APPLIED MACHINE LEARNING AND DATA MINING (AMLDM) COURSEWORK REPORT

Please complete all sections clearly in ball point pen

Do not write your name on your work unless your lecturer has explicitly told you to do so.

Student ID number	Title of degree studying	Level/Year
UP220923	Applied Machine Learning and Data Mining (AMLDM)	5

Short unit name:	M32365 AMLDM | **Due date: 20 February 2023** | **Deadline: 20 February 2023**	
Full unit name:	Applied Machine Learning and Data Mining	
Unit lecturer name:	Akshay Sachdeval	Group: (if applicable)
Additional items e.g. CD/disk/USB:	**Yes** | | **No** | | **Details:** |	

All additional items should be clearly labelled with ID number and unit name and securely attached to your work.

Candidates are reminded that the following are defined as Assessment Offences and will be dealt with in accordance with the University's Code of Student Discipline: _

a) Any attempt to complete an assessment by means considered to be unfair;
b) Plagiarism, whi_ch the University defines as the incorporation by a student in work for assessment of material which is not their own, in the sense that all or a substantial part of the work has been copied without any adequate attempt at attribution or has been incorporated as if it were the student's own work when in fact it is wholly or substantially the work of another person or persons.

Please note: Group coursework will be filed under the first Student ID number at the top of the list. Ensure you know all group member's ID numbers.

NB: Coursework not collected will be disposed of six months after the hand-in date.

FOR OFFICIAL USE ONLY

Date received/Office stamp	Academic Staff Member Provisional mark % / Comments

Task A – Supervised Learning

A.1.1 - Developing Classification Models using Python – Fetal Health Classification

A) Summarising the dataset

This dataset contains **2126** records of features extracted from Cardiotocogram exams, which were then classified by three expert obstetritians into **3 classes**:

1. Normal
2. Suspect
3. Pathological

B) Performing initial exploratory data analysis

Running through the dataset on python, I have found that there is a total of **2126 entries** with **no** null values. There are a total of **22 columns** of which all are **numerical values (float64).**

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2126 entries, 0 to 2125
Data columns (total 22 columns):
 #   Column                                                Non-Null Count  Dtype
---  ------                                                --------------  -----
 0   baseline value                                        2126 non-null   float64
 1   accelerations                                         2126 non-null   float64
 2   fetal_movement                                        2126 non-null   float64
 3   uterine_contractions                                  2126 non-null   float64
 4   light_decelerations                                   2126 non-null   float64
 5   severe_decelerations                                  2126 non-null   float64
 6   prolongued_decelerations                              2126 non-null   float64
 7   abnormal_short_term_variability                       2126 non-null   float64
 8   mean_value_of_short_term_variability                  2126 non-null   float64
 9   percentage_of_time_with_abnormal_long_term_variability  2126 non-null   float64
 10  mean_value_of_long_term_variability                   2126 non-null   float64
 11  histogram_width                                       2126 non-null   float64
 12  histogram_min                                         2126 non-null   float64
 13  histogram_max                                         2126 non-null   float64
 14  histogram_number_of_peaks                             2126 non-null   float64
 15  histogram_number_of_zeroes                            2126 non-null   float64
 16  histogram_mode                                        2126 non-null   float64
 17  histogram_mean                                        2126 non-null   float64
 18  histogram_median                                      2126 non-null   float64
 19  histogram_variance                                    2126 non-null   float64
 20  histogram_tendency                                    2126 non-null   float64
 21  fetal_health                                          2126 non-null   float64
dtypes: float64(22)
memory usage: 365.5 KB
```

All features described as mentioned from the dataset as follows:

baseline value: Baseline Fetal Heart Rate (FHR) (beats per minute)

accelerations: Number of accelerations per second

fetal_movement: Number of fetal movements per second

uterine_contractions: Number of uterine contractions per second

light_decelerations: Number of light decelerations (LDs) per second

severe_decelerations: Number of severe decelerations (SDs) per second

prolongued_decelerations: Number of prolonged decelerations (PDs) per second

abnormal_short_term_variability: Percentage of time with abnormal short term variability

mean_value_of_short_term_variability: Mean value of short term variability

percentage_of_time_with_abnormal_long_term_variability: Percentage of time with abnormal long term variability

mean_value_of_long_term_variability: Mean value of long term variability

histogram_width: Width of histogram made using all values from a record

histogram_min: Histogram minimum value

histogram_max: Histogram maximum value

histogram_number_of_peaks: Number of peaks in the exam histogram

histogram_number_of_zeroes: Number of zeros in the exam histogram

histogram_mode: Histogram mode

histogram_mean: Histogram mean

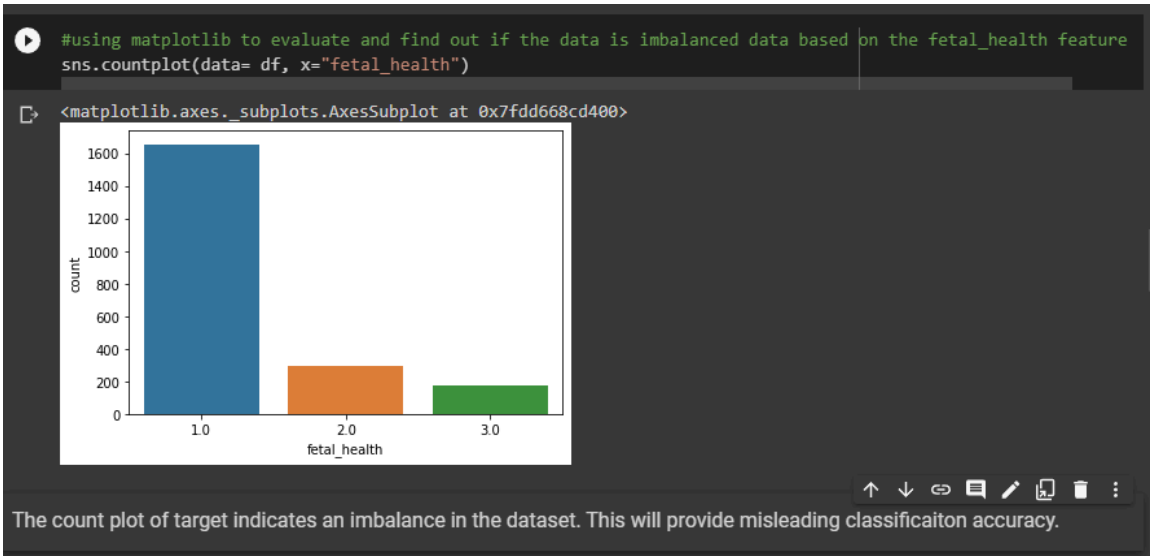
histogram_median: Histogram median

histogram_variance: Histogram variance

histogram_tendency: Histogram tendency

fetal_health: Encoded as 1-Normal; 2-Suspect; 3-Pathological.

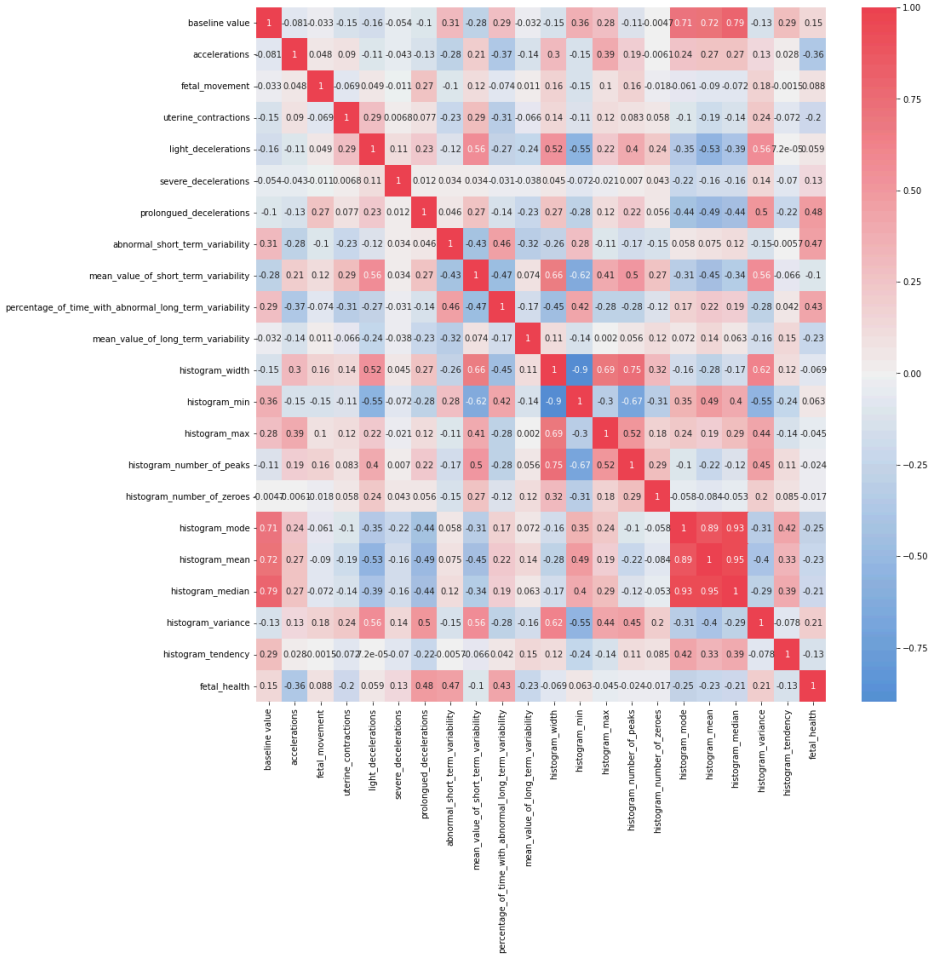
Since the aim of the project is to predict child and maternal mortality, the target feature was Fetal Health which was the result of the Cardioctocogram (CTG). A count plot was performed to look for class imbalance to decide whether data scaling was required.



C) Preparing the dataset for training

Before training and testing the model. I used a Correlation Matrix figure to find out which attributes highly correlates to the target feature (fetal_health).

