mini-Forage's KPMG for Sprocket Central Pty Ltd-Copy1

January 16, 2023

1 Customer Segmentation of Sprocket Central Pty Ltd

1.1 Objective:

Sprocket Central Pty Ltd is an high-end bicycle and accessories company that is seeking to develop a new marketing strategy that targets their best customers. This project will involve 2 seperate datasets; the first with transactional data, and the later without transactional data (new customer list). Our analysis will involve the RFM and K-Means modelling to achieve the identification of who our client's best customers are. Let's begin by exploring their data, developing the model and concluding with dashboard of data visuals to support our discoveries.

2 I. MERGED DATASET ANALYSIS

2.1 Variable Description of Merged Dataset, with Transactions:

Our summary of data consists of the following attributes:

CustomerDemographic data: - customer_id: unique customer id number - first_name: customers' first name - last_name: customers' last name - gender: gender orientation - amended to 'Male', 'Female' or 'Other' - past_3_years_bike_related_purchases: count of purchases made in 2017 - DOB: customers' date of birth - job_title: title of job position - job_industry_category: industry category of customers' jobs - wealth_segment: customer's wealth status - 'Affluent', 'High Net Worth' and 'Mass Customer' - deceased_indicator: indicates whether the customer has passed away - 'N' or 'Y' - owns_car: indicates whether customer owns a car - 'Y' or 'N' - tenure: the duration of years customer occupies their residence

CustomerAddress data: - customer_id: unique customer id number - postcode: customers' postal code - state: customers' states of 'NSW', 'VIC', 'QLD' - property_valuation: property valuation grade - numbers 1 to 12

Transactions data: - product_id: unique id of product - customer_id: unique customer id number - transaction_date: date of transaction - online_order: indicates whether order was purchased online - '0' Online or '1' Not Online - order_status: indicates whether the order was a approved or cancelled - brand: brand name of the product - product_line: product catergory - 'Mountain', 'Road', 'Standard' and 'Touring' - product_class: class level of the product = 'high', 'low', 'medium' - product_size: size of the product - 'large', 'medium' and 'small' - list_price: listed price of the product - standard_cost: standard cost of transaction - product_first_sold_date: date of product's first sale

2.2 1. Set Up

```
[1]: # import modules and uploading the worksheets into dataframes

     import pandas as pd
     import numpy as np
     import seaborn as sns
     import matplotlib.pyplot as plt
     import datetime as dt
     import warnings
     warnings.filterwarnings('ignore')

     # reading worksheets into different names for merging
     # select row 1 to use as the header of columns

     s1 = pd.read_excel('CustomerDemographic.xlsx', header=1)
     s2 = pd.read_excel('CustomerAddress.xlsx', header=1)
     s3 = pd.read_excel('Transactions.xlsx', header=1)
```

2.3 2. Merging Worksheets

Immediately excluding irrelevant columns from the original worksheets, such as 'default' from CustomerDemographic (due to nonsensical text), also 'address' and 'country' from CustomerAddress, as all customers are based in Australia.

```
[2]: # begin merge with s1 (CustomerDemogrpahic) and s2 (CustomerAddress) into s4
     # only required columns from original data are included into dataframe s4 for␣
      ↪analysis
     # column 'customer_id' is the common column between all 3 worksheets to join␣
      ↪data

     s4 = s1[['customer_id', 'gender', 'past_3_years_bike_related_purchases', 'DOB',␣
      ↪'job_title', 'job_industry_category',
             'wealth_segment','deceased_indicator', 'owns_car', 'tenure']].
      ↪merge(s2[['customer_id', 'postcode', 'state',
             'property_valuation']], on = 'customer_id', how='left')
```

```
[3]: # renaming long columns accurate names

     s4 = s4.rename(columns={'past_3_years_bike_related_purchases':␣
      ↪'frequency_2017', 'job_industry_category': 'industry'})
```

```
[4]: # perform last merge to include s3 (Transactions worksheet)
     # selecting only the required columns to include for analysis;
     # 'transaction_id' and 'product_first_sold_date' will not be included into␣
      ↪dataframe
```

```
s5 = s4.merge(s3[['product_id', 'customer_id', 'transaction_date',␣
↪'online_order', 'order_status', 'brand',
                'product_line', 'product_class', 'product_size',␣
↪'list_price', 'standard_cost']],
            on='customer_id', how='left')
```

2.4 3. Data Wrangling

```
[5]: # inspect the head and tail of data

     s5
```

```
[5]:        customer_id  gender  frequency_2017        DOB              job_title  \
     0                1       F              93  1953-10-12     Executive Secretary
     1                1       F              93  1953-10-12     Executive Secretary
     2                1       F              93  1953-10-12     Executive Secretary
     3                1       F              93  1953-10-12     Executive Secretary
     4                1       F              93  1953-10-12     Executive Secretary
     ...            ...     ...             ...         ...                     ...
     20499         3996  Female               8  1975-08-09  VP Product Management
     20500         3997  Female              87  2001-07-13         Statistician II
     20501         3998       U              60         NaT       Assistant Manager
     20502         3999    Male              11  1973-10-24                     NaN
     20503         4000    Male              76  1991-11-05     Software Engineer IV

                  industry     wealth_segment deceased_indicator owns_car  tenure  \
     0              Health      Mass Customer                  N      Yes    11.0
     1              Health      Mass Customer                  N      Yes    11.0
     2              Health      Mass Customer                  N      Yes    11.0
     3              Health      Mass Customer                  N      Yes    11.0
     4              Health      Mass Customer                  N      Yes    11.0
     ...               ...                ...                ...      ...     ...
     20499          Health      Mass Customer                  N       No    19.0
     20500   Manufacturing     High Net Worth                  N      Yes     1.0
     20501              IT     High Net Worth                  N       No     NaN
     20502   Manufacturing  Affluent Customer                  N      Yes    10.0
     20503             NaN  Affluent Customer                  N       No    11.0

            …  product_id transaction_date  online_order  order_status  \
     0      …        86.0       2017-12-23           0.0      Approved
     1      …        38.0       2017-04-06           1.0      Approved
     2      …        47.0       2017-05-11           1.0      Approved
     3      …        72.0       2017-01-05           0.0      Approved
     4      …         2.0       2017-02-21           0.0      Approved
     ...   ...         ...              ...           ...           ...
     20499  …         NaN              NaT           NaN           NaN
     20500  …         NaN              NaT           NaN           NaN
```

```
20501  …           NaN                NaT        NaN          NaN
20502  …           NaN                NaT        NaN          NaN
20503  …           NaN                NaT        NaN          NaN

                brand  product_line product_class product_size list_price  \
0          OHM Cycles     Standard        medium        medium     235.63
1               Solex     Standard        medium        medium    1577.53
2       Trek Bicycles         Road           low         small    1720.70
3      Norco Bicycles     Standard        medium        medium     360.40
4               Solex     Standard        medium        medium      71.49
...               ...          ...           ...           ...        ...
20499             NaN          NaN           NaN           NaN        NaN
20500             NaN          NaN           NaN           NaN        NaN
20501             NaN          NaN           NaN           NaN        NaN
20502             NaN          NaN           NaN           NaN        NaN
20503             NaN          NaN           NaN           NaN        NaN

       standard_cost
0             125.07
1             826.51
2            1531.42
3             270.30
4              53.62
...              ...
20499            NaN
20500            NaN
20501            NaN
20502            NaN
20503            NaN

[20504 rows x 23 columns]
```

[6]: ```python
view the summary of data

s5.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20504 entries, 0 to 20503
Data columns (total 23 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   customer_id      20504 non-null  int64
 1   gender           20504 non-null  object
 2   frequency_2017   20504 non-null  int64
 3   DOB              20047 non-null  datetime64[ns]
 4   job_title        18027 non-null  object
 5   industry         17180 non-null  object
 6   wealth_segment   20504 non-null  object
```

```
7    deceased_indicator  20504 non-null  object
8    owns_car            20504 non-null  object
9    tenure              20047 non-null  float64
10   postcode            20475 non-null  float64
11   state               20475 non-null  object
12   property_valuation  20475 non-null  float64
13   product_id          19997 non-null  float64
14   transaction_date    19997 non-null  datetime64[ns]
15   online_order        19637 non-null  float64
16   order_status        19997 non-null  object
17   brand               19800 non-null  object
18   product_line        19800 non-null  object
19   product_class       19800 non-null  object
20   product_size        19800 non-null  object
21   list_price          19997 non-null  float64
22   standard_cost       19800 non-null  float64
dtypes: datetime64[ns](2), float64(7), int64(2), object(12)
memory usage: 3.8+ MB
```

- There are 20504 rows and 22 columns.
- Missing values in the following columns; DOB, job_title, industry, tenure, postcode, state, property_valuation, product_id, tansaction_date, online_order, order_status, brand, product_line, product_class, list_price and statndard_cost.
- Datatypes appear in correct format.

[7]:
```
# check for any duplicate rows

s5.duplicated().any()
```

[7]: False

[8]:
```
# further drop irrelevant columns

s5.drop(['deceased_indicator'], axis=1, inplace=True)
```

2.4.1 Format Column Datatypes and Names

[9]:
```
# correcting the 'gender' error values

s5.gender = s5.gender.replace(['U', 'F', 'Femal', 'M'], ['Other', 'Female',␣
 ↪'Female', 'Male'])
```

[10]:
```
# editing the year error of a DOB value

s5.DOB = s5.DOB.replace('1843-12-21', '1943-12-21')
```

[11]:
```
# renaming 'state' values to align with existing abbreviated states
```

```
s5.state = s5.state.replace(['New South Wales', 'Victoria'], ['NSW', 'VIC'])
```

[12]:
```python
# remove $ sign from column for analysis

s5.standard_cost = s5.standard_cost.replace({r'\$':''}, regex=True)
```

2.4.2 Missing Data

[13]:
```python
# view % of missing values to determine the treatment method

round(s5.isnull().sum().sort_values(ascending = False)/len(s5)*100,2)
```

[13]:
```
industry             16.21
job_title            12.08
online_order          4.23
standard_cost         3.43
product_size          3.43
product_class         3.43
product_line          3.43
brand                 3.43
list_price            2.47
order_status          2.47
product_id            2.47
transaction_date      2.47
DOB                   2.23
tenure                2.23
property_valuation    0.14
state                 0.14
postcode              0.14
gender                0.00
owns_car              0.00
wealth_segment        0.00
frequency_2017        0.00
customer_id           0.00
dtype: float64
```

15 columns have missing data, with 'industry' having the most missing values.

If the missing values of a column doesn't exceed 5%, we can apply the imputation by mean, mode and median to fill missing values. In this instance, we assume the values are missing at random to correctly treat our missing values.

[14]:
```python
# 'brand', 'product_line', 'product_class', 'product_size', and 'standard_cost'
  ↪have corresponding missing rows

missing_values343 = s5[['brand', 'product_line', 'product_class',
  ↪'product_size', 'standard_cost']].isnull()
missing_values343.tail()
```

```
[14]:        brand  product_line  product_class  product_size  standard_cost
      20499  True          True           True           True          True
      20500  True          True           True           True          True
      20501  True          True           True           True          True
      20502  True          True           True           True          True
      20503  True          True           True           True          True
```

In the following ffill treatment of missing values, we assume that missing data are MCAR (Missing Completely At Random), occuring randomly without any pattern, hence based on the the correspsonding missing value rows of the above columns, we can use ffill method to get values that are accurately related to each other. This reduces bias as compared to mean, median or mode methods.

