Data Cleaning, Processing and Food Delivery Time Prediction

Md Shezan Ahmed

Date: 06 March 2025

Contents

Executive Summary

This report focuses on forecasting food delivery durations through machine learning techniques. Precise predictions of delivery times are crucial for customer contentment and the success of a business. When food is delivered late, clients might be dissatisfied and cease using the service. Utilizing historical delivery information, companies can enhance their forecasts and elevate their service quality.

Improved accuracy in delivery time estimates aids businesses in planning more effectively and keeping customers updated about their orders. This enhances customer trust and loyalty. In the future, incorporating real-time traffic and weather information can enhance the accuracy of predictions significantly. Companies can also investigate deep learning methods for improved outcomes. Through the application of these techniques, food delivery businesses can enhance their offerings, expand their operations, and maintain a competitive edge.

Introduction

Predictive analytics has become essential in business decision-making, especially in industries that need prompt and effective service provision. In the food delivery sector, precisely estimating delivery times is vital for boosting customer satisfaction, streamlining operations, and increasing brand loyalty (Sharma et al., 2020). In this project, we utilize multiple predictive models, such as Linear Regression, Polynomial Regression, K-Nearest Neighbors (KNN), and Decision Trees, to estimate the delivery time for food orders depending on various factors like distance, traffic conditions, and weather information. Utilizing machine learning methods, the objective is to create a model that can precisely predict delivery times, guaranteeing that customers get their orders promptly.

Prompt food delivery is crucial for sustaining customer confidence and minimizing order cancellations, especially during peak times when demand increases (Chen et al., 2021). Various elements, such as unforeseen weather conditions, traffic jams, and the complexity of orders, lead to delays in delivery. Earlier research has shown that incorporating machine learning methods into logistics processes can greatly enhance delivery time forecasting and overall effectiveness (Ghosh et al., 2022). Through the examination of past data and the use of predictive models, companies can take proactive measures to tackle delays, distribute resources efficiently, and offer customers more precise estimated delivery times, which ultimately enhances satisfaction and operational efficiency.

Dataset Selection

The dataset used for this project is sourced from Kaggle (Oliveira, 2023), containing various features related to food delivery processes, such as delivery distance, traffic conditions, and weather data. These features serve as predictors, while delivery_time_minutes is the target variable. The data was processed and cleaned to ensure it was suitable for modeling, including handling missing values, scaling numerical features, and encoding categorical variables (Kuhn & Johnson, 2013).

Dataset Link: https://www.kaggle.com/datasets/willianoliveiragibin/food-delivery-time

Feature Selection

The dataset had 17 columns and 10,000 rows. However, there were many features (columns) which were unnecessary, or other columns would have represented it. That is why several features were deleted before continuing further in the analytics.

Features like ID and Delivery_person_ID did not serve any purpose towards the target feature delivery_time_minutes, which is the delivery time. Moreover, the restaurant and delivery location latitude and longitude were given and another column Distance (km) represented those features perfectly. That is why those features were not selected.

Final Features that were not used from the original dataset:

1. Restaurant_latitude
2. Restaurant_longitude
3. Delivery_location_latitude
4. Delivery_location_longitude

5. ID
6. Delivery_person_ID

Final Features that were used from the original dataset:

1. Traffic_Level
2. weather_description
3. Type_of_order
4. Type_of_vehicle
5. Delivery_person_Age
6. Delivery_person_Ratings
7. temperature_c
8. humidity
9. precipitation
10. Distance (km)
11. delivery_time_minutes

Data Quality Resolution

There were several data quality issues were identified in the dataset. First of all the target feature was named TARGET so we changed the name from TARGET to delivery_time_minutes and temperature to temperature_c for better understanding.

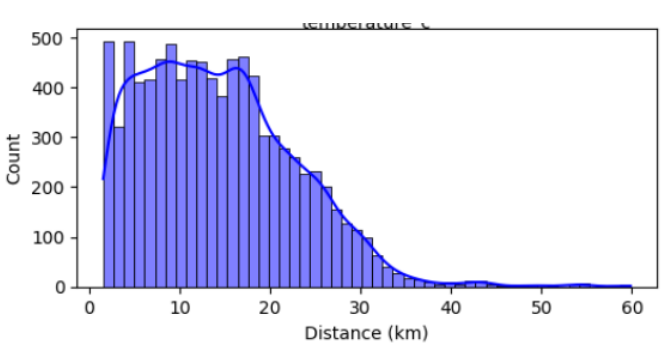
```
[134] dataset.rename(columns={'temperature': 'temperature_c', 'TARGET': 'delivery_time_minutes'}, inplace=True)
```

