2.1 Part 1: Performance Metrics in Regression

Initial Data Analysis

Most of the features are intuitive at their face value in relation a dataset regarding the cost of a diamond, for example 'clarity' obviously refers the diamonds ability to allow light to pass through it. However there are some features which require a bit more understanding as to what they exactly mean in relation to a diamond. There are five of these features, namely:

- X – Length of the diamond in millimeters
- Y – Width of the diamond in millimeters
- Z – Height of the diamond in millimeters
- Table –
- Carat – Corresponds to the weight of the diamond (I thought I knew what this meant but I actually did not)

This dataset prediction could be useful for someone who is either trying to value/buy a diamond. With a great regression model they will be able to see how far away a particular diamond is in price to buy/sell in relation to ones with similar features.

Exploratory Data Analysis

Firstly I wanted to get an understanding of the features composition so I ran the command *df.info()*. This returned to me the values given by 2.2.1 below and from this we can that there is 53,940 instances of diamond in the training set. More interestingly we can see that the dataset is composed of categorical and numerical features represented by object and float64/int64 respectively. Because of this we will have to convert the categorical data into numerical data in preprocessing. This is because most machine learning regression models cannot use the categorical data for their complex computations. Furthermore there is a feature named 'unnamed' which will be removed in the dimensionality reduction phase of preprocessing as it gives no value to predicting the price of a diamond. Finally from 2.2.1 we can deduce that there is no missing values in the diamond dataset which should save quite a bit of preprocessing time.

2.1.1 Diamond training information

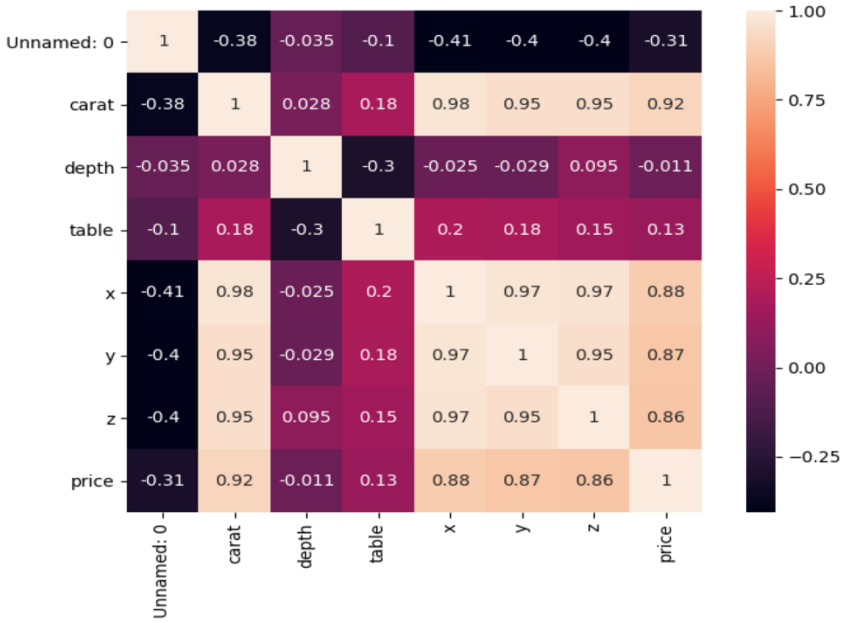
```
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 11 columns):
Unnamed: 0    53940 non-null int64
carat         53940 non-null float64
cut           53940 non-null object
color         53940 non-null object
clarity       53940 non-null object
depth         53940 non-null float64
table         53940 non-null float64
x             53940 non-null float64
y             53940 non-null float64
z             53940 non-null float64
price         53940 non-null int64
```

Using the command *df.describe()* I was able to produce the following information found in 2.2.2. Interestingly we can see that the features which represent the dimensions of the diamond (x,y,z) have a minimum value of zero. This is impossible and the values which have these occurrences will be removed during preprocessing. We can also see the standard deviation of each feature is relatively small considering the max/min/mean. This indicates that there are few outliers in this dataset which will need to be removed. However it may be noted that the standard deviation of the price feature is rather large, but that is likely because some diamonds are really expensive or relatively cheap.

