Est. 1841 | YORK ST JOHN UNIVERSITY

York St John University, London

Department of Data Science

Course Name: LDS7004M – Data Visualization

Data Visualization and the Identification of Valuable Customers for Sprocket Ltd.

Student Name: Jamiu Adeyemi Arogundade

Student ID: 240024714

Submission Date: 27 May 2024

Module Title	Data Visualization
Module code	LDS7004M
Student ID	240024714
Submission Date	27-May-2024

Student Declaration:

I confirm that I have read and understood the University Policy and Procedures on Academic Misconduct and that the work submitted is my own.	☐
I confirm that I have processed and produced this submission in accordance with the University guidelines and the assessment brief regarding the use of generative AI. Where appropriate, I have acknowledged where and how it has been used. OR Where generative AI is not permitted in this assignment. I confirm that it has not been used in any part of the process or production of this submission https://www.yorksj.ac.uk/policies-and-documents/generative-artificial-intelligence/	☐ ☐

Table Of Content

List Of Figures

Fig.38 Transaction count by state and product line.
Fig.39 Transaction count by brand and gender Fig.40
Transaction count by brand and age group.
Fig.41 Customer value vs features

1. Introduction

Data Visualization important in today's data-populated environments as businesses races to derive actionable insights from the available of data. Data visualization interpret complex data into visual formats, making it easier to understand and interpret. This report focuses on leveraging data visualization to identify valuable customers for Sprocket Ltd., a bicycle retail company. By analysing customer demographics, transaction behaviours, and revenue data, the aim is to uncover patterns and insights that can drive strategic decisions and enhance business performance while also identifying valuable customers.

2. Data Visualization

2.1 Definition

Data visualization means drawing graphic displays to show data. Sometimes every data point is drawn, as in a scatterplot, sometimes statistical summaries may be shown, as in a histogram. The displays are mainly descriptive, concentrating on 'raw' data and simple summaries. They can include displays of transformed data, sometimes based on complicated transformations. Data visualization is useful for data cleaning, exploring data structure, detecting outliers and unusual groups, identifying trends and clusters, spotting local patterns, evaluating modelling output, and presenting results. It is essential for exploratory data analysis and data mining to check data quality and to help analysts become familiar with the structure and features of the data before them (Unwin, A, 2020).

2.2 Importance

The importance of data visualization lies in its ability to simplify complex data sets, enabling stakeholders to grasp critical insights quickly. Visual information improves communication, reduces misinterpretation, and clarifies massive or complex information. Visual information can help data scientists better understand relationships and spot patterns. Good visuals draw the viewers in, allowing them to examine the visuals at their own pace and to discover current information at their leisure (Ahmed Malik & Ünlü, 2011).

Data visualization is critical in business, as it makes it easier for the top management and marketing team of a company to make data-based decisions by analysing the competition, identifying key influencers who can increase sales volume, identifying the important threats and opportunities present in the market, and making decisions about the business's foray into a particular period, including launch of new products or services. The visualization tools can be used to communicate business information effectively to both external and internal stakeholders. Hence, data is phased, summarised, and then visualised so all major findings of critical information can be obtained at crucial hours of business decision-making (Mkhinini Gahar et al., 2024). It is important to be ahead of competition and win on the market with high-quality and high-precision accurate forecasting and predictions in companies' decision-making. From a market perspective, forecast prices of commodities can help in identifying safer transactions on the stock market and more profitable transactions for other types depending on the nature of the demand, of the market and the product. Generally, a few numbers of companies prefer and decide to sell a product before conducting a complete analysis of a huge dataset. If products do not match the needs and expectations of a specified customer, it can lead to unsold stock in the company which has the direct consequence of decreasing gains by investing in unspecialised products (Zia et al., 2022).

3. Dataset Description

3.1 Dataset and Sprocket Central Pty Ltd.

Sprocket Central Pty Ltd. is a hypothetical bike store with branches in three states in Australia. The dataset stored in an Excel spreadsheet format has three different data sheets named Customer_Demographic, Customer_Address, and Transactions.

3.2 Data Source

The dataset is synthetic structured data, sourced from Kaggle and was from KPMG virtual internship assessment.

3.3 Tools and Libraries

While most languages have associated packages and libraries built specifically for visualization tasks, Python is uniquely empowered to be a convenient tool for data visualization. Python performs advanced numerical and scientific computations with libraries such as NumPy and SciPy, provides a great interface for big data manipulation due to the availability of the pandas package and generates aesthetically pleasing plots and figures with libraries such as seaborn, plotly, and more (Belorkar, 2020).

The tools and libraries used in this project are as follows;

Jupyter Notebook: Jupyter Notebook is an open-source web application that allows creation and sharing of documents containing live code, visualisations, and narrative text. It provides an interactive environment for data exploration, analysis, and visualization, making it an ideal choice for your data visualization project (Kluyver, T. et al., 2016).

Pandas: Pandas is a powerful data manipulation and analysis library for Python. It provides data structures and data analysis tools for working with structured (tabular) and time-series data (McKinney, W, 2017).

NumPy: NumPy is a fundamental library for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of highlevel mathematical functions to operate on these arrays. NumPy is often used with Pandas for data manipulation and preparation tasks (Oliphant, T E, 2006).

Matplotlib: Matplotlib is a comprehensive library for creating static, publication-quality 2D and 3D visualisations in Python. It provides a low-level framework for creating a wide variety of plots, including line plots, scatter plots, bar charts, histograms, and more. In this project, Matplotlib is used to generate basic visualisations and explore the data visually (Hunter, J. D, 2007).

Seaborn: Seaborn is a Python data visualization library based on Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Seaborn is particularly useful for creating complex visualisations, such as heatmaps, cluster plots, and regression plots, which can reveal patterns and relationships in your data (Waskom, M, 2021).

Plotly: Plotly is a Python library for creating interactive, web-based visualisations. It offers a wide range of chart types, including scatter plots, line charts, bar charts, pie charts, and more. Plotly's interactive features, such as hover tooltips, zoom, and pan, allow for a more engaging and exploratory data analysis experience (https://plotly.com/python/).

Plotly Express: Plotly Express is a high-level, declarative interface for creating Plotly visualisations. It simplifies the process of creating complex and interactive visualisations by providing a more concise and intuitive syntax, making it easier to explore and communicate your data insights visually (https://plotly.com/python/).

Dash: Dash is a Python framework for building analytical web applications. It is built on top of Plotly and allows you to create interactive dashboards and data visualization applications that can be deployed on the web. By using Dash in your project, you can create an interactive interface for exploring and presenting your visualisations, enhancing the user experience, and enabling deeper insights (https://plotly.com/python/).

Python's extensive ecosystem of data visualization libraries and tools has made it a powerful language for exploring and communicating data insights visually. By leveraging these tools, creating a wide range of static and interactive visualisations became possible, uncovering patterns, trends, and valuable insights (Kyrola, A., 2021).

3.4 Data Collection and Importation

The dataset was downloaded directly from Kaggle and saved onto the local drive. The next steps involved importing the Pandas library to read in the MS. Excel formatted dataset from the local drive and data manipulation, NumPy for mathematical functions, and Matplotlib and Seaborn libraries for graphs and charts visualisation. The three sheets from the file were imported individually and were merged into one data frame with the help of Pandas.

Importing Libraries

```python
import pandas as pd
import warnings #ignore warnings
warnings.filterwarnings('ignore')
```

Importing the Dataset

```python
pd.set_option('display.max_columns', None) # Display all the columnsin the DataFrame

file_path = 'KPMG_VI_New_raw_data_update_final.xlsx' # Declaring the path to the Excel file

needed_sheets = ['CustomerDemographic', 'CustomerAddress', 'Transactions'] # Storing the sheets in to a variable

imported_dfs = pd.read_excel(file_path, needed_sheets) # Reading the file path and sheets with pandas and saving in 'import

# Saving each sheet now in pandas dataframe into individual dataframe for easier exploration

customer_demographic_df = imported_dfs['CustomerDemographic']
customer_address_df = imported_dfs['CustomerAddress']
transaction_df = imported_dfs['Transactions']
```

```python
# Merge customer_demographics_df and customer_address_df
merged_df = pd.merge(customer_address_df, customer_demographic_df, on='customer_id', how='right')

# Perform the LEFT JOIN with transaction_df
df = pd.merge(merged_df, transaction_df, on='customer_id', how='left')

# Show DataFrame
df.head(5)
```

Fig.1 Importing libraries and data sheets.

```python
# import Numpy for data manipulations
import numpy as np

# For visualizations
import matplotlib.pyplot as plt
import seaborn as sns
```

Fig.2 Importing NumPy, Matplotlib, and Seaborn libraries.

	customer_id	address	postcode	state	country	property_valuation	first_name	last_name	gender	past_3_years_bike_related_purchases	DOB	job_
0	1	060 Morning Avenue	2016.0	New South Wales	Australia	10.0	Laraine	Medendorp	F	93	1953-10-12	Execu Secre
1	1	060 Morning Avenue	2016.0	New South Wales	Australia	10.0	Laraine	Medendorp	F	93	1953-10-12	Execu Secre
2	1	060 Morning Avenue	2016.0	New South Wales	Australia	10.0	Laraine	Medendorp	F	93	1953-10-12	Execu Secre
3	1	060 Morning Avenue	2016.0	New South Wales	Australia	10.0	Laraine	Medendorp	F	93	1953-10-12	Execu Secre
4	1	060 Morning Avenue	2016.0	New South Wales	Australia	10.0	Laraine	Medendorp	F	93	1953-10-12	Execu Secre

```python
print(f'DataFrame has {df.shape[0]} rows and {df.shape[1]} columns')
```

```
DataFrame has 20504 rows and 30 columns
```

Fig.3 Displaying Data Frame

3.5 Data Columns and Types

The dataset comprises a wide range of columns, each contributing valuable information about Sprocket Central Pty Ltd.'s customers. The columns include:

customer_id: Unique identifier for each customer (int64) address: A column representing the customer's address (object) postcode: The postal code of customers (float64) state: Customers' State of Residence (object) country: Customers' Country of Residence (Object) property_valuation: Number of the customer's property (float64) first_name: Customer's first name (object) last_name: Customer's last name (object) gender: Gender of the customer (object) past_3_years_bike_related_purchases: Number of bike-related purchases in the past three years (int64)

DOB: Date of birth (datetime64[ns]) job_title: Job title of customers (object) job_industry_category: Industry category of the job (object) wealth_segment: Customers' wealth segment classification (object) deceased_indicator: Indicator if the customer is deceased (object) default: (object) * owns_car: Column representing if a customer owns a car (object) tenure: Customer tenure with the company (float64) transaction_id: Unique identifier for transactions (float64) product_id: Product identifier (float64) transaction_date: Date of transaction (datetime64[ns]) online_order: Indicator if the order was online (float64) order_status: Status of the order (object) brand: Brand of the product (object) product_line: Product line category (object) product_class: Product class category (object) product_size: Size of the product (object) list_price: Listed price of the product (float64) standard_cost: Standard cost of the product (float64) product_first_sold_date: Date the product was first sold (datetime64[ns])

```
# Show DataFrame Info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20504 entries, 0 to 20503
Data columns (total 30 columns):
 #   Column                               Non-Null Count  Dtype
---  ------                               --------------  -----
 0   customer_id                          20504 non-null  int64
 1   address                              20475 non-null  object
 2   postcode                             20475 non-null  float64
 3   state                                20475 non-null  object
 4   country                              20475 non-null  object
 5   property_valuation                   20475 non-null  float64
 6   first_name                           20504 non-null  object
 7   last_name                            19849 non-null  object
 8   gender                               20504 non-null  object
 9   past_3_years_bike_related_purchases  20504 non-null  int64
 10  DOB                                  20047 non-null  datetime64[ns]
 11  job_title                            18027 non-null  object
 12  job_industry_category                17180 non-null  object
 13  wealth_segment                       20504 non-null  object
 14  deceased_indicator                   20504 non-null  object
 15  default                              19005 non-null  object
 16  owns_car                             20504 non-null  object
 17  tenure                               20047 non-null  float64
 18  transaction_id                       19997 non-null  float64
 19  product_id                           19997 non-null  float64
 20  transaction_date                     19997 non-null  datetime64[ns]
 21  online_order                         19637 non-null  float64
 22  order_status                         19997 non-null  object
 23  brand                                19800 non-null  object
 24  product_line                         19800 non-null  object
 25  product_class                        19800 non-null  object
 26  product_size                         19800 non-null  object
 27  list_price                           19997 non-null  float64
 28  standard_cost                        19800 non-null  float64
 29  product_first_sold_date              19800 non-null  datetime64[ns]
dtypes: datetime64[ns](3), float64(8), int64(2), object(17)
memory usage: 4.8+ MB
```

Fig.4 Overview of the Data Frame including the column names, NULL values count, and data types.

4. Explorative Data Analysis

Explorative Data Analysis is the process of data cleaning and preprocessing crucial steps in preparing the dataset for analysis. This involves identifying general pattern in the data. These patterns include handling missing values, correcting discrepancies, identifying outliers, and transforming data types to ensure accuracy and reliability. For instance, the missing values in most of the columns were addressed, discrepancies in the 'gender' column were corrected, 'transaction_date' was converted to a datetime format for temporal analysis, creation of new columns for feature engineering, and other cleaning and preprocessing were carried out on the dataset as it will be shown below.

4.1 Dropping of missing values in 'address' column:

Here, Pandas function 'dropna()' was used to drop missing values in the column.

```
# Drop rows where address is empty
df.dropna(subset=['address'], inplace=True)
df.shape
```
```
(20475, 30)
```

Fig.5 Dropping NULL values in the data frame using the address column as subset.

4.2 Changing 'postcode' column data type:

In this column, conversion of the data type from float to integer was achieved.

```
# Convert the postcode from Object data type to Interger (whole number)
df['postcode'] = df['postcode'].astype(int)
df['postcode'].dtypes
```
```
dtype('int32')
```

Fig.6 Converting postcode column data type.

4.3 Standardisation of the 'state' column values:

Here, Pandas function ‘replace()’ was used to replace the abbreviations 'NSW', 'QLD', and 'VIC' with their respective full state names 'New South Wales', 'Queensland', and 'Victoria'.

```
# Show the count of values in the state column

df['state'].value_counts()
```

```
NSW                10472
VIC                 4682
QLD                 4356
New South Wales      485
Victoria             480
Name: state, dtype: int64
```

Fig.7 Overview of state column.

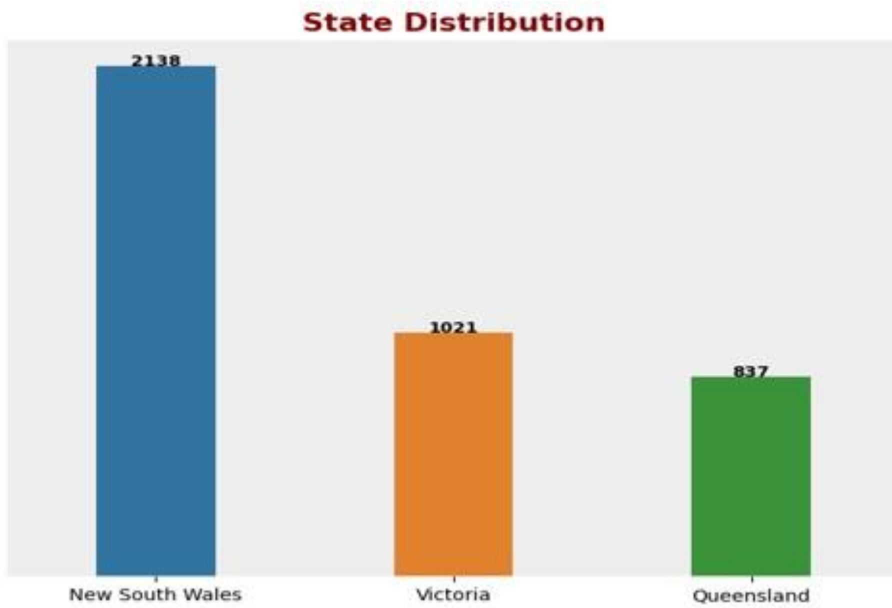


Fig.8 State column distribution after standardisation.

The purpose of this operation was likely to standardise the values in the 'state' column. Initially, the column had a mix of abbreviations ('NSW', 'VIC', 'QLD') and full state names ('New South Wales', 'Victoria'). This inconsistency can cause issues when analysing or visualising the data.

4.4 Conversion of ‘property_valuation’ column data type and segmentation into groups in a column:

Conversion of the column from segmentation of customers based on their property valuation and visualise the distribution of customers across these valuation groups.

```
# Convert the data type to integer
df['property_valuation'] = df['property_valuation'].astype(int)

# Print Property valuation datatype
print(df['property_valuation'].dtypes)
print()
# Print Property Valuation unique values
print(df['property_valuation'].unique())
```

```
int32

[10  9  4 12  8  6  7  3  5 11  1  2]
```

Fig.9 Overview of property valuation column.

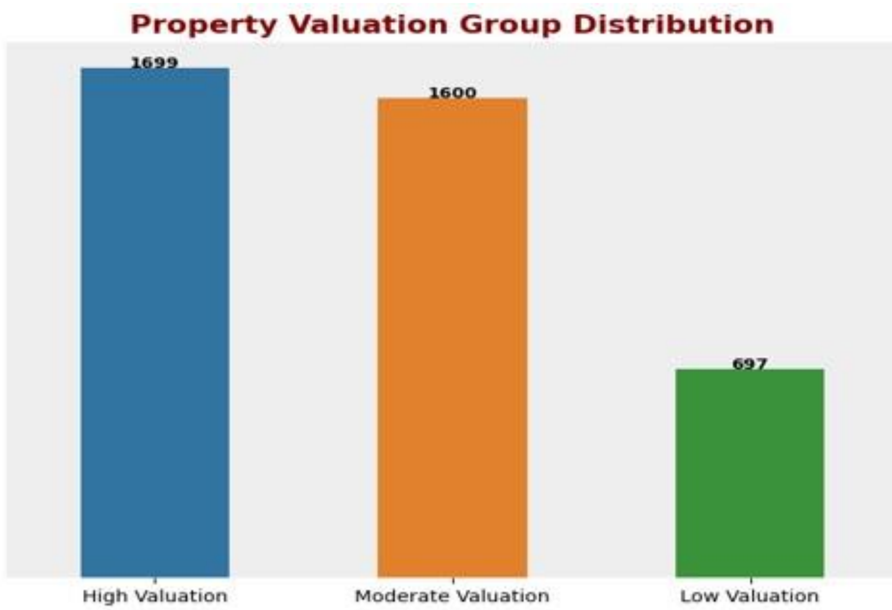


Fig.10 Property valuation group created from the property valuation column.