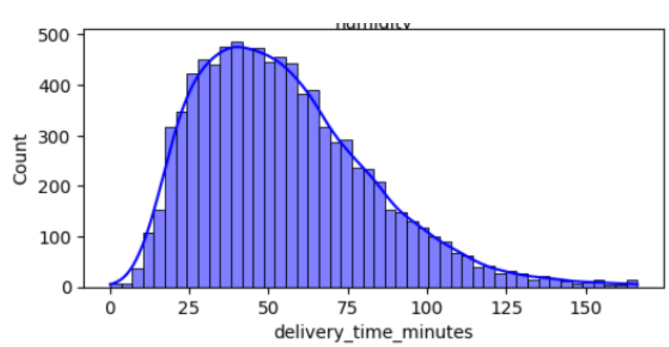
The target variable was delivery time in minutes. However, there were several entries with quality issues. Some of the entries were 3.816.666.667 like this. Which did not made any sense. To solve this problem we first removed all the dots from the number to make it an int. Then divided it by 60000000 assuming that the value was in microseconds and converted it into minutes and we kept the normal data as it was.

```
139] def convert_time(x):
        try:

            x = str(x)

            if x.count('.') > 1:
                x = x.replace('.', '')

            num = float(x)

            if len(x) > 5:
                return round(num / 60000000, 2)
            else:
                return num
        except ValueError:
            return None

    dataset['delivery_time_minutes'] = dataset['delivery_time_minutes'].apply(convert_time)
```

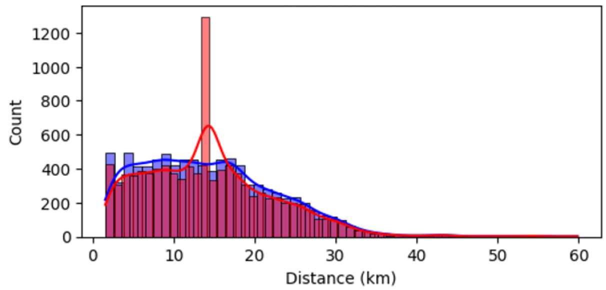
We had some nun values in the dataset. First, we tried to replace the null values with appropriate mode and mean values. However, most of the null values were in our target feature and it dramatically changed our graph.

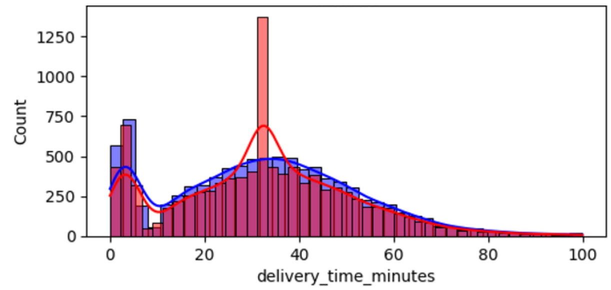
Before Replacing the Null Values:





After Replacing the Null Values:



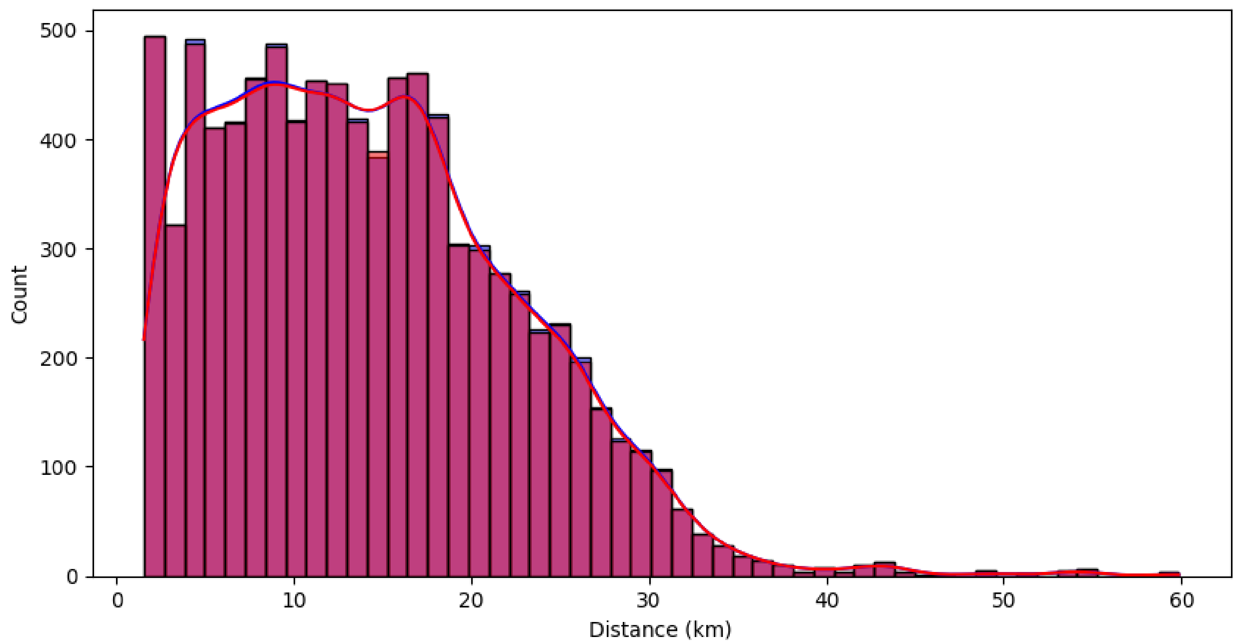


For the categorical columns it was fine with the mode values. However, with the number columns after replacing with the mean values, changed the graph dramatically as we could see above. It will not be good for the analytics model either regression, knn or decision tree.

For that reason, we decided to drop the rows which contained null values in our target feature (delivery_time_minutes).

```
dataset_cleaned = dataset.copy()
indexes_to_drop = []
for i in range(len(dataset_cleaned)):
    if pd.isna(dataset_cleaned.loc[i, 'delivery_time_minutes']):
        indexes_to_drop.append(i)
dataset_cleaned = dataset_cleaned.drop(indexes_to_drop)
dataset_cleaned = dataset_cleaned.reset_index(drop=True)
```

After that we had only 5 missing values in Distance (km) column and we replaced those null values with the mean number which did not affect the dataset that much as shown below.



Final Analytical Base Table

Features	Description
Traffic_Level	Traffic conditions at the time of delivery
weather_description	Condition of the weather at the time of delivery
Type_of_order	What kind if order is this (snack, meal, drinks)
Type_of_vehicle	Type of vehicle the delivery personnel will use to deliver
Delivery_person_Age	Age of the delivery personnel
Delivery_person	Delivery person's ratings on the platform
Temperature_c	Temperature of the outside environment
humidity	Humidity of the outside environment
precipitation	rain, snow, sleet, or hail
Distance (km)	Distance between restaurant and delivery place in Kilometres
delivery_time_minutes	Target feature. Delivery time it took to deliver the order

Data Preprocessing

Our objective was to predict delivery_time_minutes based on previous data. The target feature is not a categorical column it is a continuous number. For that reason, we used regressor of KNN and decision trees and we also did multi linear regression, polynomial regression and piece wise regression.

For that we had to do one-hot encoding because we had several categorical columns such as Traffic_level, weather_description, etc. Using the below code we did the one-hot encoding.

```
X_encoded_lr = dataset_encoded.drop(columns=['delivery_time_minutes']).values
y_lr = dataset_encoded['delivery_time_minutes'].values

scaler_lr = StandardScaler()
X_lr = scaler_lr.fit_transform(X_encoded_lr)
```

We will use this dataset_encoded for our analysis. Along with that we also scaled our data for regression analysis using StandardScaler. For regression analysis we normally use StandardScaler.

Data Splitting

The dataset was split into 80:20 ratio, where 80% of the data was for training and 20% of the data was for testing.

```
X_train_lr, X_test_lr, y_train_lr, y_test_lr = train_test_split(X_lr, y_lr, test_size=0.2, random_state=42)
```

Model Training

Regression Model

For the regression model we used three different kinds of regression models to get the optimal model for the analysis and get the best results.