2.1.2 Diamond training description

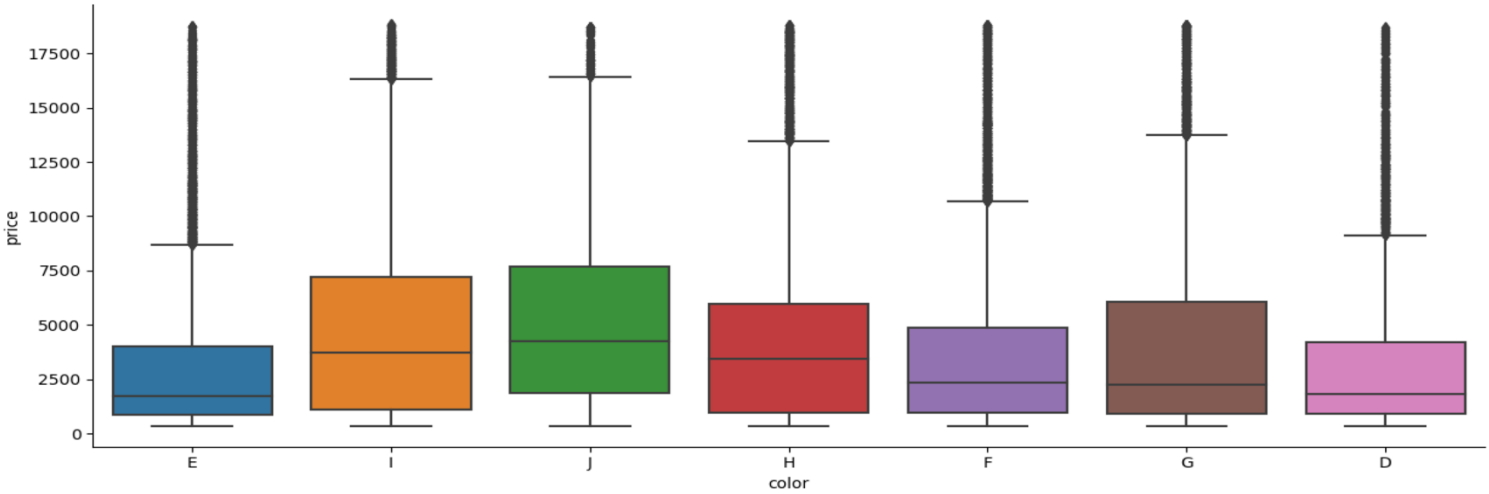
	carat	depth	table	x	y	z	price
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	0.797940	61.749405	57.457184	5.731157	5.734526	3.538734	3932.799722
std	0.474011	1.432621	2.234491	1.121761	1.142135	0.705699	3989.439738
min	0.200000	43.000000	43.000000	0.000000	0.000000	0.000000	326.000000
25%	0.400000	61.000000	56.000000	4.710000	4.720000	2.910000	950.000000
50%	0.700000	61.800000	57.000000	5.700000	5.710000	3.530000	2401.000000
75%	1.040000	62.500000	59.000000	6.540000	6.540000	4.040000	5324.250000
max	5.010000	79.000000	95.000000	10.740000	58.900000	31.800000	18823.000000

2.1.3 Correlation of each feature in relation to each other



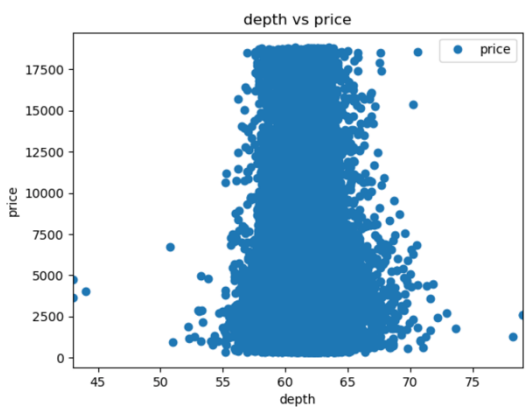
2.1.3 was a creation made by *sns.heatmap* and from it we can deduce some rather interesting things. One thing is that we can see that the feature 'Unnamed:0' has a very weak correlation to the price and because of this it furthers my proposition that it will be deleted in preprocessing. However 2.1.3 also shows which features have a strong correlation to the price of the diamond such as 'Carat', 'x','y' and 'z'. This informs us of the fact that the weight of the diamond (Carat) is the most influential feature on the price. We might also be able to deduce that the depth of the diamond is inversely correlated to the price of a diamond.