The purpose of this is to segment the customers based on their property valuation and visualise the distribution of customers across these valuation groups. This can provide insights into the customer base and potentially help in identifying valuable customer segments based on their property values.

4.5 Fixing the inconsistencies in the 'gender' column:

The presence of values like 'U', 'F', 'Femal', and 'M' in the 'gender' column suggests potential data entry errors or inconsistencies in the way the gender information was recorded. These values do not conform to the expected or standard abbreviations for gender ('Female' and

'Male'). It is crucial to identify and address such errors or inconsistencies during the data cleaning and preparation stage to ensure accurate analysis and reliable insights.

```
# Show the value count of the customers gender

df['gender'].value_counts()
```

```
Female    10258
Male       9727
U           466
F            11
Femal         7
M             6
Name: gender, dtype: int64
```

Fig.11 Overview of the gender column.

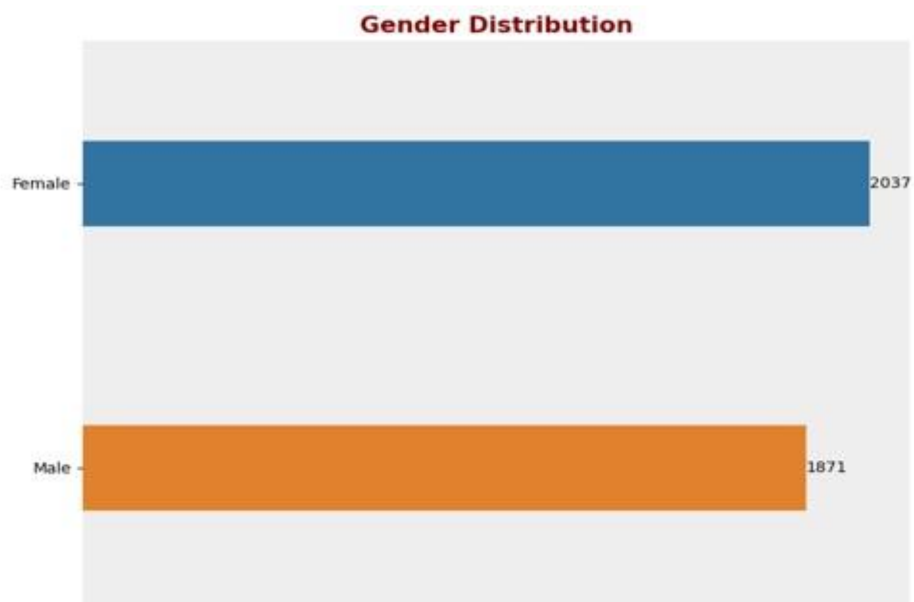


Fig.12 Gender column after fixing errors and inconsistencies.

The purpose of this is to clean the 'gender' column by addressing potential errors or inconsistencies, remove rows with missing or unknown gender values, and then visualise the distribution of customers across the 'Female' and 'Male' gender categories using a horizontal bar chart. This analysis can provide insights into the gender composition of the customer base and potentially inform gender-specific marketing or product strategies.

4.6 Categorising Customers Based on Past 3 Years Bike-Related Purchase Activity:

First step is to calculate the description of the distribution of the 'past_3_years_bike_related_purchases' column using Pandas function' providing summary statistics to understand the range and central tendency of customer purchase activity over the past three years.

```
df['past_3_years_bike_related_purchases'].describe()

count    20009.000000
mean        48.922885
std         28.678292
min          0.000000
25%         24.000000
50%         48.000000
75%         73.000000
max         99.000000
Name: past_3_years_bike_related_purchases, dtype: float64
```

Fig.13 Past 3 years bike-related purchases column description.

The second step is to categorise customers into different activity levels based on their past three years of bike-related purchases, using a custom function. This categorisation helps segment customers based on their engagement and purchase behaviour. We then create a bar chart visualization to depict the distribution of customers across the different activity levels. This visual representation aids in identifying valuable customer segments based on their purchase activity, allowing the business to tailor marketing strategies, product offerings, and customer retention efforts accordingly.

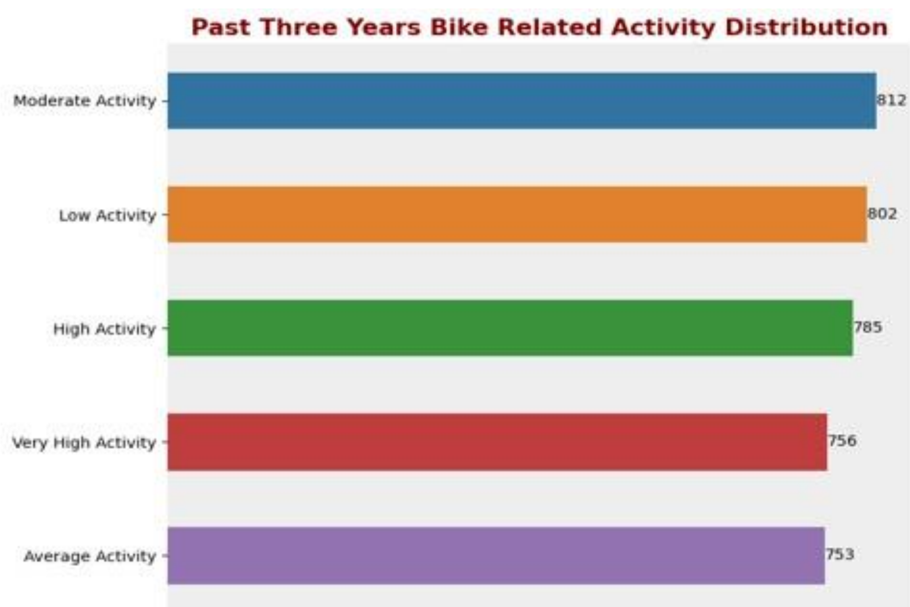


Fig.14 Newly created past 3 years bike related activity column distribution.

4.7 Customer Age Analysis and Distribution:

The primary objective here is to conduct a comprehensive analysis of the customer age distribution within the dataset by leveraging the powerful capabilities of Python's 'datetime' library to calculate the current age of each customer based on their recorded date of birth and the present date. This calculation results in the creation of a new column named 'age' within the DataFrame, which stores the age information for each respective customer.

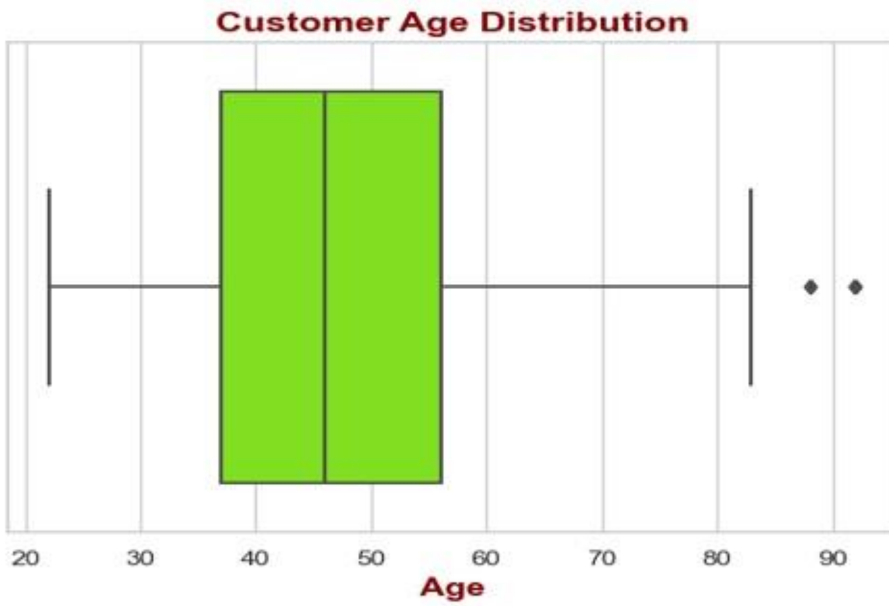


Fig.15 Customer age column with outliers.

After calculating and creating the new 'age' column, outliers were detected among the columns values and was fixed by setting the column percentile to 25 for lower quartile and 75 for upper quartile using NumPy 'percentile' function. Subsequently, using that Pandas 'clip' function, the age limit was set between 17 and 80 years old for the customers.

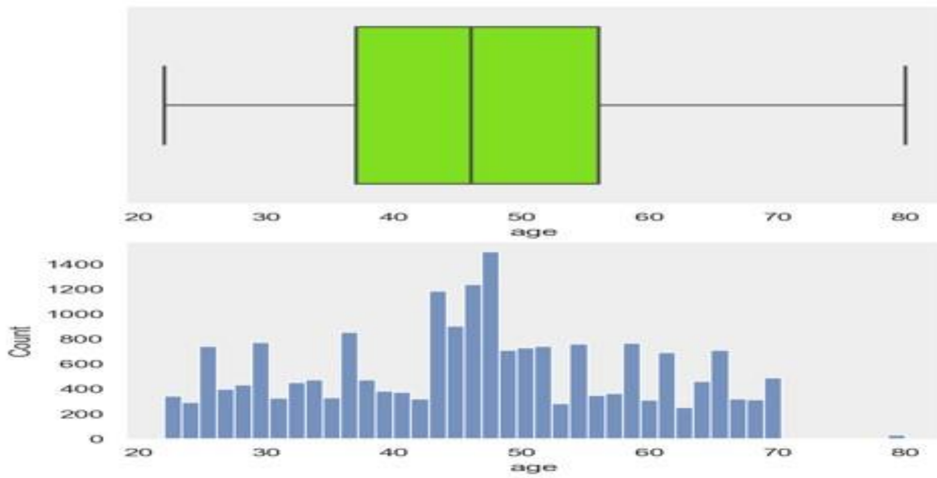


Fig.16 Customer age column distribution after eliminating with outliers.

The next step involved creating a new column ‘age_group’ to segment customers into different age categories.

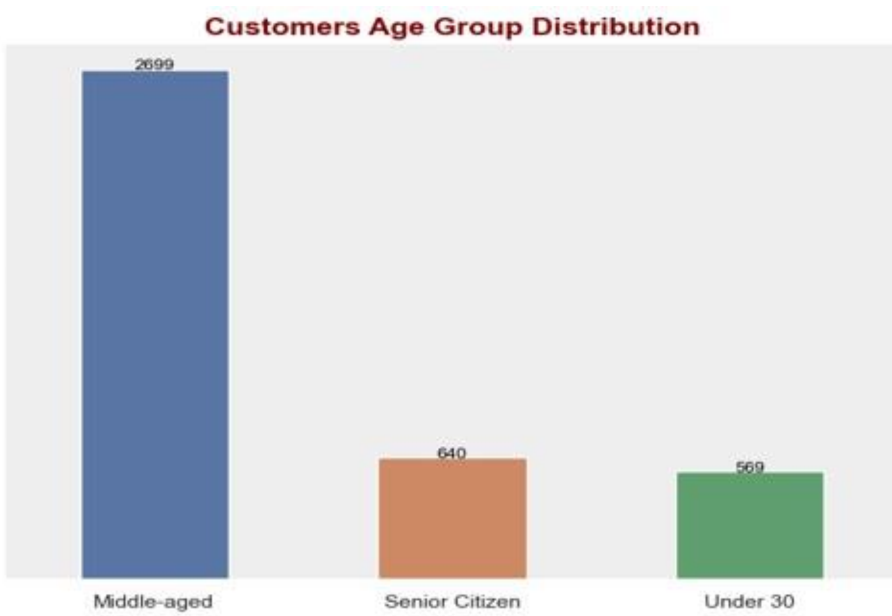


Fig.17 Customers age group column distribution.

This analysis aims to provide valuable insights into the age composition of the customer base, which can inform strategic decision-making processes. By identifying age-specific patterns and trends, businesses can tailor their marketing strategies, product offerings, and customer engagement initiatives to better cater to the diverse needs and preferences of their target audience across various age groups.

4.8 Data Cleaning and Preprocessing for ‘job_title’ Column:

Firstly, addressed a spelling error in the 'job_industry_category' column by replacing 'Argiculture' with 'Agriculture'. This ensures consistency.

Secondly, handled the missing values in the 'job_title' column. A custom function assigns a default job title based on the 'job_industry_category' value. This fills in missing information. After applying the function, rows with remaining null values in 'job_title' are dropped. This ensures data completeness.

The primary purpose is data cleaning and preprocessing for job-related columns. Spelling errors are addressed. Missing values are handled. Data quality and integrity are enhanced for rich analyses or visualisations involving job titles and industry categories.

Fig.17 Customers job title column distribution.

4.9 Handling Missing and Invalid Values in ‘job_Industry_category’ column:

Replacement of missing (NaN) and invalid (‘n/a’) values in the 'job_industry_category' column with the value 'Uncategorised'. A custom function 'update_job_category' is defined. It checks for non-empty 'job_title' values and replaces the corresponding 'job_industry_category' with 'Uncategorised' if it's NaN or ‘n/a’. The function is applied to the DataFrame using 'apply'. This step ensures consistent and meaningful data in the 'job_industry_category' column for further analysis.

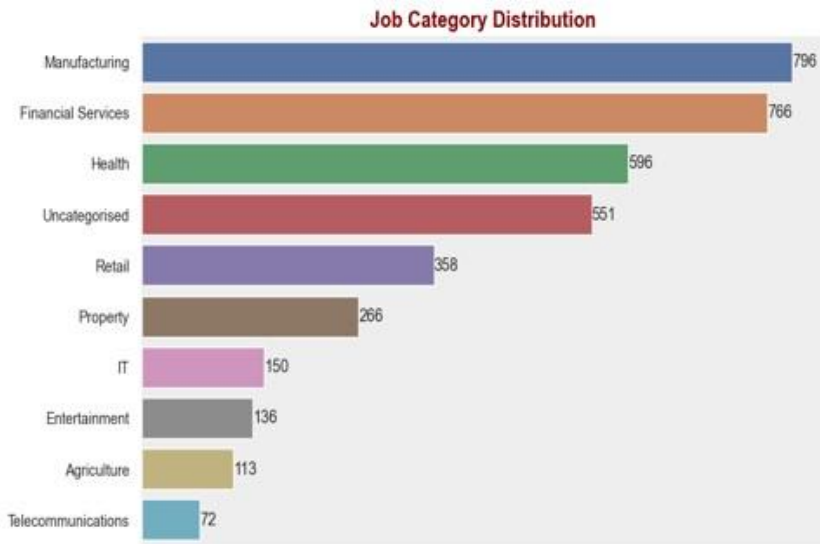


Fig.18 Customers job industry category column distribution.

4.10 Creation of New Group from the Tenure Column:

Converting the 'tenure' column to integers, creates a new 'tenure_group' column categorising customers into 'New', 'Regular', 'Loyal', and 'Long-Term' based on tenure values and then visualises the distribution using a bar plot. The analysis provides insights into customer loyalty and retention patterns.

Fig.19 Customers tenure group column distribution after creating the column.

4.11 Extraction of Month from 'transaction_date' Column:

Extracting the month from the 'transaction_date' column and created a new 'transaction_month' column providing insights into monthly transaction patterns.

Fig.20 Transaction month column distribution after extracting from the transaction date column.

The plot shows all the month have close number of transactions in the given year with October being the busiest month at the store in the year. Whereas in the month before, in September, low transactions were recorded.

4.12 Revenue Calculation and Analysis:

Calculated the revenue for each product by subtracting the standard cost from the list price. This analysis can help identify high-revenue products, assess pricing strategies, and inform decision-making related to product profitability and inventory management.

```
df['revenue'] = df['list_price'] - df['standard_cost']

df['revenue'].describe()
```

```
count    18509.000000
mean       550.413941
std        493.160645
min          4.800000
25%        133.780000
50%        445.210000
75%        827.160000
max       1702.550000
Name: revenue, dtype: float64
```

Fig.21 Created new column for revenue after calculating from list price and standard cost column.

4.13 Customer Transaction Analysis and Value Identification:

Performed several operations to analyse customer transactions and identify valuable customers based on a transaction count threshold. Calculated the transaction count for each customer, groups the data by relevant columns, sorted by transaction count in descending order, and calculated the transaction percentage. Customers are then ranked by transaction count and defined a transaction count threshold (40% of the maximum) and assigns a binary value (0 or 1) to a 'customer_value' column based on whether the customer's transaction count exceeds the threshold. The resulting DataFrame provides insights into customer transaction patterns and helps identify valuable customers for targeted marketing and retention strategies.

```
# Calculate Transaction Count
df['transaction_count'] = df.groupby(['customer_id'])['transaction_id'].transform('count')

# Group by and aggregate the data
df = df.groupby(['customer_id', 'first_name', 'last_name', 'transaction_id', 'product_id', 'state',
                 'property_valuation','property_valuation_group','gender', 'past_3_years_bike_related_purchases',
                 'past_3_years_bike_related_activity', 'age', 'age_group', 'job_title', 'job_industry_category', 'w
                 'owns_car', 'deceased_indicator', 'tenure', 'tenure_group', 'transaction_date', 'transaction_month
                 'order_status','brand', 'product_line', 'product_class', 'product_size', 'list_price',
                 'standard_cost','revenue','product_first_sold_date']).agg(
                     transaction_count=('transaction_count', 'sum')).reset_index()

# Sort by transaction_count in descending order
df = df.sort_values(by='transaction_count', ascending=False)

# Calculate transaction_percentage
transaction_count = df['transaction_count'].count()
df['transaction_percentage'] = (df['transaction_count'] * 100.0 / transaction_count
                               ).round(2)

# Calculate Rank
df['Rank'] = df['transaction_count'].rank(ascending=False, method='min').astype(int)

# Calculate Value based on a threshold (mean transaction_count)
transaction_count_threshold = 0.40 * df['transaction_count'].max()
df['customer_value'] = (df['transaction_count'] > transaction_count_threshold).astype(int)

# Display the result
print(f'Transaction Count Threshold: {transaction_count_threshold:.2f}. \n The threshold is set at 40% of the highest tran
df
```

```
df['customer_value'].value_counts()
```

```
1    11674
0     6255
Name: customer_value, dtype: int64
```

Fig.22 Calculated and created new column for the customer value.

	customer_id	first_name	last_name	transaction_id	product_id	state	property_valuation	property_valuation_group	gender	past_3_years_bik
6684	1068	Frazer	Searston	11472	29	New South Wales	8	Moderate Valuation	Male	
11303	2183	Jillie	Fyndon	134	78	Queensland	4	Low Valuation	Female	
6659	1068	Frazer	Searston	4038	97	New South Wales	8	Moderate Valuation	Male	
6680	1068	Frazer	Searston	4317	69	New South Wales	8	Moderate Valuation	Male	
6681	1068	Frazer	Searston	4437	99	New South Wales	8	Moderate Valuation	Male	
...	...	...	...	...	...	...	...	...	...	
7166	1387	Natalee	Comport	3351	94	Victoria	6	Moderate Valuation	Female	
9677	1865	Isabelita	Kitchener	7405	21	Victoria	9	High Valuation	Female	
12118	2352	Cilka	Dabbes	14554	54	Victoria	10	High Valuation	Female	
14833	2863	Alsandair	Fetherstone	2375	67	New South Wales	3	Low Valuation	Male	
9972	1921	Cybill	Wakes	14249	81	New South Wales	9	High Valuation	Female	