a. Multiple Linier Regression

For Multiple Linier Regression, we used the simple linier regression with all the features except the target feature. Below is the whole model's code block.

```
X_encoded_lr = dataset_encoded.drop(columns=['delivery_time_minutes']).values
y_lr = dataset_encoded['delivery_time_minutes'].values

scaler_lr = StandardScaler()
X_lr = scaler_lr.fit_transform(X_encoded_lr)

X_train_lr, X_test_lr, y_train_lr, y_test_lr = train_test_split(X_lr, y_lr, test_size=0.2, random_state=42)

regressor_lr = LinearRegression()
regressor_lr.fit(X_train_lr, y_train_lr)

y_pred_train_lr = regressor_lr.predict(X_train_lr)
y_pred_test_lr = regressor_lr.predict(X_test_lr)
```

After training the model we used the quality metrics MAE and MSE to evaluate the model. As a result we got the below results for Multiple Linear Regression:

- The train MSE (LR): 344.44100861334687
- The train MAE (LR): 13.114697370070816
- The test MSE (LR): 289.17148590921414
- The test MAE (LR): 12.528909870367624

```
y_pred_train_lr = regressor_lr.predict(X_train_lr)
y_pred_test_lr = regressor_lr.predict(X_test_lr)

train_mse_lr = mean_squared_error(y_train_lr, y_pred_train_lr)
test_mse_lr = mean_squared_error(y_test_lr, y_pred_test_lr)
train_mae_lr = mean_absolute_error(y_train_lr, y_pred_train_lr)
test_mae_lr = mean_absolute_error(y_test_lr, y_pred_test_lr)
```

b. Piece Wise Regression

For Piece Wise Regression, we had to do something different. First, we had to split the dataset into groups and we will do the same multiple linier regression but group wise. For our dataset we choose the Traffic_Level as our group. It had 5 values, and for each value it will split the dataset and do the regression on each of the 5 categories.

```python
dataset_encoded_pw = dataset_encoded.copy()
dataset_encoded_pw['Traffic_Level'] = dataset_cleaned['Traffic_Level']
traffic_groups = dataset_encoded_pw['Traffic_Level'].unique()
models_pw = {}

for traffic in traffic_groups:

    group_pw = dataset_encoded_pw[dataset_encoded_pw['Traffic_Level'] == traffic]
    X_group_pw = group_pw.drop(['delivery_time_minutes', 'Traffic_Level'], axis=1).values
    y_group_pw = group_pw['delivery_time_minutes'].values

    X_train_pw, X_test_pw, y_train_pw, y_test_pw = train_test_split(X_group_pw, y_group_pw, test_size=0.1, random_state=42)

    regressor_pw = LinearRegression()
    regressor_pw.fit(X_train_pw, y_train_pw)
```

We also used quality metrics MAE and MSE to evaluate this model. And got the below results.

	Traffic Level	Train MSE	Test MSE	Train MAE	Test MAE
0	**High**	202.165175	214.943149	12.306581	12.597095
1	**Low**	56.325854	55.039010	6.555699	6.309544
2	**Moderate**	110.412344	116.606749	9.210721	9.502729
3	**Very High**	610.400375	572.847270	19.609180	18.979768
4	**Very Low**	1259.115479	954.448128	23.961314	21.441254

From this result we can see that the error are very low for the **Low** and **Moderate** groups, average in the **High** group and very high in the **Very High and Very Low** groups.

```python
    y_pred_train_pw = regressor_pw.predict(X_train_pw)
    y_pred_test_pw = regressor_pw.predict(X_test_pw)

    mse_train_pw = mean_squared_error(y_train_pw, y_pred_train_pw)
    mse_test_pw = mean_squared_error(y_test_pw, y_pred_test_pw)
    mae_train_pw = mean_absolute_error(y_train_pw, y_pred_train_pw)
    mae_test_pw = mean_absolute_error(y_test_pw, y_pred_test_pw)

    models_pw[traffic] = {
        'model': regressor_pw,
        'mse_train': mse_train_pw,
        'mse_test': mse_test_pw,
        'mae_train': mae_train_pw,
        'mae_test': mae_test_pw
    }

results_pw = []

for traffic, result in models_pw.items():
    results_pw.append({
        'Traffic Level': traffic,
        'Train MSE': result['mse_train'],
        'Test MSE': result['mse_test'],
        'Train MAE': result['mae_train'],
        'Test MAE': result['mae_test']
    })

results_df_pw = pd.DataFrame(results_pw)

print(results_df_pw)
```

c. Polynomial Regression

For Polynomial Regression, was similar to multi linear regression. However, it had an extra attribute which was degree. We selected degree = 2 because it was giving us the best results.

```
X_encoded_poly = dataset_encoded.drop(columns=['delivery_time_minutes'])
y_poly = dataset_encoded['delivery_time_minutes']

X_train_poly, X_test_poly, y_train_poly, y_test_poly = train_test_split(X_encoded_poly, y_poly, test_size=0.2, random_state=42)

poly_reg = PolynomialFeatures(degree=2)
X_train_poly_transformed = poly_reg.fit_transform(X_train_poly)
X_test_poly_transformed = poly_reg.transform(X_test_poly)

poly_model = LinearRegression()
poly_model.fit(X_train_poly_transformed, y_train_poly)
```