```
#Plotting correlation matrix
corrmat = df.corr()
plt.figure(figsize=(15,15))
cmap = sns.diverging_palette(250, 10, s=80, l=55, n=9, as_cmap=True)
sns.heatmap(corrmat, annot=True, cmap=cmap, center=0)
```



From the correlation table, I can see that acceleration, prolonged deceleration, abnormal short term variability and percentage of time with abnormal long-term variability has the largest coefficient with fetal health.

accelerations has a -0.36 correlation with fetal_health

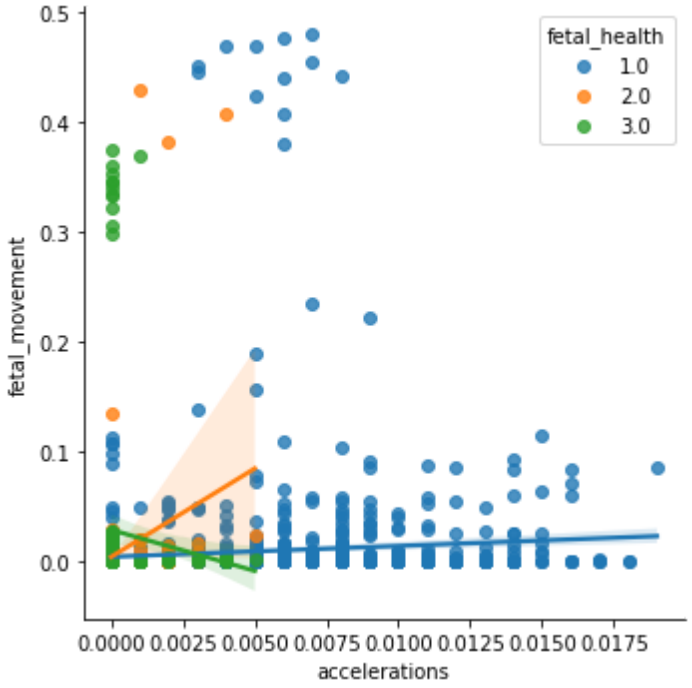
prolonged_decelerations has a 0.48 correlation with fetal_health

abnormal_short_term_variability has a 0.47 corelation with fetal_health

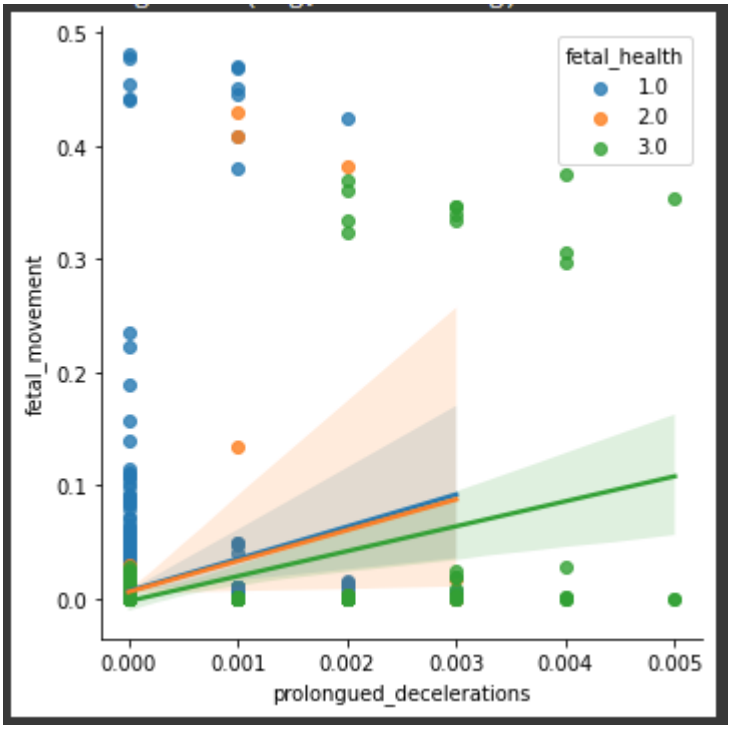
percentage_of_time_with_abnormal_long_term_variability has a 0.43 correlation with fetal_health

Using the 4 different features selected, I ran them through a scatter plot to look for the trends.

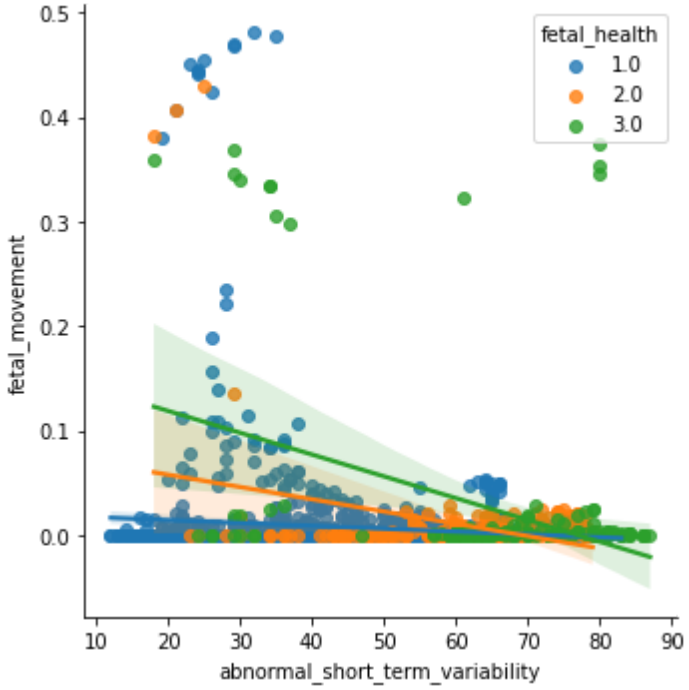
```
sns.lmplot(data =df,x="accelerations",y="fetal_movement", hue="fetal_health",legend_out=False)
plt.show()
```



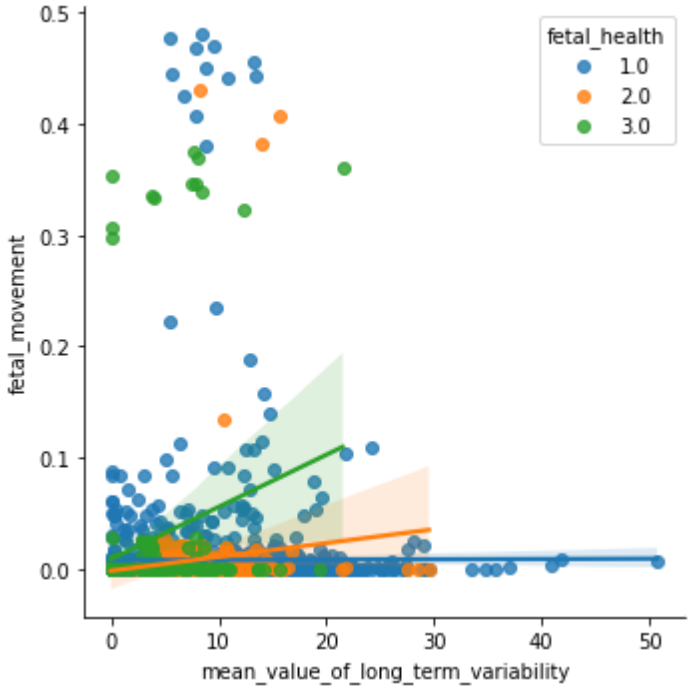

```
sns.lmplot(data =df,x="prolongued_decelerations",y="fetal_movement", hue="fetal_health",legend_out=False)
plt.show()
```




```
sns.lmplot(data =df,x="abnormal_short_term_variability",y="fetal_movement", hue="fetal_health",legend_out=False)
plt.show()
```




```
sns.lmplot(data =df,x="mean_value_of_long_term_variability",y="fetal_movement", hue="fetal_health",legend_out=False)
plt.show()
```



With accelerations vs fetal movement, higher fetal movement with lower accelerations will trend towards abnormal fetal health; with normal fetal health trending towards a gradual acceleration.

With prolonged decelerations vs Fetal movement, the longer the prolonged deceleration, the trend of abnormal fetal health can be seen.

With Abnormal short term variability vs Fetal movement, higher fetal movement during high percentage of abnormal short term variability will trend towards a abnormal fetal health.

With mean value of long term variability vs Fetal Movement, the faster the fetal movement over a greater mean value of long term variability, the trend of abnormal fetal health can be seen.

The graphs shows the rate of change with each graph showing a specific trend. The outliers are also clearly shown. The outliers may be a measurement error or data entry error but `since the dataset is the outcome of a CTG report,

it is assumed that it is unlikely to be a data entry error.

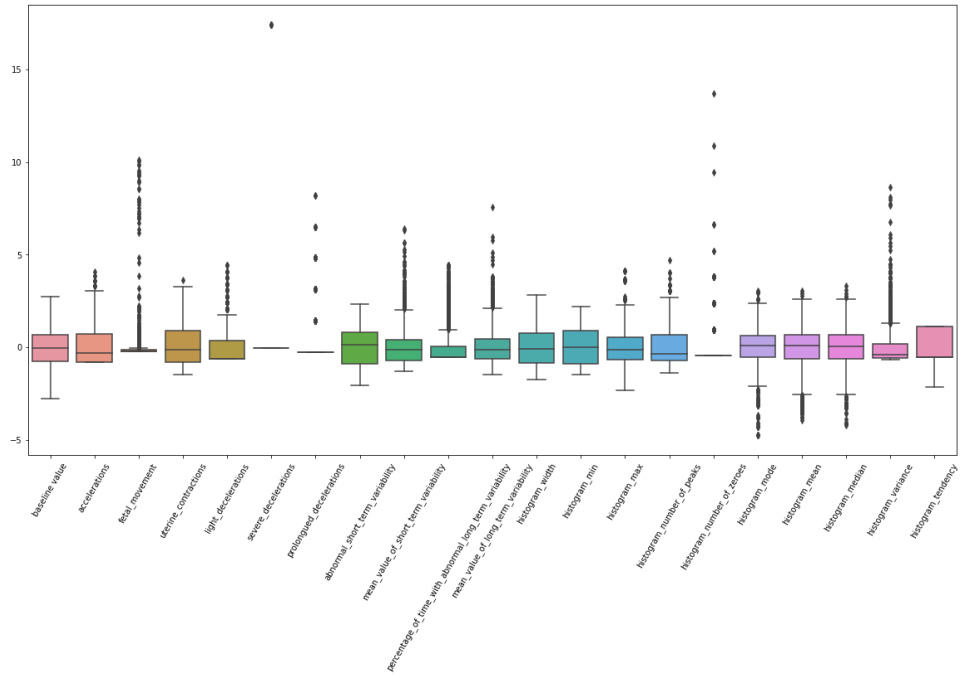
D) Training the model

```
#Defining values to features as X and target as Y
X=df.drop(["fetal_health"],axis=1)
y=df["fetal_health"]
```

X was defined with fetal health dropped since it was the target feature (y).