2.1.4 The 'cut' of the diamond in relation to its price



From 2.1.4 we ca see which types of cuts normally cost the least or the most. From the box and whisker plots we can see that 'I' and 'J' have the highest average price while 'D' and 'E' have the lowest. Now, exactly what does that mean? I found a website (https://www.lumeradiamonds.com/diamond-education/diamond-cut) which informs people about grading on a diamond cut. It essentially means how well the diamond is shaped and crafted by the person cutting it.

2.1.5 the depth of the diamond in relation to its price



In 2.1.5 we can see the inverse relationship between depth and price in the sense that if the diamond in to tall or short its price dramatically falls. However more interestingly we can see that there is a gold zone of depth which all diamond cutters are trying to attain and that the price of the diamonds which land in that zone vary significantly.

Preprocessing

Dimensionality reduction

For the reasons I have outlined in the Exploratory Data Analysis I have decided to remove the feature 'Unamed:0'. This is because it simply acts as a counter on instances and does not give any useful data as to what the diamond should cost, hence its poor correlation displayed in 2.1.3.

Feature manipulation

Firstly I had to convert the categorical data into numerical data so that the machine learning algorithms can process them correctly. Below in 2.1.6 are the commands I ran to get the cat codes of the categorical data.

2.1.6 Conversion of categorical features in the diamonds dataset

```
df['color'] = df['color'].astype('category').cat.codes
df['clarity'] = df['clarity'].astype('category').cat.codes
df['cut'] = df['cut'].astype('category').cat.codes
```

With regard to the dimension features ('x', 'y' and 'z') they had instances which had values of zero. This is an impossible value considering that any metric relating to the size of something cannot be zero without there actually being nothing there. Because of this I have decided that either the people who input the data made a mistake/could not find the data or there in fact is no diamond and consequentially each of these instances must be deleted. This is to ensure the patterns which the machine learning algorithm in relation to size features do not get skewed by strange outliers such as a dimension with no value. In 2.1.7 we can see that I have created essentially a new dataset for which will be used to train with excluding any instance which has any value of zero. The results of this command are displayed in 2.1.8.

2.1.7 Removal of all instances which have size dimensions of zero.

```
df = df[(df[['x','y','z']] != 0).all(axis=1)]
```

There is no missing data in this dataset which is great…

2.1.8 Size features *describe()* after 2.1.7

	x	y	z
53920.000000	53920.000000	53920.000000
5.731627	5.734887	3.540046
1.119423	1.140126	0.702530
3.730000	3.680000	1.070000
4.710000	4.720000	2.910000
5.700000	5.710000	3.530000
6.540000	6.540000	4.040000
10.740000	58.900000	31.800000

Evaluation of Regression models

Below in 2.1.9 are the reports results for the classifiers trying to predict the price of a diamond.

2.1.9 Results for the Regression Classifiers trying to predict the price of a diamond.

Classifier	**MSE**	**RMSE**	**R2**	**MAE**	**EX time (s)**
Linear Regression	1763535.97	1327.98	0.89	853.29	0.025
k-neighbours Regression	914072.83	956.07	0.94	512.74	0.48
Ridge Regression	1764125.70	1328.20	0.89	853.75	0.036
Decision Tree Regression (DTR)	544703.27	738.04	0.96	360.74	0.18
Random Forest Regression	342249.16	585.02	0.97	585.02	1.08
Gradient Boosting Regression (GBR)	445325.13	667.32	0.97	365.75	1.28
SGD Regression	Extremely large	5043485.18	-156469224	38921665.77	1.40
Support Vector Regression	18315294.4	4279.63	-0.12	2791.39	88.02
Linear SVR	3529249.32	1878.62	0.78	1089.86	0.24
Multilayer Perceptron Regression	25099720.88	5009.96	-0.054	3574.66	295.05

We have multiple metrics here which I believe need to be explained in terms of how they may help make evaluations of any specific classifier and how they are different.

