REFLECTIVE REPORT

GROUP NUMBER 146

Table of Contents

Introduction

In task 1 we did a comprehensive exploration of the dirty data and missing data files wherein we had to be able to identify errors/missing values and fix/impute them with corrections. It really challenged us to put forward our investigative skills and test our python coding knowledge.

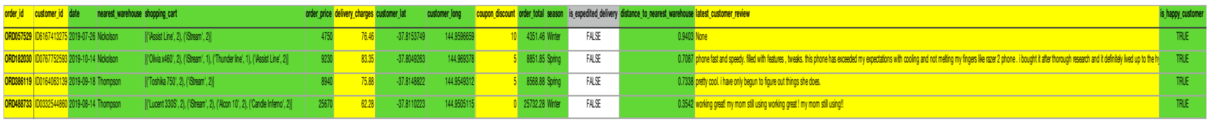
Feedback Improvements

Our TA in the applied session is Shuo Hang, who gave us insight that we had to further check on the interconnected fields and was happy with our presentation format of the data. He recommended we clean up the format and include a table of contents and guided us to look at the missing data sheet to help us clarify doubts in the dirty data task.

The team began first by discussing the specifications in detail and attending consultations to clear any doubts. We then independently worked on sections and merged the best versions together.

Dirty Data

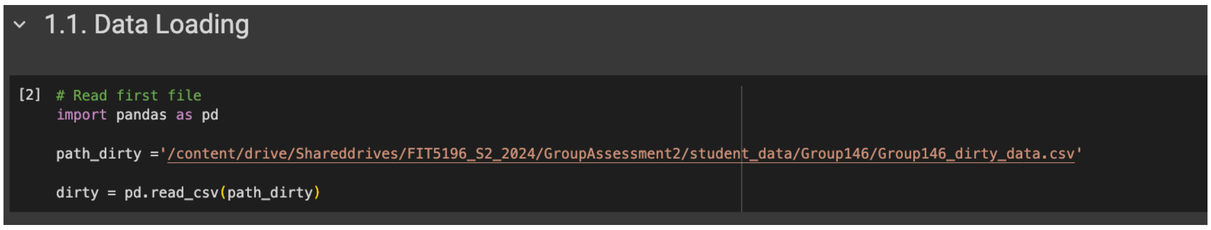
Bhavna began working on this section by first exporting the file to excel and then marking done the clean parts first and inspecting the rest and tracking my progressing marking corrected fields green and already clean ones yellow-



order_id	customer_id	date	nearest_warehouse	shopping_cart	order_price	delivery_charges	customer_lat	customer_long	coupon_discount	order_total	season	is_expedited_delivery	distance_to_nearest_warehouse	latest_customer_review	is_happy_customer
ORD057529	ID6167413275	2019-07-26	Nickolson	[('iAssist Line', 2), ('iStream', 2)]	4750	76.46	-37.8153749	144.9596659	10	4351.46	Winter	FALSE	0.9403	None	TRUE
ORD182030	ID0767752593	2019-10-14	Nickolson	[('Olivia x460', 2), ('iStream', 1), ('Thunder line', 1), ('iAssist Line', 2)]	9230	83.35	-37.8049263	144.969378	5	8851.85	Spring	FALSE	0.7087	phone fast and speedy. filled with features , tweaks. this phone has exceeded my expectations with cooling and not melting my fingers like razer 2 phone . i bought it after thorough research and it definitely lived up to the hy	TRUE
ORD386119	ID0164063139	2019-09-18	Thompson	[('Toshika 750', 2), ('iStream', 2)]	8940	75.88	-37.8148822	144.9549312	5	8568.88	Spring	FALSE	0.7338	pretty cool. i have only begun to figure out things she does.	TRUE
ORD488733	ID0332544860	2019-08-14	Thompson	[('Lucent 330S', 2), ('iStream', 2), ('Alcon 10', 2), ('Candle Inferno', 2)]	25670	62.28	-37.8110223	144.9505115	0	25732.28	Winter	FALSE	0.3542	working great! my mom still using working great ! my mom still using!!	TRUE

Data Loading:

After mounting to drive we loaded the dirty file:



1.1. Data Loading

```
[2] # Read first file
    import pandas as pd

    path_dirty ='/content/drive/Shareddrives/FIT5196_S2_2024/GroupAssessment2/student_data/Group146/Group146_dirty_data.csv'

    dirty = pd.read_csv(path_dirty)
```

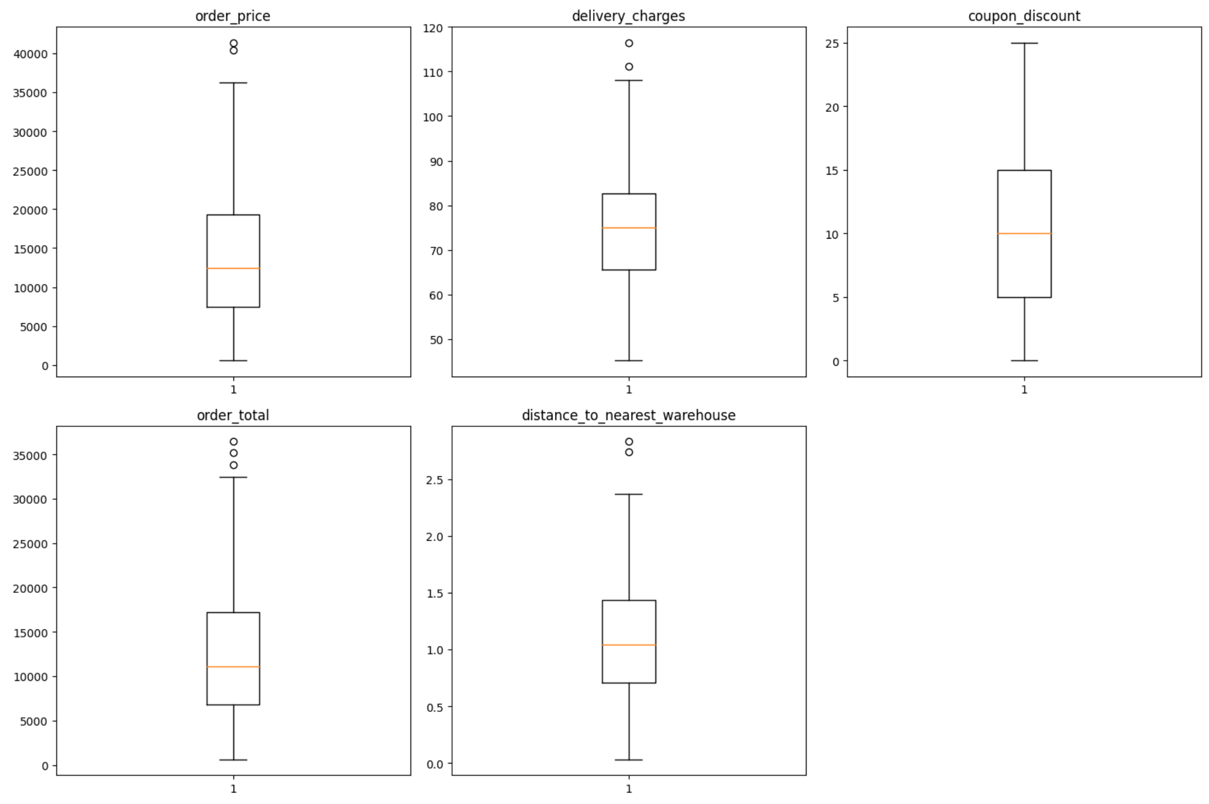
Exploratory Data Analysis:

We began by using basic exploration methods such as-

- Head ()
- Info ()
- Shape
- Describe ()

This helps us view and understand the data, data types, statistics and summary of the data being used.