```
#Standard scaler for the features
col_names = list(X.columns)
s_scaler = preprocessing.StandardScaler()
X_scale = s_scaler.fit_transform(X)
X_scale = pd.DataFrame(X_scale, columns=col_names)
X_scale.describe().T
```

Standard Scaler was performed to better fit the training.



The new plot indicated that all the features are in the same range since scaling has been done. Outliers can be spotted in a few features. Since this is from a CTG report, assuming that the outliers are not caused by typo or measurement errors, I cannot drop or remove the data from the model as it will lead to a loss of information

```
#Splitting the training and testing variables
X_train, X_test, y_train, y_test = train_test_split(X_scale, y,test_size=0.3, random_state=42)
```

```
#Making use of pipelines for various classifier ref:https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html
pipeline_lr = Pipeline([('lr_classifier',LogisticRegression())])
pipeline_dt = Pipeline([('dt_classifier', DecisionTreeClassifier())])
pipeline_rf = Pipeline([('rf_classifier', RandomForestClassifier())])
pipeline_knn = Pipeline([('knn_classifier', KNeighborsClassifier())])
#listing all pipelines
pipelines = [pipeline_lr, pipeline_dt, pipeline_rf, pipeline_knn]
#Dictionary of pipelines and classifier types for ease of reference
pipe_dict = {0: 'Logistic Regression', 1: 'Decision Tree', 2: 'Random Forest', 3: 'KNN'}
#Fitting the pipelines
for pipe in pipelines:
  pipe.fit(X_train, y_train)
```

Using pipelines to test the model. Classification techniques used:

- Logistic Regression
- Decision Tree
- Random Forest
- K-Nearest Neighbours

```
[ ]  #Validation on accuracy
     results_accuracy = []
     for i, model in enumerate(pipelines):
       score = cross_val_score(model, X_train, y_train, cv=12)
       results_accuracy.append(score)
       print("%s: %f " % (pipe_dict[i], score.mean()))

     Logistic Regression: 0.894489
     Decision Tree: 0.917339
     Random Forest: 0.940188
     KNN: 0.893145
```

Based on these test scores, Random Forest seems to have the best accuracy score among the four classifiers used, while KNN has the lowest accuracy used.

To see if Random Forest was indeed accurate, I ran it through a Confusion Matrix:

```
#Confusion Matrix RandomForest
print("Confusion Matrix :")
print(confusion_matrix(y_test, predict_rfc))

Confusion Matrix :
[[484   9   3]
 [ 22  76   3]
 [  2   2  37]]
```

Looking at the Confusion Matrix, a large portion of the data has a True Positive compared to

False positives and False negatives.

E) Conclusion

Based on the results above, I can conclude that Random Forest was indeed a better classification model to be used for predicting child and maternity mortality.

A.1.2 - Developing Classification Models using Python – Rainfall Prediction (Australia)

A) Summarising the dataset

This dataset contains about 10 years of daily weather observations from many locations across Australia.

RainTomorrow is the target variable to predict. It means -- did it rain the next day, Yes or No? This column is Yes if the rain for that day was 1mm or more.

B) Performing initial exploratory data analysis

Looking through the data file weatherAUS.csv, I have found that this dataset contains 145,560 entries, 23 variables; which 16 are numerical and 7 object classes.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 23 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   Date           145460 non-null  object 
 1   Location       145460 non-null  object 
 2   MinTemp        143975 non-null  float64
 3   MaxTemp        144199 non-null  float64
 4   Rainfall       142199 non-null  float64
 5   Evaporation    82670 non-null   float64
 6   Sunshine       75625 non-null   float64
 7   WindGustDir    135134 non-null  object 
 8   WindGustSpeed  135197 non-null  float64
 9   WindDir9am     134894 non-null  object 
 10  WindDir3pm     141232 non-null  object 
 11  WindSpeed9am   143693 non-null  float64
 12  WindSpeed3pm   142398 non-null  float64
 13  Humidity9am    142806 non-null  float64
 14  Humidity3pm    140953 non-null  float64
 15  Pressure9am    130395 non-null  float64
 16  Pressure3pm    130432 non-null  float64
 17  Cloud9am       89572 non-null   float64
 18  Cloud3pm       86102 non-null   float64
 19  Temp9am        143693 non-null  float64
 20  Temp3pm        141851 non-null  float64
 21  RainToday      142199 non-null  object 
 22  RainTomorrow   142193 non-null  object 
dtypes: float64(16), object(7)
memory usage: 25.5+ MB
```

Using a count, I managed to find that there were missing values in some of the features. RainTomorrow is also my target feature since the aim of this project is to predict if it is going to rain the next day.

```
[ ]  #Dropping date column since I only need to predict future weather.
     data.drop(['Date'], axis=1, inplace=True)
```

I chose to drop dates since these are recorded dates and have no effect on predicting the next day's weather.

```
[ ]  #Filling in the missing value of RainTomorrow using mode
     data['RainTomorrow']=data['RainTomorrow'].fillna(data['RainTomorrow'].mode()[0])
```

Due to some missing entries, I chose to fill in the values with mode. It is a considerably small set of missing values (3267 null values), which is 2% of the overall data (145,460).

Of the 7 Object Classes, I had chosen to convert them into numeric data using LabelEncoder

```
[ ]  #Converting (Yes/No) into binary (1/0) using LabelEncoder
     le = LabelEncoder()
     data['RainTomorrow'] = le.fit_transform(data['RainTomorrow'])
     data['RainToday'] = le.fit_transform(data['RainToday'])
     #Encoding other categorical Variables
     data['Location'] = le.fit_transform(data['Location'])
     data['WindDir9am'] = le.fit_transform(data['WindDir9am'])
     data['WindDir3pm'] = le.fit_transform(data['WindDir3pm'])
     data['WindGustDir'] = le.fit_transform(data['WindGustDir'])
```

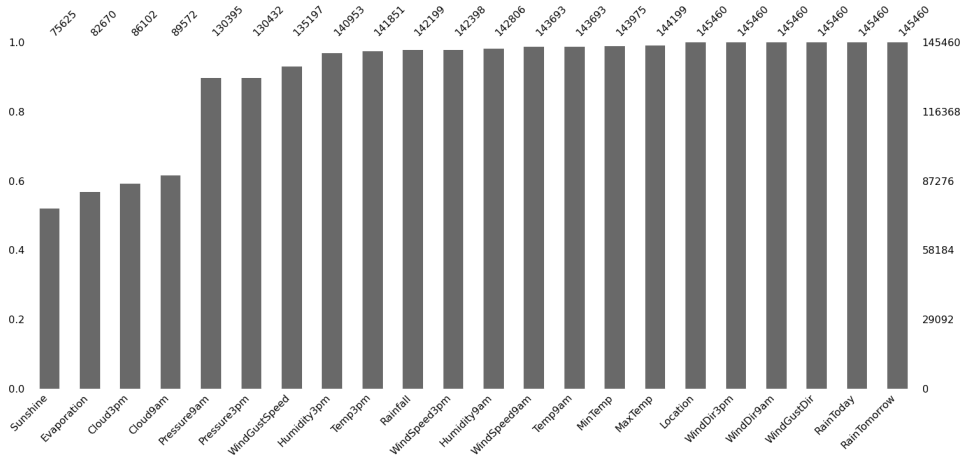
As stated, there were more missing values. I decided to do another count on missing values.

```
#Checking for null values
data.isnull().sum()
```

```
Location          0
MinTemp        1485
MaxTemp        1261
Rainfall       3261
Evaporation   62790
Sunshine      69835
WindGustDir       0
WindGustSpeed 10263
WindDir9am        0
WindDir3pm        0
WindSpeed9am   1767
WindSpeed3pm   3062
Humidity9am    2654
Humidity3pm    4507
Pressure9am   15065
Pressure3pm   15028
Cloud9am      55888
Cloud3pm      59358
Temp9am        1767
Temp3pm        3609
RainToday         0
RainTomorrow      0
dtype: int64
```

Visualizing the missing data, sorted by amount

```
#Using a bar chart to visualize the missing data
import missingno as msno
msno.bar(data, sort='ascending')
#ref: https://towardsdatascience.com/visualizing-missing-values-in-python-is-shockingly-easy-56ed5bc2e7ea
```



It is revealed that 'Sunshine', 'Evaporation', 'Cloud3pm' and 'Cloud9am' has a lot of null values. Since there are too many missing values, I chose to drop these features as it is ineffective for predicting the next day weather.

```
#Dropping the columns with too many missing data since they are ineffective for predicting the rain
data=data.drop(['Sunshine', 'Evaporation', 'Cloud3pm', 'Cloud9am'], axis=1)
data.columns
```