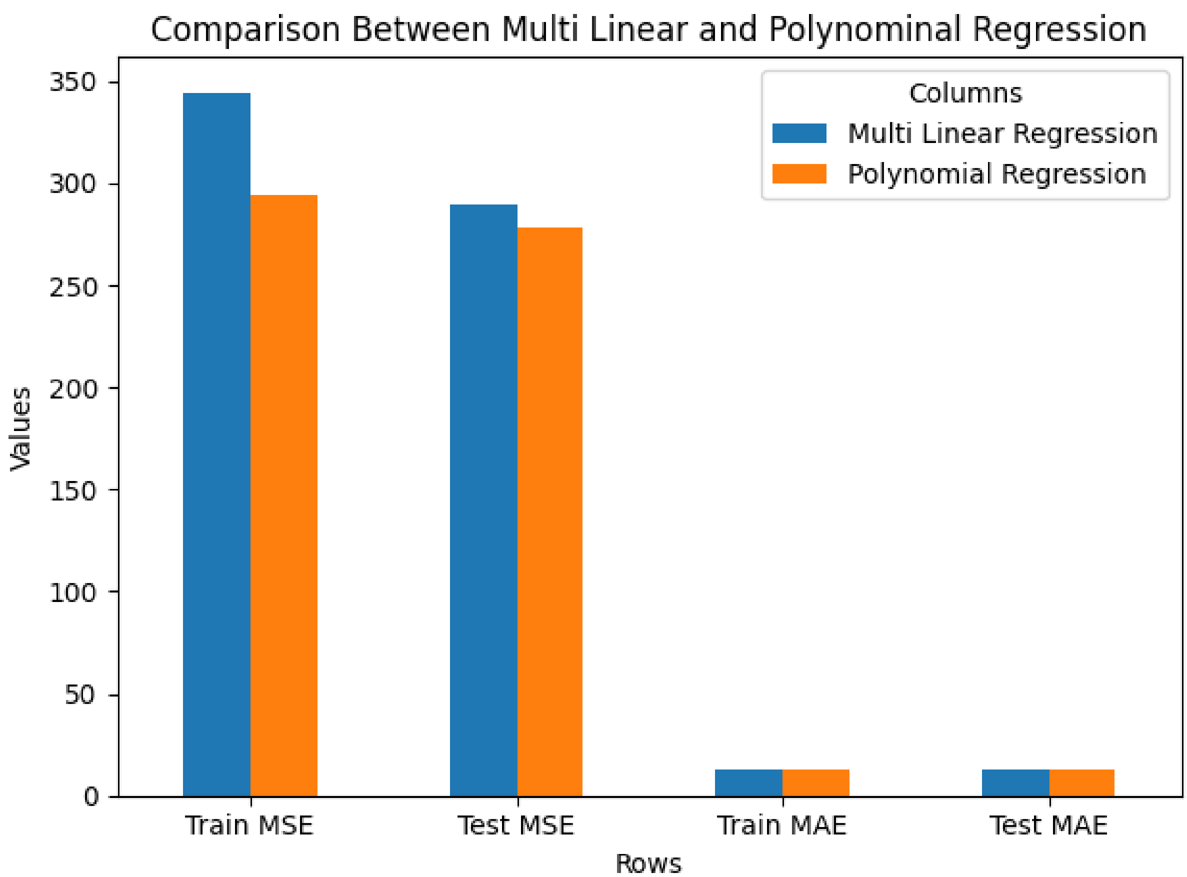
For the polynomial regression, we again used MAE and MSE to evaluate the model and got the below results.