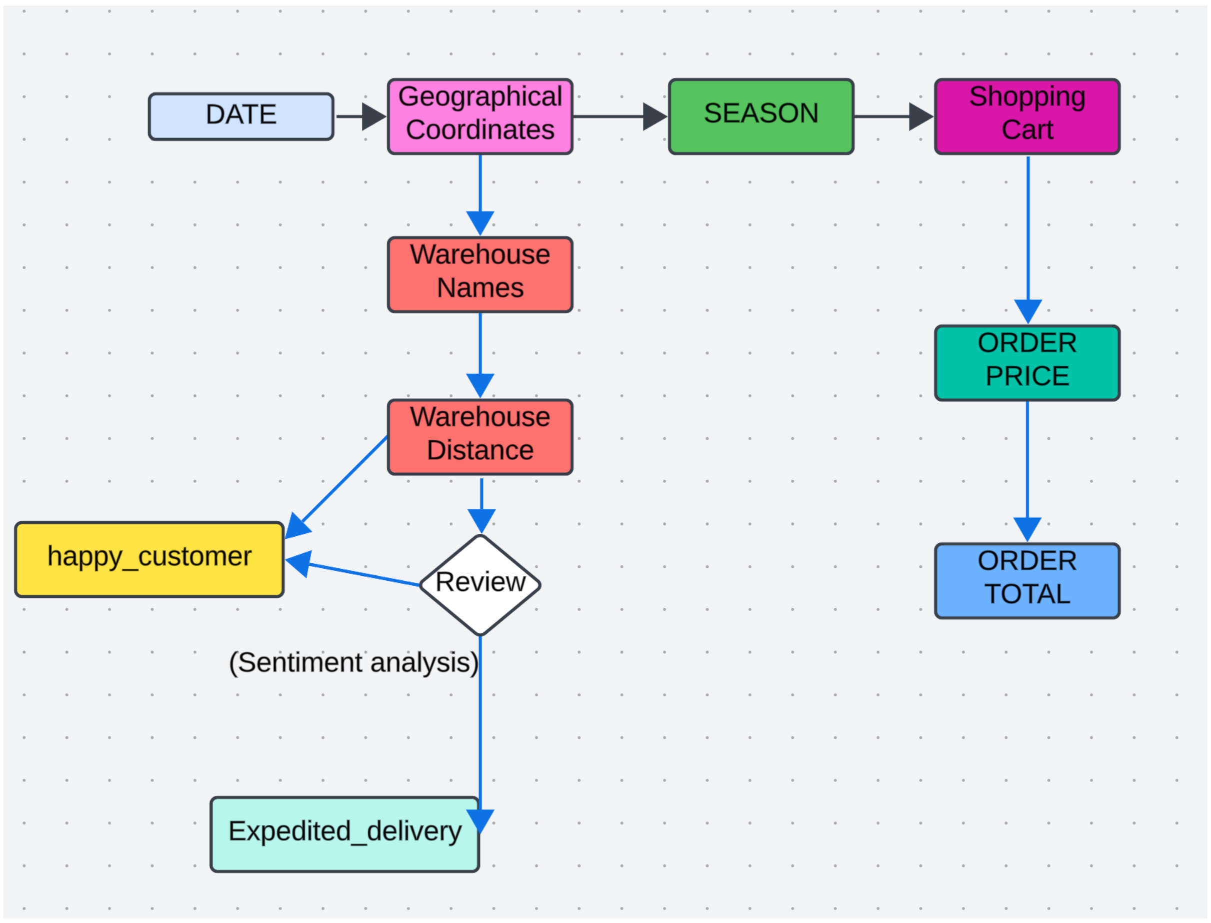
Once these preliminary steps were done, I further looked for nulls and duplicates in the data and proceeded to plot from the data using matplot-



The boxplots provide a summary of the key numerical features in the dataset, including order price, delivery_charges, coupon_discount, order_total, and distance_to_nearest_warehouse. For order price, the median is approximately 15,000, but there are significant outliers above 35,000, indicating some unusually high orders. Similarly, the order total mirrors this distribution, with several outliers above 30,000. Delivery charges have a median of around 75, with a few outliers above 100, while coupon discount has a more uniform distribution with a median around 10% and no evident outliers. Lastly, the distance_to_nearest_warehouse shows a median distance of 1 km, with a few outliers exceeding 2.5 km. Overall, the plots highlight the central tendencies of these variables and indicate potential anomalies due to the presence of outliers in certain features like order price, order_total, and distance_to_nearest_warehouse.

Flow of Column Inspection

Below we can see how the columns were inspected and corrected. Black arrows are independent and blue shows that the fields were interconnected



Date

It was observed that the date column had obvious format errors, and all the records were later corrected to be in the standard YYYY-MM-DD format per specs.

