Afzal Patel

This notebook is an analysis of public 2011-2012 Washington D.C. Bike Rental Data.

The two databases are included with the project:
bikeshare.db is used first, then bikeshare_11_12.db

In [1]:
```
#required modules and functions

import pandas as pd
import altair as alt
from altair import Chart, X, Y
from ipyleaflet import Map, Marker, Icon, CircleMarker, AntPath, Heatmap
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error,r2_score
from sklearn.model_selection import train_test_split
```

In [2]:
```
#read in station data
db="sqlite:///bikeshare.db"
query0 ="select * from bikeshare_stations where latitude is not NULL"
stations = pd.read_sql_query(query0,db)
stations.head()
```

Out[2]:

	index	station_id	name	status	latitude	longitude
0	0	31620	5th & F St NW	open	38.897637	-77.018126
1	1	31105	14th & Harvard St NW	open	38.926638	-77.032126
2	2	31400	Georgia & New Hampshire Ave NW	closed	38.935638	-77.024126
3	3	31111	10th & U St NW	open	38.917638	-77.025126
4	4	31104	Adams Mill & Columbia Rd NW	open	38.922638	-77.042126

In [3]:
```
#sorted by start_date, limited to the first 10, and using parse_dates parameter from trip_data table in bikeshare.db
query1="select * from trip_data ORDER BY start_date LIMIT 10"
calcDuration = pd.read_sql_query(query1,db,parse_dates=['start_date','end_date'])
```

In [4]:
```
#new column that shows trip durations based on difference between star
t and end dates
calcDuration['calc_duration'] = calcDuration["end_date"] - calcDuratio
n["start_date"]
```

In [5]:
```
print(calcDuration['calc_duration'].dtypes)
calcDuration
```

```
timedelta64[ns]
```

Out[5]:

	index	duration	start_date	end_date	start_station	end_station	bike_number	member_type
0	0	3548	2011-01-01 00:01:29	2011-01-01 01:00:37	31620	31620	W00247	Membe
1	1	346	2011-01-01 00:02:46	2011-01-01 00:08:32	31105	31101	W00675	Casua
2	2	562	2011-01-01 00:06:13	2011-01-01 00:15:36	31400	31104	W00357	Membe
3	3	434	2011-01-01 00:09:21	2011-01-01 00:16:36	31111	31503	W00970	Membe
4	4	233	2011-01-01 00:28:26	2011-01-01 00:32:19	31104	31106	W00346	Casua
5	5	158	2011-01-01 00:32:33	2011-01-01 00:35:11	31605	31618	W01033	Membe
6	6	560	2011-01-01 00:35:48	2011-01-01 00:45:09	31203	31201	W00766	Membe
7	7	503	2011-01-01 00:36:42	2011-01-01 00:45:05	31203	31201	W00506	Membe
8	8	449	2011-01-01 00:45:55	2011-01-01 00:53:24	31201	31202	W00506	Membe
9	9	442	2011-01-01 00:46:06	2011-01-01 00:53:28	31201	31202	W00766	Membe

In [6]:
```
#may be required to use ipyleaflet
!jupyter nbextension enable --py --sys-prefix ipyleaflet
```

```
Enabling notebook extension jupyter-leaflet/extension...
      - Validating: OK
```

In [7]:
```
#Mapping All Station Data

#zip lat and long, and convert to list
locations = list(zip(stations.latitude, stations.longitude))
dc_center = (38.9072, -77.0369)

dcmap = Map(center=dc_center, zoom=12)

for loc in locations:
    marker = CircleMarker(location=loc, radius=2)
    dcmap.add_layer(marker)

dcmap
```