17929 rows × 36 columns

past_3_years_bike_related_purchases	past_3_years_bike_related_activity	age	age_group	job_title	job_industry_category	wealth_segment	owns_car	d
5	Low Activity	29	Under 30	Healthcare	Health	Mass Customer	Yes	
61	High Activity	52	Middle-aged	Programmer Analyst IV	Manufacturing	Mass Customer	Yes	
5	Low Activity	29	Under 30	Healthcare	Health	Mass Customer	Yes	
5	Low Activity	29	Under 30	Healthcare	Health	Mass Customer	Yes	
5	Low Activity	29	Under 30	Healthcare	Health	Mass Customer	Yes	
...	...	...	...	...	...	...	...	
33	Moderate Activity	46	Middle-aged	Chemical Engineer	Manufacturing	Mass Customer	Yes	
35	Moderate Activity	26	Under 30	Office Assistant III	Uncategorised	Mass Customer	No	
76	High Activity	52	Middle-aged	Statistician II	Uncategorised	Mass Customer	Yes	
90	Very High Activity	59	Middle-aged	Internal Auditor	Manufacturing	Mass Customer	No	
70	High Activity	41	Middle-aged	Accountant III	Uncategorised	Mass Customer	No	

deceased_indicator	tenure	tenure_group	transaction_date	transaction_month	online_order	order_status	brand	product_line
No	3	New	2017-06-05	Jun	True	Approved	Norco Bicycles	Road
No	7	Regular	2017-05-09	May	False	Approved	Giant Bicycles	Standard
No	3	New	2017-04-23	Apr	False	Approved	Solex	Standard
No	3	New	2017-12-26	Dec	False	Approved	Giant Bicycles	Road
No	3	New	2017-11-07	Nov	True	Approved	OHM Cycles	Standard
...	...	...	...	...	...	...	...	...
No	4	New	2017-05-29	May	False	Approved	Giant Bicycles	Standard
No	3	New	2017-09-22	Sep	False	Approved	Solex	Standard
No	13	Loyal	2017-11-24	Nov	True	Approved	WeareA2B	Standard
No	16	Loyal	2017-07-08	Jul	False	Approved	Norco Bicycles	Road
No	17	Loyal	2017-11-20	Nov	False	Approved	Norco Bicycles	Standard

product_class	product_size	list_price	standard_cost	revenue	product_first_sold_date	transaction_count	transaction_percentage	Rank	customer_value
medium	medium	543.39	407.54	135.85	1999-07-20	14	0.08	1	1
medium	large	1765.30	709.48	1055.82	1991-07-10	14	0.08	1	1
medium	large	202.62	151.96	50.66	1994-08-10	14	0.08	1	1
medium	medium	792.90	594.68	198.22	2015-04-11	14	0.08	1	1
medium	medium	1227.34	770.89	456.45	1994-08-10	14	0.08	1	1
...	...	...	...	...	...	...	...	...	...
medium	large	1635.30	993.66	641.64	2016-03-29	1	0.01	17870	0
medium	large	1071.23	380.74	690.49	1996-04-05	1	0.01	17870	0
medium	medium	1292.84	13.44	1279.40	2015-06-17	1	0.01	17870	0
medium	medium	544.05	376.84	167.21	2005-10-22	1	0.01	17870	0
medium	small	586.45	521.94	64.51	1991-07-10	1	0.01	17870	0

Fig.23 Dataframe after cleaning and preprocessing.

5. Visualization and Findings

At this stage of analysing Sprocket Central Pty Ltd. dataset for the identification of valuable customers, several visualization's methods and techniques were employed in getting a better glimpse of the purchase's behaviours.

5.1 Relationships Between Numerical Features in the Dataset

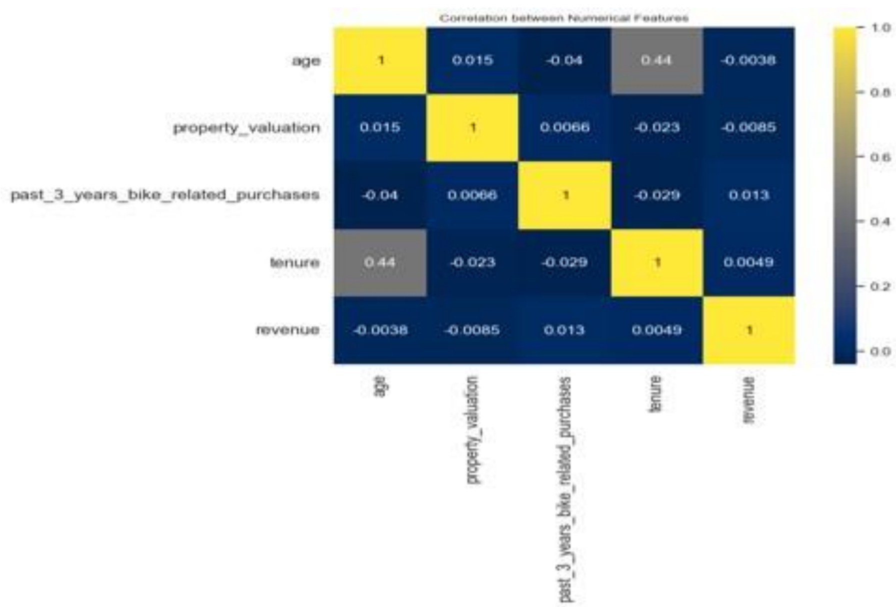


Fig.24 Numerical features heatmap correlation.

The correlation shows the correlations between age, property valuation, past 3 years bikerelated purchases, tenure, and revenue.

The correlation plot revealed a strong positive correlation between age and tenure which suggests that older customers tend to have been with Sprocket for a longer time, property valuation has weak correlations with other features, indicating it doesn't strongly relate to age, bike-related purchases, tenure, or revenue.

It can also be seen from the plot that past 3 years bike-related purchases has weak correlations with all other features, suggesting that the amount spent on bike-related purchases in the past three years does not significantly depend on the other features. Lastly, revenue has weak correlations with all other features, indicating that the revenue generated from customers does not strongly depend on their age, property valuation, bikerelated purchases, or tenure.

5.2 Transactions Count of Numerical Features by States

Visualisation of transaction count of age, property valuation, past 3 years bike-related purchases, and tenure by the three states.

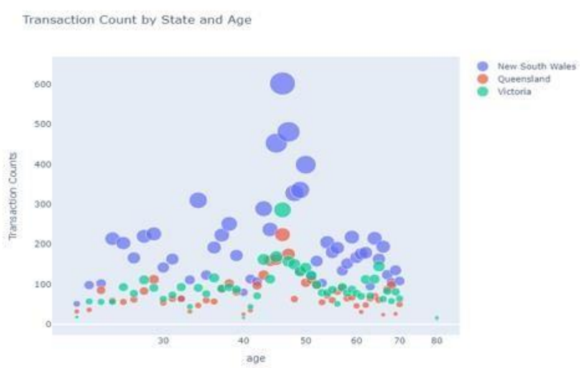


Fig.25 Transaction count by state and age. *Fig.26 Transaction count by state and property valuation.*

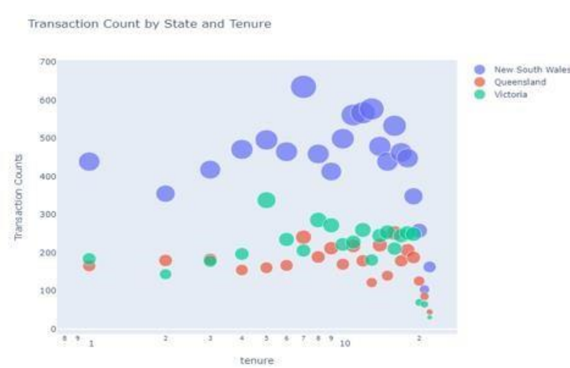


Fig.27 Transaction count by state and bike-related purchases. *Fig.28 Transaction count by state and tenure.*

Property Valuation

In New South Wales, customers with property valuations of 9, 10, and 8 have the highest transaction counts. In Queensland, customers with property valuations of 7, 8, and 3 have the highest transaction counts. In Victoria, customers with property valuations of 8, 7, and 9 have the highest transaction counts.

Age

In New South Wales, customers aged between 45 and 50 have the highest transaction counts. In Queensland, customers aged 46 have the highest transaction counts. In Victoria, the highest transaction counts are also from customers aged 46.

Tenure

In New South Wales, customers with tenures between 20-22 months have the lowest transaction counts, while other tenures show high transaction counts. In Queensland, customers with tenures of 7 and 16 months have the highest transaction counts. In Victoria, customers with tenures of 5, 8, and 9 months have the highest transaction counts.

Higher property valuations are generally associated with higher transaction counts, especially in New South Wales and Victoria. Middle-aged customers (around 46-50) tend to have the highest transaction counts across all states. Newer customers (with shorter tenures) in Queensland and Victoria have higher transaction counts, whereas in New South Wales, long-tenured customers show higher transaction counts except for those with 20-22 months of tenure. The higher the number of past 3 years bike-related purchases, the higher the transaction counts in all three states.

5.3 Transactions Count of Categorical Features by States:

Below are the results of the transactons count of customers gender, job industry category, wealth segment, age group, property valuation group, past 3 years bike related activity, and tenure group.

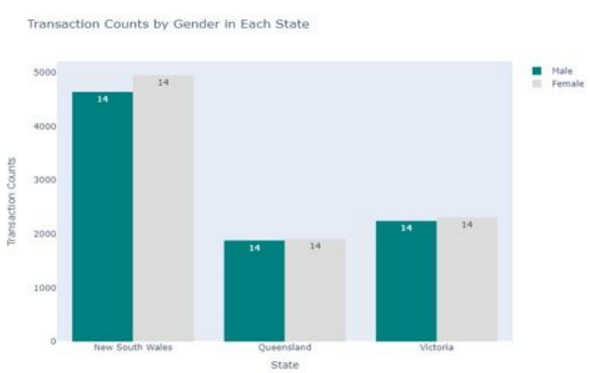


Fig.29 Transaction count by state and gender.

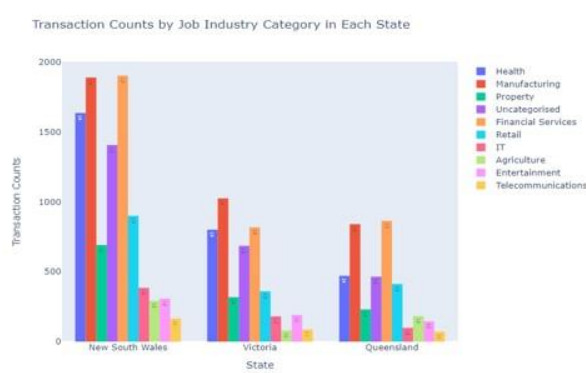


Fig.30 Transaction count by state and job industry category.

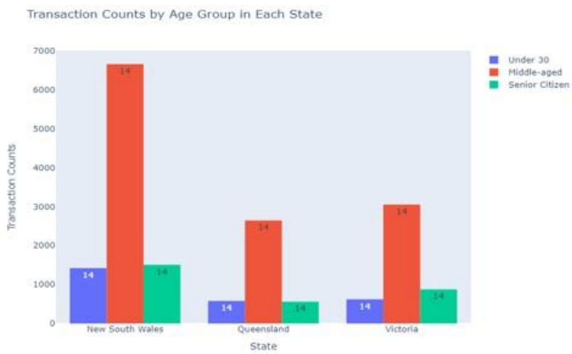


Fig.31 Transaction count by state and age group.

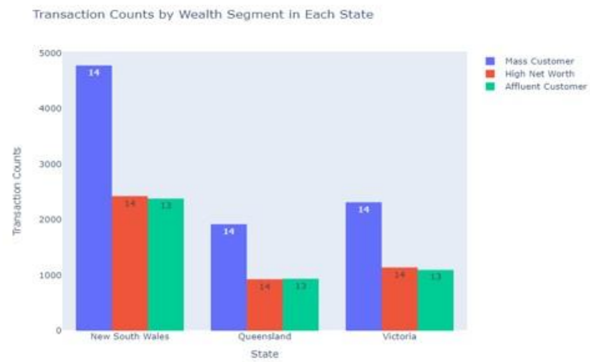


Fig.32 Transaction count by state and wealth segment.

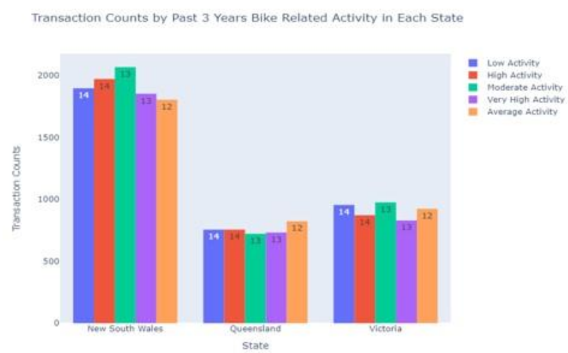


Fig.33 Transaction count by state and bike-related activity.

Fig.34 Transaction count by state and property valuation group.

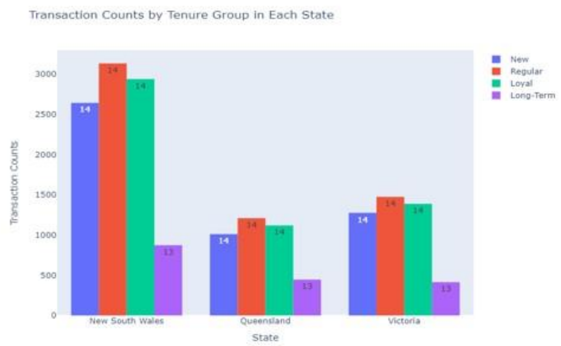


Fig.35 Transaction count by state and tenure group

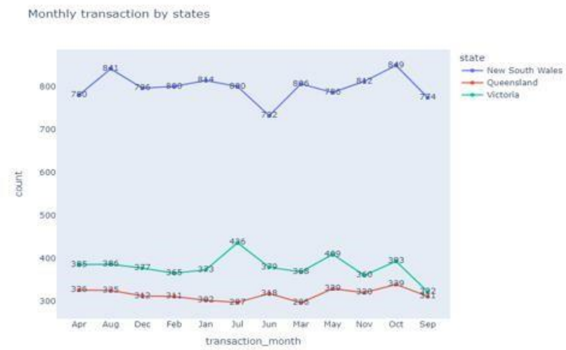


Fig.36 Transaction count by state and month.

Gender

In New South Wales and Queensland, most customers are female, followed by male. In Victoria, the majority of customers are both female and male.

Property Valuation

In New South Wales, customers with high property valuations (9-12) have the most transactions, followed by those with moderate valuations (5-8), and the least transactions come from low valuations (1-4). In Queensland, customers with moderate valuations (5-8) have the most transactions, followed by low valuations (1-4), and the least transactions come from high valuations (9-12). In Victoria, customers with moderate valuations (5-8) have the most transactions, followed by high valuations (9-12), and the least transactions come from low valuations (1-4).

Past 3 Years Bike Related Purchases

In New South Wales, customers with low activities (0-19) have the most transactions, followed by moderate (20-39), high (60-79), average (40-59), and then very high activities (80-99). In Queensland, customers with average activities (40-59) have the most transactions, followed by low (0-19), moderate (20-39), very high (60-79), and then high activities (80-99). In Victoria, customers with moderate activities (20-39) have the most transactions, followed by average (40-59), very high (60-79), low (0-19), and then high activities (80-99).

Age Group

In New South Wales and Queensland, middle-aged customers have the most transactions, followed by under-30, and the least transactions come from senior citizens. In Victoria, middle-aged customers have the most transactions, followed by senior citizens, and the least transactions come from under-30.

Job Industry Category

In New South Wales, the most transactions come from manufacturing, financial services, uncategorized, and health, followed by retail and property. The least transactions come from IT, entertainment, agriculture, and telecommunication. In Queensland, the most transactions come from manufacturing, financial services, uncategorized, and health, followed by retail. The least transactions come from property, IT, entertainment, agriculture, and telecommunication. In Victoria, the most transactions come from manufacturing, financial services, uncategorized, and health, followed by retail. The least transactions come from property, entertainment, IT, agriculture, and telecommunication.

Wealth Segment

In all three states, most customers are mass customers. In New South Wales and Victoria, high net worth customers come next, followed by affluent customers. In Queensland, high net worth customers are followed by affluent customers.

Monthly Transactions

New South Wales consistently has the highest transaction counts, with values fluctuating between 732 and 849. The highest count is in October, and the lowest is in March. Victoria has moderate transaction counts, ranging from 297 to 426. The highest count occurs in July, and the lowest in March. On the other hand, Queensland has the lowest transaction counts, varying between 296 and 326. The counts are relatively stable with minimal fluctuations throughout the year.

General Insights:

Gender: Female customers generally have more transactions than male customers in New South Wales and Queensland.

Job Industry Category: Customers from manufacturing and financial services sectors have the highest transaction counts across all states.

Wealth Segment: Mass customers dominate the transaction counts, followed by high net worth and then affluent customers in all three states.

Age Group: Middle-aged customers (around 46-50) tend to have the highest transaction counts across all states.

Property Valuation Group: Higher property valuations are generally associated with higher transaction counts, especially in New South Wales and Victoria.

Bike-Related Activities: The higher the number of past 3 years bike-related purchases, the higher the transaction counts in all three states.

Tenure Group: Newer customers (with shorter tenures) in Queensland and Victoria have higher transaction counts, whereas in New South Wales, long-tenured customers show higher transaction counts except for those with 20-22 months of tenure.

Monthly Transactions: The monthly transaction plot shows that New South Wales consistently leads with the highest counts, peaking in October and dipping in March, indicating seasonal trends. Victoria has moderate transactions, highest in July and lowest in March. Queensland has the lowest and most stable transaction counts throughout the year. This suggests New South Wales has the most dynamic market, while Queensland experiences steady, minimal fluctuations.