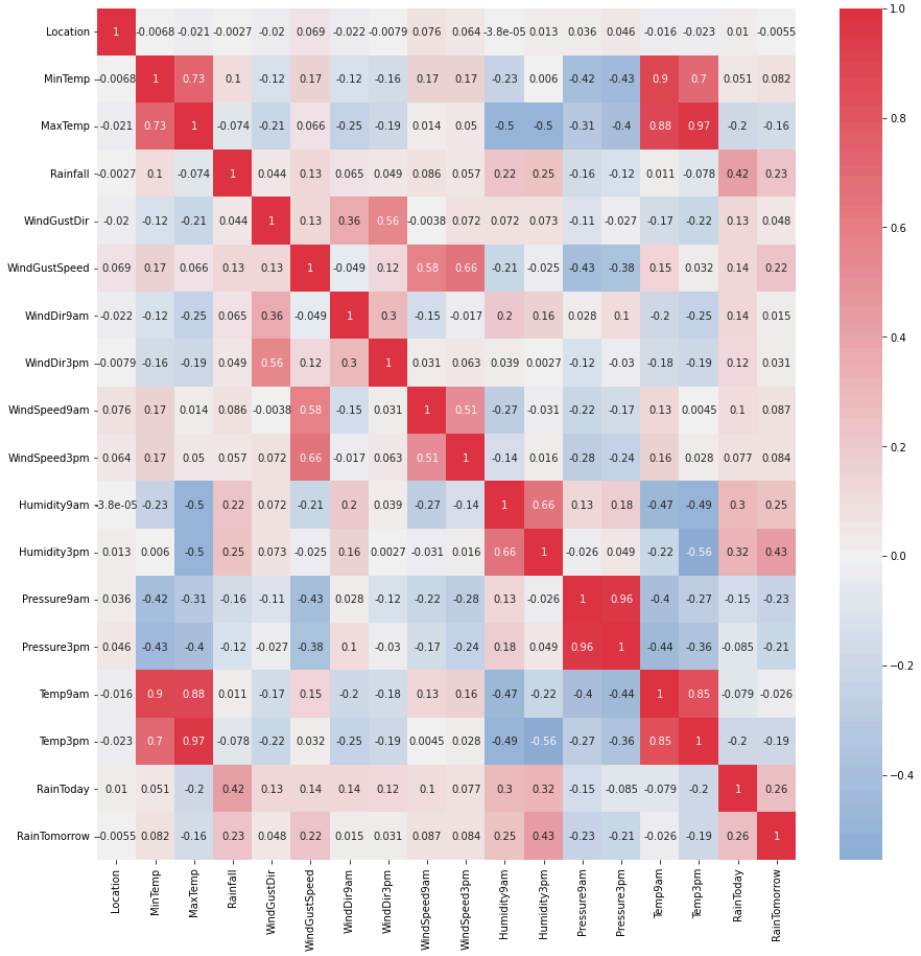
```
Index(['Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'WindGustDir',
       'WindGustSpeed', 'WindDir9am', 'WindDir3pm', 'WindSpeed9am',
       'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9am',
       'Pressure3pm', 'Temp9am', 'Temp3pm', 'RainToday', 'RainTomorrow'],
      dtype='object')
```

I will fill up the rest of the features using mean since I had converted them into numeric values, also the number of missing values is relatively small compared to existing data.

```
#Filling in missing values for the variables with mean
data['MinTemp']=data['MinTemp'].fillna(data['MinTemp'].mean())
data['MaxTemp']=data['MaxTemp'].fillna(data['MaxTemp'].mean())
data['Rainfall']=data['Rainfall'].fillna(data['Rainfall'].mean())
data['WindGustSpeed']=data['WindGustSpeed'].fillna(data['WindGustSpeed'].mean())
data['WindSpeed9am']=data['WindSpeed9am'].fillna(data['WindSpeed9am'].mean())
data['WindSpeed3pm']=data['WindSpeed3pm'].fillna(data['WindSpeed3pm'].mean())
data['Humidity9am']=data['Humidity9am'].fillna(data['Humidity9am'].mean())
data['Humidity3pm']=data['Humidity3pm'].fillna(data['Humidity3pm'].mean())
data['Pressure9am']=data['Pressure9am'].fillna(data['Pressure9am'].mean())
data['Pressure3pm']=data['Pressure3pm'].fillna(data['Pressure3pm'].mean())
data['Temp9am']=data['Temp9am'].fillna(data['Temp9am'].mean())
data['Temp3pm']=data['Temp3pm'].fillna(data['Temp3pm'].mean())
```

Afterwards, I used a Correlation Matrix to visualize which feature correlates with target feature 'RainTomorrow'.

```
#Using corrmat to visualize and find out which attribute will be correlated to RainTomorrow
plt.figure(figsize=(15,15))
corrmat = data.corr()
cmap = sns.diverging_palette(250, 10, s=80, l=50, n=8, as_cmap=True)
sns.heatmap(corrmat, annot=True, cmap=cmap, center=0)
```



Looking at the matrix table, I observed that Humidity3pm (0.43), Humidity9am (0.25), RainToday (0.26) and Rainfall (0.23) have the highest correlation with RainTomorrow.

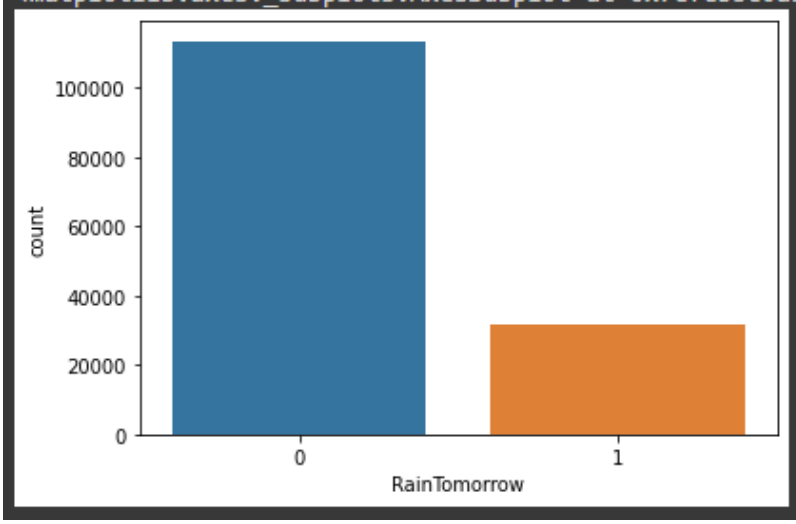
C) Preparing the dataset for training

```
#Defining features and target
X = data.drop('RainTomorrow', axis=1)
y = data['RainTomorrow']
print(y.value_counts())
```

```
0    113583
1     31877
Name: RainTomorrow, dtype: int64
```

```
#countplot to visualize the results of Yes/No in target variable
sns.countplot(x = data["RainTomorrow"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7efeb3c0a190>
```



Using a count plot to visualize the data imbalance, I realised that I had to use a sampler to balance the data. I used SMOTE to over sample the data.

```
#Train test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
#Using SMOTE for sampling
from imblearn.over_sampling import SMOTE
sm = SMOTE()
X_train_up, y_train_up = sm.fit_resample(X_train, y_train)
print(y_train_up.value_counts())
```

```
1    90911
0    90911
Name: RainTomorrow, dtype: int64
```

D) Training the model

For this project, I had chosen to use the following Classification techniques:

- Logistic Regression
- Random Forest
- K-Nearest Neighbours
- Gradient Boosting

```
#Training the model
model = LogisticRegression(max_iter= 500)
model.fit(X_train_up, y_train_up)
y_pred = model.predict(X_test)
#Scoring the accuracy of LogisticRegression
score = accuracy_score(y_test, y_pred)
ps = precision_score(y_test,y_pred)
#Printing the accuracy
print('Accuracy :', score)
print('F Score :', f1_score(y_test, y_pred))
print('Precision Score :', ps)
```

```
Accuracy : 0.77419909253403
F Score : 0.5913021837864743
Precision Score : 0.49228219206464313
```

```
#Training the model with RandomForestClassifier
model_rf = RandomForestClassifier()
model_rf.fit(X_train_up, y_train_up)
y_pred = model_rf.predict(X_test)
score = accuracy_score(y_test, y_pred)
ps = precision_score(y_test,y_pred)
print('Accuracy :', score)
print('F Score :', f1_score(y_test, y_pred))
print('Precision Score :', ps)
```

```
Accuracy : 0.8405059810257115
F1 Score : 0.6309258670060451
Precision Score : 0.6446684005201561
```

```
#Using KNN to train the model
knn = KNeighborsClassifier(n_neighbors = 8, metric = 'minkowski')
knn.fit(X_train_up, y_train_up)
y_pred = knn.predict(X_test)
score = accuracy_score(y_test, y_pred)
ps = precision_score(y_test,y_pred)
print('Accuracy :', score)
print('F Score :', f1_score(y_test, y_pred))
print('Precision Score :', ps)
```

```
Accuracy : 0.7799395022686649
F Score : 0.5888246628131021
Precision Score : 0.5009836065573771
```

```
#Using Gradient Boosting Classifier
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier()
gb.fit(X_train_up, y_train_up)
y_pred = gb.predict(X_test)
score = accuracy_score(y_test, y_pred)
ps = precision_score(y_test, y_pred)
print('Accuracy :', score)
print('F Score :', f1_score(y_test, y_pred))
print('Precision Score :', ps)
```

```
Accuracy : 0.8160318988037949
F Score : 0.6168933428775949
Precision Score : 0.5707284768211921
```

E) Conclusion

From the results above, I observed that Random Forest Classifier performed the best among the four. It had the highest Accuracy (84.07%) and Precision (64.48%) with Gradient Boosting close behind, Logistic Regression and KNN performed poorly.

A.2.1 Regression Using Python – Revenue vs Temperature

- Summary

The dataset 'IceCreamData.csv' contains Ice Cream Revenue against recorded Temperature. Doing a simple exploration reveals that there are only 2 columns: Temperature and Revenue.

```
#Looking at the dataset
data.head()
```

	Temperature	Revenue
0	24.566884	534.799028
1	26.005191	625.190122
2	27.790554	660.632289
3	20.595335	487.706960
4	11.503498	316.240194

- Initial Exploratory into the dataset

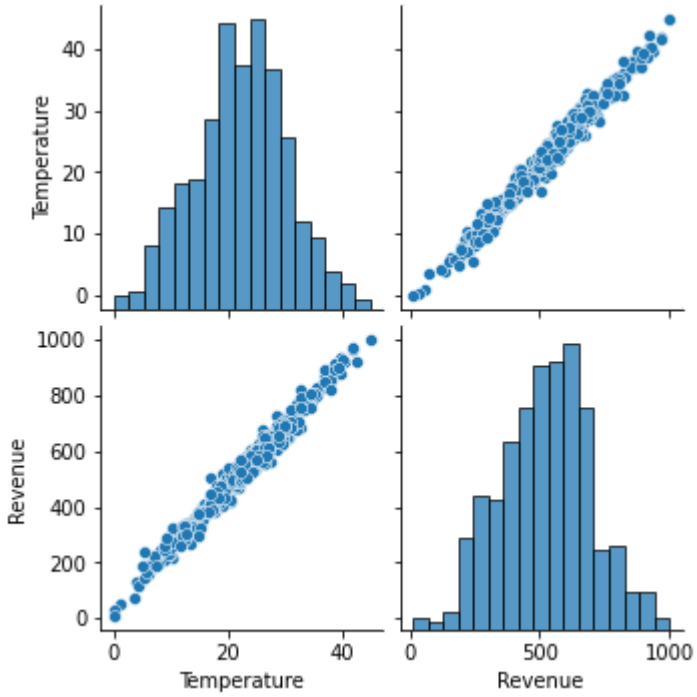
```
#info for Dtype and entry counts
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Temperature  500 non-null    float64
 1   Revenue      500 non-null    float64
dtypes: float64(2)
memory usage: 7.9 KB
```

Through the initial exploration, I found out that there are only 2 columns, both of which are numerical values (float64). There is a total of 500 values and there are no missing values in this dataset.