The approach taken here focuses on identifying and correcting invalid date formats in the dataset. Initially, a **regex pattern** is defined to match the correct YYYY-MM-DD format, and rows that do not follow this pattern are flagged as invalid. These rows are displayed to help visualize the extent of the issue. Following this, a **conversion attempt** is made using pd.to_datetime to forcefully convert the entries into the correct date format, where any failed conversions (invalid dates) are marked as NaN. After validating the successful conversion, the original date column is replaced with the corrected values, and the temporary column is dropped. Finally, the clean data is saved to a CSV file for further use. This methodology ensures that all dates in the dataset conform to the expected format, helping to improve data quality.

Geographical Coordinate

Here there were invalid coordinates that were observed to have apparently been swapped. The methodology here involves validating and correcting geographic coordinates (latitude and longitude). First, the **valid range** for latitude (-90 to 90) and longitude (-180 to 180) is checked, and rows with invalid values are flagged and displayed for review. To correct potential errors, a **function** is defined to swap latitude and longitude values when latitude is out of range, but longitude is valid, as this indicates a likely mix-up. The function is then applied across the dataset to swap the coordinates where necessary. After making the corrections, the updated latitude and longitude values are displayed for validation. This approach ensures that all geographic coordinates fall within their valid ranges, improving the accuracy of location data.

Warehouse Names

Here we consolidated and standardised the names, The methodology in this code focuses on **normalising text data** in the nearest warehouse column. Initially, all values in the nearest warehouse column are converted to lowercase to ensure consistent capitalization across the dataset. This helps standardise warehouse names that may differ only in capitalization. After normalisation, the unique warehouse names are counted and compared to their original forms. A **grouping operation** is performed to display the original warehouse names alongside their normalised versions, showing how different capitalizations have been consolidated. Finally, the original nearest warehouse column is replaced with the normalised values, and the changes are verified by displaying the unique, standardised warehouse names. This approach ensures consistency in warehouse names, reducing potential errors caused by case sensitivity in the dataset.

Distance to Warehouses

Since this is directly linked to the previous two parts which are the customer coordinates and the warehouses, I have continued to this part with following approach-

- Create function haversine () to calculate the distance between two geographic points using the haversine formula.
- Iterate through a defined function compute_nearest_warehouse_distance () and find the smallest distance.
- Validate newly computed columns with existing and make corrections.

Shopping Cart

- **Check for Missing or Null Values**: The code identifies rows where the shopping cart is either missing (NaN) or empty (i.e., an empty string), which can lead to data quality issues.
- **Detect Invalid Quantities**: A regular expression is used to extract quantities from the shopping cart, and any invalid (negative or zero) quantities are flagged as problematic.
- **Identify Improperly Formatted Carts**: The code searches for shopping carts that do not contain valid list structures (e.g., missing brackets or commas), as this indicates structural inconsistencies in the data.
- **Check for Duplicate Products**: It checks for rows where the same product appears multiple times in a shopping cart, indicating a duplicate entry issue.
- **Combine Duplicate Products**: Using a custom function, duplicate products in the cart are combined by summing their quantities. This ensures that instead of having multiple entries for the same product, the shopping cart reflects a single entry with the correct quantity.

This step-by-step approach ensures that the shopping cart column is cleaned and normalized by addressing missing values, invalid quantities, improper formatting, and duplicate entries. The cleaned shopping cart column is then displayed to verify the changes made.

Expedite

I then used the newly calculated distances along with the latest customer reviews to check for errors in the 'is_expedited_delivery' column.
I first got the range of the distances and set a threshold of 1km from the warehouse and proceeded to do a sentiment analysis on the reviews. This code integrates sentiment analysis and distance evaluation to identify expedited deliveries. The Text Blob library is used to analyse customer reviews, assigning a sentiment score (ranging from -1 to 1) to each review. A threshold of 1 km is set for expedited deliveries, meaning deliveries within 1 km of the nearest warehouse and with positive sentiment are flagged as expedited. The computed results are compared with the original is_expedited_delivery column, and any discrepancies are corrected. Finally, the corrected rows are displayed for verification.