5.4 Transactions Count by Products:

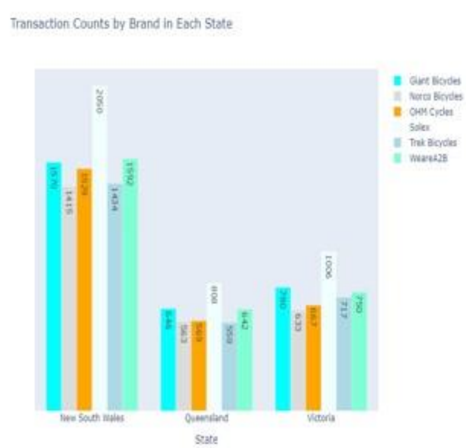


Fig.37 Transaction count by state and month.

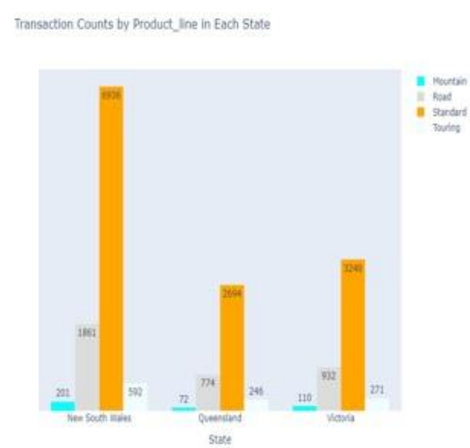


Fig.38 Transaction count by state and product line.

The top three Brands in the three states are the Solex brand, Giant Bicycles brand, and the WeareA2B brands while the Standard line bikes were the most popular among the customers.

The others were recorded low sales. The Mountain bikes line did better than the Touring and Road bikes.

5.5 Product Brands by Customer Demographics:

Fig.39 Transaction count by brand and gender.

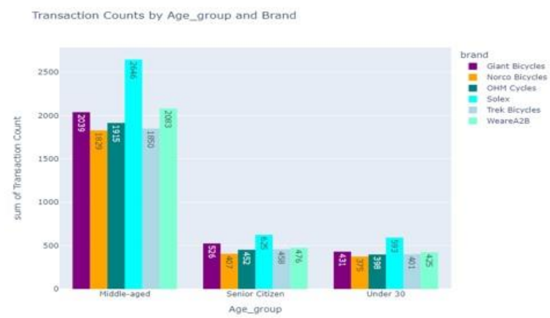


Fig.40 Transaction count by brand and age group.

The brands by gender plot above shows similarity in both male and female customers brand choices as the most popular between the two genders is the Solex brand, followed by the Giant Bicycles and the WeareA2B brand. Likewise, the brands by age group plot also shows that the most preferred brand across the three age groups is the Solex followed by the Giant Bicycles and the WeareA2B brand.

5.6 Customers Value:

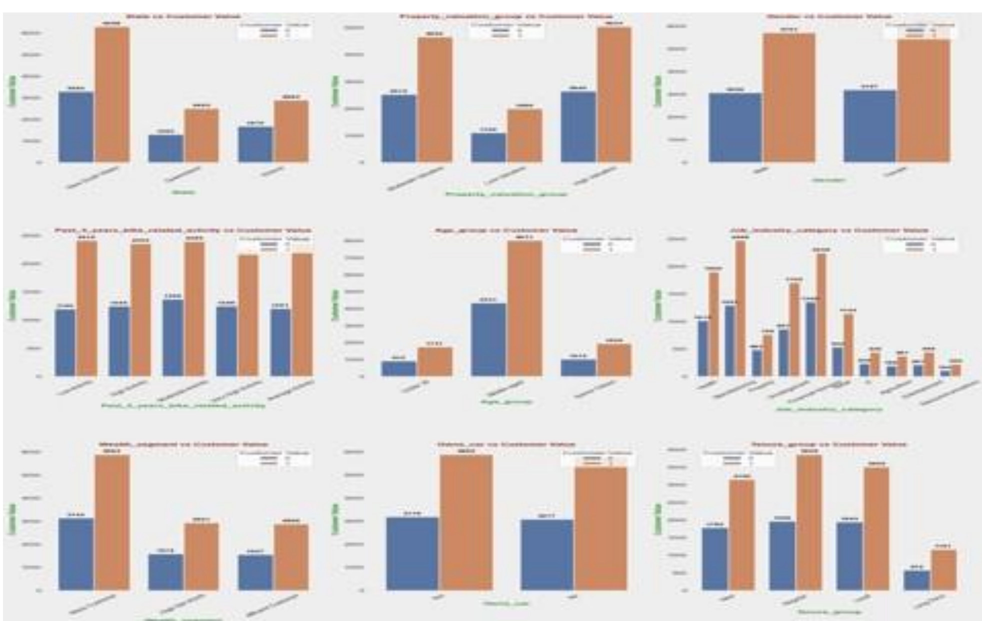


Fig.41 Customer value vs features

It can be interpreted from the plot above that across all the features, the valuable customers are more than the non-valuable customers.

5.7 Revenue Generated by Features

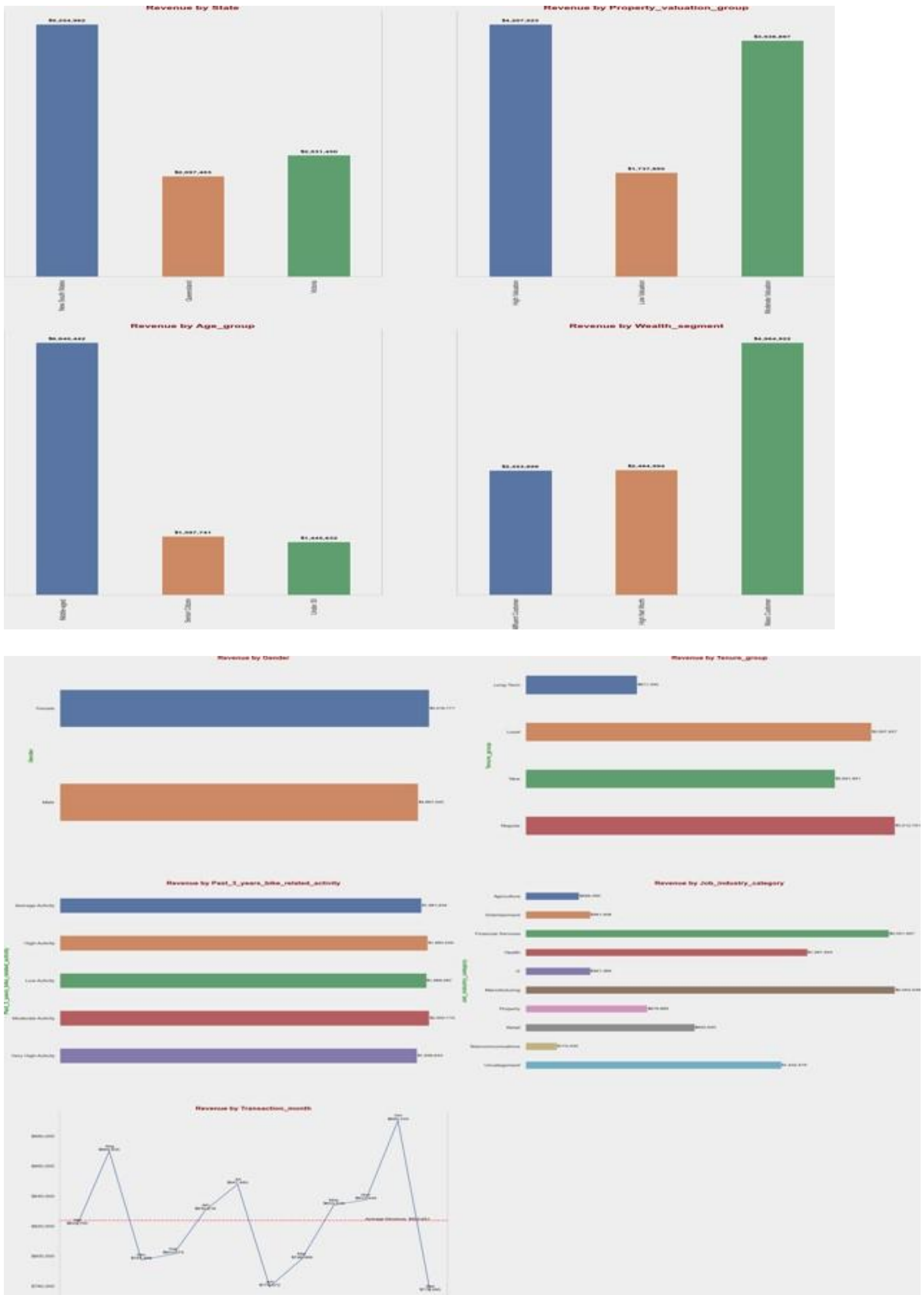


Fig.42 Revenue Generated.

As anticipated, New South Wales exhibited the highest revenue generation among the three states, aligning with its record of the highest transaction volumes. Notably, customers with high and medium property valuations contributed almost identical revenue figures. Furthermore, middle-aged customers emerged as the highest revenue generators, mirroring their status as the segment with the highest transaction frequency. Significantly, the mass customers within the wealth segment yielded the highest generated revenue.

An intriguing observation is that female customers generated the highest revenue, although their male counterparts closely followed. The company recorded substantial revenue from regular customers, trailed by loyal and new customers, respectively. However, long-term customers exhibited the lowest recorded revenue figures.

Remarkably, all past three-year bike-related activity levels yielded high revenue. The most profitable job industry categories in terms of revenue were manufacturing, financial services, and healthcare, while telecommunications, agriculture, entertainment, and IT sectors, respectively, recorded lower revenue contributions.

6. Conclusion

The data visualization analysis for Sprocket Central Pty Ltd. indicates that New South Wales is the most dynamic market, showing the highest transaction and revenue figures. Middle-aged, mass customers, high or medium property valuation segments are particularly valuable. Female customers and those engaged in manufacturing and financial services industries also generate significant revenue.

7. Recommendations

Focus marketing efforts on middle-aged customers, especially females, within the high and medium property valuation segments.

Increase inventory and marketing of the Solex brand and Standard line bikes, as they are the most popular among customers.

Capitalize on the seasonal trends by launching promotions during peak months (e.g., October in New South Wales).

Implement strategies to convert new and regular customers into loyal ones, given that loyal customers show significant revenue potential.

Develop partnerships and targeted campaigns for customers in the manufacturing, financial services, and healthcare sectors.

Implement predictive models like Random Forest, Gradient Boosting, or Neural Networks to forecast future customer value and churn. This will aid in proactive customer retention strategies.

Collection of customer reviews and feedback to provide more insights into customer satisfaction and areas for improvement.

References

Unwin, A. (2020) 'Why Is Data Visualization Important? What Is Important in Data Visualization?', Harvard Data Science Review, 2(1).

Ahmed Malik, W. & Ünlü, A., 2011. Interactive Graphics: Exemplified with Real Data Applications.

Mkhinini Gahar, Rania & Arfaoui, Olfa & Sassi Hidri, Minyar. (2024). Open Research Issues and Tools for Visualization and Big Data Analytics. International Journal of Computing and Digital Systems. 15. 1103-1117. 10.12785/ijcds/150178.

Zia, A., Aziz, M., Popa, I., Ahmed Khan, S., Fazely Hamedani, A., & R. Asif, A., 2022. Artificial Intelligence-Based Medical Data Mining.

Belorkar, Abha, et al. Interactive Data Visualization with Python : Present Your Data As an Effective and Compelling Story, 2nd Edition,

Kluyver, T. et al., 2016. Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides & B. Schmidt, eds. Positioning and Power in Academic Publishing: Players, Agents and Agendas. pp. 87–90.

McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media, Inc.

Oliphant, T. E. (2006). A guide to NumPy. Trelgol Publishing.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90-95.

Waskom, M. (2021). Seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021.

Plotly Technologies Inc. (n.d.). Plotly Python Open Source Graphing Library. https://plotly.com/python/

Plotly Technologies Inc. (n.d.). Plotly Express. https://plotly.com/python/plotly-express/

Plotly Technologies Inc. (n.d.). Dash. https://plotly.com/dash/

Kyrola, A. (2021). Data Visualization with Python: A Practical Introduction. O'Reilly Media, Inc.

Appendix

5/27/24, 12:36 AM assessment - Jupyter Notebook

Importing Libraries

In [1]:
```
import pandas as pd
import warnings #ignore warnings
warnings.filterwarnings('ignore')
```

Importing the Dataset

In [2]:
```
pd.set_option('display.max_columns', None) # Display all the columnsin the
file_path = 'KPMG_VI_New_raw_data_update_final.xlsx' # Declaring the path t
needed_sheets = ['CustomerDemographic', 'CustomerAddress', 'Transactions']
imported_dfs = pd.read_excel(file_path, needed_sheets) # Reading the file p
# Saving each sheet now in pandas dataframe into individual dataframe for e
customer_demographic_df = imported_dfs['CustomerDemographic']
customer_address_df = imported_dfs['CustomerAddress']
transaction_df = imported_dfs['Transactions']
```

localhost:8891/notebooks/Desktop/MyYorkSJ/LDS7004-Data Visualization/Assessment/Raw_Data/assessment.ipynb# 1/90

5/27/24, 12:36 AM assessment - Jupyter Notebook

In [3]:
```
# Merge customer_demographics_df and customer_address_df
merged_df = pd.merge(customer_address_df, customer_demographic_df, on='cust

# Perform the LEFT JOIN with transaction_df
df = pd.merge(merged_df, transaction_df, on='customer_id', how='left')

# Show DataFrame
df
```

Out[3]:

	customer_id	address	postcode	state	country	property_valuation	first_name	la
0	1	060 Morning Avenue	2016.0	New South Wales	Australia	10.0	Laraine	Me
1	1	060 Morning Avenue	2016.0	New South Wales	Australia	10.0	Laraine	Me
2	1	060 Morning Avenue	2016.0	New South Wales	Australia	10.0	Laraine	Me
3	1	060 Morning Avenue	2016.0	New South Wales	Australia	10.0	Laraine	Me
4	1	060 Morning Avenue	2016.0	New South Wales	Australia	10.0	Laraine	Me
...	...	...	...	...	...	...	...	
20499	3996	0 Transport Center	3977.0	VIC	Australia	6.0	Rosalia	
20500	3997	4 Dovetail Crossing	2350.0	NSW	Australia	2.0	Blanch	
20501	3998	736 Roxbury Junction	2540.0	NSW	Australia	6.0	Sarene	
20502	3999	1482 Hauk Trail	3064.0	VIC	Australia	3.0	Patrizius	
20503	4000	57042 Village Green Point	4511.0	QLD	Australia	6.0	Kippy	

20504 rows × 30 columns

In [4]:
```
print(f'DataFrame has {df.shape[0]} rows and {df.shape[1]} columns')
```
```
DataFrame has 20504 rows and 30 columns
```

localhost:8891/notebooks/Desktop/MyYorkSJ/LDS7004-Data Visualization/Assessment/Raw_Data/assessment.ipynb# 2/90

In [5]:
```
# import Numpy for data manipulations
import numpy as np

# For visualizations
import matplotlib.pyplot as plt
import seaborn as sns
```

Let's check view more informatons on the dataset by looking at all the columns name, the data types, and count of null values

In [6]:
```
# Show DataFrame Info
df.info()
```
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20504 entries, 0 to 20503
Data columns (total 30 columns):
 #   Column                               Non-Null Count  Dtype
---  ------                               --------------  -----
 0   customer_id                          20504 non-null  int64
 1   address                              20475 non-null  object
 2   postcode                             20475 non-null  float64
 3   state                                20475 non-null  object
 4   country                              20475 non-null  object
 5   property_valuation                   20475 non-null  float64
 6   first_name                           20504 non-null  object
 7   last_name                            19849 non-null  object
 8   gender                               20504 non-null  object
 9   past_3_years_bike_related_purchases  20504 non-null  int64
 10  DOB                                  20047 non-null  datetime64[ns]
 11  job_title                            18027 non-null  object
 12  job_industry_category                17180 non-null  object
 13  wealth_segment                       20504 non-null  object
 14  deceased_indicator                   20504 non-null  object
 15  default                              19005 non-null  object
 16  owns_car                             20504 non-null  object
 17  tenure                               20047 non-null  float64
 18  transaction_id                       19997 non-null  float64
 19  product_id                           19997 non-null  float64
 20  transaction_date                     19997 non-null  datetime64[ns]
 21  online_order                         19637 non-null  float64
 22  order_status                         19997 non-null  object
 23  brand                                19800 non-null  object
 24  product_line                         19800 non-null  object
 25  product_class                        19800 non-null  object
 26  product_size                         19800 non-null  object
 27  list_price                           19997 non-null  float64
 28  standard_cost                        19800 non-null  float64
 29  product_first_sold_date              19800 non-null  datetime64[ns]
dtypes: datetime64[ns](3), float64(8), int64(2), object(17)
memory usage: 4.8+ MB
```

Data Cleaning and Exploration

We have some columns where the cells are empty. We are going to fix this by exploring individuals columns

Address

In [7]:
```
# Drop rows where address is empty

df.dropna(subset=['address'], inplace=True)

df.shape
```
Out[7]:
```
(20475, 30)
```

Now the number of rows in the dataframe has dropped from 20504 to 20475 showing we have successfully dropped rows where address is empty

Postcode

The postcode column is stored in Float datatype. We are going to convert it into Integer.

In [8]:
```
# Convert the postcode from Object data type to Interger (whole number)

df['postcode'] = df['postcode'].astype(int)

df['postcode'].dtypes
```
Out[8]:
```
dtype('int32')
```

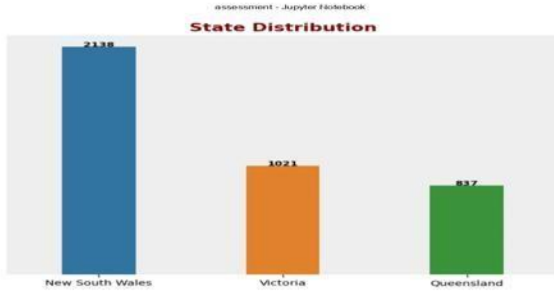
State

In [9]:
```
# Show the count of values in the state column

df['state'].value_counts()
```
Out[9]:
```
NSW                10472
VIC                 4682
QLD                 4356
New South Wales      485
Victoria             480
Name: state, dtype: int64
```

There are disparities in the state column. Fixing it by renaming the abbreviated names in full name.

Fixing the name disparities in State column

In [10]:
```
# Replace the abbreviated name into fullname
df['state'].replace({'NSW': 'New South Wales', 'QLD': 'Queensland', 'VIC':

# To show customer count in each state, we drop the transaction count dupli
unique_customers_df = df.drop_duplicates(subset='customer_id')

# Reset the index
df.reset_index(drop=True, inplace=True)

state_count = unique_customers_df['state'].value_counts()

