BRP Report

Data Quality Analyst Assignment

Kallil de Araujo Bezerra

October 21, 2023

Contents

1 Task 1

The goal of the first task is to present the data so we can familiarize ourselves with what's presenting.

1.1 A - Read both .csv files into Python

Reading both files using Python:

First, two variables were created. One for each file, and they carry the string that represents their location, so if we ever need to change the path it can be easily done by just changing the value stored by the variable.

```
dealer_data_path = "/content/sample_data/DEALER.csv"
retail_data_path = "/content/sample_data/RETAIL_SALES.csv"
```

After that, we can load the files using:

```
df_dealer = pd.read_csv(dealer_data_path)
df_retail = pd.read_csv(retail_data_path)
```

1.2 B - Display the data types for each column for both datasets

Again, we create variables to store their data types, and if we need to change something here it's possible to do it by just changing the values in the variables.

```
dealer_data_types = df_dealer.dtypes
retail_data_types = df_retail.dtypes
```

```
print("1 B:\n")
print("Data types for the dealer dataset:\n")
print(dealer_data_types)

print("\n----//----//----//----//----//----//----//----")
print("\nData types for the retail dataset:\n")
print(retail_data_types)
```

The result of this code (for the dealer dataset) can be seen in figure 1.

Figure 1: Data types

```
1 B:

Data types for the dealer dataset:

CUSTOMER_NUMBER               int64
ATV                          object
ATV_CREATION_DATE            object
CERTIFICATION_LEVEL_CODE    float64
CITY_DEALER                  object
COUNTRY_CODE                 object
DEALER_TYPE_CODE              int64
DEALER_TYPE_DESC             object
GROUP_NUMBER                  int64
PWC                          object
PWC_CREATION_DATE            object
SALES_GROUP                  object
SALES_OFFICE                  int64
SALES_ORG                     int64
SNOW                         object
SNOW_CREATION_DATE           object
SSV                          object
SSV_CREATION_DATE            object
STATE_CODE                   object
TERRITORY_CODE                int64
THREE_W                      object
THREE_W_CREATION_DATE        object
GEOGRAPHY                   float64
dtype: object
```

1.3 C - What is the number of rows and columns for each dataset?

Moving on, we need to see the size of our dataset. To do this we can use the *shape* method from *Pandas*.

```
dealer_data_rows = df_dealer.shape
retail_data_rows = df_retail.shape
```

```
print("1 C:\n")
print("Number of rows and columns in the dealer dataset")
print("# of rows: {}".format(dealer_data_rows[0]))
print("# of columns: {}\n".format(dealer_data_rows[1]))

print("----//----//----//----//----//----//----//----")
print("\nNumber of rows and columns in the retail dataset")
print("# of rows: {}".format(retail_data_rows[0]))
print("# of columns: {}".format(retail_data_rows[1]))
```

The results can be seen in figure 2.

Figure 2: Data shape

```
1 C:

Number of rows and columns in the dealer dataset
# of rows: 4153
# of columns: 23

----//----//----//----//----//----//----//----

Number of rows and columns in the retail dataset
# of rows: 134261
# of columns: 21
```

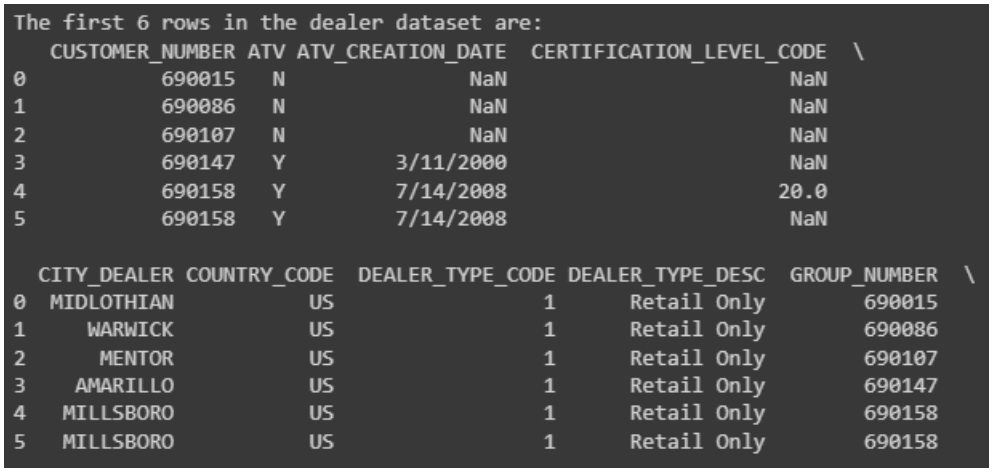
1.4 D - Display the first 6 rows for the DEALER dataset and the last 8 for the RETAIL_SALES one

```
dealer_top_six = df_dealer.head(6)
retail_bot_eight = df_retail.tail(8)
```

```
print("The first 6 rows in the dealer dataset are:")
print(dealer_top_six)

print("\n----//----//----//----//----//----//----//----")
print("\nThe last 8 rows in the retail data set are:")
print(retail_bot_eight)
```

