

**School of InfoComm Technology**

**Applied Analytics Assignment**

Diploma in Cybersecurity & Digital Forensics

Diploma in Data Science

Diploma in Information Technology

Year 2/3 (2023/2024), Semester 3/5

**INDIVIDUAL ASSIGNMENT 1**

(30% of Applied Analytics Module)

**Deadline for Submission:**

**10th Jun 2023 (Saturday), 23:59 HRS**

|  |  |
| --- | --- |
| Tutorial Group: | P05 |
| Student Name: | Lim Wee Liang Kelven |
| Student Number: | S10221788K |

**Penalty for late submission:**

10% of the marks will be deducted every day after the deadline.

**NO** submission will be accepted after 17th Jun 2023, 23:59.

Contents

[Overview 3](#_Toc137206394)

[Data Exploration 4](#_Toc137206395)

[Creating a Calculated Column 4](#_Toc137206396)

[Removing Null Values 4](#_Toc137206397)

[Drop Redundant Columns 6](#_Toc137206398)

[LabelEncoder 7](#_Toc137206399)

[make 9](#_Toc137206400)

[fuel\_type 10](#_Toc137206401)

[transmission 10](#_Toc137206402)

[color 11](#_Toc137206403)

[owner 11](#_Toc137206404)

[seller\_type 12](#_Toc137206405)

[drivetrain 12](#_Toc137206406)

[OneHotEncoder 13](#_Toc137206407)

[Feature Scaling 13](#_Toc137206408)

[Normalising 14](#_Toc137206409)

[Outlier Trimming 14](#_Toc137206410)

[K-Means Clustering Model 15](#_Toc137206411)

[Hierarchical Clustering Model 16](#_Toc137206412)

[Model Evaluation, Exploration, and Solution 17](#_Toc137206413)

[Heatmap 17](#_Toc137206414)

[Boxplots 18](#_Toc137206415)

[Scatterplot 22](#_Toc137206416)

[Solution 23](#_Toc137206417)

[Reflection 24](#_Toc137206418)

[Possible Improvements 24](#_Toc137206419)

[Skills I Could Have Learnt Better 24](#_Toc137206420)

Overview

Cluster analysis is a method used to discover similarities between observations.

In this report, I will analyse the used car data (Used\_Car\_Price.csv) and provide insights on key factors affecting sales, pricing, demand for used cars in the market, and use the result to advise the used car reseller how they can leverage it to improve revenue. For example, improving existing marketing strategies to become more customer-centric, streamlining car purchases, creating a new pricing strategy, etc. After cleaning the data, I will create 2 clustering models: K-means clustering and Hierarchical clustering, to analyse any correlations between the columns.

The dataset comes from a research company and contains a list of features of used cars available for sale. There are 2059 rows and 19 columns. The columns include:

* Make: Make of car
* Model: Model of car
* Price: Price of car ($)
* Year: Year of car
* Kilometer: Mileage in km
* Fuel Type: Fuel Type of car
* Transmission: Transmission of car
* Color: Colour of car
* Owner: Type of car ownership
* Seller Type: Type of car seller
* Engine (CC): Engine capacity in CC
* Max Power: Max power of car
* Max Torque: Max torque of car
* Drivetrain: Type of drivetrain
* Length: Length of car
* Width: Width of car
* Height: Height of car
* Seating Capacity: Seating capacity of car
* Fuel Tank Capacity: Fuel tank capacity in Litres

By combining several data analysis techniques, I will answer the following questions:

1. What does the data show?

I will use descriptive analysis to investigate the dataset. I will analyse the data types of each column, look for missing values and outliers, and perform data manipulation such as feature scaling if necessary.

1. Given the columns, which ones are correlated?

I will utilise scatter plots and boxplots to determine correlations after creating the clustering models. Scatter plots help to identify clusters and trends between 2 columns; and boxplots show the median values of clusters. Explanations will be included in each visual on how to interpret the results.

1. How can I use the findings to improve revenue?

After establishing which columns are correlated, new marketing strategies can be ideated using the results.