```python
[15]: # impute the missing values with ffill method

      s5.brand.fillna(method='ffill', inplace=True)
      s5.product_line.fillna(method='ffill', inplace=True)
      s5.product_class.fillna(method='ffill', inplace=True)
      s5.product_size.fillna(method='ffill', inplace=True)
      s5.standard_cost.fillna(method='ffill', inplace=True)
```

```python
[16]: # use ffill method to impute missing values from 'transaction_date' and␣
      ↪'job_title'

      s5.transaction_date.fillna(method='ffill', inplace=True)
      s5.job_title.fillna(method='ffill', inplace=True)
```

```python
[17]: # % of 'order_status' values

      s5.order_status.value_counts()/len(s5)*100
```

```
[17]: Approved     96.654311
      Cancelled     0.873000
      Name: order_status, dtype: float64
```

```python
[18]: # calculate 96% of all null values in 'order_status' to replace with 'Approved'

      round(0.96*(s5.order_status.isnull().sum()),0)
```

```
[18]: 487.0
```

```python
[19]: # replacing null values with 'Approved' with 487 rows limit, to follow 96% of␣
      ↪missing values

      s5.order_status.replace([np.nan], 'Approved', limit=487, inplace=True)
```

```python
[20]: # remove all 'Cancelled' values as we require only 'Approved' transactions for␣
      ↪analysis
```

```
s5 = s5.drop(s5[(s5['order_status'] == 'Cancelled')].index)
s5.reset_index(drop=True, inplace=True)
```

[21]:
```
# removing remaining null values from 'order_status' to maintain accuracy of␣
 ↪data

s5.dropna(subset=['order_status'], axis=0, inplace=True)

# reset the dataframe after dropping rows

s5.reset_index(drop=True, inplace=True)
```

[22]:
```
# replace selected columns' missing values with mean and mode

s5.fillna(value={'DOB':s5['DOB'].mode()[0], 'list_price':s5['list_price'].
 ↪mean(), 'product_id':s5['product_id'].mode()[0],
        'online_order':s5['online_order'].mode()[0], 'property_valuation':
 ↪s5['property_valuation'].mode()[0], 'tenure':s5['tenure'].median()},␣
 ↪inplace=True)
```

[23]:
```
# convert 'DOB' as 'age' column

from datetime import date

def calculate_age(born):
    today = date.today()
    return today.year - born.year - ((today.month, today.day) < (born.month,␣
 ↪born.day))
```

[24]:
```
s5.DOB = s5.DOB.apply(calculate_age).astype('int')
s5.DOB
```

[24]:
```
0          69
1          69
2          69
3          69
4          69
          ..
20320      47
20321      21
20322      44
20323      49
20324      31
Name: DOB, Length: 20325, dtype: int32
```

[25]:
```
# rename the 'DOB' as 'age' column
```

```
s5.rename(columns={'DOB': 'age'}, inplace=True)
```

[26]:
```
# identify the most common 'postcode'

s5.postcode.value_counts().idxmax()
```

[26]: 2153.0

[27]:
```
# replacing missing 'postcode' values with most frequent column value '2153'

s5.postcode.fillna('2153', inplace=True)
```

[28]:
```
# assigning 'NSW' value for 'state' missing values to corresspond with postcode␣
 ↪2153

s5.state.fillna('NSW', inplace=True)
```

[29]:
```
#  replacing 'indsutry' null values to 'n/a'

s5.industry.replace([np.nan], 'n/a', inplace=True)
```

By using the Arbitrary Imputation we filled the {nan} values in this column with {n/a} thus, making an additional value for the variable 'industry'.

[30]:
```
# final check that missing values have been addressed

s5.isnull().sum()
```

[30]:
```
customer_id          0
gender               0
frequency_2017       0
age                  0
job_title            0
industry             0
wealth_segment       0
owns_car             0
tenure               0
postcode             0
state                0
property_valuation   0
product_id           0
transaction_date     0
online_order         0
order_status         0
brand                0
product_line         0
product_class        0
product_size         0
```

```
list_price          0
standard_cost       0
dtype: int64
```

2.5   4. Exploratory Data Analysis

```python
[31]:   # further drop irrelevant columns

        s5 = s5.drop(columns=['postcode', 'job_title', 'order_status', 'owns_car',␣
          ↪'product_line', 'product_class', 'product_id',
                              'product_size', 'brand'])
```

```python
[32]:   # create 'total_price' column to measure customers' total spending

        total_price = s5['standard_cost'] + s5['list_price']
        s5 = pd.concat([s5, total_price], axis=1)
```

```python
[33]:   # new data shape

        s5.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20325 entries, 0 to 20324
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   customer_id        20325 non-null  int64
 1   gender             20325 non-null  object
 2   frequency_2017     20325 non-null  int64
 3   age                20325 non-null  int32
 4   industry           20325 non-null  object
 5   wealth_segment     20325 non-null  object
 6   tenure             20325 non-null  float64
 7   state              20325 non-null  object
 8   property_valuation 20325 non-null  float64
 9   transaction_date   20325 non-null  datetime64[ns]
 10  online_order       20325 non-null  float64
 11  list_price         20325 non-null  float64
 12  standard_cost      20325 non-null  float64
 13  0                  20325 non-null  float64
dtypes: datetime64[ns](1), float64(6), int32(1), int64(2), object(4)
memory usage: 2.1+ MB
```

```python
[34]:   # amending the label error of 'total_price' appearing as '0'

        s5.set_axis(['customer_id', 'gender', 'frequency_2017', 'age', 'industry',␣
          ↪'wealth_segment', 'tenure',
```

```
          'state', 'property_valuation', 'transaction_date', 'online_order',␣
↪'list_price', 'standard_cost', 'total_price'], axis=1, inplace=True)
```

- Demographic Segmentation: 'gender', 'age', 'job_title', 'industry', 'wealth_segment'
- Geographic Segmentation: 'postcode', 'state'
- Behavioral Segmentation: 'purchases_3yrs', 'transaction_date', 'online_order', 'list_price'

2.6 5. Uniqueness Summary

[35]: ```
s5.nunique()
```

[35]: ```
customer_id   3999
gender   3
frequency_2017   100
age   55
industry   10
wealth_segment   3
tenure   22
state   3
property_valuation   12
transaction_date   364
online_order   2
list_price   296
standard_cost   100
total_price   303
dtype: int64
```

- This dataframe contains 3999 in total of different customers
- 3 types of gender orientations
- There are 100 measureable frequency values
- 55 different ages
- 10 industries
- 'wealth_segment' is divided into 3 groups
- 'tenure' is measured by 22 lengths
- There are 3 states included
- 12 types of property valuation
- 'transaction_dates' cover 364 days
- 'online_order' will indicate either '0' yes or '1' no
- There are 296 different list prices
- 100 standard cost values
- 303 different total price values

[36]: ```
# saving the new dataframe into an excel file

#s5.to_excel('SprocketCleaned.xlsx')
```
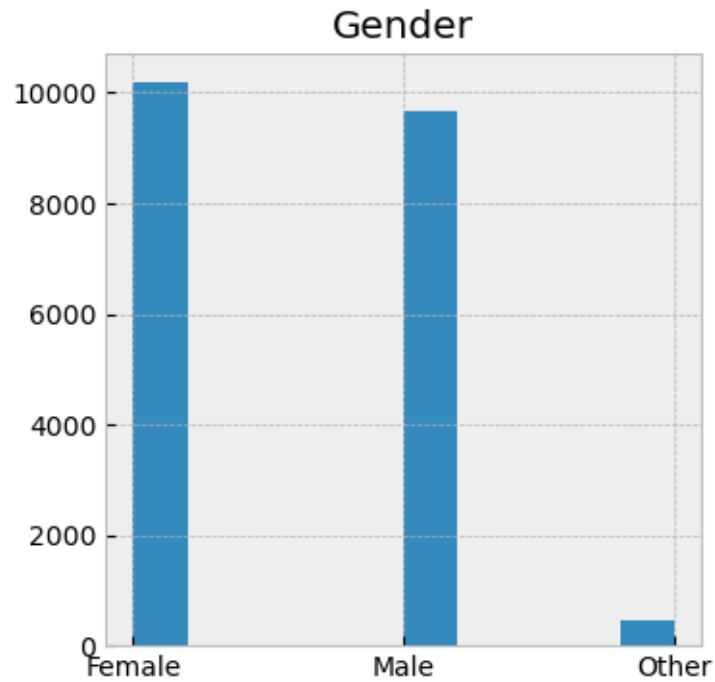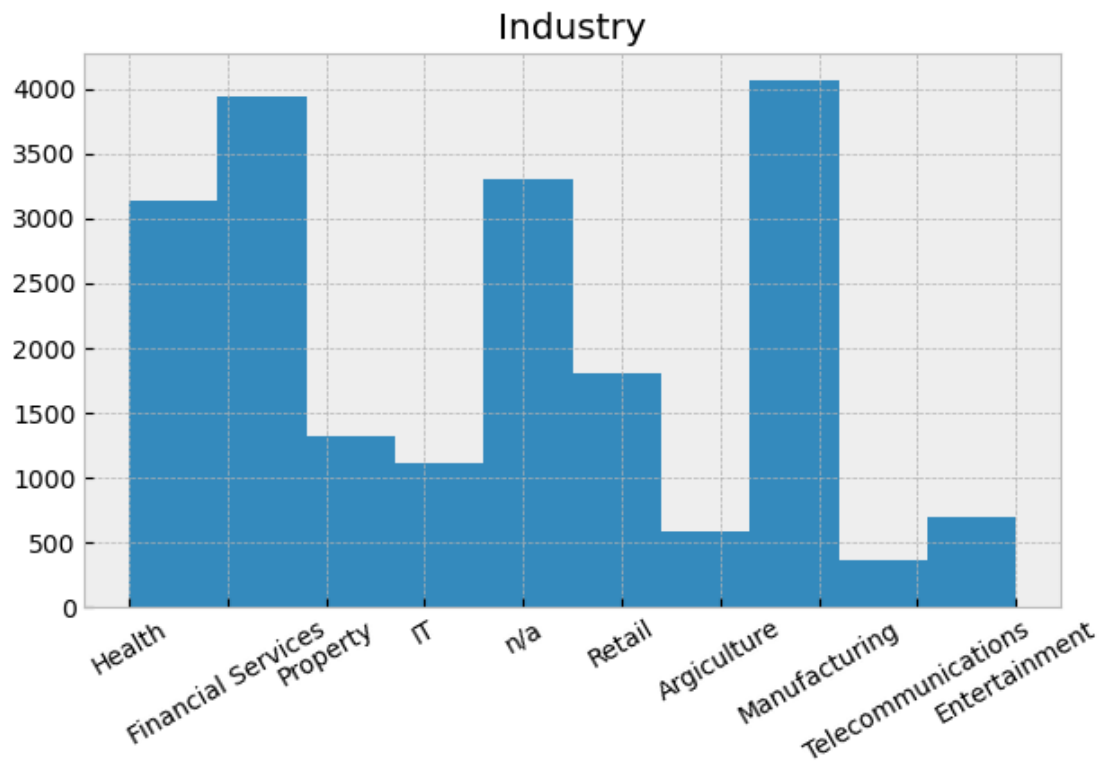
[37]: ```
df = s5.copy()
```

### 2.6.1 Univariate Analysis

Explore each attribute count to understand our data better

[38]:
```
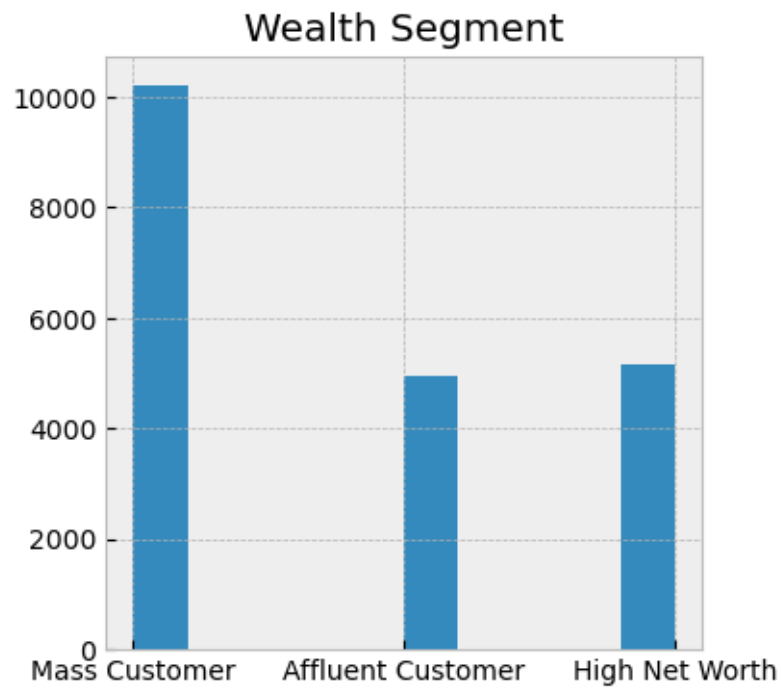plt.style.use('bmh')
plt.figure(figsize=(4, 4))
plt.hist(df['gender'])
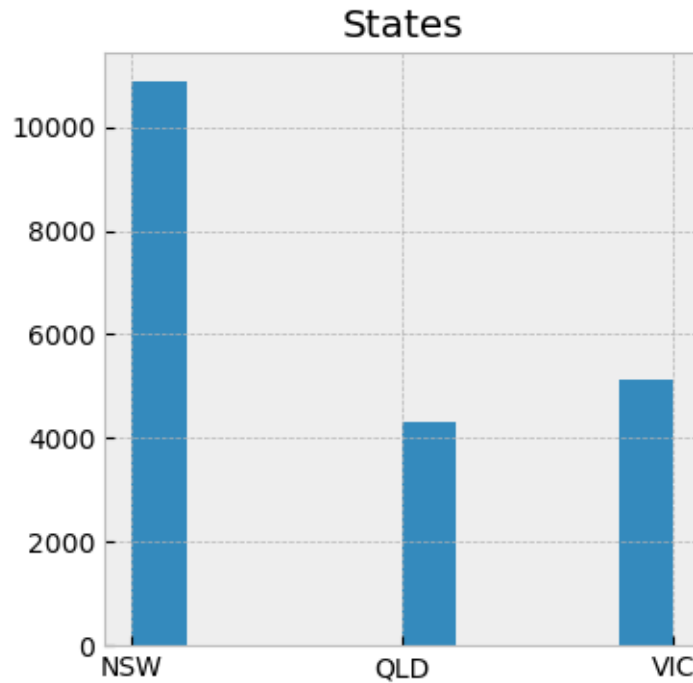plt.title('Gender')
plt.show()
```



[39]:
```
plt.style.use('bmh')
plt.figure(figsize=(7, 4))
plt.hist(df['industry'])
plt.xticks(rotation=30)
plt.title('Industry')
plt.show()
```

# Industry