Happy Customer

Since this also requires sentiment analysis I have called on previously defined functions for this column and validated it against existing columns to make corrections.

Shopping Cart

This methodology focuses on **removing duplicate products** from the shopping_cart column by combining their quantities. The combine duplicate's function uses ast. literal_eval to convert the string representation of the shopping cart into a list of tuples (product, quantity). A dictionary is used to sum up the quantities for duplicate products. After processing, the list of combined products and their quantities is reconstructed and applied to the dataset. The shopping_cart column is updated to reflect the corrected data, ensuring that each product appears only once with the correct total quantity.

Order Price

This is connected to the shopping cart that has been corrected.
This methodology is connected to the **shopping_cart** by calculating and validating the **unit price** and **order price** based on the product quantities in the cart.

Key Steps:

1. Calculate Unit Price:

    a. The function calculate_unit_price converts the shopping_cart string into a list of tuples (product, quantity) and sums the quantities of all products.

    b. The **unit price** is calculated by dividing the order_price by the total quantity of items in the cart.

    c. The result is stored in a new column unit_price.

2. Recalculate Order Price:

    a. The recalculate_order_price function uses the unit_price and multiplies it by the quantities in the shopping_cart to compute the **total order price**.

    b. This calculated price is stored in a new column computed_order_price, allowing comparison with the original order_price.

3. Connection to the Cart:

   a. The **shopping_cart** data is integral to both functions, as it provides the quantities needed to calculate the **unit price** and verify the accuracy of the **order price**.

   b. The process ensures that the order_price is consistent with the product quantities and unit prices derived from the cart.

Order Total

This is connected to prior two fields and has been approached as below-

1. Calculate Order Total:

   a. The function compute_order_total calculates the **discounted order price** by applying the coupon discount to the original order_price.

   b. It then adds the **delivery charges** to the discounted price to compute the final **order total**.

   c. The computed total is stored in a new column, computed_order_total.

2. Validation:

   a. The computed order total is compared against the existing order_total in the dataset.

   b. Rows where the original order_total does not match the computed value are flagged as having inconsistencies.

3. Correction:

   a. The incorrect order_total values are replaced with the computed_order_total values in the rows where discrepancies are found.

4. Connection:

This process ensures that the **order total** is accurately computed based on the **coupon discount** and **delivery charges**. Any mismatches between the original and computed totals are corrected to maintain data consistency.
Finally, we save all corrected columns after dropping computed as dirty_corrected_data.csv

Outlier Data

In addition to evaluating a simple predictive model for forecasting delivery charges based on order pricing, this research illustrates how to identify and manage outliers in the dataset for the reflected report. Below is a summary of the essential actions and findings to incorporate into your report:

3.1 Cleaning and Loading Data

Data Loading:

Pandas is used to load the outlier data, and missing values are examined to guarantee data completeness.

Missing Values:

Finding any gaps in the dataset that could influence further research is made easier by first checking for missing values. Any missing information is noted and shown here.

Summary Statistics:

Descriptive statistics, such as mean, standard deviation, and range, which show the distribution and central tendency of each column, provide a preliminary knowledge of the dataset's structure.

3.2. Delivery Charges

Boxplot Evaluation

Outliers can be found by using a boxplot to visualise the 'delivery_charges' column. Boxplots are useful for determining the data's distribution, central tendency, and any outliers. Here, the boxplot shows several outliers outside the interquartile range, particularly in the delivery charges exceeding 120 or below 60.

Delivery Charges Summary Statistics

Finding the mean, median, and quartiles for "delivery_charges" further demonstrates the data's central tendency and distribution while emphasizing the degree to which outliers diverge from average values.

3.2.1 Using Interquartile Range (IQR) to Identify Outliers

IQR computation

One popular technique for identifying outliers is to compute the interquartile range (IQR). It uses the division of data into quartiles to produce a statistical measure of variability. Outliers are defined as values that are not 1.5 times the IQR from Q1 or Q3.

Outlier Removal

By removing outliers from the dataset, extreme values that could skew analysis are eliminated, producing cleaner data for subsequent research.

3.2.2 Linear Regression Model:

Modelling Delivery Fees:

Based on the order price, a linear regression model is used to forecast delivery fees. The relationship between the independent variable (order price) and the dependent variable (delivery charges) is modelled using linear regression.

