Lab 2: Linear Regression

The purpose of this lab is to give you some hands on experience applying linear regression to a real-world dataset. We will use a truncated version of the Divvy Bike Share dataset that we used in the last lecture.

Learning Objectives ¶

In this lab, you should gain experience applying linear regression to a real-world dataset, after exploring the linear relationships in the dataset. You will also learn to apply some basic metrics for the goodness of fit of your linear regresison model.

After completing this lab, you should be able to:

1. Manipulate a dataset in Python/Pandas/Jupyter Notebooks.
2. Learn about the importance of pre-processing your dataset, as well as how to do so. You should learn about:
   - Various ways to truncate and subset your data.
   - Normalizing your dataset in preparation for training and testing.
3. Learn how to apply the `scikit-learn` Linear Regression model to a real-world dataset, based on concepts that we covered in class. You should learn about:

   - Splitting your data into a training and testing set.
   - Creating a model.
   - Combining data and metrics from multiple models.
4. Learn how to evaluate your model. You should learn how to evaluate the various aspects of feature importance in your dataset, including MAE, MSE and R^2.

1. Loading and Preparing Your Datasets

1.1 Load the Divvy Bike Share Data

Download the smaller version of the [Divvy Trip data (https://data.cityofchicago.org/Transportation/Divvy-Trips/fg6s-gzvg)](https://data.cityofchicago.org/Transportation/Divvy-Trips/fg6s-gzvg) that we have provided on Box called `Divvy_Trips_2018.csv.gz` . Load this as a Pandas data frame.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn import datasets, linear_model
import warnings
warnings.filterwarnings('ignore')

regr = linear_model.LinearRegression()
```

In [2]:
```
ddf = pd.read_csv("/Users/mayar/Downloads/Divvy_Trips_2018.csv")
```

In [3]:
```
ddf.head(2)
```

Out[3]:

	TRIP ID	START TIME	STOP TIME	BIKE ID	TRIP DURATION	FROM STATION ID	FROM STATION NAME	TO STATION ID	TO STATION NAME
0	18511623	05/21/2018 05:07:29 PM	05/21/2018 05:11:48 PM	153	259	51	Clark St & Randolph St	91	Clinton St & Washington Blvd
1	18002370	04/01/2018 02:24:43 PM	04/01/2018 02:38:54 PM	3101	851	106	State St & Pearson St	174	Canal St & Madison St

1.2 Load the Weather Data from NOAA

We have downloaded some historical weather data for Chicago from the National Oceanic and Atmospheric Administration (NOAA) and provided the dataset on Box for download under `chicago-weather.csv.gz` . Load this as a Pandas data frame.

If you are curious about how we obtained the dataset, you can read about the available data (and make your own requests) [here (https://www.ncdc.noaa.gov/cdo-web/search)](https://www.ncdc.noaa.gov/cdo-web/search).You will also find this [documentation (https://www1.ncdc.noaa.gov/pub/data/cdo/documentation/GHCND_documentation.pdf)](https://www1.ncdc.noaa.gov/pub/data/cdo/documentation/GHCND_documentation.pdf) about the dataset useful, particularly the part describing the meanings of various columns.

In [4]:
```
wdf = pd.read_csv("/Users/mayar/Downloads/chicago-weather.csv")
```

In [5]:
```
wdf.head(2)
```

Out[5]:

	STATION	NAME	DATE	AWND	DAPR	MDPR	PGTM	PRCP	SNOW	SNWD	TAVG
0	US1ILDP0098	DOWNERS GROVE 0.9 S, IL US	2018-03-28	NaN	NaN	NaN	NaN	0.00	0.0	NaN	NaN
1	US1ILDP0098	DOWNERS GROVE 0.9 S, IL US	2018-03-29	NaN	NaN	NaN	NaN	0.12	NaN	NaN	NaN

1.3 Basic Data Analysis and Manipulation

We have provided some summary code below from the last lab, formatted to give you a good sense of what the datasets entail, as far as size, dates, and so forth. Note that in this example, the Divvy data frame is called `ddf` and the weather data frame is called `wdf` .

Let's do a brief review/overview of the data just to see what we have. This is a bit of a review of last week.

1.4.1 How many rows are in each dataset, and what date ranges do the dataset span?

This one we've done for you, to get you started, since it's a review from last week.

In [6]:
```
print("""
Divvy Data
----------
{} Rows
First Ride: {}, Last Ride {}

Weather Data
----------
{} Rows
First Measurement: {}, Last Measurement {}
"""
      .format(
          ddf.shape[0],
          ddf['START TIME'].min(),
          ddf['START TIME'].max(),
          wdf.shape[0],
          wdf['DATE'].min(),
          wdf['DATE'].max()
     ))
```

```
Divvy Data
----------
3602745 Rows
First Ride: 01/01/2018 01:00:00 AM, Last Ride 12/31/2018 12:59:05 PM

Weather Data
----------
79643 Rows
First Measurement: 2018-01-01, Last Measurement 2019-12-31
```

We can see from the above, first of all, that the date ranges overlap, but that they aren't perfectly overlapping. We'd like to work with data from a common date range from both datasets (and ideally a smaller sample, to start with!) so below we'll ask you to truncate the data. Before we get there, though, let's try to understand the weather dataset a bit more.

1.4.2 Understanding the weather data

We can take a quick look at the weather data, since we haven't had the chance to look at that before. Call `describe` to take a look at the overall statistics.

In [7]:
```
wdf.describe()
```

Out[7]:

	AWND	DAPR	MDPR	PGTM	PRCP	SNOW	SI
count	2185.000000	1303.000000	1295.00000	1458.000000	77491.000000	45370.000000	20556.00
mean	8.991611	4.745971	0.74088	1287.608368	0.143893	0.126284	0.81
std	3.543335	4.535482	0.99868	599.380253	0.364978	0.601270	2.22
min	1.340000	2.000000	0.00000	0.000000	0.000000	0.000000	0.00
25%	6.490000	2.000000	0.11000	950.000000	0.000000	0.000000	0.00
50%	8.500000	3.000000	0.44000	1337.000000	0.000000	0.000000	0.00
75%	11.180000	5.000000	0.98500	1646.750000	0.100000	0.000000	0.00
max	25.950000	49.000000	9.66000	2359.000000	6.520000	10.500000	24.00

Number of readings

The first thing you should note is that there are two years of data, but there are different numbers of readings for each type of weather measurement. Based on the summary statistics above, which variable would you suspect reflects one reading per day? Write your answer below:

TAVG: Because there any several stations

You should also immediately see that many of these columns have more than one measurement per day. Try to find out the reason for this. Some of the exercises below walk you through this exploration.

How many unique weather stations are there?

In [8]:
```
print("Number of unique weather stations: {}".format(wdf.STATION.nunique()))
```

```
Number of unique weather stations: 165
```

2. Exploring the Data Visually

How many readings does each weather station have?

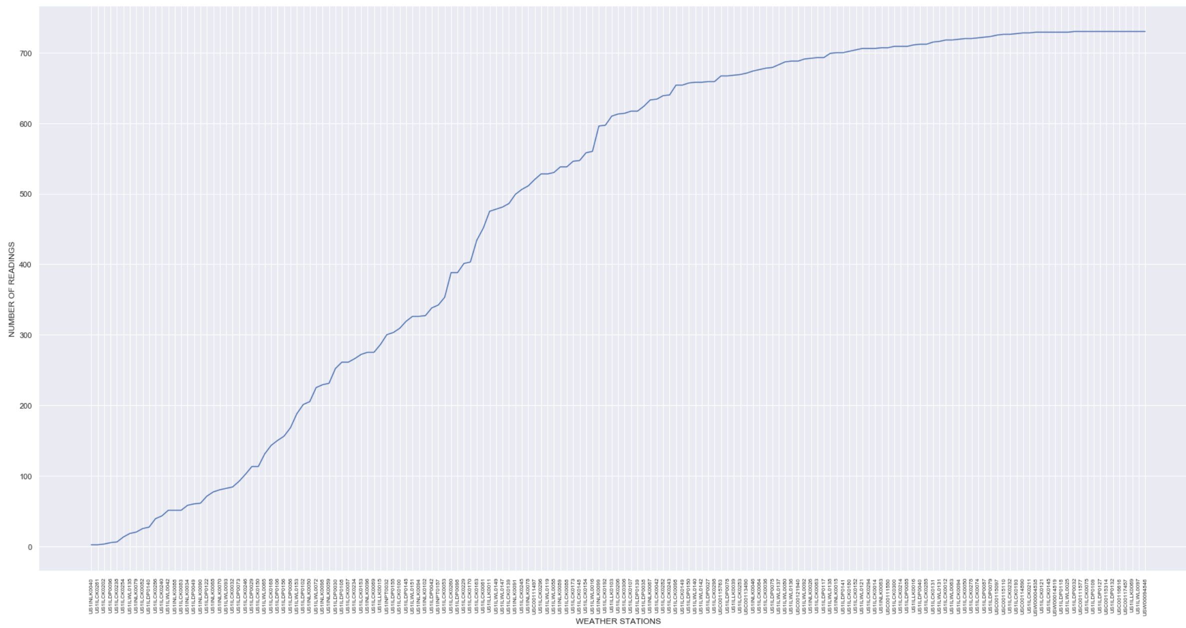
Show a plot that has the stations on the x-axis and the number of readings for that station on the y-axis. There are too many stations for a clean x label, so if possible just clean the x-axis up so that there are not a bunch of unreadable names. (We used Seaborn's `lineplot` function which automatically cleans things up, but you are welcome to take a different approach.) Be sure to label your axes.

In [9]:
```
wdf['READINGS'] = 1
```

In [10]:
```
wwdf = wdf[["STATION", "READINGS"]]
wwdf = wwdf.groupby('STATION').count().reset_index()
```

In [69]:
```
%matplotlib inline

sns.set(rc={'figure.figsize':(30, 15)})
wwdf = wwdf.sort_values(['READINGS'])
ax = sns.lineplot(x ='STATION', y = 'READINGS', data=wwdf, sort=False)
ax.set(xlabel="WEATHER STATIONS", ylabel = "NUMBER OF READINGS")
plt.draw()
labels = ax.get_xticklabels()
ax.set_xticklabels(labels, rotation=90, fontsize = 8)
plt.show()
```



What is the maximum number of readings that any station has?

Note that this number should make sense, as a sanity check.

In [12]:
```
print("The maximum number of readings a stations could have: {}".format(wwdf[
'READINGS'].max()))
```

```
The maximum number of readings a stations could have: 730
```

Which stations have the maximum number of readings?

Show your answer in a data frame. Hint: There are 12.

In [13]:
```
rslt_df = wwdf[(wwdf['READINGS'] == 730)]
rslt_df
```

Out[13]:

	STATION	READINGS
12	US1ILCK0075	730
66	US1ILDP0032	730
83	US1ILDP0109	730
87	US1ILDP0127	730
88	US1ILDP0132	730
100	US1ILLK0069	730
112	US1ILWL0097	730
152	USC00111577	730
157	USC00115314	730
159	USC00116616	730
160	USC00117457	730
164	USW00094846	730

3. Preparing the Datasets

3.1 Preparing the Weather Data

3.1.1 Selecting the appropriate fields

Build a data frame that contains (1) the date (2) the low temperature and (3) the high temperature for the Chicago Midway Airport station. Print the first few rows of the data frame.

In [14]:
```
wdf[wdf['NAME'].str.contains("CHICAGO MIDWAY")].head()
```

Out[14]:

	STATION	NAME	DATE	AWND	DAPR	MDPR	PGTM	PRCP	SNOW	SNWD	TA
32098	USC00111577	CHICAGO MIDWAY AIRPORT 3 SW, IL US	2018-01-01	NaN	NaN	NaN	NaN	0.0	0.0	2.0	N
32099	USC00111577	CHICAGO MIDWAY AIRPORT 3 SW, IL US	2018-01-02	NaN	NaN	NaN	NaN	0.0	0.0	2.0	N
32100	USC00111577	CHICAGO MIDWAY AIRPORT 3 SW, IL US	2018-01-03	NaN	NaN	NaN	NaN	0.0	0.2	2.0	N
32101	USC00111577	CHICAGO MIDWAY AIRPORT 3 SW, IL US	2018-01-04	NaN	NaN	NaN	NaN	0.0	0.0	2.0	N
32102	USC00111577	CHICAGO MIDWAY AIRPORT 3 SW, IL US	2018-01-05	NaN	NaN	NaN	NaN	0.0	0.0	2.0	N

In [15]:
```
wdf2 = wdf[wdf['STATION'] == 'USC00111577']
wdf2 = wdf2[["DATE", "TMAX", "TMIN"]]
wdf2
```

Out[15]:

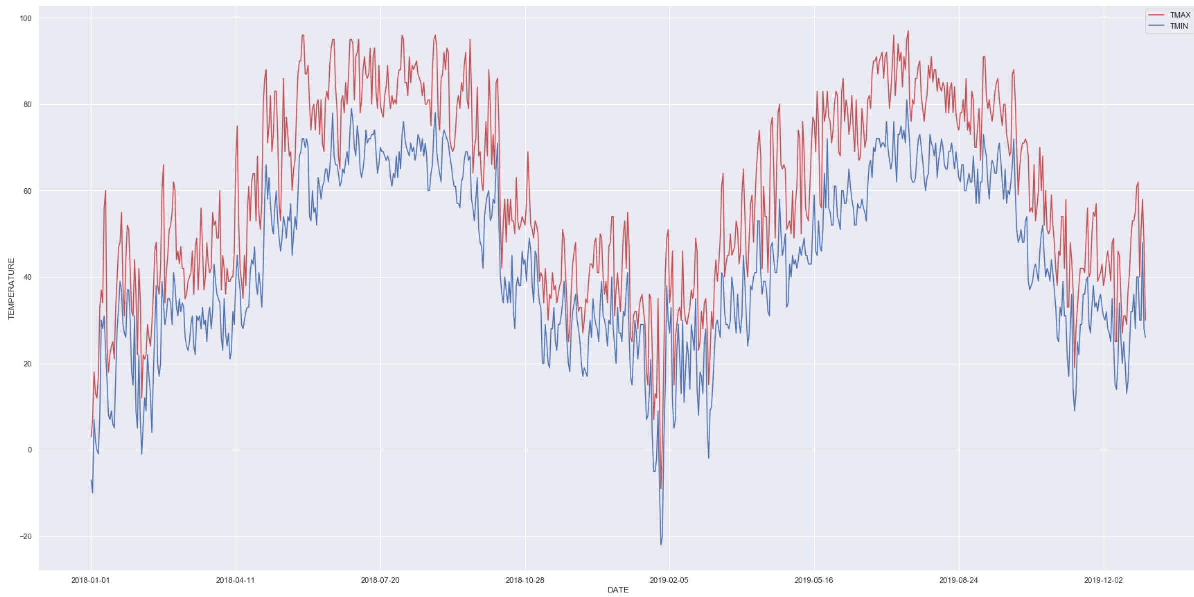
	DATE	TMAX	TMIN
32098	2018-01-01	3.0	-7.0
32099	2018-01-02	7.0	-10.0
32100	2018-01-03	18.0	7.0
32101	2018-01-04	13.0	2.0
32102	2018-01-05	12.0	0.0
...	...	...	...
32823	2019-12-27	40.0	30.0
32824	2019-12-28	51.0	30.0
32825	2019-12-29	58.0	48.0
32826	2019-12-30	48.0	28.0
32827	2019-12-31	30.0	26.0

730 rows × 3 columns

Plot the daily high and low temperature for Midway Airport for the duration of the dataset.

Does this match your expectations for what the plot should look like?

In [16]:
```
sns.set(rc={'figure.figsize':(30, 15)})
ax = wdf2.plot(x="DATE", y="TMAX", color="r")
ax.set(xlabel="DATE", ylabel = "TEMPERATURE")
ax2 = ax
wdf2.plot(x="DATE", y="TMIN", ax=ax2)
plt.show()
```



Restricting to 2018 data

You can see from the above that we have weather data through 2019, but we want to work with 2018 only. Truncate the dataset so that it only includes temperatures from 2018.

In [17]:
```
def lookup(s):
    dates = {date:pd.to_datetime(date) for date in s.unique()}
    return s.map(dates)
```

In [18]:
```
wdf2['DATE'] = lookup(wdf2['DATE'])
```

In [19]:
```
wdf2 = wdf2[wdf2['DATE'] <= '2018-12-31']
wdf2
```

Out[19]:

	DATE	TMAX	TMIN
32098	2018-01-01	3.0	-7.0
32099	2018-01-02	7.0	-10.0
32100	2018-01-03	18.0	7.0
32101	2018-01-04	13.0	2.0
32102	2018-01-05	12.0	0.0
...	...	...	...
32458	2018-12-27	54.0	40.0
32459	2018-12-28	54.0	28.0
32460	2018-12-29	31.0	24.0
32461	2018-12-30	37.0	20.0
32462	2018-12-31	41.0	33.0

365 rows × 3 columns

Check your work

Now check the shape of your dataset. You should have a $365x3$ matrix (date, low, high), for each date in 2018.

In [20]:
```
wdf2.shape
```

Out[20]: (365, 3)

3.3 Preparing the Divvy Data

Below will provide some experience with plotting rides over time.

3.3.1 Selecting the appropriate rows and columns

The Divvy data spans a longer timeframe than the weather data, and so we would like to match these up to the appropriate dates. Also note that the `START_TIME` column is more granular than we need (i.e. we are only concerned with date when merging with the weather data). Group these data so that each entry in the Divvy data corresponds to a single date.

Depending on how you performed the last lab, you may or may not be able to re-use some code from last week. Regardless, the `groupby` function should come in handy.

Truncate the data by date

The truncation of the Divvy data is not perfect because it was done by searching and selecting on the CSV. Fix the truncation so that only rides starting in 2018 are included.

In [21]:
```
ddf.head(5)
```

Out[21]:

	TRIP ID	START TIME	STOP TIME	BIKE ID	TRIP DURATION	FROM STATION ID	FROM STATION NAME	TO STATION ID	TO STATION NAME
0	18511623	05/21/2018 05:07:29 PM	05/21/2018 05:11:48 PM	153	259	51	Clark St & Randolph St	91	Clinton St & Washington Blvd
1	18002370	04/01/2018 02:24:43 PM	04/01/2018 02:38:54 PM	3101	851	106	State St & Pearson St	174	Canal St & Madison St
2	18002371	04/01/2018 02:25:30 PM	04/01/2018 02:32:41 PM	5226	431	426	Ellis Ave & 60th St	322	Kimbark Ave & 53rd St
3	18002372	04/01/2018 02:25:51 PM	04/01/2018 02:42:22 PM	4861	991	110	Dearborn St & Erie St	31	Franklin St & Chicago Ave
4	18002373	04/01/2018 02:25:58 PM	04/01/2018 02:42:23 PM	4706	985	214	Damen Ave & Grand Ave	47	State St & Kinzie St

In [22]:
```
ddf['TRIPSDAY'] = 1
```

In [23]:
```
ddf[['DATE','HOUR']] = ddf['START TIME'].str.split(" ",n=1,expand=True)
```

In [24]:
```
ddf['DATE'] = lookup(ddf['DATE'])
```

In [25]:
```
ddf.head(5)
```

Out[25]:

	TRIP ID	START TIME	STOP TIME	BIKE ID	TRIP DURATION	FROM STATION ID	FROM STATION NAME	TO STATION ID	TO STATION NAME
0	18511623	05/21/2018 05:07:29 PM	05/21/2018 05:11:48 PM	153	259	51	Clark St & Randolph St	91	Clinton St & Washington Blvd
1	18002370	04/01/2018 02:24:43 PM	04/01/2018 02:38:54 PM	3101	851	106	State St & Pearson St	174	Canal St & Madison St
2	18002371	04/01/2018 02:25:30 PM	04/01/2018 02:32:41 PM	5226	431	426	Ellis Ave & 60th St	322	Kimbark Ave & 53rd St
3	18002372	04/01/2018 02:25:51 PM	04/01/2018 02:42:22 PM	4861	991	110	Dearborn St & Erie St	31	Franklin St & Chicago Ave
4	18002373	04/01/2018 02:25:58 PM	04/01/2018 02:42:23 PM	4706	985	214	Damen Ave & Grand Ave	47	State St & Kinzie St

5 rows × 21 columns

In [26]:
```
ddf = ddf.set_index('DATE')
```

In [27]:
```
ddf = ddf.resample('1D').sum()
```

In [28]:
```
ddf.head(5)
```

Out[28]:

DATE	TRIP ID	BIKE ID	TRIP DURATION	FROM STATION ID	TO STATION ID	BIRTH YEAR	FROM LATITUDE	LON
2017-12-31	52610100	14185	3127	449	359	1988.0	125.704954	-262
2018-01-01	6576367554	1398434	249267	78622	76084	710128.0	15712.798168	-32866
2018-01-02	28640347053	6118784	981397	282266	283002	3222252.0	68417.296806	-143121
2018-01-03	43555314553	9214665	1376291	442684	435117	4853169.0	104026.865823	-217617
2018-01-04	42212356581	9062710	1334085	423455	417371	4710714.0	100803.308261	-210869

In [29]:
```
ddf2 = ddf.truncate(before = '2018-01-01', after = '2018-12-31')
```

In [30]:
```
ddf2.head(5)
```

Out[30]:

DATE	TRIP ID	BIKE ID	TRIP DURATION	FROM STATION ID	TO STATION ID	BIRTH YEAR	FROM LATITUDE	LON
2018-01-01	6576367554	1398434	249267	78622	76084	710128.0	15712.798168	-32866
2018-01-02	28640347053	6118784	981397	282266	283002	3222252.0	68417.296806	-143121
2018-01-03	43555314553	9214665	1376291	442684	435117	4853169.0	104026.865823	-217617
2018-01-04	42212356581	9062710	1334085	423455	417371	4710714.0	100803.308261	-210869
2018-01-05	38991044324	8066535	1185644	393951	395088	4336767.0	93089.411861	-194740

3.3.2 Grouping by Date

In [31]:
```
# Already grouped by date after **resampling**
ddf2 = ddf2[["TRIP DURATION", "TRIPSDAY"]]
```

Group the data by date to get number of rides

Now group the data by date so that we can align it with the weather data. Check the shape of your dataset. It should be $365x2$ (one total ride count per day).

In [32]:
```
ddf2.shape
```

Out[32]: (365, 2)

Which date in 2018 had the most number of rides?

In [33]:
```
ddf2[ddf2['TRIPSDAY']==ddf2['TRIPSDAY'].max()]
```

Out[33]:

DATE	TRIP DURATION	TRIPSDAY
2018-07-28	41886817	20663

Group the data by date to get total riding time by date

Now group the data by date so that we can align the total ride duration with the weather data. Check the shape of your dataset. It should be $365x2$ (one total duration of rides per day).

In [34]:
```
ddf2.shape
```
Out[34]:
```
(365, 2)
```

Which date in 2018 had the most riding time?

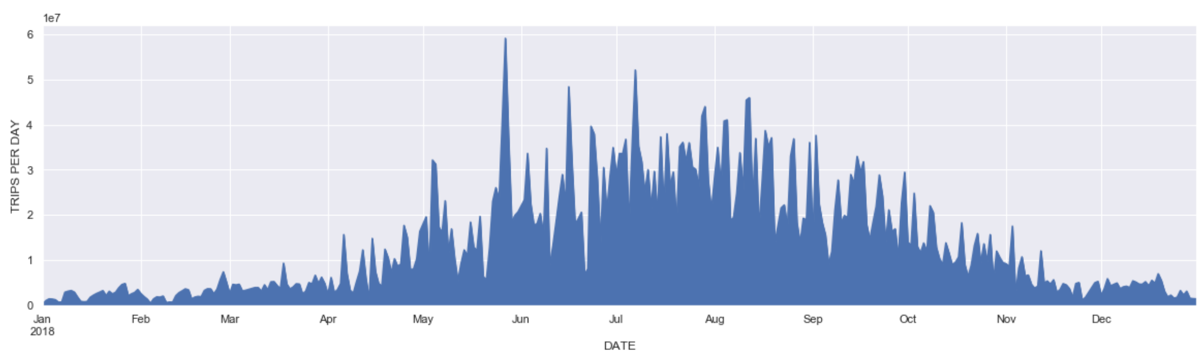
In [35]:
```
ddf2[ddf2['TRIP DURATION']==ddf2['TRIP DURATION'].max()]
```
Out[35]:

DATE	TRIP DURATION	TRIPSDAY
2018-05-27	59152068	18478

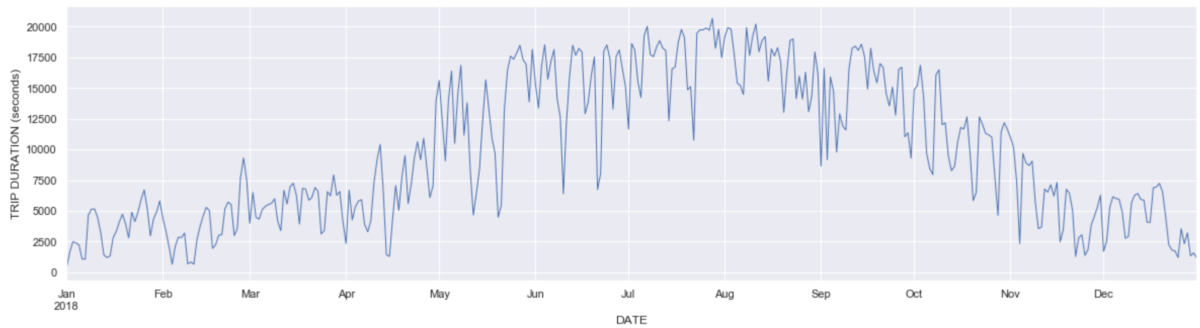
3.3.3 Visualizing the Temporal Data

As a sanity check that the Divvy data looks good and that there's a linear relationship between the datasets, let's plot the trips and duration by day (as we did last week).

In [36]:
```
sns.set(rc={'figure.figsize':(20, 5)})
ax = ddf2['TRIP DURATION'].plot.area()
ax.set(ylabel = "TRIPS PER DAY")
plt.show()
```



In [37]:
```
ax = ddf2['TRIPSDAY'].plot(linewidth=1)
ax.set(ylabel = "TRIP DURATION (seconds)")
plt.show()
```



3.3.4 Visualizing Relationships

We'll now look at a scatterplot of each of these against weather to explore the linear relationship between temperature and ride duration and number of rides.

Join your data into a single dataframe.

You may find it easy/useful to create a single dataframe with all of the data using the `merge` function.

In [38]:
```
mdf = ddf2.merge(wdf2, left_index = True, right_on = 'DATE')
```

In [39]:
```
mdf = mdf.set_index('DATE')
```

In [40]:
```
mdf.head(2)
```

Out[40]:

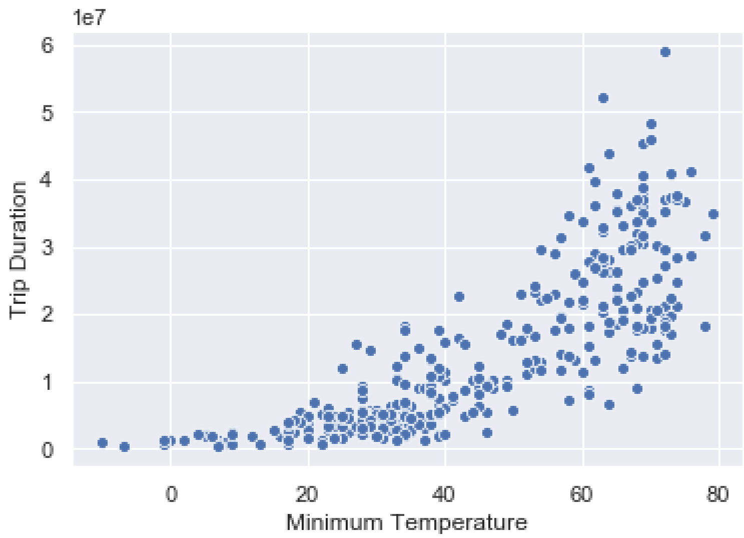
	TRIP DURATION	TRIPSDAY	TMAX	TMIN
DATE				
2018-01-01	249267	375	3.0	-7.0
2018-01-02	981397	1633	7.0	-10.0

Plot the scatterplots with the temperature and ride relationshps

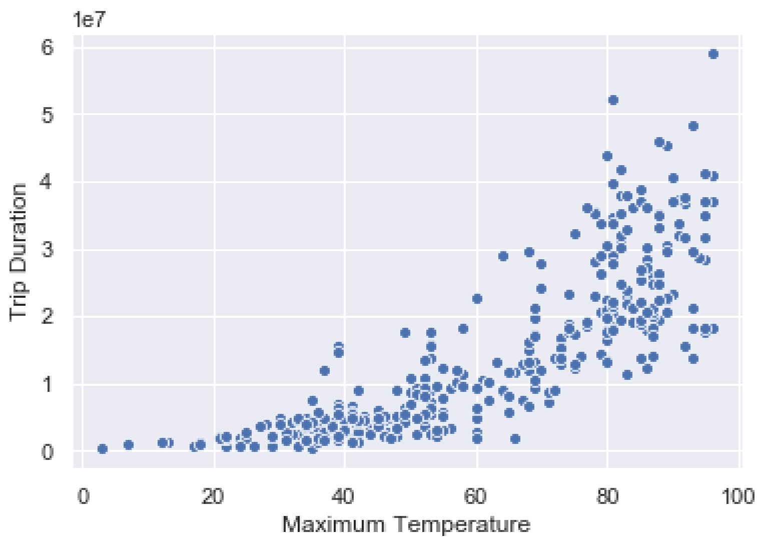
Provide four scatterplots:

1. Ride count vs. low temperature
2. Ride count vs. high temperature
3. Ride duration vs. low temperature
4. Ride duration vs. high temperature

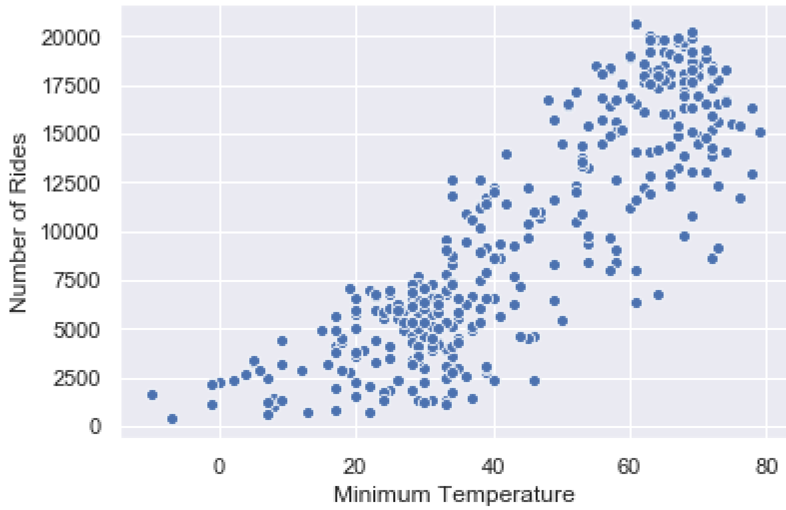
In [41]:
```
%matplotlib inline
ax = sns.scatterplot(x='TMIN', y="TRIP DURATION", data=mdf)
ax.set(xlabel="Minimum Temperature", ylabel = "Trip Duration")
plt.show()
```



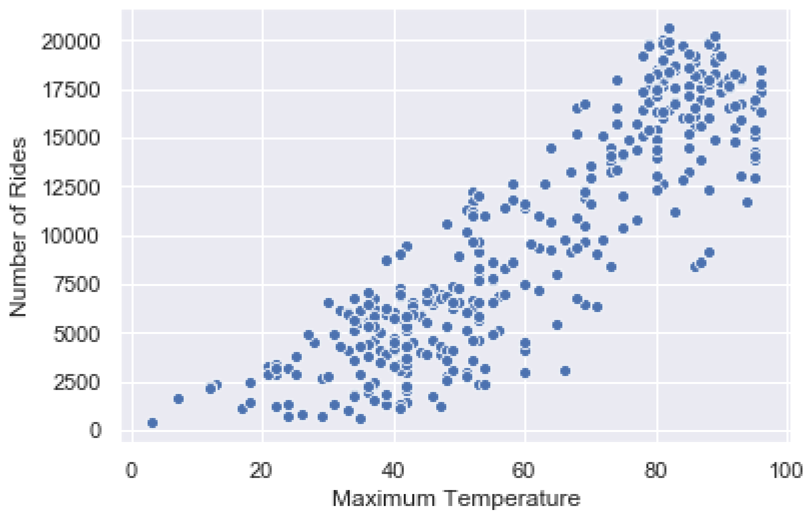
In [42]:
```
ax = sns.scatterplot(x="TMAX", y="TRIP DURATION", data=mdf)
ax.set(xlabel="Maximum Temperature", ylabel = "Trip Duration")
plt.show()
```



In [43]:
```
ax = sns.scatterplot(x="TMIN", y="TRIPSDAY", data=mdf)
ax.set(xlabel="Minimum Temperature", ylabel = "Number of Rides")
plt.show()
```



In [44]:
```
ax = sns.scatterplot(x="TMAX", y="TRIPSDAY", data=mdf)
ax.set(xlabel="Maximum Temperature", ylabel = "Number of Rides")
plt.show()
```



4. Linear Regression

At last, we are ready to apply linear regression to our data! Note that it took a **long** time to get to this stage. This is pretty much normal for real-world data science applications: You will spend a lot of time cleaning your data before you are ready to get to the machine learning/prediction.

4.1. Prepare Training and Test Sets

Although our data is in the right format, don't forget that you will want to normalize the values in the dataset before applying linear regression.

Normalize all of the temperature columns in the dataset to have zero mean and standard deviation of 1. Remember to normalize against the mean and standard deviation of the training sets only, as described [here (https://sebastianraschka.com/faq/docs/scale-training-test.html)](https://sebastianraschka.com/faq/docs/scale-training-test.html).

4.1.1 Split into training and testing

Hold out 20% of the dataset for testing. Your test set should be randomly sampled. Be sure to use a random seed.

Hint: `scikit-learn` has useful functions for doing this for you.

In [45]:
```
x = mdf[['TMAX', 'TMIN']]
y = mdf[['TRIP DURATION', 'TRIPSDAY']]
```

In [46]:
```
X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.20, random_state=42)
```

4.1.2 Normalize the features

Normalize the temperatures against the mean and standard deviation from the training set.

In [47]:
```
def normalize(df, df_train):
    df_copy = df.copy()
    for col in df:
        df_copy[col] = ((df.loc[:,(col)] - df_train.loc[:,(col)].mean()) / df_train.loc[:,(col)].std())

    return df_copy
```

In [48]:
```
X_train_normalized = normalize(X_train, X_train)
X_train_normalized.describe().round()
```

Out[48]:

	TMAX	TMIN
count	292.0	292.0
mean	0.0	0.0
std	1.0	1.0
min	-2.0	-3.0
25%	-1.0	-1.0
50%	-0.0	-0.0
75%	1.0	1.0
max	2.0	2.0

In [49]:
```
X_test_normalized = normalize(X_test, X_train)
X_test_normalized.describe().round()
```

Out[49]:

	TMAX	TMIN
count	73.0	73.0
mean	-0.0	-0.0
std	1.0	1.0
min	-3.0	-3.0
25%	-1.0	-1.0
50%	-0.0	-1.0
75%	1.0	1.0
max	2.0	1.0

Second Approach

In [50]:
```
normalizer = preprocessing.Normalizer().fit(X_train)
```

In [51]:
```
X_trainnorm = pd.DataFrame(normalizer.transform(X_train))
X_testnorm = pd.DataFrame(normalizer.transform(X_test))
```

In [52]:
```
X_trainnorm.describe().round()
```

Out[52]:

	0	1
count	292.0	292.0
mean	1.0	1.0
std	0.0	0.0
min	1.0	-1.0
25%	1.0	1.0
50%	1.0	1.0
75%	1.0	1.0
max	1.0	1.0

In [53]:
```
X_testnorm.describe().round()
```

Out[53]:

	0	1
count	73.0	73.0
mean	1.0	1.0
std	0.0	0.0
min	0.0	-1.0
25%	1.0	1.0
50%	1.0	1.0
75%	1.0	1.0
max	1.0	1.0

4.2 Apply Linear Regression to Ride Counts

Now we're ready to apply linear regression to the datasets! The ride count target `count` appears to have more of a linear relationship with minimum and maximum temperatures. Try those first to see which is the best predictor of `count` .

4.2.1 Single-Variable Linear Regression

First try each linear regression separately using `scikit-learn` 's `LinearRegression` . Report the Mean Absolute Error (MAE), Mean Squared Error (MSE) and R^2 for each instance.

4.2.1.1 Low Temperature

Compute the Linear Regression

Fit a linear regression model for `count` against daily low temperatures.

In [54]:
```
def do_regression(X_train, X_test, X_train_normalized, X_test_normalized, y_train, y_test, xcolumn, ycolumn):
    regr = linear_model.LinearRegression()
    target = y_train[ycolumn].values
    target_observed = y_test[ycolumn].values
    if xcolumn is "ALL":
        features = X_train_normalized
        features_notnorm = X_train
        features_test = X_test
        features_test_norm = X_test_normalized
        regr.fit(features,target)
        y_hat = regr.predict(features)
        target_predicted = regr.predict(features_test_norm)
        return target_predicted, features, target, target_observed, features_notnorm, features_test, features_test_norm, y_hat
    else:
        features = X_train_normalized[xcolumn].values.reshape(-1,1)
        features_notnorm = X_train[xcolumn].values.reshape(-1,1)
        features_test = X_test[xcolumn]
        features_test_norm = X_test_normalized[xcolumn].values.reshape(-1,1)
        regr.fit(features,target)
        y_hat = regr.predict(features)
        target_predicted = regr.predict(features_test_norm)
        return target_predicted, features, target, target_observed, features_notnorm, features_test, features_test_norm, y_hat
```

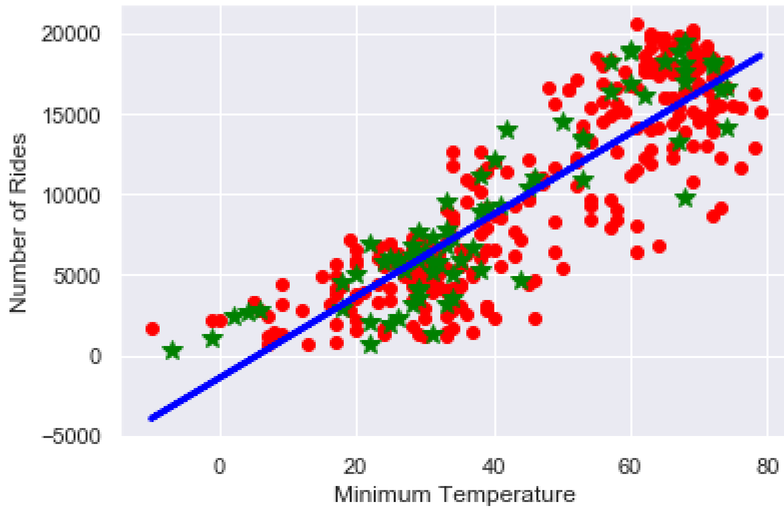
In [55]:
```
target_predicted, features, target, target_observed, features_notnorm, features_test, features_test_norm, y_hat = do_regression(X_train, X_test, X_train_normalized, X_test_normalized, y_train, y_test, 'TMIN', 'TRIPSDAY')
```

Plot the Best Fit Line Against the Complete Set of Points

Plot minimum temperature against the number of rides in the original units (NOT the normalized units used as features when training the model) for the complete set of 365 points. Add the line of best fit from the model, and be sure to label your axes.

In [56]:
```
plt.plot(features_notnorm, target, '.', color='red', markersize=12)
plt.plot(features_test, target_observed, '*', color='green', markersize=10)
plt.plot(features_notnorm, y_hat, color='blue', linewidth=3)

plt.xlabel('Minimum Temperature')
plt.ylabel('Number of Rides')
plt.show()
```



Compute the Error

Report the Mean Absolute Error (MAE), Mean Squared Error (MSE) and R^2 of this model on the testing set.

In [57]:
```
def print_metrics(string, target_observed, target_predicted):
    print(string)
    print(" ")
    mae = metrics.mean_absolute_error(target_observed, target_predicted)
    print("Mean Absolute Error:", mae)
    mse = metrics.mean_squared_error(target_observed, target_predicted)
    print("Mean Squared Error:", mse)
    r_squared = metrics.r2_score(target_observed, target_predicted)
    print("R^2:", r_squared)
```

In [58]:
```
print_metrics("**Error for linear regression model for number of trips against daily low temperatures**",target_observed, target_predicted)
```

```
**Error for linear regression model for number of trips against daily low tem
peratures**

Mean Absolute Error: 2139.0594742864246
Mean Squared Error: 6737729.444352465
R^2: 0.7929846229935164
```

4.2.1.2 High Temperature

Compute the Linear Regression

As above, fit a linear regression model for `count` against daily high temperatures.

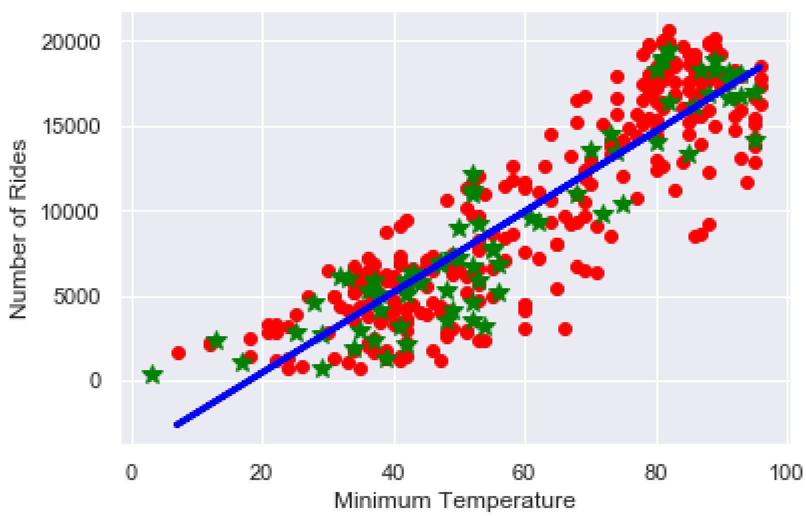
In [59]:
```
target_predicted, features, target, target_observed, features_notnorm, features_test, features_test_norm, y_hat = do_regression(X_train, X_test, X_train_normalized, X_test_normalized, y_train, y_test, 'TMAX', 'TRIPSDAY')
```

Plot the Best Fit Line Against the Complete Set of Points

As done above, plot maximum temperature against the number of rides in the original units for the complete set of points. Add the line of best fit from the model, and be sure to label your axes.

In [60]:
```
plt.plot(features_notnorm, target, '.', color='red', markersize=12)
plt.plot(features_test, target_observed, '*', color='green', markersize=10)
plt.plot(features_notnorm, y_hat, color='blue', linewidth=3)

plt.xlabel('Minimum Temperature')
plt.ylabel('Number of Rides')
plt.show()
```



Compute the Error

Report the Mean Absolute Error (MAE), Mean Squared Error (MSE) and R^2 of this model on the testing set.

In [61]:
```
print_metrics("**Error for linear regression model for number of trips against daily high temperatures**",target_observed, target_predicted)
```

```
**Error for linear regression model for number of trips against daily high te
mperatures**

Mean Absolute Error: 1813.383709133479
Mean Squared Error: 5211698.725222164
R^2: 0.8398716087731295
```

Interpret your results

Which variable (between daily minimum or maximum temperature) is a better predictor of ride count? Why?

Maximum temperature *Its R^2 is bigger which means that it explains better the variance in the number of trips. As well as lower Mean Absolute Error, which is the the average distance between each data point and the mean, and Mean Squared Error.*

4.3 Multi-Variable Linear Regression

Now try a multiple-variable regression with both low and high temperature. Plot your results and report the error.

How does it perform compared to the single-variable methods above? Why?

In [62]:
```
target_predicted, features, target, target_observed, features_notnorm, features_test, features_test_norm, y_hat = do_regression(X_train, X_test, X_train_normalized, X_test_normalized, y_train, y_test, "ALL", "TRIPSDAY")
```

In [63]:
```
print_metrics("**Error for linear regression model for number of trips against daily low and high temperatures**",target_observed, target_predicted)
```

```
**Error for linear regression model for number of trips against daily low and
high temperatures**

Mean Absolute Error: 1808.7490889509245
Mean Squared Error: 5053420.214169953
R^2: 0.8447346840729195
```

Incorporating more weather data

Include daily precipitation and snowfall from the NOAA data in your multi-variable regression. Remember to normalize the new variables.

Of the four variables now in your model, which is the best predictor of ride count? Why?

In [98]:
```
wdf3 = wdf[wdf['STATION'] == 'USC00111577']
wdf3 = wdf3[["DATE", "PRCP", "SNOW", "TMAX", "TMIN" ]]
wdf3['DATE'] = lookup(wdf3['DATE'])
wdf3 = wdf3[wdf3['DATE'] <= '2018-12-31']

mdf2 = ddf2.merge(wdf3, left_index = True, right_on = 'DATE')
mdf2 = mdf2.set_index('DATE')
```

In [99]:
```
x = mdf2[['PRCP','SNOW','TMAX', 'TMIN']]
y = mdf2[['TRIP DURATION', 'TRIPSDAY']]

X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(x,y,test_size=0.20
, random_state=42)

X_train_norm_m = normalize(X_train_m, X_train_m)
X_test_norm_m = normalize(X_test_m, X_train_m)
```

In [100]:
```
target_predicted, features, target, target_observed, features_notnorm, feature
s_test, features_test_norm, y_hat = do_regression(X_train_m, X_test_m, X_train
_norm_m, X_test_norm_m, y_train_m, y_test_m, "ALL", "TRIPSDAY")
```

In [101]:
```
print_metrics("**Error for linear regression model for number of trips against
daily low and high temperatures ADDED VARIABLES**",target_observed, target_pre
dicted)
```

```
**Error for linear regression model for number of trips against daily low and
high temperatures ADDED VARIABLES**

Mean Absolute Error: 1776.9058422547998
Mean Squared Error: 5012083.58017252
R^2: 0.8460047438077061
```

In [102]:
```
coeff_df = pd.DataFrame(regr.coef_, X_test_m.columns, columns=['Coefficient'])
coeff_df
```

Out[102]:

	Coefficient
PRCP	-1379.021964
SNOW	73.280227
TMAX	4044.753212
TMIN	1364.611097

This means that for one standard deviation more in "daily precipitation", there is a decrease of 1,379 rides. For one standard deviation more in "snowfall", there is an increase of ~73 rides. For every standard deviation more in "maximum temperature", there is an incrase of ~4,044 rides. Similarly, for every standard deviation more in "minimum temperature", there is an increase of ~1,364 rides. As a conclusion, after running multi-variable regression, apparently "maximum temperature" is the best predictor of ride count.

4.4 Polynomial Transformations of Predictors

Look back at your scatterplot of ride duration vs. daily high/low temperatures. The relationship between temperature and ride duration appears to better fit a polynomial (rather than a linear) function.

First fit a linear regression predicting `duration` using the two features of high and low temperatures. Then, apply a polynomial transformation to these predictors (e.g. square them) to see if this yields a better fit. Explain your results.

In [103]:
```
target_predicted, features, target, target_observed, features_notnorm, features_test, features_test_norm, y_hat = do_regression(X_train, X_test, X_train_normalized, X_test_normalized, y_train, y_test, "ALL", "TRIP DURATION")
```

In [104]:
```
print_metrics("**Error for linear regression model for trip duration against daily low and high temperatures**",target_observed, target_predicted)
```

```
**Error for linear regression model for trip duration against daily low and high temperatures**

Mean Absolute Error: 4238331.789521003
Mean Squared Error: 27548653830569.695
R^2: 0.7485291728380306
```

In [105]:
```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree = 2, include_bias = True)
pf = poly.fit_transform(X_train_normalized)
pft = poly.fit_transform(X_test_normalized)
```

In [106]:
```
ptarget_predicted, pfeatures, ptarget, ptarget_observed, pfeatures_notnorm, pfeatures_test, pfeatures_test_norm, py_hat = do_regression(X_train, X_test, pf, pft, y_train, y_test, "ALL", "TRIP DURATION")
```

In [107]:
```
print_metrics("**Error for polynomial regression model for trip duration against daily low and high temperatures**",ptarget_observed, ptarget_predicted)
```

```
**Error for polynomial regression model for trip duration against daily low and high temperatures**

Mean Absolute Error: 3311849.6743549453
Mean Squared Error: 20115023341185.55
R^2: 0.8163851638958415
```

The error measures are better for the polynomial regression, since R^2 is higher (0.81 vs 0.74) and errors are smaller, which means that regression with polynomial features better fits the data related with duration trips.

4.5 Regularization

We will cover the topic of regularization in class next week. For now, try out the `Ridge` and `Lasso` linear models in place of `LinearRegression`. In particular, explore how different values of the `alpha` parameter affect performance. (Hint: the `scikit-learn` documentation for Ridge (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html) and Lasso (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html) will be helpful.)

Comment on your results from changing this parameter when fitting models predicting `duration` using the four features used above (minimum temperature, maximum temperature, precipitation, and snowfall). How did changing the regularization parameter affect performance? Note that this question is intentionally meant to be open-ended to give you a chance to experiment with these parameters.

In [108]:
```
for i in range(0,1100,100):
    ls = linear_model.Lasso(alpha=i)
    rg = linear_model.Ridge(alpha=i)
    en = linear_model.ElasticNet(alpha=i)

    models = [(ls, 'Lasso'),
              (rg, 'Ridge'),
              (en, 'Elastic Net')]
    print('\n\033[1m' + 'Alpha set to: {}'.format(i) + '\033[0m\n')
    for m in models:
        (model,name) = m
        model.fit(features, target)
        target_predict = model.predict(features_test_norm)
        print('{}\n{}\n'.format(name,model.coef_))
        print_metrics("**Error for linear regression model for trip duration a
gainst daily low and high temperatures**",
                      target_observed, target_predict)
```

Alpha set to: 0

```
Lasso
[7798712.08880337 2347230.52508463]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 4238331.789521005
Mean Squared Error: 27548653830569.7
R^2: 0.7485291728380306
Ridge
[7798712.08880344 2347230.52508456]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 4238331.789521009
Mean Squared Error: 27548653830569.72
R^2: 0.7485291728380303
Elastic Net
[7798712.08880337 2347230.52508463]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 4238331.789521005
Mean Squared Error: 27548653830569.7
R^2: 0.7485291728380306
```

Alpha set to: 100

```
Lasso
[7801219.29915513 2344718.8233039 ]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 4238460.932821543
Mean Squared Error: 27549245683260.01
R^2: 0.748523770262409
Ridge
[4589681.96549726 4043915.68146669]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 4294755.812034244
Mean Squared Error: 27696642412318.06
R^2: 0.7471782969915494
Elastic Net
[192968.6482776  188817.81491078]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 9370666.31408583
```

```
Mean Squared Error: 110330088388687.39
R^2: -0.007119939819675292
```

Alpha set to: 200

```
Lasso
[7803770.09227272 2342165.20495363]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 4238592.335707303
Mean Squared Error: 27549851716205.7
R^2: 0.7485182382459599
Ridge
[3900443.14540414 3613180.58966872]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 4528610.949175856
Mean Squared Error: 30809136984289.05
R^2: 0.7187666878666725
Elastic Net
[98314.47645317 96238.26956165]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 9521288.713683676
Mean Squared Error: 113811722655470.36
R^2: -0.03890114605675943
```

Alpha set to: 300

```
Lasso
[7806178.66264375 2339748.37208572]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 4238716.3649349185
Mean Squared Error: 27550417225544.094
R^2: 0.7485130761388759
Ridge
[3422894.37805009 3227962.06031827]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 4839050.182035794
Mean Squared Error: 34932938932042.24
R^2: 0.6811236185090368
Elastic Net
[65959.9284857  64575.61482285]

**Error for linear regression model for trip duration against daily low and h
```

```
igh temperatures**

Mean Absolute Error: 9572787.907865489
Mean Squared Error: 115018498768409.94
R^2: -0.04991689256788212
```

Alpha set to: 400

```
Lasso
[7808135.15889657 2337766.33026085]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 4238816.955998914
Mean Squared Error: 27550850517385.41
R^2: 0.7485091209453333
Ridge
[3056668.48310928 2909150.51988481]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 5174270.488083643
Mean Squared Error: 39282525151696.195
R^2: 0.6414195352824748
Elastic Net
[49627.68993673 48589.38877512]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 9598786.737797983
Mean Squared Error: 115630896384802.98
R^2: -0.055507007282507015
```

Alpha set to: 500

```
Lasso
[7808083.79951632 2337715.38249739]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 4238813.448095606
Mean Squared Error: 27550691534798.992
R^2: 0.7485105721771341
Ridge
[2763688.94065786 2645032.39706792]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 5506834.833055122
Mean Squared Error: 43522585103855.62
R^2: 0.6027152345226789
Elastic Net
```

```
[39778.18108957 38947.50851801]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 9614466.718549857
Mean Squared Error: 116001263396333.98
R^2: -0.058887807640884526
```

Alpha set to: 600

```
Lasso
[7808032.44013605 2337664.43473394]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 4238809.940192295
Mean Squared Error: 27550532573520.57
R^2: 0.7485120232144304
Ridge
[2523011.67432343 2423771.21123803]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 5821253.3855024595
Mean Squared Error: 47516799620319.66
R^2: 0.566255070824844
Elastic Net
[33190.8041681  32498.55944495]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 9624953.874544082
Mean Squared Error: 116249405598134.58
R^2: -0.0611529101436914
```

Alpha set to: 700

```
Lasso
[7807981.0807558  2337613.48697049]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 4238806.432288986
Mean Squared Error: 27550373633550.16
R^2: 0.7485134740572221
Ridge
[2321412.80725111 2236127.75953651]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 6102859.148459909
```

```
Mean Squared Error: 51219274936473.73
R^2: 0.5324579736589974
Elastic Net
[28475.18784409 27881.82446039]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 9632461.34178342
Mean Squared Error: 116427256291730.17
R^2: -0.06277637462601704
```

Alpha set to: 800

```
Lasso
[7807929.72137554 2337562.53920703]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 4238802.924385675
Mean Squared Error: 27550214714887.76
R^2: 0.748514924705509
Ridge
[2149928.15686385 2075157.5225891 ]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 6350663.2368712295
Mean Squared Error: 54625571138185.47
R^2: 0.5013644716435701
Elastic Net
[24932.80052372 24413.60049899]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 9638101.069608198
Mean Squared Error: 116560977169309.89
R^2: -0.06399701139108771
```

Alpha set to: 900

```
Lasso
[7807878.36199528 2337511.59144357]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 4238799.416482366
Mean Squared Error: 27550055817533.355
R^2: 0.7485163751592914
Ridge
[2002200.03628788 1935635.80894329]

**Error for linear regression model for trip duration against daily low and h
```

```
igh temperatures**

Mean Absolute Error: 6566663.228286567
Mean Squared Error: 57749743051759.22
R^2: 0.47284626889818615
Elastic Net
[22174.24010455 21712.72408758]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 9642492.945108479
Mean Squared Error: 116665180121311.06
R^2: -0.06494820133646995
```

Alpha set to: 1000

```
Lasso
[7807827.00261502 2337460.64368012]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 4238795.908579056
Mean Squared Error: 27549896941486.957
R^2: 0.7485178254185689
Ridge
[1873569.50498874 1813588.46172314]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 6756475.474296199
Mean Squared Error: 60613309773123.11
R^2: 0.44670693386992444
Elastic Net
[19965.26494859 19549.89701686]

**Error for linear regression model for trip duration against daily low and h
igh temperatures**

Mean Absolute Error: 9646009.866140028
Mean Squared Error: 116748667393595.67
R^2: -0.06571029350794344
```

In [ ]: