

DA301 Advanced Analytics for Organisational Impact

Assignment template

Scenario

You are a data analyst working for Turtle Games, a game manufacturer and retailer. They manufacture and sell their own products, along with sourcing and selling products manufactured by other companies. Their product range includes books, board games, video games and toys. They have a global customer base and have a business objective of improving overall sales performance by utilising customer trends. In particular, Turtle Games wants to understand:

- how customers accumulate loyalty points (Week 1)
- exploring the structure using decision trees (Week 2)
- exploring clusters in customer behaviour (Week 3)
- can social data (e.g. customer reviews) be used in marketing campaigns (Week 4)
- loading, transforming and visualising data in R (Week 5)
- statistical analysis and modelling in R (Week 6)

Week 1: Linear regression using Python

The marketing department of Turtle Games prefers Python for data analysis. As you are fluent in Python, they asked you to assist with data analysis of social media data. The marketing department wants to better understand how users accumulate loyalty points. Therefore, you need to investigate the possible relationships between the loyalty points, age, remuneration, and spending scores. Note that you will use this data set in future modules as well and it is, therefore, strongly encouraged to first clean the data and then save a copy of the clean data for future use.

1. Load and explore the data

```
In [1]: # Import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.outliers_influence import variance_inflation_factor
import sklearn
from sklearn import datasets
from sklearn import linear_model
```

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
In [2]: # Load the CSV file(s) as reviews.
reviews=pd.read_csv("turtle_reviews.csv")

# View the DataFrame.
reviews.info()
reviews.head(2)
```

```
<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
```

```
Out[2]:
```

	gender	age	remuneration (k£)	spending_score (1-100)	loyalty_points	education	language	pl
--	--------	-----	----------------------	---------------------------	----------------	-----------	----------	----

0	Male	18	12.3	39	210	graduate	EN	
---	------	----	------	----	-----	----------	----	--

1	Male	23	12.3	81	524	graduate	EN	
---	------	----	------	----	-----	----------	----	--



```
In [3]: # Any missing values? - output shows no missing data
reviews_na = reviews[reviews.isna().any(axis=1)]
reviews_na.shape
```

```
Out[3]: (0, 11)
```

```
In [4]: # Basic descriptive statistics.
reviews.describe()
```

Out[4]:

	age	remuneration (k£)	spending_score (1-100)	loyalty_points	product
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000
mean	39.495000	48.079060	50.000000	1578.032000	4320.521500
std	13.573212	23.123984	26.094702	1283.239705	3148.938839
min	17.000000	12.300000	1.000000	25.000000	107.000000
25%	29.000000	30.340000	32.000000	772.000000	1589.250000
50%	38.000000	47.150000	50.000000	1276.000000	3624.000000
75%	49.000000	63.960000	73.000000	1751.250000	6654.000000
max	72.000000	112.340000	99.000000	6847.000000	11086.000000

```
In [5]: # Check for outliers in numerical data columns
def count_outliers_iqr(df):
    outlier_counts = {}

    for col in reviews.select_dtypes(include='number').columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1

        lower = Q1 - 1.5 * IQR
        upper = Q3 + 1.5 * IQR

        outliers = ((reviews[col] < lower) | (reviews[col] > upper)).sum()
        outlier_counts[col] = outliers

    return pd.Series(outlier_counts)

print(count_outliers_iqr(reviews))
```

```
age                0
remuneration (k£)  0
spending_score (1-100)  0
loyalty_points      266
product            0
dtype: int64
```

2. Drop columns

Language is EN for all records, and platform is Web for all records. Hence, these columns are not needed for analysis.

```
In [6]: # Drop unnecessary columns.
reviews.drop(labels=['language', 'platform'], axis=1, inplace=True)
```

3. Rename columns

Column names are simplified for consistency and ease of use.

```
In [7]: # Rename the column headers.
reviews=reviews.rename(columns={'remuneration (k£)': 'remuneration',
                                'spending_score (1-100)': 'spending',
                                'loyalty_points': 'loyalty'})

# View column names.
reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype  
---  -
 0   gender                2000 non-null   object 
 1   age                   2000 non-null   int64  
 2   remuneration          2000 non-null   float64 
 3   spending               2000 non-null   int64  
 4   loyalty               2000 non-null   int64  
 5   education             2000 non-null   object 
 6   product               2000 non-null   int64  
 7   review                2000 non-null   object 
 8   summary               2000 non-null   object 
dtypes: float64(1), int64(4), object(4)
memory usage: 140.8+ KB
```

4. Save the DataFrame as a CSV file

```
In [8]: # Create a CSV file as output.
reviews.to_csv("turtle.csv", index=False)
```

```
In [9]: # Import new CSV file with Pandas.
turtle=pd.read_csv("turtle.csv")

# View DataFrame.
turtle.head(3)
```


Out[9]:

	gender	age	remuneration	spending	loyalty	education	product	rev
0	Male	18	12.30	39	210	graduate	453	When it comes t DM's screen, space or
1	Male	23	12.30	81	524	graduate	466	An Open Lette GaleForce9*:\n\nY unpai
2	Female	22	13.12	6	40	graduate	254	Nice art, r printing. Why panels are

5a) spending vs loyalty

```
In [10]: # Define independent variable.
x=turtle['spending'].values.reshape(-1, 1)

# Define dependent variable.
y=turtle['loyalty'].values.reshape(-1, 1)

# Create model and print summary of metrics.
lm=LinearRegression()
lm.fit(x, y)
```

Out[10]:

LinearRegression ⓘ ?

Parameters

```
In [11]: # Extract the estimated parameters.
print("R-squared value: ",lm.score(x, y))
print("Intercept: ",lm.intercept_)
print("Coefficient: ",lm.coef_)

# Extract the predicted values.
y_pred = lm.predict(x)

# Extract the standard errors.
residuals = y - y_pred
n = len(y)          # number of observations
p = 1               # number of predictors (just spending score)

SE = np.sqrt(np.sum(residuals**2) / (n - p - 1))
print("Standard Error:", SE)
```

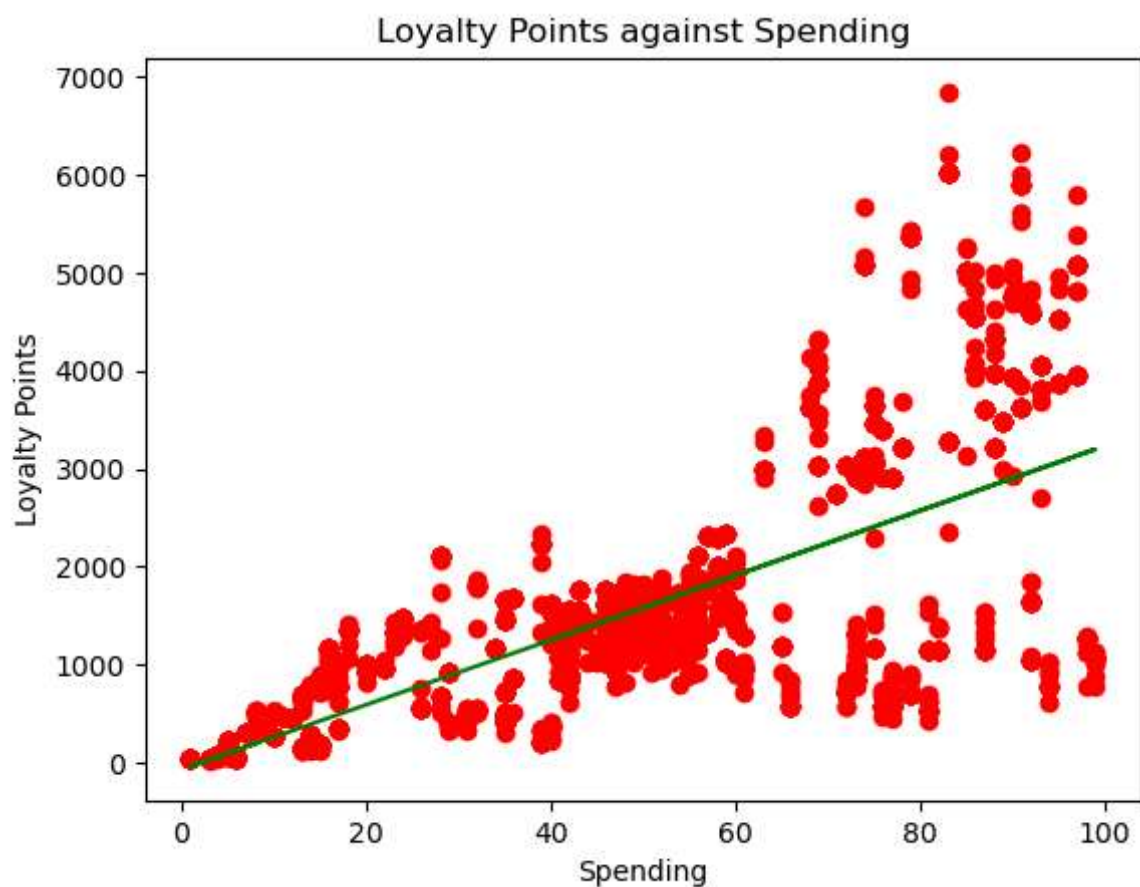
R-squared value: 0.45200088658389104
 Intercept: [-75.05266293]
 Coefficient: [[33.06169326]]
 Standard Error: 950.1810651955294

```
In [12]: # Set the X coefficient and the constant to generate the regression table.
        spending_regression_table = pd.DataFrame({
            'Variable': ['Spending Score', 'Intercept'],
            'Coefficient': [lm.coef_[0][0], lm.intercept_[0]],
        })
        # View the output.
        spending_regression_table
```

```
Out[12]:
```

	Variable	Coefficient
0	Spending Score	33.061693
1	Intercept	-75.052663

```
In [13]: # Plot the graph with a regression line.
        plt.scatter(x, y, color='red')
        plt.plot(x, y_pred, color='green')
        plt.title("Loyalty Points against Spending")
        plt.xlabel("Spending")
        plt.ylabel("Loyalty Points")
        plt.show()
```



5b) remuneration vs loyalty

```
In [14]: # Define independent variable.
        x=turtle['remuneration'].values.reshape(-1, 1)
```

```
# Define dependent variable.
y=turtle['loyalty'].values.reshape(-1, 1)

# Create model and print summary of metrics.
lm=LinearRegression()
lm.fit(x, y)
```

Out[14]:

▼ LinearRegression ⓘ ?

► Parameters

```
In [15]: # Extract the estimated parameters.
print("R-squared value: ",lm.score(x, y))
print("Intercept: ",lm.intercept_)
print("Coefficient: ",lm.coef_)

# Extract the predicted values.
y_pred = lm.predict(x)

# Extract the standard errors.
residuals = y - y_pred
n = len(y)          # number of observations
p = 1                # number of predictors (just spending score)

SE = np.sqrt(np.sum(residuals**2) / (n - p - 1))
print("Standard Error:", SE)
```

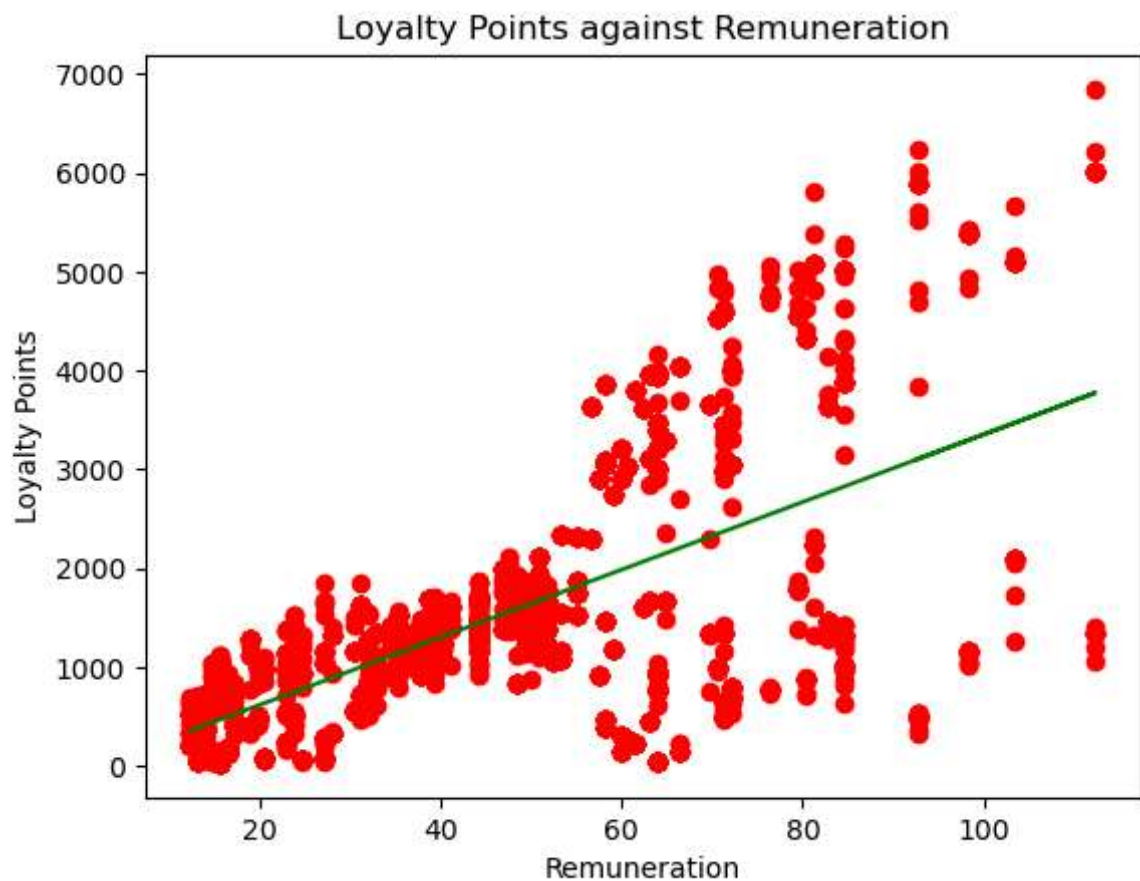
R-squared value: 0.3795357732793634
Intercept: [-65.6865128]
Coefficient: [[34.18782549]]
Standard Error: 1011.0550839438222