# Create the seaborn bar plot
fig, ax = plt.subplots(figsize=(8, 6))

sns.barplot(x=state_count.index, y=state_count.values, width=.4)
sns.despine(left=True, bottom=True)

# Annotate the bars with their heights
for i, value in enumerate(state_count.values):
    plt.text(i, value + 0.3, str(value), ha='center', fontsize=9, color='bl

ax.set_facecolor('#eeeeee')
ax.grid(False)
ax.set_yticks([])

# Set labels and title
plt.xlabel(None)
plt.ylabel(None)
plt.title('State Distribution', fontsize=15, color='maroon', fontweight='bo

# Show the plot
plt.show()
print(f'Sprocket Central LTD. have a total {state_count.sum()} customers ir
```


```
Sprocket Central LTD. have a total 3996 customers in the three states. Th
e number of customers in New South Wales is 2138, the number of customers
in Victoria is 1021, and the count of customers in Queensland is 837.
```

Country

In [11]:
```
# Display unique values in country column

df['country'].unique()
```

Out[11]:
```
array(['Australia'], dtype=object)
```

There is nothing to fix in the country column

Property Valuation

Convert the Property valuation data type to Interger from Float, then segment the number of property owns by customers into diferent groups.

In [12]:

```
# Convert the data type to integer
df['property_valuation'] = df['property_valuation'].astype(int)

# Print Property valuation datatype
print(df['property_valuation'].dtypes)
print()
# Print Property Valuation unique values
print(df['property_valuation'].unique())
```

```
int32

[10  9  4 12  8  6  7  3  5 11  1  2]
```

The maximum number of property own by customers is 10 while the minimum is 1. We are going to segmented the property into High Valuation (8-10 properties), Moderate Valuation(5-7 properties), and Low Valuation(1-4 properties)

In [13]:

```
# Create conditional statements to create a new column for property valuati

def update_property(row):
    if row['property_valuation'] <= 4:
        return 'Low Valuation'
    elif row['property_valuation'] <= 8:
        return 'Moderate Valuation'
    else:
        return 'High Valuation'

df['property_valuation_group'] = df.apply(update_property, axis=1)

# To show customer count for each property valuation group, we drop the tr
unique_customers_df = df.drop_duplicates(subset='customer_id')

# Reset the index
df.reset_index(drop=True, inplace=True)

valuation_count = unique_customers_df['property_valuation_group'].value_cou

# Create the seaborn bar plot
fig, ax = plt.subplots(figsize = (8, 6))
sns.barplot(x=valuation_count.index, y=valuation_count.values, width=0.5)
sns.despine(left=True, bottom=True)

# Annotate the bars with their heights
for i, value in enumerate(valuation_count.values):
    plt.text(i, value + 0.3, str(value), ha='center', fontsize=9, color='bl

ax.set_facecolor('#eeeeee')
ax.grid(False)
ax.set_yticks([])

# Set labels and title
plt.xlabel(None)
plt.ylabel(None)
plt.title('Property Valuation Group Distribution', fontsize=15, color='marc

# Show the plot
plt.show()
print(f'The number of {valuation_count.index[0]} customers is {valuation_cc
```


```
The number of High Valuation customers is 1699, Moderate Valuation custom
ers is 1600, and Low Valuation customers is 697
```

Gender

In [14]:

```python
# Show the value count of the customers gender

df['gender'].value_counts()
```

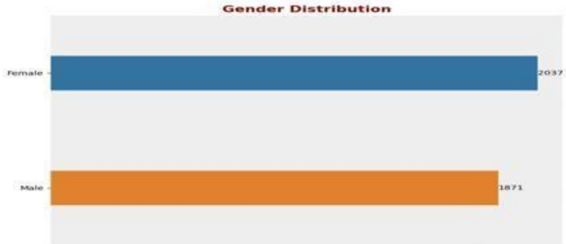
Out[14]:

```
Female    10258
Male       9727
U           466
F            11
Femal         7
M             6
Name: gender, dtype: int64
```

Noticed some disparities in the gender column and we'll fix it my renaming 'F', 'Femal', and 'M' values to represent the actual gender names and drop the 'U' values. Dropping U is essential here because the value is unknown to us and the count is low to have any impact on the dataframe.

In [15]:

```python
# Rename values in the "gender" column
df['gender'].replace({'F': 'Female', 'Femal': 'Female', 'M': 'Male'}, inpla

# Remove rows with 'U' in the "gender" column
df = df[df['gender'] != 'U']

# To show customer count for each gender, we drop the transaction count dup
unique_customers_df = df.drop_duplicates(subset='customer_id')

# Reset the index
df.reset_index(drop=True, inplace=True)

gender_count = unique_customers_df['gender'].value_counts()

# Create the seaborn bar plot
fig, ax = plt.subplots(figsize=(8, 6))

sns.barplot(y=gender_count.index, x=gender_count.values, width=0.3)
sns.despine(left=True, bottom=True)

# Annotate the bars with their values
ax.set_facecolor('#eeeeee')
ax.bar_label(ax.containers[0])
ax.grid(False)
ax.set_xticks([])

# Set labels and title

plt.xlabel('', fontsize=14, color='maroon', fontweight='bold')
plt.ylabel('', fontsize=14, color='maroon', fontweight='bold')
plt.title('Gender Distribution', fontsize=15, color='maroon', fontweight='b

# Show the plot
plt.tight_layout()
plt.show()
print(f'The store has {gender_count.values[0]} {gender_count.index[0]} cust
```




```
The store has 2037 Female customers and 1871 Male customers in the partic
ular year.
```

Past 3 years bike related purchases

In [16]:
```python
df['past_3_years_bike_related_purchases'].describe()
```

Out[16]:
```
count    20009.000000
mean        48.922885
std         28.678292
min          0.000000
25%         24.000000
50%         48.000000
75%         73.000000
max         99.000000
Name: past_3_years_bike_related_purchases, dtype: float64
```

To get the better understanding of this column, we'll create a new column and divide the past 3 years related purchases into different past 3 years bike related activity group.

- Low Activity will represnt bike related purchases less than 20
- Moderate Activity will represnt bike related purchases less than 40
- Average Activity will represnt bike related purchases less than 60
- High Activity will represnt bike related purchases less than 80
- Very High Activity will represnt bike related purchases from 80 and above

In [17]:
```python
def update_past_3_years_activity(row):
    if 0 <= row['past_3_years_bike_related_purchases'] <= 19:
        return 'Low Activity'
    elif 20 <= row['past_3_years_bike_related_purchases'] <= 39:
        return 'Moderate Activity'
    elif 40 <= row['past_3_years_bike_related_purchases'] <= 59:
        return 'Average Activity'
    elif 60 <= row['past_3_years_bike_related_purchases'] <= 79:
        return 'High Activity'
    else:
        return 'Very High Activity'

df['past_3_years_bike_related_activity'] = df.apply(update_past_3_years_act

# Dropping the transaction count duplicates to get the unique number of cu
unique_customers_df = df.drop_duplicates(subset='customer_id')

# Reset the index
df.reset_index(drop=True, inplace=True)

# Calculate customer count for the column
activity_count = unique_customers_df['past_3_years_bike_related_activity'].

# Create the seaborn bar plot
fig, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x=activity_count.values, y=activity_count.index, ax=ax, width=
sns.despine(left=True)

# Annotate the bars with their value counts
ax.set_facecolor('#eeeeee')
ax.bar_label(ax.containers[0])
ax.grid(False)
ax.set_xticks([])

# Set labels and title
plt.xlabel('', fontsize=14, color='maroon', fontweight='bold')
plt.ylabel('', fontsize=14, color='maroon', fontweight='bold')
plt.title('Past Three Years Bike Related Activity Distribution', fontsize=
plt.xticks(rotation=43, fontsize=8)

# Show the plot
plt.tight_layout()
plt.show()
print(f'The number of {activity_count.index[0]} customers is {activity_cour
```


```
The number of Moderate Activity customers is 812, Low Activity customers
is 802, High Activity customers is 785, Very High Activity customers is 7
56, and Average Activity customers is 753
```

DOB

In [18]:

```
# Get description of the Date of Birth DOB column
print(df['DOB'].describe(datetime_is_numeric=True))
```

```
count                               20009
mean     1977-08-16 21:57:30.667199776
min                 1931-10-23 00:00:00
25%                 1968-04-11 00:00:00
50%                 1977-08-28 00:00:00
75%                 1987-03-24 00:00:00
max                 2002-03-11 00:00:00
Name: DOB, dtype: object
```

With the DOB column, we can extract two new columns 'Age' to show the customers actual age and 'Age Group' to show the which group those customers belong to based on their age.

In [19]:

```
#Create a new column 'age' to claculate the age of each customers and anot

from datetime import datetime

df['DOB'] = pd.to_datetime(df['DOB'], format='%Y-%m-%d')

today = datetime.now()
df['age'] = (today - df['DOB']).astype('<m8[Y]')
df['age'] = df['age'].astype(int)

# To show customer age count, we drop the transaction count duplicates to 
unique_customers_df = df.drop_duplicates(subset='customer_id')

# Reset the index
df.reset_index(drop=True, inplace=True)

age_count = unique_customers_df['age'].value_counts()

# Create the seaborn bar plot
sns.set(style="whitegrid")
sns.boxplot(x=df['age'], color='Chartreuse')

# Set labels and title
plt.xlabel('Age', fontsize=14, color='maroon', fontweight='bold')
plt.title('Customer Age Distribution', fontsize=15, color='maroon', fontwei
print(df.age)
```

```
0        70
1        70
2        70
3        70
4        70
         ..
20004    48
20005    48
20006    22
20007    50
20008    32
Name: age, Length: 20009, dtype: int32
```

We have outliers in our age column and we'll try to remove them below.

In [20]:

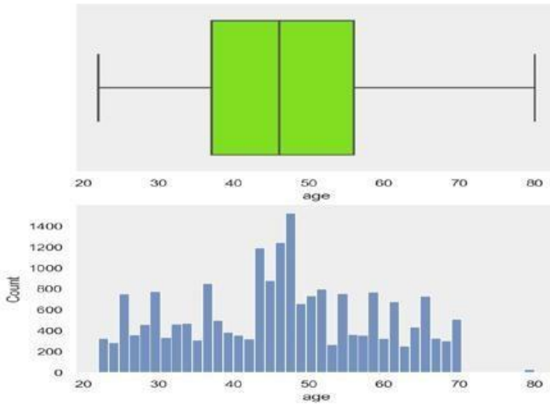
```
# Detecting the upper and lower age bound

q1,q3 = np.percentile(df['age'],[25,75]) # Setting the lower quartile(25) c

IQR = q3-q1 # Calculating the Interquartile range by subtracting q1 from q3

upper = q3+1.5*IQR # Calculating the upper quartile for customer age
lower = q1-1.5*IQR # Calculating the lower quartile for customer age

print("Upper age bound:",upper,"Lower age bound :", lower)
```

```
Upper age bound: 84.5 Lower age bound : 8.5
```


In [21]:

```
# Clipping out all values outside the upper and lower quartiles threshold c
ax, fig = plt.subplots(figsize = (6, 8),dpi=180)

# Set Customers age bound between 17 and 80 years
df.age = df.age.clip(17, 80)

# Age outliers fixed Box plot
ax = plt.subplot(2, 1, 1)
ax.grid(False)
ax.set_facecolor('#eeeeee')

sns.set(style="whitegrid")
sns.boxplot(x=df['age'], color='Chartreuse')
sns.despine(left=True, bottom=True)

# Age distribution plot
ax = plt.subplot(2, 1, 2)
ax.set_facecolor('#eeeeee')
ax.grid(False)

sns.histplot(df.age)
sns.despine(left=True, bottom=True)
print(df.age.describe())
```

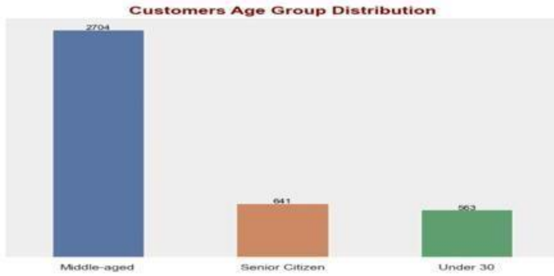
```
count    20009.000000
mean        46.271878
std         12.583144
min         22.000000
25%         37.000000
50%         46.000000
75%         56.000000
max         80.000000
Name: age, dtype: float64
```



To get better understanding of customers age distribution, classifying customers into three different age groups e.g Under 30, Middle-aged, and Senior Citizen will be make this more easier as shown below

In [22]:

```
#create different age groups
def age_groups(age):
    if age <= 30:
        return 'Under 30'
    elif age <= 60:
        return 'Middle-aged'
    else:
        return 'Senior Citizen'

# Create a new column for customers age groups
df['age_group'] = df['age'].apply(age_groups)

# To show customer count for each age group, we drop the transaction count
unique_customers_df = df.drop_duplicates(subset='customer_id')
age_count = unique_customers_df['age_group'].value_counts()

# Create the seaborn bar plot
fig, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x=age_count.index, y=age_count.values, width=0.5)
sns.despine(left=True, bottom=True)

# Annotate the bars with their heights
for i, value in enumerate(age_count.values):
    plt.text(i, value + 0.3, str(value), ha='center', fontsize=9, color='bl

ax.set_facecolor('#eeeeee')
ax.grid(False)
ax.set_yticks([])

# Set labels and title
plt.xlabel('', fontsize=14, color='maroon', fontweight='bold')
plt.ylabel('', fontsize=14, color='maroon', fontweight='bold')
plt.title('Customers Age Group Distribution', fontsize=15, color='maroon',

# Show the plot
plt.show()
print(f'The number of {age_count.index[0]} customers is {age_count.values[6
```




```
The number of Middle-aged customers is 2704, Senior Citizen customers is
641, and Under 30 customers is 563
```

Job Title

In [23]:
```python
print(df['job_title'].value_counts())

print(f'The sum of null values is:', df['job_title'].isnull().sum())
```

```
Social Worker                           225
Business Systems Development Analyst    213
Internal Auditor                        208
Legal Assistant                         208
Nuclear Power Engineer                  205
                                       ...
Systems Administrator IV                 17
Health Coach III                         15
Geologist II                             13
Research Assistant III                   10
Developer I                               7
Name: job_title, Length: 195, dtype: int64
The sum of null values is: 2420
```

The Job Title column have 2,420 null values. Dropping the cells will likely have effect on the Dataframe so fixing it using the Job Industry Category column.

In [24]:
```python
# Checking the unique values in Job Industry Category

df['job_industry_category'].unique()
```

Out[24]:
```
array(['Health', 'Financial Services', 'IT', nan, 'Retail', 'Argicultur
e',
       'Property', 'Manufacturing', 'Telecommunications', 'Entertainmen
t'],
      dtype=object)
```

Above are all the unique values of job industry category including the NAN.

Replacing NAN with "Unemployed" where both Job Title and Job Industry Category are NULL on the same row.

Where Job Title is NULL but Job Industry Category IS NOT NULL, replace the NULLs in Job Title with the corresponding Job Industry Category.

In [25]:
```python
# Correct the spelling mistake in 'Argiculture' and the NAN values in Job I
df['job_industry_category'].replace({'Argiculture': 'Agriculture'}, inplace

# Replacing Job Title's NULL values with the Job Industry Category
def update_job_title(row):
    if pd.isna(row['job_title']):
        if row['job_industry_category'] == 'Agriculture':
            return 'Agriculturist'
        elif row['job_industry_category'] == 'Entertainment':
            return 'Entertainer'
        elif row['job_industry_category'] == 'Financial Services':
            return 'Financial Officer'
        elif row['job_industry_category'] == 'Health':
            return 'Healthcare'
        elif row['job_industry_category'] == 'IT':
            return 'IT Specialist'
        elif row['job_industry_category'] == 'Manufacturing':
            return 'Production Worker'
        elif row['job_industry_category'] == 'Property':
            return 'Property Agent'
        elif row['job_industry_category'] == 'Retail':
            return 'Retailer'
        elif row['job_industry_category'] == 'Telecommunications':
            return 'Telecommunications Worker'
        else:
            return np.nan
    else:
        return row['job_title']


df.loc[:, 'job_title'] = df.apply(update_job_title, axis=1)
# Drop NAN
df.dropna(subset=['job_title'], inplace=True)
```

In [26]:

```
import squarify

# To show customer count for each property valuation group, drop the transa
unique_customers_df = df.drop_duplicates(subset='customer_id')

# Calculate the count of the job titles, show only the top 30
job_title_count = unique_customers_df['job_title'].value_counts().head(30)

# Create labels
labels = [f'{job}\n{count}' for job, count in zip(job_title_count.index, jo

# Plot the treemap
plt.figure(figsize=(20, 16))
squarify.plot(sizes=job_title_count, label=labels,
              color = sns.color_palette("mako", len(job_title_count)),
              text_kwargs={'fontsize': 12, 'color': '#ffffff', 'wrap': True


# Turn off axis
plt.axis("off")

# Show the plot
plt.show()
```

Job Industry Categories

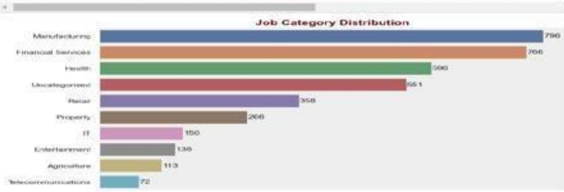
When Job Title value is not empty and Job Industry Category is not known or is empty, replace the latter to "Uncategorised".

In [27]:

```
# Replace Not Applicable cells and NULL cells in the column to Uncategorise
def update_job_category(row):
    if row['job_title'] != '' and pd.isna(row['job_industry_category']): #
        return 'Uncategorised'
    elif row['job_title'] != '' and row['job_industry_category'] == 'n/a':
        return 'Uncategorised'
    else:
        return row['job_industry_category']

# Apply to the DataFrame
df.loc[:, 'job_industry_category'] = df.apply(update_job_category, axis=1)
```

In [28]:

```
# To show customer count for each job industry category, we drop the transc
unique_customers_df = df.drop_duplicates(subset='customer_id')

# Reset the index
df.reset_index(drop=True, inplace=True)

# Calculate the customer count for job industry category
job_category_count = unique_customers_df['job_industry_category'].value_cou

# Create the seaborn bar plot
fig, ax = plt.subplots(figsize=(10, 6))

sns.barplot(x=job_category_count.values, y=job_category_count.index, )
sns.despine(left=True, bottom=True)

# Add text label to the plot
ax.set_facecolor('#eeeeee')
ax.bar_label(ax.containers[0])
ax.grid(False)
ax.set_xticks([])

# Set labels and title
plt.xlabel('', fontsize=14, color='maroon', fontweight='bold')
plt.ylabel('', fontsize=14, color='maroon', fontweight='bold')
plt.title('Job Category Distribution', fontsize=15, color='maroon', fontwei

plt.show()
print(unique_customers_df['job_industry_category'].value_counts())
```




