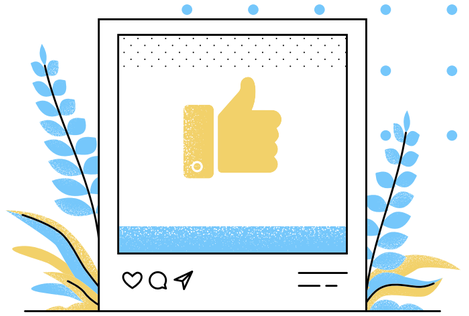
Zomato Restaurant Success Prediction

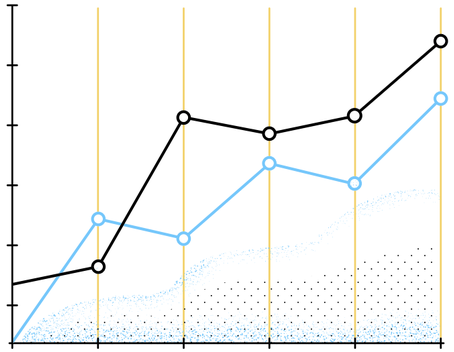
INFO6105 - 34274
DATA SCIENCE ENGINEERING METHODS
SPRING 2021
NORTHEASTERN UNIVERSITY

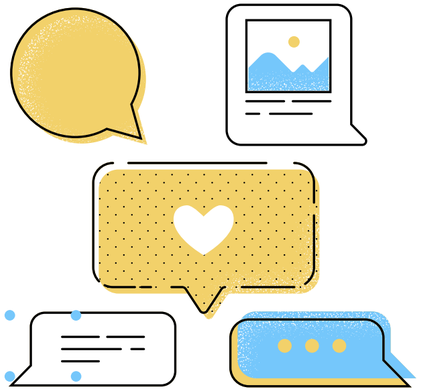
Gowtham Raghuraman | 001529954
Zarana Bhadricha | 001518953

CONTENTS

- Motivation and Goal
- About the Dataset
- Graphical Data Exploration and Analysis
- Data Cleaning and Preparation
- Modeling
- Comparison of Models
- Results and Conclusion







Motivation

Every business wants to know whether it can succeed in the future. For restaurants, rating on Zomato is one of the most important indicators. It not only reveals restaurants' quality and services, but also helps to attract more customers. Also, what factors should be kept in mind if someone wants to open new restaurant. Does the demography of an area matters? Does location of a particular type of restaurant also depends on the people living in that area? Does the theme of the restaurant matters?

Goal

We aim to visualize various factors that contribute to the popularity of a restaurant while studying the trend across the city of Bengaluru location wise. Our goal is to get helpful results like some mentioned below-

- To open a new restaurant what are the cusines that the customers like.
- The location of the restaurant and the most liked dish from various reviews.
- Features to include like online Booking for delivery and table booking.

About the Dataset

We selected this dataset from Kaggle and it includes the restaurant data from Bangalore.
Food culture of Bengalore in India is vibrant and fascinating. Various cuisines right from United States to Japan, Russia to Antarctica, is available here. Delivery, Dine-out, Pubs, Bars, Drinks, Buffet, Desserts Bengaluru has it all. There has been a constant rise in the number of restaurants for two decades, currently which stands at approximately 8000 restaurants.

Dataset at a Glance

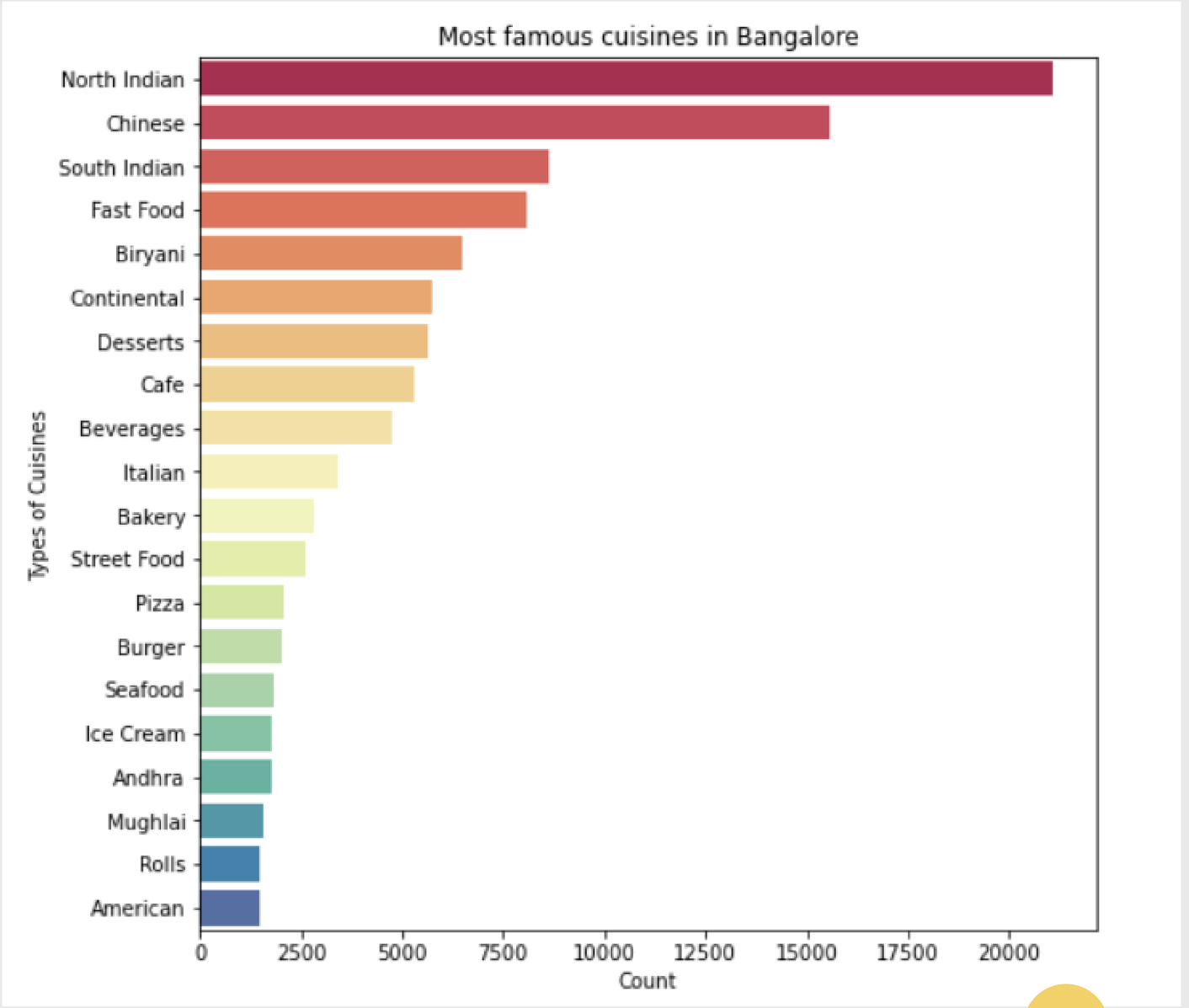
Lets have a look at the data that we are working with

1. The shape of the dataset is 51717 x 17
2. The size of data is 547.48 MB
3. The data has 51717 rows and 17 columns

Graphical Data Exploration and Analysis

What are the top cuisines in Bangalore?

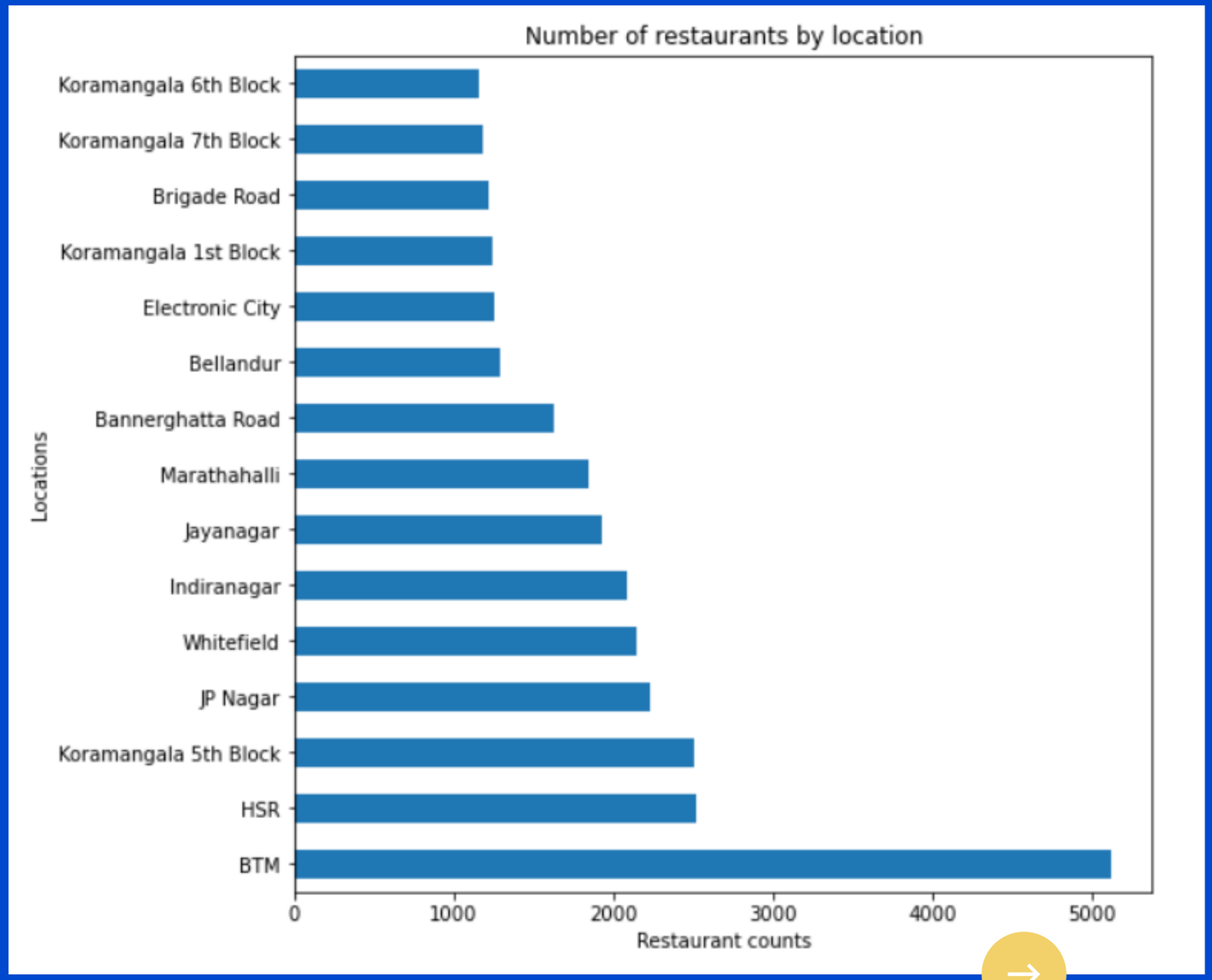
We can observe that North Indian, Chinese, South Indian, Fast Food and Biriyani are most common.



