-civ-spouse	?	Wife	Black	Female	0	
10969	18	Never-worked	6	Never-married	?	Own-child	White	Male	0	
11067	18	Never-worked	7	Never-married	?	Own-child	White	Female	0	
15652	20	Never-worked	10	Never-married	?	Own-child	Black	Male	0	

Here we can see that all the missing values (7 entries) for job type have an employer type 'Never-Worked'. We can then infer that the job type for these entries is 'Unemployed'. Using the replace function pandas offers, I changed the '?' values to 'Unemployed'.

Finally, the last column to posses missing data is Country of Birth, using the value_counts() we can see there are 556 entries with missing data. One way of resolving this, and the method I employed was to calculate the most occurring (mode) Country of Birth and replace the missing values with that. Using '.mode()' on the column we can see that United States is the most occurring with 30,725 entries. I therefore replaced the missing values with United States.

Now that we have dealt with all the missing values, the final step of the preparation phase is to ensure the data is relevant. Looking at the description of the data set provided, we can see that the column 'Relationship Status' is in reference to the relationship between the person and one other

person in the census, which isn't detailed in the dataset. Therefore, as this data isn't relevant due to the lack of surrounding information, I decided to remove it from the dataset. Now I am confident that we have a relevant and complete data set to explore. As a result of preparing the data set, the shape of it has changed, there are now only 30,725 observations with 12 columns.

DATA EXPLORATION

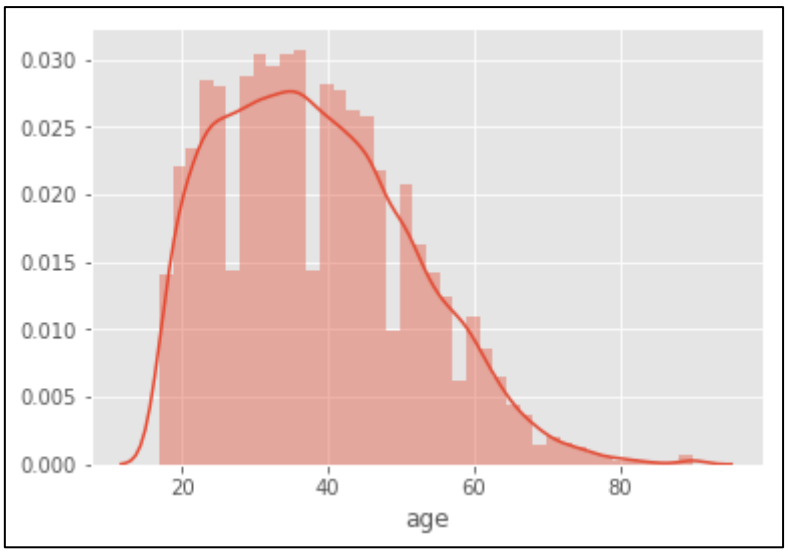
The next stage, and arguably the most important, is exploring the dataset. Here I will employ several statistical methods, both numerical and visual, in order to increase the context of the data which in turn increases my understanding of the data set. Transforming the data into information.

AGE

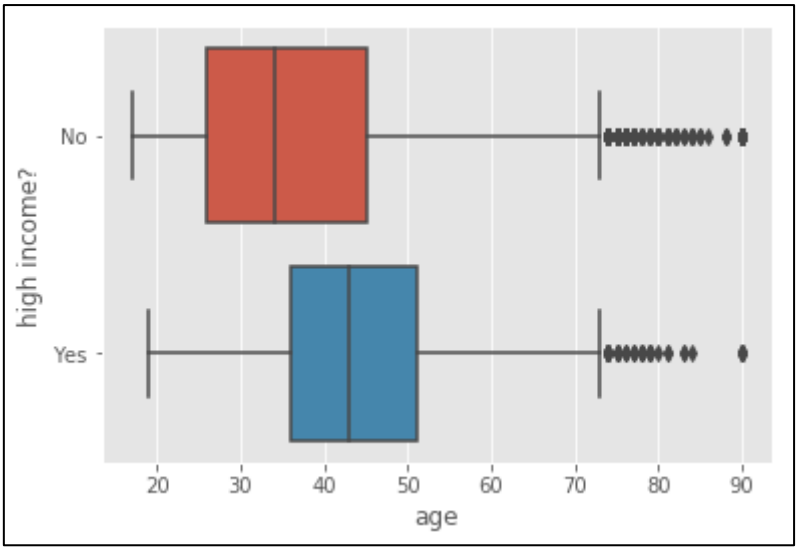
The first column I will explore is Age. Using the 'describe()', we can gain a statistical summary of the distribution of age in the dataset.

```
census_df['age'].describe()

count    30725.000000
mean        38.439512
std         13.119665
min         17.000000
25%         28.000000
50%         37.000000
75%         47.000000
max         90.000000
Name: age, dtype: float64
```

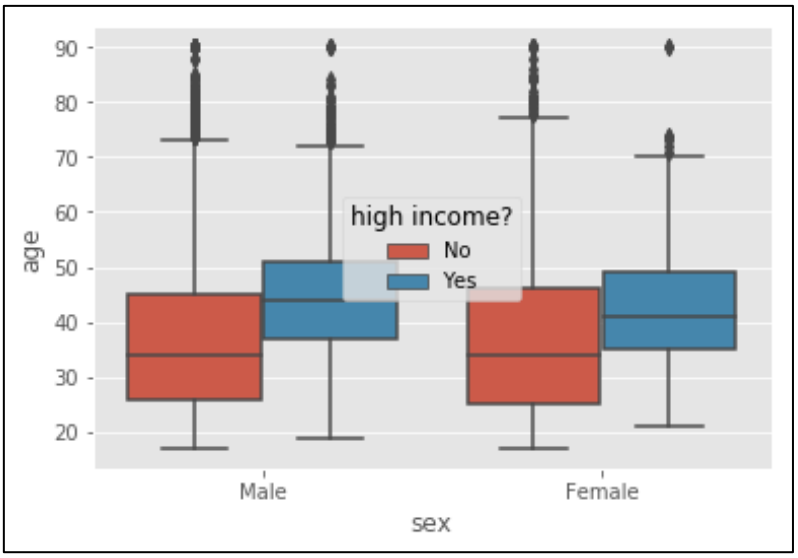
Looking at the results we can see the age ranges from 17-90 with an average age of 38.44. The IQR ranges from 28-47. Although very informative, I'm of the belief that a visual representation of the distribution will allow us to understand the age variable better.



