CS M148: Predicting DoorDash Delivery Time

DOORDASH

Group 12 - Abhinav Amanaganti (1763), Harris Song (8015), Ethan Maldonado (4340), Afnan Khawaja (2648)

UCLA

Introduction (Ethan)

Objective: Our project aims to enhance the accuracy **of Estimated Time of Arrival (ETA)** provided by the Doordash app for its deliveries

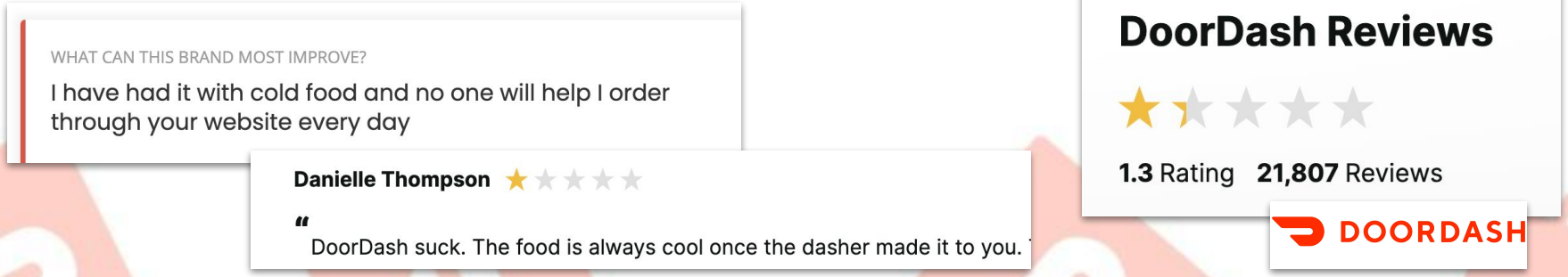
Importance and Relevance: Accurate ETAs significantly improve **user satisfaction** by setting realistic expectations for delivery times. This can lead to higher customer retention and better reviews for Doordash.

Motivation: Accurate delivery times are crucial for both customer satisfaction and operational efficiency. Timely deliveries ensure **higher customer satisfaction and repeat business**

Problem Statement (Ethan)

Problem: The current ETA predictions provided by Doordash may not always be reliable, leading to customer dissatisfaction.

Goal: Develop a predictive model to estimate delivery times more accurately using various features such as the number of on-shift dashers, total items in the order, store category, and market ID.



[1] https://www.reviews.io/company-reviews/store/doordash

Data Source: Kaggle: Doordash ETA Prediction (Ethan)

Time Features

- ❏ market_id
- ❏ **created_at (graph shown)**
- ❏ actual_delivery_time

Store Features

- ❏ store_id
- ❏ store_primary_category
- ❏ order_protocol

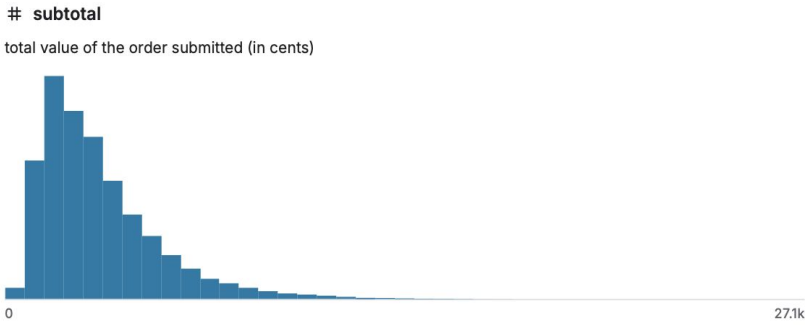
Order Features

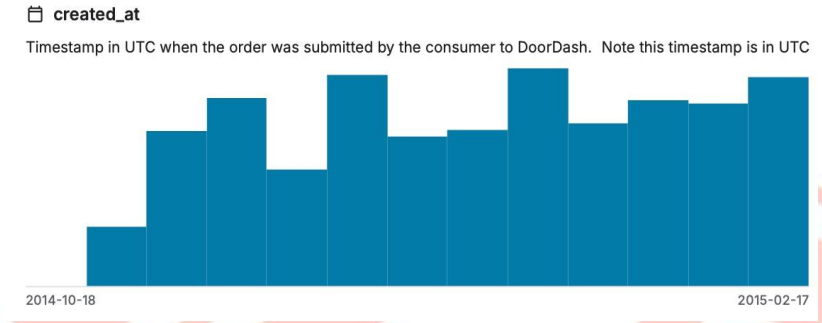
- ❏ total_items
- ❏ **subtotal (graph shown)**
- ❏ num_distinct_items
- ❏ min_item_price
- ❏ max_item_price

