Austin Micromobility Study
June 2019

Jake Grosek

Table of Contents

Introduction

Austin's terrible congestion stats:

- ~2.16 million people
- Ranked worst traffic in Texas
- # 19 worst in the USA

"The Last Mile" problem: the least efficient part, comprising up to 28% of the total cost to move goods.

Sources:
https://www.statesman.com/news/20190611/austin-has-some-of-worst-traffic-congestion-in-world-study-finds

https://medium.com/the-stigo-blog/the-last-mile-the-term-the-problem-and-the-odd-solutions-28b6969d5af8

Micromobility History

2014: Austin launches “BCycle,” a bike rental service with ~100 kiosks spread throughout the city

2018: Austin’s first dockless scooter & bike ride: April 3, 2018

Visualizations: Dockless Companies

~10% of all Dockless rides travel ZERO meters

In [20]:
```
normal_dockless_rides = dockless.loc[(dockless['trip_distance'] > 1) & (dockless['trip_distance'] <= 16000)]
len(normal_dockless_rides)
```
Out[20]: 4962659

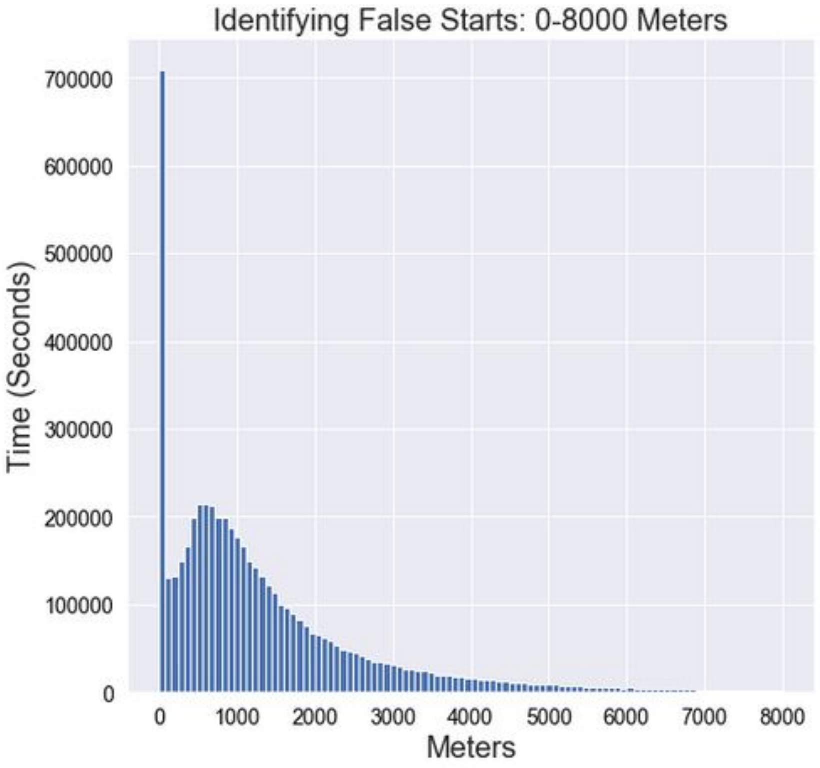
~91% of the rides are considered within this range

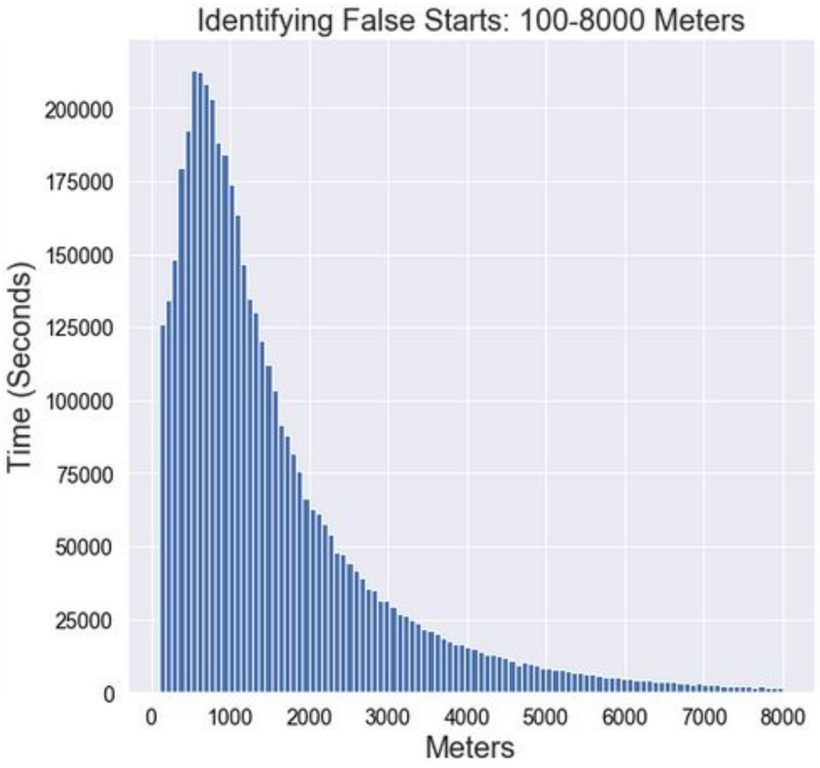
In [158]:
```
rr = len(normal_dockless_rides) / len(dockless)
rr
```
Out[158]: 0.9120383924159063