- Train MSE (Poly): 293.99424477425015
- Test MSE (Poly): 278.67859266552733
- Train MAE (Poly): 12.519386720758908
- Test MAE (Poly): 12.568732867004863

```
y_train_pred_poly = poly_model.predict(X_train_poly_transformed)
y_test_pred_poly = poly_model.predict(X_test_poly_transformed)

train_mse_poly = mean_squared_error(y_train_poly, y_train_pred_poly)
test_mse_poly = mean_squared_error(y_test_poly, y_test_pred_poly)

train_mae_poly = mean_absolute_error(y_train_poly, y_train_pred_poly)
test_mae_poly = mean_absolute_error(y_test_poly, y_test_pred_poly)

print(f'Train MSE (Poly): {train_mse_poly}')
print(f'Test MSE (Poly): {test_mse_poly}')
print(f'Train MAE (Poly): {train_mae_poly}')
print(f'Test MAE (Poly): {test_mae_poly}')
```

Comparison Between Multi Linear and Polynomial Regression

As we can see from the below graph that the results for the Polynomial Regression's quality metric MSE is way lower than multi linear regression's metric in the training and test data. However, the MAE difference is very insignificant. From that we can conclude that polynomial regression suits our data better.



K Nearest Neighbor Regressor (KNN)

Apart from regression we also used KNN to predict our target feature. For knn we used knn regressor and created a function called evaluate_model_knn. So that we can modify different parameters easily. The reason is we will use nested for loop to find the best parameters and result for the knn model.

```python
y_knn = dataset_encoded['delivery_time_minutes']
X_knn = dataset_encoded.drop('delivery_time_minutes', axis=1)

scaler_knn = MinMaxScaler()
X_scaled_knn = scaler_knn.fit_transform(X_knn)

X_train_knn, X_test_knn, y_train_knn, y_test_knn = train_test_split(X_scaled_knn, y_knn, test_size=0.2, random_state=42)

model_default_knn = KNeighborsRegressor()
model_default_knn.fit(X_train_knn, y_train_knn)

def evaluate_model_knn(model, X_train, X_test, y_train, y_test):
    y_train_pred_knn = model.predict(X_train)
    y_test_pred_knn = model.predict(X_test)

    train_mse_knn = mean_squared_error(y_train, y_train_pred_knn)
    train_mae_knn = mean_absolute_error(y_train, y_train_pred_knn)

    print("Train MSE KNN: ", train_mse_knn)
    print("Train MAE KNN: ", train_mae_knn)

    test_mse_knn = mean_squared_error(y_test, y_test_pred_knn)
    test_mae_knn = mean_absolute_error(y_test, y_test_pred_knn)

    print("Test MSE KNN: ", test_mse_knn)
    print("Test MAE KNN: ", test_mae_knn)

evaluate_model_knn(model_default_knn, X_train_knn, X_test_knn, y_train_knn, y_test_knn)
```

From this model we get the below result, which looks worse than polynomial regression. However, we will do the nested for loop to find the best results.

- Train MSE KNN: 252.99685333517698
- Train MAE KNN: 10.96027931415929
- Test MSE KNN: 305.2333487986726
- Test MAE KNN: 12.903771017699116

We used nested for loops to get the best parameters and results for the knn model.

```python
results_knn = []

param_grid_knn = {
    'n_neighbors': range(3, 15),
    'weights': ['uniform', 'distance'],
    'p': [1, 2],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
}

for n in param_grid_knn['n_neighbors']:
    for w in param_grid_knn['weights']:
        for p in param_grid_knn['p']:
            for algo in param_grid_knn['algorithm']:

                model_knn = KNeighborsRegressor(n_neighbors=n, weights=w, p=p, algorithm=algo)
                model_knn.fit(X_train_knn, y_train_knn)

                y_train_pred_knn = model_knn.predict(X_train_knn)
                y_test_pred_knn = model_knn.predict(X_test_knn)

                metrics_knn = {
                    "Train MSE KNN": mean_squared_error(y_train_knn, y_train_pred_knn),
                    "Train MAE KNN": mean_absolute_error(y_train_knn, y_train_pred_knn),
                    "Test MSE KNN": mean_squared_error(y_test_knn, y_test_pred_knn),
                    "Test MAE KNN": mean_absolute_error(y_test_knn, y_test_pred_knn),
                }

                results_knn.append({
                    'n_neighbors': n,
                    'weights': w,
                    'p': p,
                    'algorithm': algo,
                    **metrics_knn
                })
```

The results we got is below.