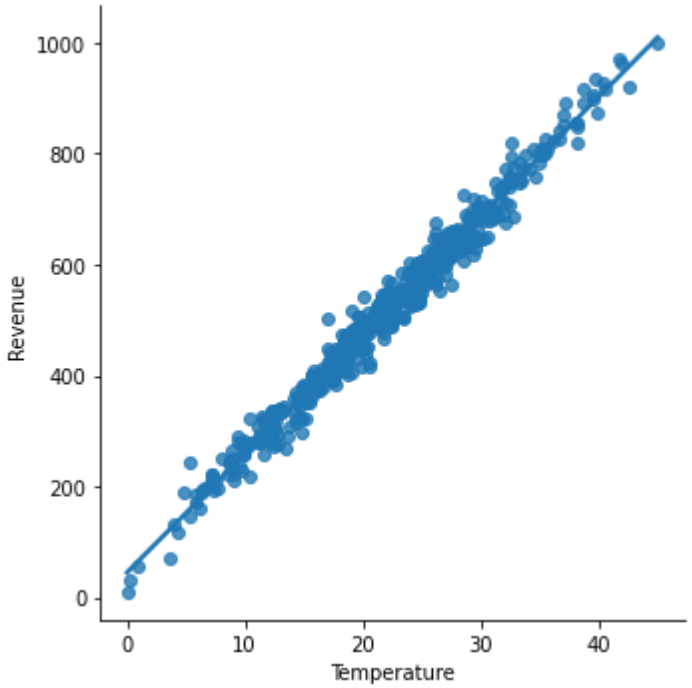
- Simple Linear Regression

```
#Visualising the values
sns.pairplot(data)
```



With the aim of this project, I need to specify the target feature (Revenue, y) against Temperature (x).

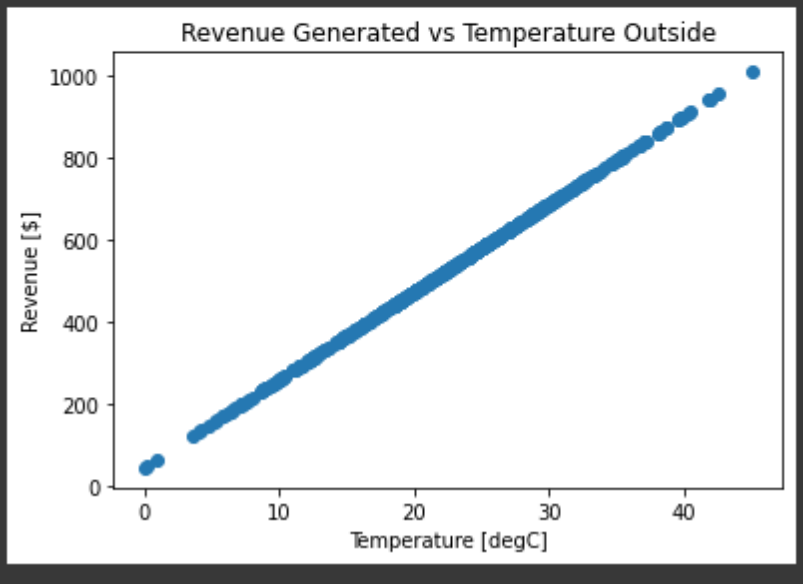
```
#Specifying  X and Y linear plot
sns.lmplot(x = 'Temperature', y = 'Revenue', data=data)
X = data[['Temperature']]
y = data[['Revenue']]
```



```
#Using Linear Regression for model
lr = LinearRegression(fit_intercept = True)
lr.fit(X, y)
#Printing coefficient to estimate relationship between a predictor variable and the response
print('LinearRegression Coefficient :', lr.coef_)
#Printing intercept to estimate the value of a dependent variable based on the values of the independent var
print('LinearRegression Intercept :', lr.intercept_)
```

```
LinearRegression Coefficient : [[21.44362551]]
LinearRegression Intercept : [44.83126709]
```

```
#Visualizing the result
plt.scatter(X, y_pred)
plt.xlabel('Temperature [degC]')
plt.ylabel('Revenue [$]')
plt.title('Revenue Generated vs Temperature Outside')
plt.show()
```



After visualizing the prediction model using Linear Regression, I observed that Revenue is directly proportional to the Temperature.

```
[ ] #Train test split model for testing
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(data[['Temperature']], data[['Revenue']], test_size=0.2, random_state=42)

[ ] #printing MSE, MAE and R2 Score
    y_pred = lr.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print("Mean Squared Error: ", mse)
    print("Mean Absolute Error: ", mae)
    print("R-squared: ", r2)

    Mean Squared Error:  646.9503783651506
    Mean Absolute Error:  19.085715031573127
    R-squared:  0.9773482908856934
```

- Conclusion

  Testing the model, the results of the scores as shown. The Mean Absolute error shows a $19.08, showing little variance between the testing and prediction dataset with a high R^2 value of 97.73%. This proves that there is a linear relationship between Temperature and revenue generated.

A.2.2 Regression Using Python – Prices of Cars vs Specifications

- Summary

This dataset (“CarPrice_Assignment.csv”) contains records of car prices compared to the car production specifications.

- Initial Exploration of the dataset

  Looking at the dataset, I can see that there is a total of 205 entries, 26 columns; with 16 numerical and 10 object features. There seem to be no missing values in the dataset.

```
#Looking for Dtypes values and count
data.info()
```

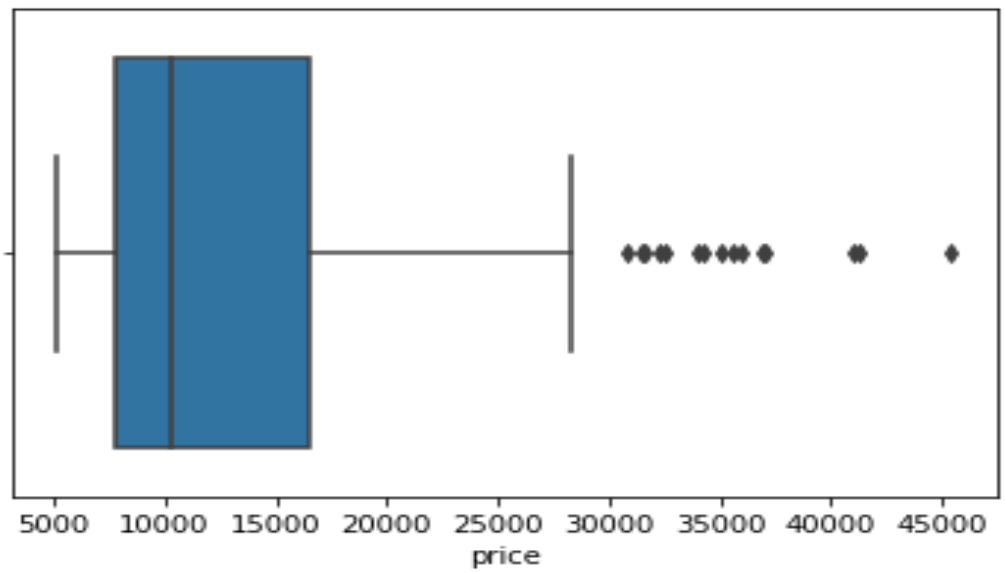
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   car_ID            205 non-null    int64  
 1   symboling         205 non-null    int64  
 2   CarName           205 non-null    object 
 3   fueltype          205 non-null    object 
 4   aspiration        205 non-null    object 
 5   doornumber        205 non-null    object 
 6   carbody           205 non-null    object 
 7   drivewheel        205 non-null    object 
 8   enginelocation    205 non-null    object 
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64  
 14  enginetype        205 non-null    object 
 15  cylindernumber    205 non-null    object 
 16  enginesize        205 non-null    int64  
 17  fuelsystem        205 non-null    object 
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64  
 22  peakrpm           205 non-null    int64  
 23  citympg           205 non-null    int64  
 24  highwaympg        205 non-null    int64  
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

```
#Looking for missing values
data.isnull().sum()
```

```
car_ID              0
symboling           0
CarName             0
fueltype            0
aspiration          0
doornumber          0
carbody             0
drivewheel          0
enginelocation      0
wheelbase           0
carlength           0
carwidth            0
carheight           0
curbweight          0
enginetype          0
cylindernumber      0
enginesize          0
fuelsystem          0
boreratio           0
stroke              0
compressionratio    0
horsepower          0
peakrpm             0
citympg             0
highwaympg          0
price               0
dtype: int64
```

Using a box plot to visualize the distribution of the car prices

```
#Visualising the price values
sns.boxplot(data['price'])
plt.figure(figsize=(10,10))
```