The results can be seen in figures 3 and 4, but they're not complete because the images would take too much space in the report.

Figure 3: First 6 rows of the DEALER dataset

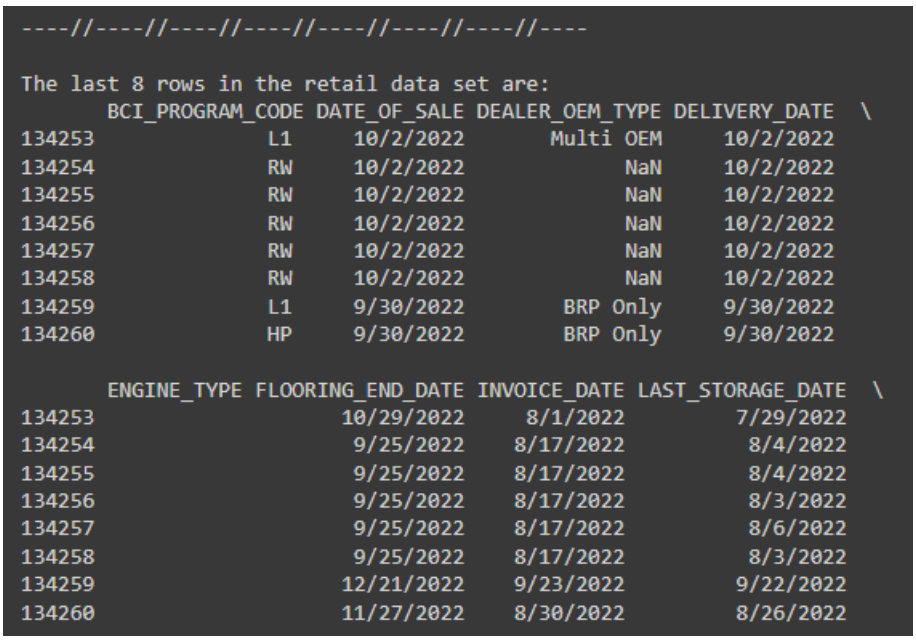


The first 6 rows in the dealer dataset are:

	CUSTOMER_NUMBER	ATV	ATV_CREATION_DATE	CERTIFICATION_LEVEL_CODE	\
0	690015	N	NaN	NaN	
1	690086	N	NaN	NaN	
2	690107	N	NaN	NaN	
3	690147	Y	3/11/2000	NaN	
4	690158	Y	7/14/2008	20.0	
5	690158	Y	7/14/2008	NaN	

	CITY_DEALER	COUNTRY_CODE	DEALER_TYPE_CODE	DEALER_TYPE_DESC	GROUP_NUMBER	\
0	MIDLOTHIAN	US	1	Retail Only	690015	
1	WARWICK	US	1	Retail Only	690086	
2	MENTOR	US	1	Retail Only	690107	
3	AMARILLO	US	1	Retail Only	690147	
4	MILLSBORO	US	1	Retail Only	690158	
5	MILLSBORO	US	1	Retail Only	690158	

Figure 4: Last 8 rows of the RETAIL dataset



----//----//----//----//----//----//----//----

The last 8 rows in the retail data set are:

	BCI_PROGRAM_CODE	DATE_OF_SALE	DEALER_OEM_TYPE	DELIVERY_DATE	\
134253	L1	10/2/2022	Multi OEM	10/2/2022	
134254	RW	10/2/2022	NaN	10/2/2022	
134255	RW	10/2/2022	NaN	10/2/2022	
134256	RW	10/2/2022	NaN	10/2/2022	
134257	RW	10/2/2022	NaN	10/2/2022	
134258	RW	10/2/2022	NaN	10/2/2022	
134259	L1	9/30/2022	BRP Only	9/30/2022	
134260	HP	9/30/2022	BRP Only	9/30/2022	

	ENGINE_TYPE	FLOORING_END_DATE	INVOICE_DATE	LAST_STORAGE_DATE	\
134253		10/29/2022	8/1/2022	7/29/2022	
134254		9/25/2022	8/17/2022	8/4/2022	
134255		9/25/2022	8/17/2022	8/4/2022	
134256		9/25/2022	8/17/2022	8/3/2022	
134257		9/25/2022	8/17/2022	8/6/2022	
134258		9/25/2022	8/17/2022	8/3/2022	
134259		12/21/2022	9/23/2022	9/22/2022	
134260		11/27/2022	8/30/2022	8/26/2022	

2 Task 2

Now that the dataset is better understood, we can proceed with some data cleaning.

2.1 A - Remove duplicated lines in the DEALER dataset. How many of duplicated lines there were?

There were 5 duplicate rows in the original dataset. Below is the code that was used to do this analysis.

```
df_duplicates_dropped = df_dealer.drop_duplicates(keep='
                            first', inplace=False)
num_duplicates_dropped = len(df_dealer) - len(
                            df_duplicates_dropped)
```

```
print("Number of Duplicate Rows Dropped: {}" .format(
                            num_duplicates_dropped))
```

Figure 5: Dropping and counting how many duplicate rows the dataset had

```
Dropped Rows:
Number of Duplicate Rows Dropped: 5
```

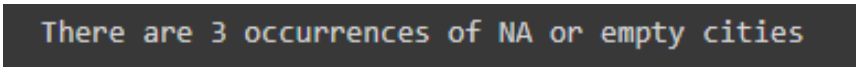
After that, I dropped and saved the new

2.2 B - How many NAs exist within the CITY_DEALER column in the DEALER dataset? Replace the NA values for the string: no city found.

There were 3 occurrences of CITY_DEALER with NA or empty values.

```
dealer_na_cities = df_dealer['CITY_DEALER'].isna().sum()
print('There are {} occurrences of NA or empty cities'.
                            format(dealer_na_cities))
```

Figure 6: Total number of NA or empty CITY_DEALER



Now we swap those NA or empty values for *no city found*.

```
df_dealer['CITY_DEALER'].fillna('no city found', inplace =
                                True)
```

2.3 C - Filter the DEALER dataset to include only dealers with COUNTRY_CODE US or CA

```
df_dealer = df_dealer[(df_dealer['COUNTRY_CODE'] == 'US')|(
                                df_dealer['COUNTRY_CODE'] == '
                                CA')]
```

2.4 D - In the RETAIL_SALES dataset, remove the trailing x (xxx) of the column REG_DEALER_NUMBER

```
df_retail['REG_DEALER_NUMBER'] = df_retail['
                                REG_DEALER_NUMBER'].str.
                                replace('x', '', regex=True)
```

2.5 E - Keep only the following columns in the DEALER dataset: CUSTOMER_NUMBER, CITY_DEALER, COUNTRY_CODE and STATE_CODE

```
df_dealer = df_dealer[['CUSTOMER_NUMBER', 'CITY_DEALER', '
                                COUNTRY_CODE', 'STATE_CODE']]
```

3 Task 3

3.1 A - In the RETAIL_SALES dataset, create a column named RETAIL and assign the integer 1 to it

```
df_retail['RETAIL'] = 1
```

3.2 B - Set the datatypes of columns CUSTOMER_NUMBER and REG_DEALER_NUMBER to integer for the DEALER and RETAIL_SALES datasets respectively

```
df_dealer['CUSTOMER_NUMBER'] = df_dealer['CUSTOMER_NUMBER']
                               .astype(int)
df_retail['REG_DEALER_NUMBER'] = df_retail['
                               REG_DEALER_NUMBER'].astype(int
                               )
```