Data Exploration

Creating a Calculated Column

Upon loading the data, a new column, *car\_age* was created using *year*. This is used to quickly identify the car’s age without looking at the *year* column. The *year* column is then dropped.

df\_copy['car\_age']=2023-df\_copy['year']  
df\_copy.drop(['year'], axis=1, inplace=True)

Removing Null Values

The .isnull() and .sum() methods are used to find the number of null values present in each column.

df\_copy.isnull().sum()

A screenshot of a computer

Description automatically generated with medium confidence

We can see that the columns *engine\_capacity*, *max\_power*, *max\_torque*, *drivetrain*, *length*, *width*, *height*, *seating\_capacity*, and *fuel\_tank\_capacity* have null values.

The null values in the categorical columns are first removed. The code below finds the numerical columns.

cat\_cols = [n **for** n **in** df\_copy.columns **if** df\_copy[n].dtypes == 'O']

A screenshot of a computer

Description automatically generated with medium confidence

From here, I replaced the null values in *max\_power*, *max\_torque*, *drivetrain* with the modal values of each column. The code used to perform this is shown below.

**for** c **in** cat\_cols:   
 df\_copy[c].fillna(df\_copy[c].mode(),inplace=True)

Next, the null values in the numerical columns are removed. The code below finds the numerical columns.

num\_cols = [n **for** n **in** df\_copy.columns **if** df\_copy[n].dtypes != 'O']

A screenshot of a computer program

Description automatically generated with medium confidence

From here, I replaced the null values in *engine\_capacity*, *length*, *width*, *height*, *seating\_capacity*, and *fuel\_tank\_capacity* with the mean values of each column. The code used to perform this is shown below.

**for** c **in** num\_cols:   
 df\_copy[c].fillna(df\_copy[c].mean(),inplace=True)

After the null values are removed, the dataset is checked once again for null values.

A screenshot of a computer code

Description automatically generated with low confidence

Drop Redundant Columns

The categorical columns, *model, max\_power*, and *max\_torque*, will be dropped because they have too many unique values to map. Below are screenshots of their unique values.

The code used to show the unique values and the output are shown below.

# ‘model’  
df\_copy['model’].value\_counts()  
  
# ‘max\_power’  
df\_copy['max\_power'].value\_counts()  
  
# ‘max\_torque’  
df\_copy['max\_torque'].value\_counts()

*model*

A screenshot of a computer

Description automatically generated with medium confidence

*max\_power*

A screenshot of a computer

Description automatically generated with low confidence

*max\_torque*

A screenshot of a computer

Description automatically generated with medium confidence

The code below is used to drop the columns.

# Drop columns 'model', 'max\_power', and 'max\_torque' because they have too many unique values to be mapped  
df\_copy=df\_copy.drop(['model','max\_power','max\_torque'], axis=1)

LabelEncoder

The value counts of each categorical column are first shown. The code used and output is shown below.

# Check the number of unique values in each column  
**for** c **in** cat\_cols:   
 print(c)  
 print(df\_encode[c].value\_counts())  
 print('\n======================================\n')

*make*

A screenshot of a computer

Description automatically generated with low confidence

*fuel\_type*

A screenshot of a computer

Description automatically generated with medium confidence

*transmission*

A number on a white background

Description automatically generated with low confidence

*color*

A picture containing text, screenshot, font

Description automatically generated

*owner*

A screenshot of a computer

Description automatically generated with medium confidence

*seller\_type*

A picture containing text, font, screenshot, white

Description automatically generated

*drivetrain*

A black text on a white background

Description automatically generated with low confidence

*LabelEncoder* will be used to encode the string values into integer values. Since most of the columns are not ordinal, the mapping can be done arbitrarily. However, *owner* will be manually mapped since there is an implied order (First -> Second -> Third -> Fourth -> 4 or More -> UnRegistered Car).

### make

Values with <50 count are replaced with *rare* before encoding.