Market Features

- ❏ total_onshift_dashers
- ❏ total_busy_dashers
- ❏ total_outstanding_orders

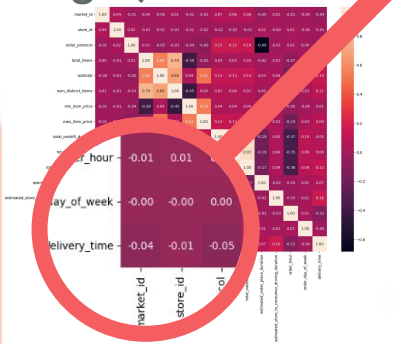
Observation: Note how some data (like subtotal) follow a right-skewed distribution.



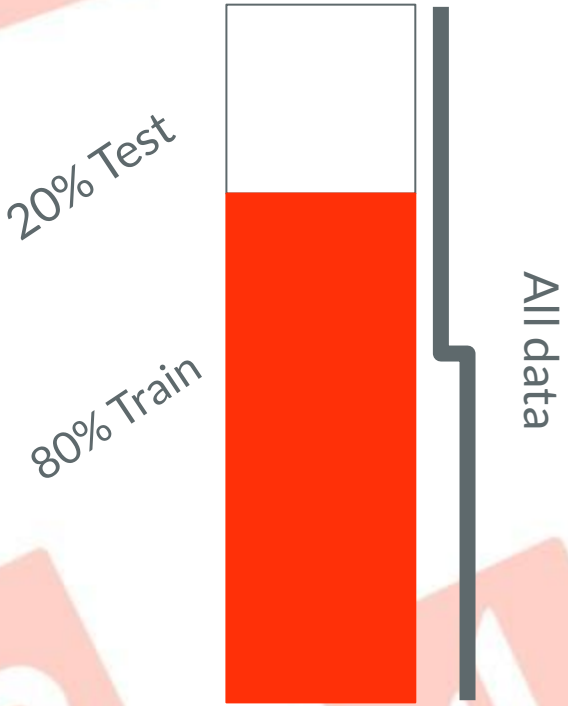


Data Collection & Preprocessing (Afnan)

- **Delivery Time Calculation:** "actual_delivery_time" - "created_at"
- **Handling Missing Data:** Fill NA columns to retain data
- **Data Splitting:** 80% training, 20% testing
- **Heatmap Analysis:** Sklearn for variable correlation **(shown right)**
- **Next Steps:** Optimize for computational efficiency



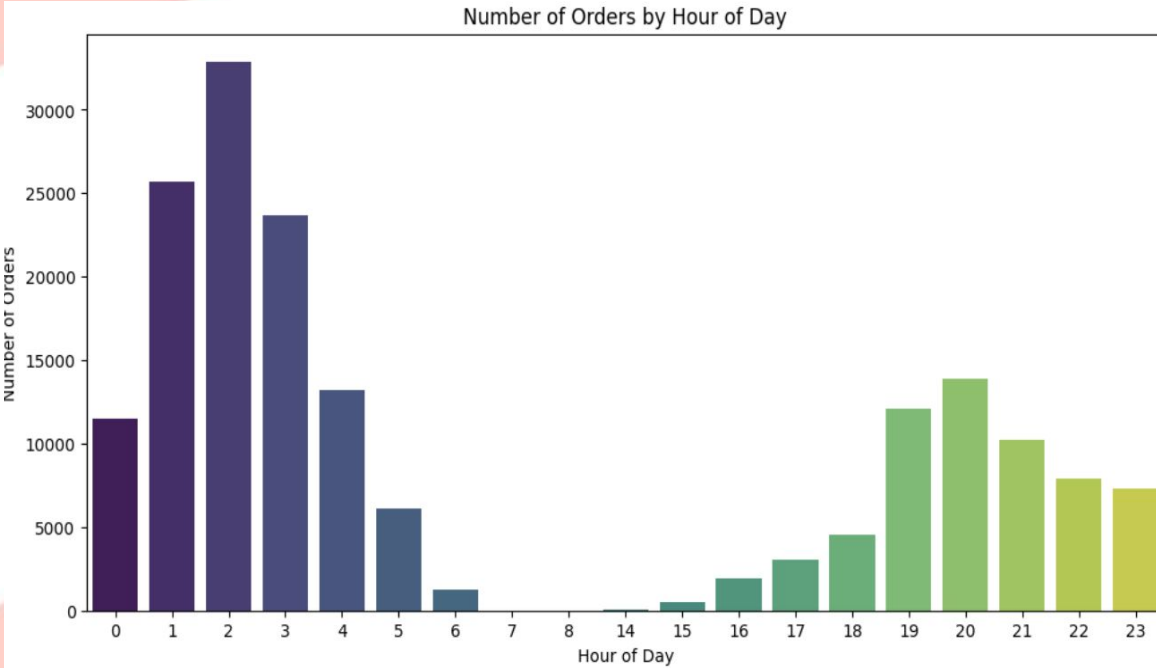
Data Validation (Afnan)