- MAE is the average difference between the predicted values given by the trained model and the observed/actual values held in the test set.
- RMSE measure the average magnitude of error and unlike MAE it penalizes a classifier for predicting something for being more wrong.
- R2 is a measure which essentially compares the created model to a simple model which just predicts the mean.

With regard to the current dataset, people who deal with diamonds and would like to use a model to inform purchases or sales want a model with minimal prediction which are extremely wrong. Hence I believe the RMSE metric is the best metric to evaluate the classifiers on as it gives accolade to classifiers which are consistent with their predictions and never extremely wrong (which could cost someone a lot of money). Because of this I would say that the Random Forest Regression classifier is better for the purpose of informing someone about whether they should sell/buy diamonds than the GBR/DTR models which has a far superior MAE. This is because when GBR/DTR models are wrong they are can be wildly wrong relative to the Random Forest Regression model which is consistently close to the price. For the reasons given above I conclude the best model for predicting diamond prices is the Random Forest Regression model.

Most regression models performed well in the R2 metric, which informs me that some of the models have picked up on some of the underlying patterns of the dataset instead of relying on predicting the mean or mode to get a good MAE. The classifiers which struggled to give a score better than a simple model is likely because they simply overfitted to the training data. However there is one regression model (SGD) which scored horrendously in most metrics. I am unsure as to why this has occurred however I don't think it is correct to label it as a consequence of overfitting. It is far more likely that the characteristics of the classifier struggled immensely on this particular dataset.

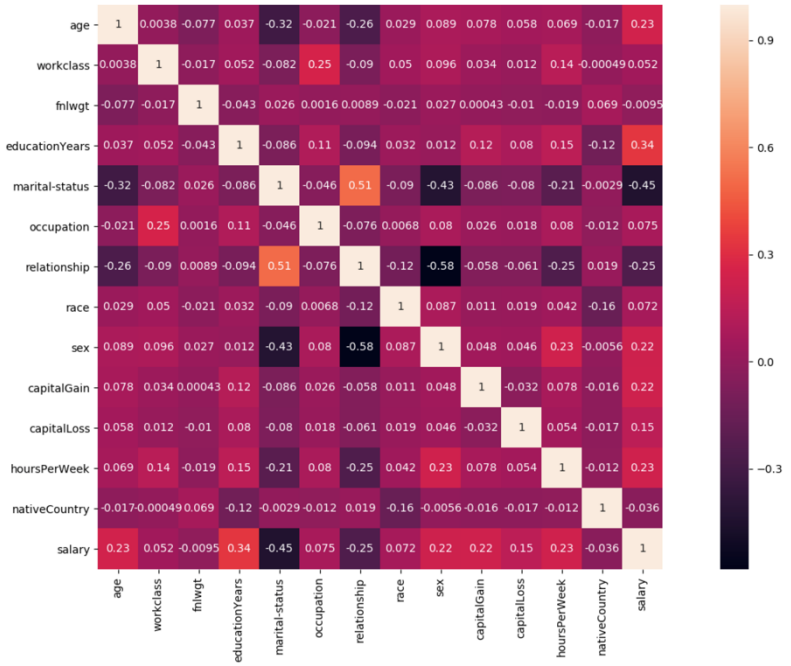
We can also analyse that some regression models execution were significantly longer than others. I know that the multilayer Perceptron Regression model spends a long time processing and creating the neural network which it uses to predict targets. To reiterate the point made above I think that the models which took a long time were likely overfitted to the training data and hence struggled to create a good model for predicting the price of a diamond. Perhaps if I had shortened the learning time/capabilities of the models they would have been able to identify the simpler patterns instead of just becoming very equipped to predict the training set.