In [8]:
```
#Redone the map so that it only shows the 20 busiest start and end sta
tions
#stations can be both starting and ending

query2 = """SELECT name,latitude,longitude, count(start_date)
            FROM trip_data
            JOIN bikeshare_stations ON station_id = start_station
            WHERE latitude is not NULL
            GROUP BY station_id
            ORDER BY count(start_date) DESC
            LIMIT 20"""

query3 = """SELECT name,latitude,longitude, count(end_date)
            FROM trip_data
            JOIN bikeshare_stations ON station_id = end_station
            WHERE latitude is not NULL
            GROUP BY station_id
            ORDER BY count(end_date) DESC
            LIMIT 20"""

busyStartStations = pd.read_sql_query(query2,db)
busyEndStations = pd.read_sql_query(query3,db)

#zip lat and long, and convert to list
locations2 = list(zip(busyStartStations.latitude, busyStartStations.lo
ngitude))
locations3 = list(zip(busyEndStations.latitude, busyEndStations.longit
ude))


#dc_center defined previously ; dc_center = (38.9072, -77.0369)
dcmap2 = Map(center=dc_center, zoom=12)
for loc in locations3:
    marker = CircleMarker(location=loc, radius=1,color='red')
    dcmap2.add_layer(marker)

for loc in locations2:
    marker = CircleMarker(location=loc, radius=1,color='green')
    dcmap2.add_layer(marker)

    #green = 20 busiest start stations
    #red = 20 busiest end stations

#It seems life a lot of the stations overlap, first adding the busiest
end stations, then layering on the start stations so they appear.
dcmap2
```

In [9]:
```
# color coded the markers based
# on the number of rides originating from that station.
#Show the 50 most popular stations using five different colors.
#1 color for the top 10 another for the next 10 and so on.

#find 50 busiest stations
query4 = """SELECT name,latitude,longitude, count(start_date)
            FROM trip_data
            JOIN bikeshare_stations ON station_id = start_station
            WHERE latitude is not NULL
            GROUP BY station_id
            ORDER BY count(start_date) DESC
            LIMIT 50"""
busyStartStations2 = pd.read_sql_query(query4,db)
busyStartStations2
```

Out[9]:

	name	latitude	longitude	count(start_date)
0	Massachusetts Ave & Dupont Circle NW	38.909764	-77.044213	49093
1	15th & P St NW	38.909637	-77.034126	32884
2	Columbus Circle / Union Station	40.763960	-74.929315	32058
3	Adams Mill & Columbia Rd NW	38.922638	-77.042126	29964
4	17th & Corcoran St NW	38.911638	-77.038126	28160
5	14th & V St NW	38.918638	-77.031126	25222
6	New Hampshire Ave & T St NW	38.915638	-77.037126	24074
7	20th St & Florida Ave NW	38.915638	-77.044126	23971
8	14th & Rhode Island Ave NW	38.908637	-77.031126	23871
9	21st & I St NW	38.897534	-77.002036	23574
10	Eastern Market Metro / Pennsylvania Ave & 7th ...	38.890369	-77.031960	23002
11	8th & H St NW	38.899864	-77.022126	22762
12	USDA / 12th & Independence Ave SW	38.887464	-77.028013	20799
13	Calvert & Biltmore St NW	38.923638	-77.047126	19613
14	25th St & Pennsylvania Ave NW	38.903637	-77.053513	19125
15	16th & Harvard St NW	38.926638	-77.036126	18322
16	Park Rd & Holmead Pl NW	38.930638	-77.030126	18172
17	5th & K St NW	38.914638	-77.018126	18066
18	21st & M St NW	38.905637	-77.046126	17764

19	14th & Harvard St NW	38.926638	-77.032126	17507
20	18th & M St NW	38.905637	-77.041126	17373
21	Lincoln Park / 13th & East Capitol St NE	38.890369	-77.031960	17351
22	North Capitol St & F St NW	38.897464	-77.009126	16852
23	New York Ave & 15th St NW	38.898637	-77.033126	16833
24	10th & U St NW	38.917638	-77.025126	16737
25	C & O Canal & Wisconsin Ave NW	38.904164	-77.062713	16646
26	10th St & Constitution Ave NW	38.892637	-77.026126	16500
27	L'Enfant Plaza / 7th & C St SW	38.890369	-77.031960	16253
28	14th & R St NW	38.912638	-77.031126	16001
29	17th & K St NW	38.902564	-77.039513	15777
30	7th & F St NW / National Portrait Gallery	38.897903	-77.023178	15739
31	Lamont & Mt Pleasant NW	38.890369	-77.031960	15587
32	5th & F St NW	38.897637	-77.018126	14942
33	19th St & Constitution Ave NW	38.892264	-77.043126	14674
34	13th St & New York Ave NW	38.899637	-77.029126	14534
35	Convention Center / 7th & M St NW	38.876395	-77.016365	14329
36	19th St & Pennsylvania Ave NW	38.900164	-77.043313	14118
37	Metro Center / 12th & G St NW	38.890369	-77.031960	13594
38	11th & Kenyon St NW	38.929764	-77.027126	13364
39	19th & L St NW	38.903637	-77.043126	13265
40	3rd & D St SE	38.885637	-77.002113	13066
41	4th & East Capitol St NE	38.889637	-77.000126	12975
42	20th & E St NW	38.896637	-77.044713	12973
43	1st & M St NE	38.905637	-77.005126	12906
44	3rd & H St NW	38.899664	-77.015113	12535
45	Eastern Market / 7th & North Carolina Ave SE	38.890369	-77.031960	11966
46	Florida Ave & R St NW	38.912774	-77.014011	11919
47	3rd & H St NE	38.900264	-77.001813	11661
48	Georgetown Harbor / 30th St NW	38.919422	-77.059459	11416
49	13th & H St NE	38.900064	-76.988313	11004

In [10]:
```
#zip lat and long, and convert to list
locations4 = list(zip(busyStartStations2.latitude, busyStartStations2.
longitude))

dcmap3 = Map(center=dc_center, zoom=12)

for counter, loc in enumerate(locations4):
    if(0  <= counter < 10):
        marker = CircleMarker(location=loc, radius=1,color='blue')
        dcmap3.add_layer(marker)
    if(10 <= counter < 20):
        marker = CircleMarker(location=loc, radius=1,color='green')
        dcmap3.add_layer(marker)
    if(20 <= counter < 30):
        marker = CircleMarker(location=loc, radius=1,color='red')
        dcmap3.add_layer(marker)
    if(30 <= counter < 40):
        marker = CircleMarker(location=loc, radius=1,color='orange')
        dcmap3.add_layer(marker)
    if(40 <= counter < 50):
        marker = CircleMarker(location=loc, radius=1,color='yellow')
        dcmap3.add_layer(marker)

#key for end stations routes: Top 10=blue, Second 10=green, Third 10=r
ed, Fourth 10=orange, Bottom 10=yellow
dcmap3
```

In [11]:
```
busyStartStations2 = busyStartStations2.rename(columns={"count(start_date)": "countz"})

#finding the busiest 15 routes:

query5 = """SELECT start_station, end_station, count(end_station),latitude,longitude
            FROM trip_data
            JOIN bikeshare_stations ON station_id = end_station
            WHERE latitude is not NULL AND longitude is not NULL
            GROUP BY end_station
            ORDER BY count(end_station) DESC
            LIMIT 10"""
#df from query
busyRoutes = pd.read_sql_query(query5,db)

#it appears two stations are the origin of the busiest routes, namely 31000 & 31001
busyRoutes = busyRoutes.rename(columns={"count(end_station)": "count1s"})
busyRoutes
```

Out[11]:

	start_station	end_station	count1s	latitude	longitude
0	31000	31200	53910	38.909764	-77.044213
1	31001	31201	36638	38.909637	-77.034126
2	31000	31623	32252	40.763960	-74.929315
3	31000	31214	29103	38.911638	-77.038126
4	31001	31104	26248	38.922638	-77.042126
5	31001	31228	25384	38.899864	-77.022126
6	31000	31205	25283	38.897534	-77.002036
7	31001	31101	25139	38.918638	-77.031126
8	31000	31203	24525	38.908637	-77.031126
9	31000	31217	24281	38.887464	-77.028013

In [12]:
```
#mapping the busiest 15 routes

dcmap4 = dcmap3

#zip lat and long, and convert to list
locations5 = list(zip(busyRoutes.latitude, busyRoutes.longitude))

#temp lists
```

```
routesRank1 = []
routesRank2 = []
routesRank3 = []
routesRank4 = []
routesRank5 = []

for counter, loc in enumerate(locations5):
    if(0  <= counter < 3):
        routesRank1.append(loc)
    if(3 <= counter < 6):
        routesRank2.append(loc)
    if(6 <= counter < 9):
        routesRank3.append(loc)
    if(9 <= counter < 12):
        routesRank4.append(loc)
    if(12 <= counter < 15):
        routesRank5.append(loc)

#ant paths from route date added as layers to previous map
busyRoutesAntPathsRank1 = AntPath(locations=routesRank1,dash_array=[1,
10],delay=1000,color='blue',pulse_color='blue')
busyRoutesAntPathsRank2 = AntPath(locations=routesRank2,dash_array=[1,
10],delay=1000,color='green',pulse_color='green')
busyRoutesAntPathsRank3 = AntPath(locations=routesRank3,dash_array=[1,
10],delay=1000,color='red',pulse_color='red')
busyRoutesAntPathsRank4 = AntPath(locations=routesRank4,dash_array=[1,
10],delay=1000,color='orange',pulse_color='orange')
busyRoutesAntPathsRank5 = AntPath(locations=routesRank5,dash_array=[1,
10],delay=1000,color='yellow',pulse_color='yellow')


dcmap4.add_layer(busyRoutesAntPathsRank1)
dcmap4.add_layer(busyRoutesAntPathsRank2)
dcmap4.add_layer(busyRoutesAntPathsRank3)
dcmap4.add_layer(busyRoutesAntPathsRank4)
dcmap4.add_layer(busyRoutesAntPathsRank5)

#key for routes: Top 3=blue, Second 3=green, Third 3=red, Fourth 3=ora
nge, Bottom 3=yellow
dcmap4
#clearly some sort of ourlier, maybe when sending bikes for inspection
at the factory, or bringing back stolen bikes
```

In [13]:
```
#Adding the heatmap layer, intensity based on (counts/10).

#reformat so data is of form (latitude, longitude, intensity) for heat
map

#helps calculate intensities
def helper1(aNum):
    return (aNum/10)


locations6 = list(zip(busyStartStations2.latitude, busyStartStations2.
longitude, helper1(busyStartStations2.countz)))
locations7 = list(zip(busyRoutes.latitude, busyRoutes.longitude, helpe
r1(busyRoutes.count1s)))

heatmapLayerForEndStations = Heatmap(locations=locations6,radius=11)
heatmapLayerForRoutes = Heatmap(locations=locations7,radius=11)

dcmap4.add_layer(heatmapLayerForEndStations)
dcmap4.add_layer(heatmapLayerForRoutes)
dcmap4
```

In [14]:
```
query6="select * from trip_data ORDER BY start_date"
bikeData = pd.read_sql_query(query6,db,parse_dates=['start_date','end_
date'])
bikeData.head()
```

Out[14]:

	index	duration	start_date	end_date	start_station	end_station	bike_number	member_type
0	0	3548	2011-01-01 00:01:29	2011-01-01 01:00:37	31620	31620	W00247	Membe
1	1	346	2011-01-01 00:02:46	2011-01-01 00:08:32	31105	31101	W00675	Casua
2	2	562	2011-01-01 00:06:13	2011-01-01 00:15:36	31400	31104	W00357	Membe
3	3	434	2011-01-01 00:09:21	2011-01-01 00:16:36	31111	31503	W00970	Membe
4	4	233	2011-01-01 00:28:26	2011-01-01 00:32:19	31104	31106	W00346	Casua

In [15]:
```
#Using the data in the bikeshare.db, I created a dataframe to see how
many rides occurred on June 8 2011, the day I turned 12

resamp = bikeData.set_index('start_date')
resamp = resamp.resample('D').count()
resamp = resamp.reset_index()
val = resamp[resamp['start_date']=='2011-06-08']
print(val['index'])
```

```
158    4358
Name: index, dtype: int64
```

Creating and Evaluating a Linear Regression Model to predcit Trips on a given day

In [16]:
```
#Read in the rental data from the database.

#from db:
db="sqlite:///bikeshare_11_12.db"
query0 ="select * from trip_data"
trips = pd.read_sql_query(query0,db,parse_dates=['start_date','end_dat
e'])
trips.head()
```

Out[16]:

	duration	start_date	end_date	start_station	end_station	bike_number	member_type
0	3548	2011-01-01 00:01:29	2011-01-01 01:00:37	31620	31620	W00247	Member
1	346	2011-01-01 00:02:46	2011-01-01 00:08:32	31105	31101	W00675	Casual
2	562	2011-01-01 00:06:13	2011-01-01 00:15:36	31400	31104	W00357	Member
3	434	2011-01-01 00:09:21	2011-01-01 00:16:36	31111	31503	W00970	Member
4	233	2011-01-01 00:28:26	2011-01-01 00:32:19	31104	31106	W00346	Casual

In [17]:
```
print("First Date: %s in dataset",(trips['start_date'].iloc[0]))
print("Last Date in dataset: %s",(trips['start_date'].iloc[-1]))
```

```
First Date: %s in dataset 2011-01-01 00:01:29
Last Date in dataset: %s 2012-12-31 23:59:23
```

In [18]:
```
#Transform the data into daily rental counts by resampling by day.

#set index to date
trips = trips.set_index('start_date')
#count dates
trips = trips.resample('D').count()

#remove other columns
trips = trips[['duration']]
#rename second column
trips.columns=['trip_count']
#reset index
trips = trips.reset_index()

#Q-1: How many days of data do you have in the transformed data set (b
efore the train test split)?
# = 731

print(trips.shape)
trips.head()
```

```
(731, 2)
```

Out[18]:

	start_date	trip_count
0	2011-01-01	959
1	2011-01-02	781
2	2011-01-03	1301
3	2011-01-04	1536
4	2011-01-05	1571

In [19]:
```
#Number each day from 0 to N, and create a daynum column to serve as a label for day number
labels = [x for x in range(0,731)]
trips['daynum'] = labels
trips.head()
```

Out[19]:

	start_date	trip_count	daynum
0	2011-01-01	959	0
1	2011-01-02	781	1
2	2011-01-03	1301	2
3	2011-01-04	1536	3
4	2011-01-05	1571	4

In [20]:
```
import datetime as dt

#four new features out of the date, a column for year, month, day, and weekday.
#Later we can experiment to see if we need it at all.
#We will keep daynum as a feature as well, so that we can use it to build a graph.

trips['year'] =  trips['start_date'].dt.year
trips['month'] =  trips['start_date'].dt.month
trips['day'] =  trips['start_date'].dt.day
trips['weekday'] =  trips['start_date'].dt.weekday

trips.head()
```

Out[20]:

	start_date	trip_count	daynum	year	month	day	weekday
0	2011-01-01	959	0	2011	1	1	5
1	2011-01-02	781	1	2011	1	2	6
2	2011-01-03	1301	2	2011	1	3	0
3	2011-01-04	1536	3	2011	1	4	1
4	2011-01-05	1571	4	2011	1	5	2

In [21]:
```
#one hot encodings for seasons

def getWinterEncoding(aNum):
    if(aNum == 11 or aNum == 12 or aNum == 1 or aNum == 2 ):
        return 1
    else:
        return 0
def getSpringEncoding(aNum):
    if(aNum == 3 or aNum == 4 or aNum == 5):
        return 1
    else:
        return 0
def getSummerEncoding(aNum):
    if(aNum == 6 or aNum == 7 or aNum == 8):
        return 1
    else:
        return 0
def getFallEncoding(aNum):
    if(aNum == 9 or aNum == 10):
        return 1
    else:
        return 0

trips['winter'] =  trips['month'].apply(lambda x: getWinterEncoding(x)
)
trips['spring'] =  trips['month'].apply(lambda x: getSpringEncoding(x)
)
trips['summer'] =  trips['month'].apply(lambda x: getSummerEncoding(x)
)
trips['fall'] =    trips['month'].apply(lambda x: getFallEncoding(x))


trips.head()
```

Out[21]:

	start_date	trip_count	daynum	year	month	day	weekday	winter	spring	summer	fall
0	2011-01-01	959	0	2011	1	1	5	1	0	0	0
1	2011-01-02	781	1	2011	1	2	6	1	0	0	0
2	2011-01-03	1301	2	2011	1	3	0	1	0	0	0
3	2011-01-04	1536	3	2011	1	4	1	1	0	0	0
4	2011-01-05	1571	4	2011	1	5	2	1	0	0	0

In [22]:
```
#gather weather data from db and parse date

query ="select * from weather"
weather = pd.read_sql_query(query,db,parse_dates=['date'])
weather.columns
```

Out[22]:
```
Index(['date', 'hour', 'weathersit', 'temp_f', 'feelslike_f', 'humid
ity',
       'windspeed'],
      dtype='object')
```

Here the weather data is reworked so it holds daily averages for chosen variables

In [23]:
```
#set index to group by date
weather = weather.set_index('date')

#average vals for each date
weather = weather.resample('D').mean()

#remove other columns except four target averages
weather = weather[['feelslike_f', 'weathersit','humidity','windspeed']
]

#rename columns
weather.columns=['Feels_Like','situation','humidity','w_speed']

#reset index
weather = weather.reset_index()

#rename to help with merge unique indentifier
weather = weather.rename(columns={'date': 'start_date'})

weather.head()
```

Out[23]:

	start_date	Feels_Like	situation	humidity	w_speed
0	2011-01-01	46.398650	1.583333	80.583333	10.749871
1	2011-01-02	45.224209	1.956522	69.608696	16.652122
2	2011-01-03	25.701260	1.000000	43.727273	16.636709
3	2011-01-04	28.400063	1.043478	59.043478	10.739809
4	2011-01-05	30.437224	1.000000	43.695652	12.522300

In [24]:
```
#merge previous trips dataframe and current weather dataframe w/ uniqu
e indentifier being the date
mergedDF = pd.merge(trips, weather, on='start_date')

#check combined weather and trips dataframes
mergedDF.head()
```

Out[24]:

	start_date	trip_count	daynum	year	month	day	weekday	winter	spring	summer	fall
0	2011-01-01	959	0	2011	1	1	5	1	0	0	0
1	2011-01-02	781	1	2011	1	2	6	1	0	0	0
2	2011-01-03	1301	2	2011	1	3	0	1	0	0	0
3	2011-01-04	1536	3	2011	1	4	1	1	0	0	0
4	2011-01-05	1571	4	2011	1	5	2	1	0	0	0

In [25]:
```
#re-scale data on a [0,1] scale into a new df
#0 is min and 1 is max, so we need max() function
# /= operator was giving invalid syntax
#this line does the following:

# rescaledDF = [ columnValues = [columnValues / columnValues.max()] ]

rescaledDF = mergedDF[['trip_count','year','month', 'weekday','Feels_L
ike', 'situation','humidity',
                       'w_speed','summer', 'spring', 'fall', 'winter']
] = mergedDF[['trip_count','year',

'month', 'weekday',

'Feels_Like', 'situation',

'humidity','w_speed','summer',

'spring', 'fall', 'winter']] / mergedDF[['trip_count','year',

'month', 'weekday','Feels_Like',

'situation','humidity','w_speed',

'summer', 'spring', 'fall', 'winter']].max()

rescaledDF.head()
```

Out[25]:

	trip_count	year	month	weekday	Feels_Like	situation	humidity	w_speed	summer
0	0.111214	0.999503	0.083333	0.833333	0.450042	0.527778	0.828620	0.316173	0.0
1	0.090572	0.999503	0.083333	1.000000	0.438651	0.652174	0.715771	0.489768	0.0
2	0.150876	0.999503	0.083333	0.000000	0.249289	0.333333	0.449638	0.489315	0.0
3	0.178128	0.999503	0.083333	0.166667	0.275466	0.347826	0.607131	0.315877	0.0
4	0.182187	0.999503	0.083333	0.333333	0.295225	0.333333	0.449313	0.368303	0.0

In [26]:
```
#remove daynum & day of the month, since those features don't help the
model
#select features
#'weather situation' not one hot encoded because 1-4 is a scale of sev
erity

X = rescaledDF[['year',
            'month',
            'weekday','winter','spring','summer','fall',
             'Feels_Like', 'situation','humidity','w_speed']].values

y = rescaledDF[['trip_count']].values
```

In [27]:
```
#Make the train test split of the data using the train_test_split func
tion.
#changed test size and random state parameters

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
18, random_state=998)


#Create a new Linear Regression model and fit the training data.
model4 = LinearRegression()
model4.fit(X_train,y_train)


#evaluate the model

#MSE:
mean_squared_Error = mean_squared_error(y_test, model4.predict(X_test)
)
print(mean_squared_Error)

#Mean absolute error:
print(mean_absolute_error(y_test, model4.predict(X_test)))

#r2_score value:
print(r2_score(y_test, model4.predict(X_test)))
```

```
0.0073424574580396645
0.06825159028203219
0.8451473030573142
```

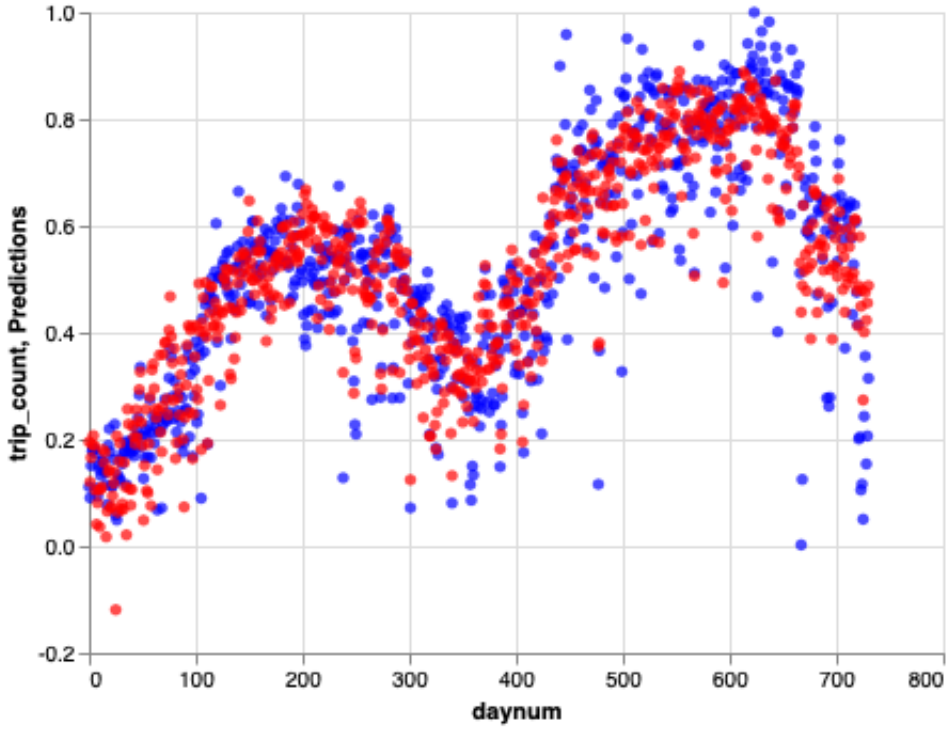
In [28]:
```
#Graph the actual Data (blue) against model's predictions (red)
mergedDF['Predictions'] = model4.predict(X)

#needs y-axis to be be scaled so values are readable...
#x-axis should be fixed to show months/grouped months to help reading

#Graph using altair Chart:

#original & #model4 data
Chart(mergedDF).mark_circle(color='blue').encode(x='daynum', y='trip_count') + \
Chart(mergedDF).mark_circle(color='red').encode(x='daynum', y='Predictions')
```

Out[28]:



Lowest values I was able to achieve?

mean squared error: 0.0073424574580396645 on a 1.00 scale mean absolute error: 0.06825159028203219 on a 1.00 scale

Which weather features improved the score the most? highest r^2 score: 0.8451473030573142

'feels like temp','weather situation', 'humidity', and 'wind speed' improved the score the most. adding others lowered the r^2 score and re-scaling the data helped, but not too much.

Perhaps a linear regression model is not the best for this data.

```
With more years worth of data, a better model could be achieved.
```

end.