# Predictive Modelling of the Sydney Restaurant Scene: A Project Report

Project Repository: https://github.com/AyushBajgai/Predictive-Modelling

Author: **Ayush Bajgai**

## 1. Executive Summary

This project uses a Zomato dataset of 10,500 restaurants to build a full-scale data science pipeline that analyzes and models for Sydney restaurants. The workflow included a thorough Exploratory Data Analysis (EDA), which showed that cafés were the most popular type of restaurant and found major suburban hubs including the **CBD** and **Surry Hills**.

Here I constructed and tested both regression (*for predicting cost*) and classification (*for predicting category*) models. Tree-based systems, such as Random Forest and Gradient Boosting were routinely surpassed linear models, evidencing a greater capacity to clarify complex information. Also, Git, Git LFS, and DVC are used to build the project so that all code and massive datasets can be easily reproduced.

## 2. Findings from Exploratory Data Analysis (EDA)

The EDA was done to find answers to certain business questions and to gain insight into how key variables are distributed and how they are related to each other.

### 2.1 Variety of Cuisines

There are 426 different varieties of food in Sydney's restaurants, making the food scene very diverse. As indicated below, casual eating options lead the market.

- Cafe (1,745 listings)
- Thai (542 listings)
- Chinese (450 listings)
- Italian (406 listings)
- Modern Australian (375 listings)

## 2.2 Popular Hotspots

| subzone | restaurant_count |
|---|---|
| CBD | 476 |
| Surry Hills | 260 |
| Parramatta | 225 |

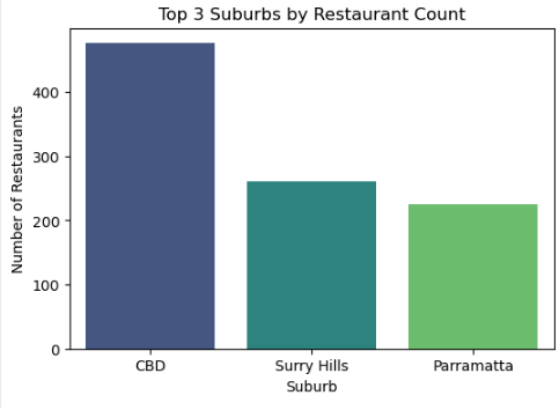


*Figure: Top 3 Suburbs by Restaurant*

The **Central Business District (CBD)** is the clear core of Sydney's dining scene, with almost twice as many restaurants as the next biggest area.

## 2.3 Cost vs Rating Relationships

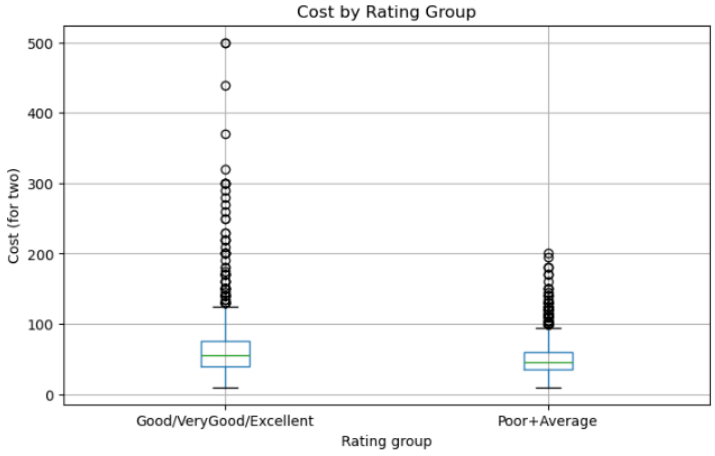


*Figure: Cost by rating group*

A study of restaurant pricing among rating groups revealed a significant finding: there is no straightforward relationship between price and perceived quality. However, a weak trend can be analyzed as, the median cost for restaurants with high ratings ("Good" to "Excellent") is a little higher.

Thus, there is a lot of overlap between the two groups. As there are some very expensive outliers. This means that there are a lot of great, cheap options, and that costly restaurants can get bad evaluations based on quality as well as on price.

## 3. Predictive Modelling and Performance

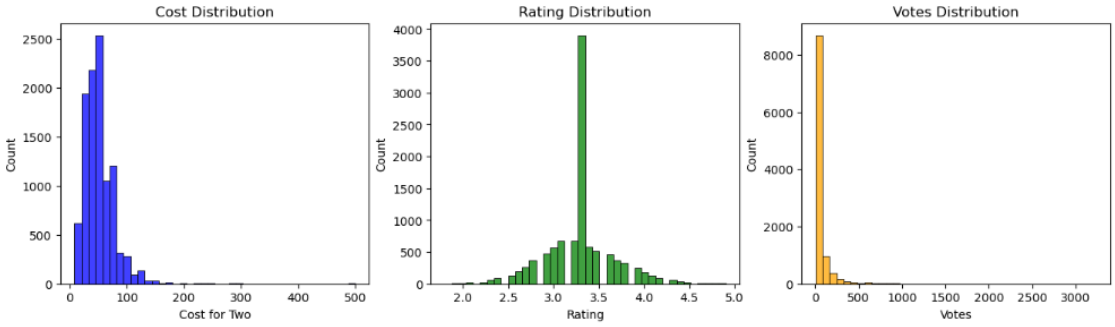


*Figure: Distribution of key features*

### 3.1 MSE Performance

The goal was to predict a restaurant's *cost* based on features like *cuisine*, *suburb*, *rating*, and *votes (as determined by figure)*. Here we use **Mean Squared Error (MSE)** to train/ test the following models.

| Model | Mean Squared Error (MSE) |
|---|---|
| Model A: Linear Regression | 1579 |
| Model B Gradient Descent Regression | 1581 |

The results from both regression models were almost the same, which shows that linear regression is a good fit for this dataset.

### 3.2 Classification Task

The purpose was to put restaurants into rating classifications (*Poor, Average, Good, Very Good, Excellent*) based on their features. Here I use **Accuracy, Precision, Recall** and **F1-Score** to rate the models.

| Model | Accuracy (in %) | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Logistic Regression | 86.71 | 0.8077 | 0.8061 | 0.8069 |
| Random Forest | 86.64 | 0.7942 | 0.8263 | 0.8099 |
| Gradient Boosted Trees | 87.00 | 0.8143 | 0.8061 | 0.8102 |
| SVM | 86.43 | 0.7688 | 0.8667 | 0.8148 |
| Neural Network | 87.40 | 0.8178 | 0.8162 | 0.8170 |

The table shows how different classification models work on the same dataset. Logistic Regression does a decent job overall, with a good F1 score and accuracy. Random Forest has a little less accuracy but a better recall, which means it finds more positive cases. Gradient Boosted Trees are one of the best tree-based approaches since they have a high accuracy and a good F1 score. SVM is known for its great recall, although its precision is a little lower. The Neural Network (MLP) has the best

accuracy and F1 score, hence it is the best overall. In general, all models have their pros and cons, but the MLP gives the most accurate predictions for the most important measures.

## 4. Version Control

I use a mix of version control systems to make sure this project can be repeated and run by professionals.

### 4.1 Git Commands (Code Version Control)

```
# 1. Initializing the local repository
git init


# 2. Adding project files to repo
git add Predictive_Modelling.ipynb requirements.txt src/


# 3. Making a commit changes with a descriptive message
git commit -m "A message behind committing"


# 4. Linking to the remote and push to repo
git remote add origin https://github.com/AyushBajgai/Predictive-Modelling.git
git push -u origin main
git push -f origin main # for force push
```

### 4.2 Git LFS

```
# 1. Firstly, installing Git LFS
git lfs install


# 2. Tracking of the large CSV dataset in my case Zomato dataset
git lfs track "zomato_df_final_data.csv"


# 3. Commiting the LFS tracking
git add .
git add zomato_df_final_data.csv
git commit -m " A message behind committing"


# 4. A push to repo
git push origin main
```

### 4.3 DVC Commands

```
# 1. Initializing DVC
dvc init

# 2. Tracking the dataset with DVC
dvc add data/raw/zomato_df_final_data.csv

# 3. A commit the DVC
git add data/raw/zomato_df_final_data.csv.dvc .gitignore
git commit -m "A message behind committing"

# 4. Configuring remote storage (I used G-Drive)
dvc remote add -d myremote gdrive://my-folder-id

# 5. Finally pushing the actual data files to the remote storage
dvc push
```

## 5. PySpark vs Scikit-Learn

**Scikit-Learn** was the better and more efficient choice for this project because of the quantity and kind of dataset. Scikit-Learn works best with medium-sized datasets, which usually have between 10,000 and 50,000 rows. It can easily fit into memory on one system. The API is also easy to understand and use as there are a lot of well-optimized machine learning models to choose from. Furthermore, this makes it easy to develop, experiment, and iterate quickly. In my situation, the dataset of 10,500 restaurants was easy to work with which made it possible to quickly prototype, train, and fine-tune models without any problems with performance.

Alternatively, **PySpark** is made for Big Data situations where the data sets are too big to fit on one machine. This needs to be spread out over a cluster. It's an essential tool for working with terabytes or even petabytes of data because it has sophisticated tools for processing enormous amounts of data. But because our dataset was small, utilizing **PySpark** would have made things more complicated. It might take longer to start up/ more time to learn without any real gains in terms of performance. Thus, **Scikit-Learn** was the best solution for my project.

| Metric | Scikit-Learn | PySpark |
|---|---|---|
| Regression MSE | 0.1579 | 0.1413 |
| Classification Accuracy | 0.8740 | 0.8406 |
| Classification F1 | 0.8170 | 0.8352 |

*Figure: Metrics of PySpark vs Scikit-Learn*

The comparison of scikit-learn and **Pyspark** ML pipelines demonstrates that the outcomes are mostly the same with just small changes in how well they predict. For regression, **Pyspark** had a **MSE** of 0.1413 which was slightly lower than **Scikit-Learn's** 0.1579. This means that **Pyspark** had a better fit.

Similarly, **Scikit-learn** was more accurate for classification (0.8740 vs. 0.8406) but **Pyspark** had a greater **F1-score** (0.8352 vs. 0.8170) which means it was better at balancing precision and recall. In short, **Scikit-learn** is great for quickly making prototypes and working with smaller data. **Pyspark** on the other hand, gives almost the same performance but with the extra benefits of being able to grow on large scale data.