```
[40]: plt.style.use('bmh')
      plt.figure(figsize=(4, 4))
      plt.hist(df['wealth_segment'])
      plt.title('Wealth Segment')
      plt.show()
```

Wealth Segment


```
[41]: plt.style.use('bmh')
      plt.figure(figsize=(4, 4))
      plt.hist(df['state'])
      plt.title('States')
      plt.show()
```

Observations: - There are more female customers than Male and Other - Manufacturing and Financial Services are the main industries customers are from - Mass Customers more than double the High Net Worth or Affluent Customers - The bulk of customers are based in NSW

2.6.2 Bivariate Analysis

Expolore relationships between 'age', 'frequency_2017' and 'total_price'

```
[127]: plt.figure(1 , figsize = (8 , 8))
       n = 0
       for x in ['age' , 'frequency_2017' , 'total_price']:
           for y in ['age' , 'frequency_2017' , 'total_price']:
               n += 1
               plt.subplot(3 , 3 , n)
               plt.subplots_adjust(hspace = 0.5 , wspace = 0.5)
               sns.regplot(x = x , y = y , data = df)
               plt.ylabel(y.split()[0]+' '+y.split()[1] if len(y.split()) > 1 else y )
       plt.show()
```
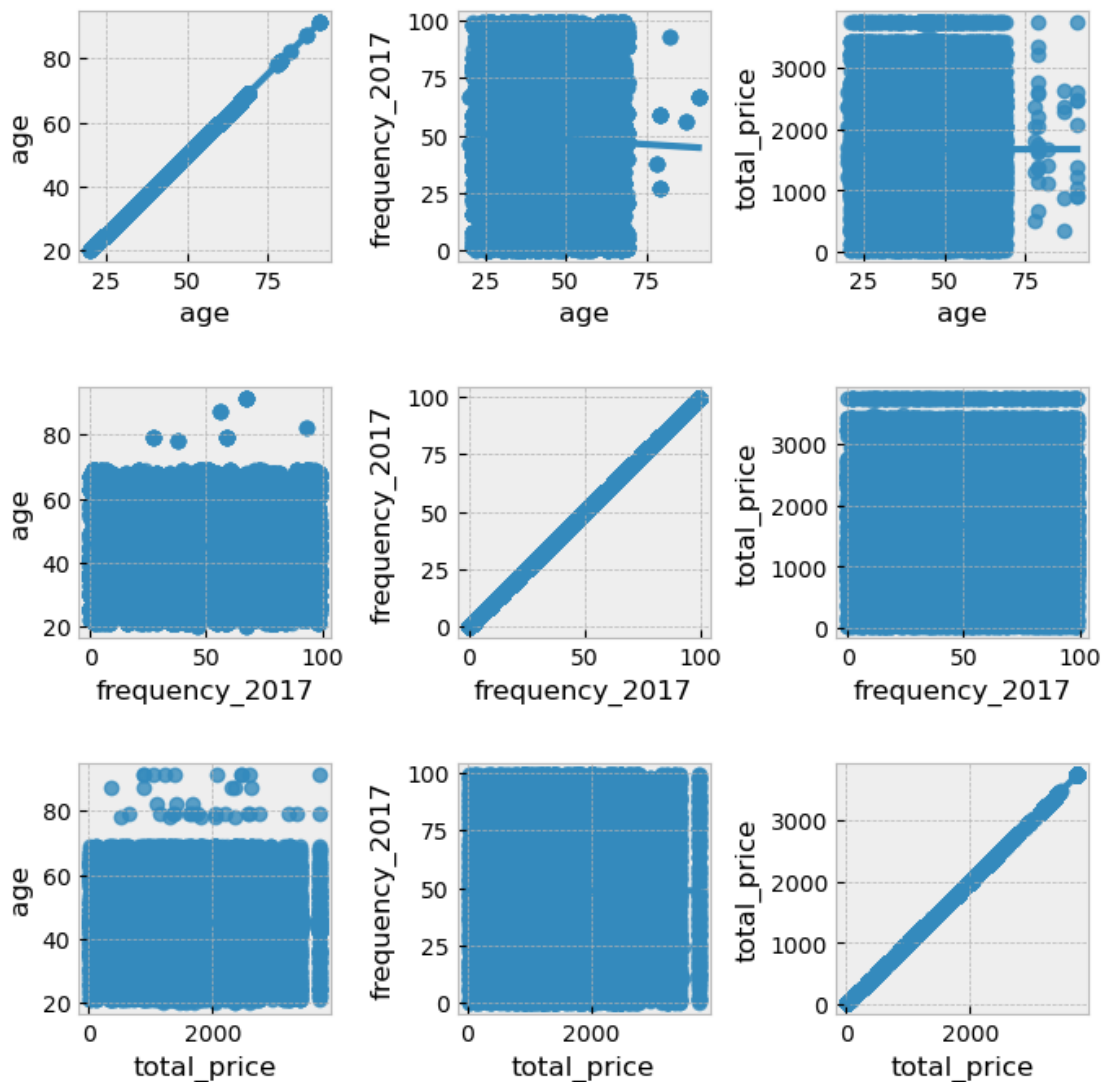
Observations: There are no evident relationships present in these features.

```
[43]: df1 = df.copy()
```

```
[44]: df1.drop(['customer_id'], axis=1, inplace=True)
```

2.7 6. Statistics of Data

```
[45]: df1.describe()
```

```
[45]:        frequency_2017            age         tenure  property_valuation  \
       count   20325.000000   20325.000000   20325.000000        20325.000000
       mean       48.816482      44.913752      10.676212            7.515031
       std        28.613997      12.503304       5.610084            2.824812
```

```
min        0.000000    20.000000     1.000000              1.000000
25%       24.000000    36.000000     6.000000              6.000000
50%       48.000000    45.000000    11.000000              8.000000
75%       73.000000    54.000000    15.000000             10.000000
max       99.000000    91.000000    22.000000             12.000000


       online_order    list_price  standard_cost    total_price
count  20325.000000  20325.000000   20325.000000   20325.000000
mean       0.478819   1107.714036     563.344223    1671.058260
std        0.499563    575.697431     404.233514     864.284644
min        0.000000     12.010000       7.210000      19.220000
25%        0.000000    586.450000     230.090000    1006.720000
50%        0.000000   1151.960000     513.850000    1681.610000
75%        1.000000   1577.530000     820.780000    2368.640000
max        1.000000   2091.470000    1759.850000    3737.210000
```

Observations: On avearge, our customers are of 44 years of age, frequent our store 49 times in 2017, and spent around $1671.

2.7.1 Correlations of All Attributes

2.7.2 Feature Scaling

Let's scale our ordinal values using LabelEncoder which encodes ordinal data with values between 0 and n_classes-1, where n is the number of distinct labels. For nominal data, we apply the pandas .get_dummies() function to convert object datatypes into numerical data for statistical analysis.

```python
[46]: from sklearn.preprocessing import LabelEncoder #preProcessing data
      import os
```

```python
[47]: # feature scale attributes using LabelEncoder and .get_dummies()

      df1['wealth_segment']=LabelEncoder().fit_transform(df1["wealth_segment"])

      dummies = pd.get_dummies(df1, columns=['gender','industry', 'state'])
      df1 = pd.concat([df1, dummies], axis=1)
```