# Replace values with < 50 count with 'rare'  
df\_encode['make'] = df\_encode['make'].replace(['Renault','Skoda','Land Rover','Kia','Jeep','Jaguar','MG','Nissan','Volvo','Porsche','MINI', 'Datsun','Chevrolet','Lexus','Mitsubishi','Ssangyong', 'Rolls-Royce', 'Isuzu', 'Fiat','Maserati','Ferrari','Lamborghini','Ford'], 'rare')

df\_encode['make'].value\_counts()

# Create LabelEncoder  
le\_make = LabelEncoder()  
  
# Fit encoder with column and transform  
encoded\_make = le\_make.fit\_transform(df\_encode['make'])  
  
# Replace column values with encoder values  
df\_encode['make'] = encoded\_make  
  
# Check column values  
df\_encode['make'].value\_counts()

To show which values are mapped to which number, the classes are shown below.

# Check how values are mapped by encoder  
le\_make.classes\_



The values are mapped starting with Audi at 0 to rare at 10.

### fuel\_type

Values with <10 count are replaced with *rare* before encoding.

df\_encode['fuel\_type'] = df\_encode['fuel\_type'].replace(['Electric','LPG','Hybrid','CNG + CNG', 'Petrol + CNG', 'Petrol + LPG'],'rare')

df\_encode['fuel\_type'].value\_counts()

# Create LabelEncoder  
le\_fuel\_type = LabelEncoder()  
  
# Fit encoder with column and transform  
encoded\_fuel\_type = le\_fuel\_type.fit\_transform(df\_encode['fuel\_type'])  
  
# Replace column values with encoder values  
df\_encode['fuel\_type'] = encoded\_fuel\_type  
  
# Check column values  
df\_encode['fuel\_type'].value\_counts()

To show which values are mapped to which number, the classes are shown below.

# Check how values are mapped by encoder  
le\_fuel\_type.classes\_



The values are mapped starting with CNG at 0 to rare at 3.

### transmission

Since *transmission* only has 2 unique values (Automatic and Manual), *LabelEncoder* can be used right away.

# Create LabelEncoder  
le\_transmission = LabelEncoder()  
  
# Fit encoder with column and transform  
encoded\_transmission = le\_transmission.fit\_transform(df\_encode['transmission'])  
  
# Replace column values with encoder values  
df\_encode['transmission'] = encoded\_transmission  
  
# Check column values  
df\_encode['transmission'].value\_counts()

To show which values are mapped to which number, the classes are shown below.

# Check how values are mapped by encoder  
le\_transmission.classes\_



The values are mapped starting with Automatic at 0 to Manual at 1.

### color

Values with <50 count are replaced with *rare* before encoding.

# Replace values with < 50 count with 'rare'  
df\_encode['color'] = df\_encode['color'].replace(['Brown','Maroon','Gold','Bronze','Green', 'Orange','Others','Yellow','Beige','Purple','Pink'],'rare')

df\_encode['color'].value\_counts()

# Create LabelEncoder  
le\_color = LabelEncoder()  
  
# Fit encoder with column and transform  
encoded\_color = le\_color.fit\_transform(df\_encode['color'])  
  
# Replace column values with encoder values  
df\_encode['color'] = encoded\_color  
  
# Check column values  
df\_encode['color'].value\_counts()

To show which values are mapped to which number, the classes are shown below.

# Check how values are mapped by encoder  
le\_color.classes\_



The values are mapped starting with Black at 0 to rare at 6.

### owner

Values with <30 count are replaced with *rare* before mapping.

# Replace values with < 30 count with 'rare'  
df\_encode['owner'] = df\_encode['owner'].replace(['UnRegistered Car', 'Fourth','4 or More'],'rare')

df\_encode['owner'].value\_counts()

To reiterate, *owner* will be manually mapped since there is an implied order (First -> Second -> Third -> rare). The code used to map the values and the output is shown below.

# Map string values to int  
df\_encode['owner'] = df\_encode['owner'].map({'First':0, 'Second':1, 'Third':2, 'rare':3})  
df\_encode['owner'].value\_counts()

A picture containing text, font, screenshot, white

Description automatically generated

### seller\_type

Since *seller\_type* only has 3 unique values (Commercial Registration, Corporate, and Individual), *LabelEncoder* can be used right away.

# Create LabelEncoder  
le\_seller\_type = LabelEncoder()  
  
# Fit encoder with column and transform  
encoded\_seller\_type = le\_seller\_type.fit\_transform(df\_encode['seller\_type'])  
  
# Replace column values with encoder values  
df\_encode['seller\_type'] = encoded\_seller\_type  
  
# Check column values  
df\_encode['seller\_type'].value\_counts()

To show which values are mapped to which number, the classes are shown below.