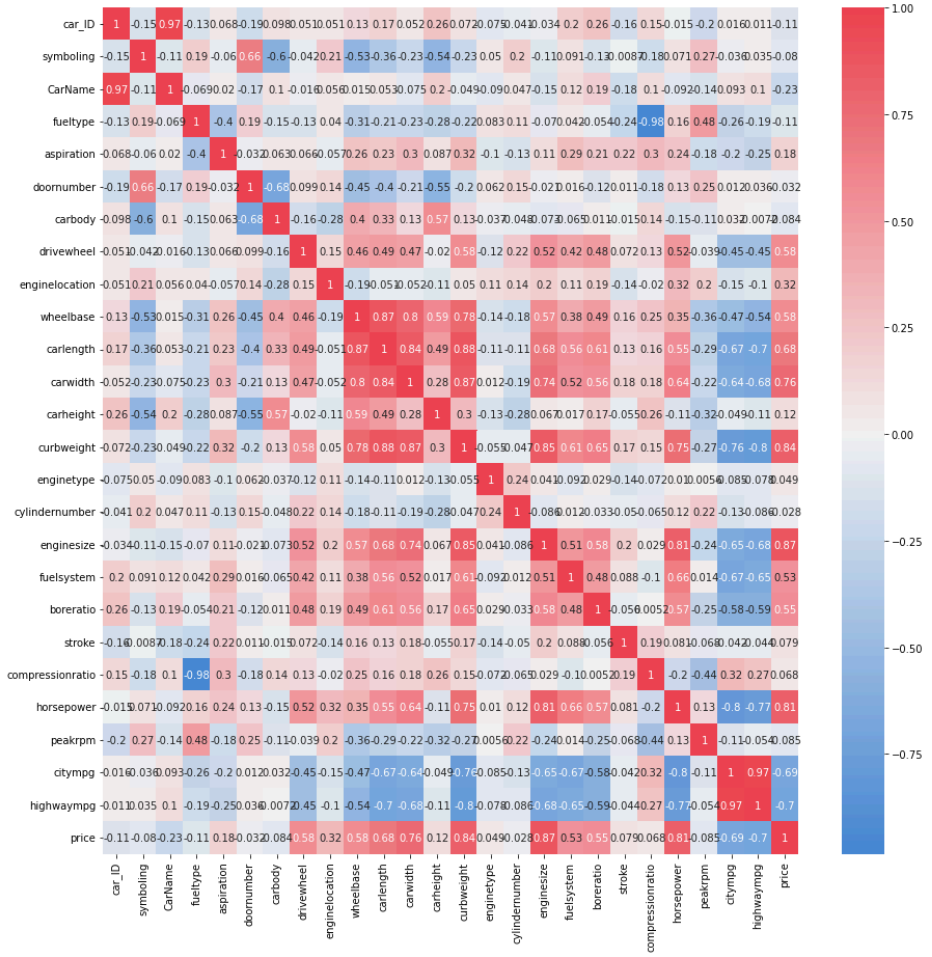
As seen above, the distribution shows that car prices vary mainly within 5,000 to 20,000 with a few outliers.

- Training the model

  To train the Multi-Linear Regression, I had to find out which feature affects the price of the cars. To do so, I had to convert all the object Dtypes into Numerical values before using a Correlation Matrix.

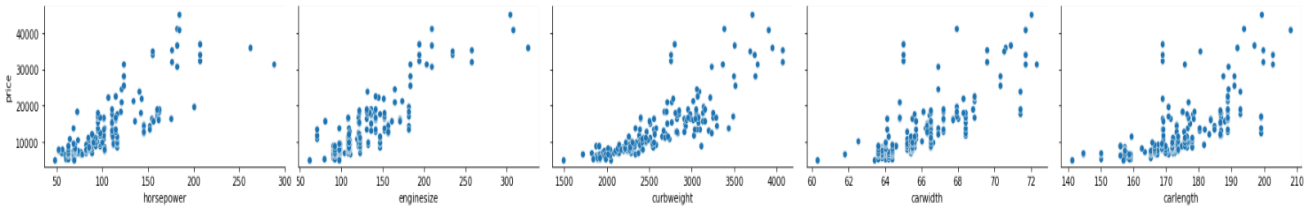
```
#Using leelencoder to transform the categorical values
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data['fuelsystem']=le.fit_transform(data['fuelsystem'])
data['cylindernumber']=le.fit_transform(data['cylindernumber'])
data['enginetype']=le.fit_transform(data['enginetype'])
data['enginelocation']=le.fit_transform(data['enginelocation'])
data['drivewheel']=le.fit_transform(data['drivewheel'])
data['carbody']=le.fit_transform(data['carbody'])
data['doornumber']=le.fit_transform(data['doornumber'])
data['aspiration']=le.fit_transform(data['aspiration'])
data['fueltype']=le.fit_transform(data['fueltype'])
data['CarName']=le.fit_transform(data['CarName'])
```

```
#Using corrmat to visualize which feature/attributes strongly affects price
plt.figure(figsize=(15,15))
corrmat = data.corr()
cmap = sns.diverging_palette(250, 10, s=80, l=55, n=9, as_cmap=True)
sns.heatmap(corrmat, annot=True, cmap=cmap, center=0)
```



From the correlation matrix above, I observed that Horsepower, Engine Size, Curb weight, Car width and Car length have the highest correlation values to Price.

```
#Scatterplot to visualize the dataset's outliers influencing the price
sns.pairplot(data, x_vars=["horsepower", "enginesize", "curbweight", "carwidth", "carlength"], y_vars=["price"], aspect=2, kind="scatter")
plt.show()
```



Since there are multiple attributes that affects Car prices, using a multi-linear regression is suitable for predicting this model. With the target: Car Prices (y) I can specify the X and Y from here.

```
#Defining X and Y for the model selection
X = data.drop(['car_ID','CarName','price'], axis=1)
y = data['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
#LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
#Importing metrics to find the accuracy
from sklearn import metrics
print("MSR :", metrics.mean_squared_error(y_test, y_pred))
print("MAE :", metrics.mean_absolute_error(y_test, y_pred))
print("RMSE :", np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
#R squared score
from sklearn.metrics import r2_score
r2 = r2_score(y_test,y_pred)
print("R2 :", r2)
```

```
MSR : 15916389.72543936
MAE : 2526.407450143416
RMSE : 3989.5350262203942
R2 : 0.7983838478445084
```

- Conclusion

  Through the test, I can conclude that the Linear Regression model has an accuracy of 79.83% with a RMSE of $3,989.53, showing little variation between the test and prediction.

Task B – Unsupervised Learning

B.1 Cluster Analysis – Customer Segment Analysis

- Summary

  This dataset contains information regarding the customers' spending scores relative to their economic standing. The aim of this project is to understand the target customers so that the marketing team can plan accordingly.

- Initial Exploration

  Looking through "Mall_Customers.csv", I can see that there is a total of 200 rows and 5 columns. Of the 5 columns, 4 are numeric and 1 object class.

```
#Getting the shape
data.shape
```

```
(200, 5)
```

```
#Getting info of dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

  Further exploration yields that there are no missing values (counting null values)

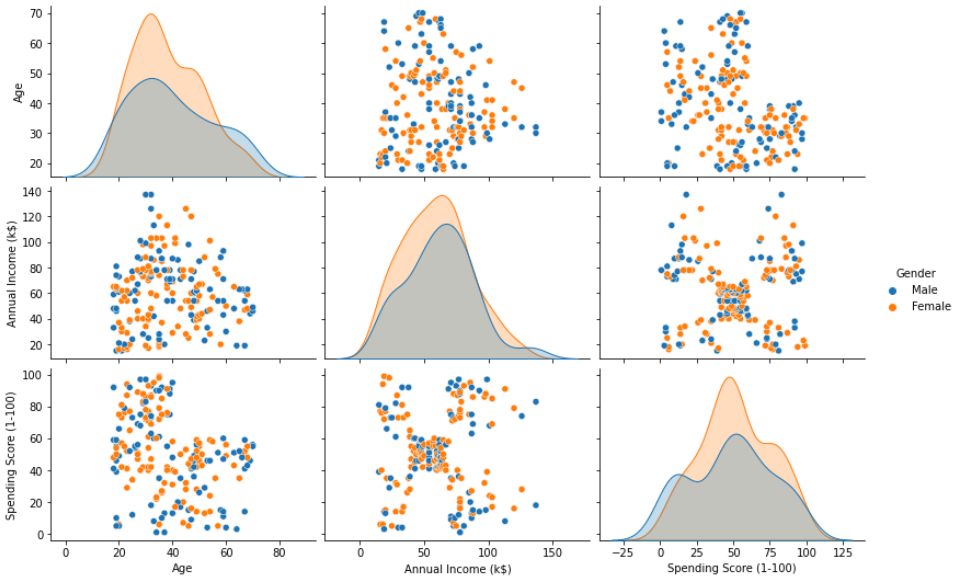
```
#looking for null values
data.isnull().sum()
```

```
CustomerID              0
Gender                  0
Age                     0
Annual Income (k$)      0
Spending Score (1-100)  0
dtype: int64
```

- Data pre-processing

  I used Gender initially to see if there is a difference in spending habits.

```
#Visualizing the data
sns.pairplot(data.drop('CustomerID', axis=1), hue='Gender', aspect=1.5)
plt.show()
```



Looking through the graphs however, Gender does not have a direct relation to segmenting customers. This allows me to drop ‘Gender’ and focus on other features.

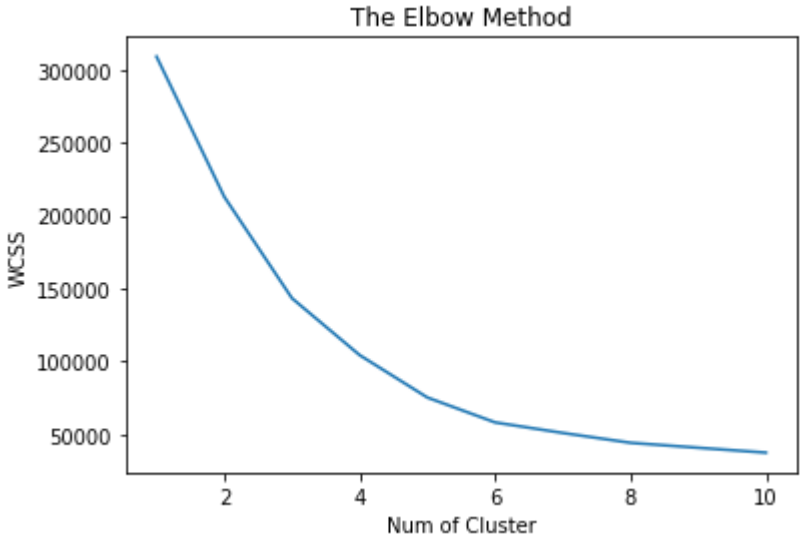
CustomerID only contains unique values and thus should not be used for training the model, I dropped it as well.

```
#Dropping gender
X = data.drop(['CustomerID', 'Gender'], axis=1, inplace=True)
```

With that, I am left with Age, Annual Income and Spending Score. But before training the model, I need to find out how many clusters are there in this dataset. I made use of the elbow method to identify the clusters.

```
#Using Kmeans to find out the clusters
wcss = []
for i in range(1,11):
  kmeans = KMeans(n_clusters=i, init="k-means++", random_state=0)
  kmeans.fit(data)
  wcss.append(kmeans.inertia_)
```

```
#Plot the graph
plt.plot(range(1,11), wcss)
plt.title("The Elbow Method")
plt.xlabel("Num of Cluster")
plt.ylabel("WCSS")
plt.show()
```



From the graph, the bends at 3 and 5 are the focus for me. I then separated the customers to the clusters.