3.3 C - In the DEALER dataset, create a column called REGION_CODE based on the STATE_CODE columns. Each region is specified as per the dictionary provided in the original doc

```
region_mapping = {
  1: ["AB", "BC", "MB", "NB", "NF",
      "NS", "NT", "NU", "ON", "PE",
      "QC", "SK", "YT"],
  2: ["AL", "CT", "DE", "DC", "FL",
      "GA", "MA", "MD", "ME", "NC",
      "NH", "NJ", "NY", "PA", "PR",
      "RI", "SC", "VA", "VT", "WV"],
  3: ["AR", "IA", "IL", "IN", "KS",
      "KY", "LA", "MI", "MN", "MO",
      "MS", "ND", "NE", "OH", "SD",
      "TN", "WI"],
  4: ["AK", "AZ", "CA", "CO", "HI",
      "ID", "MT", "NM", "NV", "OK",
```

```
        "OR", "TX", "UT", "WA", "WY"]
}

df_dealer['REGION_CODE'] = df_dealer['STATE_CODE'].apply(
lambda state: next((region for region, states in
                            region_mapping.items() if
                            state in states), 5)
)
```

4 Task 4

4.1 A - Merge the DEALER dataset into the RETAIL_SALES using CUSTOMER_NUMBER and REG_DEALER_NUMBER columns. Explain why you selected this specific join type

Before continuing, it's important to say that MODEL_CODE does not exist, so I used MODEL_NUMBER instead.

For this task I assumed that this data would be used for some kind of reporting, so we won't use the NA rows that would come up if we did an outer, left, or right join.

Therefore, the best option was a inner join, in which we will keep only the matching rows from both datasets.

```
df_merged_data = pd.merge(df_retail, df_dealer, left_on = '
                          REG_DEALER_NUMBER', right_on =
                           'CUSTOMER_NUMBER', how = '
                          inner')
```

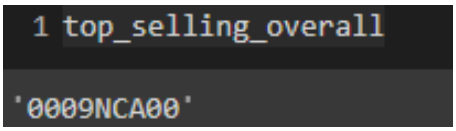
4.2 B - I - What is the overall top selling MODEL_CODE in the STATE_CODE of AZ and MB? Use the RETAIL column to calculate sales.

```
top_selling_overall = df_merged_data[df_merged_data['
                          STATE_CODE'].isin(['AZ','MB'])
```

```
].groupby('MODEL_NUMBER')['
RETAIL'].sum().idxmax()
```

This code will return a value, which is shown in figure 7.

Figure 7: Most sold product in states AZ and MB



4.3 B - II - What are the top 10 MODEL_CODEs sold in REGION_CODE number 3, in the year 2021? Use REGISTRATION_DATE to calculate the dates

```
df_merged_data['REGISTRATION_DATE'] = pd.to_datetime(
                                df_merged_data['
                                REGISTRATION_DATE'], format='%
                                m/%d/%Y')
df_merged_data['REG_YEAR'] = df_merged_data['
                                REGISTRATION_DATE'].dt.year
```

```
filtered_data = df_merged_data[(df_merged_data['REGION_CODE
                                '] == 3) & (df_merged_data['
                                REG_YEAR'] == 2021)]
```

```
model_sales = filtered_data.groupby('MODEL_NUMBER')['RETAIL
                                '].sum()
```

```
top_10_models_2021 = model_sales.nlargest(10)
print(top_10_models_2021)
```

Therefore, the most sold product is the 008JCA00, with 2147 units sold. This can be verified in figure 8.