2.2 Part 2: Performance Metrics in Classification

Initial Data Analysis

There were some features were not straight away intuitive, because of this I went to the UCI dataset archive https://archive.ics.uci.edu/ml/datasets/adult which I think is where the dataset was sourced – or at least a similar dataset was described. The webpage explained that this is a 'Census Income' dataset, which gives light to how and why the data was collected (government's use this information to inform policy). It explained that the 'capital-gain' and 'capital-loss' features relate to the person/instances asset income separate from salaries/wages. Furthermore it helped me understand the confusing 'fnlwgt' feature, which is a weighting given to each instance for the purpose of the census (not very relevant to predicting 'salary'). Other features such as 'age' or 'race' are intuitive.

2.2.1 Correlation between features of the dataset Adult Train



One of my initial analysis steps was to see what the correlation between features is. Using the *MATLAB* library features, I was able to construct a chart which displays yellow for strong correlation and purple for a weak correlation. I found that the feature 'fnlwgt' had little to zero correlation to the 'salary' feature. This is because the values of 'fnlwgt actually represents the weighting given to each instance, it gives no insight into an instances 'salary'. Because of the reasons above I will remove the feature. Furthermore we can see that 'marital-status' feature has a weak correlation to salary. I will not be removing this feature because I think it actually can provide insight into someone's salary. I might just have to do some techniques in the preprocessing phase to salvage it.

Exploratory Data Analysis

Firstly I wanted to find more about the actual features themselves. Because of this I ran the command *df.info()*

2.2.2 Adult Train information.

```
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 13 columns):
age              32561 non-null int64
workclass        30725 non-null object
educationYears   32561 non-null int64
marital-status   32561 non-null object
occupation       30718 non-null object
relationship     32561 non-null object
race             32561 non-null object
sex              32561 non-null object
capitalGain      32561 non-null int64
capitalLoss      32561 non-null int64
hoursPerWeek     32561 non-null int64
nativeCountry    31978 non-null object
salary           32561 non-null object
```

From 2.2.2 we can observe that the Adult training set has **32,561** entries. The dataset consists of both 'Categorical' and 'Numeric' features, respectively identified respectively as 'object' and 'int64' in 2.2.2. It can also be seen that three features, namely 'workclass', 'occupation' and 'nativeCountry' have missing values. This not similar to the test set which has **16,281** entries and no missing data.

Numeric Features Analysis

To understand the numerical values I ran the command *df.describe()*,which returns a series of values which are useful to analyse these features.

2.2.3 Adult Train Description (Numerical Features)

	age	educationYears	capitalGain	capitalLoss	hoursPerWeek
count	32561.000000	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	10.080679	1077.648844	87.303830	40.437456
std	13.640433	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.000000	0.000000	0.000000	1.000000
25%	28.000000	9.000000	0.000000	0.000000	40.000000
50%	37.000000	10.000000	0.000000	0.000000	40.000000
75%	48.000000	12.000000	0.000000	0.000000	45.000000
max	90.000000	16.000000	99999.000000	4356.000000	99.000000

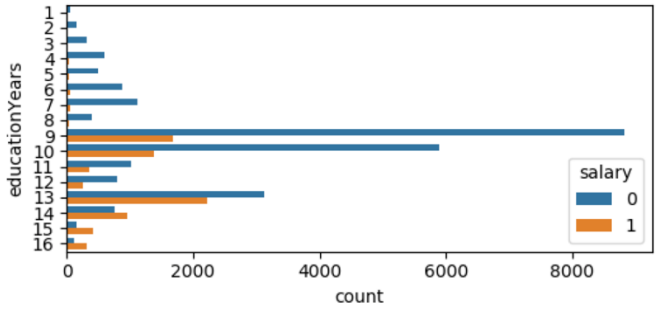
We can see in 2.2.3 more mundane values such as the mean and standard deviation…etc. These values may provide useful insight as to what values should be imputed if there was missing data, however the numerical features have none.