In [159]:
```
false_rides = dockless.loc[(dockless['trip_distance'] < 1)]
len(false_rides)
```
Out[159]: 464633

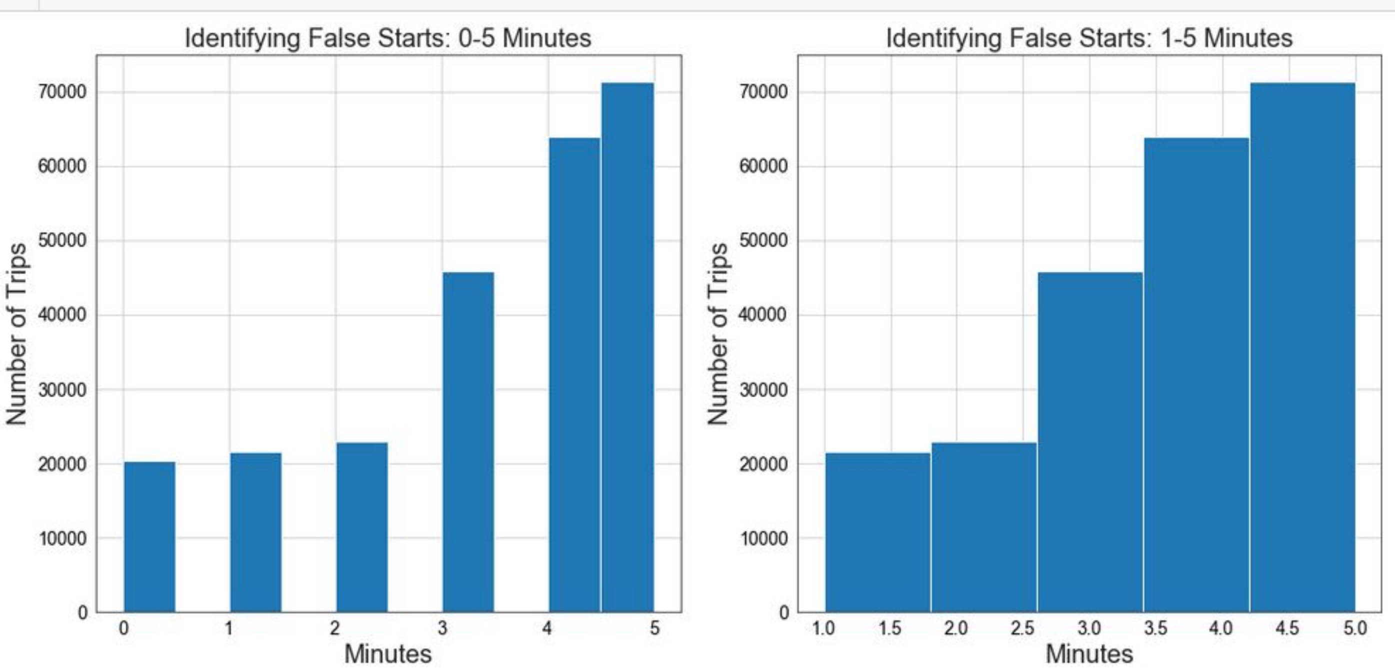
In [160]:
```
fr = len(false_rides) / len(dockless)
fr
```
Out[160]: 0.08539033900644388

“False Starts” part 1 - Dockless





“False Starts” part 2: Austin BCycle



Hypothesis & Features Engineering

Hypothesis

Hypothesis:

The “wear and tear” on individual scooters has a relationship with how likely a future ride will register as a “false start.”

Null Hypothesis:

Wear and tear does not have a relationship with “false starts.”

Feature Engineering - Average Speed

1. Creating Average Speed Feature

```
# speed = distance / time. Creating a new column for avg speed.
dockless['avg_speed'] = round((dockless['trip_distance'] / dockless['trip_duration_seconds']), 2)
```

```
#Calling the varaible again to no include the new 'avg_speed' column
normal_dockless_rides = dockless.loc[(dockless['trip_distance'] >= 1)
                                    & (dockless['trip_distance'] <= 16000)
                                    & (dockless['trip_duration_seconds'] >= 1)]
```

```

#Calling the varaible again to no include the new 'avg_speed' column
normal_dockless_rides_s = dockless.loc[(dockless['trip_distance'] >= 1)
                                    & (dockless['trip_distance'] <= 16000)
                                    & (dockless['vehicle_type'] == 'scooter')
                                    & (dockless['trip_duration_seconds'] >= 1)]

normal_dockless_rides_b = dockless.loc[(dockless['trip_distance'] >= 1)
                                    & (dockless['trip_distance'] <= 16000)
                                    & (dockless['vehicle_type'] == 'bicycle')
                                    & (dockless['trip_duration_seconds'] >= 1)]
```

Feature Engineering - Average Speed

Since we don't have a column with distance information in BCycle, we will use the average speed we found in Dockless to understand the distance these bikes may have taken.

Applying Trip_Distance using our new bike speed

```
# distance = speed x time; rounding up to conserve space
bcycle['trip_distance'] = round((avg_bike_speed * bcycle.trip_duration_seconds), 2)
```

```
bcycle.trip_distance.head(3)
```

```
0    15323.63
1    11694.35
2     1613.01
Name: trip_distance, dtype: float64
```

Feature Engineering - Trip Counter

2. The number of trips per unique device

```python
# new variable to measure length of dataset
device_id_set_length_dock = len(set(dockless['device_id']))

# Dictionary to store cumulative counts
device_id_dict_dock = dict(zip(set(dockless['device_id']), np.zeros(device_id_set_length_dock)))

# Empty list to store running count
running_count_dock = []

# Loop through all values
for row in dockless.itertuples():
  device_id_dict_dock[row[2]] += 1
  running_count_dock.append(device_id_dict

dockless['device_id_trip_count'] = running
```

Now we will do the same in the BCycle data set

```python
# new variable to measure length of dataset
device_id_set_length_bcycle = len(set(bcycle['bicycle_id']))

# Dictionary to store cumulative counts
device_id_dict_bcycle = dict(zip(set(bcycle['bicycle_id']), np.zeros(device_id_set_length_bcycle)))

# Empty list to store running count
running_count_bcycle = []

# Loop through all values; the row has to be changed to 3 to count the bicycle_id column
for row in bcycle.itertuples():
  device_id_dict_bcycle[row[3]] += 1
  running_count_bcycle.append(device_id_dict_bcycle[row[3]])

bcycle['device_trip_count'] = running_count_bcycle
```

Feature Engineering - Odometer

Making the Odometer on a large scale

```
dockless['odometer'] = dockless.groupby('device_id')['trip_distance'].transform(pd.Series.cumsum)
```

Now I'll do the same for the Bcycle dataset

```
# odometer
bcycle['odometer'] = bcycle.groupby('bicycle_id')['trip_distance'].transform(pd.Series.cumsum)
```

Feature Engineering - Converting to Unix time

```
from datetime import datetime
# set the parameter for how the date is being interpretted
merge_time_dock = datetime.strptime('04/29/2019 05:30:00 PM', '%m/%d/%Y %H:%M:%S %p')
type(merge_time_dock)
# how i want the time coverted to (seconds)
merge_time_dock.strftime('%s')
# define unix timestamp function
def dockless_to_timestamp(str):
    merge_time_dock = datetime.strptime(str, '%m/%d/%Y %H:%M:%S %p')
    return merge_time_dock.strftime('%s')
```

```
# apply timestamp to new unix start time
dockless['unix_start_time'] = dockless.unix_start_time.apply(dockless_to_timestamp)
```

```
# apply timestamp to new unix end time
dockless['unix_end_time'] = dockless.unix_end_time.apply(dockless_to_timestamp)
```

```
# convert to integer
dockless['unix_start_time'] = pd.to_numeric(dockless['unix_start_time'], errors='ignore', downcast='integer')
```

```
# convert to integer
dockless['unix_end_time'] = pd.to_numeric(dockless['unix_end_time'], errors='ignore', downcast='integer')
```

Feature Selection & Comparison

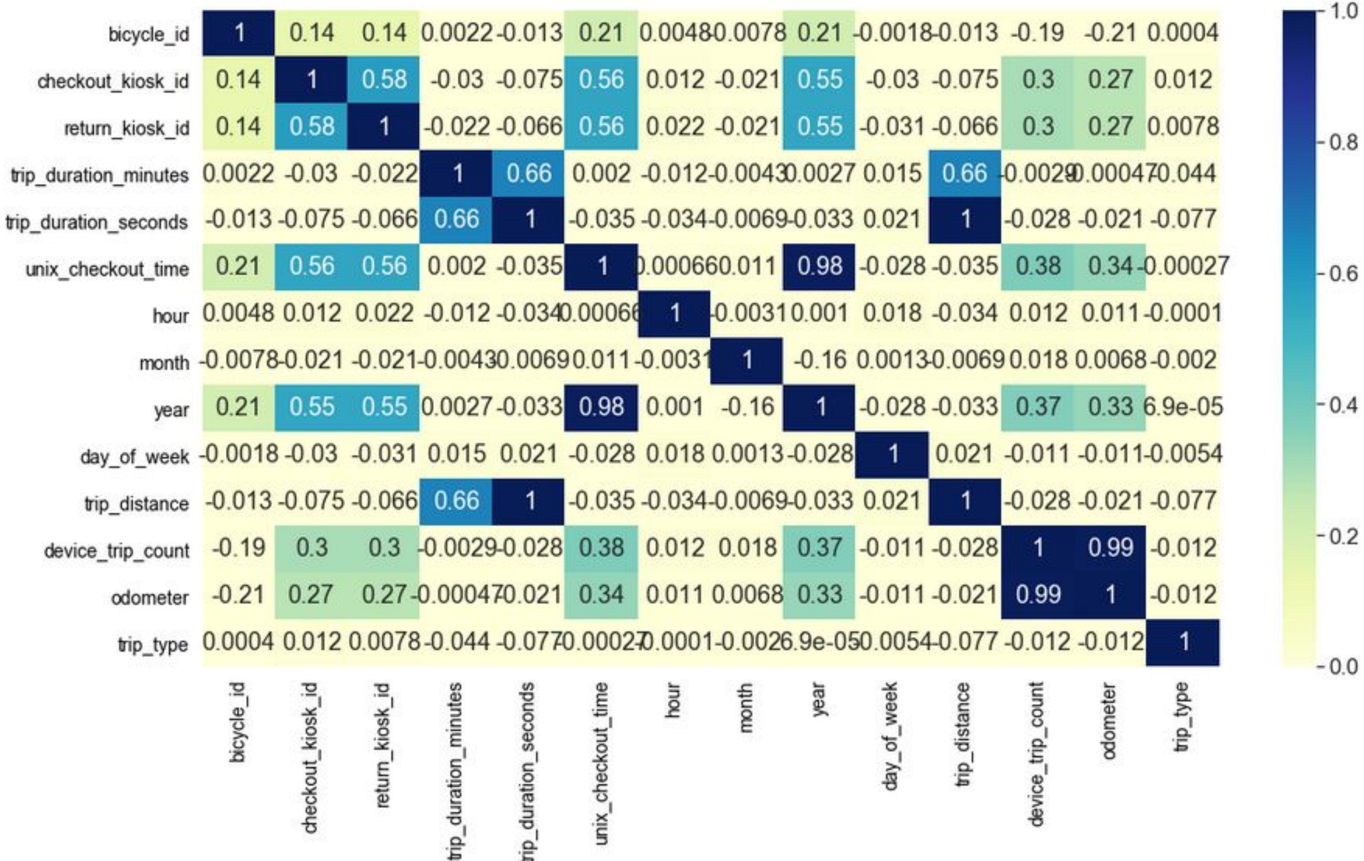
Feature Selection - BCycle

```python
selector_b = SelectKBest(k=25)
X_bnew = selector.fit_transform(X_btrain, y_btrain)
names_b = X_b.columns.values[selector.get_support()]
scores_b = selector.scores_[selector.get_support()]
names_scores_b = list(zip(names, scores))
ns_bcycle = pd.DataFrame(data = names_scores_b, columns=['Feat_names', 'F_Scores'])
ns_bcycle_sorted = ns_bcycle.sort_values(['F_Scores', 'Feat_names'], ascending = [False, True])
print(ns_bcycle_sorted)
```