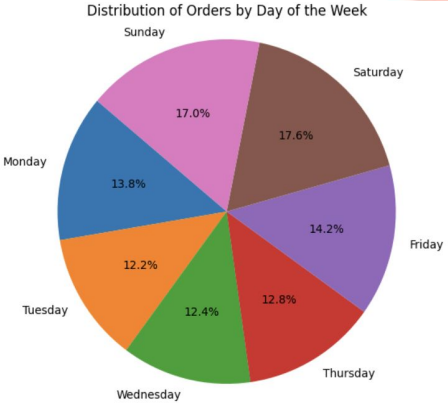


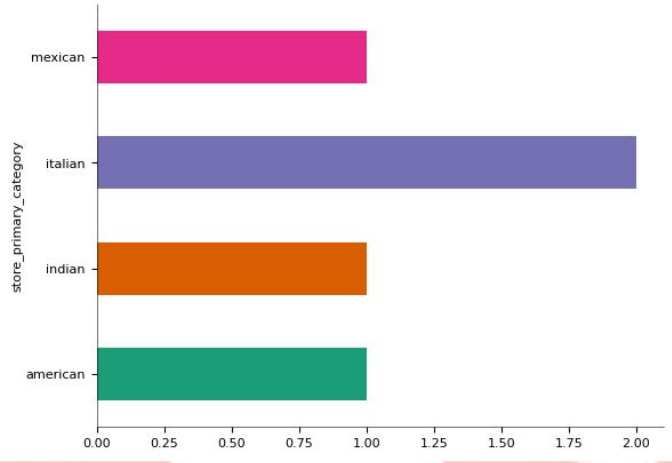
Matches percentage from Homeworks

```python
# Fill missing values for simplicity
data = data.fillna(method='ffill')


# Convert 'created_at' and 'actual_delivery_time' to datetime
data['created_at'] = pd.to_datetime(data['created_at'])
data['actual_delivery_time'] = pd.to_datetime(data['actual_delivery_time'])

# Feature Engineering: Calculate delivery duration
data['delivery_duration'] = (data['actual_delivery_time'] - data['created_at']).dt.total_seconds()

# Selecting a subset of the data for efficiency
data_subset = data.sample(frac=0.1, random_state=42)

# Define features and target variable
features = ['market_id', 'store_id', 'order_protocol', 'total_items', 'subtotal',
            'num_distinct_items', 'min_item_price', 'max_item_price',
            'total_onshift_dashers', 'total_busy_dashers', 'total_outstanding_orders',
            'estimated_order_place_duration', 'estimated_store_to_consumer_driving_duration']
target = 'delivery_duration'

X = data_subset[features]
y = data_subset[target]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Preprocessing Complete. Data is ready for model training.")
```

Some Relevant Figures (Afnan)





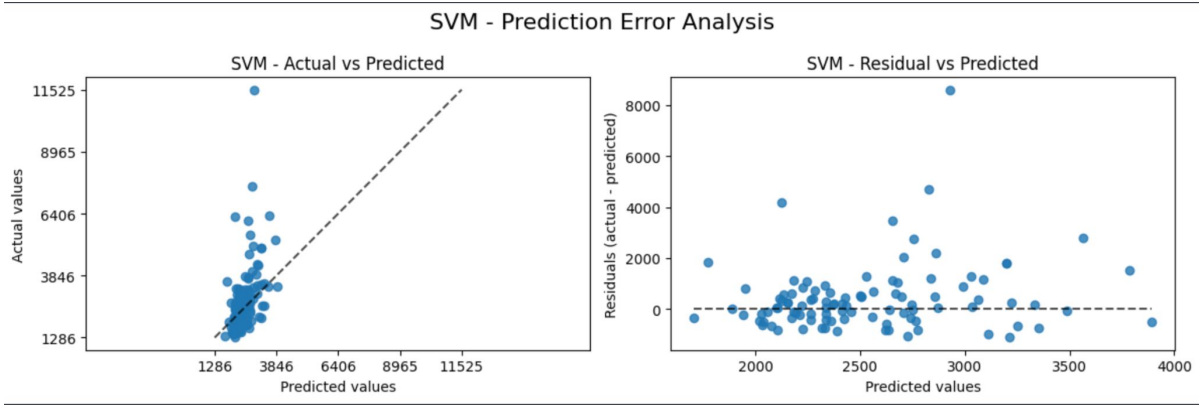


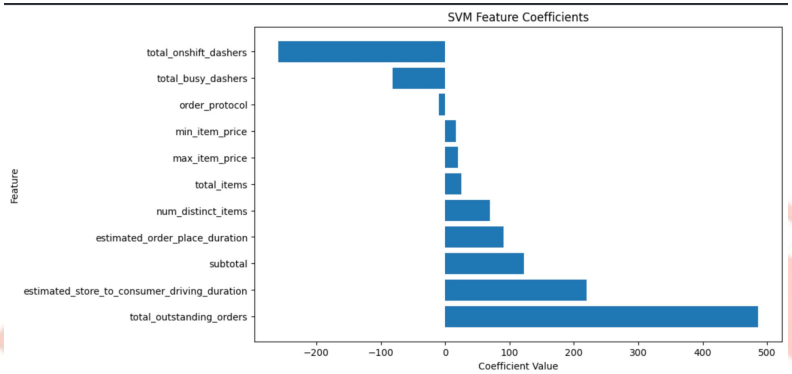
To sanity-check the data-set and for some observations

All Algorithms used (Abhinav)

Chosen Algorithms:

- **Decision Tree**
- **Random Forest**
- **Linear Regression**
- KNN
- SVM





Pictured: Graphs of SVM, with the Feature Coefficients Actual / Predicted, and Residuals. Residuals is particularly useful to analyze any bias (there is none in this case).

Decision Trees (Abhinav)

Used due to its easy interpretation, though it tends to overfit

MAE: 841.2 *seconds*
MSE: 1151088.3 *seconds*^{2}

```
from sklearn.tree import DecisionTreeRegressor
tree_model = DecisionTreeRegressor()
tree_model.fit(X_train, y_train)
y_pred_tree = tree_model.predict(X_test)
```

Decision Tree Analysis (Abhinav)

Methodology

Scatter Plot:

- Points are fairly clustered around diagonal line; reasonable alignment of predictions with actual values
- However, noticeable spread of points especially as actual delivery time increases = worse predictions for longer delivery times

Prediction Error:

- Underestimation for high ground truth values
- Overestimation for lower ground truth values

Pros:

- **Captures non-linear relationship** between our features and delivery **time target variable**
- No need for Feature Scaling

Cons:

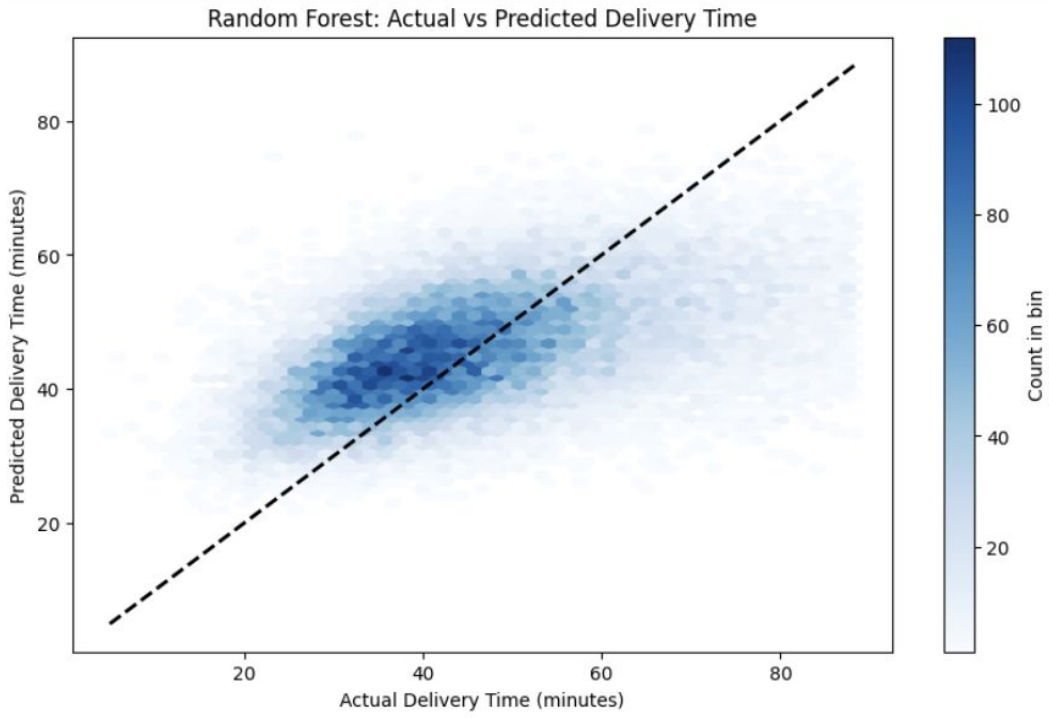
- High variance = unstable model
- Bad performance for continuous data, which we are dealing with

Random Forest (Abhinav)

Multiple Decision Trees, though more computationally expensive

MAE: 600.3 *seconds*
MSE: 580,845.6 *seconds*^{2}

```python
from sklearn.ensemble import RandomForestRegressor

# Initialize the model
rf_model = RandomForestRegressor(n_estimators=100,
random_state=42)

# Fit the model
rf_model.fit(X_train, y_train)

# Get predictions
rf_predictions = rf_model.predict(X_test)

# Calculate regression metrics
rf_mae = mean_absolute_error(y_test, rf_predictions)
rf_mse = mean_squared_error(y_test, rf_predictions)
```



Random Forest Analysis (Abhinav)

Methodology

Scatter Plot: Points are more tightly clustered around diagonal line compared to decision tree model->indicates more accurate predictions
Spread: Random Forest Model showcases less spread + fewer extreme outliers also demonstrating better predictive performance

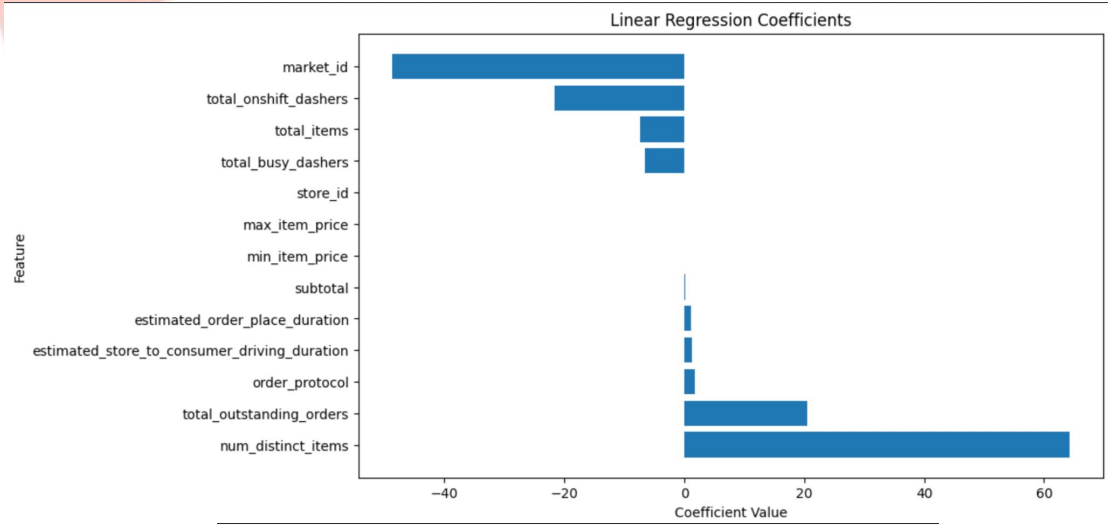
Pros:

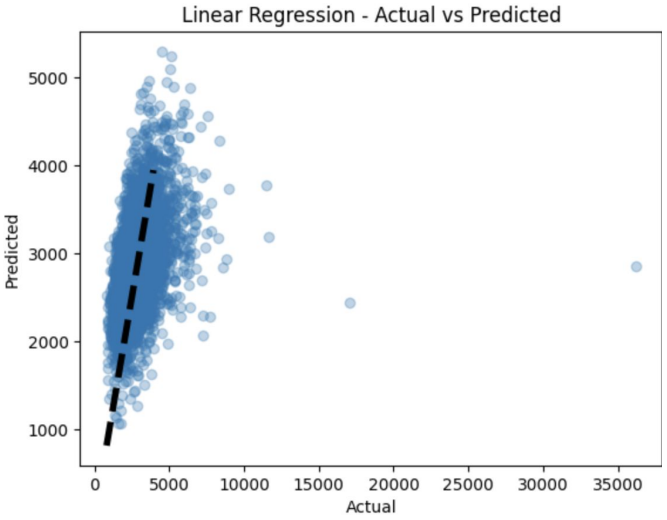
- Robust + stable
  - Random forest reduces overfitting by averaging predictions of multiple decision trees.
- Reduces overfitting by utilizing bootstrap samples + random subsets of features

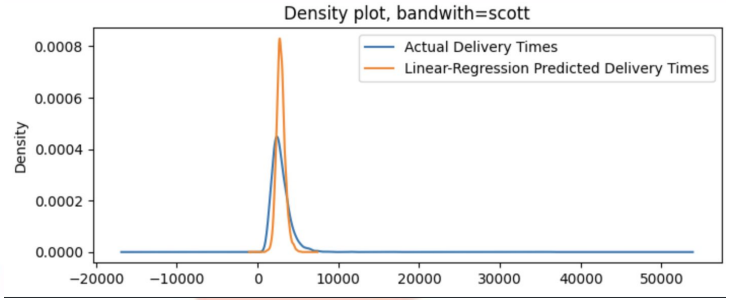
Cons:

- Model is more complex and less interpretable, especially for higher-level datasets like this one
- Requires tuning of many hyperparameters for optimal performance, which is hard on a local setup

Linear Regression (Harris)







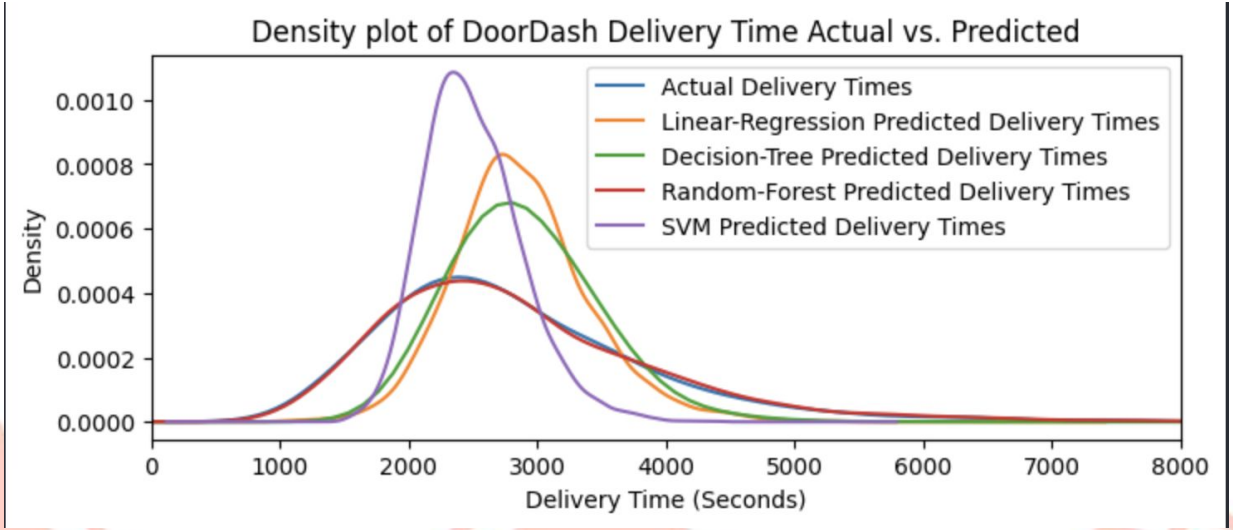
MAE: 716.9 *seconds*
MSE: 1,236,274.4 *seconds*^{2}

Conclusion & Individual Case Study (Harris)

Mathematical Approach:

Algorithm	MAE *(seconds)*
Random Forest	601.8
K-NN	700.8
Linear Regression	716.9
SVM	772.3
Decision Tree	841.2

Visual Inspection:



Conclusion & Individual Case Study (Harris)

Example using Random Forest:

created_at	actual_delivery_time	store_primary_category	total_items	subtotal	num_distinct_items	min_item_price
2015-02-06 22:24:17	2015-02-06 23:27:16	american	4.0	3441.0	4.0	557.0

max_item_price	total_onshift_dashers	total_busy_dashers	total_outstanding_orders	estimated_order_place_duration	estimated_store_to_consumer_driving_duration	delivery_time
1239.0	33.0	14.0	21.0	446.0	861.0	62.983333

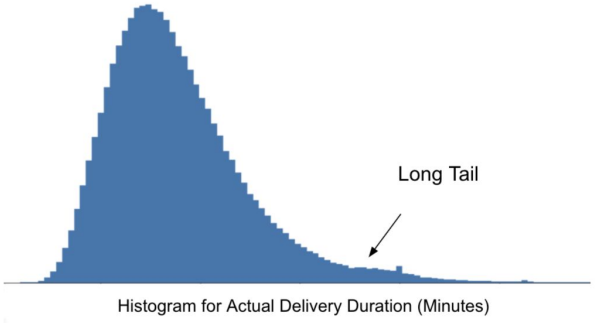
Random Forest Prediction:

62.09466667 minutes!

```
print(rf_model.predict(x_individual))
```

Takeaways (Harris)

- Relatively Accurate Predictions
- Long tail, coined by Doordash, is a phenomenon with no clear outlier but ~10% data skewed[1]

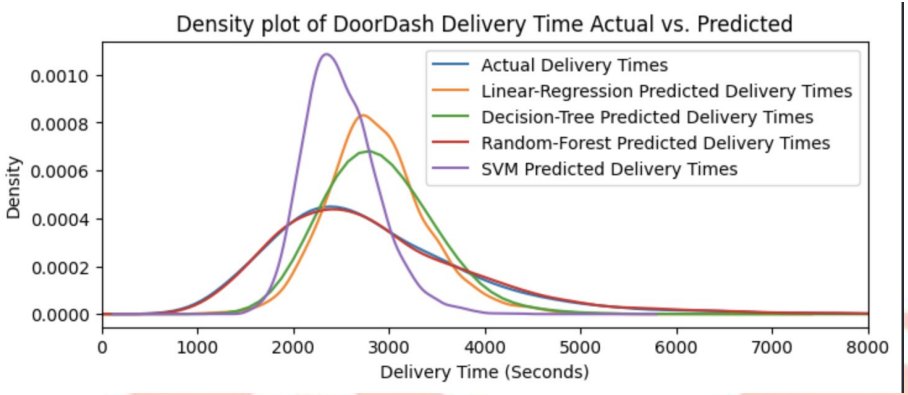


Histogram for Actual Delivery Duration (Minutes)

Asymmetric MSE loss function:

$$\frac{1}{n}\sum_{i=1}^{n}|\alpha - \mathbb{1}_{(g(x_i)-\hat{g}(x_i)<0)}|(g(x_i)-\hat{g}(x_i))^2$$

with $\alpha \in (0,1)$ being the parameter we can adjust to change the degree of asymmetry



Our data closely matches the analysis @Doordash!

(Top Right: From Doordash Engineering[1], Bottom Right: Our Analysis)

[1] https://doordash.engineering/2021/04/28/improving-eta-prediction-accuracy-for-long-tail-events/