However also in 2.2.3 we can interestingly see that all the values are *standardized* similarly, which is required by many machine learning models. Because of this we will not have to use for example StandardScaler from the sklearn library.

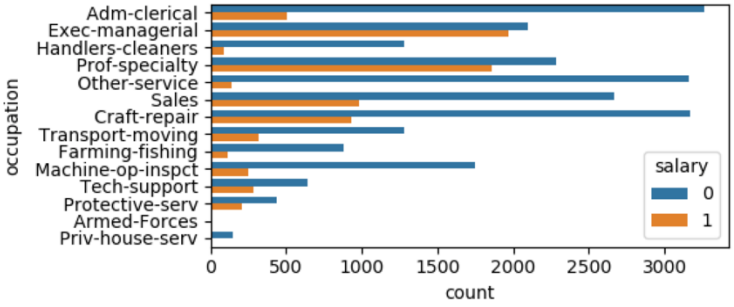
Categorical Features Analysis

Using *sns.countplot* I was able to visualise some categorical features in relation to their salary. I had to encode the 'salary' features values to meet the requirements for a count plot. Because of this if a person earns over/under $50,000 it will be binarily represented by 1 and 0 respectively.

2.2.4 Education years in relation to Salary



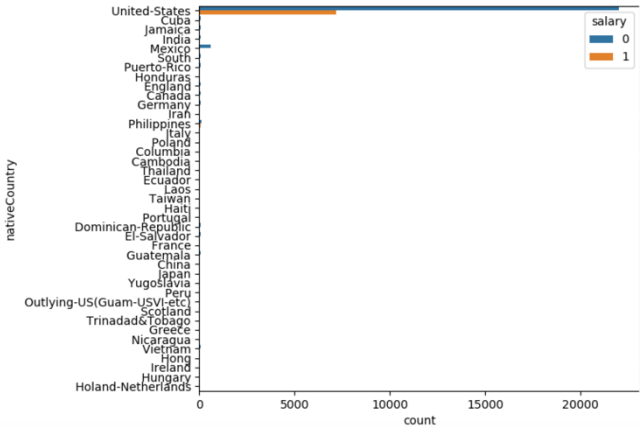
2.2.5 Occupation in relation to Salary



By having regard to 2.2.4 the 'Education' feature offers no separate information to informing whether someone earns over/under $50,000 from the 'EducationYears' feature. Because of this I will be removing the feature 'Education' and hence will be reducing the dimension of the dataset.

In 2.2.5 we can see that the "Armed-Forces" values for the 'occupation' feature has missing data. The 'workclass' and 'nativeCountry' features also share this characteristic. Because of this we will be imputing these values by giving them the most frequent value from each column. However I must note that I will only be doing this if I cannot see anyway that it will introduce or exacerbate any bias in/into the dataset.

2.2.6 Native Country in relation to Salary



From 2.2.6 we can see that there is a serious imbalance in values in favor of US to non-US countries. This is likely because the results are from a census which was carried out in the United States. Because of this I have decided that it would be a good idea to combine all non-US country into one value 'Non-US'. I will do this in the hope that it may alleviate the extreme bias that can come from this characteristic of the dataset.

Furthermore, because of the nature of categorical values they are not easily handled by machine learnings mathematical equations. Because of this I am going to have to encode them similarly to the 'salary' feature. This will make it easy for all the models to process the data.

Preprocessing

Dimensionality Reduction

For the reasons given in the Exploratory Data Analysis and Initial Data Analysis I have decided to remove the namely features before classification:

- 'fnlwgt' – It is irrelevant for predicting the salary of someone.
- 'education' – It is essentially the string version of 'educationYears' and thus provides no benefit to predicting 'salary' beyond slowing down processing speed.

Feature Manipulation

The marital-status feature in its current form provides little correlation to the instances 'salary'. This is because I don't believe the classification understood how each of the values relate to each other. For example people who are classified as 'Married-civ-spouse' and 'Married-AF-spouse' relate in the sense that the person is a couple. I don't think the models will be able to pick this up as they will be treated as separate values. Therefore I have decided to merge all the values which align to being in a couple together and similarly to people who are not. Below in 2.2.6 is how I carried out this task using the sklearn library.

2.2.6 Merging of marital-status values

```
df['marital-status'] = df['marital-status'].replace([' Divorced',' Married-spouse-absent',' Never-married',' Separated',' Widowed'],'Single')
df['marital-status'] = df['marital-status'].replace([' Married-AF-spouse',' Married-civ-spouse'],'Couple')
```

Because of this manipulation I decided to see how the 'relationship' and 'marital-status' feature would now interact together. We can see in 2.2.7 when having regard to each combinations mean salary we can see that there are clear combination which correlate to low or high salaries. Now because of this change we have some feature interaction which the classifier may use to predict better results.

2.2.7 Marital status and relationship and their respective salary mean

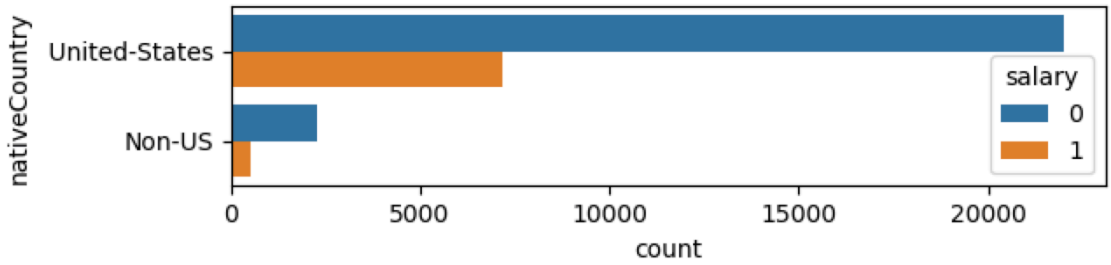
relationship	marital-status	salary
Husband	Couple	0.448571
Not-in-family	Couple	0.235294
	Single	0.102799
Other-relative	Couple	0.144000
	Single	0.022196
Own-child	Couple	0.177083
	Single	0.010056
Unmarried	Single	0.063262
Wife	Couple	0.475128

With regard to the 'nativeCountry' feature, I decided to group all countries which are not the United States as 'Non-US'. This will hopefully both alleviate the huge value imbalance posed by the feature 'nativeCountry' and ensure that the classifier gives countries who are not from the US fair weight and does not disregard them as outliers. Below in 2.2.8 we can see how I went about implementing this. After this we can see in 2.2.9 that the imbalance is significantly less than what it was in 2.2.6.

2.2.8 Merging of nativeCountry values

```
df['nativeCountry'] = df['nativeCountry'].replace([' Cuba',' Jamaica',' India',' Mexico',' South',' Puerto-Rico',' Honduras',' England','
```

2.2.9 Native Country in relation to salary



Finally I acquired the cat codes from each of the categorical features. This was so that the categorical features can now be easily processed by the classification algorithms. Below in 2.2.9 are the commands I ran to carry out this task

2.2.9 Encoding of categorical features

```
#Enconde the data so that it can be converted from categorical data for classification
df['workclass'] = df['workclass'].astype('category').cat.codes
df['marital-status'] = df['marital-status'].astype('category').cat.codes
df['occupation'] = df['occupation'].astype('category').cat.codes
df['relationship'] = df['relationship'].astype('category').cat.codes
df['race'] = df['race'].astype('category').cat.codes
df['sex'] = df['sex'].astype('category').cat.codes
df['nativeCountry'] = df['nativeCountry'].astype('category').cat.codes
```

Missing data

With regard to the missing data in the 'nativeCountry' feature I don't think it is appropriate for me to impute any value from the existing data because I believe it will only exacerbate an already existing bias in favour of US citizens. Because of this I have decided to impute the value of "no country", which will likely be treated by the classifier as outliers and hence disregarded as the test data has no missing values.

