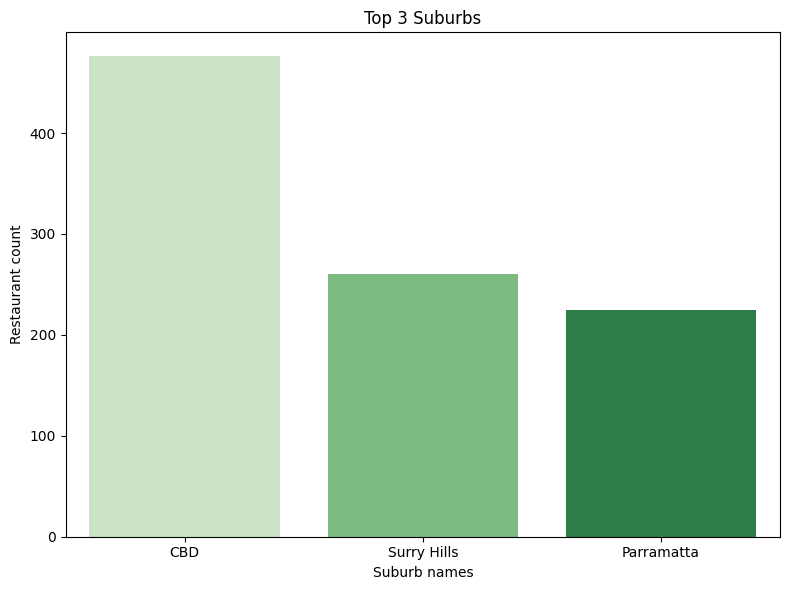
Github link: <https://github.com/ARAVIND341/ASSIGNMENT>

Part A – Exploratory Data Analysis

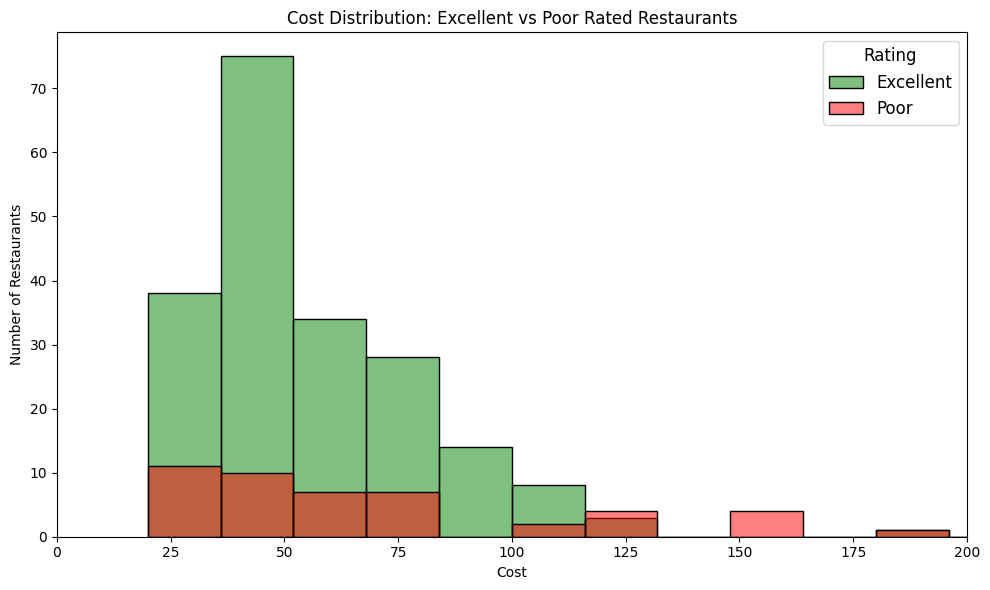
The dataset we imported here wad the Zomato\_df\_final\_data.csv and we loaded it using the Pandas library. So, we made some initial look into it by analyzing the rows, columns, datatypes, structure using df.info() and summary statistics using the df.describe(). We also used he Seaborn and Matplotlib libraries for visualization steps. In the summary statistics step, we could identify the count, mean, min, max etc for each column.

We then identified the missing values using df.isnull().sum() and did necessary steps for imputation and cleaning process for dropping or imputing the missing values.