Figure 8: Top 10 most sold products

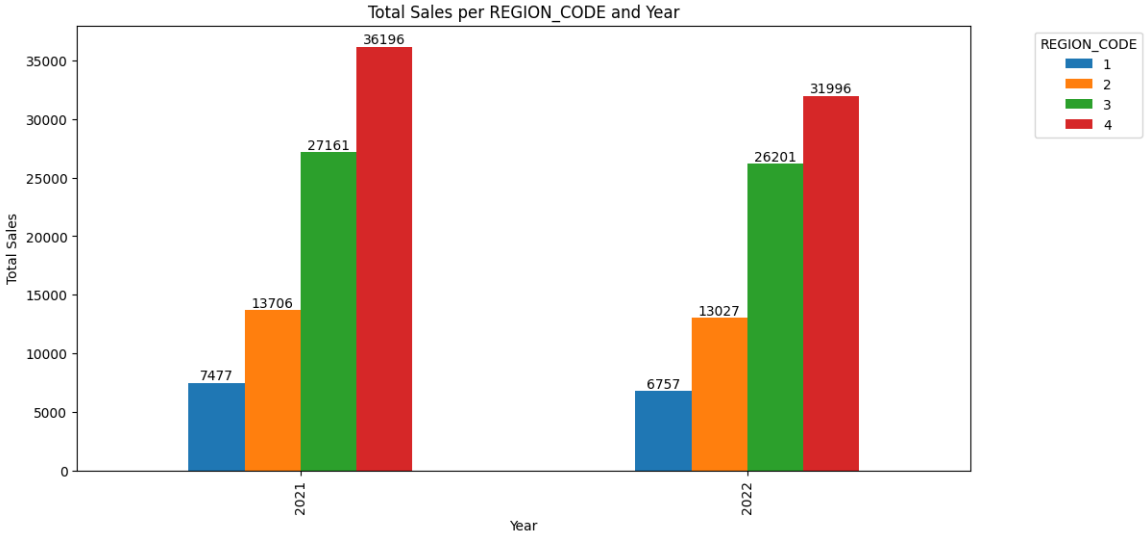
```
MODEL_NUMBER
0008JCA00    2147
0008JCC00    1199
0005CCG00     907
0005CCA00     888
0005CCB00     828
0008ACA00     593
0009CCA00     519
0002TCC00     510
0008CCA00     496
0005JCA00     451
Name: RETAIL, dtype: int64
```

4.4 B - III - Create a chart to display the total sales per REGION_CODE and year

```
import matplotlib.pyplot as plt

# Group the data by REGION_CODE and REG_YEAR and calculate
                            the total sales
total_sales_per_region_year = df_merged_data.groupby(['
                            REGION_CODE', 'REG_YEAR'])['
                            RETAIL'].sum()

# Reset the index to make REGION_CODE and REG_YEAR columns
                            accessible for plotting
total_sales_per_region_year = total_sales_per_region_year.
                            reset_index()

# Create a pivot table to reshape the data for plotting
pivot_table = total_sales_per_region_year.pivot(index='
                            REG_YEAR', columns='
                            REGION_CODE', values='RETAIL')

# Plot the data as a bar chart
ax = pivot_table.plot(kind='bar', stacked=False, figsize=(
                            12, 6))
plt.xlabel('Year')
plt.ylabel('Total Sales')
plt.title('Total Sales per REGION_CODE and Year')
```

```
plt.legend(title='REGION_CODE', loc='upper right',
                                bbox_to_anchor=(1.2, 1))

for p in ax.patches:
ax.annotate(str(int(p.get_height())), (p.get_x() + p.
                                get_width() / 2., p.get_height
                                ()), ha='center', va='bottom')

plt.show()
```

Figure 9: Bar chart with sales per region and year



5 Task 5

5.1 A - Define at least 5 quality checks you would perform in the dataset with a brief explanation of their objective.

1. Date Standardization: Ensure consistent date formats by converting all date entries to a standardized format (e.g., YYYY-MM-DD). This standardization prevents format-related issues when processing the data.

2. Special Character Cleanup: Remove special characters from data entries to maintain consistency when transferring data across different systems, such as from a Microsoft Server to Power BI or a CSV file.

3. Data Validation and Standardization: Perform data validation to verify the accuracy and consistency of location-related data. Ensure that cities and state codes are valid, adhere to standard naming conventions, and eliminate duplicated or similar but distinct entries (e.g., 'Alexandria Bay' and 'Alexandria Bay,') to avoid data duplication.

4. Duplicate Record Detection: Detect and handle duplicated records, which can distort analysis and decision-making. This can involve identifying identical records or reconciling records representing the same entity with variations in data entry.

5. Missing Value Assessment: Check critical columns for missing values. Columns such as 'REG_DEALER_NUMBER' and 'MODEL_NUMBER' should be complete, as they are important identifiers. Address any null or empty values to maintain data integrity.