Performance of the Model:

- One way to quantify prediction error is via the Mean Squared Error (MSE). The MSE of 307.24 in this case indicates a moderate degree of prediction error.
- The R-squared value is -0.03, which suggests that order price does not seem to have a substantial impact on delivery prices and that the model only explains a small portion of the variability in delivery charges based on order price.

Reflection Points for the Report:

Elimination of Outliers

Consider how identifying and eliminating outliers prevented distorted results and why this is essential for accurate analysis.

Assessment of the Model

The model may not have been successful, based on the low R-squared score. Talk about the necessity to investigate more complex models or more features for improved prediction, and why basic linear regression might not always be the best option for datasets.

Visual Perspectives

Regression plots and boxplots are two examples of visualizations that are used to help understand data distribution and model performance. Consider how these illustrations facilitated analysis and helped convey conclusions.

Missing Data

Like dirty data, we have done some basic EDA on the missing file that was loaded in and focused on imputing missing values identified using missing. is null (). sum ()

We begin first by importing the warehouse data to use the warehouse coordinates.

Nearest Warehouse

This code uses the haversine formula to compute distance between geographic points by loading in the warehouse coordinates and using the formula to find the distance between the customer's location and the warehouse.

Distance to Warehouse

Since this is connected to the prior section, we used the imputed values and proceed to calculate the nearest warehouse distances.
The calculate distance () function retrieves the coordinates (latitude, longitude) of the nearest warehouse for each customer and uses the Haversine formula to calculate the distance from the customer's location.
The code also rechecks previous function distance_to_nearest_warehouse value is missing for a row. If missing, it calculates the distance using the nearest warehouse; otherwise, it retains the original value. It then imputes the values and displays them.

Happy Customer

The code performs sentiment analysis on customer reviews using **VADER** (Valence Aware Dictionary and sentiment Reasoner) from the nltk library, which classifies whether a customer is "happy" based on the sentiment of their latest review. It then initializes Sentiment Analyzer and defines its classification:

- A compound score ≥ 0.05 indicates a **positive (happy)** sentiment (1).
- A score < 0.05 indicates **neutral or negative** sentiment (0).

The function is applied to the latest_customer_review column to create a new column is_happy_customer that stores the sentiment classification (1 for happy, 0 for not happy) and then displays results.

Order Price

This code performs **order price imputation** by solving a system of equations to determine item-level unit prices and using these to estimate the total price of orders where the price is missing.

This code performs **order price imputation** by solving a system of equations to determine item-level unit prices and using these to estimate the total price of orders where the price is missing. For each order with a known order_price, the code extracts the quantities of each item in the shopping cart. This is stored in a dictionary item_quantities, where the keys are item names, and the values are lists of tuples containing the order index and quantity for that item.

Then we build a system of equations in a matrix and solve to get unit price and then impute the missing order prices that are displayed.

Delivery Charge

This code imputes missing **delivery charges** using a **Ridge regression model** and ensures **coupon discounts** are applied correctly to the order_price. First, the dataset is cleaned, and key features such as distance_to_nearest_warehouse, is_expedited_delivery, and coupon_discount are used along with **one-hot encoded** categorical data (season). **Polynomial features** (degree 2) are added to capture interactions between variables, and the data is **scaled** for consistency. A **GridSearchCV** is used to find the best hyperparameter (alpha) for the **Ridge regression model**, which is then trained and evaluated using the **R2 score**. If the R2 score exceeds a threshold (0.97), the model is applied to predict missing delivery_charges. Additionally, a function ensures the **coupon discount** is correctly applied to the order_price. The updated dataset is saved and reviewed.

Key Points:

- **Polynomial features** are used to capture interactions between variables.
- **Ridge regression** with **GridSearchCV** is employed to find the best model for imputing delivery charges.
- **Coupon discount** is correctly applied to order_price where applicable.

Task 2

1. Overview

Reflecting on the process of feature transformation, data cleaning, and developing a linear regression model to forecast median home prices is the aim of this project. Managing a suburban information dataset containing characteristics including the number of homes, apartments, residents, median income, and more was part of the project. This reflective report is organised to offer process insights, consider difficulties and feedback, and talk about potential future enhancements to improve the model's functionality.

2. Feedback

Feedback on the performance of the linear regression model, feature engineering, and data cleaning procedures was obtained during the development phase. Important points were as follows:

Data cleaning: Problems were found with treating non-numeric symbols in the dataset, such as dollar signs in the'median_income' and'median_house_price' columns and percentage signs in the 'aus_born_perc' column. Transformations that changed them into numerical values were effective in addressing these. Feedback, however, suggested that additional verifications of the data's consistency were required, particularly about outliers and missing numbers.

Model performance: The large mean squared error (MSE) and low R-squared (0.0069) of the regression model indicated that it did not perform well in forecasting home prices. This suggested that the associations between the selected attributes and home values were not adequately represented.

Feature engineering: To increase predicted accuracy, more input recommended investigating non-linear models and adding more pertinent characteristics, like geographic location and accessibility to amenities. The data's intricate patterns were difficult for the linear model to identify.

3. Reflection

This effort brought us to light several crucial lessons:

Data cleaning: At first, the main difficulties were managing symbols and guaranteeing numerical consistency throughout the collection. The dataset was successfully transformed by the cleaning procedure, however after some thought, outlier detection should have received more attention, especially in strongly skewed columns like "number_of_houses" and "median_house_price."

Skewness and transformations: After skewness in columns such as 'number_of_houses' and 'population' was investigated, log transformations were used to help lessen skewness. Even though this crucial step increased the characteristics' normality, the linear regression model continued to perform poorly. I discovered that some characteristics might still need more intricate handling or the insertion of new variables even after changes.

Model limitations: Despite being straightforward and easy to understand, linear regression was unable to fully capture the underlying relationships between the target variable and the features. The low R-squared value showed that the model explained very little volatility in home prices, and the high MSE suggested that it was far from reliable. After some thought, it became evident that a linear model might not be appropriate for this specific dataset and that more sophisticated modelling approaches had to have been considered.

4. Future Improvement

The following enhancements are suggested for the future to solve the constraints noted:

Adding additional features: Stronger predictors for median house prices may be found in other attributes that record more specific information about the suburbs, such as crime rates, closeness to schools, or distance to public transportation. The performance of the model can be improved by extracting and engineering these features.

Advanced models: Predictive accuracy may be increased by investigating non-linear models like decision trees and ensemble techniques (Random Forest, Gradient Boosting) or advanced regression techniques like polynomial regression, Ridge and Lasso regression for regularisation, or both. These models are more capable of managing intricate feature associations.

Managing outliers and missing data: Although some data cleaning procedures were followed, more reliable techniques for managing outliers and missing values ought to be used. Methods like employing k-nearest neighbours or imputing missing variables with median values could increase the robustness of the model. Highly skewed data distributions may be easier to handle with the use of outlier discovery techniques like Z-score analysis and the IQR (Interquartile Range) method.

Feature scaling: Although scaling methods and log transformations were employed, more research into data transformations, such as power transformations, may be able to stabilise variance and lessen skewness for features that don't respond well to logarithmic scaling.

5. Conclusion

The project offered insightful information about the feature transformation, model building, and data cleaning processes. The regression model's performance was below ideal, though, suggesting that future iterations will need a more sophisticated strategy. To increase forecast accuracy, additional information must be added, sophisticated models must be investigated, and data processing methods must be improved. We were able to pinpoint important areas for improvement thanks to this reflective process, and we have no doubt that the knowledge We have gained from this experience will be useful in future endeavours.

Documentation

https://colab.research.google.com/drive/1abtwZna7Uo-Rn820IBenumuG4oxc7cgT?usp=sharing – Bhavna Missing data
https://colab.research.google.com/drive/1hbe9rY7OWXY-CT-ZU0fbuO_de4kkvYVu?usp=sharing – Subbu Missing Data
https://colab.research.google.com/drive/1SoppwPM57FOqajFLYgY_cXxxyge2dyva?authuser=1 – Task 1
https://colab.research.google.com/drive/1k_yQfLnTZL6wFvuNwI6Av1pBNJr7QYbe?authuser=1#scrollTo=m_o7PZ6jPTVV – Task 2