Does number of restaurants depends on area?

We can see from above barplot that BTM, HSR and Koramangala are top locations with maximum number of restaurants. We can infer below points.

1. The competition amomg these locations can be tough to break
2. Most of the foodie are in this area or prefer to go to these locations



How many unique restaurants does Bangalore have?

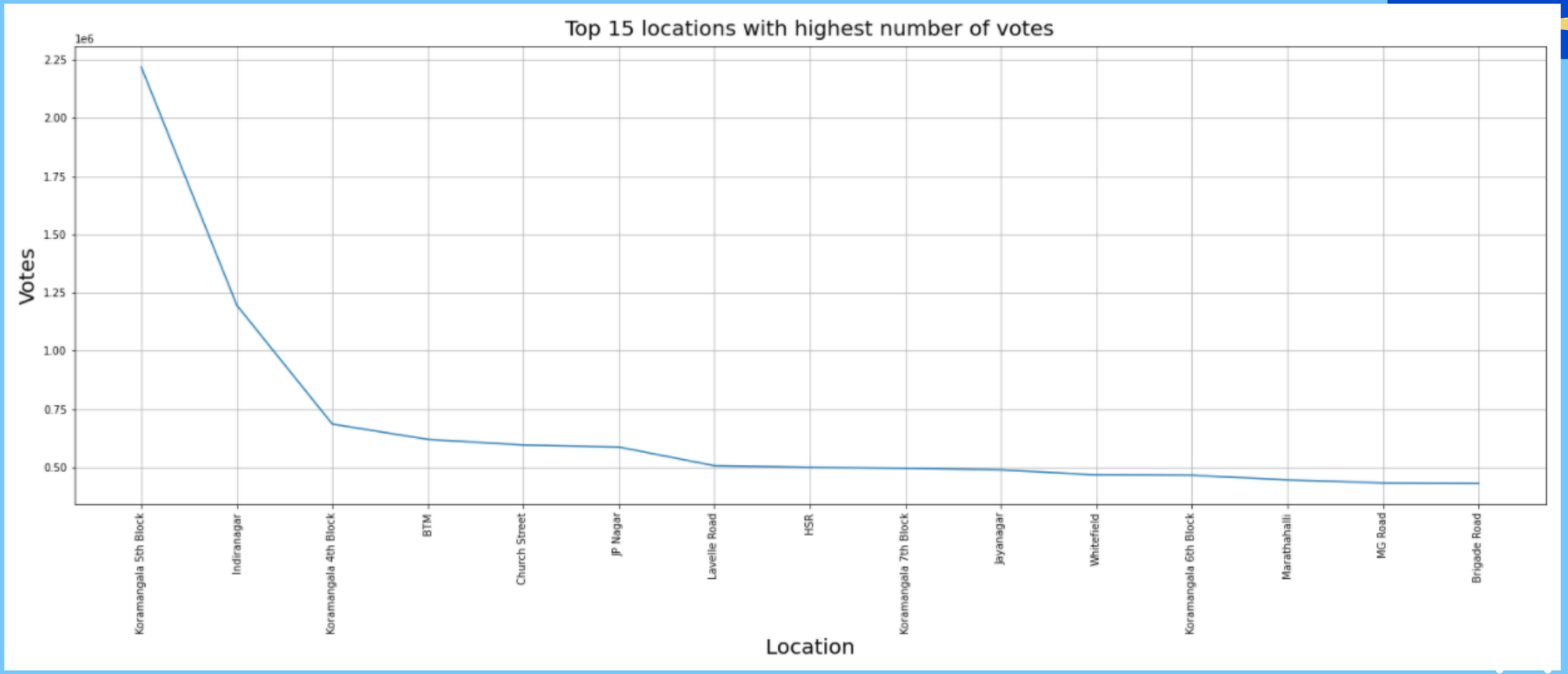
```
print('There are total {} unique Restaurants in Bangalore'.format(len(df['name'].unique())))
```

```
There are total 8792 unique Restaurants in Bangalore
```

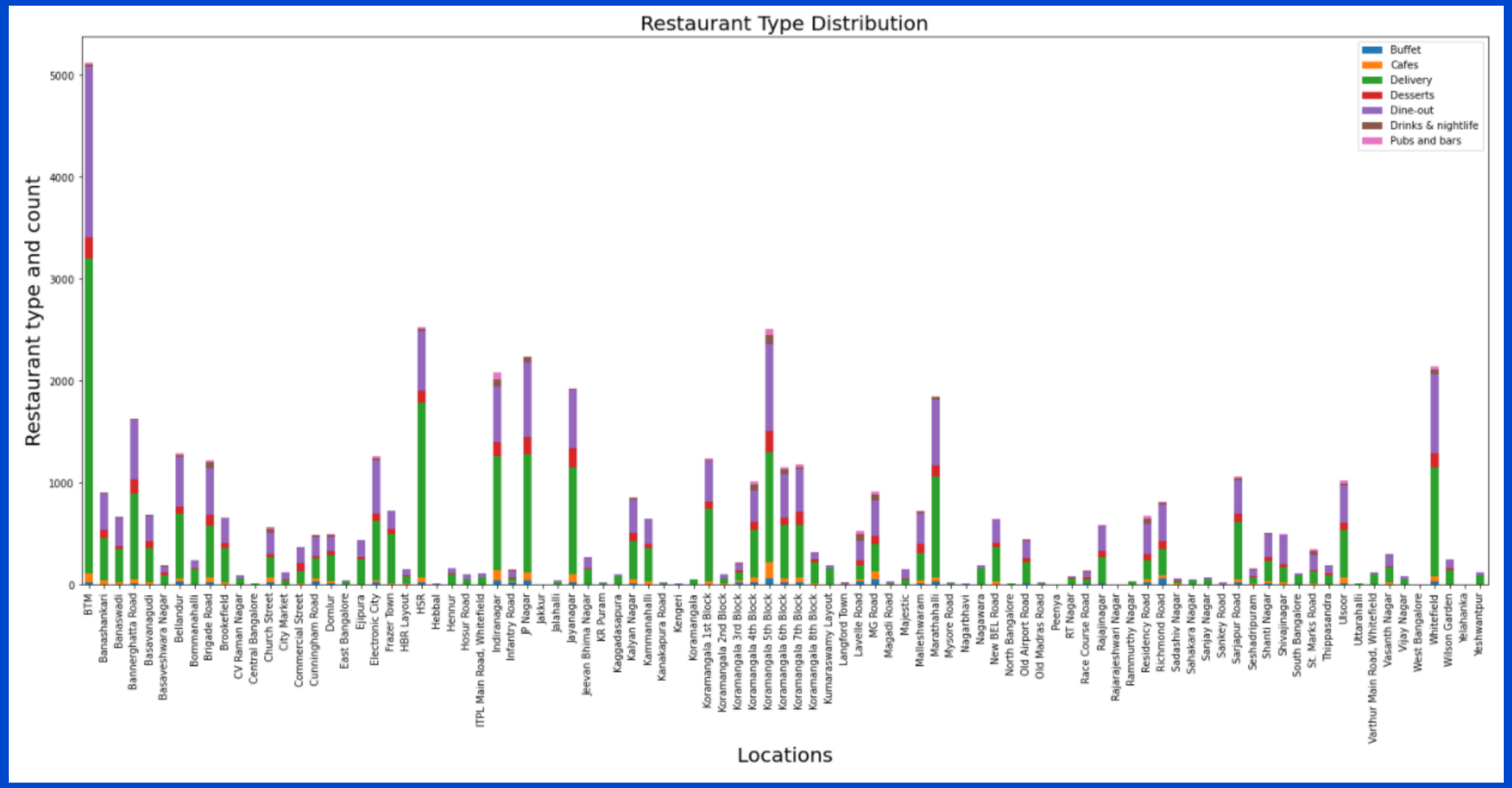
If someone wants to open a new business it is crucial that they understand the existing market first. This gives us an idea of how many unique restaurants are present in Bangalore

Where are the good restaurants located?

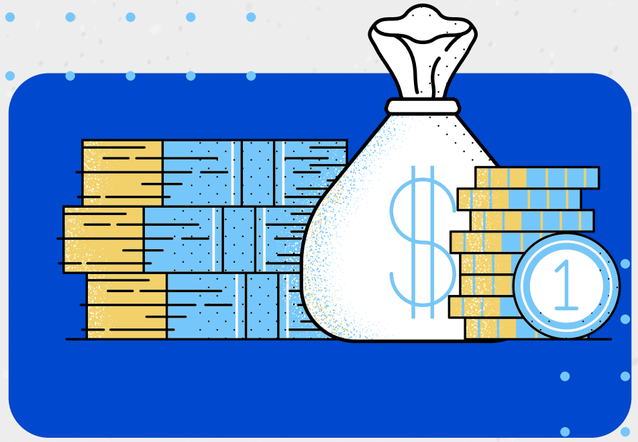
From the graph, we can clearly see Koramangala dominates this scenario.
Koramangala is the heart of Bangalore and attracts the foodies

