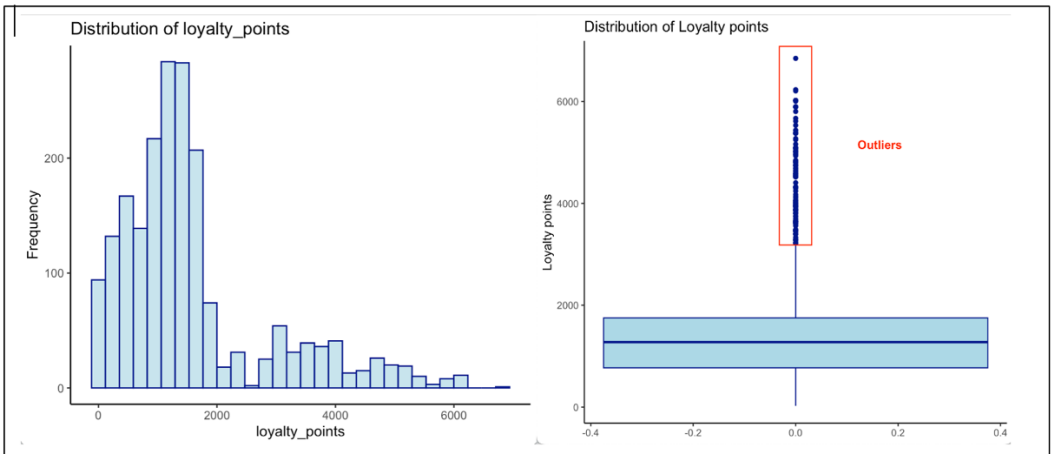
Assignment: **Predicting future outcomes. Turtle Games Case Study**
Jurgita Cepure
LSE_DA301_Advanced Analytics for Organisational Impact_P4_2024

Scope. To improve sales performance, Turtle Games has posed set of business questions:

- How do customers engage with and accumulate loyalty points?
- How can customers be segmented into groups for targeted marketing?
- How can customer reviews be used to inform marketing campaigns and make improvements to the business?
- Are loyalty points suitable data to create predictive models?

The approach was to investigate which factors influence loyalty point accumulation through a multi-method analysis: MLR and Decision Tree Regressor were applied in Python, with key findings cross-validated in R, the company's preferred language. Customer segmentation was performed using K-Means clustering to support targeted marketing. Basic text preprocessing and lexicon-based sentiment models (VADER, TextBlob) were applied to customer reviews to demonstrate the potential of leveraging sentiment insights for better decision-making.

Following data validation and light cleaning/wrangling in both Python and R (*Appendix 1*), **EDA** revealed that the target variable—loyalty points—has a long right-skewed distribution. With a mean of 1,578 and median of 1,276, this highlights high-value outliers. A standard deviation of 1,283 and range from 25 to 6,847 reflect wide variability,



Summary statistics:

	age	remuneration (k£)	spending_score (1-100)	loyalty_points
count	2000.000000	2000.000000	2000.000000	2000.000000
mean	39.495000	48.079060	50.000000	1578.032000
std	13.573212	23.123984	26.094702	1283.239705
min	17.000000	12.300000	1.000000	25.000000
25%	29.000000	30.340000	32.000000	772.000000
50%	38.000000	47.150000	50.000000	1276.000000
75%	49.000000	63.960000	73.000000	1751.250000
max	72.000000	112.340000	99.000000	6847.000000

with 13% of customers (266) falling into the outlier group (>3,220 points).

```
[28]: # Outliers
Q1 = df3['loyalty_points'].quantile(0.25)
Q3 = df3['loyalty_points'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outliers = df3[(df3['loyalty_points'] < lower_bound) | (df3['loyalty_points'] > upper_bound)]
print(f"Outliers are above this number of loyalty points: {upper_bound}")
print(f"Number of outliers: {len(outliers)}")

Outliers are above this number of loyalty points: 3220.125
Number of outliers: 266
```

```
> Q1 <- quantile(df2$loyalty_points, 0.25)
> Q3 <- quantile(df2$loyalty_points, 0.75)
> IQR <- Q3 - Q1
> # Determine upper threshold for loyalty points outliers.
> # Calculate the IQR for loyalty points.
> Q1 <- quantile(df2$loyalty_points, 0.25)
> Q3 <- quantile(df2$loyalty_points, 0.75)
> IQR <- Q3 - Q1
> # Define the outlier thresholds, more importantly the upper one.
> lower_bound <- Q1 - 1.5 * IQR
> upper_bound <- Q3 + 1.5 * IQR
> upper_bound
     75%
3220.125
> # Output: extreme outliers are above count of 3220 of loyalty points

> dim(df2)
[1] 2000    7
> #Output 2000 out of 2000 customers (100%)
> dim(outliers_loyalty_points)
[1] 266   7
> #Output 266 out of 2000 customers (13.3%)
> dim(typical_loyalty_points)
[1] 1734    7
```

From a business perspective, these outliers represent high-value customers.
While statistically they could be removed for clearer modelling, retaining them is important for building models that predict valuable loyalty behaviour and not just the average.

Nevertheless, basic feature comparison across split datasets indicated (*Appendix 2*):

- Splitting of loyalty points improved distribution symmetry.
- Gender: Female-dominant across all subsets.
- Age: Outliers skew younger (30–40), typical customers wider (25–50).
- Remuneration: Median income rises across subsets (44.28K typical, 72.16K outliers); high earners dominate outlier group.
- Spending Score: Overall symmetric (1–99); outliers left-skewed, concentrated at 85–95.
- Education: Majority graduates; notable share of postgraduates/PhDs.
- Products: Outliers purchase from fewer product types (108/200), suggesting possible high-value or luxury-focused spending and need for deeper product analysis.

Correlation matrix across data sets revealed changing predictors of loyalty points:

- Full data: Spending Score (0.672), Remuneration (0.616).
- Typical subset: Spending Score (0.548), Remuneration (0.408).
- Outliers: Remuneration (0.789), Age (0.192), Spending Score (0.243).

```
[22]: # Creat dataframe with numeric columns only to print corelation matrix for comparison.
      df3 = df2[['age', 'remuneration', 'spending_score', 'loyalty_points']]
      df3.corr()
```

[22]:

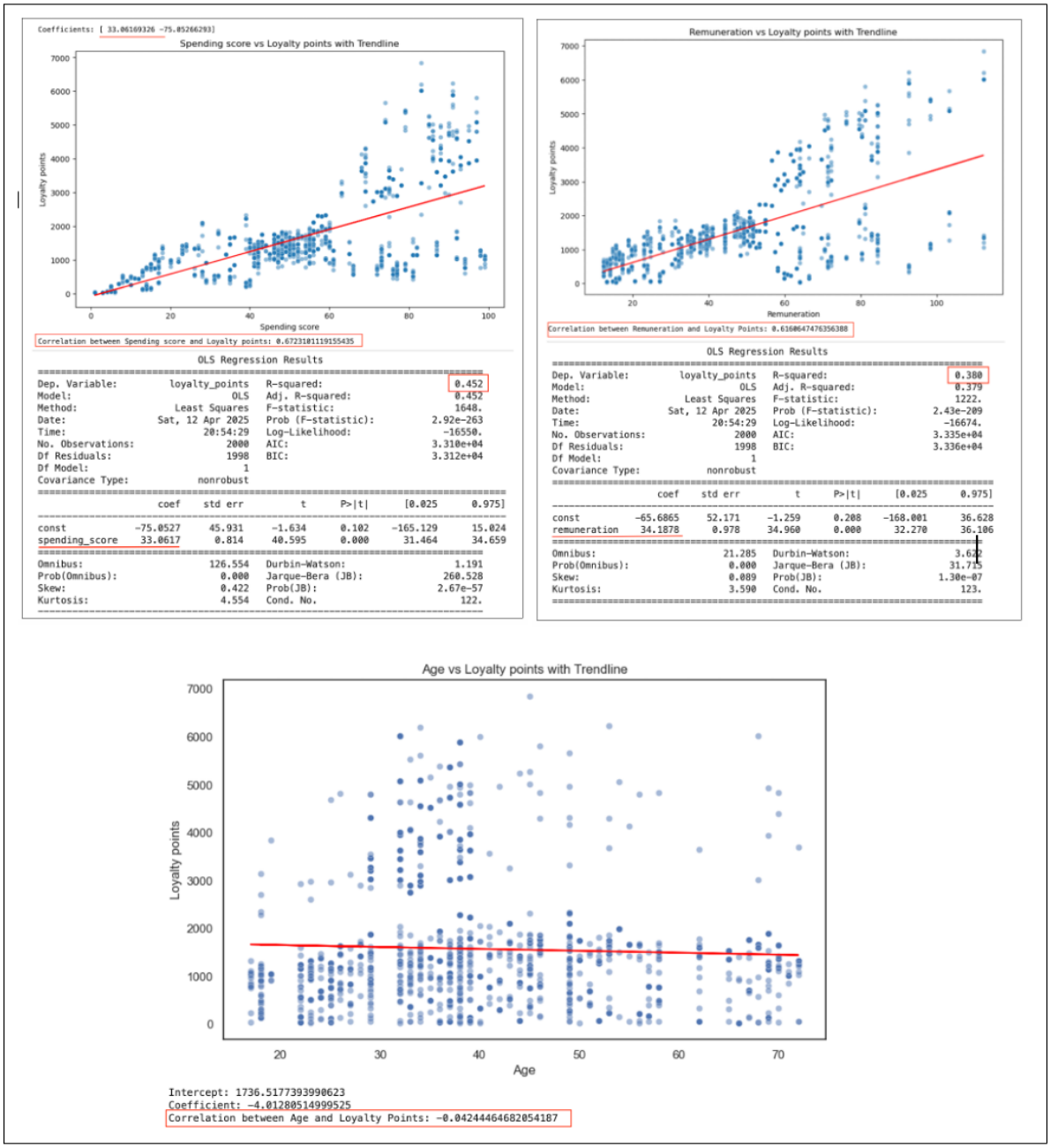
	age	remuneration	spending_score	loyalty_points
age	1.000000	-0.005708	-0.224334	-0.042445
remuneration	-0.005708	1.000000	0.005612	0.616065
spending_score	-0.224334	0.005612	1.000000	0.672310
loyalty_points	-0.042445	0.616065	0.672310	1.000000

```
> cor_matrix
                     age remuneration spending_score loyalty_points
age           1.000000000 -0.005708284   -0.224334309    -0.04244465
remuneration -0.005708284  1.000000000    0.005612492     0.61606475
spending_score -0.224334309  0.005612492    1.000000000     0.67231011
loyalty_points -0.042444647  0.616064748    0.672310112     1.00000000
> cor_matrix_typical_lp
                     age remuneration spending_score loyalty_points
age           1.00000000   0.03943081     -0.2163077     0.04295684
remuneration  0.03943081   1.00000000     -0.3317709     0.40770556
spending_score -0.21630767  -0.33177087      1.0000000     0.54760420
loyalty_points  0.04295684   0.40770556      0.5476042     1.00000000
> cor_matrix_outliers_lp
                     age remuneration spending_score loyalty_points
age           1.00000000   0.07128058    -0.05516622     0.1928325
remuneration  0.07128058   1.00000000    -0.31408773     0.7898965
spending_score -0.05516622  -0.31408773     1.00000000     0.2426874
loyalty_points  0.19283252   0.78989649     0.24268740     1.0000000
```

Regression analysis + modelling. Following the observed associations between independent variables and loyalty points, simple linear regression models were first fitted in Python for each numeric predictor, followed by a combined multiple linear regression (MLR) working with the entire dataset, but applying train-test-split for the latter. In R, MLR models were applied both to the full dataset and to a subset excluding outliers and focusing specifically on typical loyalty point holders.

It was observed:

- Spending score (correlation: 0.672) and remuneration (0.616) are strong predictors. A unit increase in each leads to a ~33 and ~34.2 point rise in loyalty, respectively.
- Age showed no meaningful relationship.



- The multiple regression model was of the strongest performance and achieved Ajusted-R^2 of 0.842 (train) and 0.827 (test), indicating that ~80% of variance in loyalty points can be explained by these independent variables that were not corelated among each other.

[54]:
```
# Compute Adjusted R-squared manualy for both sets since scikit-learn only returns the regular R-squared.
# Number of observations and predictors
n_train = X_train.shape[0]  # number of samples
p_train = X_train.shape[1]  # number of predictors

# Calculate Adjusted R-squared
adjusted_r2_train = 1 - (1 - r2_train) * (n_train - 1) / (n_train - p_train - 1)

print("Adjusted R-squared for train set:", adjusted_r2_train)


# Number of observations and predictors
n_test = X_test.shape[0]  # number of samples
p_test = X_test.shape[1]  # number of predictors

# Calculate Adjusted R-squared
adjusted_r2_test = 1 - (1 - r2_test) * (n_test - 1) / (n_test - p_test - 1)

print("Adjusted R-squared for test set:", adjusted_r2_test)
```

```
Adjusted R-squared for train set: 0.8421931056540901
Adjusted R-squared for test set: 0.8277774331136378
```

[55]:
```
# Check multicollinearity with VIF.
x_temp = sm.add_constant(X_train)

# Create an empty DataFrame.
vif = pd.DataFrame()

# Calculate the VIF for each value.
vif['VIF Factor'] = [variance_inflation_factor(x_temp.values,
                                               i) for i in range(x_temp.values.shape[1])]

# Create the feature columns.
vif['features'] = x_temp.columns

# Print the values to one decimal points.
print(vif.round(1))
```

```
   VIF Factor        features
0        20.7           const
1         1.1  spending_score
2         1.0     remuneration
3         1.1             age
```

However, error metrics remain high (MAE = 402.24, RMSE = 526.49), suggesting moderate predictive power with significant room for improvement.

[52]:
```
#Evaluate model using 3 metrics:
# 1. Calculate mean absolute error.
mae = metrics.mean_absolute_error(y_test, y_pred_test)
print('Mean Absolute Error:', mae)

# 2. Calculate mean squared error.
mse = metrics.mean_squared_error(y_test, y_pred_test)
print('Mean Squared Error:', mse)

# 3. Calculate root mean squared error.
rmse = np.sqrt(mse)
print('Root Mean Squared Error:', rmse)
```

```
Mean Absolute Error: 402.2350305637691
Mean Squared Error: 277188.70233220584
Root Mean Squared Error: 526.4871340614183
```

- Two MLR models built in R returned similar insights to the above also confirmed that for business use Model2 - fitted on the entire data set - may suite better due to its representativeness and better explanatory power, despite Model1 having lower errors.

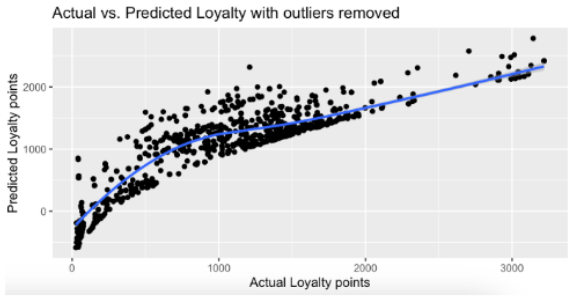
Model 1:
-Built on the dataset excluding outliers to reduce noise.
-Lower Residual Standard Error (RSE = 358.3) and small p-values indicate a strong fit, but the smaller sample may limit generalizability.
-Adjusted R-squared = 0.7248

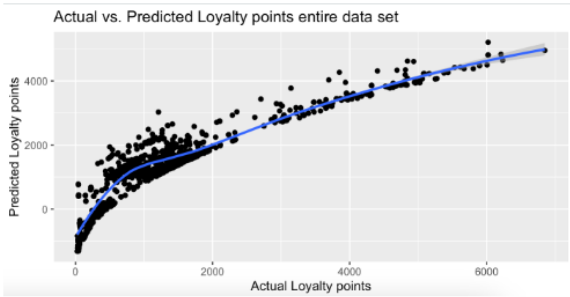
```
> # Create a new object, specify the lm function and the variables.
> model1 = lm(loyalty_points~age+remuneration+spending_score, data=df3)
> # Print the summary statistics.
> summary(model1)

Call:
lm(formula = loyalty_points ~ age + remuneration + spending_score,
    data = df3)

Residuals:
     Min       1Q   Median       3Q      Max
-1108.39  -250.21    53.25   218.00   910.65

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)    -1205.5833    42.4119  -28.43   <2e-16 ***
age                9.2167     0.6198   14.87   <2e-16 ***
remuneration      21.7164     0.4337   50.07   <2e-16 ***
spending_score    23.3754     0.3942   59.30   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 358.3 on 1730 degrees of freedom
Multiple R-squared:  0.7253,    Adjusted R-squared:  0.7248
F-statistic:  1523 on 3 and 1730 DF,  p-value: < 2.2e-16
```

Model 2:
-Built on the full dataset for broader representation.
-RSE is higher (513.08), reflecting greater variability in the full population.
-Adjusted R^2 = 0.8397 with low p-values confirms a robust fit, better capturing the complexity of the full data.

```
> # Create a new object, specify the lm function and the variables.
> model2 = lm(loyalty_points~age+remuneration+spending_score, data=df2_1)
> # Print the summary statistics.
> summary(model2)

Call:
lm(formula = loyalty_points ~ age + remuneration + spending_score,
    data = df2_1)

Residuals:
     Min       1Q   Median       3Q      Max
-1819.11  -350.84     4.61   291.00  1894.62

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)    -2203.0598    52.3609  -42.08   <2e-16 ***
age               11.0607     0.8688   12.73   <2e-16 ***
remuneration      34.0084     0.4970   68.43   <2e-16 ***
spending_score    34.1832     0.4519   75.64   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 513.8 on 1996 degrees of freedom
Multiple R-squared:  0.8399,    Adjusted R-squared:  0.8397
F-statistic:  3491 on 3 and 1996 DF,  p-value: < 2.2e-16
```

predicted vs. actual plots for both models shows most values close to the diagonal, indicating **strong predictive accuracy**.





These large errors in both best fit – in Python and R - MLR models are likely due to the influence of outliers, yet these are the very customers the business should aim to understand and retain. This trade-off must be considered when optimizing the model further.

The next step was to develop and test more robust, non-linear models (e.g., decision trees, random forests) and apply feature engineering for improved prediction.

Decision Tree regressor. The objective was to identify the optimal model by building and pruning one with all features and another trained on a dataset with only the significant features (*Appendix 3*).

The model, which combines feature reduction (keeping only the most significant features: remuneration, spending score, and age) and pruning to a maximum depth of 10, was found to perform optimally.

3.3. Evaluate the importance of features in the model pruned to depth 12 (dtr_12) and the unpruned one (dtr)

3.3.1 the model pruned to depth 12 (dtr_12)

```
#Extract feature importances
feature_importances_12 = dtr_12.feature_importances_

#Create a DataFrame to display feature importances
feature_importance_df4 = pd.DataFrame({'Feature': feature_cols, 'Importance': feature_importances_12})
feature_importance_df4 = feature_importance_df4.sort_values(by='Importance', ascending=False)
```

3.4 Retrain and refit decision tree regresor to data frame with significant features only

in hope of a more efficient, interpretable, and less overfitted model.

[137]:
```
#Set 1% threshold for feature importance
threshold = 0.01

# Find the features where the importance is greater than the threshold.
important_features_1perc = feature_importance_df4[feature_importance_df4['Importance'] > threshold]

# Extract the names of the important features.
important_feature_names = important_features_1perc['Feature'].values

# Filter the X_train and X_test dataframes to include only the important features
X_train_impt_feat = X_train[important_feature_names]
X_test_impt_feat = X_test[important_feature_names]
```

4.2 Prune the feature ajusted model and plot it

in the data set where all features were used to train the model max_depth of 12(half of the initial tree depth) was the optimal fit for better handling of unseen data, hence will prune the feature ajusted model to 10 and will plot for comparison.

[146]:
```
# Creat decission tree regressor to the train set with selected features only and prune to depth of 10.
dtr_impt_feat_10 = DecisionTreeRegressor(max_depth=10, random_state=42)

# Train Decision Tree Regressor
dtr_impt_feat_10.fit(X_train_impt_feat, y_train)

# Predict loyalty_points on test data
y_pred_impt_feat = dtr_impt_feat_10.predict(X_test_impt_feat)

# For R2 evaluation purposes
y_pred_impt_feat_train = dtr_impt_feat.predict(X_train_impt_feat)
```

[147]:
```
#Evaluate the model accuracy with importance of feature >1% & when it is pruned to depth of 10.
mse = mean_squared_error(y_test, y_pred_impt_feat)
mae = mean_absolute_error(y_test, y_pred_impt_feat)
rmse = mean_squared_error(y_test, y_pred_impt_feat, squared=False)
r2 = r2_score(y_test, y_pred_impt_feat)

# For R2 evaluation purposes
r2_train = r2_score(y_train, y_pred_impt_feat_train)

print(f"Mean Squared Error with importance of feature >1% & depth of 10: {mse}")
print(f"Mean Absolute Error with importance of feature >1% & depth of 10: {mae}")
print(f"R-squared with importance of feature >1% & depth of 10: {r2}")
print(f"R-squared with importance of feature in train set >1% & depth of 10: {r2_train}") # For R2 evaluation purposes
print(f"Root Mean Squared Error with importance of feature >1% & depth of 10: {rmse}")
```

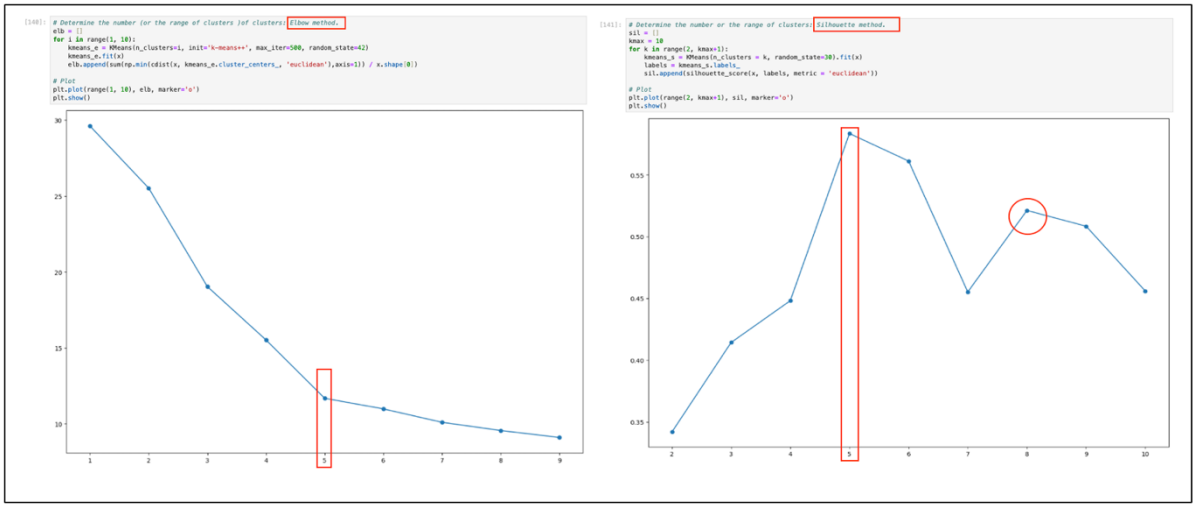
```
Mean Squared Error with importance of feature >1% & depth of 10: 6197.465399130485
Mean Absolute Error with importance of feature >1% & depth of 10: 41.51814699843586
R-squared with importance of feature >1% & depth of 10: 0.9961783497957358
R-squared with importance of feature in train set >1% & depth of 10: 1.0
Root Mean Squared Error with importance of feature >1% & depth of 10: 78.72398236325755
```

The Mean Squared Error (MSE) is **6,197.47**, and the Root Mean Squared Error (RMSE) is **78.72**, indicating improved predictive accuracy. With an R-squared of **0.99618**, it explains **99.6%** of the variance in loyalty points. Pruning to depth 10 strikes a balance between model complexity and performance, reducing overfitting while maintaining high accuracy.

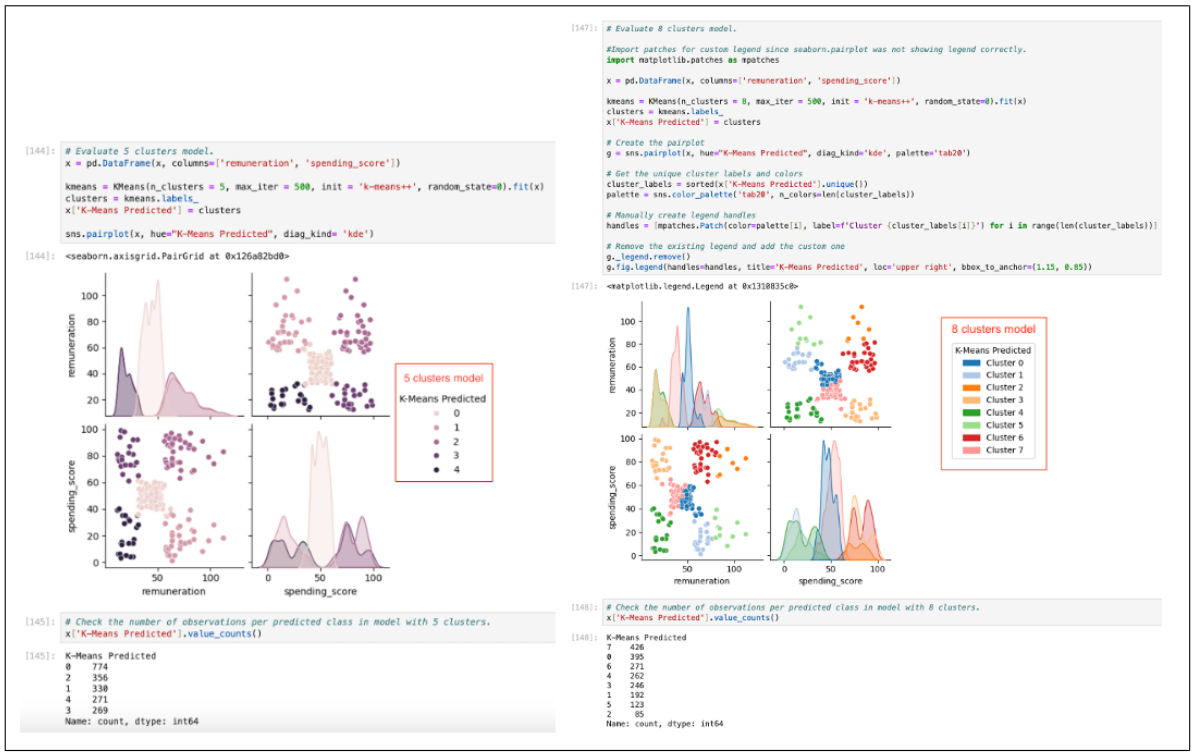
The model builds on insights from linear regression, where spending score and remuneration were the strongest predictors of loyalty points, with age contributing just somewhat. By focusing on these key features, the model achieved better predictive accuracy and provides more robust and interpretable predictions than the linear regression approach. It must be noted, that as new data becomes available, the inclusion of additional features should be explored, ensuring predictions remain accurate without overcomplicating the model.

Next part will explore interaction effects between spending score and remuneration to capture more nuanced patterns.

Using **K-means clustering**, the analysis identified 5 clusters as the optimal number based on the elbow and silhouette methods.

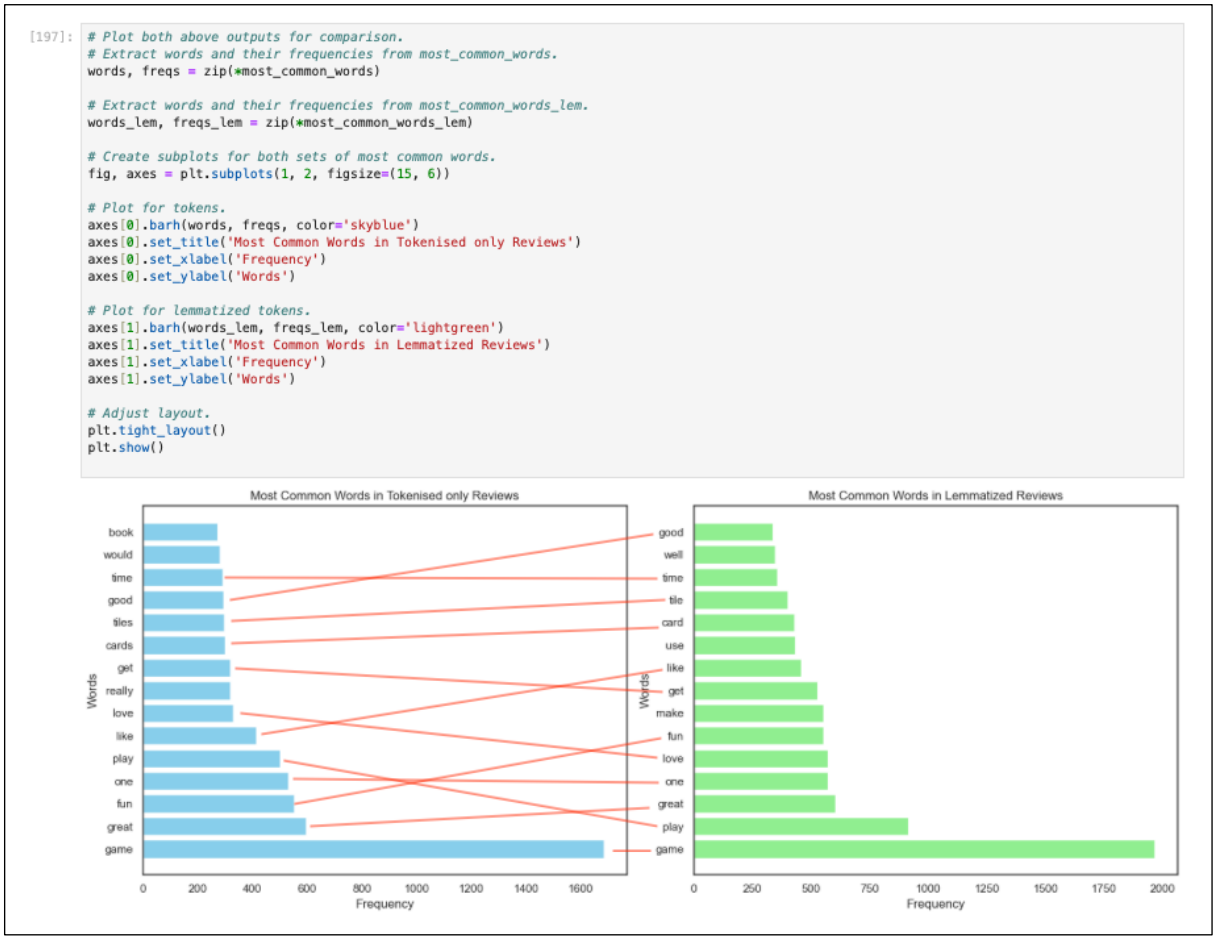


Both methods suggest that 5 clusters capture most customer segments effectively. However, the silhouette plot revealed a secondary peak at 8 clusters, pointing to a potential substructure, perhaps within higher-value customer segments. This in turn suggests that, while 5 clusters represent the broader customer base, 8 clusters may isolate valuable outliers—high-income, high-spending customers, which could be important for targeted marketing strategies.



The analysis of 8 clusters, based on mean remuneration and spending scores, uncovered smaller, more specialized segments that may otherwise have been missed. (*Appendix 4*) These clusters provide actionable insights for the marketing team to tailor strategies. By recognizing these distinct segments, such strategies can become more personalized and focused, improving customer engagement and retention.

Sentiment analysis was conducted on 2,000 customer reviews using lexicon-based models VADER and TextBlob to evaluate their potential for marketing campaigns. Accuracy was manually tested by reviewing 20 of the most negative and positive reviews. Both models were applied to raw and pre-processed data (including tokenization, stopword removal, and lemmatization). (*Appendix 5*)
Analysis showed that lemmatization had limited impact on word frequency



likely due to the simpler words' structure of reviews but improved data quality and reduced vocabulary breadth for model consistency.

Both models consistently identified strong positive sentiment, regardless of processing. VADER struggled with strongly negative reviews on raw text, improving slightly post-lemmatization, but misclassifying 45% of the 20 most negative reviews as checked manually. TextBlob fared better, misclassifying 40-50% of negative sentiment.
As it is know VADER tends to overestimate neutrality and positivity, while TextBlob produces more conservative polarity scores. Both models showed limited ability to detect nuanced negative sentiment, a common limitation of lexicon-based tools, which could lead to overlooking dissatisfaction.
The next step could be to explore machine learning models (e.g., BERT, RoBERTa) trained on review-specific data to better detect subtle negative sentiment.

Conclusion. Loyalty points are strongly influenced by spending score and remuneration, with decision trees offering highly accurate predictions. Customer segmentation and sentiment analysis provide actionable insights, though more advanced tools are needed to detect dissatisfaction. Overall, loyalty points offer strong potential for predictive modelling and targeted marketing.

Appendixes

Appendix 1

In Python

```
[10… # Sense-check data set to determine data types, unique values, missing values, duplicates, summary sta
validate_data(df)
```

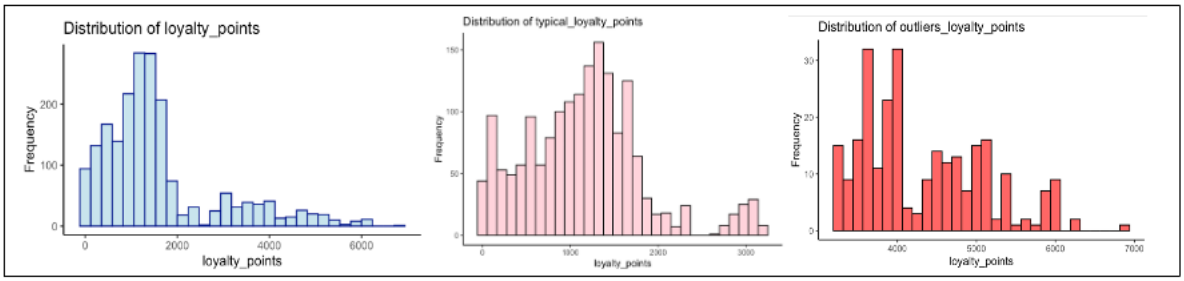
```
 DataFrame info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 11 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   gender                   2000 non-null   object
 1   age                      2000 non-null   int64
 2   remuneration (k£)        2000 non-null   float64
 3   spending_score (1-100)   2000 non-null   int64
 4   loyalty_points           2000 non-null   int64
 5   education                2000 non-null   object
 6   language                 2000 non-null   object
 7   platform                 2000 non-null   object
 8   product                  2000 non-null   int64
 9   review                   2000 non-null   object
 10  summary                  2000 non-null   object
dtypes: float64(1), int64(4), object(6)
memory usage: 172.0+ KB

Unique values per column:
gender                      2
age                        45
remuneration (k£)          64
spending_score (1-100)     84
loyalty_points            627
education                   5
language                    1
platform                    1
product                   200
review                   1980
summary                  1432
dtype: int64

Number of missing values:
gender                     0
age                        0
remuneration (k£)          0
spending_score (1-100)     0
loyalty_points             0
education                  0
language                   0
platform                   0
product                    0
review                     0
summary                    0
dtype: int64

Number of duplicate rows:
0

 Summary statistics:
                age  remuneration (k£)  spending_score (1-100)  loyalty_points  \
count  2000.000000        2000.000000             2000.000000     2000.000000
mean     39.495000          48.079060               50.000000     1578.032000
std      13.573212          23.123984               26.094702     1283.239705
min      17.000000          12.300000                1.000000       25.000000
25%      29.000000          30.340000               32.000000      772.000000
50%      38.000000          47.150000               50.000000     1276.000000
75%      49.000000          63.960000               73.000000     1751.250000
max      72.000000         112.340000               99.000000     6847.000000

            product
count   2000.000000
mean    4320.521500
std     3148.938839
min      107.000000
25%     1589.250000
50%     3624.000000
75%     6654.000000
max    11086.000000
```

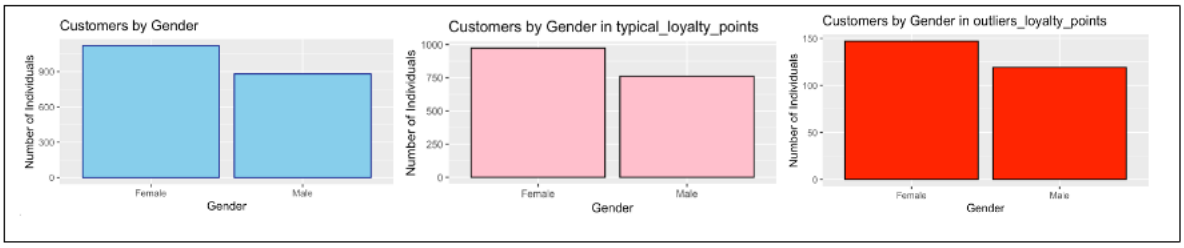
After importing tidyverse library with collection of packages (tidyr, dplyr, readr) suitable for cleaning and manipulating data in **R**, the following has been done to sense-check the data for types, unique values, missing values, duplicates, summary stats:

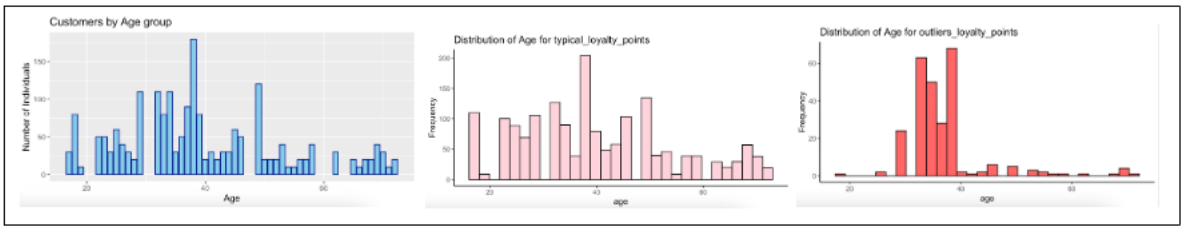
```
> # Import tidyverse package.
> library(tidyverse)
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✔ dplyr     1.1.4     ✔ readr     2.1.5
✔ forcats   1.0.0     ✔ stringr   1.5.1
✔ ggplot2   3.5.1     ✔ tibble    3.2.1
✔ lubridate 1.9.4     ✔ tidyr     1.3.1
✔ purrr     1.0.4
> # Set the working directory
> # Load clean data set
> df1 <- read.csv('df1.csv', header = TRUE)
> # Determine the structure of the data set.
> str(df1)
'data.frame':   2000 obs. of  9 variables:
 $ gender        : chr  "Male" "Male" "Female" "Female" ...
 $ age           : int  18 23 22 25 33 24 37 25 66 32 ...
 $ remuneration  : num  12.3 12.3 13.1 13.1 13.9 ...
 $ spending_score: int  39 81 6 77 40 76 6 94 3 72 ...
 $ loyalty_points: int  210 524 40 562 366 573 61 772 31 724 ...
 $ education     : chr  "graduate" "graduate" "graduate" "graduate" ...
 $ product       : int  453 466 254 263 291 977 979 1012 1031 1459 ...
 $ review        : chr  "When it comes to a DM's screen, the space on the screen itself
> # Convert $product type to character
> df1$product <- as.character(df1$product)
> # Summary stats of the data set.
> summary(df1)
    gender               age          remuneration    spending_score
 Length:2000        Min.   :17.00   Min.   : 12.30   Min.   : 1
 Class :character   1st Qu.:29.00   1st Qu.: 30.34   1st Qu.:32
 Mode  :character   Median :38.00   Median : 47.15   Median :50
                    Mean   :39.49   Mean   : 48.08   Mean   :50
                    3rd Qu.:49.00   3rd Qu.: 63.96   3rd Qu.:73
                    Max.   :72.00   Max.   :112.34   Max.   :99
 loyalty_points  education            product              review
 Min.   :  25   Length:2000        Length:2000        Length:2000
 1st Qu.: 772   Class :character   Class :character   Class :character
 Median :1276   Mode  :character   Mode  :character   Mode  :character
 Mean   :1578
 3rd Qu.:1751
 Max.   :6847
    summary
 Length:2000
 Class :character
 Mode  :character

> # To search for missing values in a data set.
> df1[is.na(df1)]
character(0)
> # Find duplicate rows
> duplicates <- df1[duplicated(df1), ]
> duplicates
[1] gender         age            remuneration   spending_score loyalty_points
[6] education      product        review         summary
<0 rows> (or 0-length row.names)
> # drop unnecessary columns 'review' and 'summary'
> df2 <- subset(df1, select = -c(review, summary))
```

Appendix 2

Entire data set	Loyalty points <3220 Typical customers	Loyalty points >3220 High-value customers



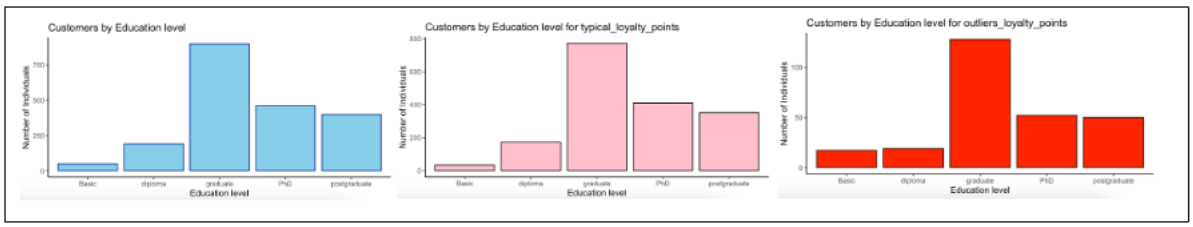





```
> median(df2$remuneration)
[1] 47.15
```

```
> median(typical_loyalty_points$remuneration)
[1] 44.28
```

```
> median(outliers_loyalty_points$remuneration)
[1] 72.16
```




```
> # Output the number of unique products
> num_unique_products
[1] 200
```

```
> # Output the number of unique products
> num_unique_products_typical_lp
[1] 200
```

```
> # Output the number of unique products
> num_unique_products_outliers_lp
[1] 108
```

Appendix 3

Decision Tree Regressor

[71]:
```
# Encode categorical variables. LabelEncoder for 'education' and One-Hot Encoding for 'gender' and 'product'.
# Define the custom order for the 'education' column.
custom_order = ['Basic', 'diploma', 'graduate', 'postgraduate', 'PhD']

# Convert 'education' to a categorical type with the specified order.
df4['education'] = pd.Categorical(df4['education'], categories=custom_order, ordered=True)

# Use LabelEncoder to convert the 'education' column to numeric labels.
le = LabelEncoder()
df4['education_encoded'] = le.fit_transform(df4['education'])

# Apply One-Hot Encoding for the 'gender' and 'product' columns using pd.get_dummies.
df4 = pd.get_dummies(df4, columns=['gender', 'product'], drop_first=True)

# Drop the original 'education' column and keep the encoded one.
df4.drop(columns=['education'], inplace=True)
```

1.3 Define target and the features

- target is loyalty_points
- features - everything else in data frame df4

[74]:
```
# Specify Y.
y = df4['loyalty_points']

# Specify X.
X = df4.drop('loyalty_points', axis=1)
```

2. Create train and test data sets.

[80]:
```
# Split the data into 20% test and 80% train data.
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=42)
```

3. Create Decision tree regressor

[82]:
```
# Create your decision tree regressor.
dtr = DecisionTreeRegressor(random_state=42)

# Train Decision Tree Regressor
dtr.fit(X_train, y_train)

# Predict loyalty_points on test data.
y_pred = dtr.predict(X_test)

# Predict loyalty_points on test data for evaluation.
y_pred_train = dtr.predict(X_train)
```

3.1 Evaluate the model by testing accuracy

[107]:
```
#Evaluate the model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)

# For R2 evaluation purposes
r2_train = r2_score(y_train, y_pred_train)

print(f"Mean Squared Error: {mse}")
print(f"Mean Absolute Error: {mae}")
print(f"R-squared: {r2}")
print(f"R-squared in train set: {r2_train}")
print(f"Root Mean Squared Error: {rmse}")
```

```
Mean Squared Error: 13225.5
Mean Absolute Error: 29.37
R-squared: 0.9918445313492855
R-squared in train set: 1.0
Root Mean Squared Error: 115.00217389249649
```

3.2. Prune the model to various depths and fit for comparison.

[111]:
```
# Get the depth of the decision tree.
initial_tree_depth = dtr.get_depth()
print(f"The depth of the initial decision tree is: {initial_tree_depth}")
```

```
The depth of the initial decision tree is: 24
```

Observation for all pruned vs unpruned

- Unpruned Model best fits (R2 = 0.99596) but is likely overfitting (good on training data but may perform poorly on unseen data).
- Pruned Depth = 12 offering the best trade-off between complexity and performance,although slightly worse than the unpruned model, it avoids overfitting and generalizes well.
- Pruned Depth = 8 has good balance between bias and variance, providing a strong fit with moderate errors. Likely generalizes well.
- Pruned Depth = 5 has Moderate performance, with larger errors and a slightly worse fit than Depth = 8.
- Pruned Depth = 3 is clearly underfitting, with high error metrics and a low R2. This model is too simple to capture enough of the data's patterns.

For best performance on training data and minimal error, the unpruned model is superior. **For generalization and better handling of overfitting**, the model pruned to a depth of 12, provides good accuracy with reduced complexity.

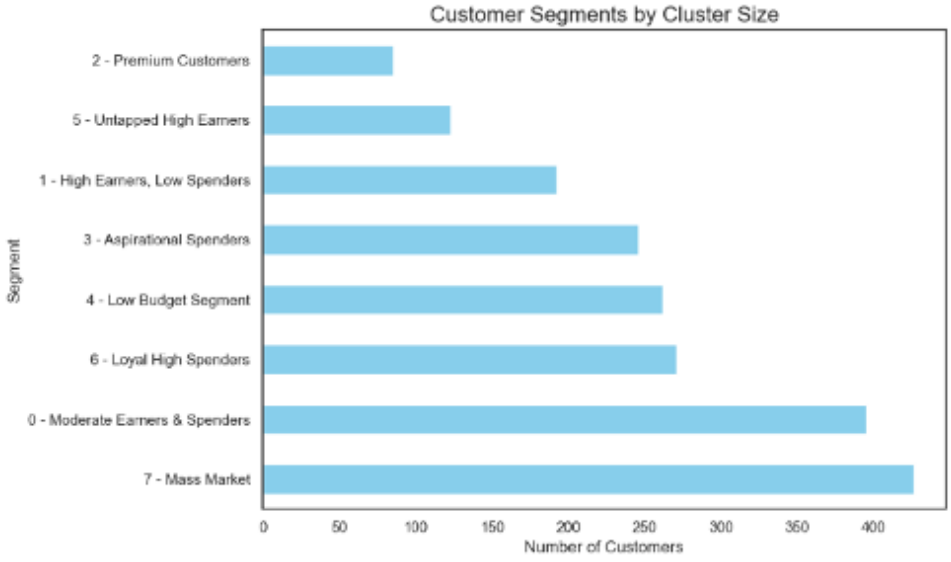
Appendix 4

[155]:
```
# Basic statistics to further understand each cluster.
x.groupby('K-Means Predicted')[['remuneration', 'spending_score']].describe()
```

[155]:

	remuneration								spending_score							
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	max
K-Means Predicted																
0	395.0	50.810937	4.720947	44.28	48.38	50.84	52.89	64.78	395.0	46.744304	6.381122	34.0	42.0	46.0	51.0	59.0
1	192.0	66.172292	5.402403	57.40	61.50	63.96	71.34	76.26	192.0	13.609375	7.612921	1.0	8.0	13.0	20.0	29.0
2	86.0	92.901176	9.989221	82.82	84.46	92.66	98.40	112.34	86.0	78.658824	8.274391	68.0	69.0	79.0	86.0	91.0
3	246.0	19.743333	5.454766	12.30	15.58	18.86	23.78	31.16	246.0	80.975610	9.449728	66.0	73.0	77.0	87.0	99.0
4	262.0	20.055573	5.452758	12.30	15.58	18.86	23.78	31.98	262.0	19.240458	12.557889	3.0	8.0	15.0	31.0	40.0
5	123.0	89.666667	10.077961	79.54	81.18	84.46	98.40	112.34	123.0	21.178862	8.732859	8.0	16.0	18.0	28.0	39.0
6	271.0	67.073579	7.125888	58.58	61.50	63.96	71.34	81.18	271.0	83.059041	9.424762	63.0	75.0	86.0	90.0	97.0
7	426.0	37.954836	5.088384	22.96	34.44	38.54	41.00	47.56	426.0	52.023474	6.811714	35.0	47.0	52.0	57.0	65.0

[156]:
```
# Rename clusters and plot for clarity.
cluster_map = {
    0: "0 - Moderate Earners & Spenders",
    1: "1 - High Earners, Low Spenders",
    2: "2 - Premium Customers",
    3: "3 - Aspirational Spenders",
    4: "4 - Low Budget Segment",
    5: "5 - Untapped High Earners",
    6: "6 - Loyal High Spenders",
    7: "7 - Mass Market"
}

x['Segment'] = x['K-Means Predicted'].map(cluster_map)

x['Segment'].value_counts().plot(kind='barh', figsize=(10,6), color='skyblue')
plt.title("Customer Segments by Cluster Size", fontsize=16)
plt.xlabel("Number of Customers")
plt.tight_layout()
plt.show()
```



Appendix 5

Prepare the data for NLP modelling

2a) Change to lower case and join the elements in relevant review column only

```
[169]: # 'review' column: Change all to lower case and join words with a space into single string.
       df6['review'] = df6['review'].apply(lambda x: " ".join(x.lower() for x in x.split()))

       # Preview.
       df6['review'].head()
```

```
[169]: 0    when it comes to a dm's screen, the space on t...
       1    an open letter to galeforce9*: your unpainted ...
       2    nice art, nice printing. why two panels are fi...
       3    amazing buy! bought it as a gift for our new d...
       4    as my review of gf9's previous screens these w...
       Name: review, dtype: object
```

2b) Replace punctuation in relevant 'review' column only

```
[171]: # Replace all the punctuations in review column.
       df6['review'] = df6['review'].str.replace(r"[^\w\s]", "", regex=True)

       # View output.
       df6['review'].head()
```

```
[171]: 0    when it comes to a dms screen the space on the...
       1    an open letter to galeforce9 your unpainted mi...
       2    nice art nice printing why two panels are fill...
       3    amazing buy bought it as a gift for our new dm...
       4    as my review of gf9s previous screens these we...
       Name: review, dtype: object
```

```
# Apply tokenisation to 'review'.
df7['review_tokens'] = df7['review'].apply(word_tokenize)

# View DataFrame.
df7.head()
```

	review	summary	review_tokens
0	when it comes to a dms screen the space on the...	The fact that 50% of this space is wasted on a...	[when, it, comes, to, a, dms, screen, the, spa...
1	an open letter to galeforce9 your unpainted mi...	Another worthless Dungeon Master's screen from...	[an, open, letter, to, galeforce9, your, unpai...
2	nice art nice printing why two panels are fill...	pretty, but also pretty useless	[nice, art, nice, printing, why, two, panels, ...
3	amazing buy bought it as a gift for our new dm...	Five Stars	[amazing, buy, bought, it, as, a, gift, for, o...
4	as my review of gf9s previous screens these we...	Money trap	[as, my, review, of, gf9s, previous, screens, ...

```
[182]: # Create a set of English stopwords.
       english_stopwords = set(stopwords.words('english'))
```

```
[183]: # Filter out 'review_tokens' to only have alphanumeric & number values and apply stopword removal variable for each row in df7
       df7['review_tokens_no_stopwords'] = df7['review_tokens'].apply(
           lambda tokens: [word for word in tokens if word.isalnum() and word.lower() not in english_stopwords]
       )
```

```
[184]: df7.tail()
```

	review	summary	review_tokens	review_tokens_no_stopwords
1995	the perfect word game for mixed ages with mom ...	The perfect word game for mixed ages with Mom	[the, perfect, word, game, for, mixed, ages, w...	[perfect, word, game, mixed, ages, mom, perhap...

3.5. Lemmatize tokenised reviews

```
[190]: # Initialize the lemmatizer.
       lemmatizer = WordNetLemmatizer()

       # Define the tag map for POS tagging.
       tag_map = defaultdict(lambda: wn.NOUN)  # Default to NOUN
       tag_map['J'] = wn.ADJ  # Adjectives
       tag_map['V'] = wn.VERB  # Verbs
       tag_map['R'] = wn.ADV  # Adverbs

       # Lemmatize the tokens in 'review_tokens_no_stopwords' with POS tagging.
       df7['review_tokens_lemmatized'] = df7['review_tokens_no_stopwords'].apply(
           lambda tokens: [
               lemmatizer.lemmatize(word, pos=tag_map[tag[0]])  # Uses the first letter of the tag (e.g., 'N', 'V')
               for word, tag in nltk.pos_tag(tokens)
           ]
       )
```

```
[191]: df7.head()
```

	review	summary	review_tokens	review_tokens_no_stopwords	review_tokens_lemmatized
0	when it comes to a dms screen the space on the...	The fact that 50% of this space is wasted on a...	[when, it, comes, to, a, dms, screen, the, spa...	[comes, dms, screen, space, screen, absolute, ...	[come, dms, screen, space, screen, absolute, p...