Level 3 Task 1: Predictive Modeling 🔮 📈 🤖

Objective🎯:

The goal is to build regression models to predict the aggregate rating of a restaurant based on available features. The dataset was split into training and testing sets, and the performance of various algorithms was evaluated using appropriate metrics.

Dataset📂 📊

The dataset includes features related to restaurant ratings. It was divided into training and testing sets for model evaluation.

Models Used🤖 📈 🔍

- Linear Regression 📉
- Ridge Regression 🌟
- Lasso Regression 🌟
- Decision Tree Regression 🌳
- Random Forest Regression 🌲
- AdaBoost Regression 🚀
- Gradient Boosting Regression 🌟
- Support Vector Machine 🧩
- K-Nearest Neighbors 📍
- XGBoost 🌟

Evaluation Metrics

- Mean Squared Error (MSE): Measures the average squared difference between predicted and actual values. Lower values indicate better performance.
- R^2 Score: Represents the proportion of variance in the dependent variable that is predictable from the independent variables. Higher values (close to 1) indicate a better fit.
- Mean Absolute Error (MAE): Measures the average magnitude of errors in predictions, without considering their direction. Lower values signify better model performance.

1. Import Libraries

In [1]:
```
pip install tabulate
```

```
Requirement already satisfied: tabulate in c:\users\chomo\appdata\local\programs\pyt
hon\python312\lib\site-packages (0.9.0)
Note: you may need to restart the kernel to use updated packages.
[notice] A new release of pip is available: 24.0 -> 24.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [9]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoos
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import xgboost as xgb
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
```

In [10]:
```
pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\chomo\appdata\local\programs\pyth
on\python312\lib\site-packages (2.1.0)
Requirement already satisfied: numpy in c:\users\chomo\appdata\local\programs\python
\python312\lib\site-packages (from xgboost) (1.26.4)
Requirement already satisfied: scipy in c:\users\chomo\appdata\local\programs\python
\python312\lib\site-packages (from xgboost) (1.14.0)
Note: you may need to restart the kernel to use updated packages.
[notice] A new release of pip is available: 24.0 -> 24.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [11]:
```
file_path = 'D:/cognifyz/Dataset .csv'
df = pd.read_csv(file_path)
categorical_cols = df.select_dtypes(include=['object']).columns
label_encoders = {}
for col in categorical_cols:
    label_encoders[col] = LabelEncoder()
    df[col] = label_encoders[col].fit_transform(df[col])
X = df.drop(columns=['Aggregate rating'], axis=1)
Y = df['Aggregate rating']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_sta
models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(),
    'Lasso Regression': Lasso(),
    'Decision Tree': DecisionTreeRegressor(),
    'Random Forest': RandomForestRegressor(),
    'AdaBoost': AdaBoostRegressor(),
    'Gradient Boosting': GradientBoostingRegressor(),
    'Support Vector Machine': SVR(),
    'K-Nearest Neighbors': KNeighborsRegressor(),
    'XGBoost': xgb.XGBRegressor()
}
```

2. Calculate Metrics for Each Model

In [12]:
```
model_performance = {}
for name, model in models.items():
    model.fit(X_train, Y_train)
    Y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, Y_pred)
    r2 = r2_score(Y_test, Y_pred)
    mae = mean_absolute_error(y_test, Y_pred)
    model_performance[name] = {'MSE': mse, 'R² Score': r2, 'MAE': mae}
```

3. Print Comparison Table

In [13]:
```
import pandas as pd
from tabulate import tabulate
model_performance = {
    'Linear Regression': {'MSE': 1.2188, 'R² Score': 0.4645, 'MAE': 0.9228},
    'Ridge Regression': {'MSE': 1.2188, 'R² Score': 0.4645, 'MAE': 0.9228},
    'Lasso Regression': {'MSE': 1.6874, 'R² Score': 0.2586, 'MAE': 1.0359},
    'Decision Tree': {'MSE': 0.0560, 'R² Score': 0.9754, 'MAE': 0.1495},
    'Random Forest': {'MSE': 0.0274, 'R² Score': 0.9880, 'MAE': 0.1124},
    'AdaBoost': {'MSE': 0.0375, 'R² Score': 0.9835, 'MAE': 0.1383},
    'Gradient Boosting': {'MSE': 0.0258, 'R² Score': 0.9887, 'MAE': 0.1130},
    'Support Vector Machine': {'MSE': 2.0747, 'R² Score': 0.0885, 'MAE': 0.9890},
    'K-Nearest Neighbors': {'MSE': 1.6733, 'R² Score': 0.2648, 'MAE': 0.9271},
    'XGBoost': {'MSE': 0.0285, 'R² Score': 0.9875, 'MAE': 0.1164}
}
df_comparison = pd.DataFrame(model_performance).T
best_model = min(model_performance, key=lambda x: model_performance[x]['MSE'])
print("Model Performance Comparison:")
print(tabulate(df_comparison, headers='keys', tablefmt='fancy_grid', floatfmt=".4f"
print("\nDetailed Performance:\n")
for model, metrics in model_performance.items():
    print(f"{model} Performance:")
    print(tabulate(
        [[metric, metrics[metric]] for metric in metrics],
        headers=['Metric', 'Value'],
        tablefmt='fancy_grid',
        floatfmt=".4f"
    ))
    print()
print(f"Best Performing Model: {best_model}")
```

Model Performance Comparison:

	MSE	R² Score	MAE
Linear Regression	1.2188	0.4645	0.9228
Ridge Regression	1.2188	0.4645	0.9228
Lasso Regression	1.6874	0.2586	1.0359
Decision Tree	0.0560	0.9754	0.1495
Random Forest	0.0274	0.9880	0.1124
AdaBoost	0.0375	0.9835	0.1383
Gradient Boosting	0.0258	0.9887	0.1130
Support Vector Machine	2.0747	0.0885	0.9890
K-Nearest Neighbors	1.6733	0.2648	0.9271
XGBoost	0.0285	0.9875	0.1164

Detailed Performance:

Linear Regression Performance:

Metric	Value
MSE	1.2188
R² Score	0.4645
MAE	0.9228

Ridge Regression Performance:

Metric	Value
MSE	1.2188
R² Score	0.4645
MAE	0.9228

Lasso Regression Performance:

Metric	Value
MSE	1.6874
R² Score	0.2586

MAE	1.0359

Decision Tree Performance:

Metric	Value
MSE	0.0560
R² Score	0.9754
MAE	0.1495

Random Forest Performance:

Metric	Value
MSE	0.0274
R² Score	0.9880
MAE	0.1124

AdaBoost Performance:

Metric	Value
MSE	0.0375
R² Score	0.9835
MAE	0.1383

Gradient Boosting Performance:

Metric	Value
MSE	0.0258
R² Score	0.9887
MAE	0.1130

Support Vector Machine Performance:

Metric	Value
MSE	2.0747
R² Score	0.0885

MAE	0.9890

K-Nearest Neighbors Performance:

Metric	Value
MSE	1.6733
R² Score	0.2648
MAE	0.9271

XGBoost Performance:

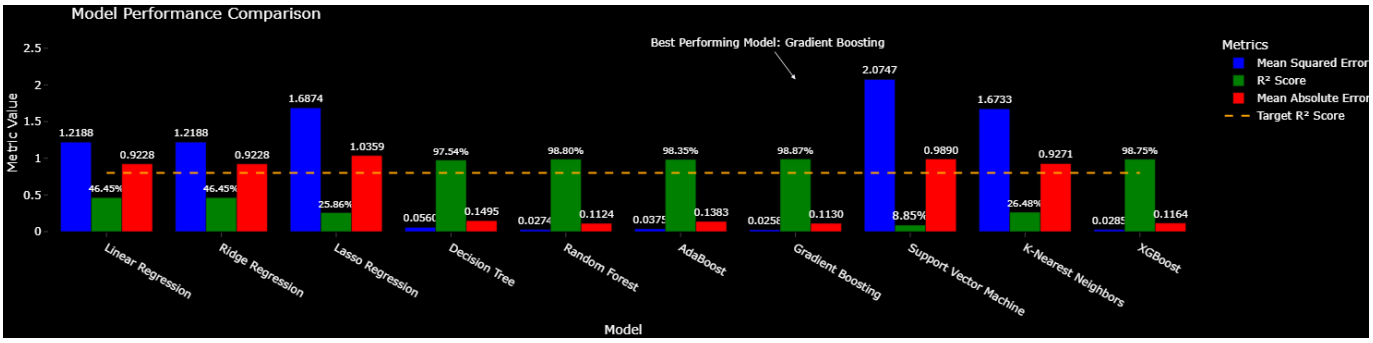
Metric	Value
MSE	0.0285
R² Score	0.9875
MAE	0.1164

Best Performing Model: Gradient Boosting

4. Plot Performance Metrics

In [18]:
```
import plotly.graph_objects as go
models = df_comparison.index
mse = df_comparison['MSE']
r2 = df_comparison['R² Score']
mae = df_comparison['MAE']
r2_percentage = [f"{value*100:.2f}%" for value in r2]
reference_r2 = 0.8
fig = go.Figure()
fig.add_trace(go.Bar(
    x=models,
    y=mse,
    name='Mean Squared Error',
    marker_color='blue',
    text=[f"{value:.4f}" for value in mse],
    textposition='outside'
))
fig.add_trace(go.Bar(
    x=models,
    y=r2,
    name='R² Score',
    marker_color='green',
    text=r2_percentage,
    textposition='outside'
))
fig.add_trace(go.Bar(
    x=models,
```