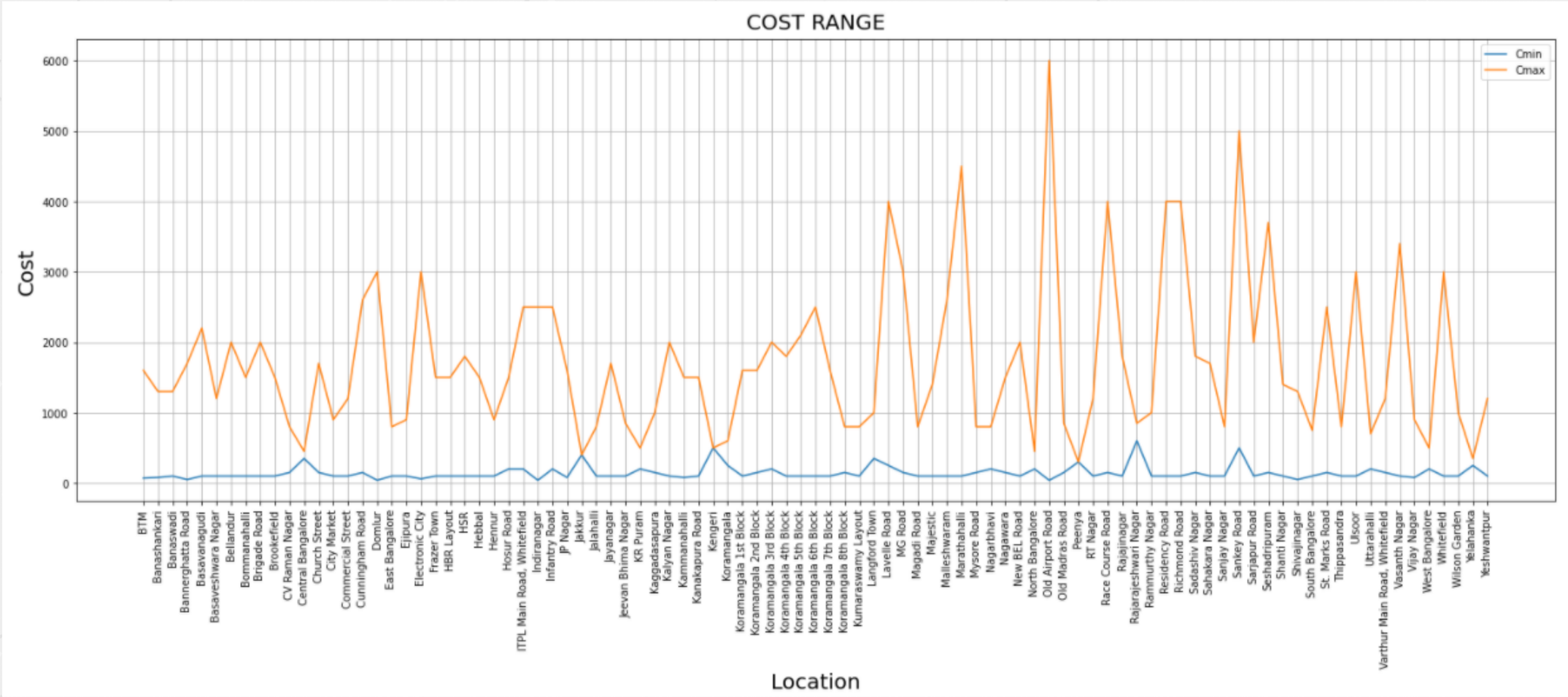


What is the distribution of types of restaurants across different locations?

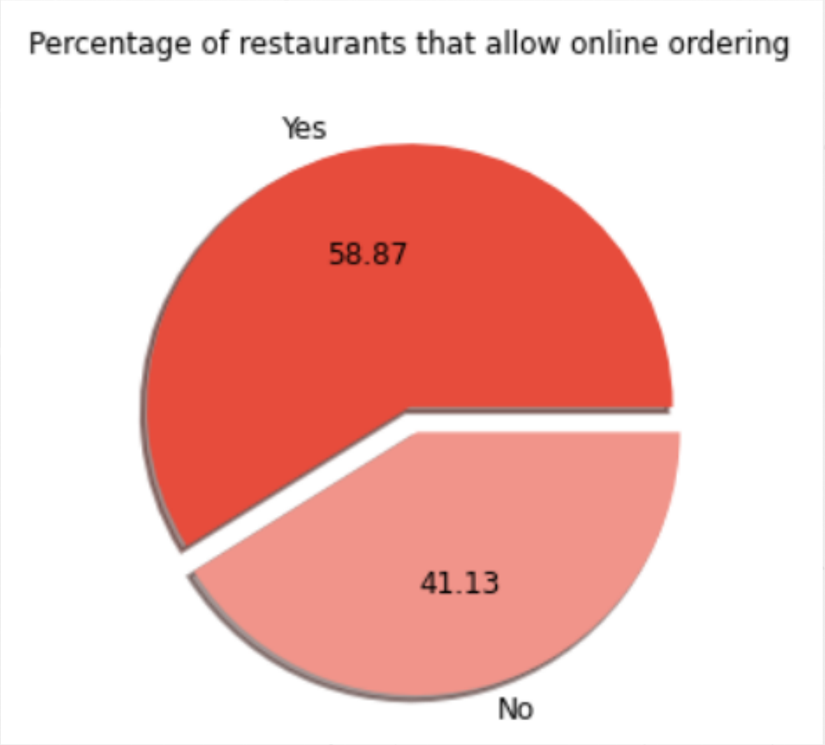


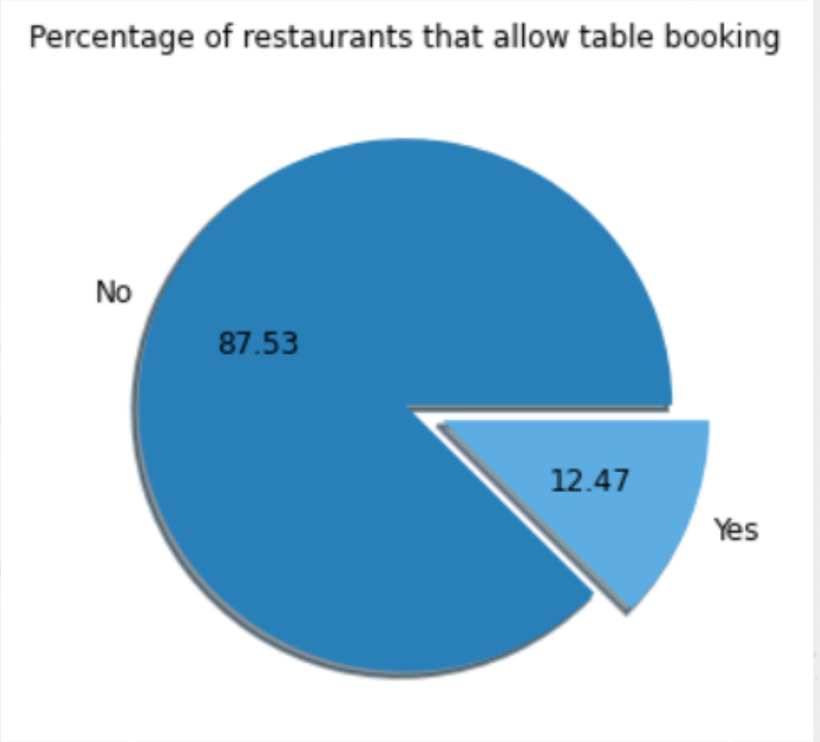
What is the approximate spending in a particular area for two people?



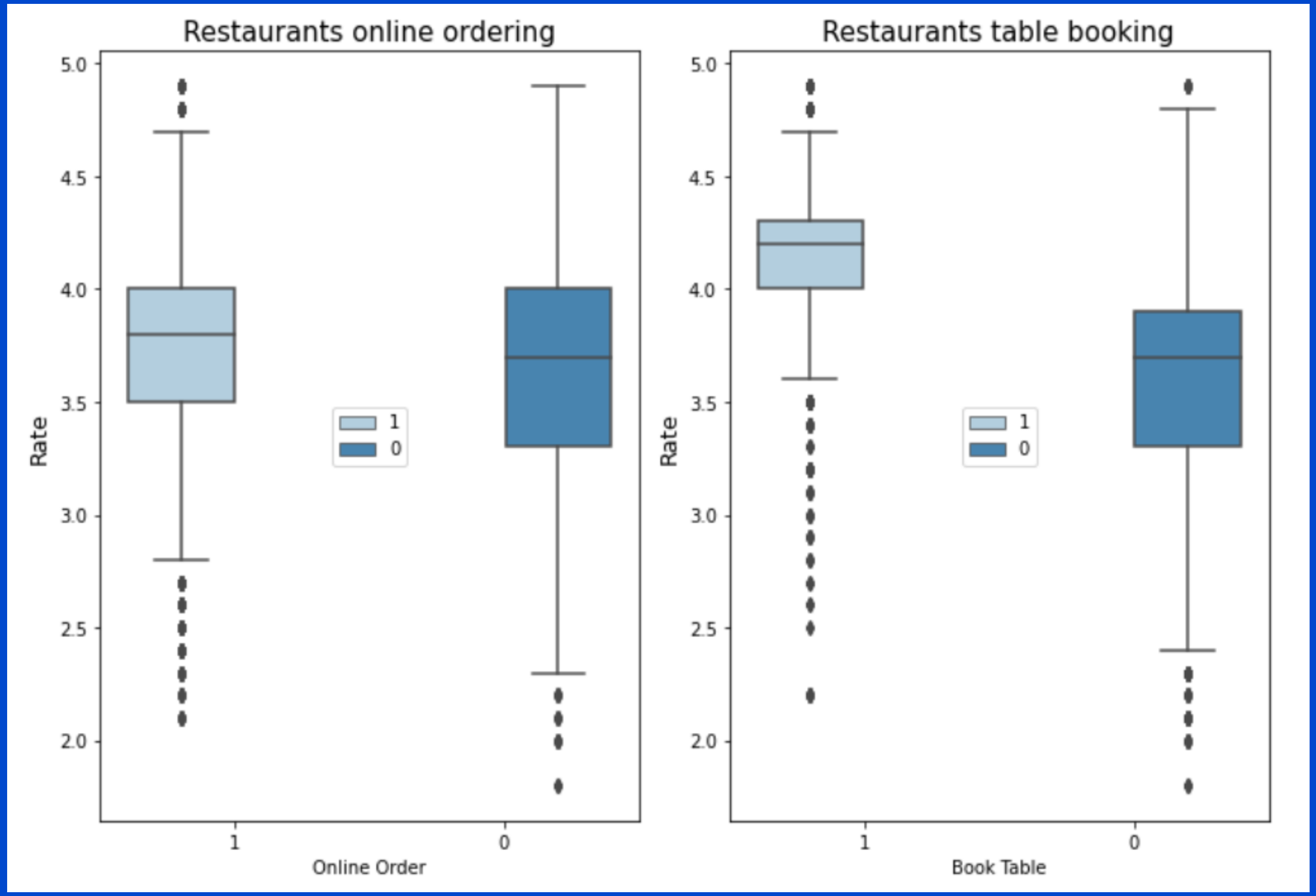


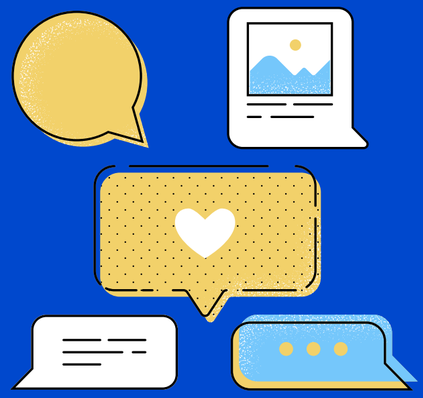
How do table booking and order online facilities impact the rating?



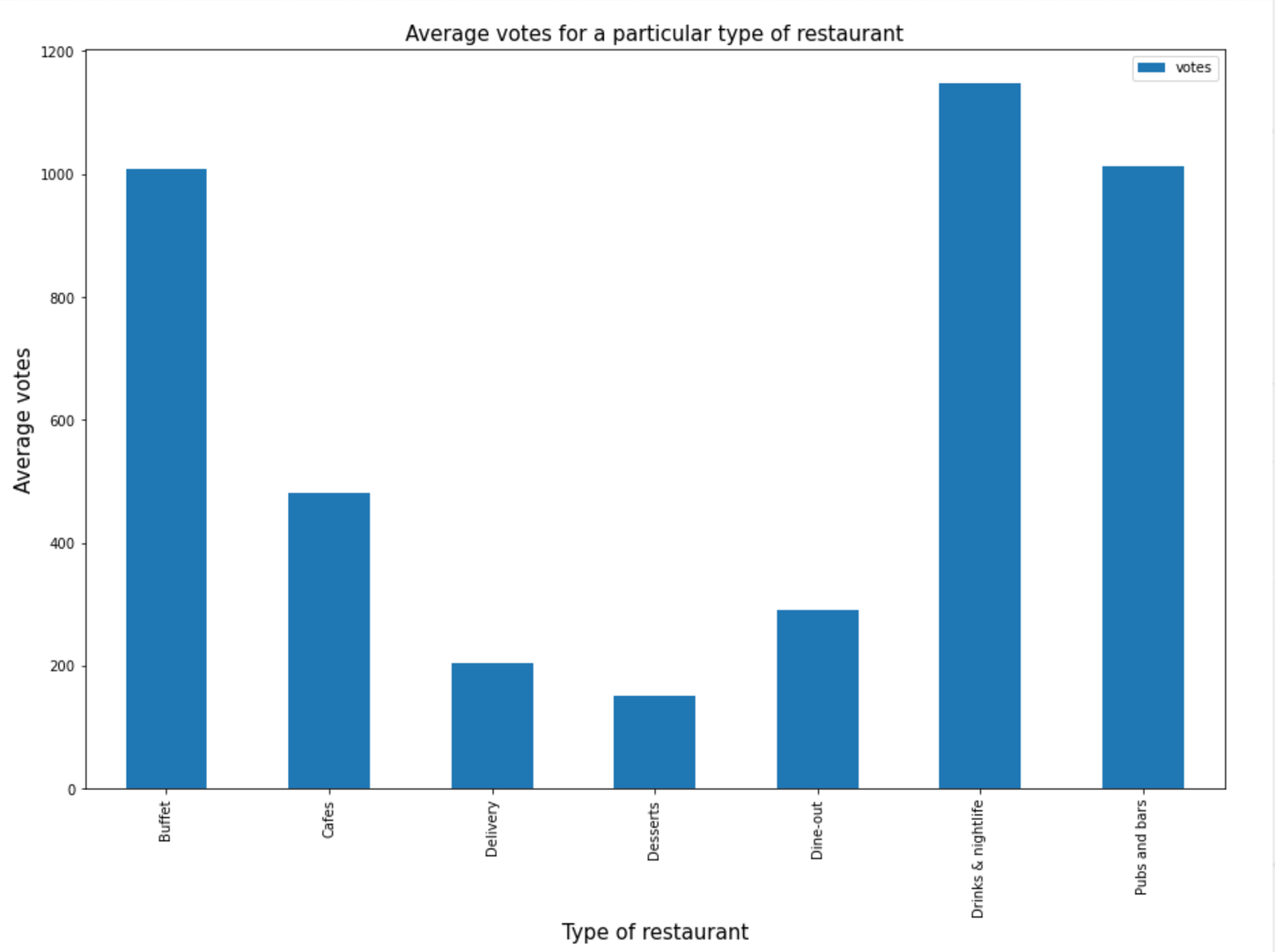


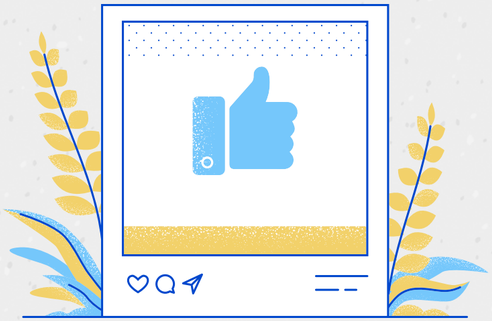
Now lets check how to rating affects the restaurant's overall rating based on these two services



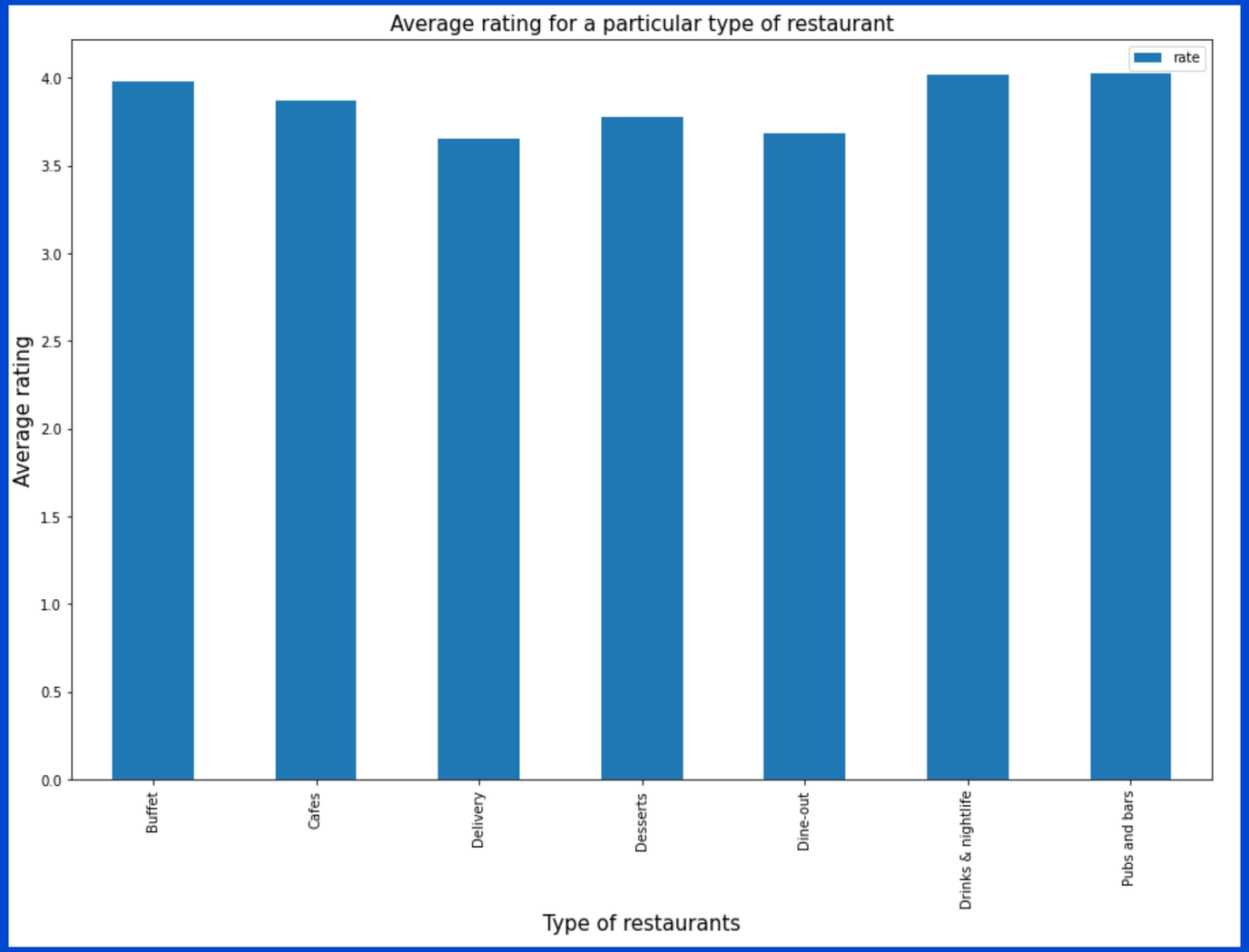


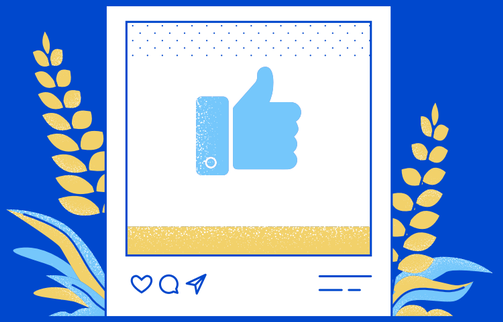
Does particular type of restaurants get more votes than other type?



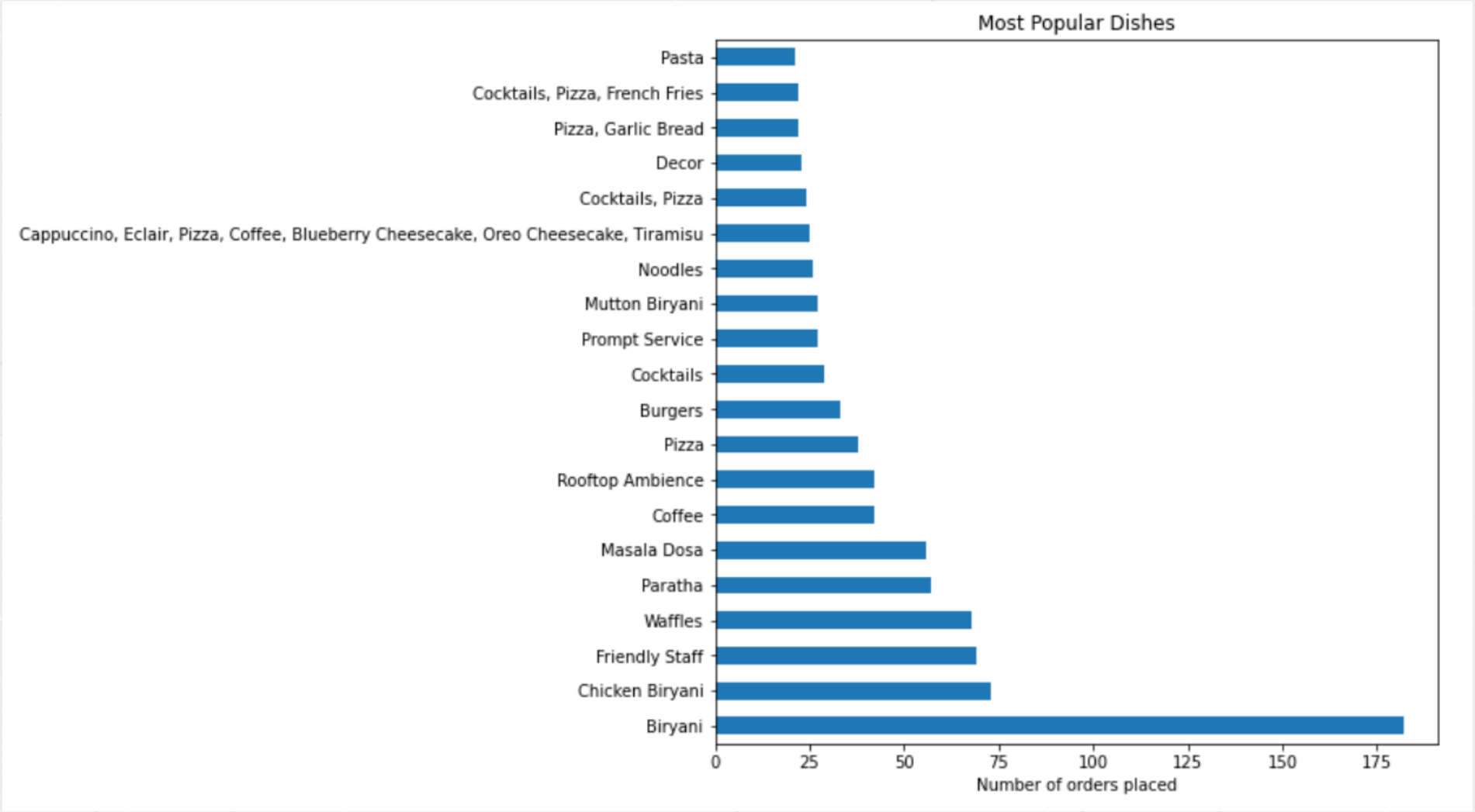


Does particular type of restaurants get more rating than other type?

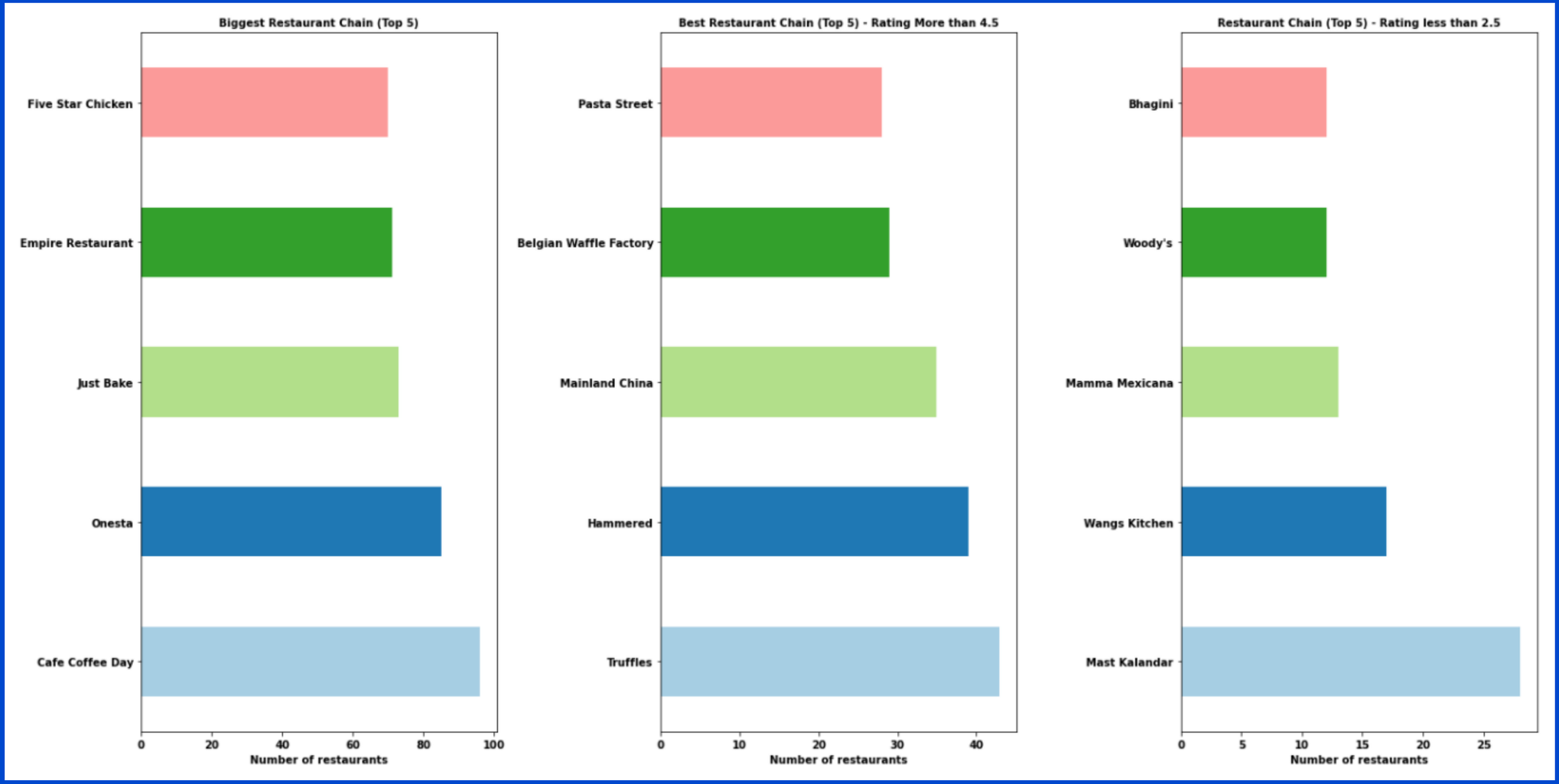




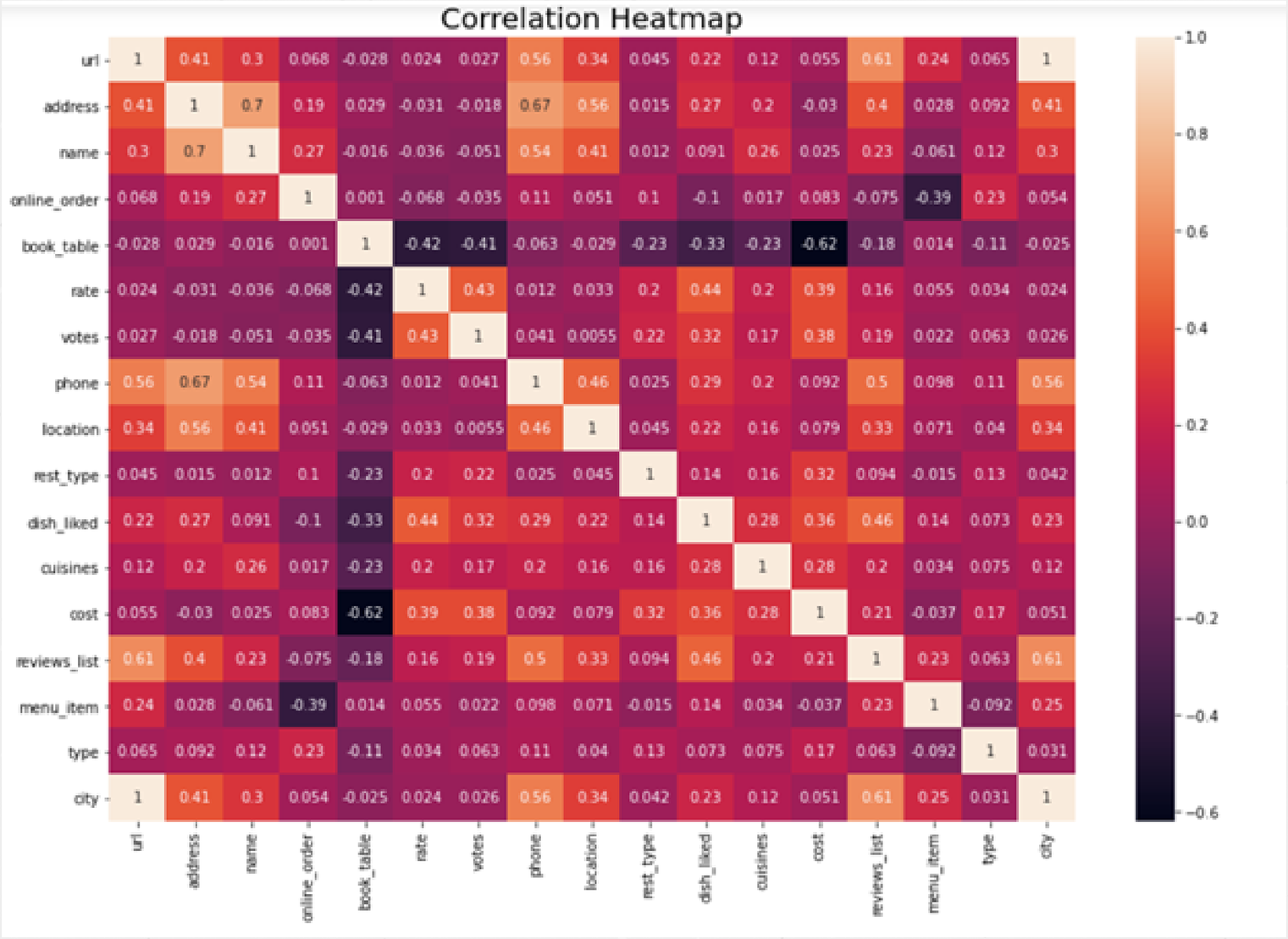
What are the dishes enjoyed by people the most?