```
Manufacturing         796
Financial Services    766
Health                596
Uncategorised         551
Retail                358
Property              266
IT                    150
Entertainment         136
Agriculture           113
Telecommunications     72
Name: job_industry_category, dtype: int64
```

Most of the store's customers are from the Manufacturing job industry, followed the Financial Services job category while the Telecommunications job industry category has the lowest customer counts at the store.

Wealth Segment

In [29]:

```
df['wealth_segment'].unique()
```

Out[29]:

```
array(['Mass Customer', 'Affluent Customer', 'High Net Worth'],
      dtype=object)
```

In [30]:
```
# To show customer count for each wealth segment, we drop the transaction c
unique_customers_df = df.drop_duplicates(subset='customer_id')

wealth_count = unique_customers_df['wealth_segment'].value_counts()

# Create the seaborn bar plot
fig, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x=wealth_count.index, y=wealth_count.values, width=.4)
sns.despine(left=True, bottom=True)

# Annotate the bars with their heights
for i, value in enumerate(wealth_count.values):
    plt.text(i, value + 0.3, str(value), ha='center', fontsize=9, color='bl

ax.set_facecolor('#eeeeee')
ax.grid(False)
ax.set_yticks([])

# Set labels and title
plt.xlabel(None)
plt.ylabel(None)
plt.title('Customers Wealth Segment Distribution', fontsize=15, color='marc

plt.show()

print(f'The number of {wealth_count.index[0]} customers is {wealth_count.va
```


```
The number of Mass Customer customers is 1901, High Net Worth customers i
s 963, and Affluent Customer customers is 940
```

Deceased Indicator

In [31]:
```
df['deceased indicator'].unique()
```

Out[31]:
```
array(['N', 'Y'], dtype=object)
```

In [32]:

```
# Replace the 'N' values in Customers Deceased Indicator column to 'No' and
df['deceased_indicator'].replace({'N': 'No', 'Y': 'Yes'}, inplace=True)

# To show customer count for each deceased indicator, we drop the transacti
unique_customers_df = df.drop_duplicates(subset='customer_id')
indicator_count = unique_customers_df['deceased_indicator'].value_counts()

# Create the seaborn bar plot
fig, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x=indicator_count.index, y=indicator_count.values, width=0.3)
sns.despine(left=True, bottom=True)

# Annotate the bars with their heights
for i, value in enumerate(indicator_count.values):
    plt.text(i, value + 0.3, str(value), ha='center', fontsize=9, color='bl

ax.set_facecolor('#eeeeee')
ax.grid(False)
ax.set_yticks([])

# Set labels and title
plt.xlabel(None)
plt.ylabel(None)
plt.title('Customers Deceased Indicator Distribution', fontsize=15, color='

plt.show()

print(f'{indicator_count.values[0]} customers has thier deceased indicator
```



```
3802 customers has thier deceased indicator as No, and 2 customers has th
eir deceased indicator as Yes. This shows that most of the customers are
not dead.
```

Default

We can't make sense of what the Default is about. We'll drop the entire column

In [33]:

```
df = df.drop(columns=['default'])
```

Owns Car

In [34]:

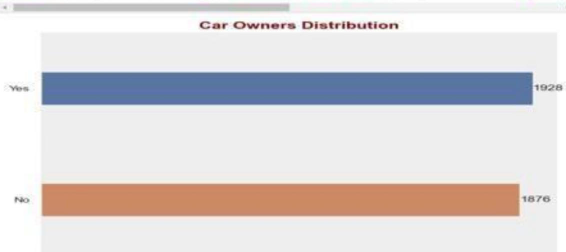
```
df['owns car'].unique()
```

Out[34]:

```
array(['Yes', 'No'], dtype=object)
```

In [35]:

```
# Drop the transaction count duplicates to get the unique number of custome
unique_customers_df = df.drop_duplicates(subset='customer_id')

# Get car count
car_count = unique_customers_df['owns_car'].value_counts()

# Create the seaborn bar plot
fig, ax = plt.subplots(figsize=(8, 6))
sns.barplot(y=car_count.index, x=car_count.values, width=0.3)
sns.despine(left=True, bottom=True)

# Annotate the plot
ax.set_facecolor('#eeeeee')
ax.bar_label(ax.containers[0])
ax.grid(False)
ax.set_xticks([])

# Set labels and title
plt.xlabel('', fontsize=14, color='maroon', fontweight='bold')
plt.ylabel('', fontsize=14, color='maroon', fontweight='bold')
plt.title('Car Owners Distribution', fontsize=15, color='maroon', fontweigh

plt.show()

print(f'{car_count.values[0]} customers owns car ({car_count.index[0]}), ar
```




```
1928 customers owns car (Yes), and 1876 customers do not own car (No)
```


Tenure

In [36]:

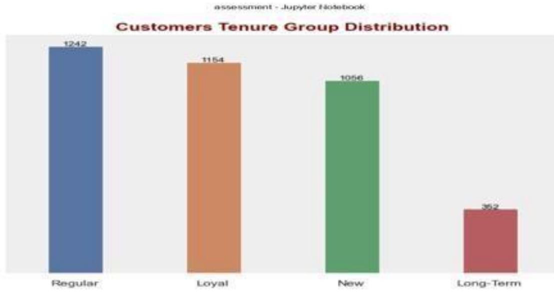
```
df['tenure'].describe()
```

Out[36]:

```
count    19506.000000
mean        10.694248
std          5.677682
min          1.000000
25%          6.000000
50%         11.000000
75%         16.000000
max         22.000000
Name: tenure, dtype: float64
```

We first convert the column from decimal to whole number, then we create a new column to split the tenure into groups

In [37]:
```
# Convert Tenure column from Float to Integer
df['tenure'] = df['tenure'].astype(int)

def update_tenure(row):
    if row['tenure'] <= 6:
        return 'New'
    elif row['tenure'] <= 12:
        return 'Regular'
    elif row['tenure'] <= 18:
        return 'Loyal'
    else:
        return 'Long-Term'

# Add the new column to the DataFrame
df['tenure_group'] = df.apply(update_tenure, axis=1)

# Drop the transaction count duplicates to get the unique number of custome
unique_customers_df = df.drop_duplicates(subset='customer_id')

# Cuustomer counts by tenure
tenure_count = unique_customers_df['tenure_group'].value_counts()

# Create the seaborn bar plot
fig, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x=tenure_count.index, y=tenure_count.values, width=0.4)
sns.despine(left=True, bottom=True)

# Annotate the bars with their heights
for i, value in enumerate(tenure_count.values):
    plt.text(i, value + 0.3, str(value), ha='center', fontsize=9, color='bl

ax.set_facecolor('#eeeeee')
ax.grid(False)
ax.set_yticks([])

# Set labels and title
plt.xlabel(None)
plt.ylabel(None)
plt.title('Customers Tenure Group Distribution', fontsize=15, color='maroor

plt.show()

print(f'The number of {tenure_count.index[0]} customers is {tenure_count.va
```


```
The number of Regular customers is 1242, Loyal customers is 1154, New cus
tomers is 1056, and Long-Term customers is 352
```

Transaction ID

In [38]:
```
df['transaction_id'].isnull().sum()
```

Out[38]: 479

Drop rows where Transaction ID is null and convert to Integer datatype

In [39]:
```
# Drop all rows where Transaction ID is null
df = df.dropna(subset=['transaction_id'])

df.shape
```

Out[39]: (19027, 34)

In [40]:
```
# Convert the Transaction ID column from float to integer
df['transaction_id'] = df['transaction_id'].astype(int)
```

Product ID

In [41]:
```
df['product_id'] = df['product_id'].astype(int)

df['product_id'].isnull().sum()
```
Out[41]: 0

Transaction Date

In [42]:
```
df['transaction_date'].dtype
```
Out[42]: dtype('<M8[ns]')

Create a new column for the transaction month extraction

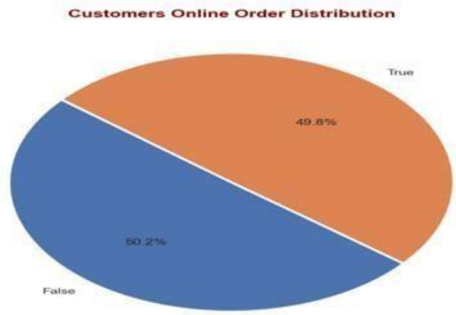
In [43]:
```
# Create new column Transaction Month
df['transaction_month'] = df['transaction_date'].dt.strftime('%b')

# Count the number of tranactions in each month
transaction_month = df['transaction_month'].value_counts().sort_index()

# Create the seaborn bar plot
fig, ax = plt.subplots(figsize=(10, 6))
sns.lineplot(y=transaction_month.values, x=transaction_month.index)
sns.despine(left=True, bottom=True)

# Annotate the bars with their heights
for i, value in enumerate(transaction_month.values):
    plt.text(i, value + 0.3, str(value), ha='center', fontsize=9, color='bl

ax.set_facecolor('#eeeeee')
ax.grid(False)
ax.set_yticks([])

# Set labels and title
plt.xlabel(None)
plt.ylabel(None)
plt.title('Transaction Month Distribution', fontsize=15, color='maroon', fc

plt.show()

#print(f'The number of {tenure_count.index[0]} customers is {tenure_count.\
```

The plot above show that all the month have close number of transactions in the given year with October being the most busy month at the store in the year and in the month before it, September low transactions were recorded.

Online Order

In [44]:
```
df['online_order'].unique()
```
Out[44]:
```
array([ 0.,  1., nan])
```


In [45]:
```
# Drop the NAN values in the column
df = df.dropna(subset=['online_order'])

# Change the value in the column to True and False after converting the dat
#df['online_order'] = df['online_order'].astype(int)

df['online_order'].replace({1: 'True', 0: 'False'}, inplace=True)


# Drop the transaction count duplicates to get the unique number of custome
unique_customers_df = df.drop_duplicates(subset='customer_id')

# Get car count
order_counts = unique_customers_df['online_order'].value_counts()

# Plot the pie chart
explode = (0.01, 0)  # Explode the first slice (owns car) by 0.1

plt.figure(figsize=(8, 8))
plt.pie(order_counts, labels=order_counts.index, autopct='%1.1f%%', explode

# Set title
plt.title('Customers Online Order Distribution', fontsize=15, color='maroon

plt.show()

print(f' The number of customers that use online order ({order_counts.index
```




```
 The number of customers that use online order (True) are 1654, while the
number of customers that does not use online order (False) are 1670.
```

Order Status

In [46]:
```
df['order_status'].unique()
```

Out[46]:
```
array(['Approved', 'Cancelled'], dtype=object)
```


In [47]:
```
# To show customer count for each order status, we drop the transaction cou
unique_customers_df = df.drop_duplicates(subset='customer_id')

# Count of order status
status_count = df['order_status'].value_counts()

# Drop the transaction count duplicates to get the unique number of custome
unique_customers_df = df.drop_duplicates(subset='customer_id')

# Get car count
status_count = unique_customers_df['order_status'].value_counts()

# Plot the pie chart
explode = (0, 0.1)  # Explode the first slice (owns car) by 0.1

plt.figure(figsize=(8, 8))
plt.pie(status_count, labels=status_count.index, autopct='%1.1f%%', explode

# Set title
plt.title('Customers Order status Distribution', fontsize=15, color='maroon

plt.show()

print(f' {status_count.values[0]} customers has thier order {status_count.i
```


```
 3294 customers has thier order Approved , while 30 customers has thier o
rder Cancelled
```

The plot above shows 99% of customers that made their order online successfully have their order approved.

Brand

In [48]:
```
df['brand'].unique()
```

Out[48]:
```
array(['OHM Cycles', 'Solex', 'Trek Bicycles', 'Norco Bicycles',
       'Giant Bicycles', 'WeareA2B', nan], dtype=object)
```

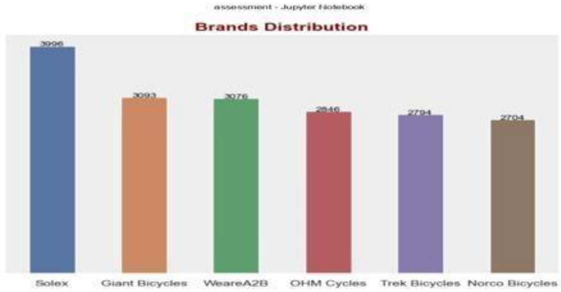
The Brand column seven distinct values including the NANs. Checking the number of NANs in the column will help in deciding removing all NANs or replacing it.

In [49]:
```
brand = df['brand']

print(brand.value_counts())

print(f'There are {brand.isnull().sum()} NULLs values in the column')
```

```
Solex             3996
Giant Bicycles    3093
WeareA2B          3076
OHM Cycles        2846
Trek Bicycles     2794
Norco Bicycles    2704
Name: brand, dtype: int64
There are 177 NULLs values in the column
```

There are 184 NULLs value in the column. Dropping them won't affect the structure of the dataframe.

In [50]:
```
# Drop the NAN values in the column
df = df.dropna(subset=['brand'])

# To show customer count for each brands, we drop the transaction count dup
unique_customers_df = df.drop_duplicates(subset='customer_id')

brand_count = df['brand'].value_counts()

# Create the seaborn bar plot
fig, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x=brand_count.index, y=brand_count.values, width=.5)
sns.despine(left=True, bottom=True)

# Annotate the bars with their heights
for i, value in enumerate(brand_count.values):
    plt.text(i, value + 0.3, str(value), ha='center', fontsize=9, color='bl

ax.set_facecolor('#eeeeee')
ax.grid(False)
ax.set_yticks([])

# Set labels and title
plt.xlabel(None)
plt.ylabel(None)
plt.title('Brands Distribution', fontsize=15, color='maroon', fontweight='b

plt.show()

#print(f' The number of customers that use online order ({order_count.inde
```

The chart above shows that the Solex brand is the best selling brand in the store and the Norco Bicycles have the lowest purchases.

Product Line

In [51]:
```
df['product line'].unique()
```

Out[51]: `array(['Standard', 'Road', 'Mountain', 'Touring'], dtype=object)`

In [52]:
```
# To show customer count for each product line, we drop the transaction cou
unique_customers_df = df.drop_duplicates(subset='customer_id')

pro_line_count = df['product_line'].value_counts()

# Create the seaborn bar plot
fig, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x=pro_line_count.index, y=pro_line_count.values, width=.5)
sns.despine(left=True, bottom=True)

# Annotate the bars with their heights
for i, value in enumerate(pro_line_count.values):
    plt.text(i, value + 0.3, str(value), ha='center', fontsize=9, color='bl

ax.set_facecolor('#eeeeee')
ax.grid(False)
ax.set_yticks([])

# Set labels and title
plt.grid(False)
plt.xlabel(None)
plt.ylabel(None)
plt.title('Product Line Distribution', fontsize=15, color='maroon', fontwei

plt.show()
```

From the above, it can be seen that the standard bikes were the most popular among the customers. The road, touring, and mountain bikes respectfully were least popular with the customers.

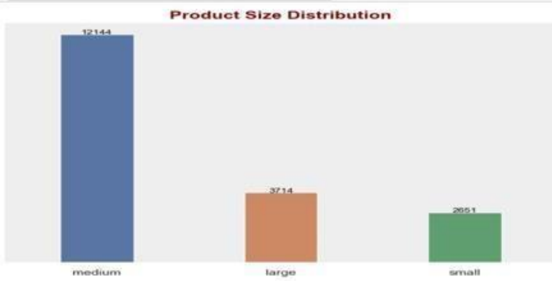
Product Size

In [53]:
```
df['product_size'].unique()
```

Out[53]:
```
array(['medium', 'small', 'large'], dtype=object)
```

In [54]:

```
# To show customer count for each product size, we drop the transaction cou
unique_customers_df = df.drop_duplicates(subset='customer_id')

pro_size_count = df['product_size'].value_counts()

# Create the seaborn bar plot
fig, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x=pro_size_count.index, y=pro_size_count.values, width=.4)
sns.despine(left=True, bottom=True)

# Annotate the bars with their heights
for i, value in enumerate(pro_size_count.values):
    plt.text(i, value + 0.3, str(value), ha='center', fontsize=9, color='bl

ax.set_facecolor('#eeeeee')
ax.grid(False)
ax.set_yticks([])

# Set labels and title
plt.xlabel(None)
plt.ylabel(None)
plt.title('Product Size Distribution', fontsize=15, color='maroon', fontwei

plt.show()
```



Medium size bikes were more purchased by the customers. Large and small bike have low purchase counts by the customers.

List Price

In [55]:

```
list_price = df['list_price']

print(list_price.describe())

print("Null Count:", list_price.isnull().sum())

print(f"The maximum product list price is ${list_price.max()},  minimum is
```

```
count    18509.000000
mean      1106.512786
std        582.443256
min         12.010000
25%        575.270000
50%       1163.890000
75%       1577.530000
max       2091.470000
Name: list_price, dtype: float64
Null Count: 0
The maximum product list price is $2091.47,  minimum is $12.01, and the a
verage list price is $1106.51
```

Standard Cost

In [56]:

```
std_cost = df['standard_cost']

print(std_cost.describe())

print("Null Count:", std_cost.isnull().sum())
print()
print(f"The maximum product standard cost is ${std_cost.max()},  minimum is
```

```
count    18509.000000
mean       556.098845
std        405.715763
min          7.210000
25%        215.140000
50%        507.580000
75%        795.100000
max       1759.850000
Name: standard_cost, dtype: float64
Null Count: 0

The maximum product standard cost is $1759.85,  minimum is $7.21, and the
average standard cost is $556.1
```

Product First Sold Date

Extract new column "Year" from the Product First Sold Date column to have more spefic view on product historical sales by year

In [57]:
```
#Convert column to current date
df['product first sold date'] = pd.to datetime(df['product first sold date'
```

Calculation for Revenue Gained

In [58]:
```
df['revenue'] = df['list_price'] - df['standard_cost']

df['revenue'].describe()
```

Out[58]:
```
count    18509.000000
mean       550.413941
std        493.160645
min          4.800000
25%        133.780000
50%        445.210000
75%        827.160000
max       1702.550000
Name: revenue, dtype: float64
```

The chart above shows most of the products were first sold to the customers in 2015.

Calculation of Customers Rank and Value

In [59]:
```
# Calculate Transaction Count
df['transaction_count'] = df.groupby(['customer_id'])['transaction_id'].tra

# Group by and aggregate the data
df = df.groupby(['customer_id', 'first_name', 'last_name', 'transaction_id'
                          'property_valuation','property_valuation_group','ge
                          'past_3_years_bike_related_activity', 'age', 'age_g
                          'owns_car', 'deceased_indicator', 'tenure', 'tenure
                          'order_status','brand', 'product_line', 'product_cl
                          'standard_cost','revenue','product_first_sold_date'
                              transaction_count=('transaction_count', 'sum'))