	Feat_names	F_Scores
23	membership_type_U.T. Student Membership	36698.461331
21	membership_type_HT Ram Membership	7175.257922
7	year_2018	6217.842646
0	unix_checkout_time	5691.097252
16	checkout_kiosk_id_4055	5420.752547
17	checkout_kiosk_id_4059	3240.373279
15	checkout_kiosk_id_3798	2873.704791
18	checkout_kiosk_id_4061	2409.092922
19	checkout_kiosk_id_4062	2190.196286
24	membership_type_Walk Up	1840.175314
1	device_trip_count	1272.937506
4	year_2015	1170.734433
3	year_2014	1135.168718
12	checkout_kiosk_id_3377	1126.108770
22	membership_type_Local365	1088.065856
2	odometer	1014.794557
11	checkout_kiosk_id_2575	984.664865
6	year_2017	941.255446
5	year_2016	891.637187
14	checkout_kiosk_id_3794	884.276755
10	checkout_kiosk_id_2574	808.880423
8	checkout_kiosk_id_2500	779.199069
9	checkout_kiosk_id_2561	778.868370
20	membership_type_24-Hour Kiosk (Austin B-cycle)	749.945536
13	checkout_kiosk_id_3790	726.857824

- Get_dummies on the membership types to rank importance.
- Certain memberships and kiosk had the best scores.

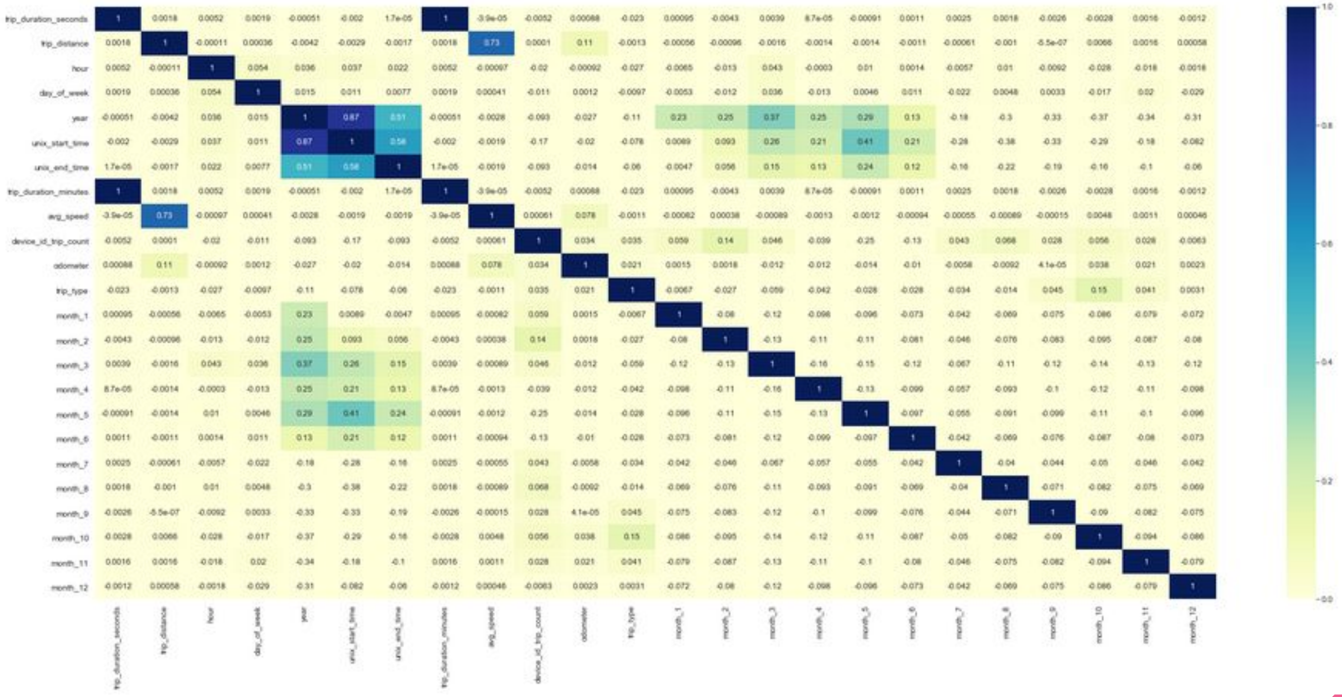
Feature Selection - BCycle



Relationships of note:

- kiosk locations
- Checkout time
- Year
- Device trip counter
- odometer

Feature Selection - Dockless (Month)



Relationships of note:

- Year
- Start time
- End time
- Months

Model Selection & Comparison BCycle: Regression & Multinomial Classification

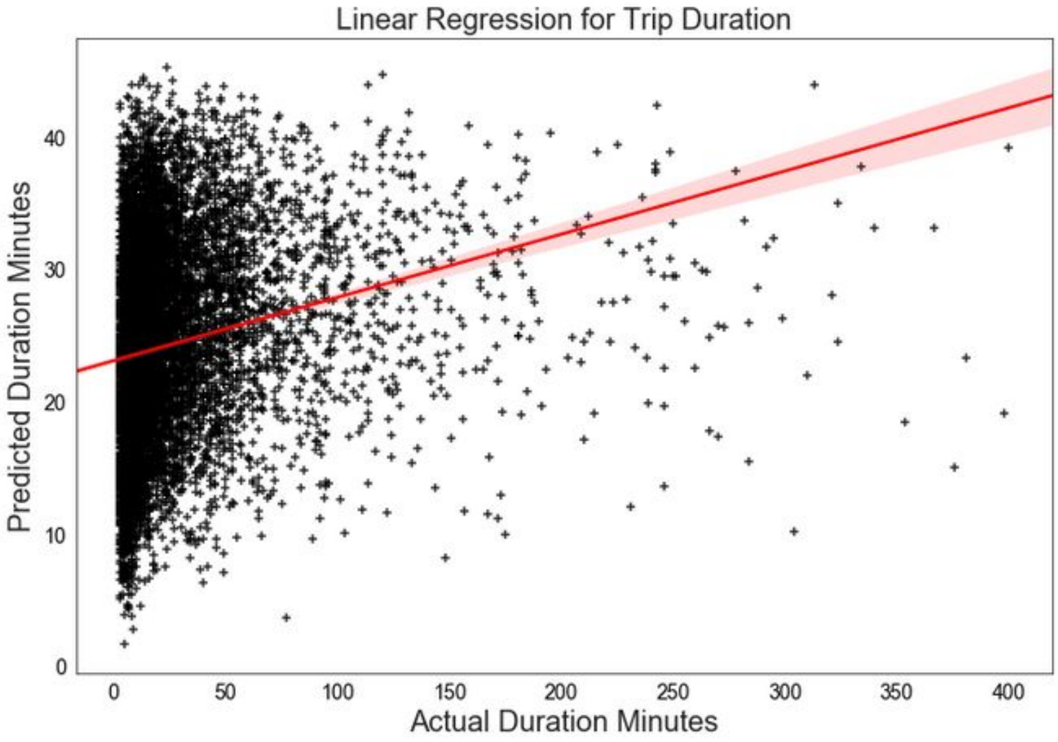
Regression

```
bcycle_10k = normal_bcycle_rides.sample(10000, random_state=10)
# Original
# drop the extra columns
Xreg = bcycle_10k[['unix_checkout_time', 'device_trip_count',
               'odometer', 'month', 'day_of_week', 'hour',
                 'year', 'trip_type']]
#X = bcycle.drop(['trip_id', 'membership_type', 'b' 'trip_type'],  1)
# create a value for the various membership types so it can pass and get insight
Xreg = pd.get_dummies(Xreg, columns=['month', 'day_of_week',
                                     'year', 'hour', 'trip_type'])
# get dummies left out: 'checkout_kiosk_id', 'return_kiosk_id'
# predicting for trip type
yreg = bcycle_10k['trip_duration_minutes']

# test train split function
Xreg_train, Xreg_test, yreg_train, yreg_test = train_test_split(Xreg, yreg, test_size=0.25)
```

Based on the heatmaps, the best features for Bcycle were checkout time, trip counter, odometer, month, day of the week, hour, year, and whether or not it was a “false start.”

```
ax = sns.regplot(yreg, y_predict, data=bcycle_10k, x_estimator=None, x_bins=None, x_ci='ci',
            scatter=True, fit_reg=True, ci=95, n_boot=1000, units=None, order=1, logistic=False,
            lowess=False, robust=False, logx=False, x_partial=None, y_partial=None, truncate=False,
            dropna=True, x_jitter=None, y_jitter=None, label=None, color=None, marker='+',
            scatter_kws={"color": "black"}, line_kws={"color": "red"}, ax=None)
ax.set_title('Linear Regression for Trip Duration', fontsize=20)
ax.set_ylabel('Predicted Duration Minutes', fontsize=20 )
ax.set_xlabel('Actual Duration Minutes', fontsize=20 )
ax.figure.set_size_inches(12,8)
ax.tick_params(labelsize=14,labelcolor="black")
```

Linear Regression

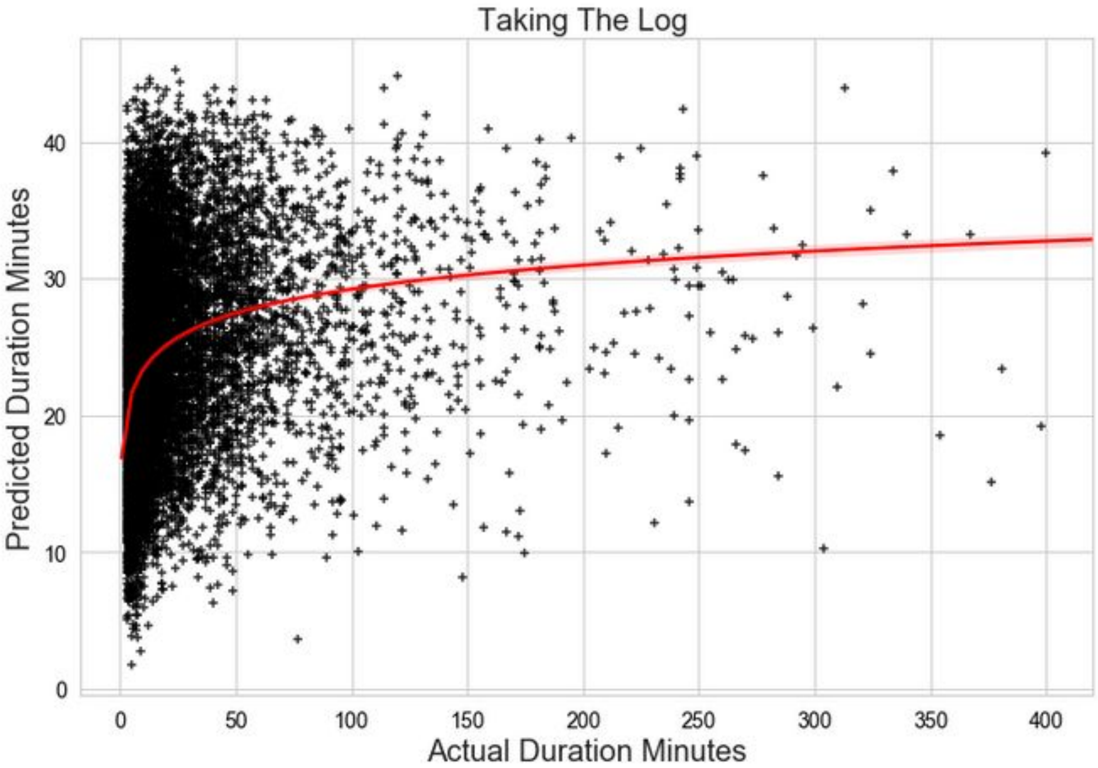


Multivariable:

- Checkout time
- Device trip count
- Odometer
- Month
- Day of the week
- Hour
- Year

Predicting for: Trip Duration Minutes

Logarithmic Regression



Multivariable:

- Checkout time
- Device trip count
- Odometer
- Month
- Day of the week
- Hour
- Year

Predicting for: Trip Duration Minutes

How well could I predict the Return Kiosk?

Multinomial Classification

There are 96 kiosks available to predict where they will be returned

```
kiosk_length_r = pd.value_counts(bcycle['return_kiosk_id'].values, sort=False)
kiosk_length_c = pd.value_counts(bcycle['checkout_kiosk_id'].values, sort=False)
print(len(kiosk_length_r))
print(len(kiosk_length_c))
```

```
96
96
```

```
# Original
# drop the extra columns
X96 = bcycle[['checkout_kiosk_id', 'odometer', 'unix_checkout_time', 'device_trip_count', 'year']]
#X = bcycle.drop(['trip_id', 'membership_type', 'b' 'trip_type'], 1)
# create a value for the various membership types so it can pass and get insight
X96 = pd.get_dummies(X96, columns=['checkout_kiosk_id', 'year'])
# predicting for trip type
y96 = bcycle['return_kiosk_id']

# test train split function
X96_train, X96_test, y96_train, y96_test = train_test_split(X96, y96, test_size=0.25)
```

Predicting the Return Kiosk out of 96 Options

```python
# Fit regression model (using the natural log of one of the regressors)
results = smf.ols('return_kiosk_id ~ checkout_kiosk_id + unix_checkout_time + device_trip_count + odometer',
                  data=bcycle).fit()

# Inspect the results
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:        return_kiosk_id   R-squared:                       0.423
Model:                            OLS   Adj. R-squared:                  0.423
Method:                 Least Squares   F-statistic:                 2.057e+05
Date:                Thu, 08 Aug 2019   Prob (F-statistic):               0.00
Time:                        19:00:44   Log-Likelihood:            -8.4152e+06
No. Observations:             1122091   AIC:                         1.683e+07
Df Residuals:                 1122086   BIC:                         1.683e+07
Df Model:                           4
Covariance Type:            nonrobust
======================================================================================
                         coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------
Intercept          -3820.7737     15.502   -246.473      0.000   -3851.157   -3790.391
checkout_kiosk_id      0.3824      0.001    438.260      0.000       0.381       0.384
unix_checkout_time  3.731e-06   1.13e-08    331.479      0.000    3.71e-06    3.75e-06
device_trip_count      0.2964      0.004     67.924      0.000       0.288       0.305
odometer           -4.666e-05    8.2e-07    -56.898      0.000   -4.83e-05   -4.51e-05
==============================================================================
Omnibus:                    69676.224   Durbin-Watson:                   1.492
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           118677.961
Skew:                          -0.488   Prob(JB):                         0.00
Kurtosis:                       4.259   Cond. No.                     5.58e+10
==============================================================================
```

Again, I kept the best performing features:

- Checkout kiosk
- Return kiosk
- Checkout time
- Device trip count
- Odometer

R-squared of 0.423

Predicting the Return Kiosk out of 96 Options

```python
# Fit regression model (using the natural log of one of the regressors)
results = smf.ols('return_kiosk_id ~ checkout_kiosk_id + unix_checkout_time + device_trip_count + odometer',
                  data=bcycle).fit()

# Inspect the results
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:         return_kiosk_id   R-squared:                       0.423
Model:                             OLS   Adj. R-squared:                  0.423
Method:                  Least Squares   F-statistic:                 2.057e+05
Date:                 Thu, 08 Aug 2019   Prob (F-statistic):               0.00
Time:                         19:00:44   Log-Likelihood:            -8.4152e+06
No. Observations:              1122091   AIC:                         1.683e+07
Df Residuals:                  1122086   BIC:                         1.683e+07
Df Model:                            4
Covariance Type:             nonrobust
======================================================================================
                         coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------
Intercept          -3820.7737     15.502   -246.473      0.000   -3851.157   -3790.391
checkout_kiosk_id      0.3824      0.001    438.260      0.000       0.381       0.384
unix_checkout_time  3.731e-06   1.13e-08    331.479      0.000    3.71e-06    3.75e-06
device_trip_count      0.2964      0.004     67.924      0.000       0.288       0.305
odometer           -4.666e-05    8.2e-07    -56.898      0.000   -4.83e-05   -4.51e-05
==============================================================================
Omnibus:                    69676.224   Durbin-Watson:                   1.492
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           118677.961
Skew:                          -0.488   Prob(JB):                         0.00
Kurtosis:                       4.259   Cond. No.                     5.58e+10
==============================================================================
```

Again, I kept the best performing features:

- Checkout kiosk
- Return kiosk
- Checkout time
- Device trip count
- Odometer

R-squared of 0.423

Naive Bayes & Random Forest Classification

```
# Display our results.
print("Number of mislabeled points out of a total {} points : {}".format(
    X96.shape,
    (y96_test != y_pred_gnb).sum()
))

print('\nR-squared:')
print(gnb.score(X96_test, y96_test))
```

```
Number of mislabeled points out of a total (1122091, 106) points : 254914

R-squared:
0.0912901972387
```

```
# Display our results.
print("Number of mislabeled points out of a total {} points : {}".format(
    X96.shape[0],
    (y96_test != y_pred_rfc).sum()
))

print('\nR-squared:')
print(rfc.score(X96_test, y96_test))
```

```
Number of mislabeled points out of a total 1122091 points : 238486

R-squared:
0.14985224028
```

- Naive Bayes accurately predicted 9% of the time
- RFC accurately predicted 15% !!
- X96 (the X_train) was over 800,000 rows of data.

Model Selection & Comparison Dockless: Predicting False Starts Using Binary Classifiers

Downsample the majority class

```
# creating dataframe of rides from 0 to 16000 meters
false_start_rides = dockless.loc[(dockless['trip_distance'] <= 16000)]
len(normal_dockless_rides)
```

```
4962659
```

```
# Separate majority and minority classes
df_majority = false_start_rides[false_start_rides.trip_type==0]
df_minority = false_start_rides[false_start_rides.trip_type==1]
```

```
from sklearn.utils import resample
# Downsample majority class
df_majority_downsampled = resample(df_majority,
                                   replace=False,    # sample without replacement
                                   n_samples=464633,     # to match minority class
                                   random_state=123) # reproducible results
# Combine minority class with downsampled majority class
df_downsampled = pd.concat([df_majority_downsampled, df_minority])

# Display new class counts
df_downsampled.trip_type.value_counts()
```

```
1    464633
0    464633
```

Model Evaluation

Logistic Regression: 50%

Naive Bayes: 49%

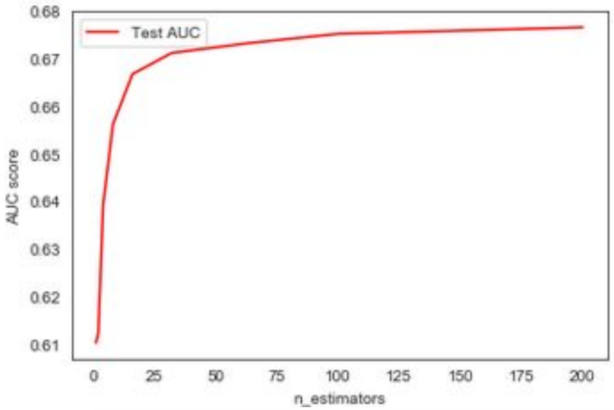
Random Forest Classifier: 67%

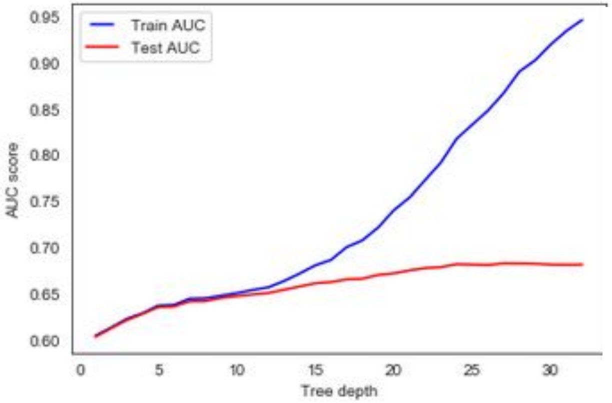
XGBoost: 66%

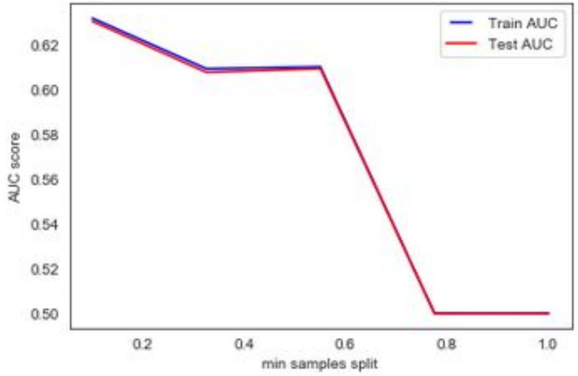
```
# RFC confusion matrix
results_rfc_fs = confusion_matrix(ydown_test, y_pred_rfc_fs)
print('\nConfusion Matrix :')
print(results_rfc_fs)
print('Accuracy Score :',accuracy_score(ydown_test, y_pred_rfc_fs))
print('\nReport : ')
print(classification_report(ydown_test, y_pred_rfc_fs))
```

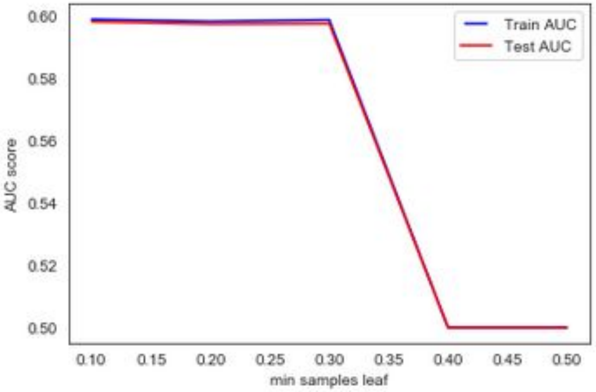
```
Confusion Matrix :
[[79407 36334]
 [39010 77566]]
Accuracy Score : 0.675684517276

Report :
              precision    recall  f1-score   support

           0       0.67      0.69      0.68    115741
           1       0.68      0.67      0.67    116576

   micro avg       0.68      0.68      0.68    232317
   macro avg       0.68      0.68      0.68    232317
weighted avg       0.68      0.68      0.68    232317
```

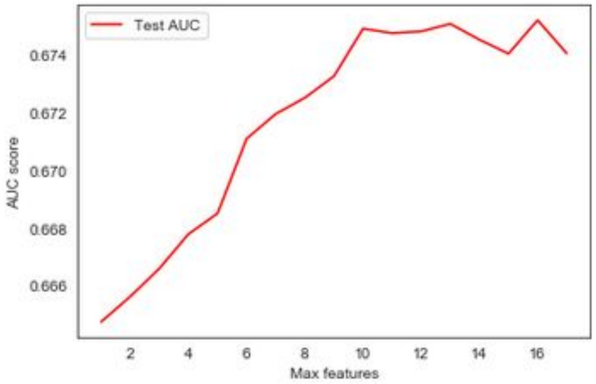
Tuning the RFC











Tuning the RFC

Tuning Summary:

1. **n_estimators - 37**
2. **max_depth - 15**
3. **min_sample_split - 0.1**
4. **min_samples_leaf - 0.26**
5. **max_features - 10**

Unfortunately, the model worsened

```
rfc_fs = ensemble.RandomForestClassifier(n_estimators=37, max_depth=15, min_samples_split=2,
                                         min_samples_leaf=0.1, min_weight_fraction_leaf=0.0,
                                         max_features=10, max_leaf_nodes=None, min_impurity_decrease=0.0,
                                         min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=-1,
                                         random_state=42, verbose=0, warm_start=False, class_weight="balanced")
cv_rfc_fs = cross_val_score(rfc_fs, Xdown_train, ydown_train, cv=5)
print("Cross Validation Scores: ", cv_rfc_fs)
print("Cross Validation Mean: ", cv_rfc_fs.mean())
print("Cross Validation Score Variance: ", cv_rfc_fs.var())
```

```
Cross Validation Scores:  [ 0.59897698  0.60639931  0.6078987   0.59899992  0.60600908]
Cross Validation Mean:  0.603656798955
Cross Validation Score Variance:  1.49271160965e-05
```

Conclusion

Model Shortcomings:

- Dockless data set is a year old with a rapidly growing dataset; Bcycle is 7 years old.
- Feature Engineering likely has more to do with improving the model vs tuning.
- I downsampled for ~1 million samples (~20% of the data set) for computation reasons (1.4GB combined). Given more time, I would evaluate whether 1 million data points is a good standard and measure f-1 scores of various samples sizes logarithmically distributed to confirm.

Future Proposals to Find New Relationships:

- Reveal device ID’s by company and scooter manufacturer.
- NLP project on the customer reviews per unique ride & scooter.
- Reexamine “seasonality” with a full year of Dockless data.