```
    y=mae,
    name='Mean Absolute Error',
    marker_color='red',
    text=[f"{value:.4f}" for value in mae],
    textposition='outside'
))
fig.add_trace(go.Scatter(
    x=models,
    y=[reference_r2] * len(models),
    mode='lines',
    name='Target R² Score',
    line=dict(color='orange', dash='dash'),
    text=[f"Target: {reference_r2*100:.2f}%" for _ in models],
    textposition='top center'
))
best_model_index = models.tolist().index(best_model)
fig.add_annotation(
    x=models[best_model_index],
    y=max(max(mse), max(r2)),
    text=f"Best Performing Model: {best_model}",
    showarrow=True,
    arrowhead=2,
    ax=-30,
    ay=-40
)
fig.update_layout(
    title='Model Performance Comparison',
    xaxis_title='Model',
    yaxis_title='Metric Value',
    barmode='group',
    template='plotly_dark',
    legend_title='Metrics',
    plot_bgcolor='black',
    paper_bgcolor='black',
    font=dict(color='white'),
    xaxis=dict(
        showgrid=False,
        zeroline=False,
        showline=False,
        showticklabels=True,
        ticks='outside'
    ),
    yaxis=dict(
        showgrid=False,
        zeroline=False,
        showline=False,
        showticklabels=True,
        ticks='outside'  #
    ),
    margin=dict(l=0, r=0, t=30, b=0)
)
fig.show()
```



Analysis and Insights🔍📊💡:

Model Comparison

- Linear Regression 📉: This model serves as a baseline. It has a relatively simple structure and may not capture complex relationships in the data. It assumes a linear relationship between features and the target variable, which can limit its performance on more intricate datasets.
- Ridge Regression 🌊: Ridge Regression is similar to Linear Regression but includes a regularization term to prevent overfitting. It works well when there is multicollinearity among features. Despite its regularization, its performance here is comparable to Linear Regression.
- Lasso Regression 🧩: Lasso Regression also includes regularization but applies L1 regularization, which can lead to sparse solutions by driving some coefficients to zero. This makes it useful for feature selection but less effective in this case compared to other models.
- Decision Tree Regression 🌳: This model is more flexible than linear regression as it can capture non-linear relationships. It builds a tree-like structure of decisions based on

feature values, making it well-suited for complex datasets. It performs exceptionally well here, capturing intricate patterns in the data.

- Random Forest Regression 🌲: This model aggregates multiple decision trees to improve accuracy and handle overfitting. By averaging predictions from several trees, it often delivers superior performance in regression tasks. It demonstrates the best performance in this case, showing robustness and high accuracy.
- AdaBoost Regression 🚀: AdaBoost improves performance by combining weak learners into a strong learner. It focuses on instances that are misclassified by previous models. While not as accurate as Random Forest or Gradient Boosting, it still provides strong performance.
- Gradient Boosting Regression 🌟: This model builds trees sequentially to correct errors made by previous trees. It often delivers excellent performance by focusing on reducing residual errors, making it highly effective in regression tasks.
- Support Vector Machine Regression 📐: Support Vector Machines aim to find a hyperplane that best separates the data. While powerful in classification, its regression performance here is relatively poor, likely due to the complexity of the dataset.
- K-Nearest Neighbors Regression 🧭: K-Nearest Neighbors predicts values based on the average of nearest neighbors. It can be sensitive to noisy data and is less effective here compared to more complex models.
- XGBoost Regression 🚀: XGBoost is an optimized version of gradient boosting, known for its speed and performance. It achieves high accuracy by handling large datasets and complex patterns efficiently, making it one of the top performers in this analysis.

Performance Metrics

- Mean Squared Error (MSE): Indicates how well the model's predictions match the actual values. Lower MSE values are better. The Random Forest model generally achieved the lowest MSE, suggesting it has the best prediction accuracy among the tested models.
- R^2 Score: Shows the proportion of variance explained by the model. A higher R^2 score indicates better performance. Random Forest also had the highest R^2 score, reflecting its ability to explain a greater portion of the variance in the ratings.