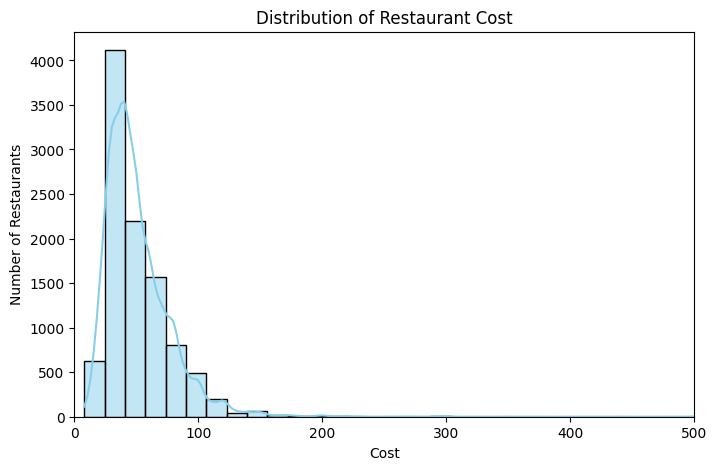
The next step we did was to identify unique cuisines using nunique() function and it was found to be 1759 unique cuisines. Then the next thing we did was to find the 3 suburbs having the most number of cuisines using bar plot and we found that the suburbs CBD , Surry hills and Parramatta was serving the most number of restaurants with a count of 476,260 and 225 respectively.



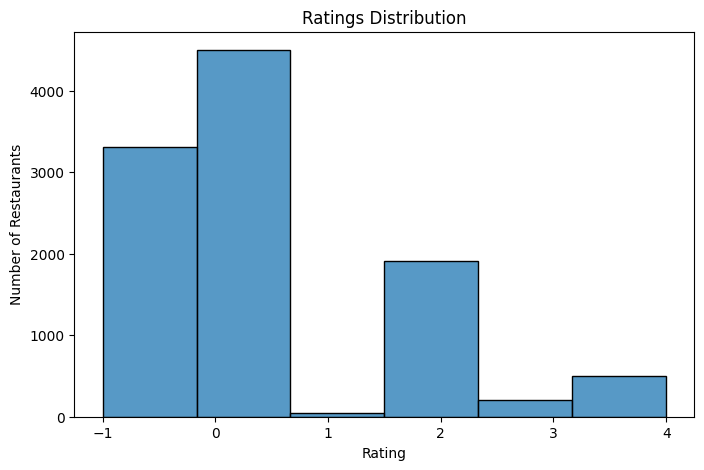
After that we did a comparison of Excellent and poor rated restaurants and found that, excellent rated restaurants tends to be more expensive than the poor rated restaurants and plotted a histogram for that.



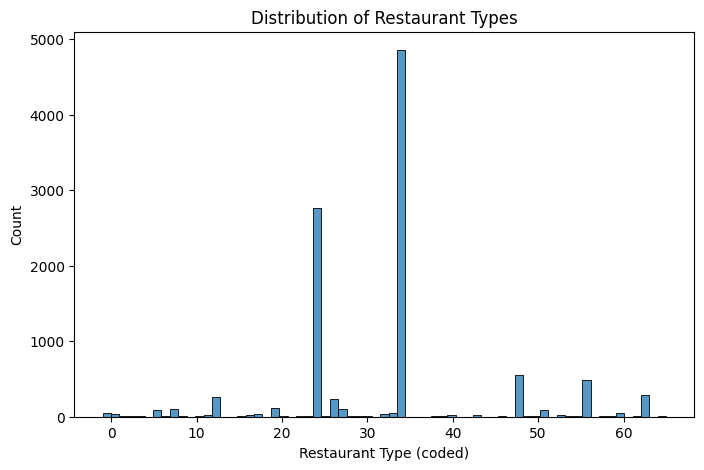
After that we plotted a histogram for the distribution of cost with cost in the x axis and number of restaurants in the y-axis. The graph showed that the most of the restaurants were in the lower to middle cost ranges



Then the bar graph for rating distribution was plotted where 0 shows excellent, 1 very good,2 good etc. and the visualization proved that most of the restaurants were having excellent ratings.