```
In [16]: # Set the X coefficient and the constant to generate the regression table.
loyalty_regression_table = pd.DataFrame({
    'Variable': ['Remuneration', 'Intercept'],
    'Coefficient': [lm.coef_[0][0], lm.intercept_[0]],
})
# View the output.
loyalty_regression_table
```

Out[16]:

	Variable	Coefficient
0	Remuneration	34.187825
1	Intercept	-65.686513

```
In [17]: # Plot graph with regression line.
plt.scatter(x, y, color='red')
plt.plot(x, y_pred, color='green')
plt.title("Loyalty Points against Remuneration")
plt.xlabel("Remuneration")
plt.ylabel("Loyalty Points")
plt.show()
```



5c) age vs loyalty

```
In [18]: # Define independent variable.
x=turtle['age'].values.reshape(-1, 1)

# Define dependent variable.
y=turtle['loyalty'].values.reshape(-1, 1)

# Create model and print summary of metrics.
lm=LinearRegression()
lm.fit(x, y)
```

Out[18]:

LinearRegression

Parameters

```
In [19]: # Extract the estimated parameters.
print("R-squared value: ",lm.score(x, y))
print("Intercept: ",lm.intercept_)
print("Coefficient: ",lm.coef_)

# Extract the predicted values.
y_pred = lm.predict(x)

# Extract the standard errors.
residuals = y - y_pred
n = len(y)          # number of observations
p = 1               # number of predictors (just spending score)
```

```
SE = np.sqrt(np.sum(residuals**2) / (n - p - 1))
print("Standard Error:", SE)
```

R-squared value: 0.0018015480437203468

Intercept: [1736.5177394]

Coefficient: [[-4.01280515]]

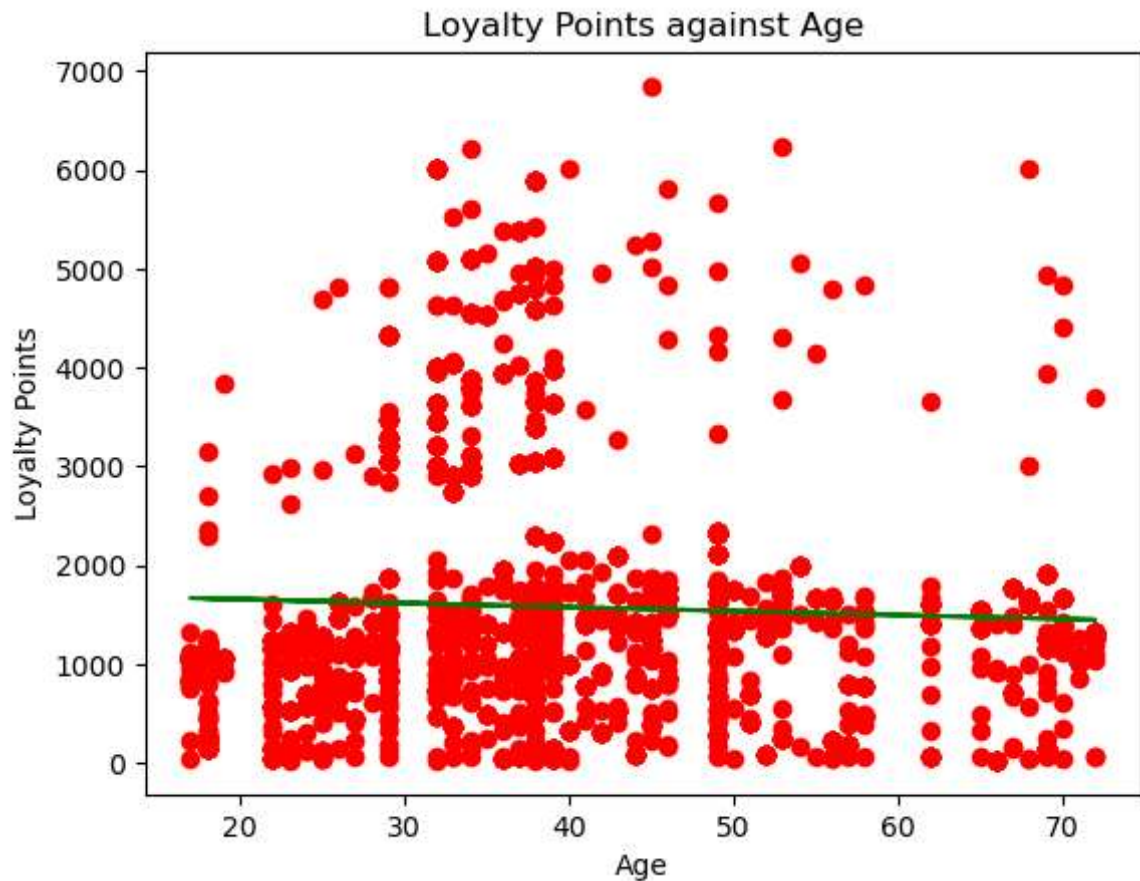
Standard Error: 1282.4040762868394

```
In [20]: # Set the X coefficient and the constant to generate the regression table.
age_regression_table = pd.DataFrame({
    'Variable': ['Age', 'Intercept'],
    'Coefficient': [lm.coef_[0][0], lm.intercept_[0]],
})
# View the output.
age_regression_table
```

```
Out[20]:
```

	Variable	Coefficient
0	Age	-4.012805
1	Intercept	1736.517739

```
In [21]: # Plot graph with regression line.
plt.scatter(x, y, color='red')
plt.plot(x, y_pred, color='green')
plt.title("Loyalty Points against Age")
plt.xlabel("Age")
plt.ylabel("Loyalty Points")
plt.show()
```



6. Observations and insights from SINGLE linear regression

Data was clean and without nulls; loyalty_points has a right-skewed distribution with outliers, which OLS can tolerate but is worth flagging.

Spending ($R^2=0.452$) — spending score is a moderate predictor. Every 1-point increase in spending score is associated with ~33 additional loyalty points. However, 55% of variance is unexplained, suggesting other factors are at play.

Remuneration ($R^2=0.380$) — income is also a meaningful predictor (~34 points per £1k increase) but slightly weaker than spending behaviour alone.

Age ($R^2=0.002$) — essentially no linear relationship. Age is not a useful predictor of loyalty points.

Conclusion — since neither predictor alone explains the majority of variance, this justifies building a multiple linear regression combining both spending score and remuneration.

7. Multiple Linear Regression - MLR Model

7a) Define Variables and fit model

```
In [22]: # Define variables.
y = turtle['loyalty']
X = turtle[['spending', 'remuneration', 'age']]

# Fit model.
mlr=linear_model.LinearRegression()
mlr.fit(X, y)
mlr.predict(X)
```

```
Out[22]: array([ -252.52008641, 1238.47746341, -1308.43606252, ...,
                4434.8905332 , 2066.35892261, 1575.56389984], shape=(2000,))
```

```
In [23]: # Extract the estimated parameters.
print("R-squared value: ",mlr.score(X, y))
print("Intercept: ",mlr.intercept_)
print("Coefficient: ", list(zip(X, mlr.coef_)))
```

R-squared value: 0.8399114912748796

Intercept: -2203.0597513079

Coefficient: [('spending', np.float64(34.18319657763366)), ('remuneration', np.float64(34.00838549260249)), ('age', np.float64(11.060658711843338))]

7b) Split data into train/test sets and check for multicollinearity

```
In [24]: # Split data into train and test sets.
X_train, X_test, y_train, y_test = sklearn.model_selection.train_test_split(X, y)

# Create model.
```



```
blr_model = sm.OLS(y_train, sm.add_constant(X_train)).fit()
y_pred = blr_model.predict(sm.add_constant(X_test))
print_model = blr_model.summary()
print(print_model)
```

OLS Regression Results

```
=====
Dep. Variable:          loyalty    R-squared:                0.842
Model:                  OLS       Adj. R-squared:           0.842
Method:                 Least Squares   F-statistic:            2846.
Date:                   Thu, 16 Jul 2026   Prob (F-statistic):      0.00
Time:                   11:11:24    Log-Likelihood:         -12246.
No. Observations:      1600        AIC:                   2.450e+04
Df Residuals:          1596        BIC:                   2.452e+04
Df Model:               3
Covariance Type:        nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
const	-2197.0105	58.134	-37.792	0.000	-2311.037	-2082.984
spending	33.9681	0.505	67.253	0.000	32.977	34.959
remuneration	34.2457	0.552	62.004	0.000	33.162	35.329
age	11.0137	0.974	11.313	0.000	9.104	12.923

```
=====
Omnibus:                14.722    Durbin-Watson:           2.050
Prob(Omnibus):           0.001    Jarque-Bera (JB):        15.856
Skew:                    0.189    Prob(JB):                0.000360
Kurtosis:                3.308    Cond. No.                 377.
=====
```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
In [25]: # Check for multicollinearity using VIF.
x_temp = sm.add_constant(X_train)
vif = pd.DataFrame()
vif["Vif Factor"]=[variance_inflation_factor(x_temp.values, i)
                    for i in range(x_temp.values.shape[1])]
vif['features'] = x_temp.columns
print(vif.round(2))
```

	Vif Factor	features
0	20.73	const
1	1.06	spending
2	1.00	remuneration
3	1.06	age

7c) Check model using test data

```
In [26]: mlr=LinearRegression()
mlr.fit(X_train, y_train)
y_pred_mlr = mlr.predict(X_test)
print("MLR Score: ", mlr.score(X_test, y_test))
print("Mean absolute error: ", metrics.mean_absolute_error(y_test, y_pred_mlr))
print("Mean squared error: ", metrics.mean_squared_error(y_test, y_pred_mlr))
```

```
MLR Score: 0.8290723396315803
Mean absolute error: 402.23503056376904
Mean squared error: 277188.70233220584
```

7d) Visualise components

```
In [27]: predictors = ['age', 'remuneration', 'spending']
r2_individual = {}

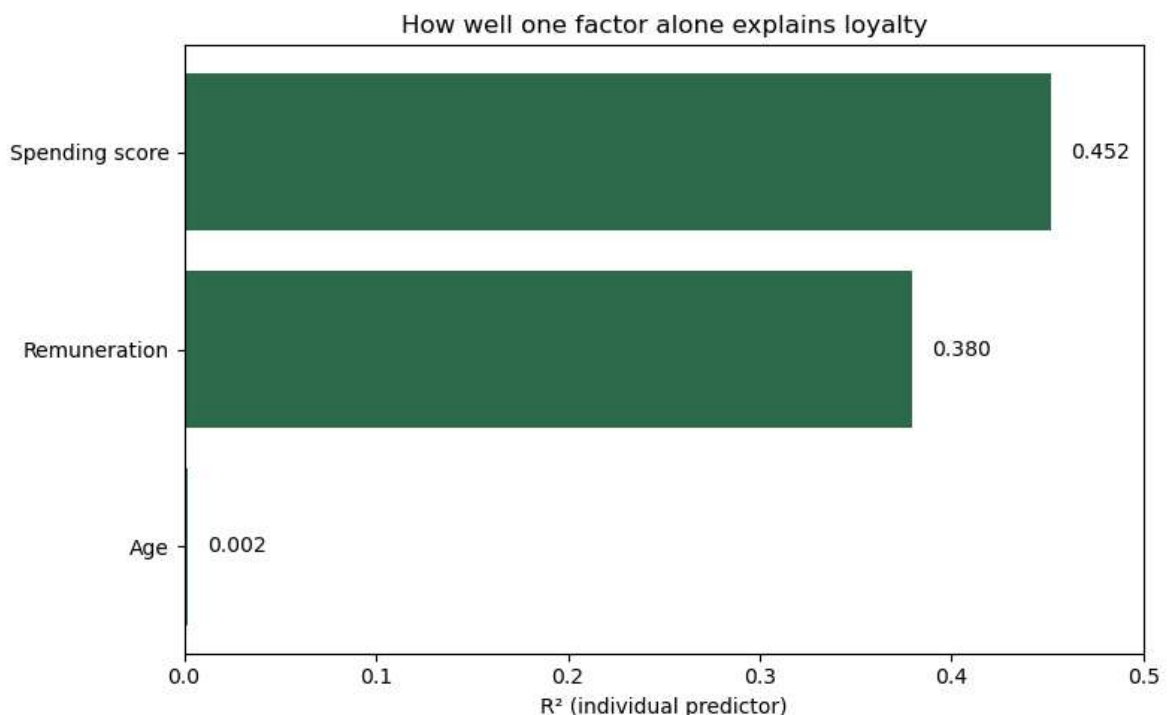
for col in predictors:
    x = turtle[col].values.reshape(-1, 1)
    y = turtle['loyalty'].values.reshape(-1, 1)
    lm = LinearRegression().fit(x, y)
    r2_individual[col] = lm.score(x, y)