# Sort by transaction_count in descending order
df = df.sort_values(by='transaction_count', ascending=False)

# Calculate transaction_percentage
transaction_count = df['transaction_count'].count()
df['transaction_percentage'] = (df['transaction_count'] * 100.0 / transacti
                                                          ).round(2)

# Calculate Rank
df['Rank'] = df['transaction_count'].rank(ascending=False, method='min').as

# Calculate Value based on a threshold (mean transaction_count)
transaction_count_threshold = 0.40 * df['transaction_count'].max()
df['customer_value'] = (df['transaction_count'] > transaction_count_thresho

# Display the result
print(f'Transaction Count Threshold: {transaction_count_threshold:.2f}. \n
print(df['customer_value'].value_counts())
df
```

```
Transaction Count Threshold: 5.60.
 The threshold is set at 40% of the highest transaction
1    11674
0     6255
Name: customer_value, dtype: int64
```

Out[59]:

	customer_id	first_name	last_name	transaction_id	product_id	state	property
5564	1068	Frazer	Searston	11472	29	New South Wales	
11303	2183	Jillie	Fyndon	134	78	Queensland	
5559	1068	Frazer	Searston	4038	97	New South Wales	
5560	1068	Frazer	Searston	4317	69	New South Wales	
5561	1068	Frazer	Searston	4437	99	New South Wales	
...	...	...	...	...	...	...	
7166	1387	Natalee	Comport	3351	94	Victoria	
9677	1865	Isabelita	Kitchener	7405	21	Victoria	
12118	2352	Cilka	Dabbes	14554	54	Victoria	
14833	2863	Alisander	Fetherstone	2375	67	New South Wales	
9972	1921	Cybill	Wakes	14249	81	New South Wales	

17929 rows × 36 columns

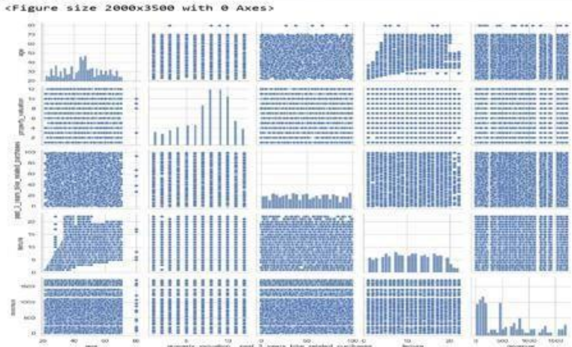
In [60]:
```
df.to csv('cleaned and processed Sprocket.csv')
```

Data Visualization

Using Pairplot to visualize relationship between the numerical values in each state.

In [61]:
```
# Select the numerical features to plot
plt.figure(figsize=(20, 35))
sns.pairplot(df,
             vars= ['age', 'property_valuation', 'past_3_years_bike_relatec
             palette='husl');
plt.show()
```

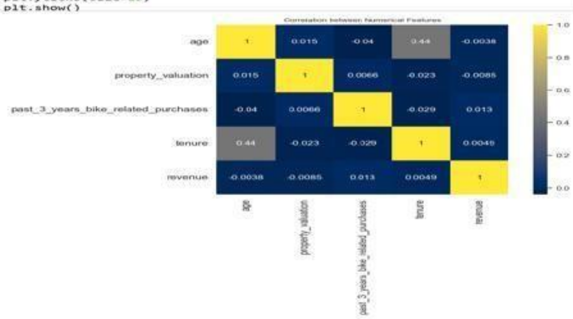
<Figure size 2000x3500 with 0 Axes>



The pairplot above shows the relationship between the numerical features and the distribution of each feature.

For Heatmap:

In [62]:
```
column_to_plot = ['age', 'property_valuation', 'past_3_years_bike_related_p

plt.figure(figsize = (10, 6))
sns.heatmap(df[column_to_plot].corr(), annot=True, square=True, cmap='civic
plt.title("Correlation between Numerical Features", size=10)
plt.xticks(size=13)
plt.yticks(size=13)
plt.show()
```



The heatmap shows that the only correlation that exists between the numerical features is the one between the customers age and revenue the time(tenure) they have spent with the store. While there's no correlation between age and revenue and age and property valuation.

Features Transaction Count by States

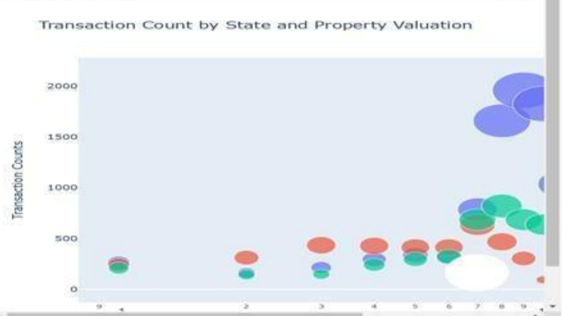
Visualizing the customers features transaction counts by state since the customers are all in three different states.

In [63]:
```
import dash
from dash import html, dcc, Output, Input
import plotly.graph_objects as go
import plotly.io as pio
import pandas as pd

# Sample DataFrame
# df = pd.read_csv('your_dataframe.csv')

app = dash.Dash(__name__)

app.layout = html.Div([
    html.H4("Property Valuation"),

    dcc.Dropdown(
        id='dropdown',
        options=[
            {'label': 'Property Valuation', 'value': 'property_valuation'},
            {'label': 'Age', 'value': 'age'},
            {'label': 'Tenure', 'value': 'tenure'},
            {'label': 'Past 3 Years Bike Related Purchases', 'value': 'past
        ],
        value='property_valuation'
    ),

    dcc.Graph(id='mixed-chart'),
])

@app.callback(
    Output('mixed-chart', 'figure'),
    Input('dropdown', 'value')
)
def update_mixed_chart(selected_dropdown):
    # Group by state and attribute, and count the number of transactions in
    valuation_counts = df.groupby(['state', 'property_valuation']).size().r
    age_counts = df.groupby(['state', 'age']).size().reset_index(name='tran
    tenure_counts = df.groupby(['state', 'tenure']).size().reset_index(name
    bike_related_counts = df.groupby(['state', 'past_3_years_bike_related_p

    if selected_dropdown == 'property_valuation':
        data = valuation_counts
        x = 'property_valuation'
        title = 'Transaction Count by State and Property Valuation'
    elif selected_dropdown == 'age':
        data = age_counts
        x = 'age'
        title = 'Transaction Count by State and Age'
    elif selected_dropdown == 'tenure':
        data = tenure_counts
        x = 'tenure'
        title = 'Transaction Count by State and Tenure'
    elif selected_dropdown == 'past_3_years_bike_related_purchases':
        data = bike_related_counts
        x = 'past_3_years_bike_related_purchases'
        title = 'Transaction Count by State and Past 3 Years Bike Related P

    fig = go.Figure()
```

```
    for state in data['state'].unique():
        state_data = data[data['state'] == state]
        fig.add_trace(go.Scatter(
            x=state_data[x],
            y=state_data['transaction_count'],
            mode='markers',
            marker=dict(size=state_data['transaction_count'], sizemode='ar
            name=state,
            hovertext=state_data['transaction_count']
        ))

    fig.update_layout(
        title=title,
        xaxis_title=x,
        yaxis_title='Transaction Counts',
        xaxis=dict(type='log', showgrid=False, visible=True, showticklabel
        yaxis=dict(showgrid=False, visible=True, showticklabels=True),
        width=800,
        height=600
    )

    # Save the plot to a file
   # file_name = f"{selected_dropdown}_plot.png"
    #pio.write_image(fig, file_name)

    return fig

if __name__ == '__main__':
    app.run_server(debug=True, port=8091)
```


Property Valuation

Property Valuation



Property Valuation:

```
    New South Wales:
    Customers with property valuation of 9, 10, and 8 respectively hav
e the highest transaction count.

    Queensland:
     Customers with property valuation of 7, 8, and 3 respectively hav
e the highest transaction count.

    Victoria:
     Customers with property valuation of 8, 7, and 9 respectively has
the highest transaction count.
```

Age:

```
    New South Wales:
    Customers aged between 45 - 50 have the highest transaction count.

    Queensland:
     Customers 46 have the highest transaction count.

    Victoria:
     Same as Queensland, customers aged 46 have the highest transactio
n count.
```

Tenure:

```
    New South Wales:
    Customers that have been patronising the store in between 20-22 mo
nths have the lowest transaction count while other
    transactions are high.

    Queensland:
     Customers with Tenure value of of 7 and 16 have the highest trans
```

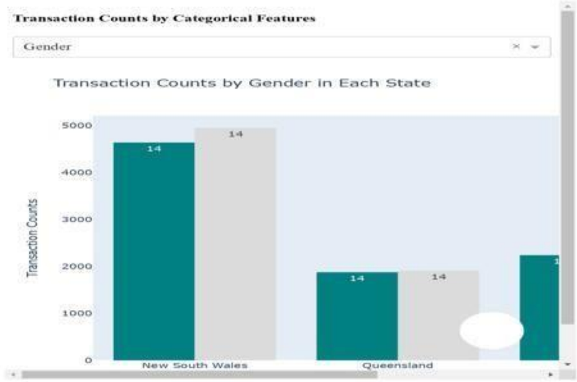

In [64]:

```python
import dash
from dash import html, dcc, Output, Input
import plotly.graph_objects as go
import plotly.io as pio
import pandas as pd

# Sample DataFrame
# df = pd.read_csv('your_dataframe.csv')

app = dash.Dash(__name__)

app.layout = html.Div([
    html.H4("Transaction Counts by Categorical Features"),

    dcc.Dropdown(
        id='dropdown',
        options=[
            {'label': 'Gender', 'value': 'gender'},
            {'label': 'Job Industry category', 'value': 'job_industry_cate
            {'label': 'Wealth Segment', 'value': 'wealth_segment'},
            {'label': 'Age Group', 'value': 'age_group'},
            {'label': 'Property Valuation Group', 'value': 'property_valuat
            {'label': 'Past 3 Year Bike Related Activity', 'value': 'past_
            {'label': 'Tenure Group', 'value': 'tenure_group'},
        ],
        value='gender'
    ),

    dcc.Graph(id='mixed-charts'),
])

@app.callback(
    Output('mixed-charts', 'figure'),
    Input('dropdown', 'value')
)
def update_mixed_charts(selected_dropdown):
    colors = ['teal', 'gainsboro']

    # Gender
    if selected_dropdown == 'gender':
        fig = go.Figure()
        for gender in df['gender'].unique():
            gender_data = df[df['gender'] == gender]
            fig.add_trace(go.Histogram(
                x=gender_data['state'],
                y=gender_data['transaction_count'],
                name=gender,
                marker_color=colors[0] if gender == 'Male' else colors[1],
                text=gender_data['transaction_count'],
                textposition='auto'
            ))
        title = 'Transaction Counts by Gender in Each State'

    # Property Valuation Group
    elif selected_dropdown == 'property_valuation_group':
        fig = go.Figure()
        for group in df['property_valuation_group'].unique():
```

```python
            group_data = df[df['property_valuation_group'] == group]
            fig.add_trace(go.Histogram(
                x=group_data['state'],
                y=group_data['transaction_count'],
                name=group,
                text=group_data['transaction_count'],
                textposition='auto'
            ))
        title = 'Transaction Counts by Property Valuation Group in Each Sta

    # Job Category
    elif selected_dropdown == 'job_industry_category':
        fig = go.Figure()
        for category in df['job_industry_category'].unique():
            category_data = df[df['job_industry_category'] == category]
            fig.add_trace(go.Histogram(
                x=category_data['state'],
                y=category_data['transaction_count'],
                name=category,
                text=category_data['transaction_count'],
                textposition='auto'
            ))
        title = 'Transaction Counts by Job Industry Category in Each State'

    # Wealth Segment
    elif selected_dropdown == 'wealth_segment':
        fig = go.Figure()
        for segment in df['wealth_segment'].unique():
            segment_data = df[df['wealth_segment'] == segment]
            fig.add_trace(go.Histogram(
                x=segment_data['state'],
                y=segment_data['transaction_count'],
                name=segment,
                text=segment_data['transaction_count'],
                textposition='auto'
            ))
        title = 'Transaction Counts by Wealth Segment in Each State'

    # Age Group
    elif selected_dropdown == 'age_group':
        fig = go.Figure()
        for group in df['age_group'].unique():
            group_data = df[df['age_group'] == group]
            fig.add_trace(go.Histogram(
                x=group_data['state'],
                y=group_data['transaction_count'],
                name=group,
                text=group_data['transaction_count'],
                textposition='auto'
            ))
        title = 'Transaction Counts by Age Group in Each State'

    # Past 3 Years Bike Related Activities
    elif selected_dropdown == 'past_3_years_bike_related_activity':
        fig = go.Figure()
        for activity in df['past_3_years_bike_related_activity'].unique():
            activity_data = df[df['past_3_years_bike_related_activity'] ==
```



```python
            fig.add_trace(go.Histogram(
                x=activity_data['state'],
                y=activity_data['transaction_count'],
                name=activity,
                text=activity_data['transaction_count'],
                textposition='auto'
            ))
        title = 'Transaction Counts by Past 3 Years Bike Related Activity i

    # Tenure Group
    elif selected_dropdown == 'tenure_group':
        fig = go.Figure()
        for group in df['tenure_group'].unique():
            group_data = df[df['tenure_group'] == group]
            fig.add_trace(go.Histogram(
                x=group_data['state'],
                y=group_data['transaction_count'],
                name=group,
                text=group_data['transaction_count'],
                textposition='auto'
            ))
        title = 'Transaction Counts by Tenure Group in Each State'

    # Set plots Layout
    fig.update_layout(
        title=title,
        xaxis_title='State',
        yaxis_title='Transaction Counts',
        barmode='group',
        xaxis=dict(showgrid=False, visible=True, showticklabels=True),
        yaxis=dict(showgrid=False, visible=True, showticklabels=True),
        width=800,
        height=600
    )

    # Save the plot to a file
 #  file_name = f"{selected_dropdown}_plot.png"
  # pio.write_image(fig, file_name)

    return fig

if __name__ == '__main__':
    app.run_server(debug=True, port=8092)
```



In [65]:

```
import plotly.graph_objects as go

app = dash.Dash(__name__)

app.layout = html.Div([
    html.H4("Transaction Count and Percentage Job Title"),

    dcc.Dropdown(
        id='state',
        options=[{'label': state, 'value': state} for state in df['state'].
        value=df['state'].unique()[0]
    ),
    dcc.Input(id='job-title-filter', type='text', placeholder='Filter by jo

    dcc.Graph(id='table'),
])

@app.callback(
    Output('table', 'figure'),
    Input('state', 'value'),
    Input('job-title-filter', 'value')
)
def update_table(selected_state, job_title_filter):
    # Filter the data based on selected state
    df_state = df[df['state'] == selected_state]

    # Apply job title filter
    if job_title_filter:
        df_state = df_state[df_state['job_title'].str.contains(job_title_fi

    # Group by job title and calculate transaction count and percentage
    job_title_counts = df_state['job_title'].value_counts()
    total_transactions = len(df_state)
    job_title_percentage = (job_title_counts / total_transactions) * 100
      # Round the percentage to two decimal places
    job_title_percentage = job_title_percentage.round(2)

    # Create DataFrame for table display
    df_display = pd.DataFrame({
        'Job Title': job_title_counts.index,
        'Transaction Count': job_title_counts.values,
        'Transaction Percentage': job_title_percentage.values
    })

    # Create table figure
    fig = go.Figure(data=[go.Table(
        header=dict(values=list(df_display.columns), fill_color='paleturquo
        cells=dict(values=[df_display['Job Title'], df_display['Transactior
                   fill_color='lavender', align='left'))])

    # Update Layout
    fig.update_layout(title_text=f'Transaction Count and Percentage by Job

    return fig

if __name__ == '__main__':
```

```
    app.run_server(debug=True, port=8062)
```

Transaction Count and Percentage Job Title

New South Wales

Filter by job title...

Transaction Count and Percentage by Job Title in New Sout

Job Title	Transaction Count	Trans
Production Worker	257	2.68
Financial Officer	215	2.24
Healthcare	156	1.63
Retailer	148	1.54
Social Worker	124	1.29
Legal Assistant	123	1.28
Executive Secretary	118	1.23
Business Systems Development Analyst	117	1.22
Speech Pathologist	116	1.21
Assistant Professor	111	1.16
Sales Representative	105	1.09
Tax Accountant	104	1.08
Associate Professor	103	1.07
VP Product Management	103	1.07
Information Systems Manager	101	1.05
Nuclear Power Engineer	100	1.04
Internal Auditor	99	1.03
Compensation Analyst	95	0.99

In [66]:

```
import plotly.express as px
app = dash.Dash(__name__)

app.layout = html.Div([
    html.H4("Transaction Counts by Categorical Features"),

    dcc.Dropdown(
        id='dropdown',
        options=[
            {'label': 'Online Order', 'value': 'online_order'},
            {'label': 'Deceased Indicator', 'value': 'deceased_indicator'},
            {'label': 'Order Status', 'value': 'order_status'},
            {'label': 'Owns Car', 'value': 'owns_car'},
        ],
        value='online_order'
    ),

    dcc.Graph(id='mixed-charts'),
])

@app.callback(
    Output('mixed-charts', 'figure'),
    Input('dropdown', 'value')
)
def update_mixed_charts(selected_dropdown):
    colors= ['teal', 'gainsboro']
    # Online Order
    if selected_dropdown == 'online_order':
        fig = px.histogram(df, x='state', color='online_order', barmode='gr
                            title='Transaction Counts by Online Order in Eac
                           color_discrete_sequence=colors)
        fig.update_layout(yaxis={'visible':False, 'showticklabels':False, '

    # Deceased Indicator
    elif selected_dropdown == 'deceased_indicator':
        fig = px.histogram(df, x='state', color='deceased_indicator', barmc
                            title='Transaction Counts by Deceased Indicator
                           color_discrete_sequence=colors)
        fig.update_layout(yaxis={'visible':False, 'showticklabels':False, '

    # Order Status
    elif selected_dropdown == 'order_status':
        fig = px.histogram(df, x='state', color='order_status', barmode='gr
                            title='Transaction Counts by Order Status in Eac
                           color_discrete_sequence=colors)
        fig.update_layout(yaxis={'visible':False, 'showticklabels':False, '