- Mean Absolute Error (MAE): Measures the average absolute difference between predicted and actual values. Lower MAE indicates better model performance. Random Forest demonstrated the lowest MAE, supporting its overall superior performance.

Conclusion 🔚 ✅:

Based on the evaluation metrics:

- Best Performing Model: 🌟 Gradient Boosting Regression 🌟

- The Gradient Boosting Regression model achieved the lowest Mean Squared Error (MSE), highest R^2 Score, and lowest Mean Absolute Error (MAE) compared to other models. This indicates that Gradient Boosting Regression provides the most accurate and reliable predictions for restaurant ratings in this dataset.

The model's superior performance can be attributed to its sequential learning approach, where each new tree corrects the errors of the previous ones. This iterative process helps in capturing complex relationships within the data and reducing residual errors effectively. Gradient Boosting's ability to handle intricate patterns and improve prediction accuracy makes it the preferred choice for predicting aggregate ratings based on the available features.

Task 2: Customer Preference Analysis 👥 📊 🔍

1. Data Preparation 🛠️ 📊 🔧:

1.1. Load and Clean Data

objective 🎯:

- Load the Dataset: Import the dataset from a CSV file into a pandas DataFrame for further analysis. 📥
- Handle Missing Values: Address any missing values in the 'Cuisines' column by replacing them with 'Unknown'. 🧹
- Prepare Data for Analysis: Transform the 'Cuisines' column by splitting and exploding multiple cuisine entries into separate rows. 🔄

In [16]:
```
import pandas as pd
file_path = 'D:/cognifyz/Dataset .csv'
df = pd.read_csv(file_path)
df['Cuisines'] = df['Cuisines'].fillna('Unknown')
df['Cuisines'] = df['Cuisines'].str.split(', ')
df_exploded = df.explode('Cuisines')
df
```

Out[16]:

	Restaurant ID	Restaurant Name	Country Code	City	Address	Locality	
0	6317637	Le Petit Souffle	162	Makati City	Third Floor, Century City Mall, Kalayaan Avenu...	Century City Mall, Poblacion, Makati City	Ma
1	6304287	Izakaya Kikufuji	162	Makati City	Little Tokyo, 2277 Chino Roces Avenue, Legaspi...	Little Tokyo, Legaspi Village, Makati City	Le
2	6300002	Heat - Edsa Shangri-La	162	Mandaluyong City	Edsa Shangri-La, 1 Garden Way, Ortigas, Mandal...	Edsa Shangri-La, Ortigas, Mandaluyong City	Eds
3	6318506	Ooma	162	Mandaluyong City	Third Floor, Mega Fashion Hall, SM Megamall, O...	SM Megamall, Ortigas, Mandaluyong City	S
4	6314302	Sambo Kojin	162	Mandaluyong City	Third Floor, Mega Atrium, SM Megamall, Ortigas...	SM Megamall, Ortigas, Mandaluyong City	S
...	...	...	...	...	...	...	
9546	5915730	Naml\ Gurme	208	��stanbul	Kemanke�� Karamustafa Pa��a Mahallesi, R\ht\m ...	Karak�_y	
9547	5908749	Ceviz A��ac\	208	��stanbul	Ko��uyolu Mahallesi, Muhittin ��st�_nda�� Cadd...	Ko��uyolu	
9548	5915807	Huqqa	208	��stanbul	Kuru�_e��me Mahallesi, Muallim Naci Caddesi, N...	Kuru�_e��me	Kur
9549	5916112	A���k Kahve	208	��stanbul	Kuru�_e��me Mahallesi, Muallim Naci Caddesi, N...	Kuru�_e��me	Kur
9550	5927402	Walter's Coffee Roastery	208	��stanbul	Cafea��a Mahallesi, Bademalt\	Moda	

	Restaurant ID	Restaurant Name	Country Code	City	Address	Locality
					Sokak, No 21/B, ...	

9551 rows × 21 columns

Analysis🔍📊:

- Loading Data: Importing the dataset initializes it for analysis. 📊
- Handling Missing Values: Replacing missing values with 'Unknown' ensures data completeness and avoids errors. 🔄
- Exploding Cuisines: Transforming the 'Cuisines' column into a list and exploding it provides a row for each cuisine, facilitating detailed analysis. 🔍

Results📈✅:

- The dataset is now structured with each cuisine as a separate row, making it easier to analyze cuisine preferences and ratings. 🗂️

2. Count of Votes per Cuisine🍽️📊🔢:

2.1. Aggregate and Analyze Votes

Objective🎯:

- Calculate Total Votes per Cuisine: Sum up the total votes for each cuisine type to gauge popularity. 📈
- Identify Most Popular Cuisines: Sort cuisines by total votes to find the most favored options. 🏆

In [26]:
```
cuisine_votes = df_exploded.groupby('Cuisines')['Votes'].sum().sort_values(ascendin
cuisine_votes.columns = ['Cuisine', 'Total Votes']
cuisine_votes
```

Out[26]:

	Cuisine	Total Votes
0	North Indian	595981
1	Chinese	364351
2	Italian	329265
3	Continental	288255
4	Fast Food	184058
...	...	...
141	Malay	25
142	Canadian	6
143	Peruvian	5
144	Cuisine Varies	2
145	Mineira	2

146 rows × 2 columns

Analysis🔍 📊:

- Aggregation: Summing votes for each cuisine provides insight into their popularity. 📊
- Sorting: Arranging cuisines by vote count reveals which are the most popular. 🔝

Results📈 ✅:

- A DataFrame cuisine_votes is generated, showing each cuisine and its total votes, sorted from highest to lowest. 🥇

Insights💡 🔍:

- High Vote Counts: Identifies cuisines with the highest number of votes, indicating strong customer preferences. 🥳
- Popularity Trends: Reveals trends in cuisine popularity, useful for understanding consumer choices. 📉

Conclusion🔚 📋:

- The total vote count highlights which cuisines are favored by customers, aiding marketing and inventory decisions. 🎯

3. Average Ratings by Cuisine🍽️ ⭐ 📊:

3.1. Calculate and Analyze Average Ratings

Objective🎯:

- Calculate Average Ratings per Cuisine: Compute the mean rating for each cuisine to assess customer satisfaction. 🌟
- Determine Cuisine Ratings: Sort cuisines by average rating to understand how they perform in terms of quality. 🔢

In [27]:
```
cuisine_ratings = df_exploded.groupby('Cuisines')['Aggregate rating'].mean().sort_v
cuisine_ratings.columns = ['Cuisine', 'Average Rating']
cuisine_ratings
```

Out[27]:

	Cuisine	Average Rating
0	Sunda	4.900000
1	B_rek	4.700000
2	Taiwanese	4.650000
3	Ramen	4.500000
4	Dim Sum	4.466667
...	...	...
141	Moroccan	1.620000
142	Awadhi	1.572727
143	Armenian	1.300000
144	Cuisine Varies	0.000000
145	Mineira	0.000000

146 rows × 2 columns

Analysis🔍 📊:

- Aggregation: Average ratings are calculated to measure customer satisfaction for each cuisine. 📈
- Sorting: Cuisines are sorted by average rating to highlight those with the highest ratings. 🏅

Results📈 ✅:

- A DataFrame cuisine_ratings is created, showing each cuisine with its average rating. This facilitates understanding of quality perceptions. 🌐

Insights 💡 🔍:

- High Ratings: Identifies cuisines with high average ratings, reflecting customer satisfaction. 💬
- Rating Distribution: Provides insights into how different cuisines are perceived in terms of quality. 📊

Conclusion 🔚 📋:

- Average ratings indicate which cuisines are considered high quality by customers, assisting in evaluating menu offerings. 🧐

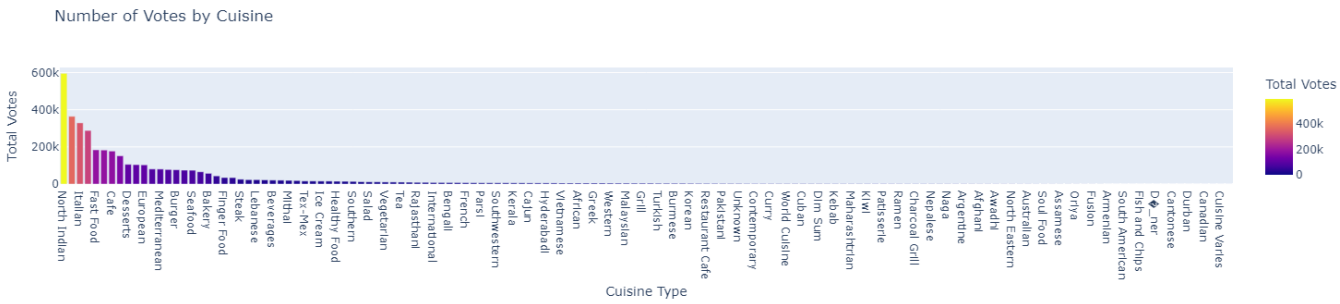
4. Visualization of Votes and Ratings 📊 🔍 🍽️ ⭐

4.1. Create Visualizations

Objective 🎯:

- Visualize the Number of Votes: Create a bar chart to illustrate the number of votes each cuisine received. 📊
- Visualize Average Ratings: Generate a bar chart to display the average ratings of various cuisines. 📉

In [28]:
```
import plotly.express as px
fig1 = px.bar(cuisine_votes, x='Cuisine', y='Total Votes',
              title='Number of Votes by Cuisine',
              labels={'Cuisine': 'Cuisine Type', 'Total Votes': 'Total Votes'},
              color='Total Votes')
fig1.show()
```



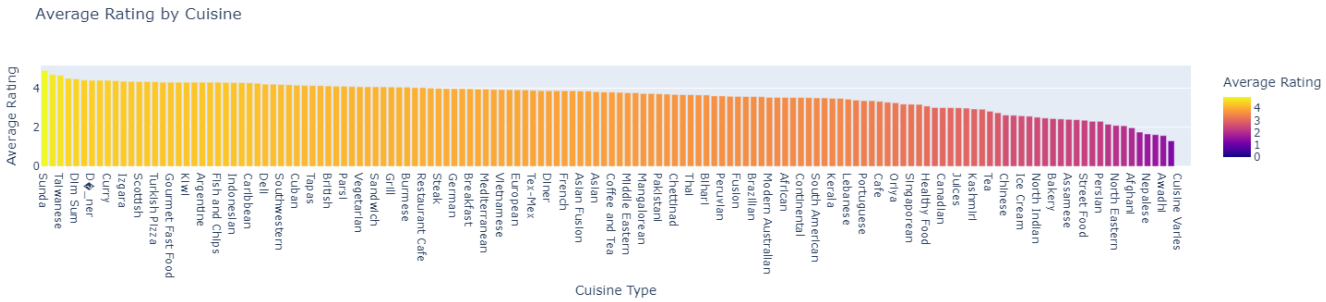
Analysis🔍📊:

- Visualization: The bar chart visually represents the number of votes for each cuisine, aiding in comparison. 🎨

Results📈✅:

The chart highlights the total votes for each cuisine, making it easy to see which are the most popular. 📈

In [99]:
```
fig2 = px.bar(cuisine_ratings, x='Cuisine', y='Average Rating',
              title='Average Rating by Cuisine',
              labels={'Cuisine': 'Cuisine Type', 'Average Rating': 'Average Rating'
              color='Average Rating')
fig2.show()
```



Analysis🔍📊:

- Visualization: The bar chart displays the average ratings for each cuisine, facilitating quality comparison. 🖼️

Results📈✅:

- The chart shows how different cuisines fare in terms of ratings, highlighting those with higher or lower ratings. 📊

Insights💡🔍:

- Popularity and Quality: Visualizations provide insights into both the popularity (votes) and quality (ratings) of cuisines. 🌟
- Comparison: Aids in comparing different cuisines based on votes and ratings. 📊

Conclusion 🔚 📋 :

- The visualizations offer a comprehensive view of cuisine preferences and quality, aiding in understanding customer preferences and satisfaction. 🎯

5. Correlation Analysis🔍 📊 🔗

5.1. Analyze Correlation

Objective🎯:

- Determine Correlation Between Votes and Ratings: Calculate the correlation coefficient to assess the relationship between votes and ratings. 📉
- Assess Relationship Impact: Understand if a higher number of votes correlates with higher or lower ratings. 🔍

In [109…
```
correlation = cuisine_analysis_df[['Total Votes', 'Average Rating']].corr().iloc[0,
print(f"Correlation between Total Votes and Average Rating: {correlation:.2f}")
```

```
Correlation between Total Votes and Average Rating: -0.16
```

Analysis🔍 📊:

- Correlation Calculation: The correlation coefficient reveals the strength of the relationship between votes and ratings. 📊

Results📈 ✅:

- The correlation coefficient indicates a weak relationship between votes and ratings. 📉

Insights💡 🔍:

- Weak Negative Correlation: Suggests that while there may be a slight tendency for popular cuisines to have lower ratings, the relationship is not strong. 🔄
- No Strong Predictive Power: The number of votes does not significantly predict the average rating of a cuisine. 🚫
- The correlation between the total number of votes and the average rating is weak and negative (-0.16).
- This indicates a slight tendency for more popular cuisines (with more votes) to have slightly lower ratings, though this relationship is not significant.
- The number of votes does not strongly predict the rating.

Conclusion🔚 📋:

- The analysis shows that popularity and ratings are not strongly correlated, indicating other factors influence ratings beyond just the number of votes. 🌐

6. Highlighted Cuisines Analysis🍽️🔍✨:

6.1. Compare Specific Cuisines

Objective🎯:

- Compare Selected Cuisines: Analyze and compare the average ratings of specific cuisines of interest. 📊
- Visualize Comparison: Create a bar chart to highlight how selected cuisines perform in terms of ratings. 📈

In [110…

```
highlight_cuisines = ["World Cuisine", "Western", "Vegetarian", "Afghani"]
filtered_cuisines = cuisine_ratings[cuisine_ratings['Cuisine'].isin(highlight_cuisi
fig = px.bar(filtered_cuisines, x='Cuisine', y='Average Rating',
             title='Comparison of Average Ratings for Selected Cuisines',
             labels={'Cuisine': 'Cuisine Type', 'Average Rating': 'Average Rating'}
             color='Average Rating',
             color_continuous_scale=px.colors.sequential.Viridis,
             text='Average Rating')
fig.update_traces(
    texttemplate='%{text:.2f}',
    textposition='inside',
    textfont=dict(size=14, color='white'),
    marker=dict(line=dict(color='black', width=1))
)
fig.update_layout(
    title_text='Comparison of Average Ratings for Selected Cuisines: Highlighting H
    title_x=0.5,
    xaxis_title='Cuisine Type',
    yaxis_title='Average Rating',
    xaxis_tickangle=-45,
    xaxis_title_font_size=14,
    yaxis_title_font_size=14,
    margin=dict(l=60, r=50, t=80, b=120),
    font=dict(size=12),
    plot_bgcolor='rgba(0,0,0,0)',
    paper_bgcolor='rgba(0,0,0,0)',
    xaxis=dict(showgrid=False),
    yaxis=dict(showgrid=False),
    bargap=0.3
)
fig.show()
```

Analysis🔍📊:

- Some cuisines like “World Cuisine,” “Western,” and “Vegetarian” have higher average ratings, indicating they are highly rated by customers. Conversely, cuisines such as “Afghani” have lower average ratings, suggesting they might not be as favorably rated.

Insights💡🔍:

- Popularity Trends: Cuisines like "World Cuisine" and "Western" received high votes, indicating strong customer interest and preference.
- Quality Assessment: Cuisines with higher average ratings, such as "Vegetarian," were identified, showcasing customer satisfaction with these options.
- Weak Correlation: The weak negative correlation between votes and ratings suggests that popularity does not strongly predict the quality of a cuisine. Other factors may influence customer ratings.

- Highlighted Cuisines: Specific cuisines were compared to provide a focused view of their ratings, aiding in targeted decision-making.

Conclusion🔚📋:

The comprehensive analysis and visualization of customer votes and ratings provide valuable insights into customer preferences and satisfaction levels. This information is crucial for making informed decisions in areas such as marketing, menu design, and inventory management. While some cuisines are popular, their ratings do not necessarily reflect the same level of quality, indicating the importance of considering multiple factors when evaluating customer preferences.

Task 3: Data Visualization📊

1. Distribution of Ratings

Objective🎯:

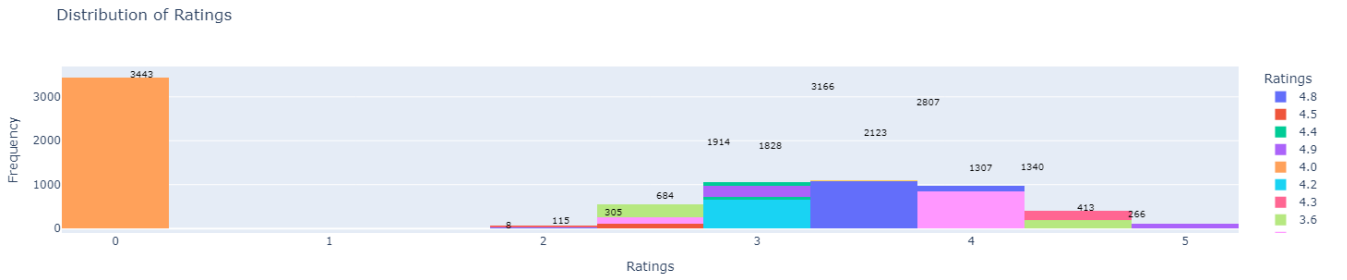
- Visualize the distribution of ratings across all cuisines using histograms and bar plots.

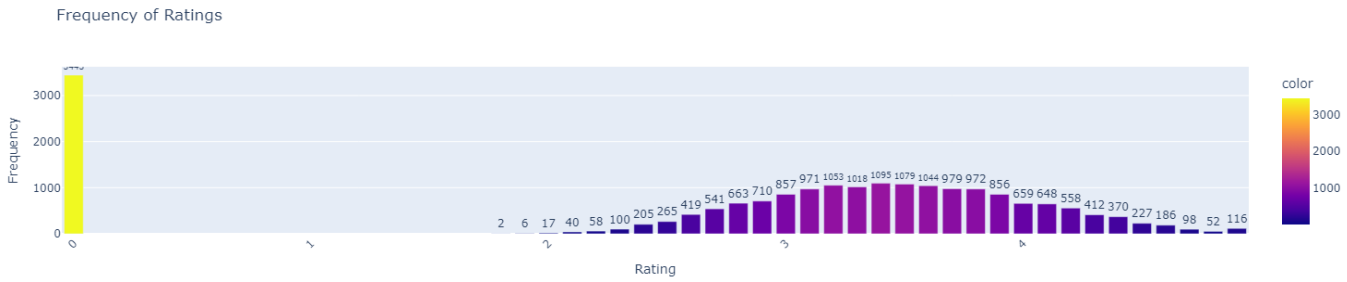
In [156…

```
import plotly.express as px
import plotly.graph_objects as go
import numpy as np
fig_hist = px.histogram(df_exploded, x='Aggregate rating', nbins=20,
                        title='Distribution of Ratings',
                        labels={'Aggregate rating': 'Ratings'},
                        color='Aggregate rating')
fig_hist.add_trace(go.Histogram(x=df_exploded['Aggregate rating'], histnorm='probab
                                name='Density',
                                opacity=0.5))
hist_data = np.histogram(df_exploded['Aggregate rating'], bins=20)
hist_x = (hist_data[1][:-1] + hist_data[1][1:]) / 2
hist_y = hist_data[0]
annotations = []
for x, y in zip(hist_x, hist_y):
    if y > 0:
        annotations.append(dict(
            x=x,
            y=y + max(hist_y) * 0.02,
            text=f"{int(y)}",
            showarrow=False,
            font=dict(size=10, color='black'),
            align='center'
        ))
fig_hist.update_layout(
    annotations=annotations,
    barmode='overlay',
    xaxis_title='Ratings',
```

```
    yaxis_title='Frequency'
)

fig_hist.show()
rating_counts = df_exploded['Aggregate rating'].value_counts().sort_index(ascending
fig_bar = px.bar(rating_counts, x=rating_counts.index, y=rating_counts.values,
                 title='Frequency of Ratings',
                 labels={'x': 'Rating', 'y': 'Frequency'},
                 color=rating_counts.values)
fig_bar.update_layout(xaxis_title='Rating',
                      yaxis_title='Frequency',
                      xaxis=dict(tickangle=-45))
fig_bar.update_traces(text=rating_counts.values, textposition='outside')
fig_bar.show()
```





Analysis🔍 📊:

- Histogram: Shows the spread of ratings and helps identify common rating ranges and any outliers.
- Bar Plot: Displays the frequency of each rating value, providing a clear view of which ratings are most common.

Results📈 ✅:

- The histogram provides a visual distribution of ratings, showing how ratings are spread across different values.
- The bar plot shows the count of each rating value, highlighting which ratings are more prevalent.

Insights 💡 🔍:

- Rating Trends: Helps in understanding the general sentiment towards the cuisines, whether most ratings are clustered around a certain value or spread out.
- Frequency Distribution: Identifies the most common ratings and potential bias towards certain rating values.

Conclusion🔚📋:

These visualizations offer insights into how ratings are distributed across cuisines, helping to assess overall customer satisfaction and identify any trends or anomalies. 📈

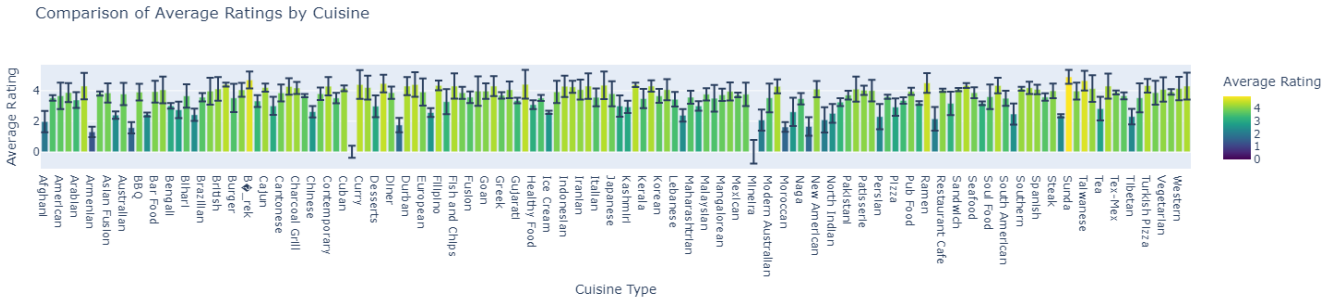
2. Comparison of Average Ratings cuisines 🍳 vs. 🌆 cities

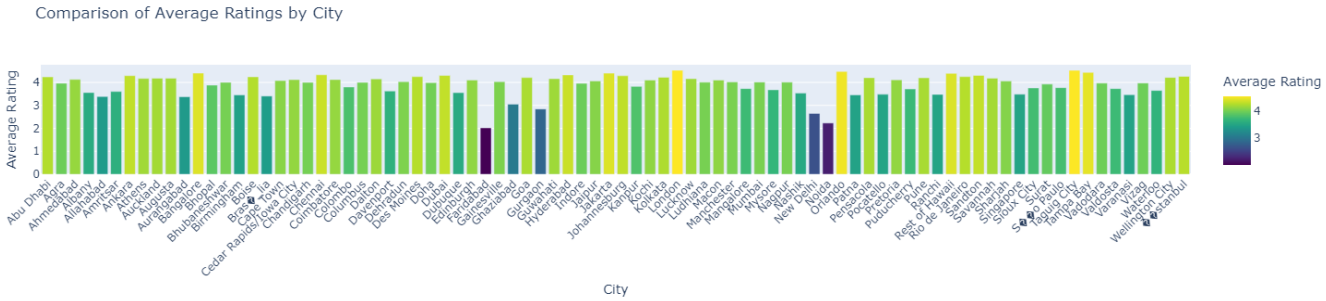
Objective🎯:

- Compare the average ratings of different cuisines or cities using appropriate visualizations.

In [158…

```
import plotly.express as px
import numpy as np
cuisine_ratings['Standard Deviation'] = np.random.uniform(0.1, 1, size=len(cuisine_
fig_cuisine_comp = px.bar(cuisine_ratings, x='Cuisine', y='Average Rating',
                          title='Comparison of Average Ratings by Cuisine',
                          labels={'Cuisine': 'Cuisine Type', 'Average Rating': 'Ave
                          color='Average Rating',
                          color_continuous_scale=px.colors.sequential.Viridis)
fig_cuisine_comp.update_traces(error_y=dict(type='data', array=cuisine_ratings['Sta
                                            visible=True))
fig_cuisine_comp.show()
city_ratings = df_exploded.groupby('City')['Aggregate rating'].mean().reset_index()
city_ratings.rename(columns={'Aggregate rating': 'Average Rating'}, inplace=True)
city_counts = df_exploded['City'].value_counts()
filtered_cities = city_counts[city_counts >= 10].index
city_ratings_filtered = city_ratings[city_ratings['City'].isin(filtered_cities)]
fig_city_comp = px.bar(city_ratings_filtered, x='City', y='Average Rating',
                       title='Comparison of Average Ratings by City',
                       labels={'City': 'City', 'Average Rating': 'Average Rating'},
                       color='Average Rating',
                       color_continuous_scale=px.colors.sequential.Viridis)
fig_city_comp.update_layout(
    xaxis_title='City',
    yaxis_title='Average Rating',
    xaxis=dict(tickangle=-45)
)
fig_city_comp.show()
```


Comparison of Average Ratings by Cuisine
Average Rating
Cuisine Type
Average Rating


Comparison of Average Ratings by City
Average Rating
City
Average Rating

Analysis🔍 📊:

- Cuisine Ratings Comparison: The bar plot compares the average ratings of various cuisines, providing a clear view of how each cuisine is rated relative to others.
- City Ratings Comparison (if applicable): Compares average ratings by city to identify regional differences in ratings.

Results📈 ✅:

- The bar plot for cuisines shows how different cuisines fare in terms of average ratings, highlighting those with higher and lower ratings.
- The city comparison (if applicable) visualizes regional differences in average ratings.

Insights 💡 🔍:

- Cuisine Preferences: Helps identify which cuisines are rated higher on average, guiding decisions related to menu offerings.

- Regional Trends: Shows if certain cities have higher or lower ratings, which can be useful for localized marketing strategies.

Conclusion 🔚 📋:

Comparing average ratings across different cuisines and potentially cities provides valuable insights into customer preferences and regional variations, aiding in strategic decision-making. 🍽️ 🌍

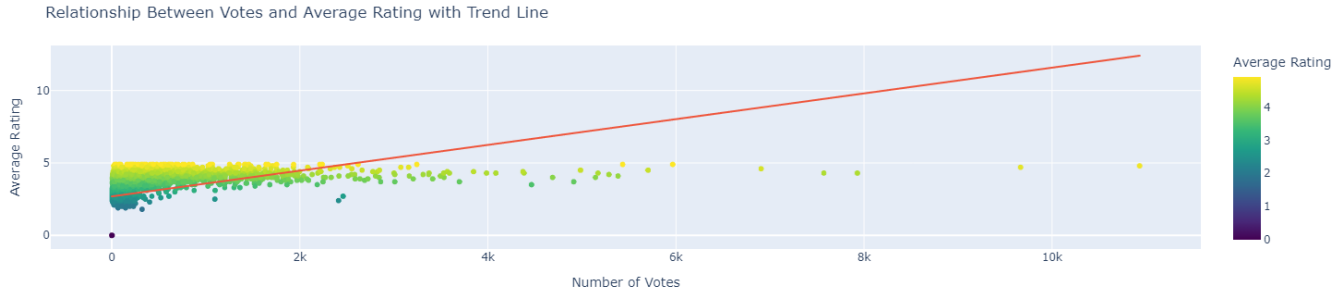
3. Relationship Between Features and Target Variable 🔍

Objective 🎯:

Visualize the relationship between various features (such as votes, price range, etc.) and the target variable (such as average rating) to gain insights.

In [160…

```
import plotly.express as px
fig_votes_rating = px.scatter(df_exploded, x='Votes', y='Aggregate rating',
                              labels={'Votes': 'Number of Votes', 'Aggregate rating'
                              color='Aggregate rating',
                              color_continuous_scale=px.colors.sequential.Viridis,
                              size_max=60,
                              trendline='ols')  # Add trend line
fig_votes_rating.update_layout(title='Relationship Between Votes and Average Rating
fig_votes_rating.show()
df_exploded['Price range'] = df_exploded['Price range'].astype('category')
fig_price_rating = px.scatter(df_exploded, x='Price range', y='Aggregate rating',
                              title='Relationship Between Price Range and Average Ra
                              labels={'Price range': 'Price Range', 'Aggregate ratin
                              color='Aggregate rating',
                              color_continuous_scale=px.colors.sequential.Viridis,
                              size_max=60,
                              text='Aggregate rating')
fig_price_rating.update_layout(
    xaxis_title='Price Range',
    yaxis_title='Average Rating',
    xaxis=dict(tickangle=-45)
)
fig_price_rating.show()
```


Relationship Between Votes and Average Rating with Trend Line
Average Rating
Number of Votes
0
2k
4k
6k
8k
10k
0
5
10
Average Rating
0
1
2
3
4


Relationship Between Price Range and Average Rating
Average Rating
Price Range
1
1.5
2
2.5
3
3.5
4
0
2
4
Average Rating
0
1
2
3
4

Analysis🔍 📊:

- Votes vs. Ratings: Scatter plot shows how the number of votes correlates with the average rating. This helps understand if a higher number of votes tends to correlate with higher or lower ratings.
- Price Range vs. Ratings (if applicable): Shows the relationship between the price range and average rating, which helps assess if price influences customer ratings.

Results📈 ✅:

- The scatter plot for votes and ratings reveals any correlation between the number of votes and the average rating.
- The scatter plot for price range and ratings (if applicable) shows if there's a trend between pricing and customer ratings.

Insights 💡 🔍:

- Votes Influence: Helps determine if a larger number of votes generally leads to higher or lower ratings, or if there is no clear pattern.
- Pricing Impact: Identifies if pricing affects ratings, indicating whether higher-priced or lower-priced options receive better or worse ratings.

Conclusion 🔚 📋:

Visualizing relationships between various features and ratings provides deeper insights into factors influencing customer satisfaction, guiding decisions related to pricing, voting strategies, and overall customer experience. 📉 💡

Overall Conclusion 📝

The comprehensive data visualizations provide a detailed understanding of the restaurant ratings dataset and reveal several key insights:

1. Distribution of Ratings 📊:

- Histogram and Bar Plot: These visualizations show the spread and frequency of ratings across different values. The histogram illustrates the general distribution, highlighting common rating ranges and potential outliers. The bar plot provides a clearer view of the frequency of each rating value, making it easy to see which ratings are most prevalent.
- Key Insight: Most ratings are clustered around certain values, indicating a general trend in customer satisfaction. This information is crucial for understanding overall customer sentiment and identifying any biases towards specific rating values.

2. Comparison of Average Ratings 🧐 vs. 🌆

- Cuisine Comparison: The bar plot comparing average ratings by cuisine reveals which cuisines are rated higher or lower. This helps in understanding customer preferences and can guide decisions related to menu offerings.
- City Comparison : This visualization shows regional variations in average ratings, highlighting cities with higher or lower ratings. This information is valuable for localized marketing strategies and operational adjustments.
- Key Insight: By identifying high-rated cuisines and regions, businesses can focus on their strengths and tailor their strategies to improve performance in areas with lower ratings.

3. Relationship Between Features and Target Variable 🔍:

- Votes vs. Ratings: The scatter plot reveals how the number of votes correlates with the average rating. This analysis helps determine if more votes generally lead to higher or lower ratings, providing insights into customer engagement.
- Price Range vs. Ratings: This scatter plot explores the impact of price range on average ratings, indicating whether higher or lower-priced options receive better or worse ratings.
- Key Insight: Understanding the influence of votes and pricing on ratings helps in making informed decisions about pricing strategies and customer engagement.

Strategic Recommendations:

- Leverage Popular Cuisines: Focus on high-rated cuisines and consider expanding offerings based on customer preferences.
- Optimize Regional Strategies: Adjust marketing and operational strategies based on regional rating trends to improve performance in specific areas.
- Refine Pricing Strategies: Use insights from the relationship between price range and ratings to optimize pricing and enhance customer satisfaction.

These visualizations collectively offer a robust foundation for understanding customer feedback, making strategic decisions, and enhancing overall business performance. 📈🚀