```python
[48]: plt.figure(figsize=(20,22))
      sns.heatmap(df1.corr(), annot=True)
      plt.show()
```
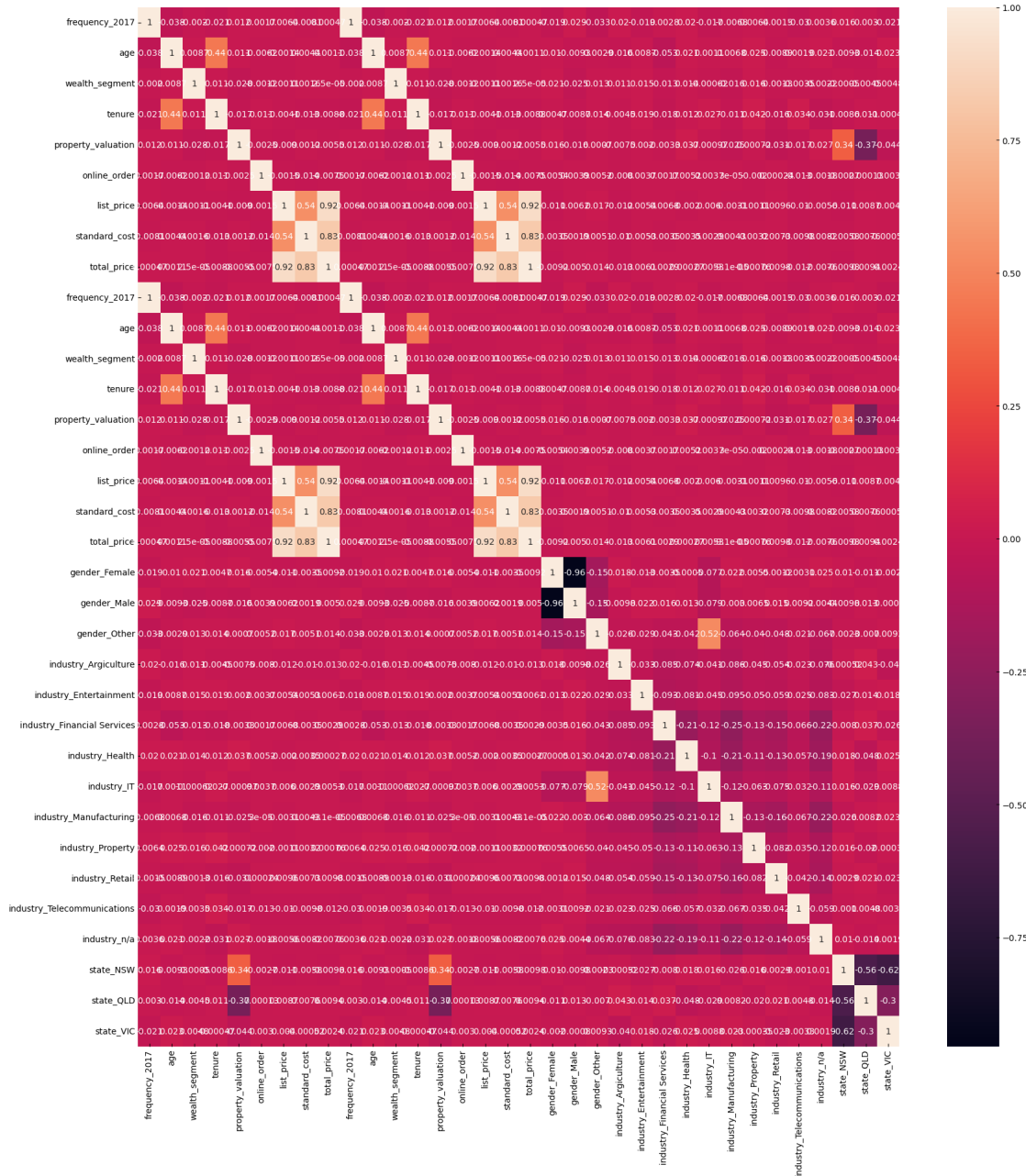
Observation: - state_NSW has a correlation of 0.34 to 'property_valuation', which implies NSW has a higher property valuation - state_NSW is also strongly correlated to state_VIC, with a correlation of -0.62 - 'gender_Other' has a correaltion of 0.52 to 'industry_IT' which indicates that the IT industry has many gender_Other

2.8 7. Recency Frequency Monetary (RFM) Analysis

Here, we will apply the RFM model to identify our best customers, based on their Recency and Frequency of patroning the Sprocket Central shops, and according to their spending value (Monetary).

RFM analysis allows us to segment our customers according to their spending, demographic and geographical behaviour, by sampling convinience of our customer database. Businesses can benefit from customer segmentation which fine tunes target marketing campaigns,improves resource allocation, and improves sales.

Customer segmentation models are usually built using unsupervised machine learning algorithms such as K-Means clustering or hierarchical grouping. These models can pick up on similarities between user groups that often go unnoticed by the human eye, by reducing the distortion of our dataset.

Our best customers will have the following attributes: - Lowest Recency - Highest Frequency - Highest Monetary Value

2.8.1 i. Calculate dates for analysis

```
[49]: # first order date

print("The first date of transaction is:", df['transaction_date'].min())
```

The first date of transaction is: 2017-01-01 00:00:00

```
[50]: # last order date

print("The last date of transation is:", df['transaction_date'].max())
```

The last date of transation is: 2017-12-30 00:00:00

```
[51]: # recency is calculated as a point in time, so based on the last␣
      ↪transaction_date we'll use 2017-12-31 to calculate recency

Now = dt.datetime(2017,12,31)
df['transaction_date'] = pd.to_datetime(df['transaction_date'])
```

2.8.2 ii. Create RFM Table

```
[52]: # create RFM Table

rfmTable = df.groupby('customer_id').agg({'transaction_date': lambda x: (Now -␣
      ↪x.max()).days,
                                          'frequency_2017': 'first',␣
      ↪'total_price': lambda x: round(x.sum(),2)})

rfmTable['transaction_date'] = rfmTable['transaction_date'].astype(int)

rfmTable.rename(columns={'transaction_date': 'recency',
                         'frequency_2017': 'frequency',
                         'total_price': 'monetary_value'}, inplace=True)
```

```
[53]: # view our RFM table

      rfmTable
```

```
[53]:              recency  frequency  monetary_value
      customer_id
      1                  8         93        15150.81
      2                129         81         6071.88
      3                103         61        16413.65
      4                196         33         1874.87
      5                 17         56         9411.46
      ...              ...        ...             ...
      3996             292          8         1982.61
      3997             292         87         1982.61
      3998             292         60         1982.61
      3999             292         11         1982.61
      4000             292         76         1982.61

      [3999 rows x 3 columns]
```

Below, we can check for accuracy of the RFM table. Let's inspect the first customer: - Her last purchase was 8 days from 31 Dec 2017, meaning her last 'transaction_date' was on 23 Dec - She has shopped 93 times in 2017 - She has spent a total of $15,150.81

```
[54]: # check accuracy of rfmTable against our first customer

      cust_1 = df[df['customer_id'] == 1]
      cust_1.head()
```

```
[54]:    customer_id  gender  frequency_2017  age industry wealth_segment  tenure  \
      0            1  Female              93   69   Health  Mass Customer    11.0
      1            1  Female              93   69   Health  Mass Customer    11.0
      2            1  Female              93   69   Health  Mass Customer    11.0
      3            1  Female              93   69   Health  Mass Customer    11.0
      4            1  Female              93   69   Health  Mass Customer    11.0

        state  property_valuation transaction_date  online_order  list_price  \
      0   NSW                10.0       2017-12-23           0.0      235.63
      1   NSW                10.0       2017-04-06           1.0     1577.53
      2   NSW                10.0       2017-05-11           1.0     1720.70
      3   NSW                10.0       2017-01-05           0.0      360.40
      4   NSW                10.0       2017-02-21           0.0       71.49

        standard_cost  total_price
      0        125.07       360.70
      1        826.51      2404.04
      2       1531.42      3252.12
      3        270.30       630.70
```

4	53.62	125.11

2.8.3 iii. Visualise RFM Distribution

```
[126]: plt.style.use('bmh')
       plt.figure(figsize=(10,3))
       plt.subplot(1,3,1); sns.distplot(rfmTable['recency'])
       plt.subplot(1,3,2); sns.distplot(rfmTable['frequency'])
       plt.subplot(1,3,3); sns.distplot(rfmTable['monetary_value'])
       plt.subplots_adjust(hspace =0.5 , wspace = 0.5)
       plt.show()
```
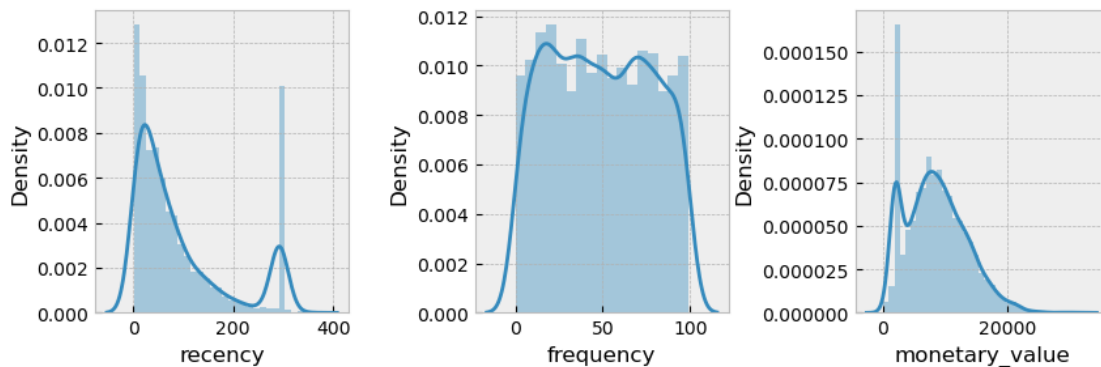

Check Skewness of RFM

```
[56]: rfmTable[['recency', 'frequency', 'monetary_value']].skew()
```

```
[56]: recency           1.230968
      frequency         0.045081
      monetary_value    0.513079
      dtype: float64
```

Observations: - 'recency' presents skewness of 1.23 which reflects the distribution of having more 'Active Customers' - 'frequency' is fairly evenly distributed across the range around the 9 mark

2.8.4 iv. Identify Outliers

```
[57]: f, axes = plt.subplots(1, 3)

      sns.boxplot(  y= "recency", data=rfmTable,  orient='v' , ax=axes[0])
      sns.boxplot(  y= "frequency", data=rfmTable,  orient='v' , ax=axes[1])
      sns.boxplot(  y= "monetary_value", data=rfmTable,  orient='v' , ax=axes[2])
      plt.subplots_adjust(hspace =0.5 , wspace = 0.8)
      plt.show()
```
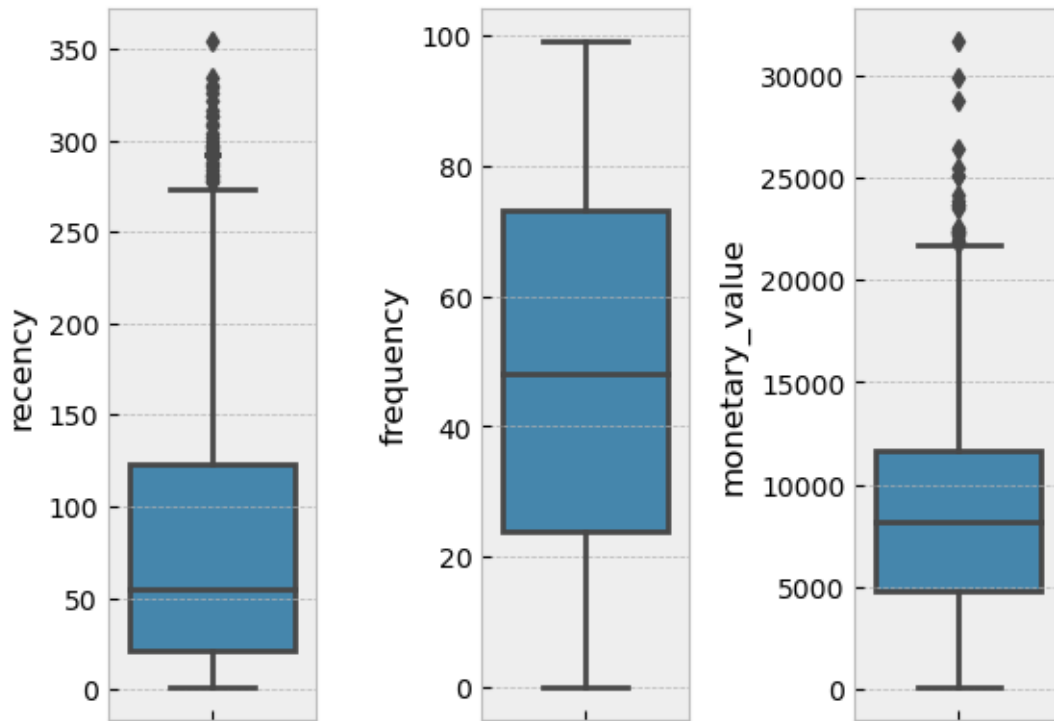
Observations: - Outliers exists for 'recency' and 'monetary_value' - As some bicycles are known to be valued in high price ranges, we can assume that 'Big Spenders' customers are serious biking hobbyists/connoisseurs who have the financial means for the luxury bike items - There outliers in 'recency' represent the 'Lost Customers'

2.8.5   v. Calculate RFM Quartiles

Next, the RFM values will be split into 4 percentile groups using quantiles to divide the RFM values into 4 groups 1. This will allow us a starting point for analysis 2. Label value ranges from 1-4, where 4 is the best quantile score.