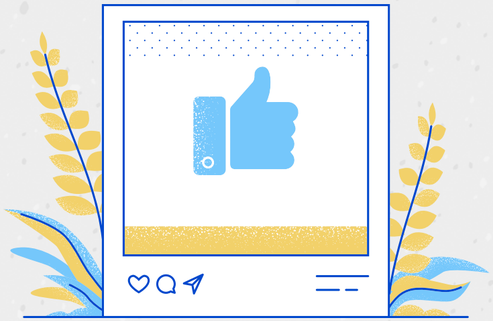


Comparing Biggest Restaurant Chain and Best Restaurant Chain



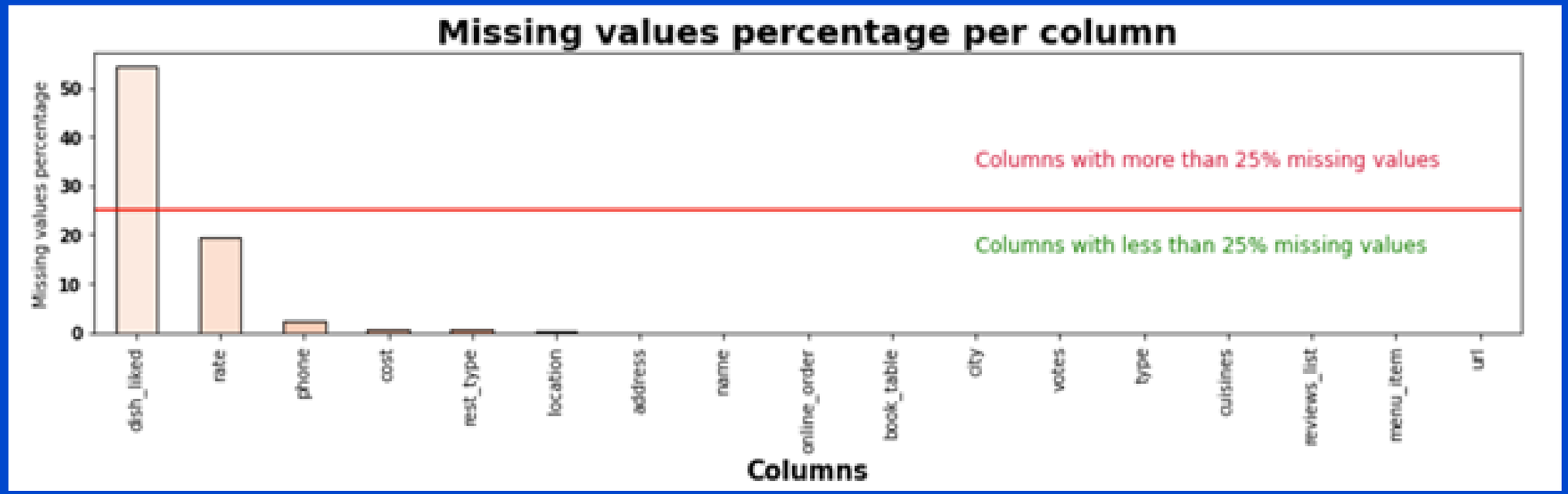
Correlation between features





Modeling

Exploring data to check the columns with the amount of missing data



Data Cleaning

```
# Renaming a few columns for ease of use
df = df.rename(columns={'approx_cost(for two people)':'cost','listed_in(type)':'type','listed_in(city)':'city'})
```

```
# Cleaning data
df['rate'] = df['rate'].replace('NEW',np.NaN)
df['rate'] = df['rate'].replace('-',np.NaN)

# Changing categorical features for statstical computation
df.online_order = df.online_order.apply(lambda x: '1' if str(x)=='Yes' else '0')
df.book_table = df.book_table.apply(lambda x: '1' if str(x)=='Yes' else '0')


# Changing datatype of features for statstical computation
df.rate = df.rate.astype(str)
df.rate = df.rate.apply(lambda x : x.replace('/5',''))
df.rate = df.rate.astype(float)

df.cost = df.cost.astype(str)
df.cost = df.cost.apply(lambda y : y.replace(',',''))
df.cost = df.cost.astype(float)

# Viewing all null values across columns
df.isnull().sum()
```

```
# Drop unwanted columns
df = df.drop(['url', 'address', 'phone', 'location'], axis = 1)
```

```
# Dropping all rows with null values
df.dropna(how='any',inplace=True)
```

```
# Removing duplicates
df.drop_duplicates(keep='first',inplace=True)
```

```
#Converting caterogical variables into dummy variables
dummy_rest_type = pd.get_dummies(df['rest_type'])
dummy_type = pd.get_dummies(df['type'])
dummy_city = pd.get_dummies(df['city'])
dummy_cuisines = pd.get_dummies(df['cuisines'])
dummy_dishliked = pd.get_dummies(df['dish_liked'])
```

```
# Concatenating dummy variables to the dataframe
df = pd.concat([df, dummy_rest_type, dummy_type, dummy_city, dummy_cuisines, dummy_dishliked],axis=1)
```

```
# Initializing Standard Scaler
scaler = StandardScaler()

# Standardizing features by removing the mean and scaling to unit variance
x_fit = scaler.fit_transform(df)

# Creating a new dataframe from the transformed data
X = pd.DataFrame(x_fit, columns=df.columns)

# Viewing the new dataframe
X.head()
```

```
# Dropping the target variable
X_init = X.drop(['rate'],axis=1)

# Dropping all columns except target variable
Y_init = X.drop(X.columns.difference(['rate']),axis=1)

# Initialzing the best feature selector
bestfeatures = SelectKBest(k='all')
fit = bestfeatures.fit(X_init, Y_init)
```

Models

- Linear Regressor
- Decision Trees
- Random Forests
- KNN
- SVR
- Gradient Boosting

Linear Regressor

```
# Initializing and fitting Linear Regressor
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
LinearRegression()
```

```
# Prediction from Linear Regressor algorithm
y_pred_lr = lr.predict(X_test)

acc_len=lr.score(X_test,y_test)
print("The accuracy  of LinearRegression is ",acc_len)

# R-squared value of Linear Regressor model
r2_score_lr = r2_score(y_test, y_pred_lr)

# Mean-squared value of Linear Regressor model
mse_score_lr = mean_squared_error(y_test, y_pred_lr)

print("R-2 score from the Linear Regressor is :", r2_score_lr, " Mean Squared Error is : ", mse_score_lr)
```

```
The accuracy  of LinearRegression is  0.7228886695659074
R-2 score from the Linear Regressor is : 0.7228886695659074  Mean Squared Error is :  0.04926423424519672
```

Results

The accuracy score of the linear Regression is 72% and not the accurate model for the dataset.

```
# Predictions for the first 10 samples from the dataset
top_pred = lr.predict(X_test.iloc[0:10, :])
print("Predictions of the first 10 examples from the test dataset \n")
for idx, v in enumerate(top_pred):
    print("True Rating : ", y_test.iloc[idx], " and, Predicted Rating :", v)
```

```
Predictions of the first 10 examples from the test dataset

True Rating :  4.1  and, Predicted Rating : 3.996442085103749
True Rating :  3.6  and, Predicted Rating : 3.9680454786584365
True Rating :  3.7  and, Predicted Rating : 3.959717994527577
True Rating :  4.0  and, Predicted Rating : 3.947175269918202
True Rating :  4.2  and, Predicted Rating : 3.906464820699452
True Rating :  4.4  and, Predicted Rating : 3.9481251295373427
True Rating :  3.5  and, Predicted Rating : 3.4545376356408584
True Rating :  3.7  and, Predicted Rating : 3.8433010633752334
True Rating :  3.9  and, Predicted Rating : 3.8615391310021865
True Rating :  4.2  and, Predicted Rating : 4.40632749159789
```

Decision Tree

```
dt = DecisionTreeRegressor(random_state=0, criterion="mae")
dt.fit(X_train, y_train)
```

```
DecisionTreeRegressor(criterion='mae', random_state=0)
```

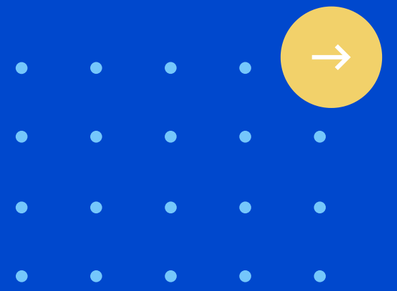
```
y_pred_dt = dt.predict(X_test)
r2_score_dt = r2_score(y_test, y_pred_dt)

acc_dt=dt.score(X_test,y_test)
print("The accuracy score  of Decision Tree is ",acc_dt)

acc_dtreg =dt.score(X_train,y_train)
mse_score_dt = mean_squared_error(y_test, y_pred_dt)

print("R-2 score from the Decision Tree Regressor is :", r2_score_dt, " Mean Squared Error is : ", mse_score_dt)
```

```
The accuracy score  of Decision Tree is  0.9043435574887287
R-2 score from the Decision Tree Regressor is : 0.9043435574887287  Mean Squared Error is :  0.01700558899398108
```



Results

We achived a Accuracy score for Decision tree is 90% which is considered as a good score.

```
# Predictions for the first 10 samples from the dataset
top_pred = dt.predict(X_test.iloc[0:10, :])
print("Predictions of the first 10 examples from the test dataset \n")
for idx, v in enumerate(top_pred):
    print("True Rating : ", y_test.iloc[idx], " and, Predicted Rating :", v)
```

```
Predictions of the first 10 examples from the test dataset

True Rating :  4.1  and, Predicted Rating : 4.1
True Rating :  3.6  and, Predicted Rating : 3.6
True Rating :  3.7  and, Predicted Rating : 3.7
True Rating :  4.0  and, Predicted Rating : 4.0
True Rating :  4.2  and, Predicted Rating : 3.9
True Rating :  4.4  and, Predicted Rating : 4.5
True Rating :  3.5  and, Predicted Rating : 3.5
True Rating :  3.7  and, Predicted Rating : 3.7
True Rating :  3.9  and, Predicted Rating : 3.9
True Rating :  4.2  and, Predicted Rating : 4.2
```

Random Forest Regressor

```
# Initializing and fitting training data to Random Forest with 100 estimators
rf_ds = RandomForestRegressor(random_state=42)
rf_ds.fit(X_train, y_train)
```

```
RandomForestRegressor(random_state=42)
```

```
# Prediction from Random Forest algorithm
y_pred_rf = rf_ds.predict(X_test.iloc[:, :])

acc_rf = rf_ds.score(X_test,y_test)
print("The accuracy of Random Forest",acc_rf)

# R-squared value of Random Forest model
r2_score_rf = r2_score(y_test, y_pred_rf)

# Mean-squared value of Random Forest model
mse_score_rf = mean_squared_error(y_test, y_pred_rf)

print("R-2 score from the Random Forest is :", r2_score_rf, " Mean Squared Error is : ", mse_score_rf)
```

```
The accuracy of Random Forest 0.9147626557205035
R-2 score from the Random Forest is : 0.9147626557205035  Mean Squared Error is :  0.015153304949478813
```

Results

The model provided the accuracy of 91% which is good score so far and the below are the 10 sample predicted values and true values.

```
# Predictions for the first 10 samples from the dataset
top_pred = rf_ds.predict(X_test.iloc[0:10, :])
print("Predictions of the first 10 examples from the test dataset \n")
for idx, v in enumerate(top_pred):
    print("True Rating : ", y_test.iloc[idx], " and, Predicted Rating :", v)
```

```
Predictions of the first 10 examples from the test dataset

True Rating :  4.1  and, Predicted Rating : 4.112000000000003
True Rating :  3.6  and, Predicted Rating : 3.600000000000003
True Rating :  3.7  and, Predicted Rating : 3.7039999999999935
True Rating :  4.0  and, Predicted Rating : 3.998
True Rating :  4.2  and, Predicted Rating : 3.9119999999999946
True Rating :  4.4  and, Predicted Rating : 4.112000000000002
True Rating :  3.5  and, Predicted Rating : 3.5
True Rating :  3.7  and, Predicted Rating : 3.707999999999994
True Rating :  3.9  and, Predicted Rating : 3.898999999999995
True Rating :  4.2  and, Predicted Rating : 4.229400000000001
```

K-Nearest Neighbors

```
knn = KNeighborsRegressor(n_jobs=-1)
knn.fit(X_train,y_train)
```

```
KNeighborsRegressor(n_jobs=-1)
```

```
Y_knn_pred=knn.predict(X_test)
```

```
r2_score(y_test,Y_knn_pred)
```

```
0.7936635760201317
```

```
r2_score_knn = r2_score(y_test, Y_knn_pred)

mse_score_knn = mean_squared_error(y_test, Y_knn_pred)

print("R-2 score from the K nearest neighbors is :", r2_score_knn, " Mean Squared Error is : ", mse_score_knn)
```

```
R-2 score from the K nearest neighbors is : 0.7936635760201317  Mean Squared Error is :  0.03668202923473774
```

The above model resulted in accuracy score of 79% which is low so and we applied hyperpamater to figureout if it could increase the accuracy score of the model.