```
#fitting KMeans to 5 clusters model
kmeans = KMeans(n_clusters=5)
kmeans_fmodel = kmeans.fit(data)
y_kmean = kmeans_fmodel.predict(data)
data['Clusters'] = y_kmean
```

```
data.head()
```

	Age	Annual Income (k$)	Spending Score (1-100)	Clusters
0	19	15	39	3
1	21	15	81	4
2	20	16	6	3
3	23	16	77	4
4	31	17	40	3

+ Code + Text

```
data.tail()
```

	Age	Annual Income (k$)	Spending Score (1-100)	Clusters
195	35	120	79	0
196	45	126	28	1
197	32	126	74	0
198	32	137	18	1
199	30	137	83	0

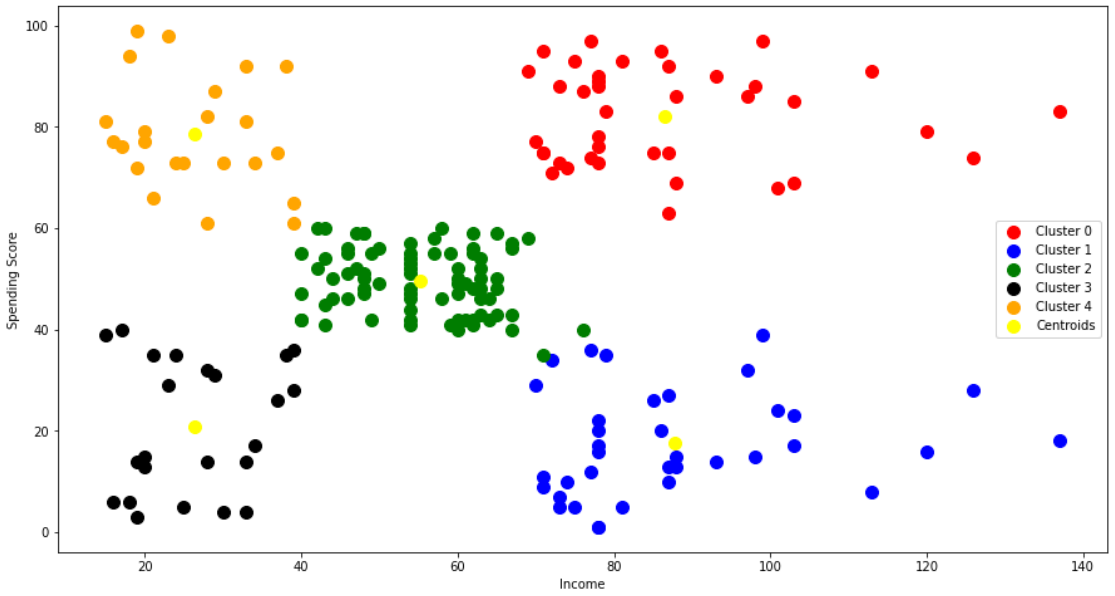
- Training the model

  For this project, I used the following Clustering Analysis:

  1. K-Means

  2. DBSCAN

```
#Scatterplot to visualize the cluster
X = data.iloc[:,[1,2]].values
plt.figure(figsize=(15,8))
#Plotting the scatterplot
plt.scatter(X[y_kmean==0,0], X[y_kmean==0,1], s=100, c='red', label="Cluster 0")
plt.scatter(X[y_kmean==1,0], X[y_kmean==1,1], s=100, c='blue', label="Cluster 1")
plt.scatter(X[y_kmean==2,0], X[y_kmean==2,1], s=100, c='green', label="Cluster 2")
plt.scatter(X[y_kmean==3,0], X[y_kmean==3,1], s=100, c='black', label="Cluster 3")
plt.scatter(X[y_kmean==4,0], X[y_kmean==4,1], s=100, c='orange', label="Cluster 4")
#Plotting the centroids
plt.scatter(kmeans_fmodel.cluster_centers_[:,1], kmeans_fmodel.cluster_centers_[:,2], s=100, c="yellow", label='Centroids')
plt.xlabel("Income")
plt.ylabel("Spending Score")
plt.legend()
plt.show()
```



Through K-Means Analysis, I can clearly see the 5 clusters and the centroid:

Cluster 0 is High Income, high spending.

Cluster 2 is High Income, Low Spending.

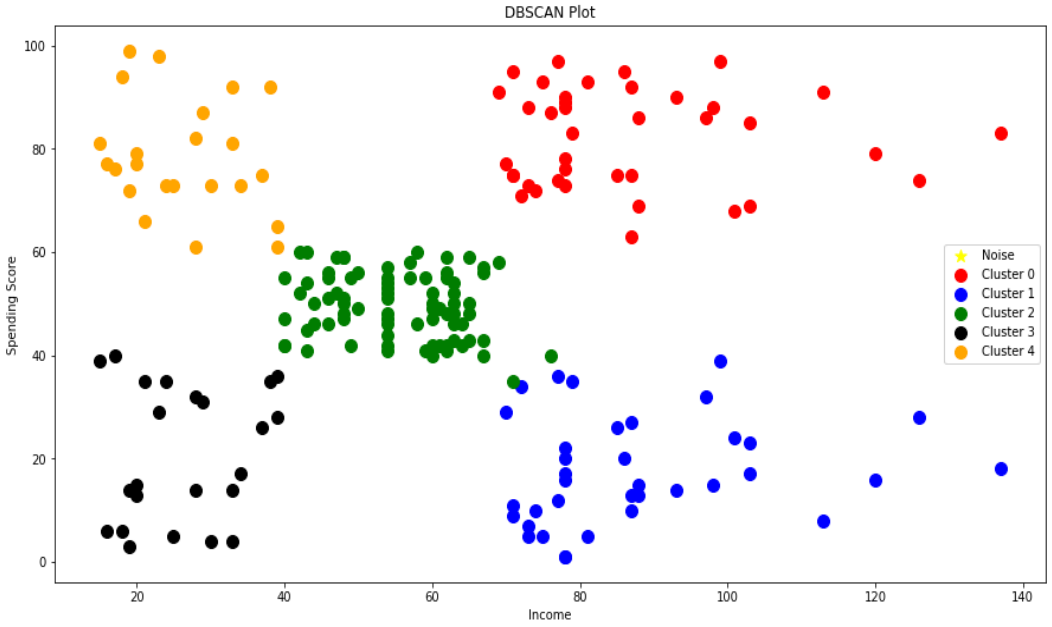
Cluster 1 is Low Income, low spending.

Cluster 3 is Mid Income, mid spending.

Cluster 4 is Low Income, High Spending.

Using DBSCAN,

```
#Using Density Based Clustering DBSCAN
db = DBSCAN(eps= 12, min_samples=8, metric='euclidean')
X = data.iloc[:,[1,2]].values
plt.figure(figsize=(15,8))
plt.scatter(X[y_kmean ==-1,0], X[y_kmean ==-1,1], s=100, c='yellow', marker="*", label='Noise')
plt.scatter(X[y_kmean==0,0], X[y_kmean==0,1], s=100, c='red', label="Cluster 0")
plt.scatter(X[y_kmean==1,0], X[y_kmean==1,1], s=100, c='blue', label="Cluster 1")
plt.scatter(X[y_kmean==2,0], X[y_kmean==2,1], s=100, c='green', label="Cluster 2")
plt.scatter(X[y_kmean==3,0], X[y_kmean==3,1], s=100, c='black', label="Cluster 3")
plt.scatter(X[y_kmean==4,0], X[y_kmean==4,1], s=100, c='orange', label="Cluster 4")
plt.title("DBSCAN Plot")
plt.xlabel("Income")
plt.ylabel("Spending Score")
plt.legend()
plt.show()
```



Looking at the DBSCAN graph, it is similar to K-Means. With the clusters indicating similar results

- Conclusion

It seems that majority of the customers are clustered around Cluster 2 in both graphs. The mall can use this opportunity to target this cluster to get this group to visit the mall regularly through marketing campaigns and discounts. Furthermore, looking at Cluster 1, the mall can stand to attract them to spend more.

B.2.1 Association Rules – Online Retail

- Summary

The aim of this project is to perform at least 1 association rule analysis and provide detailed interpretation of the findings

- Initial Exploration

From initial exploration, there are a total of 541,909 rows and 8 columns

```
data.shape

(541909, 8)
```

Among the 8 columns, there is a total of 4 object classes, 3 numerical and 1 date class.

```
#Getting info from column types and entries
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype         
---  ------       --------------   -----         
 0   InvoiceNo    541909 non-null  object        
 1   StockCode    541909 non-null  object        
 2   Description  540455 non-null  object        
 3   Quantity     541909 non-null  int64         
 4   InvoiceDate  541909 non-null  datetime64[ns]
 5   UnitPrice    541909 non-null  float64       
 6   CustomerID   406829 non-null  float64       
 7   Country      541909 non-null  object        
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB
```

Further exploration reveals that there are many missing values in CustomerID and some in Description. Since CustomerID has too many missing values and only unique values, I will drop this column and then start looking into the cancelled invoices.