# ---- Plot ----
labels = ['Age', 'Remuneration', 'Spending score']
values = [r2_individual['age'], r2_individual['remuneration'], r2_individual['spending']]

fig, ax = plt.subplots(figsize=(8, 5))
ax.barh(labels, values, color='#2D6A4F')
ax.set_xlim(0, 0.5)
ax.set_xlabel('R2 (individual predictor)')
ax.set_title("How well one factor alone explains loyalty")

for i, v in enumerate(values):
    ax.text(v + 0.01, i, f"{v:.3f}", va='center', fontsize=10)

plt.tight_layout()
plt.show()
```



8. Observations and insights from MULTIPLE linear regression

The multiple linear regression model, including age alongside spending and remuneration, achieves an R^2 of 0.842. All three predictors are statistically significant. VIF values remain close to 1 for all predictors, confirming multicollinearity is not a concern.

despite age's addition. The model was validated via an 80/20 train/test split, with test R^2 of 0.829 closely matching training R^2 of 0.842, indicating good generalisation. Although age showed no meaningful standalone relationship with loyalty in the single regression ($R^2=0.002$), it contributes a small but statistically real improvement once combined with spending and remuneration — likely reflecting an interaction effect rather than a direct relationship.

The coefficients indicate that for each additional point in spending score, a customer accumulates approximately 34 loyalty points, and for each additional £1,000 in remuneration, approximately 34 loyalty points are earned — both predictors contribute almost equally to the model. An extra year of age accounts for approximately 11 loyalty points.

The mean absolute error of 402 loyalty points represents a meaningful reduction in prediction error compared to the simple regression models ($SE \approx 950-1,011$). However, the remaining error reflects that spending score, remuneration, and age together account for 84.0% of the variance in loyalty points — suggesting that additional variables may exist that further explain customer loyalty behaviour."

Week 2 assignment: Exploring the structure using decision trees.

The team wants you to use decision trees to attempt to better understand the structure found in the data. You need to grow and prune a decision tree regressor and then visualise and interpret the output. Make sure to comment on the potential usefulness in decision-making processes and your observations regarding the model.

1. Load and prepare the data

```
In [28]: # Import all the necessary packages
import numpy as np
import pandas as pd
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn import metrics
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

import statsmodels.api as sm
import warnings
import matplotlib.pyplot as plt

# Settings for the notebook.
warnings.filterwarnings("ignore")
```

```
In [29]: # Create your new DataFrame.
tgames = pd.read_csv('turtle.csv')
```

```
print(tgames.info())
print(tgames.head(2))
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 2000 entries, 0 to 1999
```

```
Data columns (total 9 columns):
```

#	Column	Non-Null Count	Dtype
0	gender	2000 non-null	object
1	age	2000 non-null	int64
2	remuneration	2000 non-null	float64
3	spending	2000 non-null	int64
4	loyalty	2000 non-null	int64
5	education	2000 non-null	object
6	product	2000 non-null	int64
7	review	2000 non-null	object
8	summary	2000 non-null	object

```
dtypes: float64(1), int64(4), object(4)
```

```
memory usage: 140.8+ KB
```

```
None
```

	gender	age	remuneration	spending	loyalty	education	product
0	Male	18	12.3	39	210	graduate	453
1	Male	23	12.3	81	524	graduate	466

	review
0	When it comes to a DM's screen, the space on t...
1	An Open Letter to GaleForce9*:\n\nYour unpaint...

	summary
0	The fact that 50% of this space is wasted on a...
1	Another worthless Dungeon Master's screen from...

```
In [30]: # Create encoders
gender_encoder = LabelEncoder()
education_encoder = LabelEncoder()

# Encode categorical columns
tgames["gender"] = gender_encoder.fit_transform(tgames["gender"])
tgames["education"] = education_encoder.fit_transform(tgames["education"])

# Specify Y.
y = tgames['loyalty']

# Specify X.
X = tgames[
    [
        'gender',
        'age',
        'remuneration',
        'spending',
        'education',
        'product'
    ]
]
```

```
In [31]: # Review X and Y.
print(X.head())
print(y.head())
```

	gender	age	remuneration	spending	education	product
0	1	18	12.30	39	3	453
1	1	23	12.30	81	3	466
2	0	22	13.12	6	3	254
3	0	25	13.12	77	3	263
4	0	33	13.94	40	3	291

0 210
1 524
2 40
3 562
4 366
Name: loyalty, dtype: int64

2. Create train and test data sets.

```
In [32]: # Split the data into test and train data.
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=42,test_size=0.3
```

3. Create Decision tree regressor

```
In [33]: # Create your decision tree regressor.
```

```
dtr_model = DecisionTreeRegressor(
    max_depth=10,
    random_state=42
)

# Train
dtr_model.fit(X_train, y_train)

# Predict
y_pred = dtr_model.predict(X_test)
```

```
In [34]: # Evaluate the model.
```

```
print("Train R² =", dtr_model.score(X_train, y_train))
print("Test R²  =", dtr_model.score(X_test, y_test))
print("MAE =", metrics.mean_absolute_error(y_test, y_pred))
```

Train R² = 0.9965962791815747

Test R² = 0.990490037630003

MAE = 64.05223394456186

4a. Pruning - Maximum Depth

```
In [35]: train_scores = []
test_scores = []

depths = range(1, 21)

for depth in depths:

    testing_model = DecisionTreeRegressor(
        max_depth=depth,
        random_state=42
    )
```

```

testing_model.fit(X_train, y_train)

train_pred = testing_model.predict(X_train)
test_pred = testing_model.predict(X_test)

train_scores.append(testing_model.score(X_train, y_train))
test_scores.append(testing_model.score(X_test, y_test))

# Plot
plt.figure(figsize=(10,6))

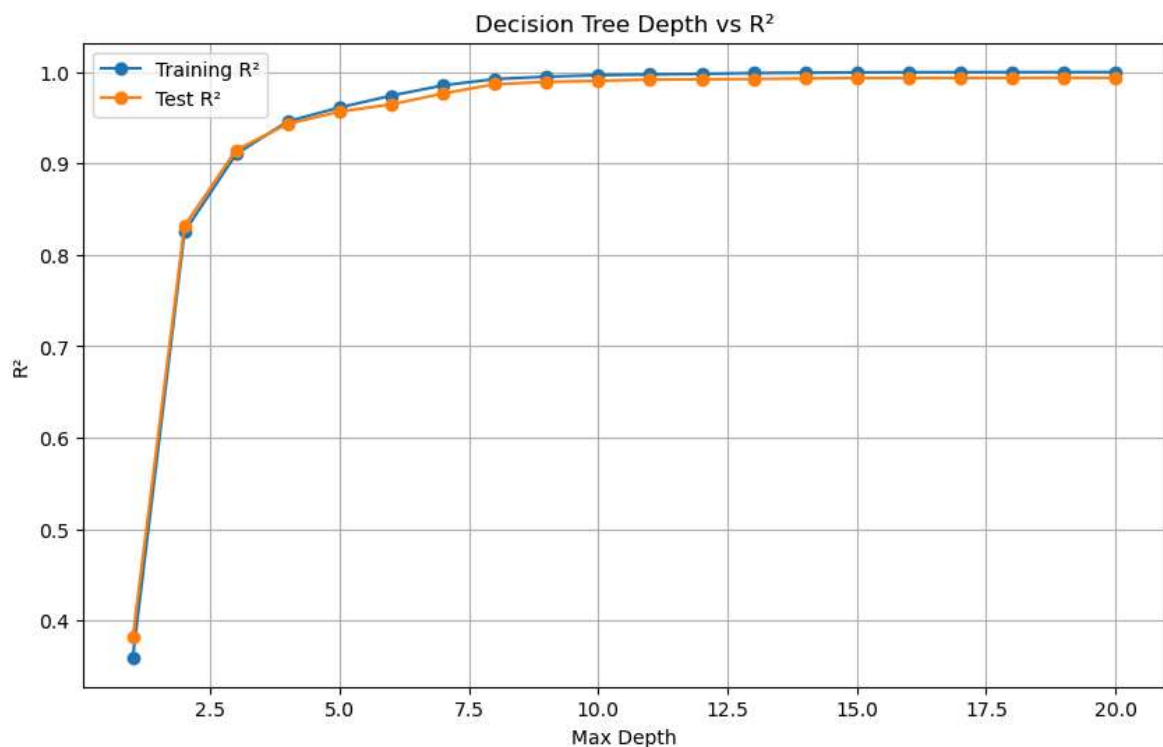
plt.plot(
    depths,
    train_scores,
    marker="o",
    label="Training R2"
)

plt.plot(
    depths,
    test_scores,
    marker="o",
    label="Test R2"
)

plt.xlabel("Max Depth")
plt.ylabel("R2")
plt.title("Decision Tree Depth vs R2")
plt.legend()
plt.grid(True)

plt.show()

```



4b. Pruning - Feature Selection.

Max Depth = 5, All Features

```
In [36]: # Fit a max_depth=5 model with all features to evaluate performance and inform f
model_depth5 = DecisionTreeRegressor(max_depth=5, random_state=42)
model_depth5.fit(X_train, y_train)

print("Train R² =", model_depth5.score(X_train, y_train))
print("Test R²  =", model_depth5.score(X_test, y_test))
print("MAE =", metrics.mean_absolute_error(y_test, model_depth5.predict(X_test)))
```

Train R² = 0.9612665165428146

Test R² = 0.9567099046782617

MAE = 176.80646418311764

Check importances

```
In [37]: importances = pd.DataFrame({
        'feature': X_train.columns,
        'importance': np.round(model_depth5.feature_importances_, 3)
    }).sort_values('importance', ascending=False)
importances
```

Out[37]:

	feature	importance
2	remuneration	0.503
3	spending	0.487
1	age	0.009
0	gender	0.000
4	education	0.000
5	product	0.000

Remove low-importance features

```
In [38]: # Remove Low-importance features (Education, Product, Gender) based on the impor
X_reduced = tgames[['age', 'remuneration', 'spending']]
X_train_r, X_test_r, y_train_r, y_test_r = train_test_split(X_reduced, y, random

model_reduced = DecisionTreeRegressor(max_depth=5, random_state=42)
model_reduced.fit(X_train_r, y_train_r)

print("Train R² =", model_reduced.score(X_train_r, y_train_r))
print("Test R²  =", model_reduced.score(X_test_r, y_test_r))
print("MAE =", metrics.mean_absolute_error(y_test_r, model_reduced.predict(X_test_r)))
```

Train R² = 0.9612665165428146

Test R² = 0.9567099046782617

MAE = 176.80646418311764

Also remove age and re-train

```
In [39]: # Also remove Age and apply min_samples tuning.
X_reduced2 = tgames[['remuneration', 'spending']]
X_train_r2, X_test_r2, y_train_r2, y_test_r2 = train_test_split(X_reduced2, y, r
```

```

model_reduced2 = DecisionTreeRegressor(max_depth=5, min_samples_split=50, min_sa
model_reduced2.fit(X_train_r2, y_train_r2)

print("Train R² =", model_reduced2.score(X_train_r2, y_train_r2))
print("Test R²  =", model_reduced2.score(X_test_r2, y_test_r2))
print("MAE =", metrics.mean_absolute_error(y_test_r2, model_reduced2.predict(X_t

```

Train R² = 0.9657035823607174
 Test R² = 0.9624527097992384
 MAE = 177.0968670781634

5. Fit and plot final model.

```

In [40]: # Specify Y.
y = tgames['loyalty']

# Specify X.
X = tgames[['remuneration', 'spending']]

# Split the data into test and train data.
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=42,test_size=0.3

# Create your decision tree regressor.

final_model = DecisionTreeRegressor(
    max_depth=3,
    min_samples_split = 50,
    min_samples_leaf = 10,
    random_state=42
)

# Train
final_model.fit(X_train, y_train)

# Predict
y_pred = final_model.predict(X_test)

# Evaluate the model.
print("Train R² =", final_model.score(X_train, y_train))
print("Test R²  =", final_model.score(X_test, y_test))
print("MAE =", metrics.mean_absolute_error(y_test, y_pred))

```

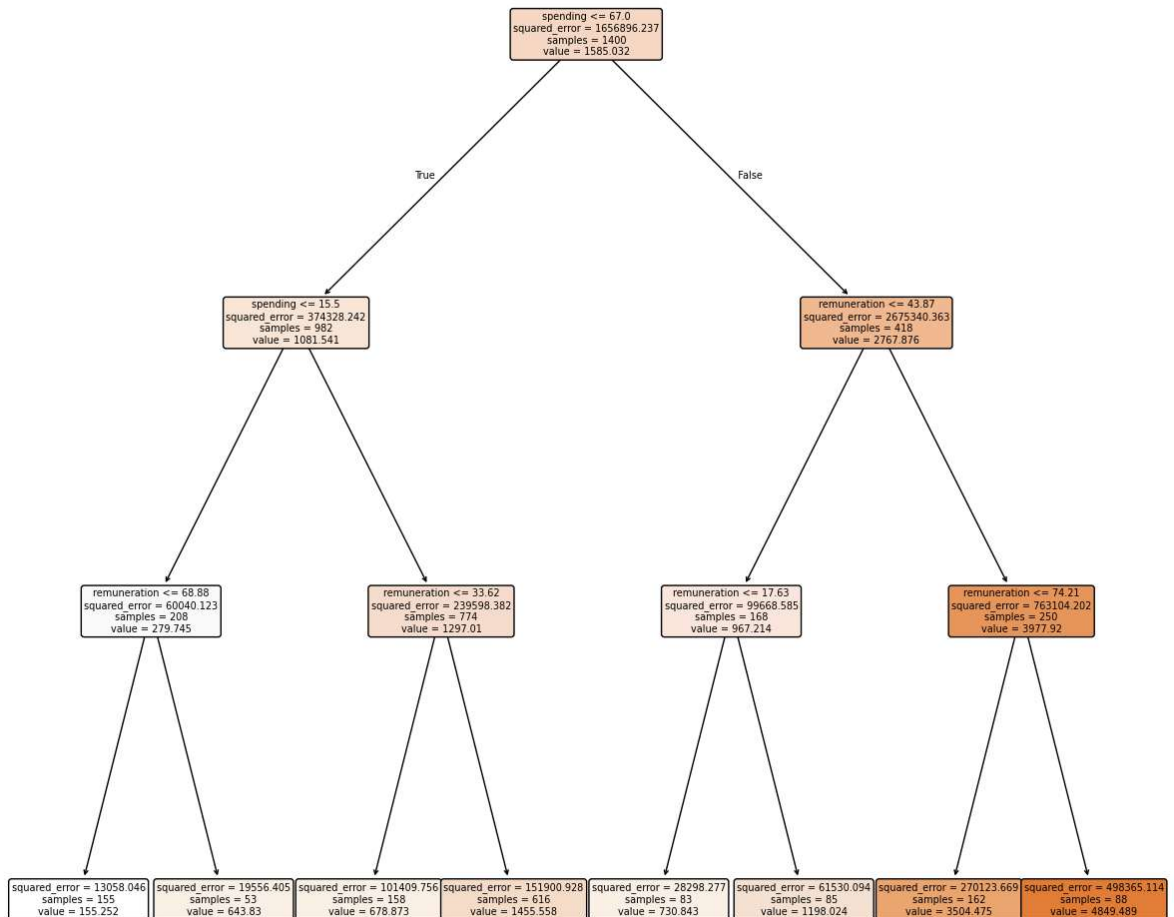
Train R² = 0.9103965014695461
 Test R² = 0.9148271803581075
 MAE = 267.1299251233641

```

In [41]: # Plot the decision tree to create the visualisation:
fig, ax = plt.subplots(figsize=(15, 15))
tree.plot_tree(final_model,
                feature_names=X.columns.tolist(),
                filled=True,
                rounded=True,
                fontsize=7)

# Print the plot with plt.show().
plt.show()

```



6. Discuss: Insights and observations

Original statistics - all numeric/categorical fields - max_depth = 10 Train $R^2 = 0.997$ Test $R^2 = 0.990$ MAE = 64.1

Considering tree depth, both training and test R^2 rise together and stay very close to each other at different max_depth levels, suggesting the model is not overfitting significantly at any depth. The max_depth at 5 is kept at this stage, achieving near-identical predictive performance to a deeper tree, while producing a simpler, more interpretable and business-friendly decision tree.

Considering importances, remuneration, spending and age appear to demonstrate some importance. Education, Product and Gender have negligible impact and so can be removed from the tree.

After removing Education, Product and Gender, Train $R^2 = 0.961$ Test $R^2 = 0.957$ MAE = 177

Although age contributes a small, statistically significant effect in the linear regression, the decision tree's own feature importance ranking shows it adds negligible predictive

value here. Age was removed to test changes. At the same time, min-split was set to 50 and min-leaf to 10. Train $R^2 = 0.966$ Test $R^2 = 0.962$ MAE = 177

Hence minimal changes to overall accuracy. Age was therefore also removed.

However, this still results in an over-complicated tree. Decision made at this stage to cut max_depth to 3. Although errors increase, the R^2 value remains high. However, the tree becomes much more manageable and hence implementable for the business. Train $R^2 = 0.910$ Test $R^2 = 0.915$ MAE = 267

The final tree reveals that remuneration and spending score are sufficient to segment customers into loyalty point groups, providing a simple, actionable framework that Turtle Games marketing can use to target high-value customers.

Week 3 assignment: Clustering with *k*-means using Python

The marketing department also wants to better understand the usefulness of remuneration and spending scores but do not know where to begin. You are tasked to identify groups within the customer base that can be used to target specific market segments. Use *k*-means clustering to identify the optimal number of clusters and then apply and plot the data using the created segments.

1. Load and explore the data

```
In [42]: # Import necessary Libraries.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.metrics import accuracy_score
from scipy.spatial.distance import cdist

import warnings
warnings.filterwarnings('ignore')
```

```
In [43]: # Load the CSV file(s) as df2.
df2 = pd.read_csv('turtle.csv')

# View DataFrame.
print(df2.info())
df2.head(4)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 9 columns):
#   Column                Non-Null Count  Dtype
---  -
0   gender                2000 non-null   object
1   age                   2000 non-null   int64
2   remuneration          2000 non-null   float64
3   spending              2000 non-null   int64
4   loyalty               2000 non-null   int64
5   education             2000 non-null   object
6   product               2000 non-null   int64
7   review                2000 non-null   object
8   summary              2000 non-null   object
dtypes: float64(1), int64(4), object(4)
memory usage: 140.8+ KB
None
```

Out[43]:

	gender	age	remuneration	spending	loyalty	education	product	rev
0	Male	18	12.30	39	210	graduate	453	When it comes t DM's screen, space or
1	Male	23	12.30	81	524	graduate	466	An Open Lette GaleForce9*:\n\nY unpai
2	Female	22	13.12	6	40	graduate	254	Nice art, r printing. Why panels are
3	Female	25	13.12	77	562	graduate	263	Amazing k Bought it as a gift our new

In [44]:

```
# Drop unnecessary columns.
df3 = df2[['remuneration', 'spending']].copy()

# View DataFrame.
df3.head(4)
```

Out[44]:

	remuneration	spending
0	12.30	39
1	12.30	81
2	13.12	6
3	13.12	77

In [45]: *# Descriptive statistics.*
df3.describe()

Out[45]:

	remuneration	spending
count	2000.000000	2000.000000
mean	48.079060	50.000000
std	23.123984	26.094702
min	12.300000	1.000000
25%	30.340000	32.000000
50%	47.150000	50.000000
75%	63.960000	73.000000
max	112.340000	99.000000

In [46]: *# Check for outliers*
Check for outliers in numerical data columns
def count_outliers_iqr(df):
 outlier_counts = {}

 for col in df.select_dtypes(include='number').columns:
 Q1 = df[col].quantile(0.25)
 Q3 = df[col].quantile(0.75)
 IQR = Q3 - Q1

 lower = Q1 - 1.5 * IQR
 upper = Q3 + 1.5 * IQR

 outliers = ((df[col] < lower) | (df[col] > upper)).sum()
 outlier_counts[col] = outliers

 return pd.Series(outlier_counts)

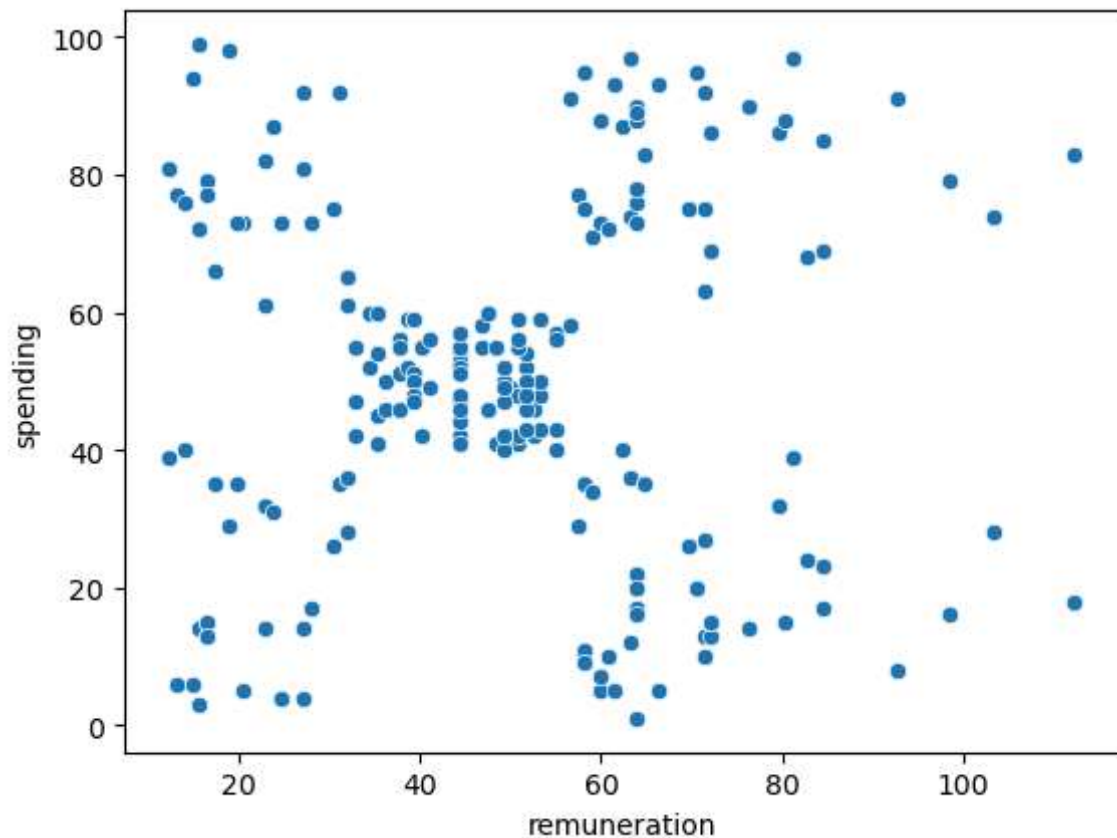
print(count_outliers_iqr(df3))

remuneration 0
spending 0
dtype: int64

2. Plot

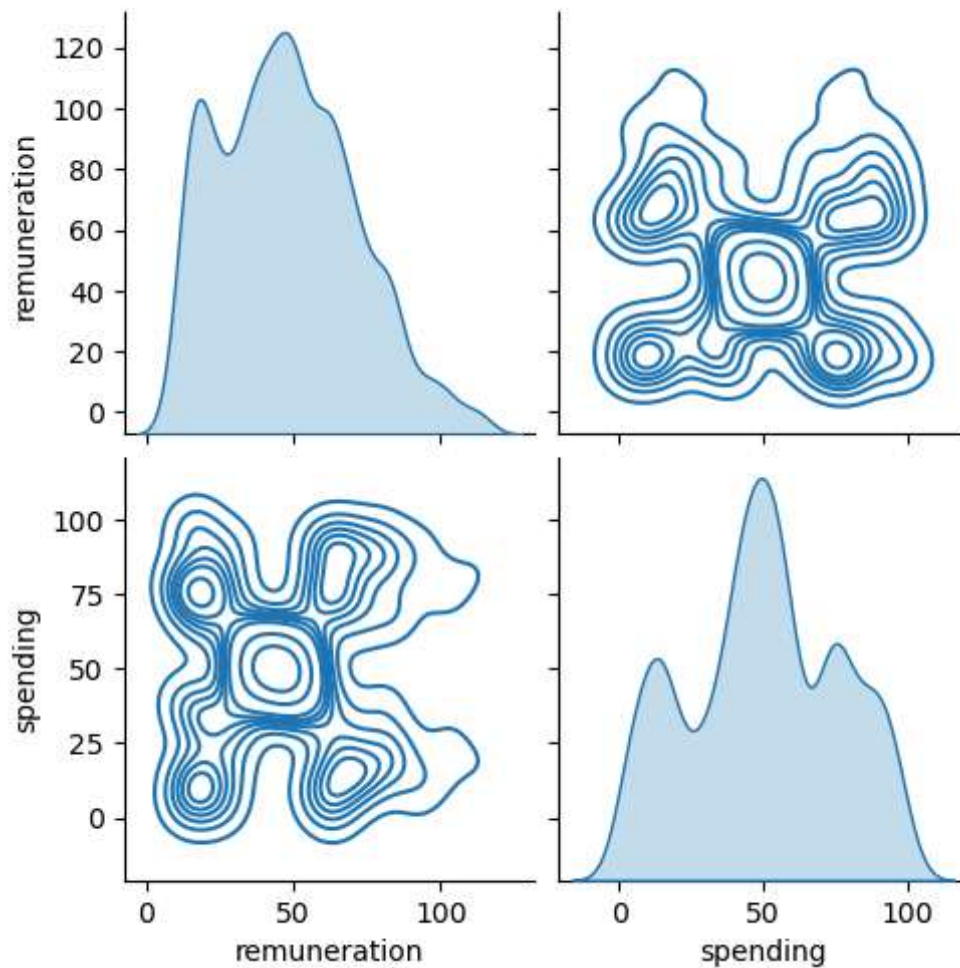
In [47]: *# Create a scatterplot with Seaborn.*
sns.scatterplot(x='remuneration', y='spending', data=df3)

Out[47]: <Axes: xlabel='remuneration', ylabel='spending'>



```
In [48]: # Create a pairplot with Seaborn.  
sns.pairplot(df3, kind="kde", x_vars=['remuneration', 'spending'], y_vars=['remu
```

Out[48]: <seaborn.axisgrid.PairGrid at 0x269678670e0>



3. Elbow and silhouette methods

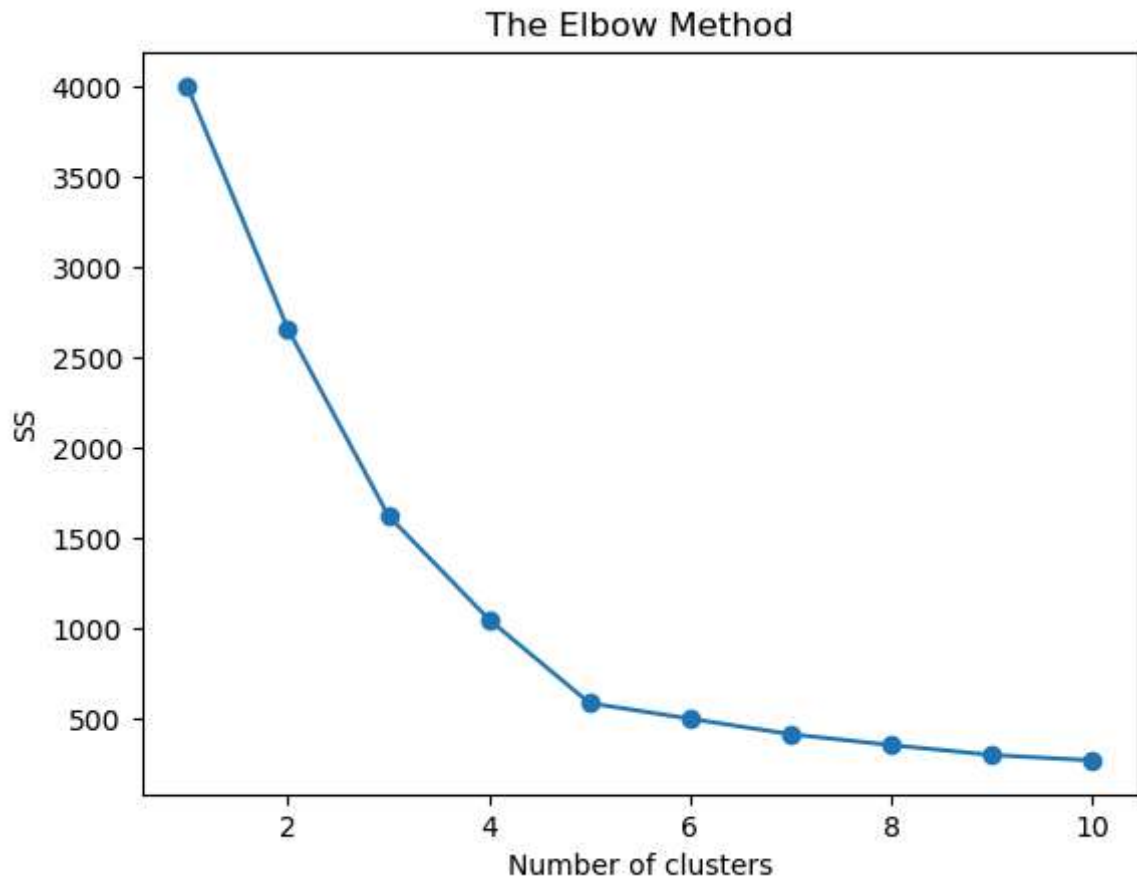
```
In [49]: # Scale variables
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df3)
```

```
In [50]: # Determine the number of clusters: Elbow method.
ss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i,
                    init = 'k-means++',
                    max_iter = 500,
                    n_init = 10,
                    random_state = 42)
    kmeans.fit(df_scaled)
    ss.append(kmeans.inertia_)

plt.plot(range(1, 11),
         ss,
         marker='o')

plt.title("The Elbow Method")
plt.xlabel("Number of clusters")
plt.ylabel("SS")

plt.show()
```



```
In [51]: # Determine the number of clusters: Silhouette method.
sil = []
kmax = 10

for k in range(2, kmax+1):
    kmeans = KMeans(n_clusters = k,
                    init = 'k-means++',
                    max_iter = 500,
                    n_init = 10,
                    random_state = 42)

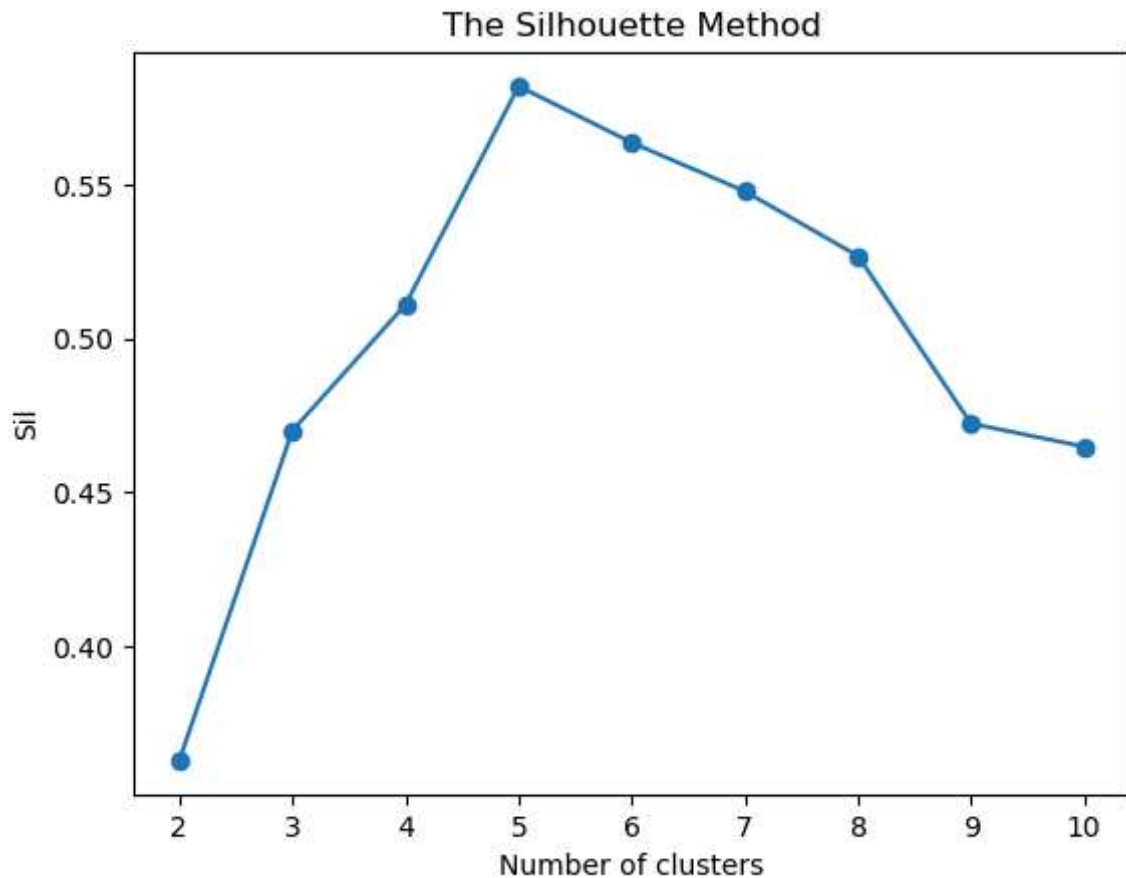
    labels = kmeans.fit_predict(df_scaled)

    sil.append(silhouette_score(df_scaled, labels))

# Plot the silhouette method.
plt.plot(range(2, kmax+1),
        sil,
        marker='o')

plt.title("The Silhouette Method")
plt.xlabel("Number of clusters")
plt.ylabel("Sil")

plt.show()
```



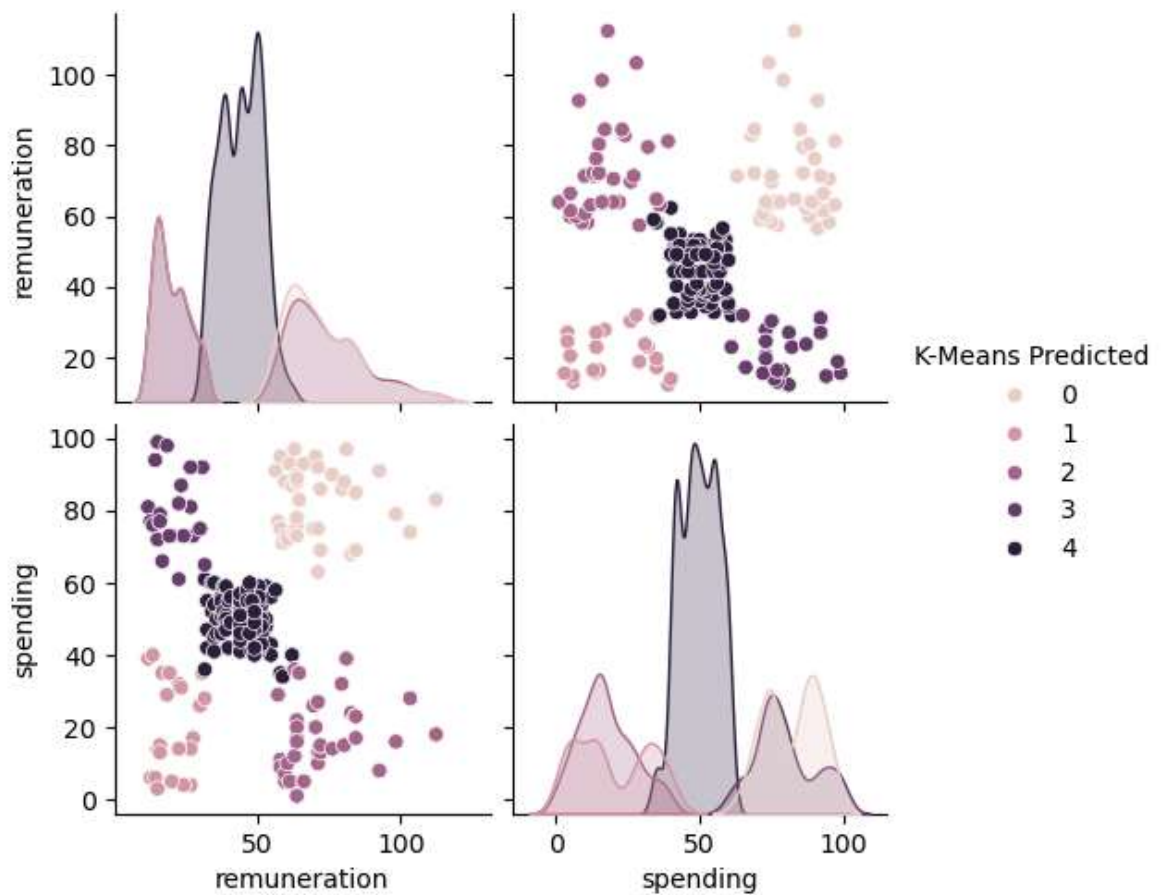
4. Evaluate k-means model at different values of k

```
In [52]: # Evaluating at k = 5
kmeans = KMeans(n_clusters = 5,
                 max_iter = 15000,
                 init='k-means++',
                 random_state=42).fit(df_scaled)

clusters = kmeans.labels_
df3['K-Means Predicted'] = clusters

# Plot the predicted.
sns.pairplot(df3,
              hue='K-Means Predicted',
              diag_kind= 'kde')

plt.show()
# Check the number of observations per predicted class.
df3['K-Means Predicted'].value_counts()
```

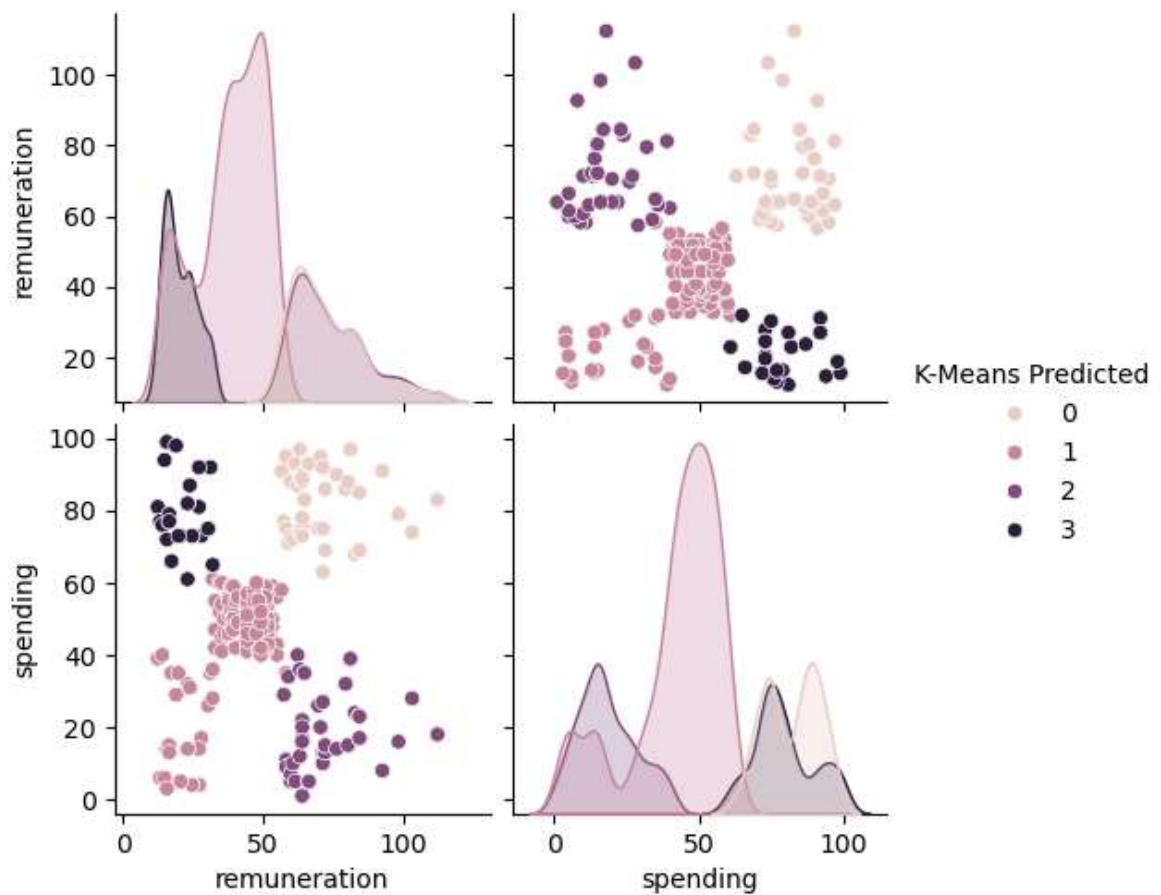
```
Out[52]: K-Means Predicted
4      774
0      356
2      330
1      271
3      269
Name: count, dtype: int64
```

```
In [53]: # Evaluating at k = 4
kmeans = KMeans(n_clusters = 4,
                max_iter = 15000,
                init='k-means++',
                random_state=42).fit(df_scaled)

clusters = kmeans.labels_
df3['K-Means Predicted'] = clusters

# Plot the predicted.
sns.pairplot(df3,
             hue='K-Means Predicted',
             diag_kind= 'kde')

plt.show()
# Check the number of observations per predicted class.
df3['K-Means Predicted'].value_counts()
```



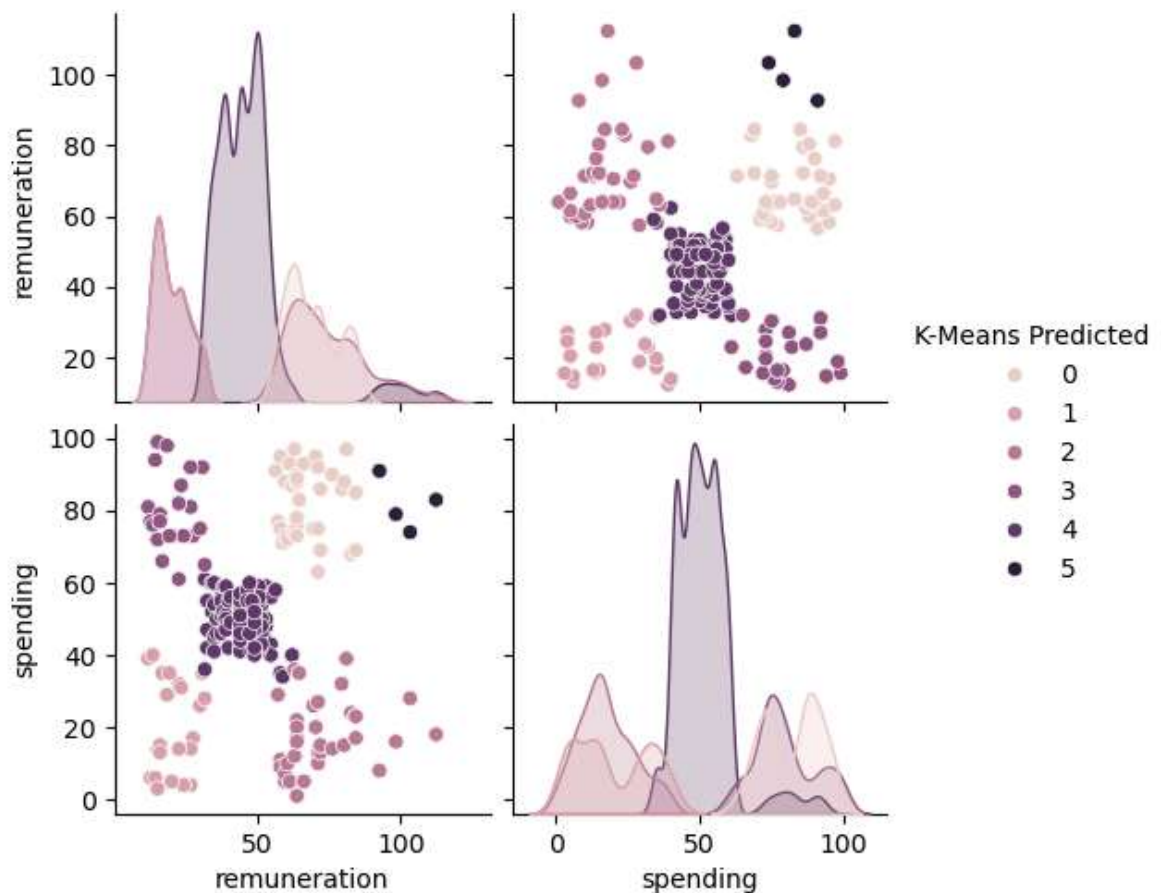
```
Out[53]: K-Means Predicted
1      1031
0       356
2       344
3       269
Name: count, dtype: int64
```

```
In [54]: # Evaluating at k = 6
kmeans = KMeans(n_clusters = 6,
                max_iter = 15000,
                init='k-means++',
                random_state=42).fit(df_scaled)

clusters = kmeans.labels_
df3['K-Means Predicted'] = clusters

# Plot the predicted.
sns.pairplot(df3,
             hue='K-Means Predicted',
             diag_kind= 'kde')

plt.show()
# Check the number of observations per predicted class.
df3['K-Means Predicted'].value_counts()
```



```
Out[54]: K-Means Predicted
4      774
2      330
0      311
1      271
3      269
5       45
Name: count, dtype: int64
```

5. Fit final model and justify your choice

k=5 chosen for the final model. This value gave a clear peak using the silhouette method, while still being part of the elbow. Visually, the groupings seem very clear on the pairplot, and all clusters are of a decent size.

```
In [55]: # Apply the final model.
kmeans = KMeans(n_clusters = 5,
                max_iter = 15000,
                init='k-means++',
                random_state=42).fit(df_scaled)

clusters = kmeans.labels_
df3['K-Means Predicted'] = clusters
```

```
In [56]: # Check the number of observations per predicted class.
df3['K-Means Predicted'].value_counts()
```

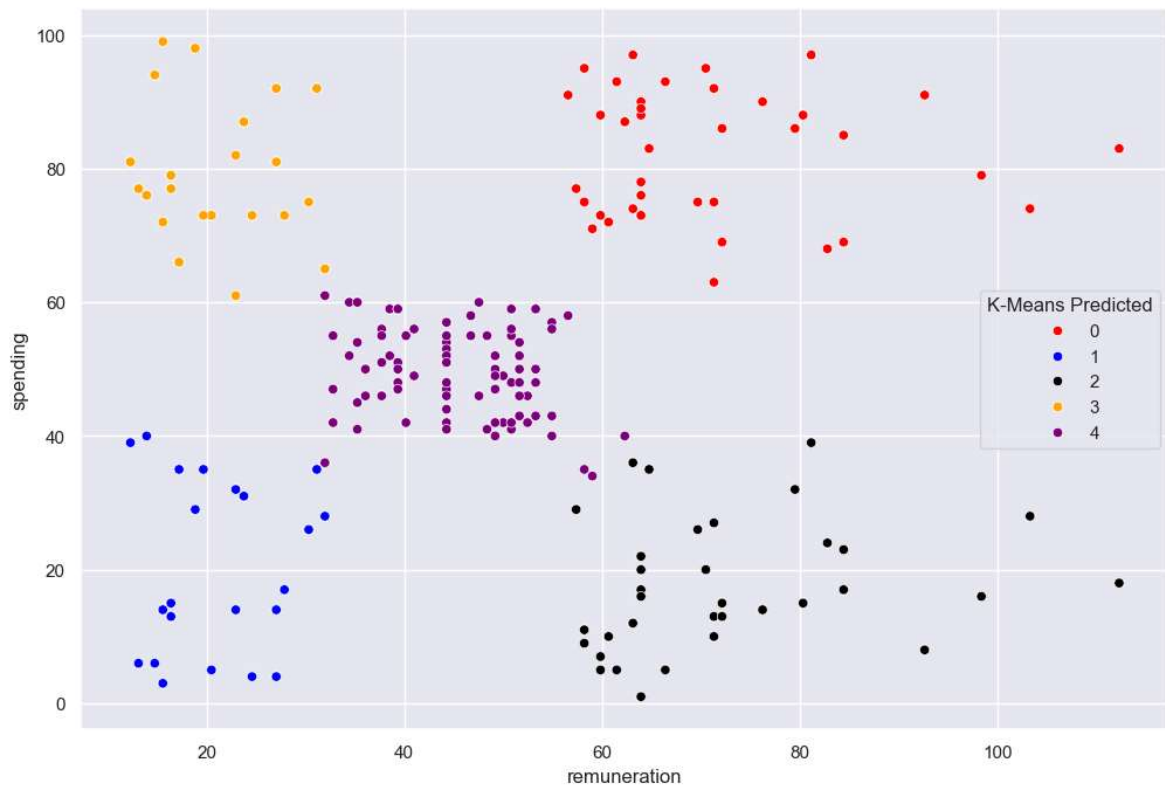
```
Out[56]: K-Means Predicted
4      774
0      356
2      330
1      271
3      269
Name: count, dtype: int64
```

6. Plot and interpret the clusters

```
In [57]: # Visualising the clusters.
# Set plot size.
sns.set(rc = {'figure.figsize':(12, 8)})

sns.scatterplot(x='remuneration' ,
                y='spending',
                data=df3,
                hue='K-Means Predicted',
                palette=['red', 'blue', 'black', 'orange', 'purple'])
```

```
Out[57]: <Axes: xlabel='remuneration', ylabel='spending'>
```



```
In [58]: # Calculate averages of main variables
print(df3.groupby('K-Means Predicted')[['remuneration', 'spending']].mean().round(2))
```

	remuneration	spending
K-Means Predicted		
0	73.24	82.01
1	20.42	19.76
2	74.83	17.42
3	20.35	79.42
4	44.42	49.53

7. Discuss: Insights and observations

Report Findings

Kmeans clustering was applied to the target columns of 'remuneration' and 'spending'.

Although the ranges were similar for both variable (remuneration, 12.3 - 112.3 and spending, 1 - 99), StandardScaler was still used to ensure one variable did not dominate the other.

Neither variables had outliers. Both the 'elbow' method and 'silhouette' method suggested 5 clusters. Although other values of k were explored, k = 5 was chosen for the final model.

The results were:

Group	Size	Mean Renumeration	Mean Spending	Description
0	356	73.24	82.01	High remuneration, High Spending
1	271	20.42	19.76	Low remuneration, Low Spending
2	330	74.83	17.42	High remuneration, Low Spending
3	269	20.35	79.42	Low remuneration, High Spending
4	774	44.42	49.53	Avg remuneration, Avg Spending

All clusters have a good size.

Week 4 assignment: NLP using Python

Customer reviews were downloaded from the website of Turtle Games. This data will be used to steer the marketing department on how to approach future campaigns.

Therefore, the marketing department asked you to identify the 15 most common words used in online product reviews. They also want to have a list of the top 20 positive and negative reviews received from the website. Therefore, you need to apply NLP on the data set.

1. Load and explore the data

```
In [59]: # Import all the necessary packages.
import pandas as pd
import numpy as np
import nltk
import os
import matplotlib.pyplot as plt
import seaborn as sns

# nltk.download('punkt').
# nltk.download('stopwords').
```

```

from wordcloud import WordCloud
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
from nltk.corpus import stopwords
from textblob import TextBlob
from scipy.stats import norm
import string

# Import Counter.
from collections import Counter

import warnings
warnings.filterwarnings('ignore')

```

```

In [60]: # Load the data set as df3.
df3=pd.read_csv('turtle.csv')

# View DataFrame.
print(df3.shape)
print(df3.head(5))

```

```

(2000, 9)

```

	gender	age	remuneration	spending	loyalty	education	product	\
0	Male	18	12.30	39	210	graduate	453	
1	Male	23	12.30	81	524	graduate	466	
2	Female	22	13.12	6	40	graduate	254	
3	Female	25	13.12	77	562	graduate	263	
4	Female	33	13.94	40	366	graduate	291	


```

review \
0 When it comes to a DM's screen, the space on t...
1 An Open Letter to GaleForce9*:\n\nYour unpaint...
2 Nice art, nice printing. Why two panels are f...
3 Amazing buy! Bought it as a gift for our new d...
4 As my review of GF9's previous screens these w...

```



```

summary
0 The fact that 50% of this space is wasted on a...
1 Another worthless Dungeon Master's screen from...
2 pretty, but also pretty useless
3 Five Stars
4 Money trap

```

```

In [61]: # Explore data set.
print(df3.info())

```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 9 columns):
#   Column                Non-Null Count  Dtype
---  -
0   gender                2000 non-null  object
1   age                   2000 non-null  int64
2   remuneration          2000 non-null  float64
3   spending              2000 non-null  int64
4   loyalty               2000 non-null  int64
5   education             2000 non-null  object
6   product               2000 non-null  int64
7   review                2000 non-null  object
8   summary               2000 non-null  object
dtypes: float64(1), int64(4), object(4)
memory usage: 140.8+ KB
None
```

```
In [62]: # Keep necessary columns. Drop unnecessary columns.
df3=df3[['review','summary']]

# View DataFrame.
print(df3.shape)
print(df3.head(5))
```

```
(2000, 2)
```

```

                                review \
0  When it comes to a DM's screen, the space on t...
1  An Open Letter to GaleForce9*:\n\nYour unpaint...
2  Nice art, nice printing.  Why two panels are f...
3  Amazing buy! Bought it as a gift for our new d...
4  As my review of GF9's previous screens these w...

                                summary
0  The fact that 50% of this space is wasted on a...
1  Another worthless Dungeon Master's screen from...
2                                pretty, but also pretty useless
3                                Five Stars
4                                Money trap
```

```
In [63]: # Determine if there are any missing values.
df3.isnull().sum()
```

```
Out[63]: review      0
summary      0
dtype: int64
```

```
In [64]: # Drop duplicates in both columns.
df3 = df3.drop_duplicates(subset=['review', 'summary'])

# View DataFrame.
df3.head()
```


Out[64]:

	review	summary
0	When it comes to a DM's screen, the space on t...	The fact that 50% of this space is wasted on a...
1	An Open Letter to GaleForce9*:\n\nYour unpaint...	Another worthless Dungeon Master's screen from...
2	Nice art, nice printing. Why two panels are f...	pretty, but also pretty useless
3	Amazing buy! Bought it as a gift for our new d...	Five Stars
4	As my review of GF9's previous screens these w...	Money trap

2. Prepare the data for NLP

2a) Change to lower case and join the elements in each of the columns respectively (review and summary)

```
In [65]: # Review: Change all to lower case and join with a space.
review_text = ' '.join(df3['review'].astype(str)).lower()
```

```
In [66]: # Summary: Change all to lower case and join with a space.
summary_text = ' '.join(df3['summary'].astype(str)).lower()
```

2b) Replace punctuation in each of the columns respectively (review and summary)

```
In [67]: # Replace all the punctuations in review column.
review_text = review_text.translate(str.maketrans('', '', string.punctuation))

# View output.
print(review_text[:200])
```

when it comes to a dms screen the space on the screen itself is at an absolute premium the fact that 50 of this space is wasted on art and not terribly informative or needed art as well makes it compl

```
In [68]: # Replace all the punctuations in summary column.
summary_text = summary_text.translate(str.maketrans('', '', string.punctuation))

# View output.
print(summary_text[:200])
```

the fact that 50 of this space is wasted on art and not terribly informative or needed art another worthless dungeon masters screen from galeforce9 pretty but also pretty useless five stars money tra

```
In [69]: # Create new dataframe
df4 = pd.DataFrame([review_text, summary_text], columns=['review', 'summary'])
```


3. Tokenise and create wordclouds

```
In [70]: # Apply tokenization to the review column
df4['review_tokens'] = df4['review'].astype(str).apply(word_tokenize)

# Apply tokenization to the summary column
df4['summary_tokens'] = df4['summary'].astype(str).apply(word_tokenize)

# View the new tokenized columns
df4[['review_tokens', 'summary_tokens']]
```

```
Out[70]:
```

	review_tokens	summary_tokens
0	[when, it, comes, to, a, dms, screen, the, spa...	[the, fact, that, 50, of, this, space, is, was...

```
In [71]: # Review: Create a word cloud.
# Set the colour palette.
sns.set(color_codes=True)

# Create a WordCloud object.
word_cloud_r = WordCloud(width = 1600, height = 900,
                          background_color = 'white',
                          colormap = 'plasma',
                          stopwords = 'none',
                          min font size = 10).generate(df4['review'].iloc[0])
```

```
In [72]: # Plot the WordCloud image.
plt.figure(figsize = (16, 9), facecolor = None)
plt.imshow(word_cloud_r)
plt.axis('off')
plt.tight_layout(pad = 0)
plt.title("Word Cloud from Review")
plt.show()
```



```
In [73]: # Summary: Create a word cloud.  
# Set the colour palette.
```

```
sns.set(color_codes=True)

# Create a WordCloud object.
word_cloud_s = WordCloud(width = 1600, height = 900,
                          background_color = 'white',
                          colormap = 'plasma',
                          stopwords = 'none',
                          min_font_size = 10).generate(df4['summary'].iloc[0])
```

```
In [74]: # Summary: Plot the WordCloud image.
plt.figure(figsize = (16, 9), facecolor = None)
plt.imshow(word_cloud_s)
plt.axis('off')
plt.tight_layout(pad = 0)
plt.title("Word Cloud from Summary")
plt.show()
```



4. Frequency distribution and polarity

4a) Create frequency distribution

```
In [75]: # Determine the frequency distribution.
review_freq_dist = FreqDist(df4['review_tokens'].iloc[0])
summary_freq_dist = FreqDist(df4['summary_tokens'].iloc[0])

# View output.
print(review_freq_dist)
print(review_freq_dist.most_common(15))
print(summary_freq_dist)
print(summary_freq_dist.most_common(15))
```

```
<FreqDist with 7520 samples and 112166 outcomes>
[('the', 5452), ('and', 3233), ('to', 3163), ('a', 3160), ('of', 2488), ('i', 2091), ('it', 2085), ('is', 1782), ('this', 1776), ('game', 1682), ('for', 1545), ('with', 1265), ('you', 1177), ('in', 1148), ('that', 1067)]
<FreqDist with 1550 samples and 9323 outcomes>
[('stars', 448), ('five', 363), ('game', 319), ('great', 295), ('the', 261), ('a', 240), ('for', 232), ('fun', 218), ('to', 192), ('and', 168), ('it', 150), ('of', 139), ('this', 135), ('is', 103), ('love', 93)]
```

4b) Remove alphanumeric characters and stopwords

```
In [76]: # Delete all the alphanumeric characters.
review_alpha = [word for word in df4['review_tokens'].iloc[0] if word.isalpha()]
summary_alpha = [word for word in df4['summary_tokens'].iloc[0] if word.isalpha()]

# View output.
print(review_alpha[:15])
print(summary_alpha[:15])
```

```
['when', 'it', 'comes', 'to', 'a', 'dms', 'screen', 'the', 'space', 'on', 'the',
'screen', 'itself', 'is', 'at']
['the', 'fact', 'that', 'of', 'this', 'space', 'is', 'wasted', 'on', 'art', 'and', 'not', 'terribly', 'informative', 'or']
```

```
In [77]: # Remove all the stopwords
stop_words = set(stopwords.words('english'))

review_clean = [word for word in review_alpha if word not in stop_words]
summary_clean = [word for word in summary_alpha if word not in stop_words]

# View output.
print(review_clean[:15])
print(summary_clean[:15])
```

```
['comes', 'dms', 'screen', 'space', 'screen', 'absolute', 'premium', 'fact', 'space', 'wasted', 'art', 'terribly', 'informative', 'needed', 'art']
['fact', 'space', 'wasted', 'art', 'terribly', 'informative', 'needed', 'art', 'another', 'worthless', 'dungeon', 'masters', 'screen', 'pretty', 'also']
```

4c) Create wordcloud without stopwords

```
In [78]: # Create a wordcloud without stop words.
# Summary: Create a word cloud.
# Set the colour palette.
sns.set(color_codes=True)

# Create a wordcloud without stop words.
review_clean_text = ' '.join(review_clean)
summary_clean_text = ' '.join(summary_clean)

word_cloud_r_clean = WordCloud(width=1600, height=900, background_color='white',
                                colormap='plasma', min_font_size=10).generate(review_clean_text)

word_cloud_s_clean = WordCloud(width=1600, height=900, background_color='white',
                                colormap='plasma', min_font_size=10).generate(summary_clean_text)
```

```
In [79]: # Plot the wordcloud image.
plt.figure(figsize=(16, 9), facecolor=None)
plt.imshow(word_cloud_r_clean)
```


[illegible]

```
In [80]: # Determine the 15 most common words.
review_freq_clean = FreqDist(review_clean)
summary_freq_clean = FreqDist(summary_clean)
```

```
# Create dataframe for ease of display.
review_top15 = pd.DataFrame(review_freq_clean.most_common(15), columns=['word',
summary_top15 = pd.DataFrame(summary_freq_clean.most_common(15), columns=['word'

# View output.
print(review_top15)
print(summary_top15)
```

	word	count
0	game	1682
1	great	589
2	fun	553
3	one	530
4	play	502
5	like	414
6	love	325
7	really	319
8	get	319
9	cards	301
10	tiles	297
11	good	293
12	time	291
13	would	280
14	book	273

	word	count
0	stars	448
1	five	363
2	game	319
3	great	295
4	fun	218
5	love	93
6	good	92
7	four	58
8	like	54
9	expansion	52
10	kids	50
11	cute	45
12	book	43
13	one	38
14	awesome	36

5. Review polarity and sentiment: Plot histograms of polarity (use 15 bins) and sentiment scores for the respective columns.

```
In [81]: # Provided function.
def generate_polarity(comment):
    '''Extract polarity score (-1 to +1) for each comment'''
    return TextBlob(comment).sentiment[0]
```

```
In [82]: # Determine polarity of both columns.
# Determine polarity of both columns.
df3['review_polarity'] = df3['review'].apply(generate_polarity)
df3['summary_polarity'] = df3['summary'].apply(generate_polarity)

# View output.
print(df3[['review', 'review_polarity', 'summary', 'summary_polarity']].head())
```

```
# Determine sentiment (subjectivity) scores for both columns.
df3['review_subjectivity'] = df3['review'].apply(lambda x: TextBlob(x).sentiment
df3['summary_subjectivity'] = df3['summary'].apply(lambda x: TextBlob(x).sentime

# View output.
print(df3[['review', 'review_subjectivity', 'summary', 'summary_subjectivity']]).
```

	review	review_polarity \
0	When it comes to a DM's screen, the space on t...	-0.036111
1	An Open Letter to GaleForce9*:\n\nYour unpaint...	0.035952
2	Nice art, nice printing. Why two panels are f...	0.116640
3	Amazing buy! Bought it as a gift for our new d...	0.628788
4	As my review of GF9's previous screens these w...	-0.316667

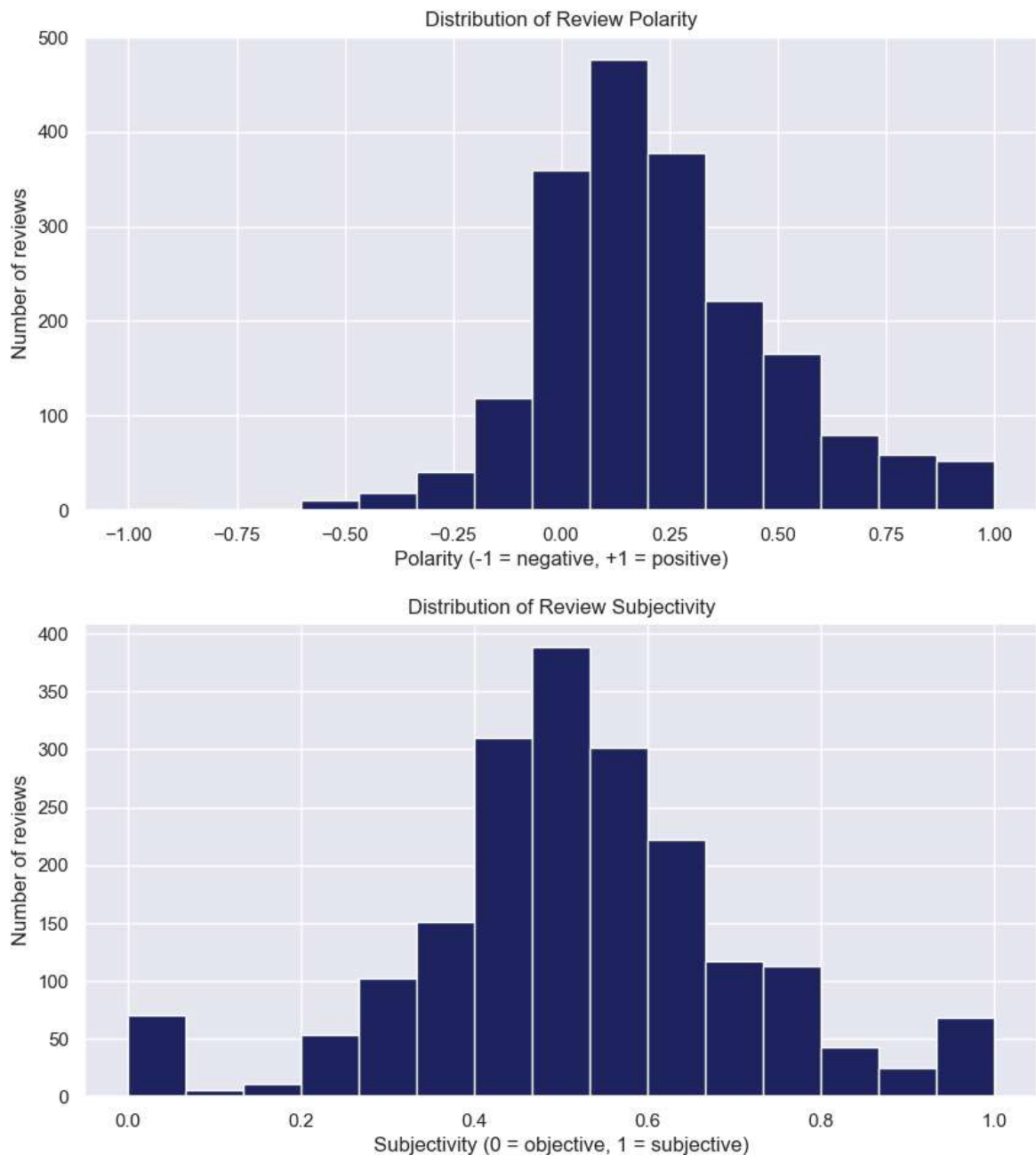
	summary	summary_polarity
0	The fact that 50% of this space is wasted on a...	0.15
1	Another worthless Dungeon Master's screen from...	-0.80
2	pretty, but also pretty useless	0.00
3	Five Stars	0.00
4	Money trap	0.00

	review	review_subjectivity \
0	When it comes to a DM's screen, the space on t...	0.486111
1	An Open Letter to GaleForce9*:\n\nYour unpaint...	0.442976
2	Nice art, nice printing. Why two panels are f...	0.430435
3	Amazing buy! Bought it as a gift for our new d...	0.784848
4	As my review of GF9's previous screens these w...	0.316667

	summary	summary_subjectivity
0	The fact that 50% of this space is wasted on a...	0.500000
1	Another worthless Dungeon Master's screen from...	0.900000
2	pretty, but also pretty useless	0.733333
3	Five Stars	0.000000
4	Money trap	0.000000