With regard to the missing data in the 'occupation' feature I have decided to impute the missing values with the most frequent value identified which is "Prof-specialty". Furthermore with regard to the missing data in 'workclass' I have decided to use the same imputation method as for 'occupation' which will lead me to replace all the missing data with "Private".

2.2.10 Dealing with missing data

```
df['workclass'].fillna("Private", inplace = True)
df['occupation'].fillna("Prof-speciality", inplace = True)
df['nativeCountry'].fillna("No Country", inplace = True)
```

Evaluation of Regression models

Below in 2.2.11 are the reports results for the classifiers trying to predict whether someone earns over/under $50,000 a year.

2.2.11 Reported results of all ten classifiers trying to predict someone's salary

Classifier	Precision %	Recall %	F1-score %	AUC %	R2	Accuracy %
kNN	67	69	68	54.98	-0.68	69.30
Naïve Bayes	64	70	66	50.15	-0.62	70.46
SVM	69	74	70	54.53	-0.43	73.85
Decision tree	69	70	69	56.66	-0.65	69.79
Random forest	67	76	67	50.98	-0.34	75.50
AdaBoost	69	73	71	56.53	-0.48	72.88
Gradient boosting	70	73	71	56.91	-0.46	73.25
Linear Discriminant Analysis	70	73	71	57.11	-0.45	73.44
Multi-layer Perceptron	6	24	9	50.00	-3.17	23.93
Logistic regression	67	72	69	53.26	-0.52	72.29

Is accuracy the best performance metric to evaluate a classifier?

Firstly accuracy is not a good performance metric to evaluate a classifier on a dataset if the dataset is heavily imbalanced. The adult dataset is heavily imbalanced in favour of people earning less than $50,000 a year. This means that a classifier can have little to no ability to distinguish between the two classes (over/under $50,000 a year) and still receive a really good accuracy score.

This is why the **area under the curve** metric is useful, it determines a classifiers ability to discern multiclass datasets. Therefore a classifier which hides its ability by predicting a heavily favored class in a imbalanced class will not perform well in this metric. For example we can see that the Random forest classifier in 2.2.11 acquired the highest accuracy score, however it struggled in AUC. This means that it is not actually a very good model for this imbalanced adult data set as it likely is not picking up on any of the patterns of the dataset and instead is just playing the numbers.

Furthermore **the f1-score** metric is also useful when trying to evaluate whether a classifier is good in an imbalanced class. The f1-score is a weighted average of the precision and recall metrics. This means it both 'false positives' and 'false negatives' are taken into account, which is something the accuracy metric neglects. Furthermore if the f1-score is low we can look to the recall or precision and the confusion matrix to see whether the classifier is having trouble with 'false positives' and 'false negatives' and consequentially we may be able to debug the classifier while having regard to this. This is something that accuracy is unable to do.

The two best algorithms according to the metrics given in 2.2.11

The two algorithms which I believe to the best classifier on this specific dataset is the Linear Discriminant Analysis and Gradient Boosting Classifier. This is because they have the highest AUC scores out of the ten classifiers which means they are the best at discerning between whether any person is going to have a salary over/under $50,000. The classifiers also doesn't appear to have a problem with producing either too many 'false positives' or 'false negatives' as the metrics of recall, precision and f1-score are all relatively balanced. However we must also note that the actual respective values of the recall, precision and f1-score for the models are the best out of all the classifiers. This likely means that the classification model has actually picked up on some of the underlying patterns which correlate to certain classes, instead of just picking the most likely outcome with meagre analysis. The models do also have a relatively good accuracy score but land in the middle, but that means almost nothing considering that the dataset is so imbalanced.

These two classifiers are the same in the sense that they perform well with regard to the metrics which I have valued considering that the dataset is imbalanced (AUC, f1-score, recall, precision). Both of these classifiers are similar in the fact they both carry out their classification mathematical equations with regard to gradient and creating trend lines (patterns) it finds in the data.

2.3 Part 3: Optimisation Methods

I began the first optimization method of mini-batch classifier and you can see the code which I did produce. However I could not work out how to shuffle the mini-batch so that it did not produce the same results every time.