bike_sharing_demand

July 19, 2019

[1]:
```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
```

Loading the dataset

[2]:
```
df = pd.read_csv("train.csv", parse_dates=["datetime"])
df.head()
```

[2]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

Cleaning and Optimizing the dataset

[3]:
```
#categorising(?) datetime
df["year"] = df["datetime"].dt.year
df["month"] = df["datetime"].dt.month
df["hour"] = df["datetime"].dt.hour
df["dayofweek"] = df["datetime"].dt.dayofweek
df.head()
```

[3]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	year	month	hour	dayofweek
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16	2011	1	0	5
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40	2011	1	1	5
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32	2011	1	2	5
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13	2011	1	3	5
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1	2011	1	4	5

```
   humidity  windspeed  casual  registered  count  year  month  hour  \
0        81        0.0       3          13     16  2011      1     0
1        80        0.0       8          32     40  2011      1     1
2        80        0.0       5          27     32  2011      1     2
3        75        0.0       3          10     13  2011      1     3
4        75        0.0       0           1      1  2011      1     4

   dayofweek
0          5
1          5
2          5
3          5
4          5
```

[4]:
```
#checking for missing values
df.isnull().values.any()
```

[4]:
```
False
```

[5]:
```
#dropping featrues that are seemingly worthless
df = df.drop(["month",  "datetime", "casual", "registered", "windspeed"], axis=1)
df.head()
```

[5]:
```
   season  holiday  workingday  weather  temp   atemp  humidity  count  year  \
0       1        0           0        1  9.84  14.395        81     16  2011
1       1        0           0        1  9.02  13.635        80     40  2011
2       1        0           0        1  9.02  13.635        80     32  2011
3       1        0           0        1  9.84  14.395        75     13  2011
4       1        0           0        1  9.84  14.395        75      1  2011

   hour  dayofweek
0     0          5
1     1          5
2     2          5
3     3          5
4     4          5
```

[6]:
```
#Categorising Season and Weather features
#season -  1 = spring, 2 = summer, 3 = fall, 4 = winter
#weather - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
#          2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
#          3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light␣
 ↪Rain + Scattered clouds
#          4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
df = pd.get_dummies(df, columns = ['season', 'weather'])
#The dataframe with indicator columns
df.head()
```

[6]:
```
   holiday  workingday  temp   atemp  humidity  count  year  hour  dayofweek  \
0        0           0  9.84  14.395        81     16  2011     0          5
1        0           0  9.02  13.635        80     40  2011     1          5
2        0           0  9.02  13.635        80     32  2011     2          5
3        0           0  9.84  14.395        75     13  2011     3          5
4        0           0  9.84  14.395        75      1  2011     4          5

   season_1  season_2  season_3  season_4  weather_1  weather_2  weather_3  \
0         1         0         0         0          1          0          0
1         1         0         0         0          1          0          0
2         1         0         0         0          1          0          0
3         1         0         0         0          1          0          0
4         1         0         0         0          1          0          0

   weather_4
0          0
1          0
2          0
3          0
4          0
```

[7]:
```
#Splitting and the data
#Training and Testing dataset ready for LinearRegression model
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(df.drop('count', axis=1),␣
 ↪df['count'], test_size = 0.2, random_state = 42)
```

[8]:
```
#Training the LinearRegression Model
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(train_x, train_y)
```

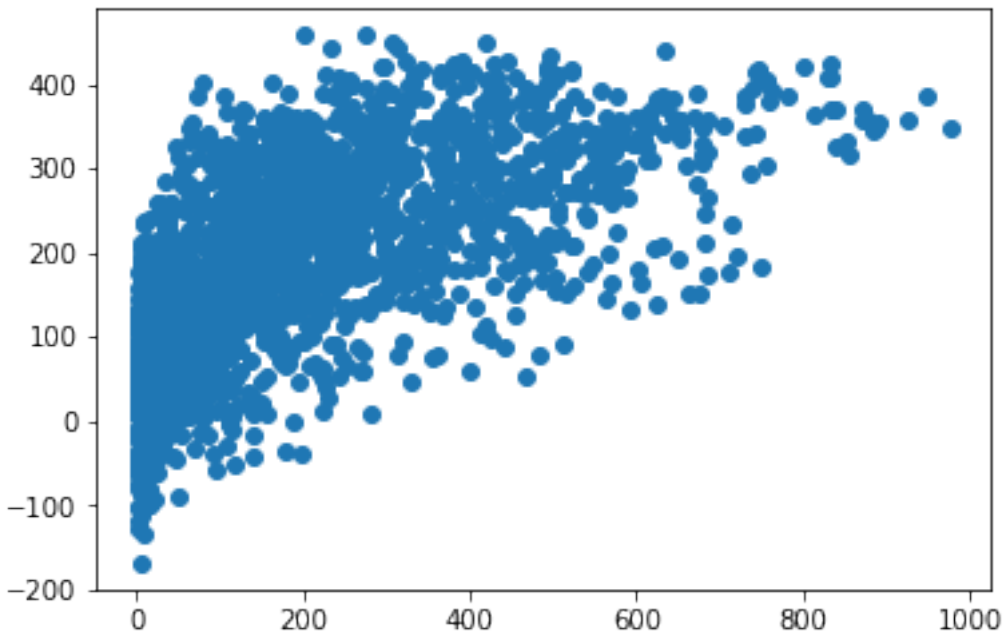
[8]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

[9]:
```
#coefficients showing correlation of dataset
#doesn't seem numerically accurate/interpretable to me
model.coef_
```

[9]:
```
array([ -8.97124488,   3.26963456,   5.72588874,   2.41009541,
        -1.93038813,  80.43790284,   7.78317244,   1.01467576,
       -22.26582547,  -0.13556794, -24.76053346,  47.16192688,
        -7.15810181,  -0.94809345, -35.84092659,  43.94712185])
```

[10]:
```
#Scatterplot of the actual and predicted count values
predicted = model.predict(test_x)
plt.scatter(test_y, predicted)
#It shows more pattern of a log graph.
#I want to study RMSLE and try to make use of it here.
```

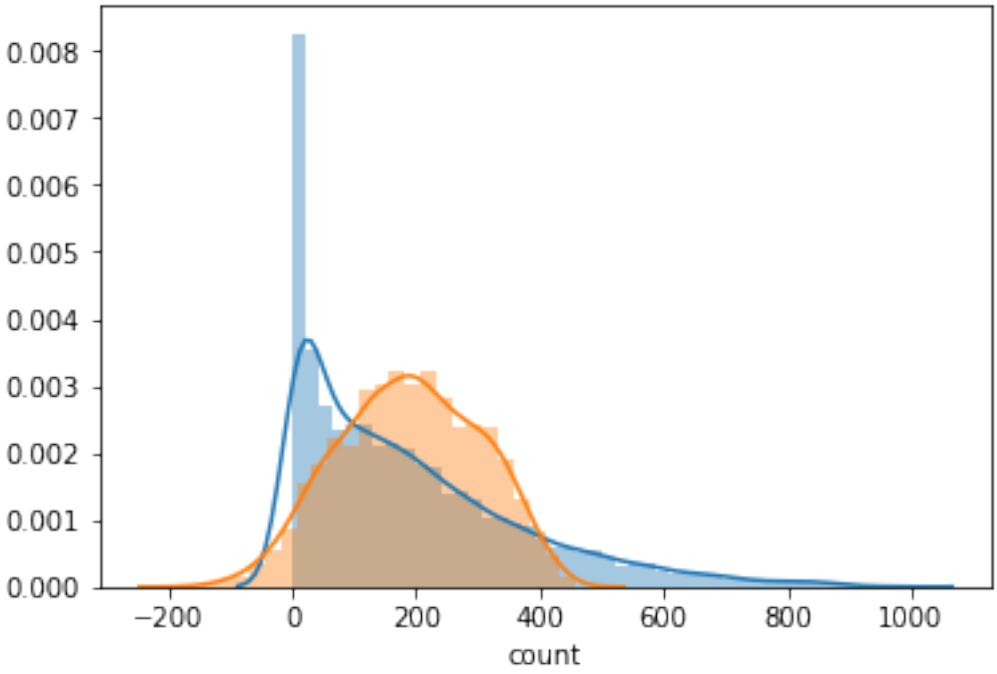
[10]: <matplotlib.collections.PathCollection at 0x7f261e7e8e80>



[11]:
```
#Scoring the model
print(model.score(test_x,test_y))
#Comparing distribution of actual and predicted rental count per hour
sns.distplot(df['count'])
sns.distplot(predicted)

#the score value is very low.

#the comparison of actual data and prediction
#shows very different distribution (skewed vs centred)
```

```
0.3983351154906686

/home/nbuser/anaconda3_501/lib/python3.6/site-
packages/scipy/stats/stats.py:1713: FutureWarning: Using a non-tuple sequence
for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of
`arr[seq]`. In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a different
result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7f261e745e80>



1 LinearRegression

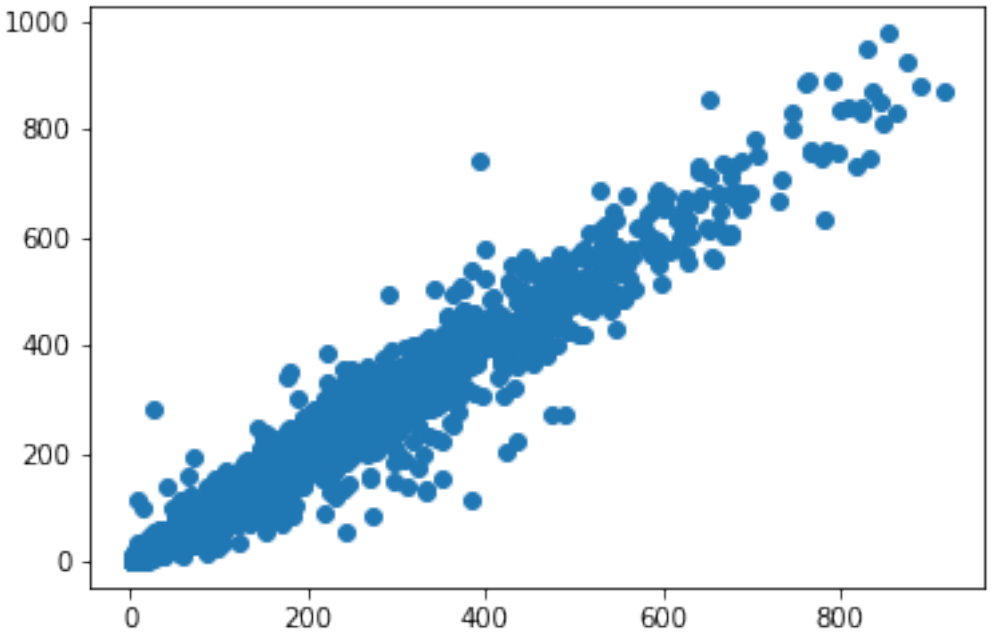
I wanted to make use of all the valuable variables to drive accurate satisfactory prediction and to fulfil the original goal of the project: predicting the total count of bike rented during the given hour. I came up with a conclusion that the Linear Regression was not an ideal Machine Learning model to map all variables from the dataset.

Essemble Model - Random Forest

[12]:
```
#training Random Forest Model with the ready prepared dataset.

from sklearn.ensemble import RandomForestRegressor
rfModel = RandomForestRegressor(n_estimators=100)

rfModel.fit(train_x, train_y)

predicted = rfModel.predict(test_x)

plt.scatter(predicted, test_y)

print(rfModel.score(test_x,test_y) )

#The Score of the model is relatively high.
#I suppose the RandomForest model works well at this point.
```

```
#once again I want to study further into RMSLE model to apply here.
#I see a few outliers in the scatterplot.
#By the property of RMSLE not penalising huge difference values alot,
#I believe I can improve this model.
```

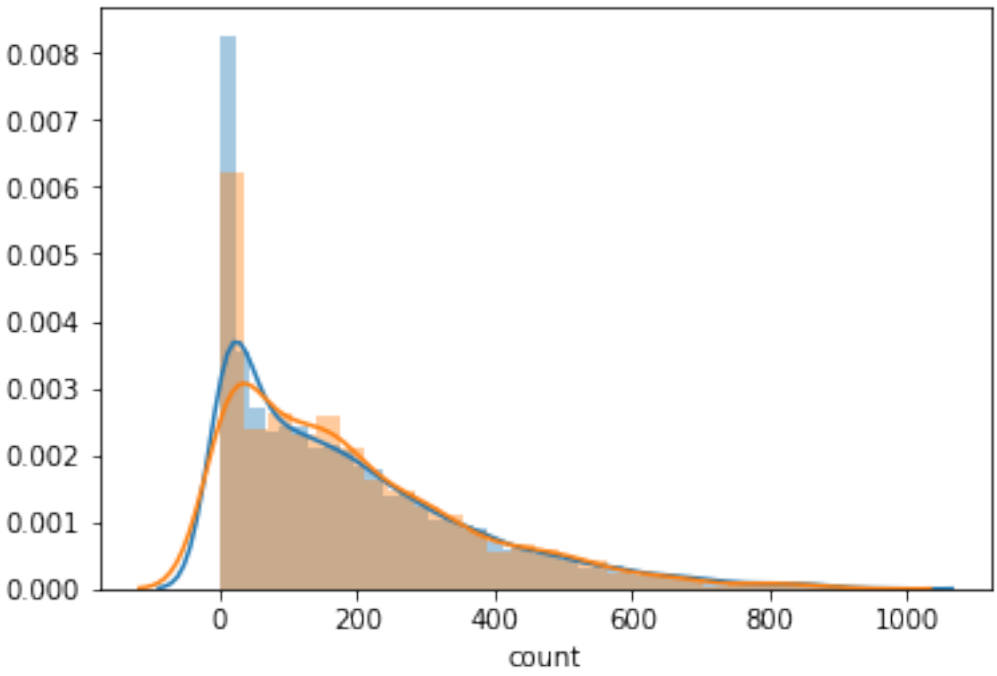
```
0.9512178956757191
```



[13]:
```
#I decided to compare scattorplot of df['count'] and predicted count values

sns.distplot(df['count'])
sns.distplot(predicted)

#Surprising outcome!
#Very Similar distribution with same trend.
```

```
/home/nbuser/anaconda3_501/lib/python3.6/site-
packages/scipy/stats/stats.py:1713: FutureWarning: Using a non-tuple sequence
for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of
`arr[seq]`. In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a different
result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f261e290e80>
```



[14]:
```
#I decided to make a function to predict bike rental demand
#during the given timeframe(1 hour from the given time)

def predict_demand(rental_date_time, season, holiday, workingday, weather, temp,␣
 ↪atemp, humidity):

#rental_date_time - e.g.'2/10/2018 21:45:00'
#season  -  1 = spring, 2 = summer, 3 = fall, 4 = winter
#weather - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
#          2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
#          3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light␣
 ↪Rain + Scattered clouds
#          4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
#holiday    - 1: yes, 0: no
#workingday - 1: yes, 0: neither weekend/holiday
#temp  - temp in celsius
#atemp - "feels like" temp

    from datetime import datetime

    try:
        rental_date_time_parsed = datetime.strptime(rental_date_time, '%d/%m/%Y␣
 ↪%H:%M:%S')
    except ValueError as e:
        return 'Error parsing date/time - {}'.format(e)
```

```
    year = rental_date_time_parsed.year
    month = rental_date_time_parsed.month
    hour = rental_date_time_parsed.hour
    dayofweek = rental_date_time_parsed.isoweekday()



    input = [{'holiday': holiday,
              'workingday': workingday,
              'dayofweek': dayofweek,
              'temp': temp,
              'atemp': atemp,
              'humidity': humidity,
              'year': year,
              'hour': hour,
              'dayofweek': dayofweek,

              'season_1': 1 if season == 1 else 0,
              'season_2': 1 if season == 2 else 0,
              'season_3': 1 if season == 3 else 0,
              'season_4': 1 if season == 4 else 0,

              'weather_1':1 if season == 1 else 0,
              'weather_2':1 if season == 2 else 0,
              'weather_3':1 if season == 3 else 0,
              'weather_4':1 if season == 4 else 0,}]

    return rfModel.predict(pd.DataFrame(input))
```

[15]:
```
#predict_demand(rental_date_time, Season, holiday, workingday, weather, temp,␣
 ↪atemp, humidity)
#given that 1 hour from 10/4/2018 00:00:00 is
#Spring,
#not a holiday,
#workingday,
#Clear weather
#10.66 celsius degree,
#11.365 "feel like" temp in celsius(I suppose it's calculated using wind and sun)
#and humidity is 56%,

bike_sharing_demand_forecast = predict_demand('10/4/2018 00:00:
 ↪00','1','0','1','1','10.66','11.365','56')
print("The predicted Bike Rental Count during the given timeframe␣
 ↪is",bike_sharing_demand_forecast[0],'times.')
```

```
The predicted Bike Rental Count during the timeframe is 13.74 times.
```

The Timeframe is during one hour from the provided timestamp.