The histogram produced allows us to see that the distribution of age is skewed to the left. Taking into account my domain knowledge of work with the retirement age being roughly around 60-70 depending on the country, it makes perfect sense that the distribution is skewed to the left. Employing more visual techniques, let see if we can gain any more understanding of age:



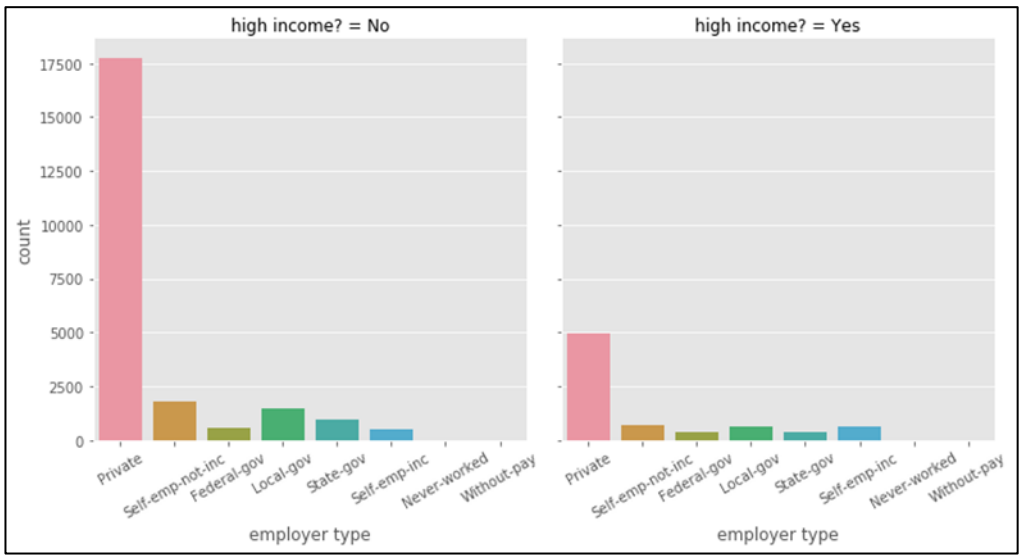
If we split the distribution of age by the high-income column, we can see that on average high earners tend to be older. The median of low earners is roughly 33 whilst high earners is roughly 42. We therefore have evidence to suggest that age is a factor that determines whether you are a high earner or not. Similarly, let see if sex is a factor:



We can see here that both the male and female distributions of age and high income are very similar. Male low earners on average are roughly 35, which is the same for female low earner. Male high earners however tend to be slightly older with a median of 44 whilst female high earners on average are 41.

EMPLOYER TYPE

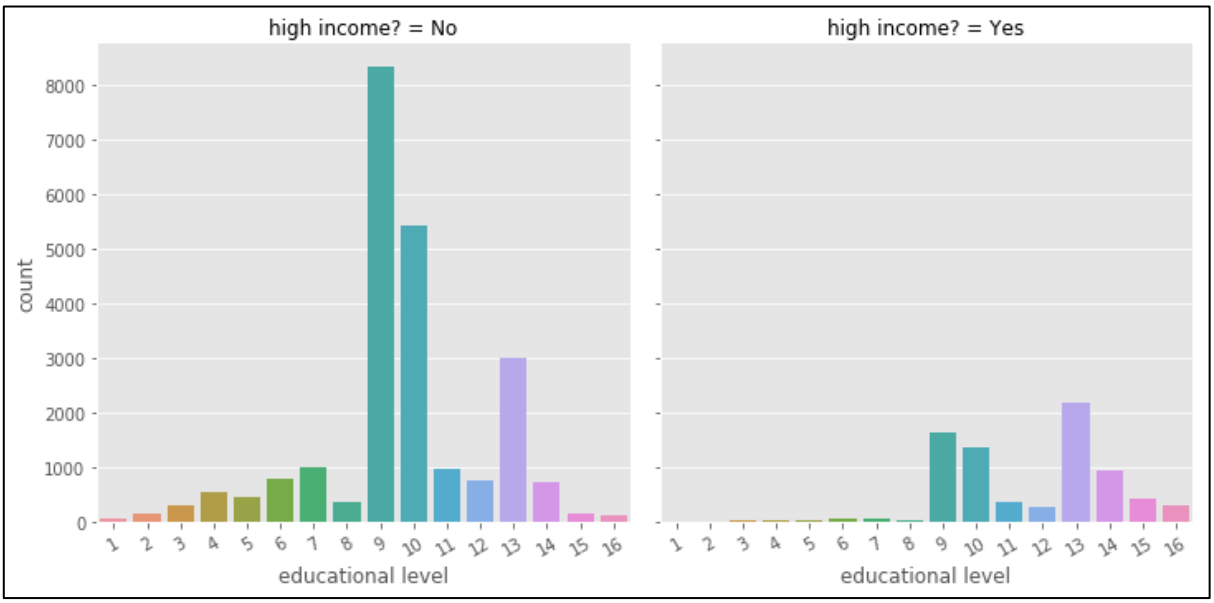
The next column I will explore is the employer type. As our target variable is high income, I will see if the distribution of employer type differs from high earners to low.