```python
# define labels for RFM
r_labels = range(4, 0, -1); f_labels = range(1, 5); m_labels = range(1, 5)

# assign labels to 4 equal percentile groups
recency_group = pd.qcut(rfmTable['recency'].rank(method='first'), q=4,␣
 ↪labels=r_labels)

frequency_group = pd.qcut(rfmTable['frequency'].rank(method='first'), q=4,␣
 ↪labels=f_labels)

monetary_group = pd.qcut(rfmTable['monetary_value'].rank(method='first'), q=4,␣
 ↪labels=m_labels)
```

```
# Create new columns R and F
rfmTable = rfmTable.assign(R = recency_group.values, F = frequency_group.
   ↪values,  M = monetary_group.values)
rfmTable.head()
```

[58]:

customer_id	recency	frequency	monetary_value	R	F	M
1	8	93	15150.81	4	4	4
2	129	81	6071.88	1	4	2
3	103	61	16413.65	2	3	4
4	196	33	1874.87	1	2	1
5	17	56	9411.46	4	3	3

- The best customers segment will have a RFM score of 444
- 444 customers will have bought the most recently, shop more oftenly, and spent the most
- 444 customers can continue to thrive as a customer with loyalty programs and new products

2.8.6 vi. RFM Segments

[59]:
```
def join_rfm(x): return str(x['R']) + str(x['F']) + str(x['M'])
rfmTable['RFM_Segment'] = rfmTable.apply(join_rfm, axis=1)
rfmTable.head()
```

[59]:

customer_id	recency	frequency	monetary_value	R	F	M	RFM_Segment
1	8	93	15150.81	4	4	4	4.04.04.0
2	129	81	6071.88	1	4	2	1.04.02.0
3	103	61	16413.65	2	3	4	2.03.04.0
4	196	33	1874.87	1	2	1	1.02.01.0
5	17	56	9411.46	4	3	3	4.03.03.0

[60]:
```
# number of segment variations

print("Number of unique RFM_Segment variations:", rfmTable.RFM_Segment.
   ↪nunique())
```

```
Number of unique RFM_Segment variations: 64
```

These 64 unique RFM Segment variations will need to be classified into groups to give a general description of the customer type. A RFM Score of the sum of the RFM quartiles will provide the scale of our RFM Score.

RFM Score

[115]:
```
# Calculate RFM_Score

rfmTable['RFM_Score'] = rfmTable[['R','F','M']].sum(axis=1)
```

```
[62]: # remove the '.0' in the RFM_Segment values

      rfmTable.RFM_Segment = rfmTable['RFM_Segment'].str.replace(".0","",regex=False)
```

2.8.7 vii. Manually Grouping RFM Segments

```
[64]: def rfm_level(df):
          if df['RFM_Segment'] == '444':
              return 'Best Customers'

          elif df['RFM_Segment'] == '411':
              return 'New Customers'

          else:
              if df['M'] == 4:
                  return 'Big Spenders'

              elif df['F'] == 4:
                  return 'Loyal Customers'

              elif df['R'] == 4:
                  return 'Active Customers'

              elif df['R'] == 1:
                  return 'Lost Customers'

              elif df['M'] == 1:
                  return 'Frugal Spenders'

              return 'Regular Customers'

      # Create a new column RFM_Level
      rfmTable['RFM_Level'] = rfmTable.apply(rfm_level, axis=1)
```

```
[65]: rfmTable.head()
```

```
[65]:              recency  frequency  monetary_value  R  F  M RFM_Segment  \
      customer_id
      1                  8         93        15150.81  4  4  4         444
      2                129         81         6071.88  1  4  2         142
      3                103         61        16413.65  2  3  4         234
      4                196         33         1874.87  1  2  1         121
      5                 17         56         9411.46  4  3  3         433

                   RFM_Score         RFM_Level
      customer_id
      1                   12     Best Customers
```

```
2                    7   Loyal Customers
3                    9      Big Spenders
4                    4    Lost Customers
5                   10  Active Customers
```

Calculate Count and Percentage of RFM Segments

[66]: `rfmTable.RFM_Level.value_counts()`

```
[66]: Big Spenders        911
      Regular Customers   867
      Loyal Customers     752
      Lost Customers      701
      Active Customers    479
      Frugal Spenders     182
      Best Customers       89
      New Customers        18
      Name: RFM_Level, dtype: int64
```

```python
[67]: # calculate average and total values for each RFM_Level
      rfm_agg = rfmTable.groupby('RFM_Level').agg({
          'recency': 'mean',
          'frequency': 'mean',
          'monetary_value': ['mean', 'count']}).round(0)

      rfm_agg.columns = rfm_agg.columns.droplevel()
      rfm_agg.columns = ['recencyMean','frequencyMean','monetaryMean', 'count']
      rfm_agg['percent'] = round((rfm_agg['count']/rfm_agg['count'].sum())*100, 2)

      # reset the index
      rfm_agg = rfm_agg.reset_index()

      # Print the aggregated dataset
      rfm_agg
```

[67]:

	RFM_Level	recencyMean	frequencyMean	monetaryMean	count	percent
0	Active Customers	11.0	38.0	7977.0	479	11.98
1	Best Customers	9.0	85.0	15293.0	89	2.23
2	Big Spenders	47.0	45.0	14928.0	911	22.78
3	Frugal Spenders	69.0	35.0	3349.0	182	4.55
4	Lost Customers	238.0	36.0	3807.0	701	17.53
5	Loyal Customers	112.0	87.0	6250.0	752	18.80
6	New Customers	10.0	13.0	3674.0	18	0.45
7	Regular Customers	60.0	37.0	8233.0	867	21.68

2.8.8 viii. RFM Scatterplot

```
[68]: import plotly.express as px

      fig = px.scatter(rfm_agg, x="recencyMean", y="monetaryMean",␣
        ↪size="frequencyMean", color="RFM_Level",
               hover_name="RFM_Level", size_max=100)

      fig.show()
```

We are able to view the top 5 'Best Customers' with RFM Segment of 444

```
[69]: rfmTable[rfmTable['RFM_Segment']=='444'].sort_values('monetary_value',␣
        ↪ascending=False).head()
```

```
[69]:             recency  frequency  monetary_value  R  F  M RFM_Segment  \
      customer_id
      173              16         99        21573.45  4  4  4         444
      2464              3         78        21331.02  4  4  4         444
      2816              9         87        21246.99  4  4  4         444
      3420              6         96        20962.72  4  4  4         444
      2914             13         76        20813.10  4  4  4         444


                   RFM_Score       RFM_Level
      customer_id
      173                 12  Best Customers
      2464                12  Best Customers
      2816                12  Best Customers
      3420                12  Best Customers
      2914                12  Best Customers
```

Customer ID 173 is our top Best Customer, with recency 16 days, frequency of 99 days and spending total of $21,573.45 in 2017.

Concluding the RFM Model: When used alone, the RFM model may be too simplistic and may mislead. Notably, RFM models are not predictive and are easily skewed due to seasonal sales and subjective to product price. We can supplement the RFM model with another analytical model called K-Means Clusterring to get a fuller scope of our customers.

2.9 8. K-Means Clusterring Model

The unsupervised K-Means clustering algorithm segments unlabled data into non-overlapping sub-groups (k-clusters), that are distinct from each other. Each cluster has its own centroid and the main goal of this technique is to reduce the distortion between centroids, thus forming individual clusters according to their characteristics.

2.9.1 i. Elbow Method to predetermine K-clusters

The first step involves standardising the data (using StandardScaler) before predetermining the number of clusters the model will build, using the Elbow Method.

```
[70]:  kmeans_rfm = rfmTable[['recency', 'frequency', 'monetary_value']]
```

```
[71]:  from sklearn.preprocessing import StandardScaler
       std_scaler = StandardScaler()
       df_scaled = std_scaler.fit_transform(kmeans_rfm)
```

```
[128]:  from sklearn.cluster import KMeans
        SSE = []
        for k in range(1, 10):
            kmeans = KMeans(n_clusters=k, random_state=42)
            kmeans.fit(df_scaled)
            SSE.append(kmeans.inertia_) #SSE to nearest clustter centroid

        frame = pd.DataFrame({'Cluster':range(1,10), 'SSE':SSE})
        plt.figure(figsize=(8,5))
        plt.plot(frame['Cluster'], frame['SSE'], marker='o')
        plt.title('The Elbow Method')
        plt.xlabel('Number of clusters')
        plt.ylabel('Inertia')

        plt.savefig("Elbow.png")
```
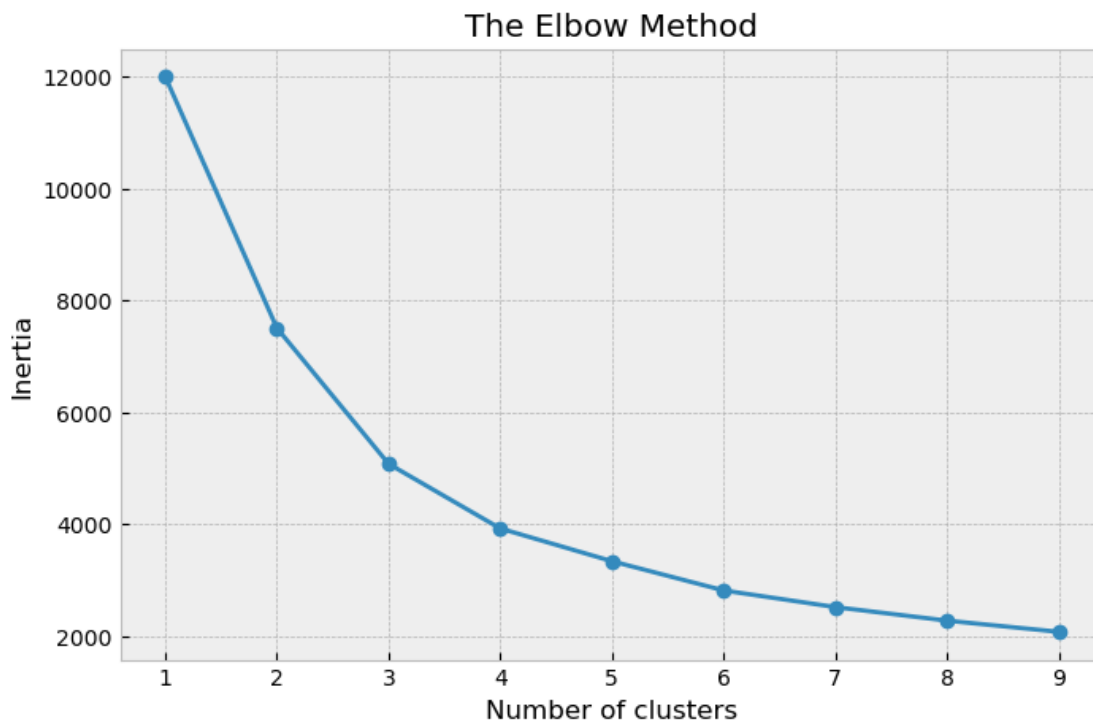
The graph indicates that the 'elbow' is on the number 3-cluster mark. Therefore, we will build our Kmeans model using **3 clusters**.