- **Best Configuration for KNN:**
- n_neighbors: 12
- weights: uniform
- p: 1
- algorithm: auto

- **Quality Metrics for Best KNN Model:**
- Train MSE KNN: 282.92926481071777
- Train MAE KNN: 11.954096607669618
- Test MSE KNN: 267.33119531557276
- Test MAE KNN: 12.276785582595869

Here we can see it performed better than the polynomial regressor. We will use graphs to better clarify.

Decision Tree

Along with KNN and regression, we also used decision tree to predict our target feature. In this case we used decision tree regressor. Decision tree is not the best for when there is so many features. It could easily be overfitted.

```
y_dt = dataset_encoded['delivery_time_minutes']
X_dt = dataset_encoded.drop('delivery_time_minutes', axis=1)

X_train_dt, X_test_dt, y_train_dt, y_test_dt = train_test_split(X_dt, y_dt, test_size=0.2, random_state=42)

model_dt = DecisionTreeRegressor()
model_dt.fit(X_train_dt, y_train_dt)

y_train_pred_dt = model_dt.predict(X_train_dt)
y_test_pred_dt = model_dt.predict(X_test_dt)

mse_train_dt = mean_squared_error(y_train_dt, y_train_pred_dt)
mae_train_dt = mean_absolute_error(y_train_dt, y_train_pred_dt)

mse_test_dt = mean_squared_error(y_test_dt, y_test_pred_dt)
mae_test_dt = mean_absolute_error(y_test_dt, y_test_pred_dt)

print(f"Train MSE DT: {mse_train_dt}")
print(f"Train MAE DT: {mae_train_dt}")
print(f"Test MSE DT: {mse_test_dt}")
print(f"Test MAE DT: {mae_test_dt}")
```

From the above code we got the below results.

- Train MSE DT: 0.002331602599557521
- Train MAE DT: 0.0010301438053097343
- Test MSE DT: 542.4501901548673
- Test MAE DT: 15.470365044247787

Here we can see the results are the worst out of all the models. The reason is there is a lot of features and for that reason it has been overfitted. That’s why the train MSE and MAE is close to 0 and the test MSE and MAE so high.

To combat that we will use a similar tactic used for KNN, which is nested for loops to get the best results, and parameters. After running the loops, we got the below results.

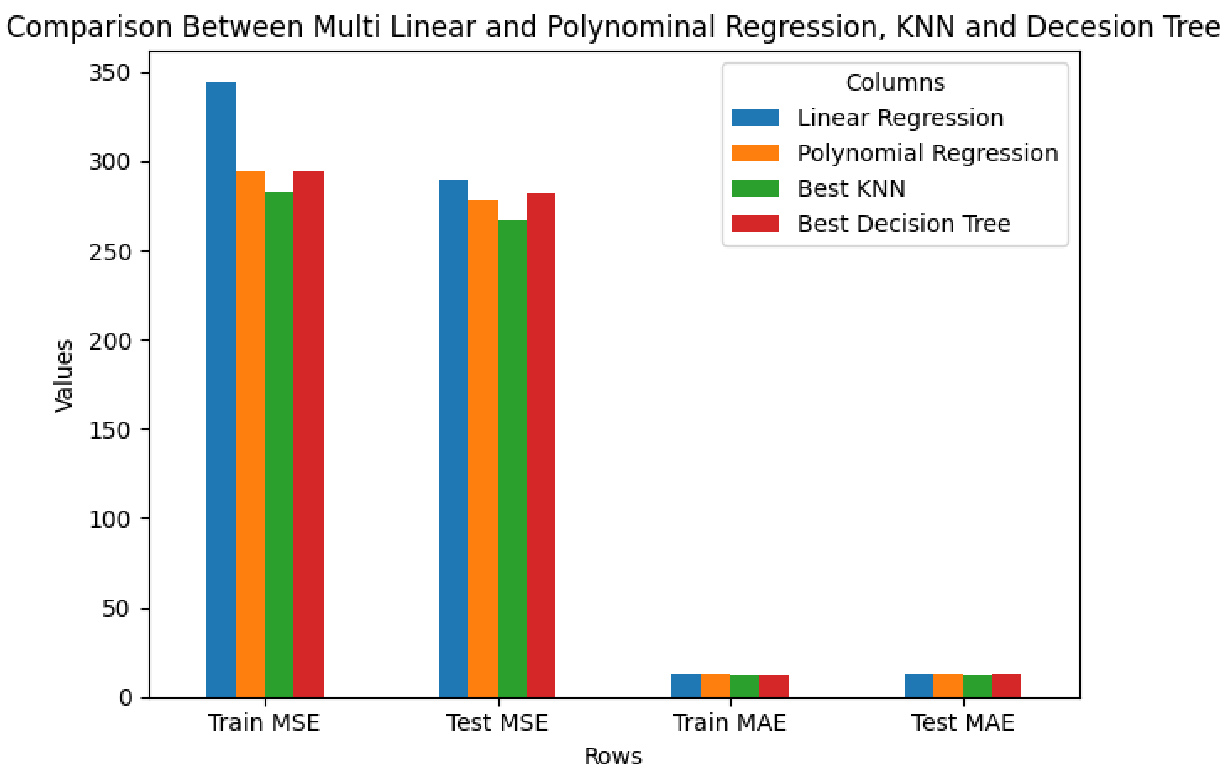
- **Best Configuration for Decision Tree Tuning:**
- max_depth: 7
- criterion: squared_error
- min_samples_leaf: 10
- min_samples_split: 2

- **Quality Metrics for Best Model:**
- MSE Train: 294.4553754013948
- MAE Train: 12.295406785658502
- MSE Test: 282.2938764576383
- MAE Test: 12.464115277068137

Now the results are similar to the previous models.

Best Model Selection

Now that we have all the results for all the models. Let's compare them in a graph.



The graph shows the difference between the 4 models. From where we can see that Multi Linear Regression has the highest error rate in MSE train and test. Where as best knn result has the lowest error rate in MSE and MAE train and test, from which we can conclude that the best model for our dataset is KNN. Though we have to find the best parameters using the nested for loop.

Conclusion

This report shows how predictive analytics can enhance food delivery services by estimating delivery times using historical data. We utilized many machine learning models like Linear Regression, Polynomial Regression, K-Nearest Neighbors (KNN), and Decision Trees to examine the different elements impacting delivery time. Our findings indicate that some models do better than others in various scenarios. Polynomial Regression and KNN provided improved predictions in specific instances, whereas Decision Trees required adjustment to prevent overfitting. These results align with earlier studies, indicating that machine learning can enhance the precision of delivery times and boost customer satisfaction (Sharma et al., 2020; Ghosh et al., 2022).

Accurately forecasting delivery times is crucial in the food delivery industry. Customers anticipate their food arriving punctually, and any delays may result in complaints, negative reviews, and a decline in customers. Through the application of predictive models, food delivery services can improve their planning, allocate delivery staff more effectively, and provide customers with precise estimated delivery times. This may enhance customer confidence and brand loyalty (Chen et al., 2021).

In the future, real-time information like current traffic updates and weather conditions may be incorporated into the models to enhance the accuracy of predictions. Methods in machine learning, such as deep learning, may be examined to enhance performance. When companies effectively utilize these models, they can achieve a significant competitive edge by providing quicker and more dependable deliveries. This may result in improved business expansion, increased customer contentment, and more streamlined processes in the food delivery sector.

References

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1986). Classification and regression trees. CRC Press.

Chen, Y., Wang, J., & Zhang, X. (2021). Machine learning approaches for predicting food delivery times: A case study. Journal of Artificial Intelligence and Applications, 15(2), 87-103.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). Pattern classification (2nd ed.). John Wiley & Sons.

Ghosh, A., Banerjee, S., & Gupta, R. (2022). Predictive analytics in food delivery logistics: A data-driven approach. International Journal of Logistics Research and Applications, 25(3), 289-310.

Han, J., Kamber, M., & Pei, J. (2012). Data mining: Concepts and techniques (3rd ed.). Elsevier.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: Data mining, inference, and prediction (2nd ed.). Springer.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (1st ed.). Springer.

Kuhn, M., & Johnson, K. (2013). Applied predictive modeling. Springer.

Oliveira, W. (2023). Food Delivery Time [Data set]. Kaggle. Retrieved from https://www.kaggle.com/datasets/willianoliveiragibin/food-delivery-time.

Sharma, A., Soni, S., & Gupta, V. (2020). Predictive analytics in customer satisfaction and operational improvement in delivery systems. International Journal of Data Science and Analytics, 9(4), 195-208.

Sharma, P., Verma, S., & Rao, K. (2020). Enhancing food delivery efficiency using predictive modeling techniques. Journal of Business Analytics, 7(1), 45-62.