We can see that the distribution of employer type is similar in the sense that private is the most occurring value for both high and low earner. The government jobs have similar distributions, however, there is more low earners than high earners in government jobs. Finally, Self employed incorporated has an increase in frequency from low to high earners, meaning that there are more high earners who are self-employed incorporated than low earners who are self-employed incorporated.

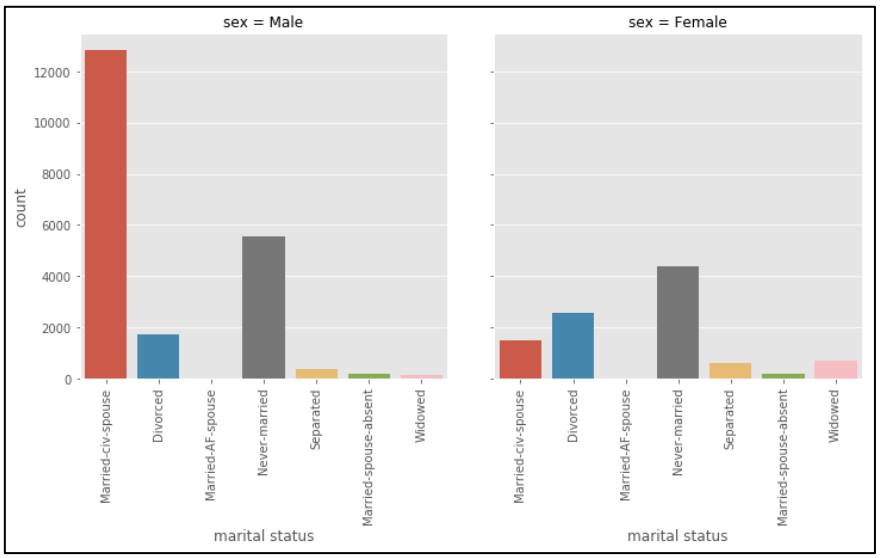
EDUCATIONAL LEVEL

Looking at education level dependant on high income, we can see that the majority of high earners have higher education levels, ranging from 9-16 (High School Graduate to Doctorate). The most occurring high earners are educated to a level of 'some college'. That being said there are several high earners who have been educated to level 3 or 5th-6th year at school. Low earners tend to be educated to a high school graduate level. Education levels 14-16 (Bachelors – Doctorate) are more likely to be high earners.

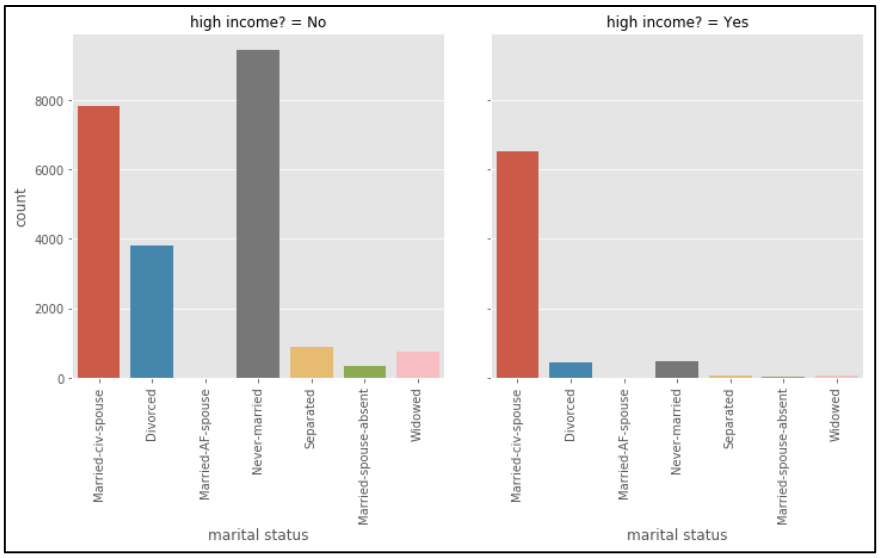


MARITAL STATUS

Looking at marital status dependant on sex, we see some interesting result:



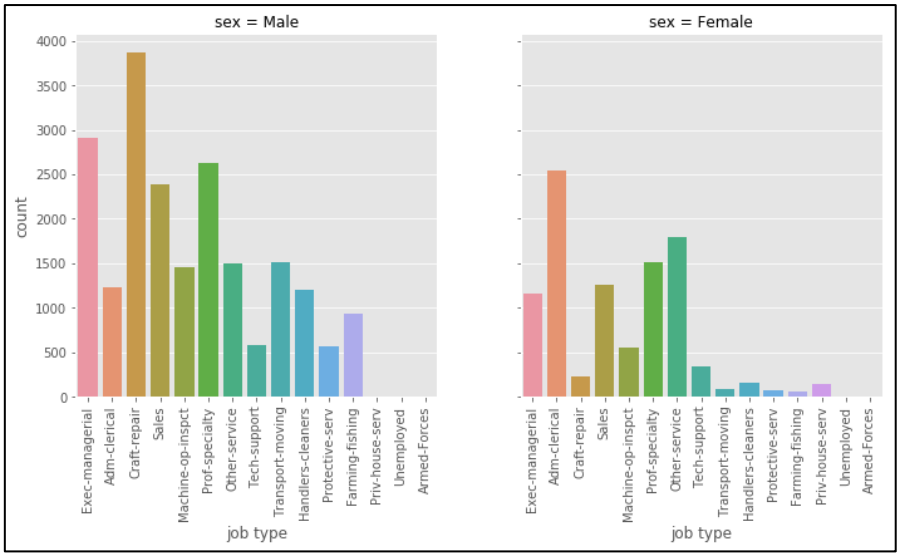
We can see that generally more men are married than women. We can also see that more men have never married than women. Women generally have more divorce and widow marital status.
If we look at marital status dependant on high income, we see some even more interesting results:



We can see that people who have never married, separated, married without spouse present and widowed all tend to have low incomes.