```
[73]: model = KMeans(n_clusters=3, random_state=42)
      model.fit(df_scaled)
```

```
[73]: KMeans(n_clusters=3, random_state=42)
```

```
[74]: kmeans_rfm = kmeans_rfm.assign(ClusterLabel= model.labels_)
```

```
[75]: kmeans_rfm
```

```
[75]:              recency  frequency  monetary_value  ClusterLabel
      customer_id
      1                  8         93        15150.81             2
      2                129         81         6071.88             2
      3                103         61        16413.65             2
      4                196         33         1874.87             0
      5                 17         56         9411.46             2
      ...              ...        ...             ...           ...
      3996             292          8         1982.61             0
      3997             292         87         1982.61             0
      3998             292         60         1982.61             0
      3999             292         11         1982.61             0
      4000             292         76         1982.61             0

      [3999 rows x 4 columns]
```

2.9.2 ii. Visualising the K-Means Model

```
[76]: fig = px.scatter_3d(kmeans_rfm, x='recency', y='frequency', z='monetary_value',
                          color = 'ClusterLabel', opacity=0.5)
      #fig.show()
      fig.update_traces(marker=dict(size=5),

                        selector=dict(mode='markers'))
      # tight layout
      fig.show()
```

2.9.3 iv. Calculating the Mean, Count and Percentage of K-Mean Clusters

```
[77]: # create new df with cluster means, counts and percentage

      agg_clusters = kmeans_rfm.groupby('ClusterLabel').agg({
          'recency': 'mean',
```

```
    'frequency': 'mean',
    'monetary_value': ['mean', 'count']}).round(0)

agg_clusters.columns = agg_clusters.columns.droplevel()
agg_clusters.columns = ['RecencyMean','FrequencyMean','MonetaryMean', 'Count']
agg_clusters['Percent'] = round((agg_clusters['Count']/agg_clusters.Count.
 ↪sum())*100, 2)


# Reset the index
agg_clusters = agg_clusters.reset_index()

# Change ClusterLabel into discrete values
agg_clusters['ClusterLabel'] = agg_clusters['ClusterLabel'].astype('str')


agg_clusters
```

[77]:
	ClusterLabel	RecencyMean	FrequencyMean	MonetaryMean	Count	Percent
0	0	263.0	49.0	2665.0	745	18.63
1	1	51.0	25.0	9842.0	1674	41.86
2	2	52.0	75.0	9813.0	1580	39.51

2.10 9. Interpreting the Cluster Labels:

Cluster 0: Lost or Low Spending Customers: - On average Cluster 0 customers visit the store 49 times, and spent approximately $2665 in 2017. - Cluster 0 make up 19% of the customer base. These customers shop during special promotions and prefer economical products. - Using special marketing promotions can help bring these customers back to the store - Promote cycling events, membership rewards and hold free training sessions to attract these customers back

Cluster 1: Infrequent Big Spenders: - Cluster 1 customers has the lowest average frequency of 25 counts, a moderate recency average of 51 days and high spending mean amount $9842 - These customers have high spending power, despite not being frequent as other clusters. They may buy according to times of need over want, or are busy people with little time to shop - Introduce premier loyalty membership reward programs, promote cycling events and training programs to encourage Cluster 1 customers to purchase more frequently.

Cluster 2: Best Customers: - Sprocket Central's best customers spent $9813 on avearge, has the highest frequency mean of 75, and recenecy mean of 52 days. - Cluster 2 are loyal customers and are cycling hobbyists. - They are always on the look out for the newest products and have a strong spending power. - They will continue to thrive as customers with membership programs, marketing of new products, promotions and inivitations to cycling events.

3 II. NEW CUSTOMERS DATA ANALYSIS

Applying K-Means Model to the New Customer Data

3.1 Variable Descriptions (Assumptions of New Customer Data):

- gender: sex of customer - 'Female', 'Male' or 'Other'
- past_3_years_bike_related_purchases: count of purchases in the last 3 years - numbers ranging 0 to 99
- DOB: customers' date of birth
- wealth_segment: the wealth level the customer is recorded as - 'Mass Customer', 'Affluent Customer', 'High Net Worth'
- tenure: the duration of residence at customers' address, in years
- state: the state the customer is based in - 'QLD', 'NSW', 'VIC'
- property_valuation: the valuation score of customers' property - numbers from 1 to 12
- Rank: customers' ranking based on recency score - ranking numbers from 1 (most recent) to 1000 (least recent)
- Value: customers' perceived value score of Sprocket's products, services, benefits, and costs - numbers ranging from 0.34 (lowest) to 1.71875 (highest).

3.2 1. Importing Data

```
[78]: # import data
      newdf = pd.read_excel('NewCustomerList.xlsx', header=1)
```

```
[80]: # selecting relevant columns for analysis

      newdf = newdf[['first_name', 'last_name', 'gender',
       ↪'past_3_years_bike_related_purchases', 'DOB', 'wealth_segment', 'tenure',
       ↪'state', 'property_valuation', 'Rank', 'Value']]
```

3.3 2. Data Wrangling

3.3.1 Check for Duplicates

```
[81]: newdf.duplicated().any()
```

```
[81]: False
```

3.3.2 Missing Values

```
[82]: # % of missing values

      round(newdf.isnull().sum().sort_values(ascending = False)/len(newdf)*100,2)
```

```
[82]: last_name                              2.9
      DOB                                    1.7
      first_name                             0.0
      gender                                 0.0
      past_3_years_bike_related_purchases    0.0
      wealth_segment                         0.0
      tenure                                 0.0
```

```
state                          0.0
property_valuation             0.0
Rank                           0.0
Value                          0.0
dtype: float64
```

[83]:
```python
newdf.last_name = newdf['last_name'].replace([np.nan],['Not Applicable'])
```

[84]:
```python
newdf.gender = newdf.gender.replace(['U'],['Other'])
```

3.3.3   iii. Data Formatting

Concat first_name and last_name

[85]:
```python
newdf['fullname'] = newdf['first_name'] + '_' + newdf['last_name']
newdf = newdf.drop(columns=['first_name', 'last_name'])
```

3.3.4   Converting DOB to Age Values

[86]:
```python
newdf.fillna(value={'DOB':newdf['DOB'].mode()[0]},inplace=True)
```

[87]:
```python
def calculate_age(born):
    today = date.today()
    return today.year - born.year - ((today.month, today.day) < (born.month,
    born.day))


newdf.DOB = newdf.DOB.apply(calculate_age).astype('int')
```

3.3.5   Renaming Columns

[88]:
```python
newdf = newdf.rename(columns={'past_3_years_bike_related_purchases':
    'count_purchase', 'DOB':'age'})
```

[89]:
```python
newdf.head()
```

[89]:
```
   gender  count_purchase  age      wealth_segment  tenure state  \
0    Male              86   65        Mass Customer      14   QLD
1    Male              69   52        Mass Customer      16   NSW
2  Female              10   48    Affluent Customer      10   VIC
3  Female              64   43    Affluent Customer       5   QLD
4  Female              34   57    Affluent Customer      19   NSW

   property_valuation  Rank     Value             fullname
0                   6     1  1.718750     Chickie_Brister
1                  11     1  1.718750         Morly_Genery
2                   5     1  1.718750   Ardelis_Forrester
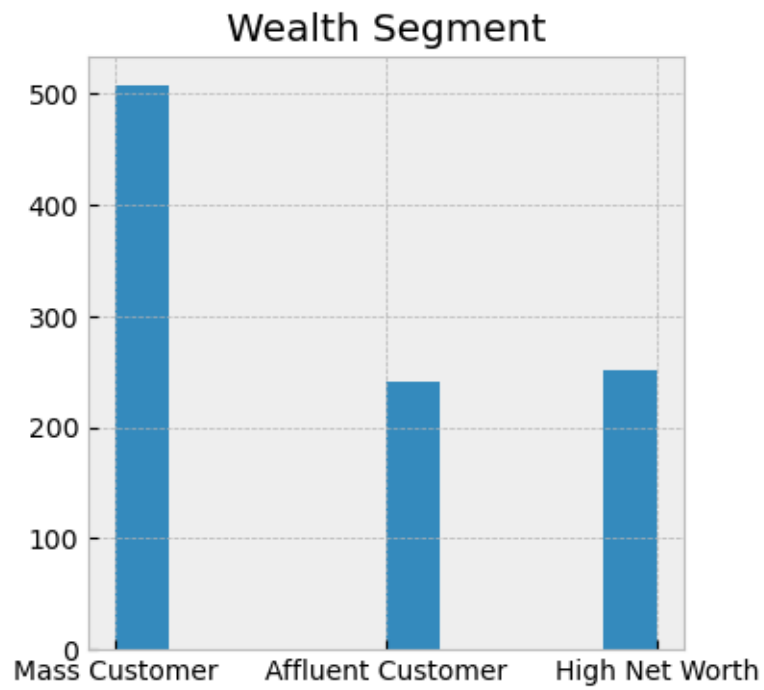3                   1     4  1.703125         Lucine_Stutt
4                   9     4  1.703125       Melinda_Hadlee
```

3.4 3. Data Exploration

3.4.1 Univariate Analysis

```
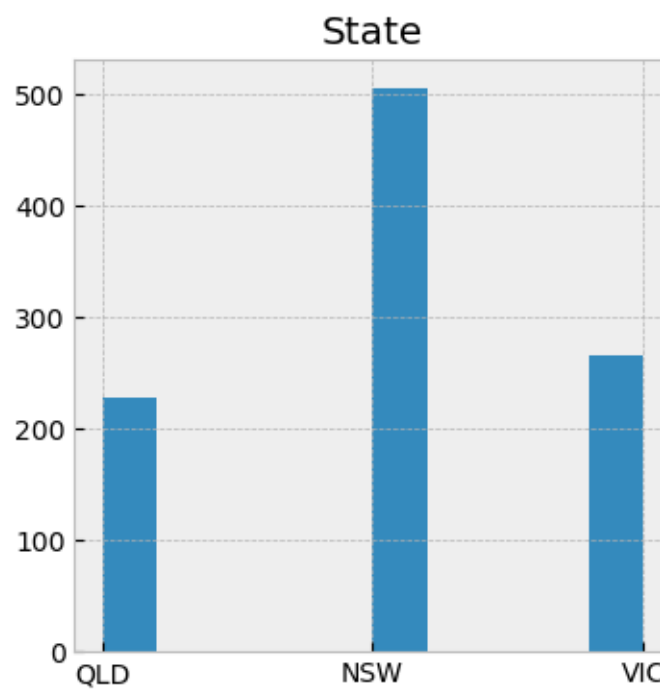[90]: plt.figure(figsize=(4, 4))
      plt.hist(newdf['gender'])
      plt.title('Gender')
      plt.show()
```
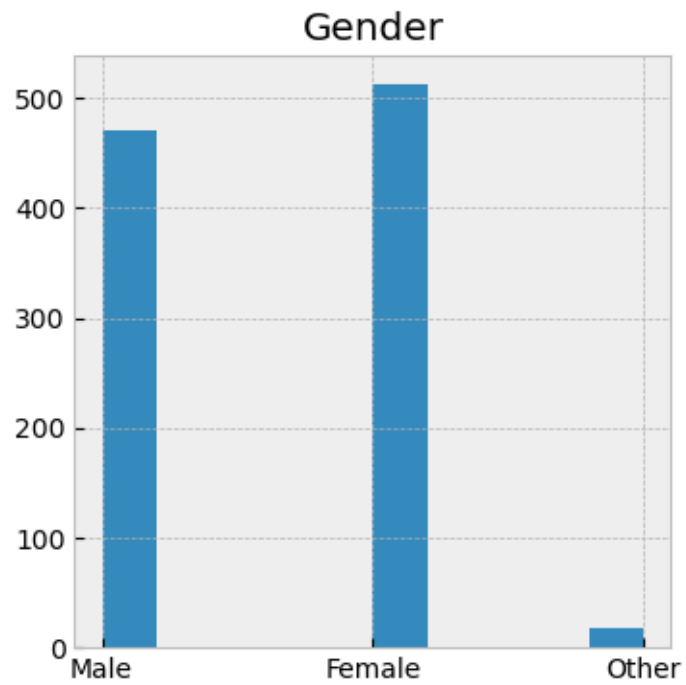


```
[91]: plt.figure(figsize=(4, 4))
      plt.hist(newdf['wealth_segment'])
      plt.title('Wealth Segment')
      plt.show()
```

Wealth Segment


```
[92]: plt.figure(figsize=(4, 4))
      plt.hist(newdf['state'])
      plt.title('State')
      plt.show()
```

State

Observations: - There are mostly Female customers, followed by Male then Other genders - Weath Segment indicates that Mass Customers make up 50% of the customer data - Most of the customers are from NSW

3.4.2 Feature Scaling

```
[93]: newdf.to_excel('NewDF.xlsx')
      newdf_copy = newdf.copy()
```

```
[94]: # scale dataset before viewing correlations using LabelEncoder for ordinal␣
       ↪column values,
      # and one-hot-encoding for nominal column values

      newdf_copy["wealth_segment"]=LabelEncoder().
       ↪fit_transform(newdf_copy["wealth_segment"])



      dummies = pd.get_dummies(newdf_copy, columns=['gender', 'state'])
      newdf_copy = pd.concat([newdf_copy, dummies], axis=1)
```

3.4.3 Correlations

