Part B – Predictive Modelling

This notebook focuses on predictive modelling using the restaurant dataset.
We will cover two tasks:

1. **Regression** – predicting the cost of restaurants.
2. **Classification** – predicting rating categories.

In [3]:
```
import pandas as pd
import numpy as np

# Read data
df = pd.read_csv("data/zomato_df_final_data.csv")

# Basic check
print("Shape:", df.shape)
print("Missing values:\n", df.isnull().sum())
df.head()
```

```
Shape: (10500, 17)
Missing values:
 address             0
cost              346
cuisine             0
lat               192
link                0
lng               192
phone               0
rating_number    3316
rating_text      3316
subzone             0
title               0
type               48
votes            3316
groupon             0
color               0
cost_2            346
cuisine_color       0
dtype: int64
```

Out[3]:

	address	cost	cuisine	lat	link	
0	371A Pitt Street, CBD, Sydney	50.0	['Hot Pot', 'Korean BBQ', 'BBQ', 'Korean']	-33.876059	https://www.zomato.com/sydney/sydney-madang-cbd	151.2
1	Shop 7A, 2 Huntley Street, Alexandria, Sydney	80.0	['Cafe', 'Coffee and Tea', 'Salad', 'Poké']	-33.910999	https://www.zomato.com/sydney/the-grounds-of-a...	151.1
2	Level G, The Darling at the Star, 80 Pyrmont ...	120.0	['Japanese']	-33.867971	https://www.zomato.com/sydney/sokyo-pyrmont	151.1
3	Sydney Opera House, Bennelong Point, Circular...	270.0	['Modern Australian']	-33.856784	https://www.zomato.com/sydney/bennelong-restau...	151.2
4	20 Campbell Street, Chinatown,	55.0	['Thai', 'Salad']	-33.879035	https://www.zomato.com/sydney/chat-thai-chinatown	151.2

1. Feature Engineering

In [4]:

```
# Drop rows missing target values
df_cleaned = df.dropna(subset=["rating_number", "votes"])

# Fill missing cost with median
df_cleaned["cost"] = df_cleaned["cost"].fillna(df_cleaned["cost"].median())

# Fill missing type with mode
df_cleaned["type"] = df_cleaned["type"].fillna(df_cleaned["type"].mode()[0]

# Drop any remaining missing rows
df_cleaned = df_cleaned.dropna()

# Check result
print("After cleaning:\n", df_cleaned.isnull().sum())
print("Shape:", df_cleaned.shape)
```

```
After cleaning:
 address          0
cost             0
cuisine          0
lat              0
link             0
lng              0
phone            0
rating_number    0
rating_text      0
subzone          0
title            0
type             0
votes            0
groupon          0
color            0
cost_2           0
cuisine_color    0
dtype: int64
Shape: (6969, 17)
```

```
C:\Users\AkeyT\AppData\Local\Temp\ipykernel_22204\4005009771.py:5: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned["cost"] = df_cleaned["cost"].fillna(df_cleaned["cost"].median
())
C:\Users\AkeyT\AppData\Local\Temp\ipykernel_22204\4005009771.py:8: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned["type"] = df_cleaned["type"].fillna(df_cleaned["type"].mode()
[0])
```

Handle Missing Data

Dropped rows with missing `rating_number` or `votes`. Filled missing `cost` with median and `type` with mode. Dropped remaining NAs. Final shape: ~8,400 rows.

In [5]:
```python
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

# Separate features/target
X = df_cleaned.drop(columns=["rating_number", "rating_text"])
y = df_cleaned["rating_number"]

# Identify columns
num_cols = X.select_dtypes(include=["int64", "float64"]).columns.tolist()
cat_cols = X.select_dtypes(include=["object", "category"]).columns.tolist()

# Define pipelines
num_tf = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])
```

```
cat_tf = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("ohe", OneHotEncoder(handle_unknown="ignore"))
])

# Combine
preprocessor = ColumnTransformer([
    ("num", num_tf, num_cols),
    ("cat", cat_tf, cat_cols)
])
```

Categorical Encoding

We applied One-Hot Encoding to all categorical features, with missing values imputed using the most frequent category. Numerical columns were scaled after imputing median values. This step prepares the data for regression/classification models.

2. Regression Models

In [6]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Combine preprocessor + model
model_a = Pipeline([
    ("preprocess", preprocessor),
    ("lr", LinearRegression())
])

# Train
model_a.fit(X_train, y_train)

# Predict
y_pred = model_a.predict(X_test)

# Evaluate
mse_a = mean_squared_error(y_test, y_pred)
print("Model A (LinearRegression/sklearn) MSE:", round(mse_a, 4))
```

```
Model A (LinearRegression/sklearn) MSE: 0.0071
```

About moudle A We trained a linear regression model to predict `rating_number` using Scikit-Learn. The pipeline included preprocessing (imputation, scaling, encoding) and a linear regression model. The dataset was split into 80% training and 20% testing.

Performance:

- **Mean Squared Error (MSE)**: 0.0071

This low MSE indicates that the model performs reasonably well in predicting average ratings, though further validation is needed.

In [ ]:
```python
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Select features (exclude target)
X = df_cleaned[["votes", "cost"]]
y = df_cleaned["rating_number"]

# Split train/test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Scale features for stability
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Add bias term
X_train_b = np.c_[np.ones((X_train_scaled.shape[0], 1)), X_train_scaled]
X_test_b = np.c_[np.ones((X_test_scaled.shape[0], 1)), X_test_scaled]

# Initialize parameters
theta = np.random.randn(X_train_b.shape[1], 1)

# Hyperparameters
lr = 0.01      # smaller learning rate
n_iter = 5000  # more iterations
m = len(X_train_b)

# Gradient descent loop
y_train_np = y_train.values.reshape(-1, 1)
for i in range(n_iter):
    gradients = 2/m * X_train_b.T @ (X_train_b @ theta - y_train_np)
    theta -= lr * gradients

# Predictions and MSE
y_pred = X_test_b @ theta
mse_gd = np.mean((y_pred.flatten() - y_test.values)**2)

print("Model B (Gradient Descent) MSE:", round(mse_gd, 4))
```

```
Model B (Gradient Descent) MSE: 0.161
```

3. Classification Models

In [ ]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Copy data
df_cls = df_cleaned.copy()

# Map rating_text to binary class
class_map = {
    'Poor': 0,
    'Average': 0,
    'Good': 1,
    'Very Good': 1,
    'Excellent': 1
}
df_cls["rating_binary"] = df_cls["rating_text"].map(class_map)

# Drop rows with unmapped values (if any)
```

```
df_cls = df_cls.dropna(subset=["rating_binary"])

# Select features and target
X = df_cls[["votes", "cost"]]  # You can add more features later
y = df_cls["rating_binary"].astype(int)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

In [ ]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, precision_score, recall_score,

# Train logistic regression model
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

# Predict
y_pred = logreg.predict(X_test)

# Evaluation metrics
cm = confusion_matrix(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print results
print("Confusion Matrix:\n", cm)
print(f"Precision: {precision:.3f}")
print(f"Recall:    {recall:.3f}")
print(f"F1 Score:  {f1:.3f}")
print("\nDetailed Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[860  47]
 [180 307]]
Precision: 0.867
Recall:    0.630
F1 Score:  0.730

Detailed Report:
               precision    recall  f1-score   support

           0       0.83      0.95      0.88       907
           1       0.87      0.63      0.73       487

    accuracy                           0.84      1394
   macro avg       0.85      0.79      0.81      1394
weighted avg       0.84      0.84      0.83      1394
```

Logistic Regression Results Analysis

The logistic regression model achieved an overall accuracy of **84%**, with a precision of **0.867**, recall of **0.630**, and F1 score of **0.730** for the positive class ("Good", "Very Good", "Excellent"). The confusion matrix shows the model performs well in identifying low-rated restaurants (Class 0) with high recall (0.95), but is less effective at detecting high-rated ones, with more false negatives and a lower recall (0.63). This indicates the model is more conservative in assigning the positive class and may underpredict highquality restaurants. Overall, the model demonstrates reasonable performance and serves as a solid baseline.

In [ ]:
```
# Create 'cuisine_diversity' = count of distinct cuisines
df_binary["cuisine_diversity"] = df_binary["cuisine"].apply(lambda x: len(st

# Create 'num_cuisines' = same as cuisine_diversity
df_binary["num_cuisines"] = df_binary["cuisine_diversity"]
```

In [ ]:
```
features = ["cost", "votes", "cuisine_diversity", "num_cuisines"]
X = df_binary[features]
y = df_binary["rating_class"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

In [ ]:
```
# Train and compare three classifiers (handles missing values and scales fea
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassi
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, 

# Prepare data
features = ["cost", "votes", "cuisine_diversity", "num_cuisines"]
X = df_binary[features].copy()
y = df_binary["rating_class"].copy()

# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# Impute missing values (fit on train only)
imp = SimpleImputer(strategy="median")
X_train_imp = pd.DataFrame(imp.fit_transform(X_train), columns=X_train.colum
X_test_imp  = pd.DataFrame(imp.transform(X_test), columns=X_test.columns, in

# Scale features (fit on train only)
scaler = StandardScaler()
X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train_imp), columns=X_t
X_test_scaled  = pd.DataFrame(scaler.transform(X_test_imp), columns=X_test_i

# Models to train
models = {
    "Random Forest": RandomForestClassifier(random_state=42, n_jobs=-1),
    "Gradient Boosted Trees": GradientBoostingClassifier(random_state=42),
    "SVM": SVC(kernel="rbf", probability=True, random_state=42)
}

# Train, predict and collect metrics
results = []
for name, model in models.items():
    # Use scaled data for all models (scaling is benign for tree models)
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    results.append({
        "Model": name,
        "Accuracy": round(accuracy_score(y_test, y_pred), 3),
        "Precision": round(precision_score(y_test, y_pred), 3),
        "Recall": round(recall_score(y_test, y_pred), 3),
```

```
        "F1 Score": round(f1_score(y_test, y_pred), 3)
    })

# Show comparison
comparison_df = pd.DataFrame(results).sort_values("F1 Score", ascending=Fals
print(comparison_df)
```

```
                    Model  Accuracy  Precision  Recall  F1 Score
0  Gradient Boosted Trees     0.870      0.809   0.814     0.812
1                     SVM     0.858      0.834   0.733     0.781
2           Random Forest     0.839      0.786   0.729     0.757
```

Classification Model Comparison and Analysis

Three classification models were trained to predict simplified restaurant rating classes using selected features (`cost` , `votes` , `cuisine_diversity` , `num_cuisines`). Based on the evaluation metrics, **Gradient Boosted Trees** outperformed the other two models in terms of overall performance, achieving the highest accuracy (0.870) and F1 score (0.812). It effectively balanced precision and recall, making it a strong candidate for deployment.

The **SVM model** also performed well, particularly in precision (0.834), but had slightly lower recall, which affected its F1 score. **Random Forest** showed the lowest performance among the three, although still acceptable, with a relatively lower recall (0.729), indicating it missed more positive samples.

Overall, Gradient Boosted Trees provided the most balanced and reliable classification performance on this dataset. However, its training speed and scalability may be a concern for larger datasets, which will be further evaluated in the PySpark comparison.