```
#Looking at cancelled invoice
data[data["InvoiceNo"].str.startswith('C', na=False)]
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	Country
221	C558716	10133	COLOURING PENCILS BROWN TUBE	-10	2011-07-01 13:22:00	0.42	United Kingdom
422	C571707	10135	COLOURING PENCILS BROWN TUBE	-1	2011-10-18 15:33:00	1.25	United Kingdom
602	C575257	11001	ASSTD DESIGN RACING CAR PEN	-1	2011-11-09 11:59:00	1.69	United Kingdom
603	C569682	11001	ASSTD DESIGN RACING CAR PEN	-2	2011-10-05 14:33:00	1.69	United Kingdom
604	C568412	11001	ASSTD DESIGN RACING CAR PEN	-1	2011-09-27 10:54:00	1.69	United Kingdom
...	...	...	...	...	...	...	...
541902	C544580	S	SAMPLES	-1	2011-02-21 14:25:00	20.55	United Kingdom
541903	C544580	S	SAMPLES	-1	2011-02-21 14:25:00	29.99	United Kingdom
541904	C544580	S	SAMPLES	-1	2011-02-21 14:25:00	9.74	United Kingdom
541905	C537581	S	SAMPLES	-1	2010-12-07 12:03:00	12.95	United Kingdom
541906	C537581	S	SAMPLES	-1	2010-12-07 12:03:00	52.00	United Kingdom

9288 rows × 7 columns

Since I want to find out the association between purchased items, I can safely drop the cancelled invoices.

```
n_data = data[~data['InvoiceNo'].str.startswith('C', na=False)]
n_data.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	Country
0	551429	10002	NaN	-3	2011-04-28 15:05:00	0.00	United Kingdom
1	550452	10002	INFLATABLE POLITICAL GLOBE	1	2011-04-18 12:56:00	0.85	United Kingdom
2	550272	10002	INFLATABLE POLITICAL GLOBE	62	2011-04-15 12:14:00	0.85	United Kingdom
3	548714	10002	INFLATABLE POLITICAL GLOBE	2	2011-04-03 15:07:00	0.85	United Kingdom
4	548702	10002	INFLATABLE POLITICAL GLOBE	4	2011-04-03 11:36:00	0.85	United Kingdom

Doing an exploration after removing cancelled orders

```
n_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 532621 entries, 0 to 541908
Data columns (total 7 columns):
 #   Column       Non-Null Count   Dtype         
---  ------       --------------   -----         
 0   InvoiceNo    532621 non-null  object        
 1   StockCode    532621 non-null  object        
 2   Description  531167 non-null  object        
 3   Quantity     532621 non-null  int64         
 4   InvoiceDate  532621 non-null  datetime64[ns]
 5   UnitPrice    532621 non-null  float64       
 6   Country      532621 non-null  object        
dtypes: datetime64[ns](1), float64(1), int64(1), object(4)
memory usage: 32.5+ MB
```

```
n_data.shape
```

```
(532621, 7)
```

I then sorted the Quantity against the Countries

```
#Sorting the values
country_data.sort_values('TotalPrice', ascending=False, inplace=True, ignore_index=True)
country_data
```

What I found out was that United Kingdom was the largest client with a total of 487,622 products bought.

- Association Rules

Before beginning the model, I had to replace the Description class from object to string.

```
#Changing description from object class to str
n_data = n_data[~ n_data['Description'].isna()]
n_data['Description'] = n_data['Description'].astype(str)
```

Afterwards, I applied the new string by grouping it with only purchases from United Kingdom.

```
#applying the n_data into country United Kingdom
uk_data = n_data.groupby(['InvoiceNo'])['Description'].apply(list).to_frame().reset_index()
```

```
uk_data
```

	InvoiceNo	Description
0	536365	[GLASS STAR FROSTED T-LIGHT HOLDER, SET 7 BABU...
1	536366	[HAND WARMER RED POLKA DOT, HAND WARMER UNION ...
2	536367	[HOME BUILDING BLOCK WORD, LOVE BUILDING BLOCK...
3	536368	[YELLOW COAT RACK PARIS FASHION, RED COAT RACK...
4	536369	[BATH BUILDING BLOCK WORD]
...	...	...
22059	581586	[DOORMAT RED RETROSPOT, RED RETROSPOT ROUND CA...
22060	581587	[BAKING SET 9 PIECE RETROSPOT , CHILDRENS APRO...
22061	A563185	[Adjust bad debt]
22062	A563186	[Adjust bad debt]
22063	A563187	[Adjust bad debt]

22064 rows × 2 columns

With that I now setup association rules using this new list and printed the results

```
#Setting up the association rule
association_rules = apriori(transaction, min_support=0.01, min_confidence=0.7, min_lift=3, min_length=2)

#Converting the associations to list
rules = list(association_rules)
```

```
#Printing the rules, support, conmfidence and lift
number = 1
for rule in rules:
    pair = rule[0]
    items = [x for x in pair]

    print("Rule #{}".format(number) + "\n")
    print("Antecedent: {} => Consequent: {}".format(items[0], items[1]) + "\n")
    print("Support: {}".format(str(rule[1])) + "\n")
    print("Confidence: {}".format(str(rule[2][0][2])) + "\n")
    print("Lift: {}".format(str(rule[2][0][3])) + "\n")
    print("==================================================="+"\n")

    print("Rule #{}".format(number))
    print("Antecedent: {} => Consequent: {}".format(items[0], items[1]))
    print("Support: {}".format(str(rule[1])))
    print("Confidence: {}".format(str(rule[2][0][2])))
    print("Lift: {}".format(str(rule[2][0][3])))
    print("===================================================")
    number += 1
```

- Conclusion

The results are long, but with the rules set to showing only with a confidence of at least 70%, I can infer that these are the pairs of items bought.

B.2.2 Association Rules – Titanic Survival Rates

- Summary

  This dataset contains the records of passengers aboard Titanic and the survival rates.

- Initial Exploration

Initial exploration shows that there are a total of 891 values and 12 columns, there are missing values in Age, Cabin and Embarked.

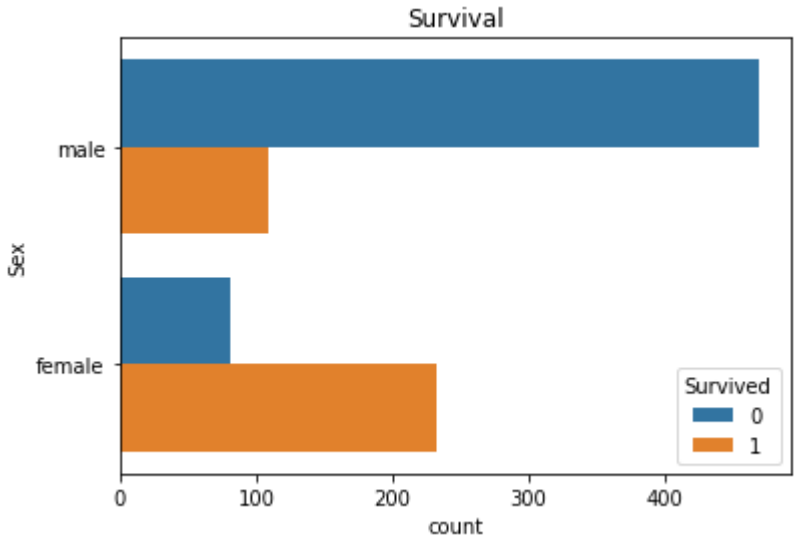
```
titanic.info()
```

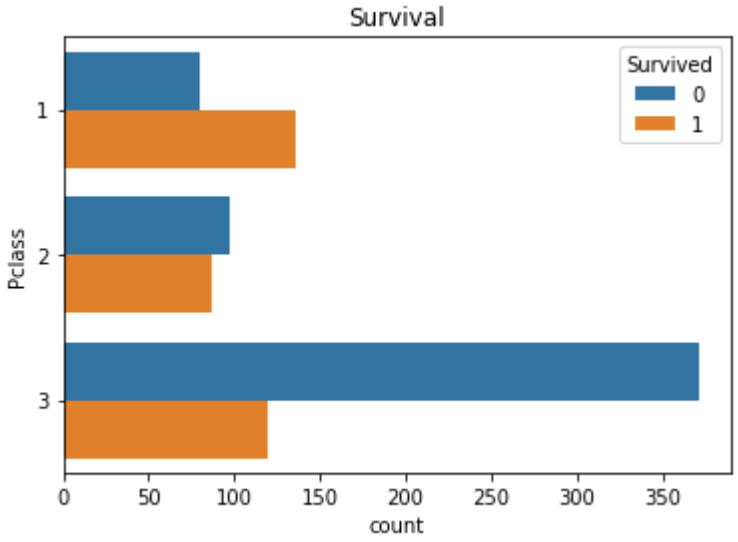
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
titanic.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
#visualzing the data
for x in ['Pclass', 'Sex']:
  grp = sns.countplot(y=x, data=titanic)
  plt.ylabel(x)
  plt.title('Survival')
  plt.show()
```





From the count plot, I observed that there are more female survivals than there are males in regardless of Passenger class.

```
#Setting up the association rules
association_rules = apriori(titanic, min_support=0.01, min_confidence=0.7, min_lift=3, min_length=2)

#Converting the associations to list
rules = list(association_rules)
```

+ Code + Text

```
#Printing the numbers
print(len(rules))
```

```
873
```

```
number = 1
for rule in rules:
    pair = rule[0]
    items = [x for x in pair]

    print("Rule #{}".format(number) + "\n")
    print("Antecedent: {} => Consequent: {}".format(items[0], items[1]) + "\n")
    print("Support: {}".format(str(rule[1])) + "\n")
    print("Confidence: {}".format(str(rule[2][0][2])) + "\n")
    print("Lift: {}".format(str(rule[2][0][3])) + "\n")
    print("==================================================="+"\n")

    print("Rule #{}".format(number))
    print("Antecedent: {} => Consequent: {}".format(items[0], items[1]))
    print("Support: {}".format(str(rule[1])))
    print("Confidence: {}".format(str(rule[2][0][2])))
    print("Lift: {}".format(str(rule[2][0][3])))
    print("===================================================")
    number += 1
```

References

Ayres de Campos et al. (2000) SisPorto 2.0 A Program for Automated Analysis of Cardiotocograms. J Matern Fetal Med 5:311-318

Fetal Health Classification | Kaggle

Random Forest Classifier, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011

sklearn.ensemble.RandomForestClassifier

Jeeteshgavande30, Geeksforgeeks.org

Box Plot in Python

Decision Tree Classifier, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Logistic Regression, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

K-Nearest Algorithm retrieved from:

https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

Correlation Matrix retrieved from:

https://datatofish.com/correlation-matrix-pandas/

Linear Regression, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Mean-Squared Error retrieved from:

https://www.simplilearn.com/tutorials/statistics-tutorial/mean-squared-error

Absolute Error & Mean Absolute Error retrieved from:

https://www.statisticshowto.com/absolute-error/

R-Squared Score, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html

Kevin Arvai - K-Means Clustering in Python: A Practical Guide retrieved from

https://realpython.com/k-means-clustering-python/

Anisha Garg (2018) - Complete guide to Association Rules (1/2) retrieved from

https://towardsdatascience.com/association-rules-2-aa9a77241654

Chonyy (2020), Apriori — Association Rule Mining In-depth Explanation and Python Implementation retrieved from

https://towardsdatascience.com/apriori-association-rule-mining-explanation-and-python-implementation-290b42afdfc6