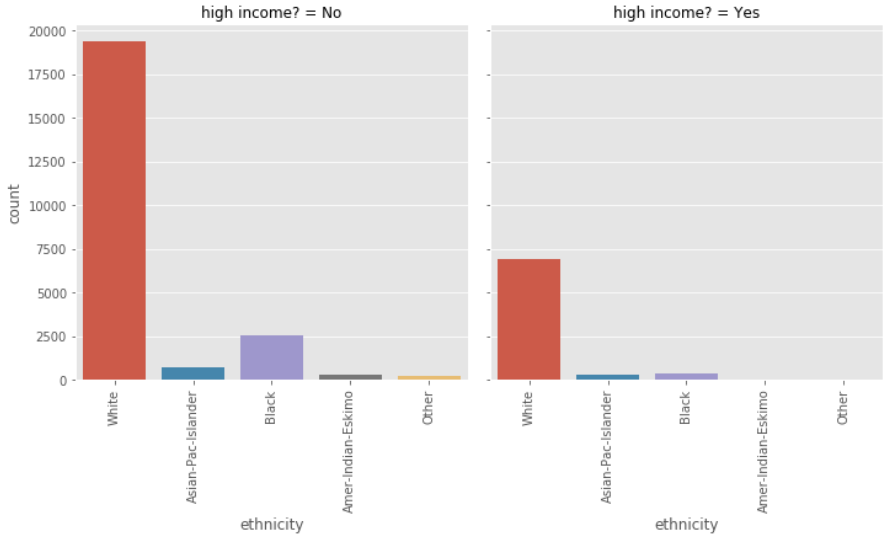
JOB TYPE

Looking at the job type frequencies dependant on sex, we can see that males tend to have job types in Exec-Managerial, craft repair, Machine op inspect and farming-fishing to name a few. Females tend to have admin-clerical and private house services job types.



ETHNICITY

If we look at ethnicity dependant on high income, we see that the majority of high earners are white. That being said, the majority of low earners are also white. American Indian Eskimos and others tend to be low earners. Comparing Asian Pacific Islanders, we see that there are more low earners than high earners however the drop is less significant than blacks and white.

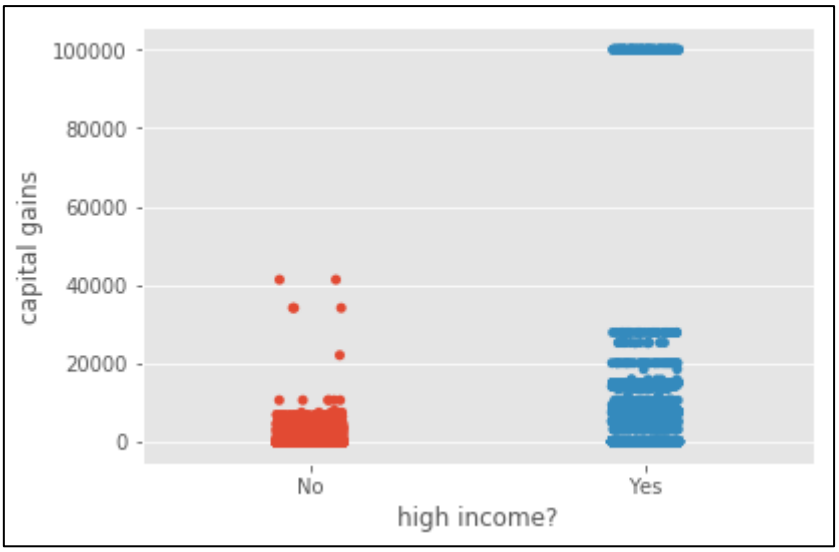


CAPITAL GAINS

Looking at the numerical summary of the distribution of capital gain we see that the mean value is $1105.79 ranging from $0 – $99,999.

```
census_df['capital gains'].describe()

count    30725.000000
mean      1105.785094
std       7497.027762
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max      99999.000000
Name: capital gains, dtype: float64
```

However, more interestingly if we split capital gains by high income, we see that high earners tend to have more capital gain. There are several low earners who have $20,000 or more but the majority lie closer to $0. If we look at high earners, we see that there is quite a few high earners with a capital gain of $100,000.

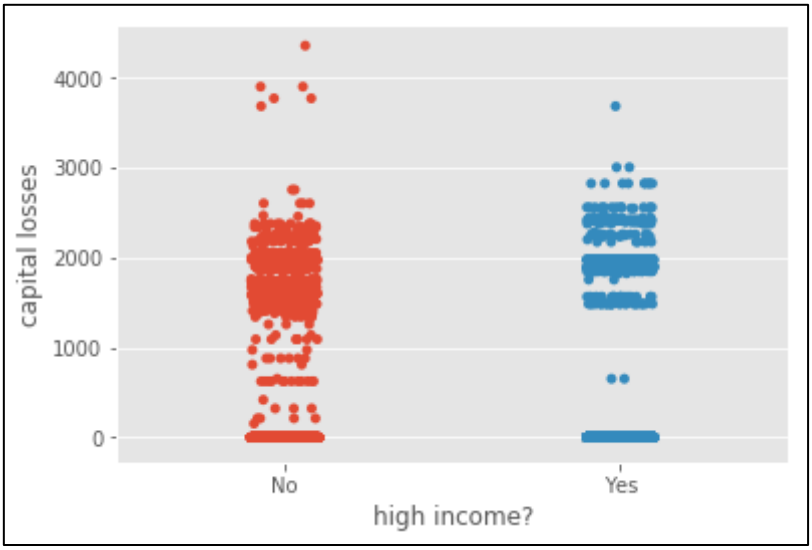


CAPITAL LOSSES

Looking at capital losses we see that the mean value is $88.89, ranging from $0 – $4,356:

```
census_df['capital losses'].describe()

count    30725.000000
mean        88.889959
std        405.613208
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max       4356.000000
Name: capital losses, dtype: float64
```