K-Nearest Neighbors

```
# Performing Grid Search and 5-fold Cross Validation for K-Nearest Neighbours with neighbours 5,7,10
knn_ds = GridSearchCV(estimator=KNeighborsRegressor(), param_grid={"n_neighbors": [5, 7, 10]})

# Fitting training data to K-Nearest Neighbours
knn_ds.fit(X_train, y_train)
```

```
GridSearchCV(estimator=KNeighborsRegressor(),
             param_grid={'n_neighbors': [5, 7, 10]})
```

```
# Printing the results of Grid Search and Cross Valitdation
knn_ds.cv_results_
```

```
{'mean_fit_time': array([4.4000855 , 4.26626096, 4.46330295]),
 'std_fit_time': array([0.17868775, 0.03395525, 0.05245669]),
 'mean_score_time': array([74.00336108, 75.88904738, 82.2989109 ]),
 'std_score_time': array([1.20675512, 0.74180074, 0.58272211]),
 'param_n_neighbors': masked_array(data=[5, 7, 10],
              mask=[False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'n_neighbors': 5}, {'n_neighbors': 7}, {'n_neighbors': 10}],
 'split0_test_score': array([0.78134905, 0.72653033, 0.65322353]),
 'split1_test_score': array([0.76468991, 0.70478476, 0.6286947 ]),
 'split2_test_score': array([0.74038744, 0.69246395, 0.63109992]),
 'split3_test_score': array([0.75708477, 0.70957491, 0.64176906]),
 'split4_test_score': array([0.75476279, 0.70840592, 0.64223682]),
 'mean_test_score': array([0.75965479, 0.70835197, 0.6394048 ]),
 'std_test_score': array([0.0133997 , 0.010928  , 0.00881187]),
 'rank_test_score': array([1, 2, 3])}
```

```
# Best parameters for the K-Nearest neighbours algorithm
knn_ds.best_params_
```

```
{'n_neighbors': 5}
```

Results

The R-squared value is predicted after applying the hyperparameter turning using GridsearchCV and the accuracy increased to 91% which is really good after the tuning method is applied.

```
# Predicting target variable using the best estimator/model returned by GridSearch CV
y_pred_knn = knn_ds.best_estimator_.predict(X_test.iloc[:, :])

# R-squared value of KNN with 5 neighbours
r2_score_knn = r2_score(y_test, y_pred_knn)

# Mean-squared value of KNN with 5 neighbours
mse_score_knn = mean_squared_error(y_test, y_pred_knn)

print("R-2 score from the K nearest neighbors is :", r2_score_knn, " Mean Squared Error is : ", mse_score_knn)
```

```
R-2 score from the K nearest neighbors is : 0.7952848142721746  Mean Squared Error is :  0.03639380911435941
```

Support Vector Regression

```
# initialize and fit Support Vector Regressor to the training data
svr = SVR()
svr.fit(X_train, y_train)
```

```
SVR()
```

```
y_pred_svr = svr.predict(X_test.iloc[:, :])
r2_score_svr = r2_score(y_test, y_pred_svr)


acc_svr = svr.score(X_test,y_test)
print("The accuracy score  of Support Vector",acc_svr)

mse_score_svr = mean_squared_error(y_test, y_pred_svr)

print("R-2 score from the K nearest neighbors is :", r2_score_svr, " Mean Squared Error is : ", mse_score_svr)
```

```
The accuracy score  of Support Vector 0.7271518569399371
R-2 score from the K nearest neighbors is : 0.7271518569399371  Mean Squared Error is :  0.04850633430261273
```

Results

The R-squared value of support vector regression is 0.72 which is least among the other models.

```
# Predictions for the first 10 samples from the dataset
top_pred = svr.predict(X_test.iloc[0:10, :])
print("Predictions of the first 10 examples from the test dataset \n")
for idx, v in enumerate(top_pred):
    print("True Rating : ", y_test.iloc[idx], " and, Predicted Rating :", v)
```

```
Predictions of the first 10 examples from the test dataset

True Rating :  4.1  and, Predicted Rating : 4.011726129845797
True Rating :  3.6  and, Predicted Rating : 3.9732280407254628
True Rating :  3.7  and, Predicted Rating : 3.977661162764552
True Rating :  4.0  and, Predicted Rating : 3.9287904031865835
True Rating :  4.2  and, Predicted Rating : 3.9212112013638514
True Rating :  4.4  and, Predicted Rating : 3.977846907892253
True Rating :  3.5  and, Predicted Rating : 3.522074009853781
True Rating :  3.7  and, Predicted Rating : 3.8728747041793925
True Rating :  3.9  and, Predicted Rating : 3.8881156667112258
True Rating :  4.2  and, Predicted Rating : 4.323433443052902
```

Gradient Boosting

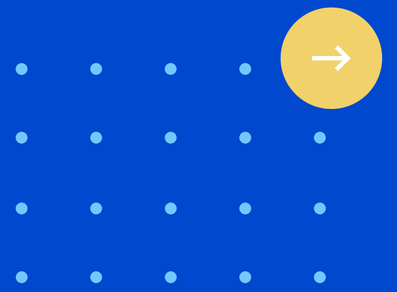
```
gbr = GradientBoostingRegressor(loss="huber")
gbr.fit(X_train, y_tr
        ain)
```

```
GradientBoostingRegressor(loss='huber')
```

```
# Performing Grid Search and 5-fold Cross Validation for Gradient Boosting Regressor
gb_ds = GridSearchCV(estimator=GradientBoostingRegressor(), param_grid={"loss": ["ls", "lad", "huber"],
                                                                        "n_estimators": [100, 200]})

# Fitting training data to Gradient Boosting Regressor
gb_ds.fit(X_train, y_train)
```

```
GridSearchCV(estimator=GradientBoostingRegressor(),
             param_grid={'loss': ['ls', 'lad', 'huber'],
                         'n_estimators': [100, 200]})
```



Results

The R-squared value of the Gradient Boosting is 0.51 and its not a conventional method for the dataset and displayed the samples of datset with predicted and true values.

```
gb_ds.best_params_
```

```
{'loss': 'ls', 'n_estimators': 200}
```

```
y_pred_gbr = gb_ds.best_estimator_.predict(X_test.iloc[:, :])
r2_score_gbr = r2_score(y_test, y_pred_gbr)

mse_score_gbr = mean_squared_error(y_test, y_pred_gbr)

print("R-2 score from the Gradient Boosting Regressor :", r2_score_gbr, " Mean Squared Error is : ", mse_score_gbr)
```

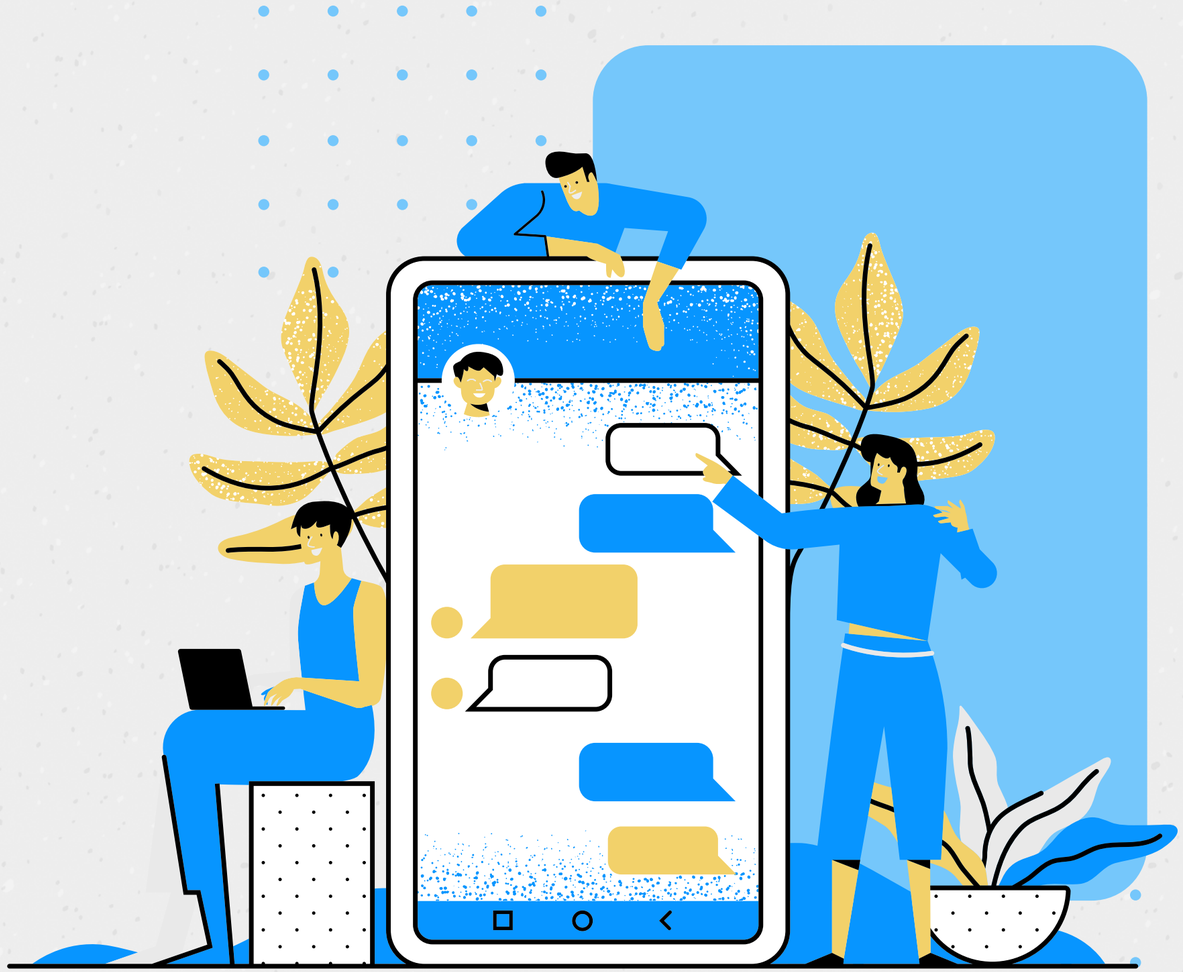
```
R-2 score from the Gradient Boosting Regressor : 0.5124206417061602  Mean Squared Error is :  0.08668077080241698
```

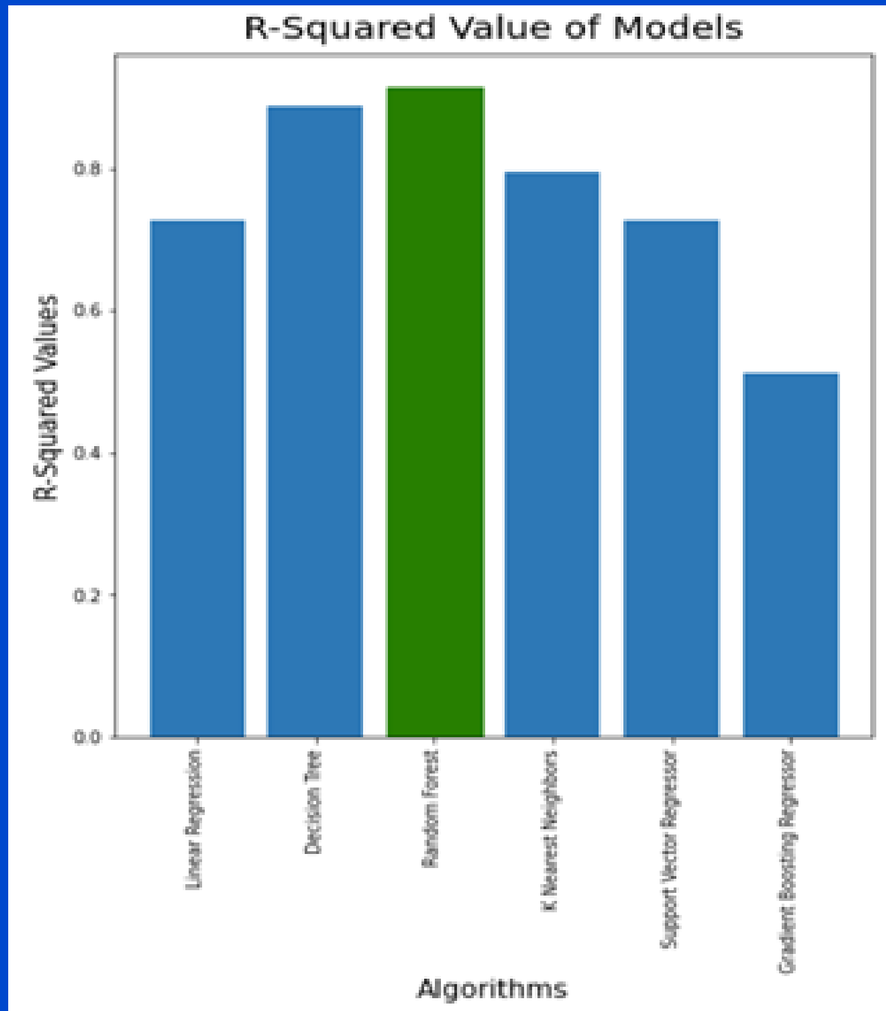
```
# Predictions for the first 10 samples from the dataset
top_pred = gb_ds.best_estimator_.predict(X_test.iloc[0:10, :])
print("Predictions of the first 10 examples from the test dataset \n")
for idx, v in enumerate(top_pred):
    print("True Rating : ", y_test.iloc[idx], " and, Predicted Rating :", v)
```

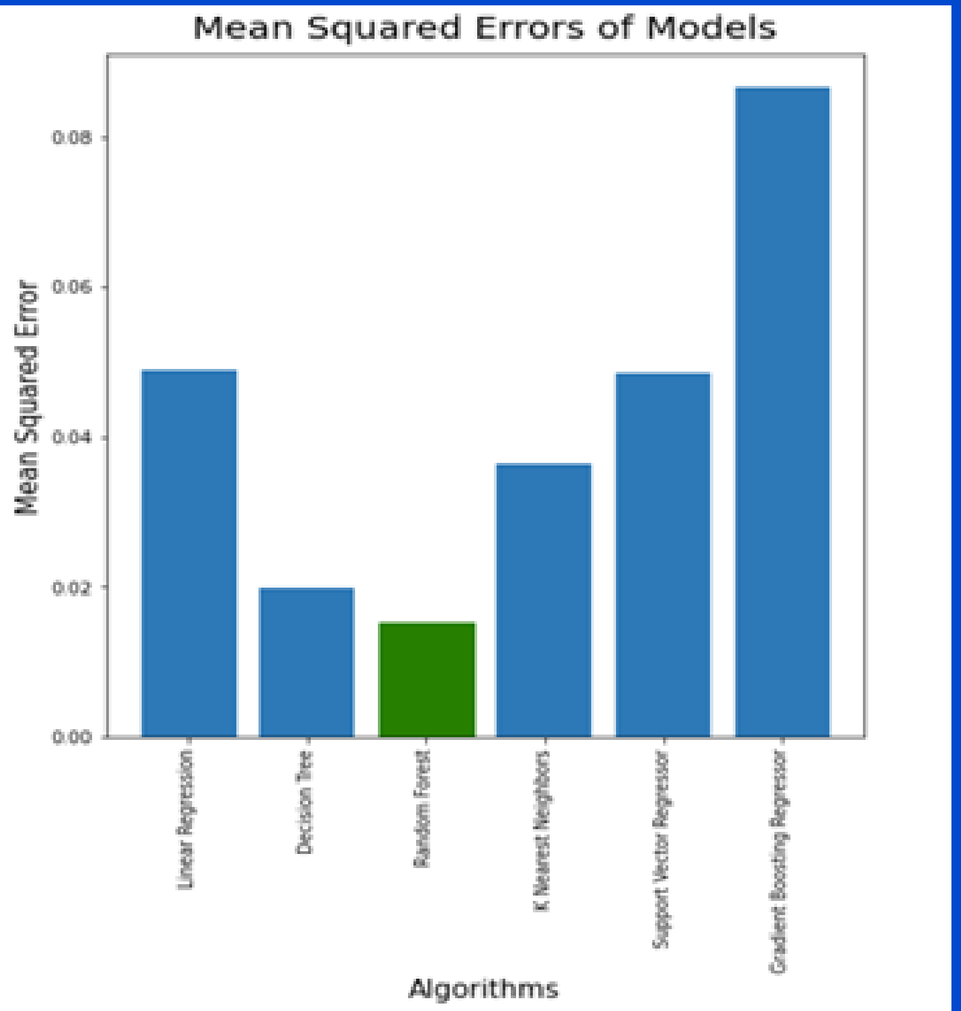
```
Predictions of the first 10 examples from the test dataset

True Rating :  4.1  and, Predicted Rating : 3.9709868921059366
True Rating :  3.6  and, Predicted Rating : 3.7374472713028357
True Rating :  3.7  and, Predicted Rating : 3.7262180375419067
True Rating :  4.0  and, Predicted Rating : 3.852235184710125
True Rating :  4.2  and, Predicted Rating : 3.84452259337817
True Rating :  4.4  and, Predicted Rating : 4.064152403630821
True Rating :  3.5  and, Predicted Rating : 3.712986750284953
True Rating :  3.7  and, Predicted Rating : 3.7782441137290315
True Rating :  3.9  and, Predicted Rating : 3.70266457374031
True Rating :  4.2  and, Predicted Rating : 4.411506545600348
```

Models Comparison



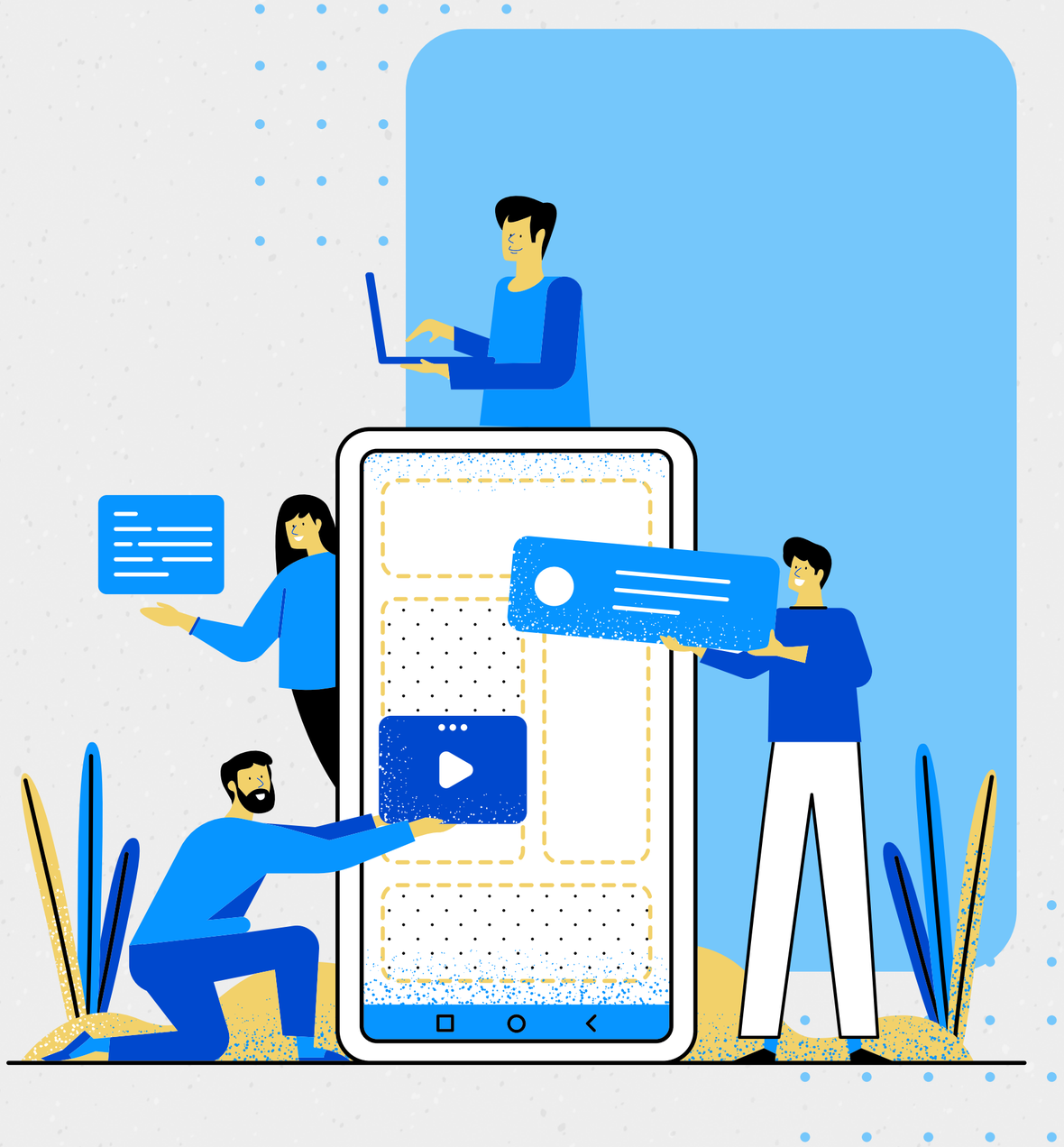

R-Squared Value of Models
R-Squared Values
0.0
0.2
0.4
0.6
0.8
Linear Regression
Decision Tree
Random Forest
K Nearest Neighbors
Support Vector Regressor
Gradient Boosting Regressor
Algorithms


Mean Squared Errors of Models
Mean Squared Error
0.00
0.02
0.04
0.06
0.08
Linear Regression
Decision Tree
Random Forest
K Nearest Neighbors
Support Vector Regressor
Gradient Boosting Regressor
Algorithms

Conclusion

From the above models applied we conclude that random forest regression gave the R-squared value of 0.91 along with KNN regression with hyperparameter tuning.The lowest among them was gradient boosting with 51% which is not conventional method for the chosen dataset.

Thank You



Questions?