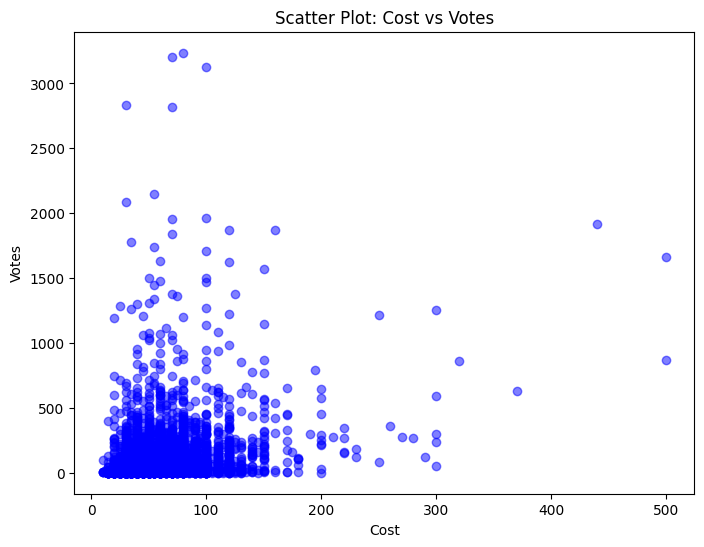


When the distribution of restaurant type were made, we encoded it into numerical codes for better analysis of the distribution. It showed the dataset were majorly dominated by the cuisines like cafes, casual dining etc.



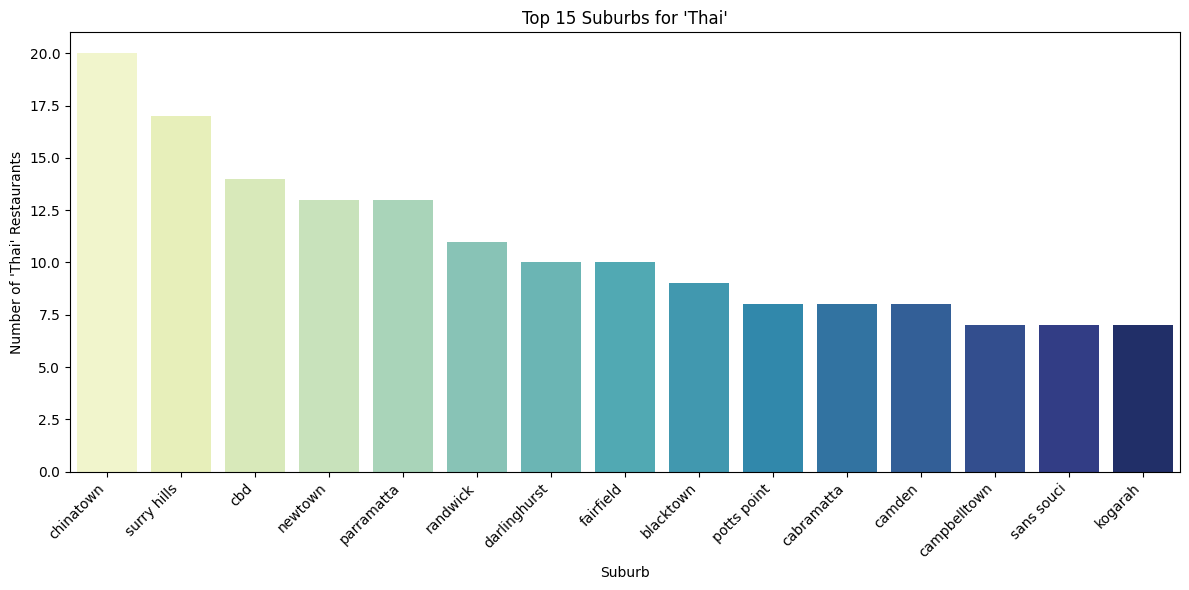
Correlation between cost and votes

Here a scatterplot was plotted to show the correlation and were found with the presence of a few outliers. Also, it was found that the higher cost restaurants are getting higher votes as well

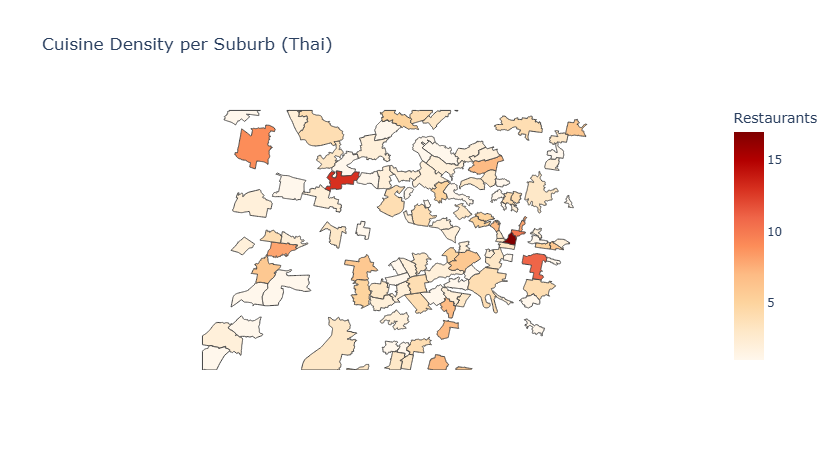


Geospatial Analysis using geopandas.

Here we used the GeoPandas for loading the Sydney.geojson file and combined it with the restaurant dataset we had by using suburb names. Also we did a normalization technique inorder to ensure proper consistency between both the datasets we are using. After that we focused on a single cuisine and obtained the count of restaurants in each suburb that served those cuisine. Here the cuisine we selected was thai. Then we created static visualization using bar graph to obtain top suburbs and displayed those through bar graphs by using the Seaborn and Matoplotlib libraries.



By overseeing the limitation caused by the thai cuisine, we moved on to interactive visualization using the Plotly and created a chloropleth maps to show the restaurant counts as it helps us to zoom, hover over the maps etc. Its more user friendly as well.



Part B -Predictive Modelling

Here in predictive modelling we undergone a cycle of Scikit-Learn pipeline where we first subjected the dataset for cleaning missing values where we treated categorical values and numeric values. After that we removed the outliers by using IQR method on cost, categorical variables like rating\_text were encoded etc.

Then we applied regression models for predicting the target variable rating\_number and classification models to predict rating class with values for Poor/Average as 1 and Good/Very good/Excellent as 2. We also did gradient descent on linear regression and obtained the desirable results.

Tables comparing model performance

* Regression models

|  |  |  |
| --- | --- | --- |
| Model | MSE | RMSE |
| Linear regression | 2.908515983930785 | 1.705437182639919 |
| Linear regression  (gradient descent) | 2.909461706211606 | 1.7057144269225157 |

* Classification models

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Confusion Matrix | Matrix | Precision | F1 |
| Random Forest | [[1465, 109], [126, 328]] | 0.751 | 0.722 | 0.736 |
| Gradient Boosted Trees | [[1490, 84],  [93, 361]] | 0.811 | 0.795 | 0.803 |
| SVM | [[1494, 80],  [100, 354]] | 0.816 | 0.78 | 0.797 |

Git/Git LFS/ DVC commands,

Git commands

* + git init – for initializing
  + git add . – for adding the files
  + git remote add origin <https://github.com/ARAVIND341/ASSIGNMENT.git>
  + git commit -m “/message/"
  + git push -u origin main – here we push the code to the main branch

Git LFS

* + git LFS install – for installing the LFS
  + git lfs track "\*.csv" – to track csv files
  + git lfs track – to see the tracked files
  + git add .gitattributes – to add the generated .gitattributes file
  + git commit -m ”/message”
  + git push otigin

DVC commands

* + dvc init – for initiating dvc
  + dvc add zomato\_df\_final\_data.csv – to load the csv file
  + dvc add zomato\_df\_final\_data.csv .gitignore – to add the .gitignore files for dvc
  + dvc remote add -d localstore ../dvcstore – for setting up the remote storage for dvcstore
  + git commit -m “configure DVC remote storage” – location to store the actual dataset
  + git add dvc.lock reports/metrics.json models/.gitignore reports/.gitignore
  + dvc push

Reflections on PySpark vs Scikit-Learn

In PySpark library, the major concentration is on the processing of bigger data. It can deal with large datasets. These datasets gets integrated properly with the Spark SQL. The usage of MLlib pipelines provides us an opportunity to undergo various processing’s at the same time across various clusters. The pipeline procedures being implemented here helps in computation that are distributed in nature. It is more scalable compared to SciKit and the cluster ready nature of the library enables it to work on large datasets. It was also providing fast results

But when I was using PySpark I had to face major challenges on the operation side like the changing of kernels, creation of virtual environments, showing jdk implementations, set up of spark session etc. Testing at a local level also took so long.

The Scikit – Learn was more easier in implementation compared to PySpark and the dataset was also fitting into it. The implementation of regression models, classification models also didn’t brought up major challenges. When the dataset is too big, it might be less useful compared to the PySpark. In terms of accuracy, Scikit was performing way better than PySpark.

One of the main setback of Scikit Learn is the limited scalability. The fit of data in its memory was seen difficult when the data set was large. The use case of PySpark to run parallel processes in the memory outweighs Scikit as it is difficult in Scikit to achieve it.