If we look at capital losses dependant on high income, we see that both low and high earners are mainly distributed from $0-$3000. However, we see that there are more low earners who have lost $3000 or more:

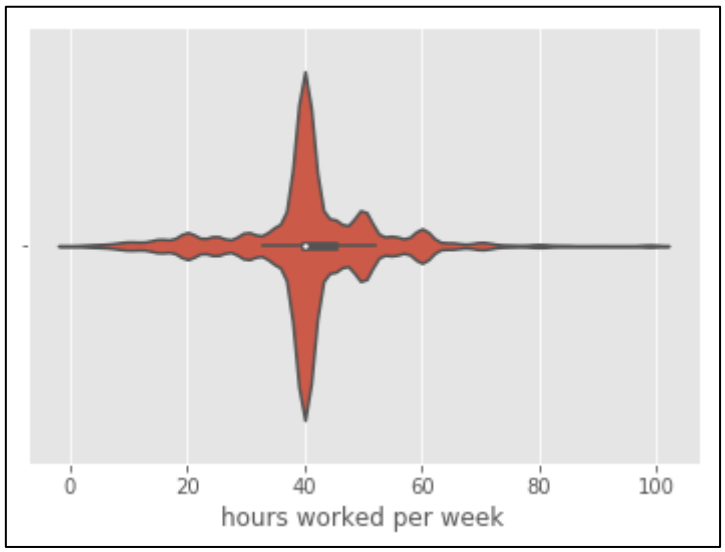


HOURS WORKED PER WEEK

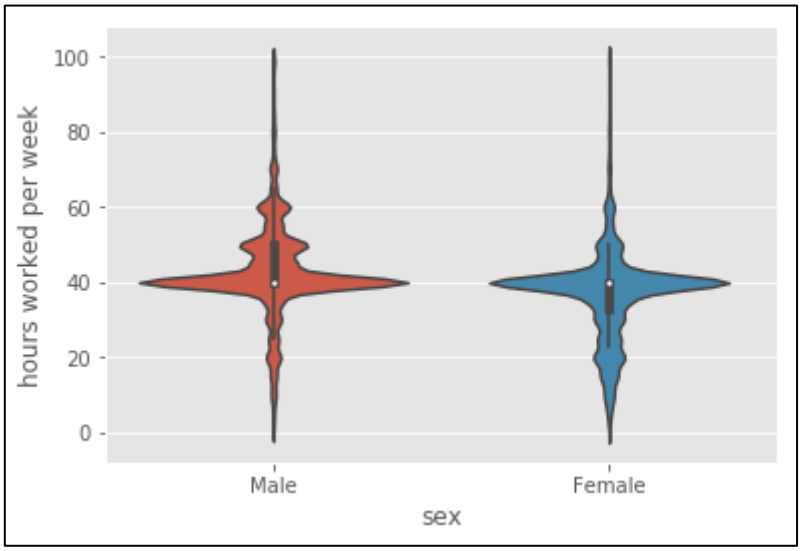
On average people tend to work 40.95 hours a week, ranging from 1 hour to 99 hours. The IQR ranges from 40-45 hour a week suggesting that the majority of people work around the 40 hours a week mark.

```
census_df['hours worked per week'].describe()

count    30725.000000
mean        40.946461
std         11.987385
min          1.000000
25%         40.000000
50%         40.000000
75%         45.000000
max         99.000000
Name: hours worked per week, dtype: float64
```

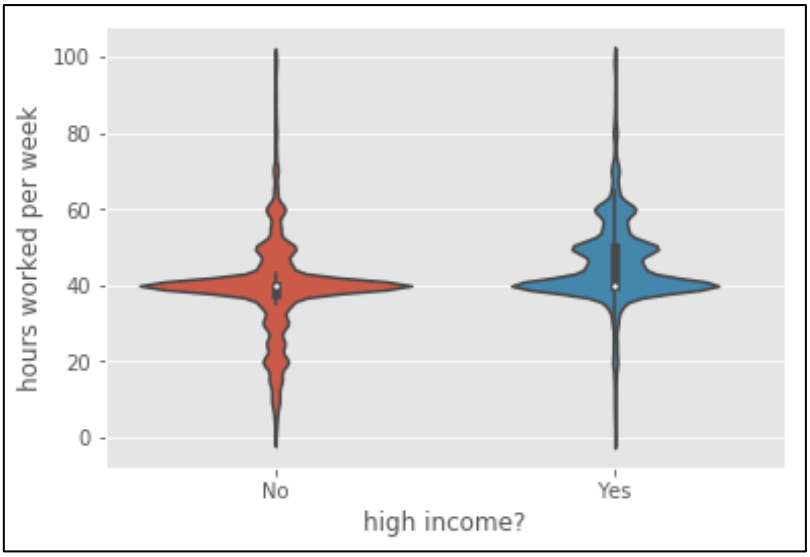
If we look at a more visual summary of the distribution of hours worked per week, we confirm the fact that the majority of people work 40 hours a week. More interestingly, we see that there are spikes in the distribution at every 10-hour interval starting at 20 hours a week to 70:



Now if we the distribution dependant on sex we see that both females and males tend to work 40 hours a week. Looking more closely we can see that above the 40 hour a week mark there are more males than females. Similarly, below the 40 hour a week mark we see that there are more females than males:

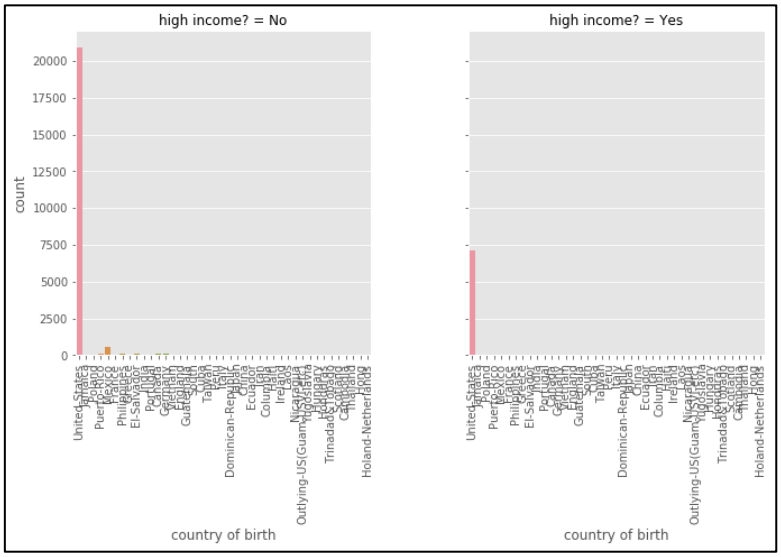


Lastly, if we look at high earners compared to low earners we see that high earners generally work more hours a week than low earners do. Both high and low earner however mostly work 40 hours a week:



COUNTRY OF BIRTH

Finally, looking at the frequencies of country of birth dependant on high income we see that if you are not from the United States then you are more likely to have a low income. We can see that Puerto-Rico, Mexico, Philippines, El-Salvador, Canada and Germany all have relatively high frequencies, if we ignore US, in low income but don't for high income.



DATA MINING & PREDICTION

Now that we have explored the dataset, increasing the context of the data therefore increasing our understanding of the data, I am confident to say that I have developed knowledge of the dataset as I have been able to connect the information together. The next step is to generate new information (wisdom) from the data in the dataset which is also known as Data Mining.

This section is split into three sub sections, the first and most important is feature engineering. This is the process of apply the knowledge we have just acquired to create features which allow the data mining algorithms work. The next section after that is apply the data to several data mining algorithms using sci-kit learn, in this section I will use several scores to measure their performance.

Lastly, the best performing algorithm will then be optimised in order to achieve the best possible performance.

FEATURE ENGINEERING

The first feature I will engineer is marital status. There are two types of married in the martial status feature – civ and af which refer to being in a civilian or armed forces family. I will therefore take this information out of the marital status feature and create a new feature called Armed Forces Family, which is of a Boolean type meaning that they either are or aren't in an armed forces family.

```
In [158]: ms = census_df['marital status']
          af = "Married-AF-spouse"

          census_df['Armed Forces Family'] = np.where (ms == af, 1, 0)
```

I then split the martial status feature into 3 categories, using numerical values to denote each type. The first type is married which encompasses married-civ-spouse, married-af-spouse and married-spouse-absent. The second group encompassing divorced, separated and widowed and finally the third being never married.

```
In [159]: census_df['marital status'] = census_df['marital status'].replace(['Married-civ-spouse',
                                                                              'Married-AF-spouse', 'Married-spouse-absent'], 0)

In [160]: census_df['marital status'] = census_df['marital status'].replace(['Divorced',
                                                                              'Separated',
                                                                              'Widowed'], 1)

In [161]: census_df['marital status'] = census_df['marital status'].replace(['Never-married'], 2)

In [162]: census_df['marital status'] = census_df['marital status'].astype(int)
```

Moving forward, I then mapped sex to numerical values, 1 being male and 0 being female. The same process was done for high income, mapping yes to 1 and no to 0:

```
In [163]: census_df['sex'] = census_df['sex'].map({"Male": 1, "Female":0})

          • and high income!

In [164]: census_df['high income?'] = census_df['high income?'].map({"Yes": 1, "No":0})
```

I then mapped Ethnicity to a binary value as well. This was due to the high number of white and a low number of other ethnicities. I used the replace function to replace black, other, Asian-pac-islander and amer-indian-eskimo to 1 and white to 0:

```
In [165]: census_df['ethnicity'] = census_df['ethnicity'].replace(['Asian-Pac-Islander',
                                                                    'Black', 'Amer-Indian-Eskimo',
                                                                    'Other'], 1)
          census_df['ethnicity'] = census_df['ethnicity'].replace(['White'], 0)
```

Using similar logic, I reduced country of birth to a binary value as well. Making United States 1 and the other remaining countries 0:

```
In [166]: census_df['country of birth'] = (census_df['country of birth'] == 'United-States').astype(int)
```

Looking at the distribution of hours worked per week, we saw spikes at every 10-hour interval from 20 – 70. Using this knowledge, I decided to reduce the range of values from 1-99 to 1-9, using the floor divide function:

```
In [167]: census_df['hours worked per week'] = census_df['hours worked per week']//10
```

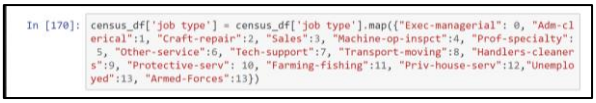
Looking at the distribution of government jobs dependant on high income we saw that there was a reduction from low to high earners however all three employer types possessed a similar shape going from low to high. I therefore decided to group the three government employer types together.

```
In [168]: census_df['employer type'] = census_df['employer type'].replace(['Federal-gov',
                                                                            'Local-gov', 'State-gov'], 'Govt')
```

I then mapped the employer types to numbers:

```
In [169]: census_df['employer type'] = census_df['employer type'].map({"Private": 0, "Self-emp-inc":1,"Self-emp-not-inc":2,  "Govt":3, "Never-worked":4, "Without-pay":5})
```

The final part of feature engineering I did was to map the job types to numerical values:



Now that we have engineered all the features in a manner that will allow the data mining algorithms to work correctly, we have to split the data set into three different groups. In order to prevent over fitting, we have to train the algorithms on one sub set of the data called the train set. We then validate the trained algorithm on the second sub set called the validation set. Then finally to make our predictions we use the test set. If we were to just train and test the algorithms on the same data we would get quite high accuracy scores as there has been no new data to test how accurately the algorithms were at modelling. I achieved this by using the test train split function from sci-kit learn:

DATA MINING

Using the Sci-kit learn library makes data mining very straight forward. The way in which it works is you first have to create an instance of the data mining algorithm with the relevant parameters. The data mining algorithm is then fit to the data using the training set. Once the algorithm has been fitted, the validation set is used to ensure the algorithm has been fitted correctly to the data. The final step is to then test the model on new data using the test set.

I will examine the K-Nearest Neighbour Algorithm, Support Vector Machine Algorithm and finally the Native Bayes algorithm. Once I have evaluated which is the most accurate at modelling the data, I will fine tune the algorithm to get the best accuracy.

KNN

k-Nearest Neighbours is a supervised learning algorithm, meaning that it is train on data where the target variable or response is known. Knn works for both classification and regression problems but we are only concerned with the classification part as this is the type of problem we are trying to solve in this assignment.

Knn works by looking at the neighbours of a certain data point and classifies the point by a majority vote, for example, if k = 3 and we give the knn algorithm a entry and the surrounding 3 neighbours of that entry are class 1, then that point to is class 1.

Now if we apply this algorithm to our data set we get:

KNN

```
In [177]: from sklearn.neighbors import KNeighborsClassifier
          knn = KNeighborsClassifier(n_neighbors=3)
          knn.fit(X_train, y_train)

Out[177]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=1, n_neighbors=3, p=2,
                     weights='uniform')

In [178]: valid = knn.predict(X_valid)
          print('Prediction {}'.format(valid))

          Prediction [1 0 0 ... 0 0 0]

In [179]: knn.score(X_valid, y_valid)

Out[179]: 0.8333550318968884

In [180]: from sklearn.metrics import classification_report
          print(classification_report(y_valid, valid))

                       precision    recall  f1-score   support

                    0       0.88      0.90      0.89      5720
                    1       0.69      0.64      0.66      1961

          avg / total       0.83      0.83      0.83      7681
```

Here we can see that the KNN algorithm has been instantiated into a variable called knn, which is then fit on the training data set. Once trained, the fitted knn is asked to predict the response for the validation set, which gives an accuracy score of 0.83, which is equivalent to 83%. Looking at the classification report we see that the f1 score is 0.89 for low earners but 0.66 for high earners meaning that knn was better at predicting low earner than high earners. The f1 score is a harmonic mean of precision and recall and is the score I will be using along with accuracy to evaluate the effectiveness of the algorithms.

If we then use the testing set to predict the response we get:

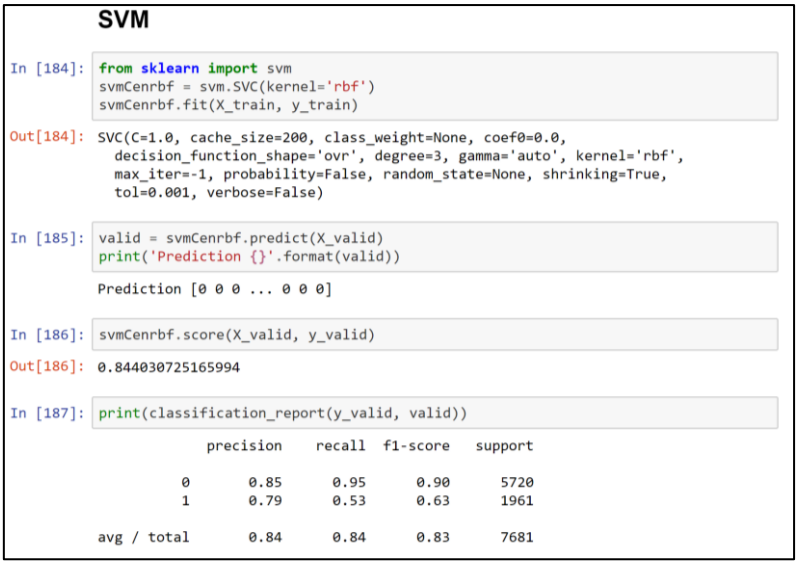
```
In [181]: prediction = knn.predict(X_test)
          print('Prediction {}'.format(prediction))

          Prediction [1 0 0 ... 0 0 0]

In [182]: knn.score(X_test, y_test)

Out[182]: 0.8342879458474356

In [183]: from sklearn.metrics import classification_report
          print(classification_report(y_test, prediction))

                       precision    recall  f1-score   support

                    0       0.88      0.90      0.89      5798
                    1       0.68      0.62      0.65      1884

          avg / total       0.83      0.83      0.83      7682
```

Here we can see that we got an accuracy score of 83.24% with an F1 score of 0.83.

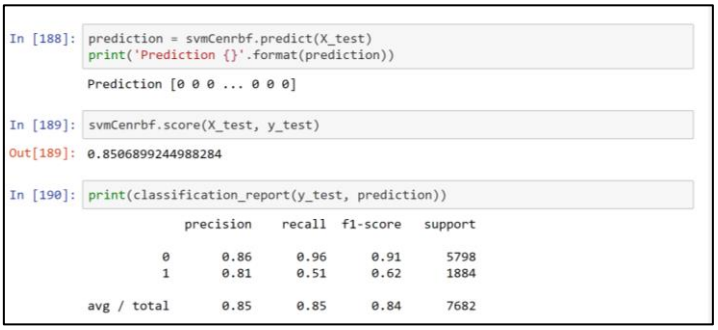
SVM

Support Vector Machines or SVM for short are a supervised learning algorithm that is used for classification problems. The SVM is based on the idea of finding a hyperplane that best divides the dataset into two classes, which is ideal for our data.

SVM's work well on smaller clean datasets, which I believe I have. Let's see how it performs:



Here we can see that the svm has been instantiated into a variable called svmCenrbf, using the rbf kernel which is the default. The instantiated svm is then fitted on the training data set. Once we run the fitted model with out validation set we get an accuracy score of 84.44% and a F1 score of 0.83 which has performed better than KNN. Similar to the KNN scores, the low earners are easier to predict. Now that we have validated the model, we will test it using the testing set:



Seeing the results of the SVM testing run we see that there is an accuracy score of 85.07% with an f1 score of 0.84 confirming that the SVM has performed better than the KNN algorithm.

NAIVE BAYES

Finally, we will examine how effective the Naive Bayes algorithms is at mining the data set. The naïve bayes algorithm is based on the probability theory called Bayes theorem. Like the previous two, the naïve bayes algorithm is good at classification. It assumes that a feature in a class is unrelated to any of the other features.

Lets see how it performs on the dataset:

```
In [191]: from sklearn.naive_bayes import GaussianNB
          nbCen = GaussianNB()
          nbCen.fit(X_train, y_train)

Out[191]: GaussianNB(priors=None)

In [192]: valid = nbCen.predict(X_valid)
          print('Prediction {}'.format(valid))

          Prediction [0 0 0 ... 0 0 0]

In [193]: nbCen.score(X_valid, y_valid)

Out[193]: 0.7884390053378466

In [194]: print(classification_report(y_valid, valid))

                       precision    recall  f1-score   support

                    0       0.80      0.95      0.87      5720
                    1       0.69      0.31      0.43      1961

          avg / total       0.77      0.79      0.76      7681
```

We can see that the NB algorithm is instantiated into the variable nbCen, which is then fitted with the training set. Once the validation set has been run on the fitted model we get an accuracy score of 78.84% and an f1 score of 0.76 which is the worst performing of the three algorithms. Finally we will run the test set on the trained model:

```
In [195]: prediction = nbCen.predict(X_test)
          print('Prediction {}'.format(prediction))

          Prediction [0 0 0 ... 0 0 0]

In [196]: nbCen.score(X_test, y_test)

Out[196]: 0.7913303827128352

In [197]: print(classification_report(y_test, prediction))

                       precision    recall  f1-score   support

                    0       0.81      0.95      0.87      5798
                    1       0.67      0.30      0.41      1884

          avg / total       0.77      0.79      0.76      7682
```

we can see that it has an accuracy of 79.13% with an f1 score of 0.76.

HYPER-PARAMETER TUNING

As we saw from the three algorithms above, the SVM algorithm performed best so this is the algorithm I will tune in order to increase the accuracy of the classification. The parameters I will be tuning are C and Gamma. The C parameter determines how many data samples are allowed to be placed in different classes whilst the gamma parameter determines the distance a single data sample exerts influence.

I will tune these parameters by going through two lists of values for the C and Gamma parameters, which will be used in a nested for loop that inputs all combinations of the parameter lists to find which performs the best:

Tuning SVM

```
In [92]: #range of values for c and gamma to see which scores best
         C_values = [0.001, 0.01, 0.1, 1, 10]
         gamma_values = [0.001, 0.01, 0.1, 1, 10]
         #init variables to store best score and params
         best_score = 0
         best_params = {'C': None, 'gamma': None}
         #nested for loop going through both lists of values
         for C in C_values:
             for gamma in gamma_values:
                 #using the values in list fit svm to data
                 svc = svm.SVC(C=C, gamma=gamma)
                 svc.fit(X_train, y_train)
                 #store the acc score in variable
                 score = svc.score(X_valid, y_valid)
                 #if statement to check if score is the best
                 if score > best_score:
                     #if so, store best score + params in variables
                     best_score = score
                     best_params['C'] = C
                     best_params['gamma'] = gamma

         best_score, best_params
Out[92]: (0.8591329254003385, {'C': 10, 'gamma': 0.01})
```

Here we can see the two lists range from 0.001 to 10. The results returned with a C value of 10 and a gamma value of 0.01. Using these parameters, we get an accuracy of 85.91%. Now lets run the SVM using these parameter:

```
In [93]: svmOpt = svm.SVC(kernel='rbf', C=10, gamma = 0.01)
         svmOpt.fit(X_train, y_train)
Out[93]: SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

In [94]: valid = svmOpt.predict(X_valid)
         print('Prediction {}'.format(valid))
         Prediction [0 0 1 ... 1 0 1]

In [95]: svmOpt.score(X_valid, y_valid)
Out[95]: 0.8591329254003385

In [96]: print(classification_report(y_valid, valid))
                      precision    recall  f1-score   support

                   0       0.87      0.96      0.91      5798
                   1       0.82      0.55      0.66      1883

         avg / total       0.86      0.86      0.85      7681
```

After validating and testing like in the previous section, we get an accuracy score of 85.91% and an f1 score of 0.85 on the validation set. Now lets use the testing set:

```
In [97]: svmOpt.score(X_test, y_test)
Out[97]: 0.8607133558969019

In [98]: print(classification_report(y_valid, valid))
                      precision    recall  f1-score   support

                   0       0.87      0.96      0.91      5798
                   1       0.82      0.55      0.66      1883

         avg / total       0.86      0.86      0.85      7681
```

Using the training set we get an accuracy score of 86.07% along with an f1 score of 0.85. Like all three algorithms in the previous section the mining algorithm found it harder to classify the high earners but the tuned SVM has produced the best f1 score for high earners of 0.66 and the best for low earner of 0.91.

CONCLUSION

In conclusion, I have cleaned and prepared data so that I was able to explore the data set to gain a deeper understanding of the data. Once I was able to formulate coherent information regarding the dataset, I was able to grasp a firm knowledge of the data which allowed me to engineer the features of the data in order to make the data mining algorithms work well. Once finding the best data mining algorithm I was able to tune the hyper parameters in order to get the best fitting model.

If I had more time, I would have like to have seen how an artificial neural network such as a multi-layer perceptron would have performed at the classification task. I say this as ANN's are the reason in which I have under taken this master's degree as I find it so compelling that we as humans are able to model the workings of a brain into a computer system in order to allow the system to learn. If you would like to see more work of mine regarding perceptron's, I created a web-based teaching aid to demonstrate the workings of the perceptron as part of my undergraduate dissertation which can be found here: https://danielray54.github.io/