# Check how values are mapped by encoder  
le\_seller\_type.classes\_



The values are mapped starting with Commercial Registration at 0 to Individual at 3.

### drivetrain

Since *seller\_type* only has 3 unique values (AWD, FWD, RWD), *LabelEncoder* can be used right away.

# Create LabelEncoder  
le\_drivetrain = LabelEncoder()  
  
# Fit encoder with column and transform  
encoded\_drivetrain = le\_drivetrain.fit\_transform(df\_encode['drivetrain'])  
  
# Replace column values with encoder values  
df\_encode['drivetrain'] = encoded\_drivetrain  
  
# Check column values  
df\_encode['drivetrain'].value\_counts()

To show which values are mapped to which number, the classes are shown below.

# Check how values are mapped by encoder  
le\_drivetrain.classes\_



The values are mapped starting with AWD at 0 to RWD at 2.

OneHotEncoder

*LabelEncoder* was used instead of *OneHotEncoder* (OHE) because the different unique values are necessary to perform clustering and creating scatter plots. For example, having 1 column with 5 unique values (1, 2, 3, 4, 5) will produce a better scatter plot than 5 columns with 2 unique values each (0, 1). Apart from changing the readability of the data, OHE increases the storage space and processing time of the data. Hence, for these reasons, OHE will not be used.

Feature Scaling

With the null values replaced and values properly encoded, the numerical columns are visualised with histograms to inspect the distribution of data. The code used is shown below.

df\_scaled.hist(bins = 50, figsize = (10, 8))

plt.tight\_layout()  
plt.show()

A picture containing text, diagram, plan, technical drawing

Description automatically generated

I rescaled the data using *StandardScaler* from *sklearn*. Scaling is performed to generalise the data, to bring the data points closer to each other, and improve the effectiveness of machine learning. The code used to scale the data is shown below.

scaler = StandardScaler()  
**for** c **in** num\_cols:  
 df\_scaled[c] = scaler.fit\_transform(df\_encode[c].values.reshape(-1,1))

After scaling, I showed the histogram using the same code as before.

A picture containing text, diagram, plan, parallel

Description automatically generated

The difference before and after scaling is the x-axis scale changes while the distribution of data remains unchanged. However, only the x-axis of the encoded columns (*fuel\_type, transmission, color, owner, seller\_type,* and *drivetrain*) are unchanged. For *price*, *kilometer*, *engine\_capacity*, and *car\_age*, we can see that the distribution is right-skewed. For *make*, *length*, *width*, *seating\_capacity*, and *fuel\_tank\_capacity*, they almost have a normal distribution. For height, it has a slight left-skew. The data distribution for all the 12 columns is not smooth as seen in the gaps of the histograms.

Normalising

Transformers such as *YeoJohnsonTransformer* and *PowerTransformer* will not be used to normalise the numerical columns. This is because normalising the data produces a worse silhouette score at the end.

Outlier Trimming

As seen in the histograms, most columns do not have normal distributions. Having a normal distribution is important in machine learning because it aids to remove biases from the data. For this dataset, outliers from the original numerical columns will be removed.

These outliers will be removed using the interquartile range (IQR) method. Values below and above 1.5 times the IQR will be removed. The code used to remove the outliers is shown below.

**for** c **in** num\_cols:  
 # Find interquartile range  
 q1=df\_trim[c].quantile(0.25)  
 q3=df\_trim[c].quantile(0.75)  
 iqr=q3-q1  
  
 # Find lower and upper bounds  
 lower\_bound=q1-1.5\*iqr  
 upper\_bound=q3+1.5\*iqr  
  
 # Trim outliers  
 df\_trim=df\_trim[(df\_trim[c]>=lower\_bound)&(df\_trim[c]<=upper\_bound)]

The number of rows before and after trimming is shown.

print(f'Number of rows before trimming: {len(df\_scaled)}')  
print(f'Number of rows after trimming: {len(df\_trim)}')



This is what the distributions look like after trimming outliers.

A picture containing text, diagram, parallel, plan

Description automatically generated

It is observed that *price* and *kilometer* are more normalised than the others. The other columns had minimal change, or some of their bins were removed.

K-Means Clustering Model

The code below creates the initial K-Means model.

kmeans = KMeans(n\_clusters=3, n\_init=20, max\_iter=300, random\_state=1)  
kmeans.fit(df\_modelling)

After creating the model, its silhouette score (SS) is calculated. The higher the SS, the better the model. The code used and the output are shown below.

print(f'Initial silhouette score: {silhouette\_score(df\_modelling, kmeans.labels\_)}')



To improve the model, the SS is calculated against a range of clusters (2 to 10 clusters). After calculating the scores from all the clusters, the best score and cluster size is printed. The code used and the output are shown below.

k\_range = range(2,11)  
silhouette\_scores =[]  
  
**for** i **in** k\_range:  
 km\_i = KMeans(n\_clusters=i, n\_init=20, max\_iter=300, random\_state=1)  
 silhouette\_scores.append(silhouette\_score(df\_modelling, km\_i.fit\_predict(df\_modelling)))  
   
print(f'Best score: {max(silhouette\_scores)}\nNumber of clusters: {silhouette\_scores.index(max(silhouette\_scores))+2}')



It is observed that using 2 clusters instead of 3 improved the SS by about 0.11. Using this result, an improved model is created using the optimal number of clusters.

kmeans = KMeans(n\_clusters=2, n\_init=20, max\_iter=300, random\_state=1)  
kmeans.fit(df\_modelling)

The code below prints out the improved score.

print(f'Improved silhouette score: {silhouette\_score(df\_modelling, kmeans.labels\_)}')



Hierarchical Clustering Model

The code below creates the initial hierarchical model.

ac = AgglomerativeClustering(n\_clusters = 3)  
ac.fit\_predict(df\_modelling)

After creating the model, its silhouette score (SS) is calculated. The code used and the output are shown below.

print(f'Initial silhouette score: {silhouette\_score(df\_modelling, ac.labels\_)}')



To improve the model, the SS is calculated against a range of clusters (2 to 10 clusters). For this hierarchical model, there are 2 ways of calculating the SS: *ward* and *average*. After calculating the scores from all the clusters from both methods, the best score and cluster size is printed. The code used and the output are shown below.

*ward*:

k\_range = range(2,11)  
silhouette\_scores =[]  
  
**for** i **in** k\_range:  
 ac\_i = AgglomerativeClustering(n\_clusters = i,linkage='ward')  
 silhouette\_scores.append(silhouette\_score(df\_modelling, ac\_i.fit\_predict(df\_modelling)))  
  
print(f'Best score: {max(silhouette\_scores)}\nNumber of clusters: {silhouette\_scores.index(max(silhouette\_scores))+2}')



*average*:

k\_range = range(2,11)  
silhouette\_scores =[]  
  
**for** i **in** k\_range:  
 ac\_i = AgglomerativeClustering(n\_clusters = i,linkage='average')  
 silhouette\_scores.append(silhouette\_score(df\_modelling, ac\_i.fit\_predict(df\_modelling)))  
   
print(f'Best score: {max(silhouette\_scores)}\nNumber of clusters: {silhouette\_scores.index(max(silhouette\_scores))+2}')



From here, the scores from both methods can be compared. It is observed that for 2 clusters, average has a higher SS than ward by about 0.001, and an overall improvement from the initial model by about 0.187.

Using this result, an improved model is created using 2 clusters and the *average* method.

ac = AgglomerativeClustering(n\_clusters = 2, linkage='average')  
ac.fit\_predict(df\_modelling)



Model Evaluation, Exploration, and Solution

By comparing the improved scores from both models, it is observed that K-Means has a better score by 0.001. Thus, the K-Means model is chosen for further analysis.

I will utilise visualisations such as heatmaps, boxplots, and scatter plots to analyse clusters and determine correlations. Heatmaps show the correlation between each column; boxplots show the distribution of data; and scatter plots show clusters. Explanations on how to interpret each visual will be included.

Heatmap

The code used to create the heatmap and the output are shown below.

# Create heatmap  
plot = sns.heatmap(df\_modelling.corr(), cmap="crest", vmin=0, vmax=1)  
  
# Set title  
plt.title('Correlation between columns')  
  
# Set layout  
plt.tight\_layout()  
  
plt.show()

A picture containing text, screenshot, diagram, line

Description automatically generated

For the heatmap, darker squares imply a stronger correlation.

1. It is observed that *price* has a positive correlation with *engine\_capacity*, *length*, *width*, and *fuel\_tank\_capacity*. This is expected because bigger cars are naturally more expensive.
2. *engine\_capacity* has a positive correlation with length, width, and fuel\_tank\_capacity. Similarly, this is expected because bigger cars require a bigger engine and fuel tank.
3. Apart from these 2 observations, there are few columns correlated to each other.

Boxplots

The code used to create the boxplots and their output are shown below.

temp = df\_modelling.copy()  
temp['cluster'] = y\_pred

# Groupped Boxplots  
i=0  
**for** c **in** temp:   
 print(c)   
 plt.cla()  
 plt.boxplot([temp.iloc[:,i][temp['cluster']==0].tolist(),  
 temp.iloc[:,i][temp['cluster']==1].tolist()],  
 labels=('Cluster 0','Cluster 1'))  
 plt.ylabel(c)  
 plt.show()  
 i+=1

A picture containing diagram, screenshot, line, rectangle

Description automatically generated A picture containing text, diagram, screenshot, technical drawing

Description automatically generated

A picture containing diagram, text, line, parallel

Description automatically generated A picture containing diagram, text, line, screenshot

Description automatically generated

A picture containing text, screenshot, rectangle, line

Description automatically generated A picture containing diagram, line, rectangle, parallel

Description automatically generated

A picture containing text, screenshot, diagram, rectangle

Description automatically generated A picture containing text, screenshot, diagram, rectangle

Description automatically generated

A picture containing text, diagram, screenshot, line

Description automatically generated A picture containing text, screenshot, diagram, number

Description automatically generated

A picture containing diagram, text, line, parallel

Description automatically generated A picture containing diagram, text, line, technical drawing

Description automatically generated

A picture containing text, diagram, screenshot, line

Description automatically generated A picture containing text, screenshot, line, diagram

Description automatically generated

A picture containing text, diagram, screenshot, line

Description automatically generated A picture containing diagram, line, technical drawing, rectangle

Description automatically generated

The boxplots give clear insight about the 16 features of the used cars. Some features are similar and will not be explained. There are 8 different features between the clusters: *make*, *price*, *kilometer*, *engine\_capacity*, *length*, *width*, *height*, and *fuel\_tank\_capacity*. The differences are shown below.

Cluster 0:

* Makes 0 – 6 (Audi, BMW, Honda, Hyundai, Mahindra, Maruti Suzuki, Mercedes-Benz)
* Lower priced
* Higher mileage
* Mostly white-coloured cars
* Lower engine capacity
* Shorter, narrower, taller
* Lower fuel tank capacity

Cluster 1:

* Makes 7 – 10 (Tata, Toyota, Volkswagen, rare)
* Higher priced
* Lower mileage
* Mostly silver-coloured cars
* Higher engine capacity
* Longer, wider, smaller
* Higher fuel tank capacity

These observations were made using the median of each boxplot. Cluster 0 can be classified as cheaper and smaller cars, while Cluster 1 can be classified as more expensive and bigger cars.

Scatterplot

The code used to create the scatterplot and the output are shown below.

# Number of PCA components  
pca\_num\_components = 2  
  
# Create PCA  
pca = PCA(n\_components=pca\_num\_components)  
  
# Fit and transform PCA model  
pc = pca.fit\_transform(temp)  
  
# Transpose features  
loadings = pca.components\_.T  
  
# Create scatterplot  
fig, ax = plt.subplots()  
  
sns.scatterplot(x=pc[:,0], y=pc[:,1], hue=temp['cluster'], style=temp['cluster'], palette="deep")  
  
# Plot the feature vectors as arrows  
**for** i, feature **in** enumerate(loadings):  
 ax.arrow(0, 0, feature[0], feature[1], head\_width=0.2, head\_length=0.2, color='r', alpha=0.8)  
 ax.text(feature[0] \* 1.2, feature[1] \* 1.2, str(i+1), color='r')  
  
# Set other ax details  
plt.title('K-means Clustering PCA Biplot')  
plt.tight\_layout()  
ax.legend(bbox\_to\_anchor=(1.02, 1), loc='upper left', borderaxespad=0, title='cluster')  
ax.set\_xlabel('PC1')  
ax.set\_ylabel('PC2')  
  
# Show the plot  
plt.show()

A picture containing screenshot, text, diagram, plot

Description automatically generated

The scatterplot above is a principal component analysis (PCA) biplot. PCA reduces the dimensionality of the dataset (i.e., reduce the number of columns) while retaining most of the information.

It is observed that the 2 clusters are separated along the x-axis or PC1, Cluster 0 range from -5 to 1 while Cluster 1 range from 2 to 5. The red arrows are the vector projections of the features. These projections show the direction and strength of the feature among the principal components. It shows that feature 1 (*make*) has a positive correlated with PC1 and feature 6 (*color*) has a negative correlation with PC2. The other features do not seem to have a strong correlation with either axis.

Solution

Through exploring the clusters, it was found that the key factors affecting sales, pricing, and demand were:

* Make
* Price
* Mileage
* Colour
* Engine Capacity
* Length
* Width
* Height
* Fuel Tank Capacity

To improve revenue, the seller could employ at least 1 of these strategies:

* Advertise to customers that bigger cars are newer or less worn out than smaller ones since they have fewer mileage.
* Promote car brands such as Audi, BMW, Honda, Hyundai, Mahindra, Maruti Suzuki, and Mercedes-Benz to cheap buyers.
* For customers who have families or drive often, car brands such as Tata, Toyota, Volkswagen, rare can promoted due to their higher fuel tank capacity.

Reflection

Possible Improvements

One possible improvement to my current solution is learning how to create clustering models with datasets having more than 2 columns. During lessons, we only learnt how to create models with datasets having only 2 columns. Hence, there was a disconnect between the weekly practical sessions and this assignment. If several subsets with 2 columns each were created, several models would have to be created, and evaluated. Because of this, if all relationships and clusters were investigated, it would be too time-consuming.

Another improvement to be made is better choices of scatter plots. Difficulties were also encountered when creating visuals. Similarly, during the weekly practical sessions, we only learnt how to create scatter plots between 2 columns. In this assignment, we were tasked with finding relationships and clusters between many features. This would require several scatter plots. I did not want to saturate this report with scatter plots. Hence, I used a PCA biplot. However, I do not know much about PCA and cannot thoroughly explain what it is. The biplot itself could be improved as well.

A third improvement to be had is a better data preparation. Perhaps I could have experimented with omitting or including certain steps like encoding, trimming outliers, or normalising, or performing them in a different way to see its effects on the final score and visuals.

Skills I Could Have Learnt Better

With reference to the module learning objectives, I have learnt to:

1. Explain the applications of data analytics in the business and organisation.
2. Describe the organisational functions where data analytics could be applied.
3. Apply customer segment techniques in a business and organisation to gain a competitive advantage.

I have learnt how Applied Analytics (AA) gives businesses a competitive advantage, manage customer relationships, manage fraudulent activities, and increase revenue. AA allows for effective marketing, profitability, better risk management, and improve decision-making.

Customer segmentation develops a clearer view of the consumer, discovers their needs, enables customised marketing, and allows us to target high-value customers.

I think one skill I could have learnt better is cluster detection. During this assignment, I evaluated my models and clusters using only the SS. I was unable to use visuals effectively to show how good they were.

Another skill I could have learnt better is the sum of squares error (SSE). SSE is another model evaluation method like SS. I find SSE more challenging to use than SS. For SS, the closer the score is to 1, the better. But for SSE, a score of 0 is ideal, and increasing the number of clusters reduces the SSE. However, it is inefficient to create many clusters, hence we must find the ‘optimal’ number of clusters. A line chart is used to find the optimal number, but I find it difficult to interpret.