    # Owns Car
    elif selected_dropdown == 'owns_car':
        fig = px.histogram(df, x='state', color='owns_car', barmode='group'
                            title='Transaction Counts by Owns Car in Each St
                           color_discrete_sequence=colors)
        fig.update_layout(yaxis={'visible':False, 'showticklabels':False, '
    return fig

if __name__ == '__main__':
```

In [67]:

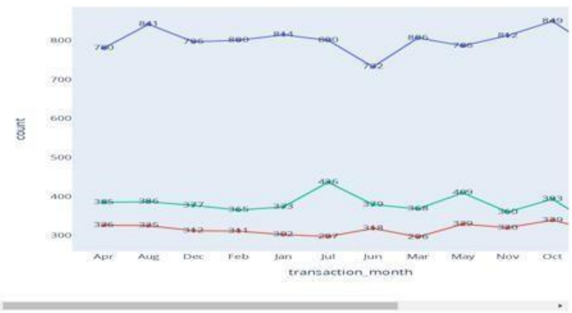
```
app = dash.Dash(__name__)

app.layout = html.Div([
    html.H4("Transaction Counts by Products"),

    dcc.Dropdown(
        id='dropdown',
        options=[
            {'label': 'Brand', 'value': 'brand'},
            {'label': 'Product Class', 'value': 'product_class'},
            {'label': 'Product Line', 'value': 'product_line'},
            {'label': 'Product Size', 'value': 'product_size'},
        ],
        value='brand'
    ),

    dcc.Graph(id='mixed-charts'),
])

@app.callback(
    Output('mixed-charts', 'figure'),
    Input('dropdown', 'value')
)
def update_mixed_charts(selected_dropdown):
    colors= ['cyan', 'gainsboro', 'orange', 'azure', 'lightblue', 'aquamari
    fig = go.Figure()

    if selected_dropdown in ['brand', 'product_class', 'product_line', 'pro
        data_grouped = df.groupby(['state', selected_dropdown]).size().rese

        unique_items = data_grouped[selected_dropdown].unique()
        color_mapping = {item: colors[i % len(colors)] for i, item in enume

        for item in unique_items:
            item_data = data_grouped[data_grouped[selected_dropdown] == ite
            fig.add_trace(go.Bar(
                x=item_data['state'],
                y=item_data['transaction_count'],
                name=item,
                text=item_data['transaction_count'],
                marker_color=color_mapping[item]
            ))

        title = f'Transaction Counts by {selected_dropdown.capitalize()} ir

        fig.update_layout(
            title=title,
            xaxis_title='State',
            yaxis_title='Transaction Counts',
            barmode='group',
            xaxis=dict(showgrid=False, visible=True, showticklabels=True),
            yaxis=dict(showgrid=False, visible=False, showticklabels=False)
            width=800,
            height=600
        )

        # Save the plot to a file
```


Monthly Transactions in Each State

In [68]:

```
# Group by transaction_month and count the transactions
transaction_count = df.groupby(['transaction_month', 'state']).size().reset

# Create the line plot
fig = px.line(transaction_count, x='transaction_month', y='count', text='cc
              title=f'Monthly transaction by states', width=800, height=60(

#fig.update_xaxes(showticklabels=False)
fig.update_layout(xaxis={'visible':True, 'showticklabels':True, 'showgrid':
fig.update_layout(yaxis={'visible':True, 'showticklabels':True, 'showgrid':

# Save the plot to a file
file_name = "monthly_transactions_by_state.png"
pio.write_image(fig, file_name)

fig.show()
```

Monthly transaction by states



From the above plots we can clearly tell the following interpretation:

Gender:

```
New South Wales:
Most: Female
Moderate: Male

Queensland:
Most: Female
Moderate: Male

Victoria:
Most: Both Female then Male
```


Property Valuation:

```
New South Wales:
Most: High Valuation (9-12)
Moderate: Moderate Valuation (5-8)
Least: Low Valuation (1-4)

Queensland:
Most: Moderate Valuation (5-8)
Moderate: Low Valuation (1-4)
Least:  High Valuation (9-12)

Victoria:
Most: Moderate Valuation (5-8)
Moderate:  High Valuation (9-12)
Least: Low Valuation (1-4)
```

Past 3 Years Bike Related Purchses:

```
    New South Wales:
    Most: Low Activities(0-19), Moderate Activities(20-39), High Activ
ites(60-79), Average Activities(40-59), then Very
    High Activities(80-99) in that order.

    Queensland:
    Most: Average Activities(40-59), Low Activities(0-19), Moderate Ac
tivities(20-39), Very High Activites(60-79), then
    High Activities(80-99) in that order.

    Victoria:
    Most: Moderate Activities(20-39), Average Activities(40-59), Very
High Activites(60-79), Low Activities(0-19), then
    High Activities(80-99) in that order.
```

Age Group:

```
    New South Wales:
    Most: Middle-Aged
    Moderate: Under-30
    Least: Senior Citizens

    Queensland:
    Most: Middle-Aged
    Moderate: Under-30
    Least: Senior Citizens

    Victoria:
    Most: Middle-Aged
    Moderate: Senior Citizens
    Least: Under-30
```

Job Induustry Category:

```
    New South Wales:
    Most: Manufactuiring, Financial Services, Uncategorized, then Heal
th
    Moderate: Retail then Property
    Least: IT, Entertainment, Agriculture, then Telecommunication

    Queensland:
    Most:Manufactuiring, Financial Services, Uncategorized, then Healt
h
    Moderate: Retail
    Least: Property, IT, Entertainment, Agriculture, then Telecommunic
ation

    Victoria:
    Most:Manufactuiring, Financial Services, Uncategorized, then Healt
h
    Moderate: Retail
    Least: Property, Entertainment, IT, Agriculture, then Telecommunic
ation
```

Wealth Segment:


```
    New South Wales:
    Most: Mass Customers
    Moderate: High Newt Worth Customers
    Least: Affluent Customers

    Queensland:
    Most: Mass Customers
    Moderate: High Net Worth then Affluent Customers

    Victoria:
    Most: Mass Customers
    Moderate: High Newt Worth Customers
    Least: Affluent Customers
```

Owns Car:

```
    New South Wales:
    Most: Yes then No

    Queensland:
    Most: No then Yes

    Victoria:
    Most: Yes then No
```

Tenure:

```
    New South Wales:
    Most: Regular then Loyal
    Moderate: New
    Least: Long-term

    Queensland:
    Most: Regular then Loyal
    Moderate: New
    Least: Long-term

    Victoria:
    Most: Regular then Loyal
    Moderate: New
    Least: Long-term
```

Job Title:

```
    New South Wales:
    Top three: Production Workers, Finanacial Officers, and Healthcare
workers

    Queensland:
    Top three: Financial Officers, Production Workers, and Senior Qual
ity, Engineers.

    Victoria:
    Top three: Production Workers, Clinical Specialists, and Healthcar
e workers.
```

Brand:

```
    The top three Brands in the three states are the Solex brand, Gian
t Bicycles brand, and the WeareA2B brands
```


In [69]:

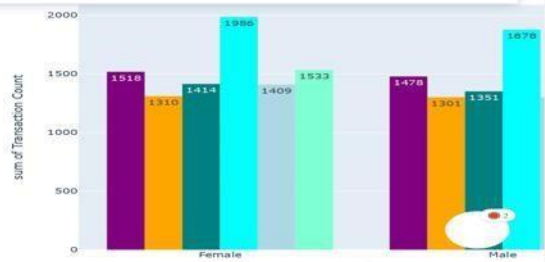
```python
# Set up Dash app
app = dash.Dash(__name__)

# Define dropdown options
options_1 = ['gender', 'job_industry_category', 'age_group', 'wealth_segmer
options_2 = ['brand', 'product_class', 'product_line', 'product_size']

# Define Layout
app.layout = html.Div([
    html.H1("Transaction Counts by Group"),
    dcc.Dropdown(
        id='dropdown_1',
        options=[{'label': col, 'value': col} for col in options_1],
        value=options_1[0]
    ),
    dcc.Dropdown(
        id='dropdown_2',
        options=[{'label': col, 'value': col} for col in options_2],
        value=options_2[0]
    ),
    html.Div(id='graph-container')
])

# Define callback
@app.callback(
    Output('graph-container', 'children'),
    [Input('dropdown_1', 'value'),
     Input('dropdown_2', 'value')]
)
def update_graph(selected_option_1, selected_option_2):
    # Filter dataframe based on selected dropdown values
    filtered_df = df[[selected_option_1, selected_option_2]]

    # Check if one of the options is 'transaction_month'
    if 'transaction_month' in [selected_option_1, selected_option_2]:
        grouped_df = filtered_df.groupby([selected_option_1, selected_optic
        # If yes, create a line plot
        fig = px.line(grouped_df, x='transaction_month', y='count', color=s
                      title=f'Transaction Counts by {selected_option_1.capi
        fig.write_image(f'plot_{selected_option_1}_{selected_option_2}.png'
        return dcc.Graph(figure=fig)

    # Check if one of the options is 'job_title'
    elif 'job_title' in [selected_option_1, selected_option_2]:
        # If yes, create a table
        df_display = filtered_df.groupby([selected_option_1, selected_optic
        df_display = df_display.sort_values(by='Transaction Count', ascendi
        # Create table figure
        fig = go.Figure(data=[go.Table(
            header=dict(values=list(df_display.columns), fill_color='paletu
            cells=dict(values=[df_display['job_title'], df_display[selectec
                       fill_color='lavender', align='left'))])
        fig.update_layout(width=800, height=600)
        fig.write_image(f'plot_{selected_option_1}_{selected_option_2}.png'
        return dcc.Graph(figure=fig)
```

```
    else:
        # Otherwise, create a grouped bar chart
        colors= ['purple', 'orange', 'teal', 'cyan', 'lightblue', 'aquamari

        grouped_df = filtered_df.groupby([selected_option_1, selected_optic
        fig = px.histogram(grouped_df, x=selected_option_1, y='count', colc
                           text_auto=True, color_discrete_sequence=colors,
                       labels={selected_option_1: selected_option_1.capitali
                       title=f'Transaction Counts by {selected_option_1.capit
        fig.write_image(f'plot_{selected_option_1}_{selected_option_2}.png'
        return dcc.Graph(figure=fig)

# Run the app
if __name__ == '__main__':
    app.run_server(debug=True)
```

Errors (2)

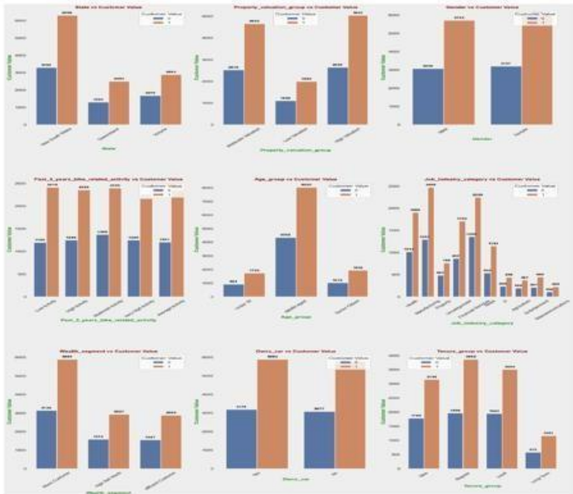
ing chunk 792 failed. (timeout: http://127.0.0.… 12:25:43 AM

ing chunk 792 failed. (timeout: http://127.0.0.… 12:25:43 AM



In [70]:

```
fig, axes = plt.subplots(3, 3, figsize=(20, 30))
fig.subplots_adjust(hspace=0.5)
fig.set_facecolor('#eeeeee') # background color

# Define the columns to plot
columns_to_plot = ['state', 'property_valuation_group', 'gender', 'past_3_y


# Iterate over the columns and create countplots
for i, column in enumerate(columns_to_plot):
    row = i // 3
    col = i % 3

    ax = axes[row, col]

    # Create a countplot
    sns.countplot(x=column, hue='customer_value', data=df, ax=ax)
    sns.despine(left=True, bottom=True)

    # Add value labels to the bars and convert values to int with decimals
    for p in ax.patches:
        height = p.get_height()
        ax.annotate(f'{int(height)}',  # Format as currency with thousands
                    xy=(p.get_x() + p.get_width() / 2, height),
                    xytext=(0, 5),  # Offset for the text above the bar
                    textcoords='offset points',  # Set text coordination
                    ha='center',  # Horizontal alignment
                    va='bottom',  # Vertical alignment
                    fontsize=10,  # Font size for the text
                    color='black',  # Color of the text
                    weight='bold'  # Text weight (e.g., 'bold', 'normal')
                   )


    # Customize titles and labels
    ax.set_facecolor('#eeeeee')
    ax.grid(False)
    ax.set_title(f'{column.capitalize()} vs Customer Value', fontweight='bc
    ax.set_xlabel(column.capitalize(), color='forestgreen', fontweight='bol
    ax.set_ylabel('Customer Value', color='forestgreen', fontweight='bold')
    ax.legend(title='Customer Value')
    ax.tick_params(axis='x', labelrotation=45)

# Remove empty subplots
for i in range(len(columns_to_plot), 9):
    row = i // 3
    col = i % 3
    fig.delaxes(axes[row, col])

# Save the figure
plt.savefig('customer_value.png', bbox_inches='tight', pad_inches=0)

plt.show()
```



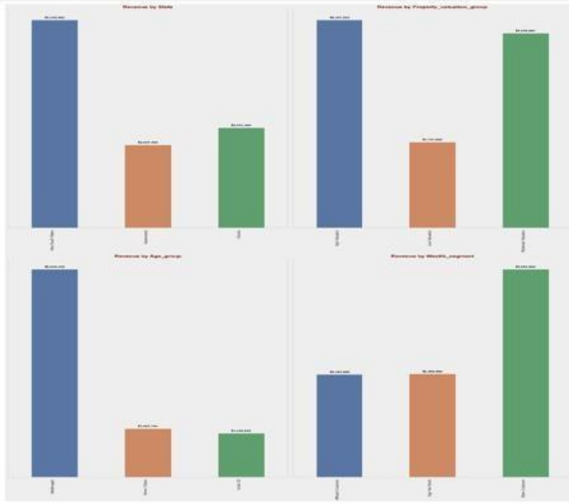
From the above plots we can clearly tell the following interpretation:

In all the features valuable customers carried out more transaction than non-valuable customers across all metrics including in each state.

In [71]:

```
# Select columns to plot
column_to_plot = ['state', 'property_valuation_group', 'age_group', 'wealth

fig, axes = plt.subplots(2, 2, figsize=(20, 30))  # Define the figure and c
#plt.subplots_adjust(wspace=0.4, hspace=0.8)
fig.set_facecolor('#eeeeee')  # background color
#fig.subplots_adjust(hspace=8)  # Set plots margin

# Loop through each category
for i, category in enumerate(column_to_plot):
    ax = axes[i // 2, i % 2]

    # Group the data by the category and calculate sum of revenue
    grouped_data = df.groupby(category)['revenue'].sum().reset_index()

    # Convert x-axis values to strings
    grouped_data[category] = grouped_data[category].astype(str)

# Plot the barplot

    # Plot vertical bar plot
    sns.barplot(x=category, y='revenue', data=grouped_data, width=0.5, ax=a
    sns.despine(right=False, bottom=False)


    ax.set_facecolor('#eeeeee') # Set chart backgroud
    ax.grid(False) # Remove gridlines

    ax.set_title(f'Revenue by {category.capitalize()}', color='maroon', for

    # Set x-axis ticks and labels
    #ax.set_xticks(range(len(grouped_data)))
    ax.set_xticklabels(grouped_data[category], ha='right', rotation=90)
    ax.set_xlabel(None)
    ax.set_ylabel('')
    ax.set_yticks([])

    # Add value text to the bars
    for x, y in enumerate(grouped_data['revenue']):
        ax.annotate(f'${int(y):,}',  # Format as currency with thousands se
                    xy=(x, y),
                    xytext=(0, 5),  # Offset for the text above the bar
                    textcoords='offset points',
                    ha='center',  # Horizontal alignment
                    va='bottom',  # Vertical alignment
                    fontsize=10,  # Font size for the text
                    color='black',  # Color of the text
                    weight='bold'  # Text weight (e.g., 'bold', 'normal')
                    )

# Save the figure
plt.savefig('revenue_plots.png', bbox_inches='tight', pad_inches=0)

plt.tight_layout()
```

```
plt.show()
```



In [72]:

```
# Select columns to plot
column_to_plot = ['gender', 'tenure_group', 'past_3_years_bike_related_acti

# Define the figure and axes
fig, axes = plt.subplots(3, 2, figsize=(30, 35))
fig.set_facecolor('#eeeeee') # background color

# Loop through each category
for i, category in enumerate(column_to_plot):
    ax = axes[i // 2, i % 2]

    # Group the data by the category and calculate sum of revenue
    grouped_data = df.groupby(category)['revenue'].sum().reset_index()

    # Convert x-axis values to strings
    grouped_data[category] = grouped_data[category].astype(str)

    ax.set_facecolor('#eeeeee') # Set chart background
    ax.grid(False) # Remove gridlines
    ax.set_title(f'Revenue by {category.capitalize()}', color='maroon', for

    # Plot the barplot
    if category in ['gender', 'tenure_group', 'past_3_years_bike_related_ac
        # Plot horizontal bar plot
        sns.barplot(y=category, x='revenue', data=grouped_data, ax=ax, widt
        sns.despine(ax=ax, left=True, bottom=True)

        # Annotate bar chart with values
        for index, value in enumerate(grouped_data['revenue']):
            ax.text(value, index, f'${int(value):,}', ha='left', va='center

        # Set x-axis and y-axis labels
        ax.set_xlabel(None)
        ax.set_xticks([])
        ax.set_ylabel(category.capitalize(), color='forestgreen', fontweigh

    elif category == 'transaction_month':
        # Plot line plot
        sns.lineplot(x=category, y='revenue', data=grouped_data, ax=ax)
        sns.despine(ax=ax, right=False, bottom=False)

        # Add average line to the line plot
        average_revenue = grouped_data['revenue'].mean()
        ax.axhline(average_revenue, color='red', linestyle='--')
        # Annotate average line
        max_x_value = grouped_data[category].max()
        ax.text(max_x_value, average_revenue, f'Average Revenue: ${int(aver

        # Annotate line chart with text
        for index, (month, revenue) in enumerate(zip(grouped_data[category]
            ax.text(month, revenue, f'{month}\n${int(revenue):,}', ha='cent

        # Set x-axis and y-axis labels
        ax.set_xlabel(None)
        ax.set_xticks([])
        ax.set_ylabel(None)
```

```python
        # Add currency format for y-axis (Revenue)
        ax.yaxis.set_major_formatter('${:,.0f}'.format)

# Remove empty subplots
for i in range(len(column_to_plot), 6):
    row = i // 2
    col = i % 2
    fig.delaxes(axes[row, col])

# Save the figure
plt.savefig('revenue_plots2.png', bbox_inches='tight', pad_inches=0)

plt.show()
```