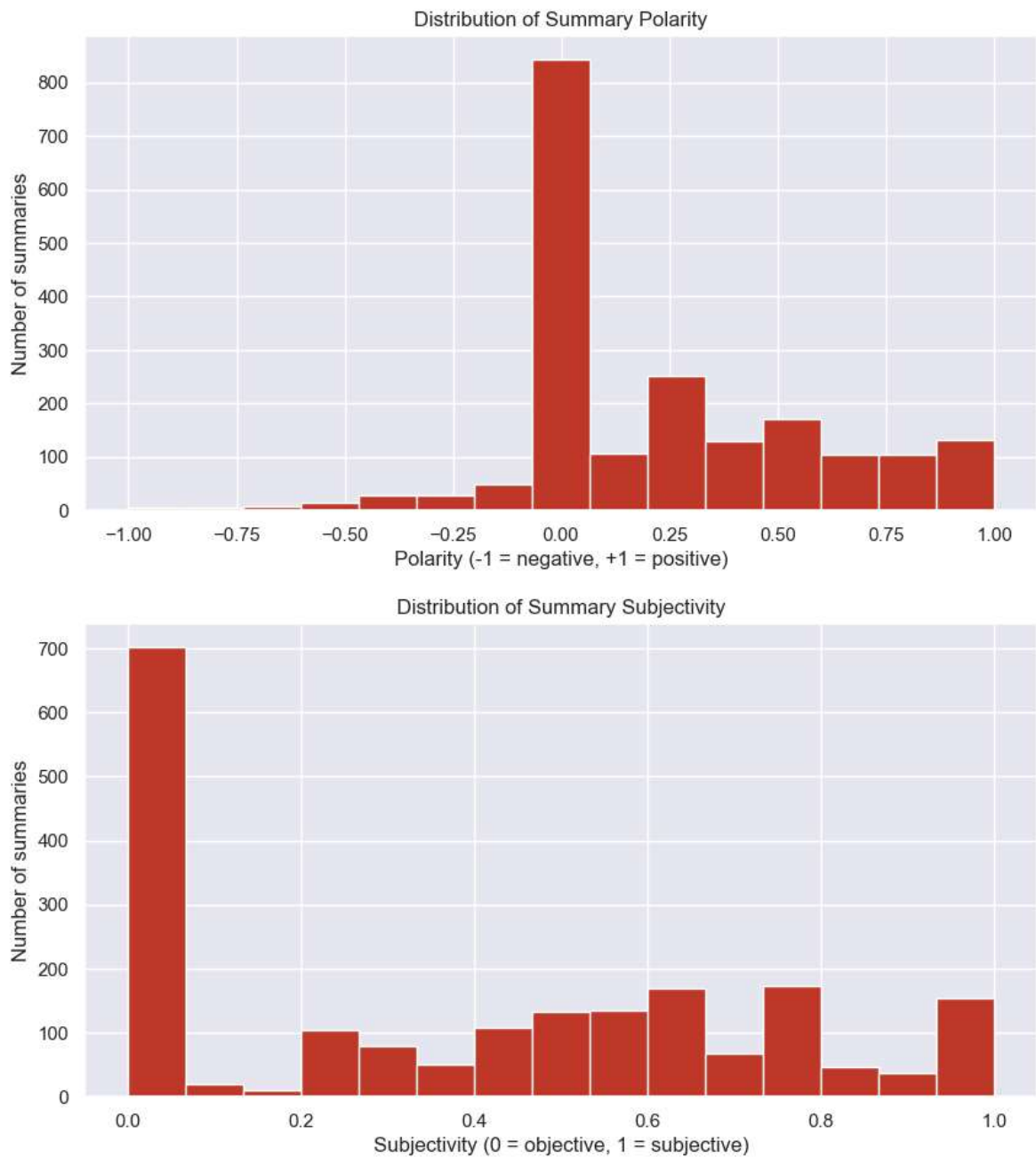
```
In [83]: # Review: Create a histogram plot with bins = 15.
# Histogram of polarity
plt.figure(figsize=(10, 5))
df3['review_polarity'].plot(kind='hist', bins=15, color='#1E2761', edgecolor='wh
plt.title('Distribution of Review Polarity')
plt.xlabel('Polarity (-1 = negative, +1 = positive)')
plt.ylabel('Number of reviews')
plt.show()

# Histogram of Sentiment Score
plt.figure(figsize=(10, 5))
df3['review_subjectivity'].plot(kind='hist', bins=15, color='#1E2761', edgecolor
plt.title('Distribution of Review Subjectivity')
plt.xlabel('Subjectivity (0 = objective, 1 = subjective)')
plt.ylabel('Number of reviews')
plt.show()
```



```
In [84]: # Summary: Create a histogram plot with bins = 15.
# Histogram of polarity
plt.figure(figsize=(10, 5))
df3['summary_polarity'].plot(kind='hist', bins=15, color='#C0392B', edgecolor='w')
plt.title('Distribution of Summary Polarity')
plt.xlabel('Polarity (-1 = negative, +1 = positive)')
plt.ylabel('Number of summaries')
plt.show()

# Histogram of sentiment score
plt.figure(figsize=(10, 5))
df3['summary_subjectivity'].plot(kind='hist', bins=15, color='#C0392B', edgecolor='w')
plt.title('Distribution of Summary Subjectivity')
plt.xlabel('Subjectivity (0 = objective, 1 = subjective)')
plt.ylabel('Number of summaries')
plt.show()
```

6. Identify top 20 positive and negative reviews and summaries respectively

```
In [85]: # Top 20 negative reviews.
top20_negative_reviews = df3.nsmallest(20, 'review_polarity')[['review', 'review

# View output.
top20_negative_reviews
```

Out[85]:

	review	review_polarity
208	BOOO UNLES YOU ARE PATIENT KNOW HOW TO MEASUR...	-1.000000
182	Incomplete kit! Very disappointing!	-0.975000
989	If you, like me, used to play D&D, but now you...	-0.625000
1804	I'm sorry. I just find this product to be bori...	-0.583333
364	One of my staff will be using this game soon, ...	-0.550000
117	I bought this as a Christmas gift for my grand...	-0.500000
227	this was a gift for my daughter. I found it d...	-0.500000
230	I found the directions difficult	-0.500000
290	Instructions are complicated to follow	-0.500000
301	Difficult	-0.500000
803	This game is a blast!	-0.500000
1524	Expensive for what you get.	-0.500000
1829	Scrabble in a card game!	-0.500000
174	I sent this product to my granddaughter. The p...	-0.491667
347	My 8 year-old granddaughter and I were very fr...	-0.452500
538	I purchased this on the recommendation of two ...	-0.440741
306	Very hard complicated to make these.	-0.439583
427	Kids I work with like this game.	-0.400000
437	This game although it appears to be like Uno a...	-0.400000
497	My son loves playing this game. It was recomme...	-0.400000

```
In [86]: # Top 20 negative summaries.
top20_negative_summaries = df3.nsmallest(20, 'summary_polarity')[['summary', 'su

# View output.
top20_negative_summaries
```

Out[86]:

	summary	summary_polarity
21	The worst value I've ever seen	-1.000
208	BORING UNLESS YOU ARE A CRAFT PERSON WHICH I A...	-1.000
829	Boring	-1.000
1166	before this I hated running any RPG campaign d...	-0.900
1	Another worthless Dungeon Master's screen from...	-0.800
144	Disappointed	-0.750
631	Disappointed.	-0.750
793	Disappointed	-0.750
1620	Disappointed	-0.750
363	Promotes anger instead of teaching calming met...	-0.700
885	Too bad, this is not what I was expecting.	-0.700
890	Bad Quality-All made of paper	-0.700
178	At age 31 I found these very difficult to make...	-0.650
101	Small and boring	-0.625
504	It's UNO for the angry!	-0.625
518	Mad dragon	-0.625
1790	Ball of weird!	-0.625
805	Disappointing	-0.600
1015	Disappointing.	-0.600
1115	Disappointing	-0.600

```
In [87]: # Top 20 positive reviews.
top20_positive_reviews = df3.nlargest(20, 'review_polarity')[['review', 'review_

# View output.
top20_positive_reviews
```

Out[87]:

	review	review_polarity
7	Came in perfect condition.	1.0
44	Absolutely great pictures even before coloring!	1.0
55	Great!	1.0
165	Awesome book	1.0
194	Awesome gift	1.0
216	Great product! Arrived on time.	1.0
318	Great buy!! My granddaughter loves it!!	1.0
418	Great resource for BHIS care coordinators!! Wo...	1.0
474	Great Seller!!! Happy with my purchase!!! 5 st...	1.0
496	Excellent activity for teaching self-managemen...	1.0
503	Great game...I use it a lot!	1.0
517	Great therapy tool!	1.0
524	Perfect, just what I ordered!!	1.0
591	Wonderful product	1.0
609	Delightful product!	1.0
620	Great Easter gift for kids!	1.0
621	Wonderful for my grandson to learn the resurre...	1.0
685	These are great!	1.0
790	Perfect!	1.0
804	great!!	1.0

```
In [88]: # Top 20 positive summaries.
top20_positive_summaries = df3.nlargest(20, 'summary_polarity')[['summary', 'sum

# View output.
top20_positive_summaries
```

Out[88]:

	summary	summary_polarity
6	Best gm screen ever	1.0
28	Wonderful designs.	1.0
32	Perfect!	1.0
37	Great buy! Can't wait to work on this book	1.0
40	So beautiful!	1.0
57	great!	1.0
80	They're the perfect size to keep in the car or...	1.0
122	Great for a gift!	1.0
134	Perfect for Preschooler	1.0
140	Awesome sticker activity for the price	1.0
161	Awesome Book...	1.0
163	He was very happy with his gift	1.0
187	Awesome	1.0
199	Great product! Darling puppies!	1.0
202	Great!	1.0
210	Awesome and well-designed for 9 year olds	1.0
335	Another great book by Klutz!	1.0
418	Perfect!	1.0
449	Great resource!	1.0
457	This is a great product! I use it as a therape...	1.0

7. Discuss: Insights and observations

Your observations here...

Word frequency

- Reviews were dominated by product/enjoyment words: game, great, fun, play, love, really
- Summaries were dominated by stars and five — most summaries are just star-rating shorthand (e.g. "Five Stars"), not descriptive text, making them a weaker source of qualitative insight than full reviews

Polarity and subjectivity

- Polarity scores spanned the full -1 to +1 range, with a cluster at exactly 0.0 — mostly short, generic summaries TextBlob can't extract strong sentiment from

- Subjectivity varied independently of polarity — e.g. "Amazing buy!" scored high on both (an enthusiastic opinion), while short factual summaries like "Money trap" scored low on both

Top 20 positive reviews/summaries

- Dominated by short, exclamatory phrases: "Great!", "Perfect!", "Awesome book"
- Nearly all hit the maximum score of 1.0, creating a ceiling effect — beyond "these are all very positive," the exact ranking is somewhat arbitrary

Top 20 negative reviews/summaries

- Negative summaries were largely accurate: "The worst value I've ever seen," "Boring," repeated instances of "Disappointed"
- Several negative-scored reviews were actually misclassified — genuinely positive text scored as negative: "This game is a blast!" (-0.50), "My son loves playing this game" (-0.40), "Kids I work with like this game" (-0.40) This reflects a known weakness of lexicon-based tools like TextBlob: they can miss slang/idioms ("blast" isn't recognised as positive) and get thrown off by a single negative-sounding word in an otherwise positive sentence

Implication for the marketing team

- Automated sentiment scoring is fast and useful for directional, large-scale trends
- It shouldn't be treated as fully reliable at the individual-review level — a human spot-check is recommended before pulling specific quotes for campaign material