```
[122]: plt.figure(figsize=(10,9))
       sns.heatmap(newdf_copy.corr(), annot=True, fmt='.1f')
       plt.show()
```
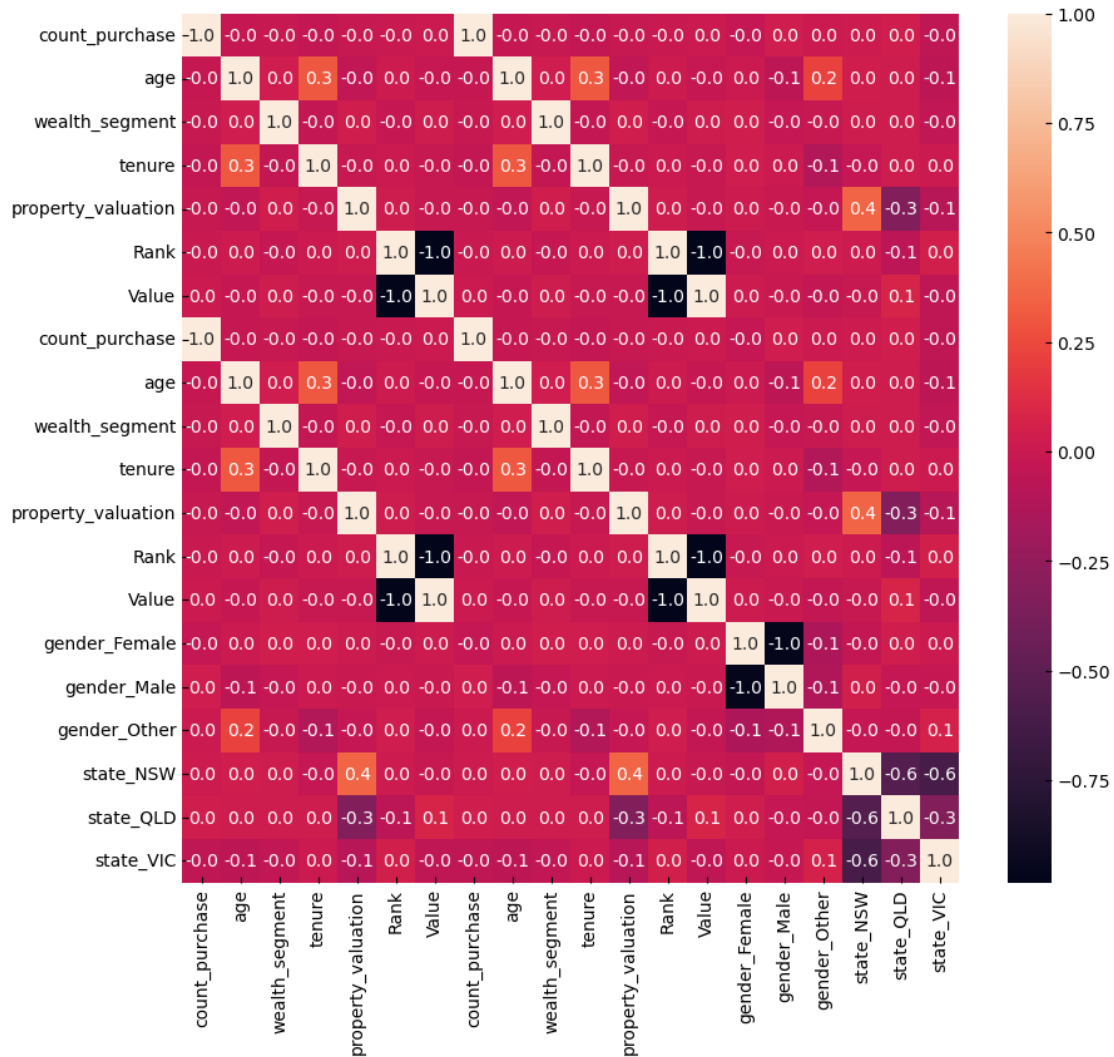
Observations: - 'age' and 'tenure' have a correlation of 0.31. This implies the older customers have a longer tenureship of their residence - 'property_valuation' has a correlation of 0.36 to 'state_NSW', suggesting that the valuation score for NSW is higher than other states - 'Value' to 'Rank' has a correlation of -0.98, indicating a strong relationship. - 'gender_Female' is strongly correlated to 'gender_Male', with a correlation of -0.97

3.4.4 Distribution Plots of Age, Rank and Value

```
[125]: plt.figure(1 , figsize = (10 , 3))
       n = 0
       for x in ['age' , 'count_purchase' , 'Value']:
           n += 1
           plt.subplot(1 , 3 , n)
           plt.subplots_adjust(hspace =0.5 , wspace = 0.5)
           sns.distplot(newdf[x] , bins = 20)
```

```
    plt.title('Distplot of {}'.format(x))
plt.show()
```
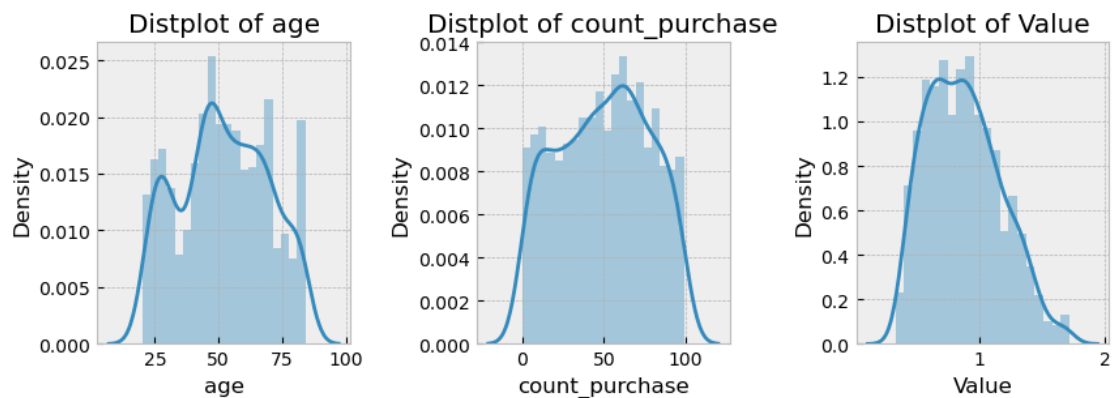

3.4.5 Skewnesss of Attributes

```
[97]: newdf.skew()
```

```
[97]: count_purchase       -0.065622
      age                   0.009863
      tenure                0.070891
      property_valuation   -0.557611
      Rank                  0.001246
      Value                 0.429903
      dtype: float64
```

Observations: All columns are fairly normally distributed

3.4.6 Identifying Outliers

```
[98]: f, axes = plt.subplots(1, 3)

      sns.boxplot(  y= "count_purchase", data=newdf,  orient='v' , ax=axes[0])
      sns.boxplot(  y= "Rank", data=newdf,  orient='v' , ax=axes[1])
      sns.boxplot(  y= "Value", data=newdf,  orient='v' , ax=axes[2])
      plt.subplots_adjust(hspace =0.5 , wspace = 0.8)
      plt.show()
```
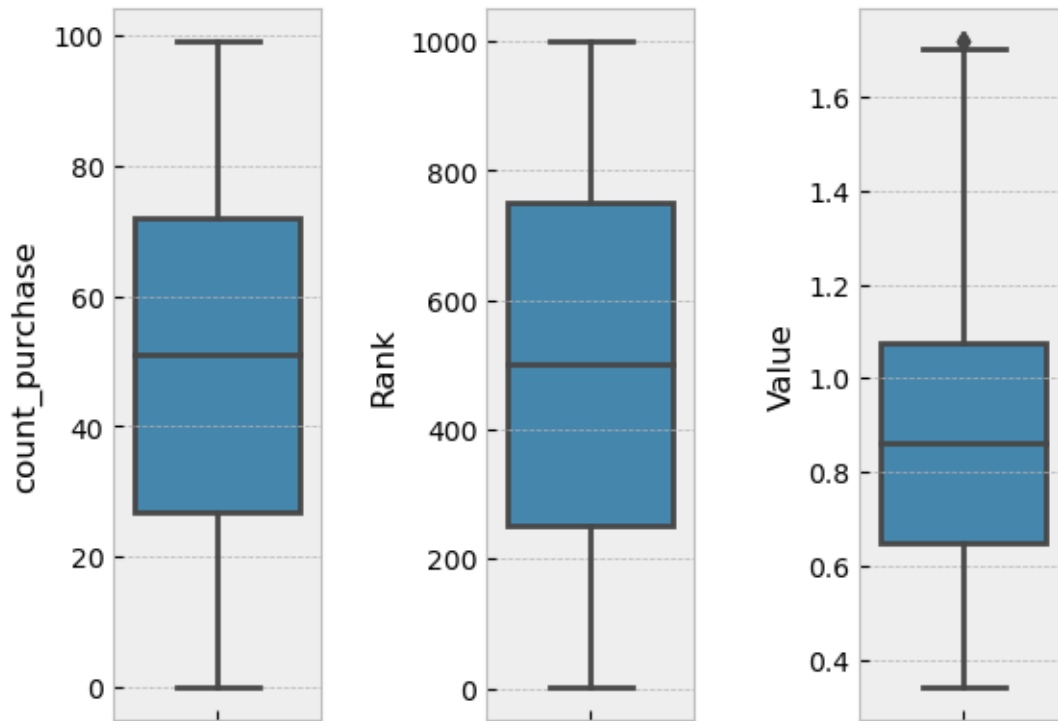
Observations: - Value has a slight outlier, attributed to the high Value scores of above 1.7 - There are no outstanding outliers for 'Rank' and 'count_purchase'

3.4.7 Regression Plots of Frequency, Rank and Value

```
[123]: plt.figure(1 , figsize = (8 , 7))
       n = 0
       for x in ['count_purchase' , 'Rank', 'Value']:
           for y in ['count_purchase' , 'Rank', 'Value']:
               n += 1
               plt.subplot(3 , 3 , n)
               plt.subplots_adjust(hspace = 0.5 , wspace = 0.5)
               sns.regplot(data = newdf, x = x , y = y)
               plt.ylabel(y.split()[0]+' '+y.split()[1] if len(y.split()) > 1 else y )
       plt.show()
```
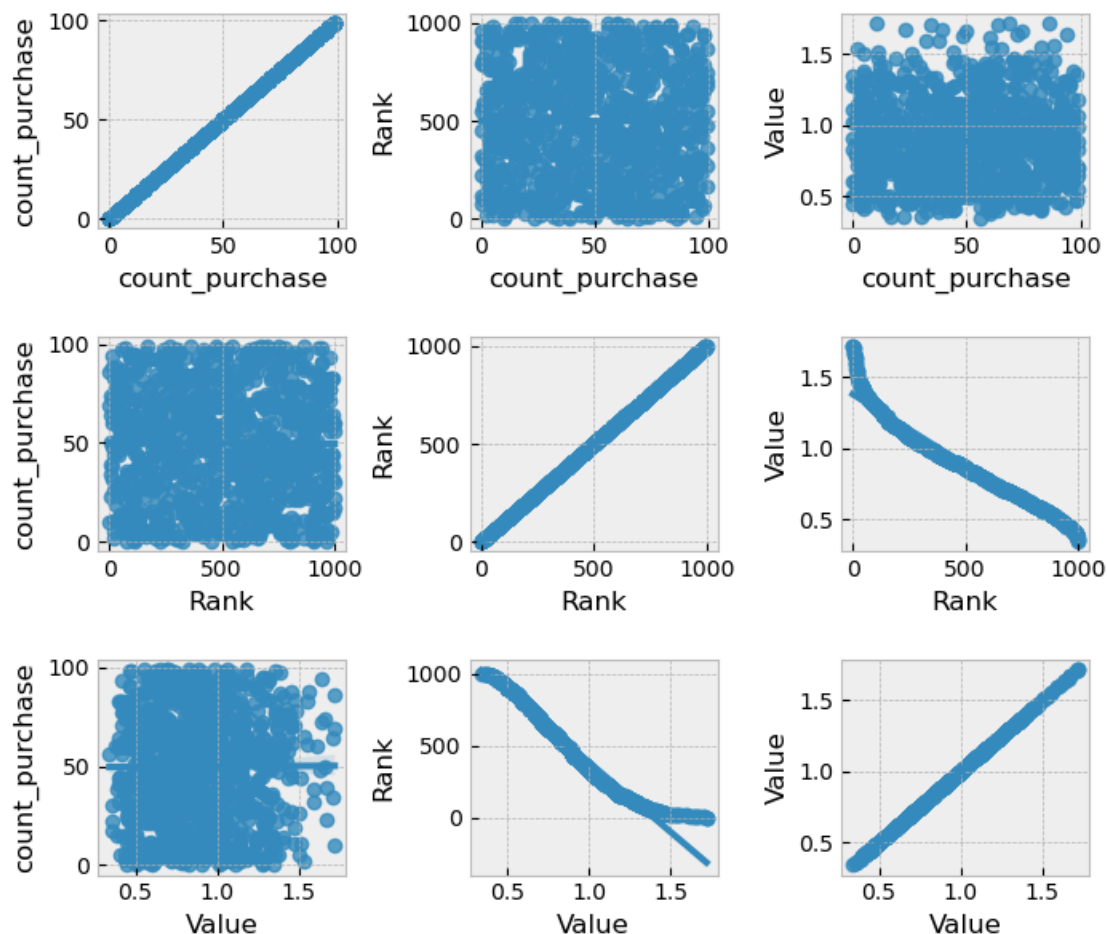
Observations: There is a relationship between Rank and Value. The lower the Rank level (measured for recency), the higher the Value score. We can attribute this to Rank score 1 being the best and most recent customers, which is in relation to a higher customers' perceived Value score, as content customers would patronage more recently.

3.5   4. Select Key Features for K-Means Analysis

```
[100]: newKmeans = newdf[['count_purchase', 'Rank', 'Value']]
```

3.6   5. Standardise the Data

```
[101]: from sklearn.preprocessing import StandardScaler
       std_scaler = StandardScaler()
       scaled_newKmeans = std_scaler.fit_transform(newKmeans)
```

3.7 6. Predetermine the K-clusters with Elbow Method

```python
[130]: from sklearn.cluster import KMeans
       SSE = []
       for k in range(1, 10):
           kmeans2 = KMeans(n_clusters=k, random_state=42)
           kmeans2.fit(scaled_newKmeans)
           SSE.append(kmeans2.inertia_) #SSE to nearest clustter centroid

       frame = pd.DataFrame({'Cluster':range(1,10), 'SSE':SSE})
       plt.figure(figsize=(8,4))
       plt.plot(frame['Cluster'], frame['SSE'], marker='o')
       plt.title('The Elbow Method')
       plt.xlabel('Number of clusters')
       plt.ylabel('Inertia')

       plt.savefig("Elbow2.png")
```
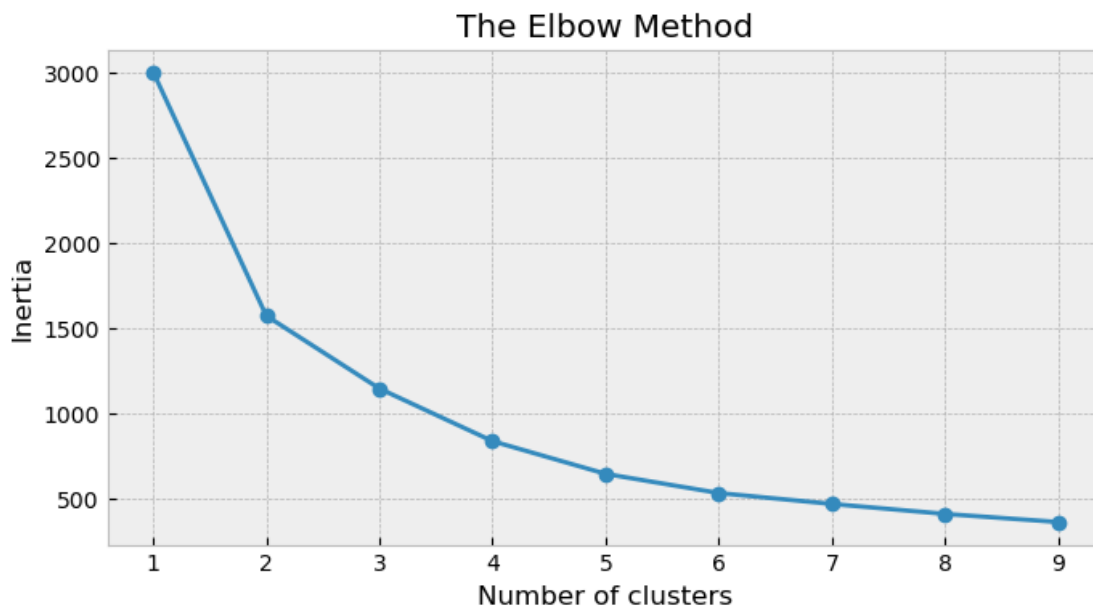

The Elbow method graph indicates that the 'elbow' is on the number 2-cluster mark. This means that the optimal number of clusters to use in this K-Means algorithm is 2. We will build our Kmeans model using 2 clusters.

3.7.1 6.a Applying Knee-Locator to Determine Optimal Cluster

```python
[103]: from kneed import KneeLocator
       kl = KneeLocator(x = range(1, 10),
                        y = SSE,
```

```
                curve="convex",
                direction="decreasing")
print('The optimal number of clusters is: ' + str(kl.elbow))
```

The optimal number of clusters is: 3

3.7.2 6b. Evaluate 3 cluster seperation

Comparison picture:

3.8 7. Fit the Model onto our Data

```
[104]: model = KMeans(n_clusters=2, init='k-means++', random_state=42)

       # fit our model int
       model.fit(scaled_newKmeans)
```

```
[104]: KMeans(n_clusters=2, random_state=42)
```

```
[105]: newKmeans = newKmeans.assign(Cluster= model.labels_)
       newKmeans
```

```
[105]:       count_purchase  Rank     Value  Cluster
       0                 86     1  1.718750        0
       1                 69     1  1.718750        0
       2                 10     1  1.718750        0
       3                 64     4  1.703125        0
       4                 34     4  1.703125        0
       ..               ...   ...       ...      ...
       995               60   996  0.374000        1
       996               22   997  0.357000        1
       997               17   997  0.357000        1
       998               30   997  0.357000        1
       999               56  1000  0.340000        1

       [1000 rows x 4 columns]
```

3.9 8. Evaluate the K-cluster Seperation

```
[106]: from sklearn.metrics import silhouette_score

       print(silhouette_score(scaled_newKmeans, model.labels_, metric='euclidean'))
```

0.4057797169525172

The silhouette coefficient of this model is 0.40, indicating reasonable cluster separation. ####
"The best value is 1 and the worst value is -1. Values near 0 indicate overlapping clusters. Negative
values generally indicate that a sample has been assigned to the wrong cluster, as a different cluster
is more similar." - source: scikit-learn.org

3.10 9. Visualising K-Means Model

```
[107]:  # 3D scatterplot of model

        fig = px.scatter_3d(newKmeans, x='count_purchase', y='Rank', z='Value',
                            color = 'Cluster', opacity=0.5)

        fig.update_traces(marker=dict(size=5),

                          selector=dict(mode='markers'))

        fig.show()
```

3.11 10. Calculating the Mean, Count and Percentage of K-Mean Clusters

```
[108]:  # create new df with cluster means, counts and percentage

        agg_newKmeans = newKmeans.groupby('Cluster').agg({
            'count_purchase': 'mean',
            'Rank': 'mean',
            'Value': ['mean', 'count']}).round(0)

        agg_newKmeans.columns = agg_newKmeans.columns.droplevel()
        agg_newKmeans.columns = ['count_purchase_Mean','Rank_Mean','Value_Mean',␣
         ↪'Count']
        agg_newKmeans['Percent'] = round((agg_newKmeans['Count']/agg_newKmeans.Count.
         ↪sum())*100, 2)

        # Reset the index
        agg_newKmeans = agg_newKmeans.reset_index()

        # Change Cluster into discrete values
        agg_newKmeans['Cluster'] = agg_newKmeans['Cluster'].astype('str')


        agg_newKmeans
```

```
[108]:    Cluster  count_purchase_Mean  Rank_Mean  Value_Mean  Count  Percent
       0     0                 49.0      237.0        1.0    477     47.7
       1     1                 50.0      737.0        1.0    523     52.3
```

```
[109]:  newKmeans.describe()
```

```
[109]:          count_purchase          Rank          Value       Cluster
       count     1000.000000   1000.000000   1000.000000   1000.000000
       mean        49.836000    498.819000      0.881714      0.523000
       std         27.796686    288.810997      0.293525      0.499721
```

```
min          0.000000     1.000000    0.340000    0.000000
25%         26.750000   250.000000    0.649531    0.000000
50%         51.000000   500.000000    0.860000    1.000000
75%         72.000000   750.250000    1.075000    1.000000
max         99.000000  1000.000000    1.718750    1.000000
```

3.12   11. Interpreting the Clusters

3.12.1   Cluster 0: Active, Regular & Satisfied Customers:

- Cluster 0 customers have a better Rank in recency score.
- These are active and regular customers, who have made approximately 50 purchases in the last 3 years.
- There are 477 Cluster 0 customers and they make up 48% of the new customer list.
- These customers also consists of new customers
- Cluster 0 customers have higher customer perceived Value (max= 1.718), indicating they are saistfied customers

Recommendated Action: - Introducing premier loyalty programs, marketing of new products, premier promotions and inivitations to cycling events, can encourage these customers to be lifelong customers

3.12.2   Cluster 1: Lost, Low Purchase or Irregular Customers

- Cluster 1 customers comprises of customers who've made none to several purchases in the past.
- They are termed Lost, Low Purchase or Irregular customers as they have low Rank in recency, implying they haven't shopped at Sprocket Central's platform recently
- They score low in customers' perceived Value, indicating either customer disappointment with Sprocket Central goods and services, switching to a competitor platform, or lost interest in the products.
- Cluster 1 Customers make up 52% of the customer database and account for 523 customers in total

Recommendated Action: - Using special marketing promotions can help bring these customers back to the store - Promote cycling events, membership rewards and hold free training sessions to attract these customers back

3.12.3   Top 10 Cluster 0 Customers

```
[110]: top = newdf.sort_values(['Rank']).head(10)
       top
```

```
[110]:    gender  count_purchase  age    wealth_segment  tenure state  \
       0    Male              86   65      Mass Customer      14   QLD
       1    Male              69   52      Mass Customer      16   NSW
       2  Female              10   48  Affluent Customer      10   VIC
       3  Female              64   43  Affluent Customer       5   QLD
       4  Female              34   57  Affluent Customer      19   NSW
```

```
5  Female                39  71      High Net Worth       22  QLD
6    Male                23  46      Mass Customer         8  NSW
7  Female                74  50      Mass Customer        10  QLD
8    Male                50  50      Mass Customer         5  NSW
9    Male                72  37      Mass Customer        17  QLD

   property_valuation  Rank    Value           fullname
0                   6     1  1.718750    Chickie_Brister
1                  11     1  1.718750        Morly_Genery
2                   5     1  1.718750  Ardelis_Forrester
3                   1     4  1.703125        Lucine_Stutt
4                   9     4  1.703125      Melinda_Hadlee
5                   7     6  1.671875        Druci_Brandli
6                   7     6  1.671875      Rutledge_Hallt
7                   5     8  1.656250         Nancie_Vian
8                  10     8  1.656250      Duff_Karlowicz
9                   5    10  1.640625      Barthel_Docket
```

[ ]: