CIS 9660 Data Mining

Group Project: Team Dig Data

Group Memebers:

1. Yu Qiao (Tom) Chen YuQiao.Chen@baruchmail.cuny.edu
2. Qinyuan Lu Qinyuan.Lu@baruchmail.cuny.edu
3. Qianqian Song QianQian.Song@baruchmail.cuny.edu
4. Yiran Ye Yiran.Ye@baruchmail.cuny.edu

0. Business Problem ¶

0.1 Identify the Business Problem

Description of Project

Bicycling is an activity which can yield multiple benefits, such as improving riders' health and reducing the air pollution from carbon emissions. Especially during pandemic, to avoid public transportation also improve immune system, more and more people choose bicycles as their major means of transport.

This project is commissioned to use regression analysis methods to identifying how the weather will influence the rental usage of CitiBike during first 6 months in Jersey City. Our ultimate goal is to generate a model to inform the operator of CitiBike bicycle sharing system which days Citi Bike users ride, under what weather conditions. With this model, the operate of bike sharing system can estimate future demand which would enable the system operator to make efficient and expansion plans. To accomplish the goal, we retrieved CitiBike ridership data, joined with daily Jersey City weather data.

Identifying the Business Problem

CitiBike bicycle sharing system went alive in 2013 and being expanding ever since, since pandemic in 2020, the number of riders who are using CitiBike is growing rapidly. To meet the requirement of usage, CitiBike added more docks in Jersey City, so far there are more than 50 CitiBike docks available for sharing. For planning purposes, the system operator needs to project future ridership in order to make good investments.

The DigData Company focuses on data analysis and has a very good reputation in this field, Therefore, CitiBike company hired us conduct a business analysis focusing on the impact of weather on the ridership of CitiBike.

0.2 Data

Inspiration

For our efforts on predicting Citi bike usage, we drew inspiration from the following Seoul Bike Sharing Dataset, where rental amount, date information, and weather conditions were recorded on an hourly basis.

https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand (https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand)

Therefore, we decided to mimick this effort by combining the two following datasets to recreate a similar structure, which we capture hourly weather and ridership counts. We decided to capture **6 months** worth of data from **January 2020** to **June 2020**.

Citi Bike Trips

We used datasets from Citi Bike: Daily Ridership and Membership Data, and Trip Histories.

https://www.citibikenyc.com/system-data (https://www.citibikenyc.com/system-data)

Jersey City Weather

Weather dataset included daily weather summaries for the year of 2020 in Jersey City and have multiple dimensions. We selected the following quantitative dimensions that best represent what bike users would care about the most: Air Temperature, Precipitation, Wind Speed, and Humidity.

http://newa.cornell.edu/index.php?page=hourly-weather (http://newa.cornell.edu/index.php?page=hourly-weather)

Import Modules and Data:

In [1]:
```
import os

import pandas as pd
import numpy as np

from pylab import rcParams

import matplotlib.pyplot as plt
import seaborn as sns

import statsmodels.api as sm
from sklearn.model_selection import train_test_split, cross_val_score, GridSea
rchCV
from sklearn.linear_model import *
from sklearn.kernel_ridge import *
from sklearn.svm import *
from sklearn.metrics import *
from sklearn.naive_bayes import GaussianNB
from dmba import regressionSummary, adjusted_r2_score, plotDecisionTree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
import datetime as datetime
```

```
no display found. Using non-interactive Agg backend
```

In [2]:
```
#read csv
df1=pd.read_csv('JC-202001-citibike-tripdata.csv')
df2=pd.read_csv('JC-202002-citibike-tripdata.csv')
df3=pd.read_csv('JC-202003-citibike-tripdata.csv')
df4=pd.read_csv('JC-202004-citibike-tripdata.csv')
df5=pd.read_csv('JC-202005-citibike-tripdata.csv')
df6=pd.read_csv('JC-202006-citibike-tripdata.csv')
```

```
/Users/luqinyuan/opt/anaconda3/lib/python3.7/site-packages/IPython/core/inter
activeshell.py:3058: DtypeWarning: Columns (16,19,20,21) have mixed types. Sp
ecify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [3]:
```
#combine all the dataset
frames = [df1, df2, df3, df4, df5, df6]
df = pd.concat(frames, sort=False)
```

In [4]:
```
df.head()
```

Out[4]:

	tripduration	Date	Hour	start station id	start station name	start station latitude	start station longitude	end station id	end station name	en statio latitud
0	226	1/1/20	0	3186	Grove St PATH	40.719586	-74.043117	3211	Newark Ave	40.72152
1	377	1/1/20	0	3186	Grove St PATH	40.719586	-74.043117	3269	Brunswick & 6th	40.72601
2	288	1/1/20	0	3186	Grove St PATH	40.719586	-74.043117	3269	Brunswick & 6th	40.72601
3	435	1/1/20	0	3195	Sip Ave	40.730897	-74.063913	3280	Astor Place	40.71928
4	231	1/1/20	0	3186	Grove St PATH	40.719586	-74.043117	3276	Marin Light Rail	40.71458

5 rows × 22 columns

1. Exploratory data analysis & Feature Engineering

1.1 Data Cleaning

1.1.1 Missing Value

In [5]:
```
#count missing vaues in each column
df.isnull().sum()
```

Out[5]:
```
tripduration                0
Date                        0
Hour                        0
start station id            0
start station name          0
start station latitude      0
start station longitude     0
end station id              0
end station name            0
end station latitude        0
end station longitude       0
bikeid                      0
usertype                    0
birth year                  0
gender                      0
Air Temp (?H)              10
Precip (inches)            10
RH (%)                     10
Wind Spd (mph)             10
Wind Dir (degrees)         10
Solar Rad (watts/m2)       10
Dewpoint (?H)              10
dtype: int64
```

In [6]:
```
#drop rows with missing value
df = df.dropna(how = 'any')
df.shape
```

Out[6]:
```
(137957, 22)
```

1.1.2 Data type

In [7]:
```
#check data types
df.dtypes
```

Out[7]:
```
tripduration                  int64
Date                         object
Hour                          int64
start station id              int64
start station name           object
start station latitude      float64
start station longitude     float64
end station id                int64
end station name             object
end station latitude        float64
end station longitude       float64
bikeid                        int64
usertype                     object
birth year                    int64
gender                        int64
Air Temp (?H)               float64
Precip (inches)              object
RH (%)                      float64
Wind Spd (mph)              float64
Wind Dir (degrees)           object
Solar Rad (watts/m2)         object
Dewpoint (?H)                object
dtype: object
```

In [8]:
```
# convert date into Datetime
df['Date'] = pd.to_datetime(df['Date'])
```

In [9]:
```
# We mainly focus on 4 weather factors: Air Temp, Precip, RH, and Wind Spd

# rename the column names
df.rename({'Air Temp (?H)': 'air_temp', \
           'Precip (inches)': 'precip',
           'RH (%)': 'humidity',
           'Wind Spd (mph)': 'wind_spd'}, axis=1, inplace=True)

# drop unrelated columns
df.drop('Wind Dir (degrees)', axis= 1, inplace= True)
df.drop('Solar Rad (watts/m2)', axis= 1, inplace= True)
df.drop('Dewpoint (?H)', axis= 1, inplace= True)
```

In [10]:
```
# check unique values
df['precip'].unique()
```

Out[10]:
```
array([0.0, 0.01, 0.02, 0.05, 0.03, 0.04, 0.06, 0.16, 0.47, 0.07, 0.08,
       0.19, 0.18, 0.14, 0.12, 0.27, 0.26, 0.09, 0.22, 0.23, 0.1, 0.11,
       0.21, 0.15, 0.17, '0', '0.04', '0.34', '0.22', '0.01', '0.09',
       '0.11', '0.15', '0.1', '0.02', '-', '0.06', '0.03', '0.17', 0.13],
      dtype=object)
```

In [11]:
```
# drop records with "-" value in "precip"
df = df[df.precip != '-']
# change the "Precip" data type to float
df['precip'] = df['precip'].astype(float)
```

In [12]:
```
#check data types
df.dtypes
```

Out[12]:
```
tripduration                        int64
Date                       datetime64[ns]
Hour                                int64
start station id                    int64
start station name                 object
start station latitude            float64
start station longitude           float64
end station id                      int64
end station name                   object
end station latitude              float64
end station longitude             float64
bikeid                              int64
usertype                           object
birth year                          int64
gender                              int64
air_temp                          float64
precip                            float64
humidity                          float64
wind_spd                          float64
dtype: object
```

1.1.3 group by Month,Week, Weekday, Hour to get the target variable: rented_bike_count and analyze the trend in related graphs

We chose “rented _bike_count” as the target value because when the operators of CitiBike sharing system need to project future plans of bike distribution,they will be more concerned under specific weather conditions, the ridership of Citi Bike at various times.

In [13]:
```
#Look at the distribution of subscribers and one-time-customer, so we need to
 use membership program to convert more one-time users to subscribers

df.usertype.value_counts()
```

Out[13]:
```
Subscriber    100529
Customer       37403
Name: usertype, dtype: int64
```

In [14]:
```
# drop unrelated columns:
df.drop(['tripduration','start station name','start station latitude','start s
tation longitude',
        'end station id','end station name','end station latitude','end statio
n longitude','gender', 'usertype'], axis = 1, inplace= True)
```

In [15]:
```
df.head()
```

Out[15]:

	Date	Hour	start station id	bikeid	birth year	air_temp	precip	humidity	wind_spd
10	2020-01-01	1	3186	29467	1976	40.1	0.0	62.0	7.1
11	2020-01-01	1	3185	29290	1984	40.1	0.0	62.0	7.1
12	2020-01-01	1	3792	26155	1963	40.1	0.0	62.0	7.1
13	2020-01-01	1	3792	29673	1964	40.1	0.0	62.0	7.1
14	2020-01-01	1	3275	26250	1990	40.1	0.0	62.0	7.1

In [16]:
```
#group by date and hour to get the target variable: "rented_bike_count"
df1 = df.groupby(["Date","Hour","air_temp","precip","humidity","wind_spd"])["H
our"].count().reset_index(name='rented_bike_count')
df1.head()
```

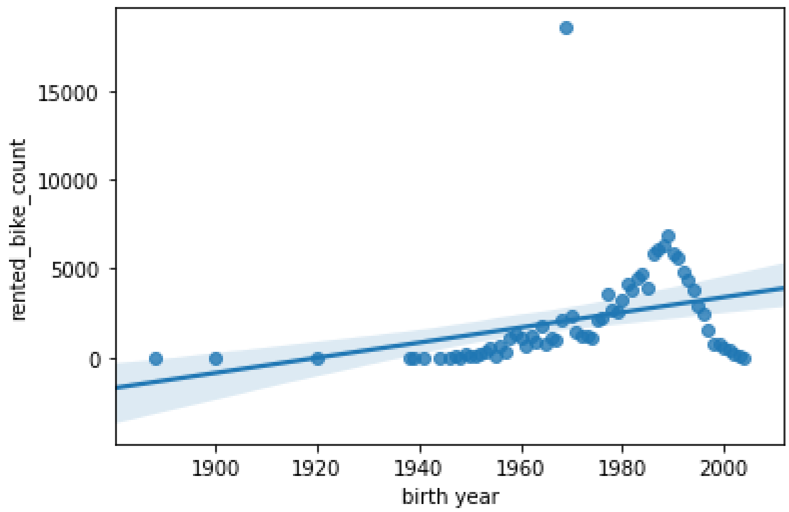
Out[16]:

	Date	Hour	air_temp	precip	humidity	wind_spd	rented_bike_count
0	2020-01-01	1	40.1	0.0	62.0	7.1	14
1	2020-01-01	2	39.7	0.0	65.0	2.0	9
2	2020-01-01	3	39.4	0.0	63.0	0.1	6
3	2020-01-01	4	38.8	0.0	59.0	0.1	2
4	2020-01-01	5	38.3	0.0	56.0	5.8	2

In [17]:
```
df['Date'] = pd.to_datetime(df['Date'])
#obtain the Month, Week, Weekdays from date
df['Month'] = df['Date'].dt.month
df['Week'] = df['Date'].dt.week
df["Weekday"] = df["Date"].dt.weekday
```

In [18]:
```
#To find out relationship between age and rented bike amount and find that 90s is the group that prefers using shared bike
Age_graph = df.groupby(["birth year"])["bikeid"].count().reset_index(name='rented_bike_count')
sns.regplot(x="birth year",y="rented_bike_count",data=Age_graph)
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1f9a7b10>



Interpretation

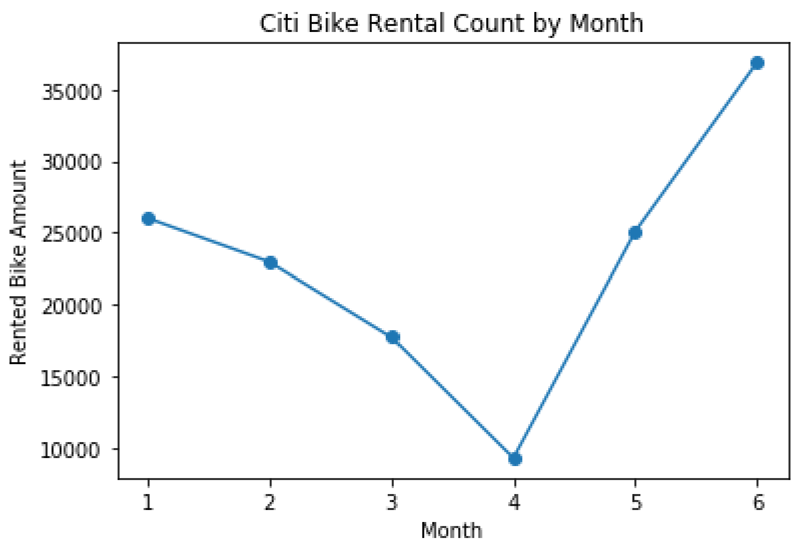
This graph above shows how much bikes are rented by rider's birth year. For example, a point in the graph will show the total bike rentals of the rider born from a certain year. We observe a "spike" pattern from the distrbution of age groups, and draw the following conclusions:

- Middle age population (30-40) are the biggest group of users of Citi Bike.
- Newer generation people (1990 onwards) are not interested in biking, and the count becomes less for newer generations.

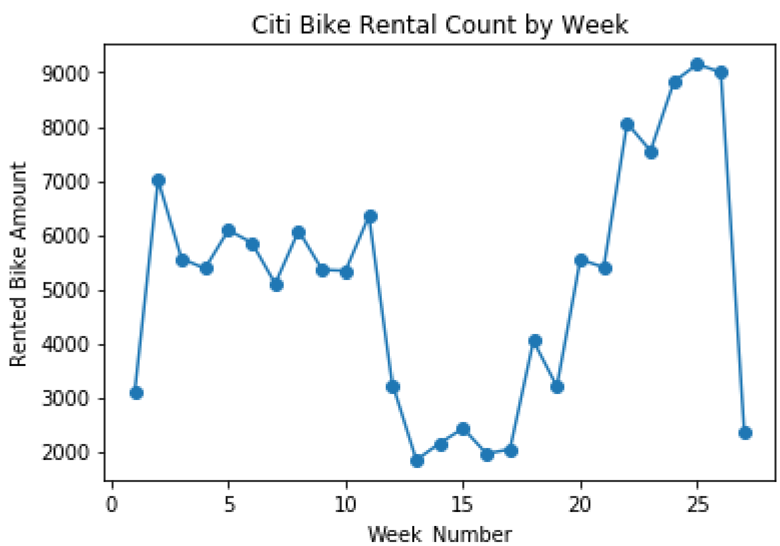
Suggestion

Although this graph can't help in predicting how many bike are needed to prepare, Citi Bike can definitely benefit from this by developing market strategies to attract younger demographics into using Citi Bikes

In [19]:
```
Month_graph = df.groupby(["Month"])["bikeid"].count().reset_index(name='rented
_bike_count')
Month_graph.head()

fig,ax=plt.subplots()
ax.plot(Month_graph["Month"], Month_graph['rented_bike_count'],marker="o")

ax.set_title('Citi Bike Rental Count by Month')
ax.set_xlabel('Month')
ax.set_ylabel('Rented Bike Amount')
plt.show()
```



In [20]:
```
#find out the week trend of rented bike, intiially, the rented bike amount was negatively impacted by Covid-19 but later it rebounded back

Week_graph = df.groupby(["Week"])["bikeid"].count().reset_index(name='rented_bike_count')


fig,ax=plt.subplots()
ax.plot(Week_graph["Week"], Week_graph['rented_bike_count'],marker="o")

ax.set_title('Citi Bike Rental Count by Week')
ax.set_xlabel('Week_Number')
ax.set_ylabel('Rented Bike Amount')
plt.show()
```



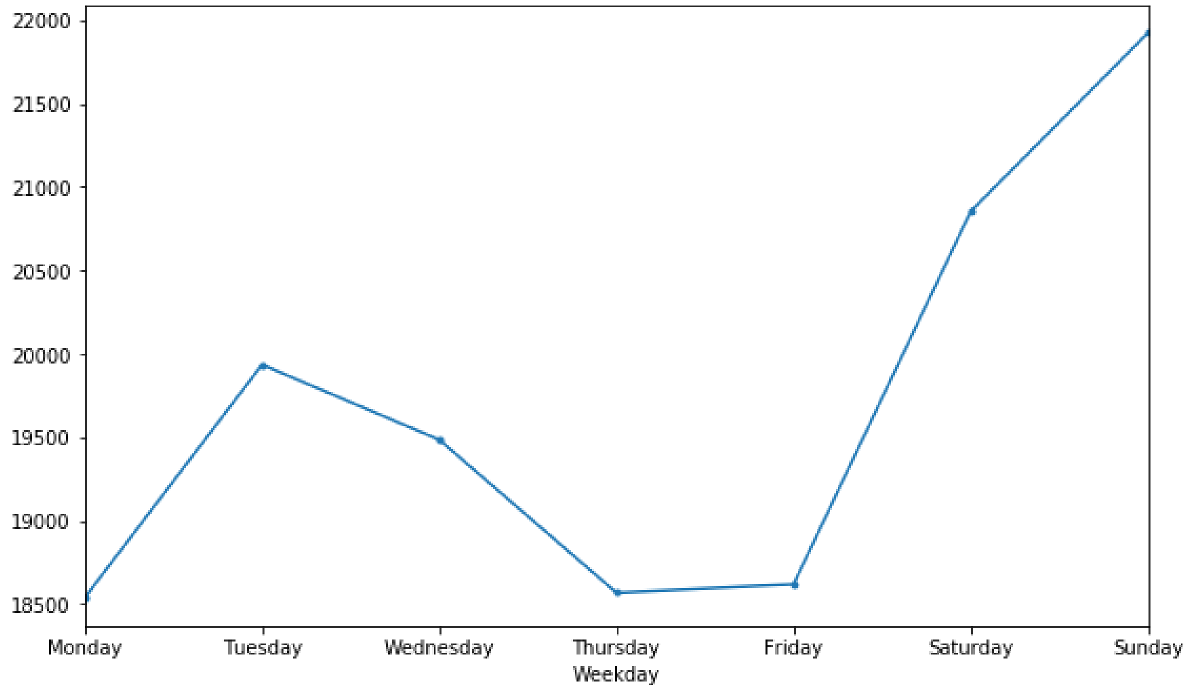
Interpretation

This two line graphs above shows the total Citi Bike rental counts per **month** and **week** from Jan 2020 to June 2020. We observe an interesting pattern here with the counts. Citi Bike was losing riders steadily from January to April as the pandemic occurred. Citi Bike lost **more than half** of its ridership at April when the pandemic reached its peak. However, when the pandemic alleviated in the summer, rider counts rebounded to a higher number than it started in January of 2020, more than **3 times** the count in April. This is likely due to people seeing biking as a safer travel option to commute.

Suggestion

For the future, we may suggest to Citi Bike to monitor the pandemic closely to anticipate more bike demands due to this pattern. If another wave of pandemic were to occur, the usage of bikes will likely be expected to decrease, and at the same time, demands will rise post-pandemic.

In [21]:
```
Weekday_graph = df.groupby('Weekday').size().plot(figsize = (10,6), marker =
'.')
Weekday_graph.set_xticklabels(["Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday"])
ax.set_title('Citi Bike Rental Count by Days of a Week')
ax.set_xlabel('Weekday')
ax.set_ylabel('Rented Bike Amount')
plt.show()
```



Interpretation

This line graph above shows the total summed bike usage of each day of the week. We can observe from this graph that Saturdays and Sundays are popular days for Citi Bike rentals. This is an expected pattern because people would go out to excercise or spend time outdoor more on weekends. During weekdays, there is an interesting pattern which rentals on Mondays, Thursdays, and Fridays are lower than the rental amounts on Tuesday and Wednesdays.

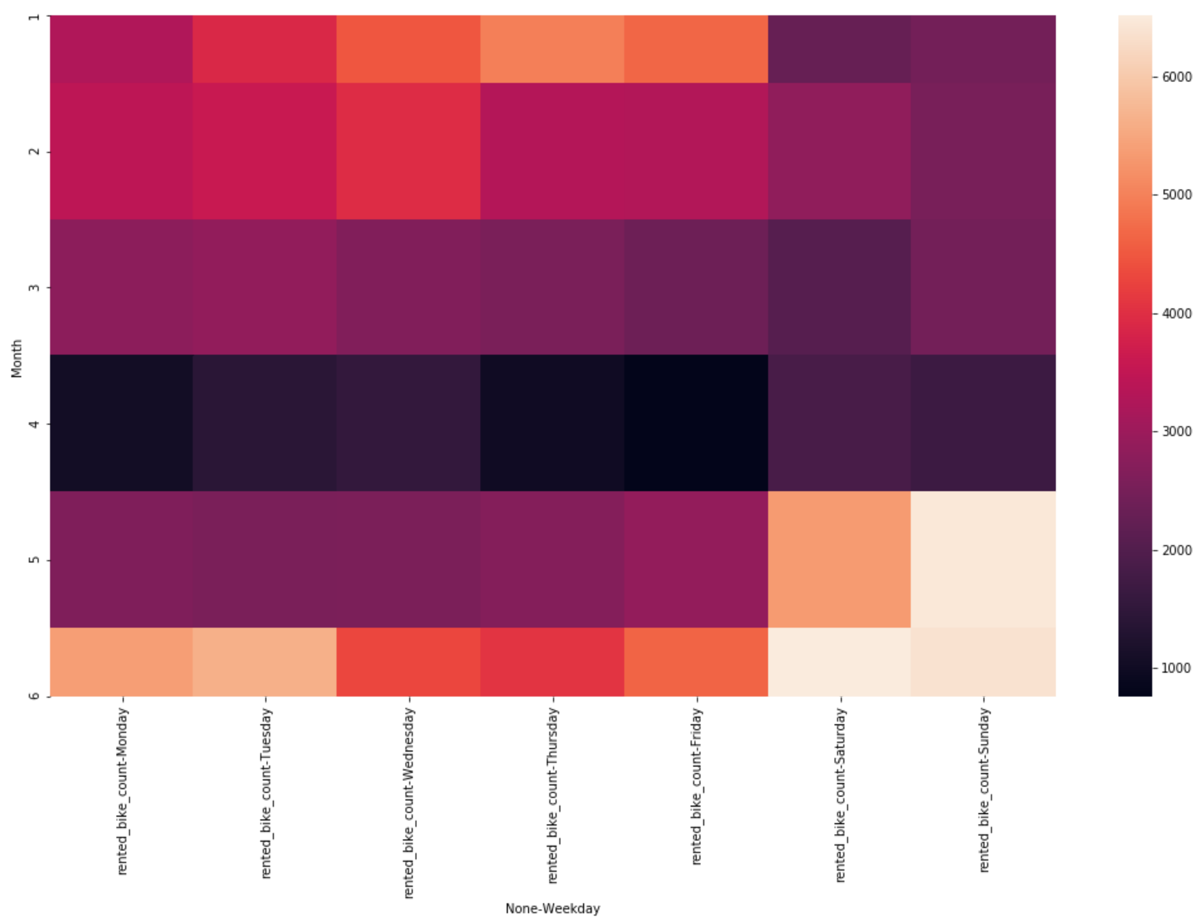
Suggestion

Fot the future, we may suggest Citi Bike to provide more bikes on weekends to meet the higher demands. We would also suggest preparing more bikes on Tuesdays and Wednesdays, too.

In [22]:
```
#turn hour into dummies
df2 = pd.get_dummies(data=df1, drop_first = True, columns=['Hour'])
```

In [23]:
```
heat = df2

heat['Month'] = df2['Date'].dt.month
heat['Weekday'] = df2['Date'].dt.weekday
heat = heat[["rented_bike_count", "Month", "Weekday"]]

heat2 = heat.pivot_table(index = 'Month', columns = 'Weekday', aggfunc = np.su
m)
heat2 = heat2.rename(columns = {0 : "Monday",
                        1 : "Tuesday",
                        2 : "Wednesday",
                        3 : "Thursday",
                        4 : "Friday",
                        5 : "Saturday",
                        6 : "Sunday"})

fig, ax = plt.subplots()
fig.set_size_inches(18, 10)

sns.heatmap(heat2)
```

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1c0c30d0>



Interpretation

This heat map shows the total summed amount of rentals by Month and day of the week. The darker the color means a lower number of rentals. The brighter the color means a higher number of rentals. As we can observe, there is a "trench" in the month of April when the Citi Bike ridership is significantly lower than the other Months. This is during the time when COVID19 pandemic was at its peak. Furthermore, we can also observe within a month, which day of a week was there more riders. And the results revealed that prior to COVID19, bikes were mostly rented on weekdays. However, after the pandemic surge in April, we noticed a rebound in bike rentals in May and June. Saturdays and Sunday became the new popular days for bike rides instead of weekdays.

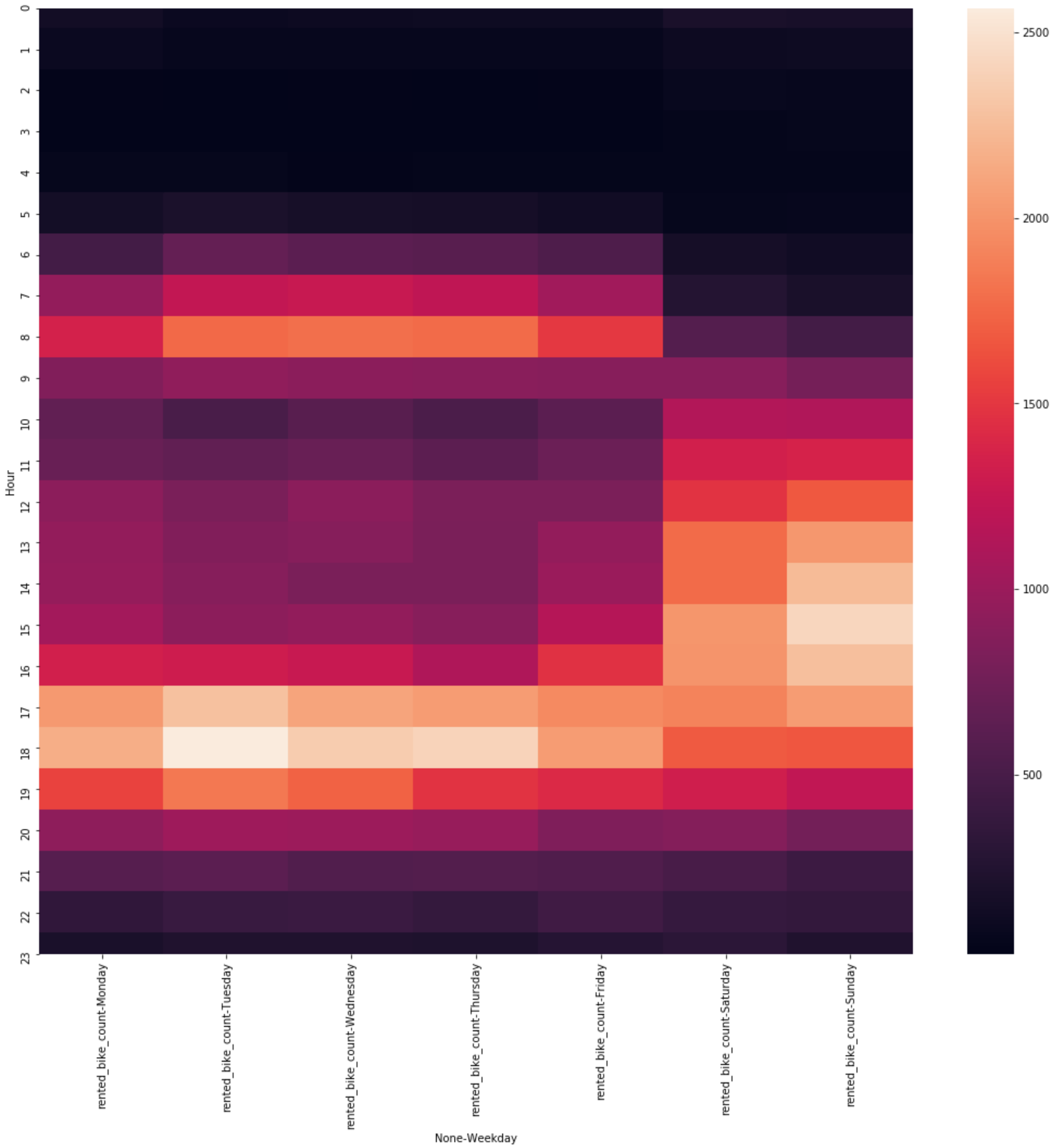
Suggestions

This heat map suggests that people are discovering biking as a more feasible activity during COVID19 when long distance travelings aren't suggested. We recommend Citi Bike to prepare sufficient bikes on weekends to meet higher demands.

In [24]:

```
heat3 = df1
heat3['Weekday'] = heat3['Date'].dt.weekday
heat3 = heat3[['Weekday', 'Hour', 'rented_bike_count']]

heat4 = heat3.pivot_table(index = 'Hour', columns = 'Weekday', aggfunc = np.su
m)
heat4 = heat4.rename(columns = {0 : "Monday",
                        1 : "Tuesday",
                        2 : "Wednesday",
                        3 : "Thursday",
                        4 : "Friday",
                        5 : "Saturday",
                        6 : "Sunday"})

fig, ax = plt.subplots()
fig.set_size_inches(18, 16)

sns.heatmap(heat4)
```

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1bfa5c10>



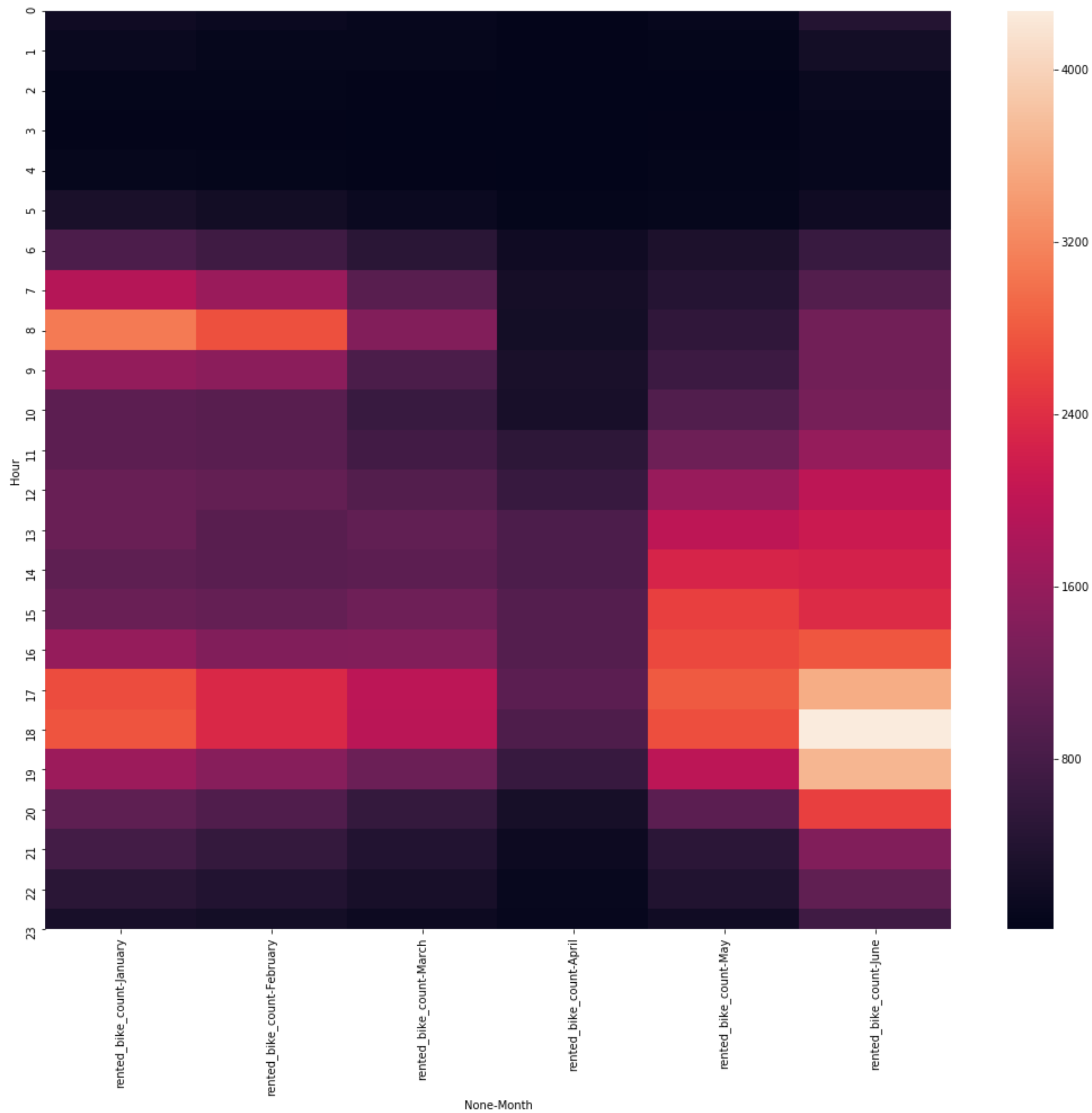
Interpretation

This heat map shows highlights the hour of a day, from Monday to Sunday, when more bikes are being rented. The brighter the color indicates more Citi Bike ridership. From this heatmap, we can draw several conclusions:

- There are more Citi Bike rentals during rush hours on weekdays. Those are times around morning 8 AM and afternoon 6 PM. Ridership in between these times, work hours are lower. Bikers during this time are likely to be commuters to work.
- This pattern inverses during weekends, usage of Citi Bike during Saturdays and Sundays are low during rush hours, but are higher in the middle of the day, particularly at 2 PM.
- We can suggest to Citi Bike to focus on bike supplies during two time periods: 1) **Rush Hours during weekdays** and 2) **Afternoon hours during weekends**.

In [25]:
```
heat5 = df1
heat5['Month'] = heat5['Date'].dt.month
heat5 = heat5[['Month', 'Hour', 'rented_bike_count']]

heat6 = heat5.pivot_table(index = 'Hour', columns = 'Month', aggfunc = np.sum)
heat6 = heat6.rename(columns = {1 :'January',
                                2 : 'February',
                                3 : 'March',
                                4 : 'April',
                                5 : 'May',
                                6 : 'June'})


fig, ax = plt.subplots()
fig.set_size_inches(18, 15)

sns.heatmap(heat6)
```

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1ea98a90>



Interpretation

This heat map is similar to the first heat map with month and weekday. Instead, we have month on the x-axis, and hour on the y-axis. We observe a very interest change in the patterns of bike rentals. Likewise, we observe April to be the dividing month for pre-COVID months (Jan -Mar) and post-COVID months (May & June).

- Before COVID, bikes were frequently rented during rush hours in the morning and afternoon.
- However in post-COVID months, more bikes are being rented after 12 PM in the afternoon, and 6 PM are usually the times when the rental reaches its peak.
- The shift in change can be attributed to less people are now commuting to work.

Suggestion

We would recommend Citi Bike to prepare more bikes in the afternoon move foward after June.

In [26]:
```
df2.head()
```

Out[26]:

	Date	air_temp	precip	humidity	wind_spd	rented_bike_count	Hour_1	Hour_2	Hour_3	Hc
0	2020-01-01	40.1	0.0	62.0	7.1	14	1	0	0	
1	2020-01-01	39.7	0.0	65.0	2.0	9	0	1	0	
2	2020-01-01	39.4	0.0	63.0	0.1	6	0	0	1	
3	2020-01-01	38.8	0.0	59.0	0.1	2	0	0	0	
4	2020-01-01	38.3	0.0	56.0	5.8	2	0	0	0	

5 rows × 31 columns

In [27]:
```
#obtain the weekdays from date
df2["Weekday"] = df2["Date"].dt.weekday
```

In [28]:
```
# 0 - Mon
# 1 - Tues
# 2 - Wed
# 3 - Thurs
# 4 - Fri
# 5 - Sat
# 6 - Sun

df2.head()
```

Out[28]:

	Date	air_temp	precip	humidity	wind_spd	rented_bike_count	Hour_1	Hour_2	Hour_3	Hc
0	2020-01-01	40.1	0.0	62.0	7.1	14	1	0	0	
1	2020-01-01	39.7	0.0	65.0	2.0	9	0	1	0	
2	2020-01-01	39.4	0.0	63.0	0.1	6	0	0	1	
3	2020-01-01	38.8	0.0	59.0	0.1	2	0	0	0	
4	2020-01-01	38.3	0.0	56.0	5.8	2	0	0	0	

5 rows × 31 columns

In [29]:
```
# turn weekday into dummies
df2 = pd.get_dummies(data=df2, columns=['Weekday'])
df2.head()
```

Out[29]:

	Date	air_temp	precip	humidity	wind_spd	rented_bike_count	Hour_1	Hour_2	Hour_3	Hc
0	2020-01-01	40.1	0.0	62.0	7.1	14	1	0	0	
1	2020-01-01	39.7	0.0	65.0	2.0	9	0	1	0	
2	2020-01-01	39.4	0.0	63.0	0.1	6	0	0	1	
3	2020-01-01	38.8	0.0	59.0	0.1	2	0	0	0	
4	2020-01-01	38.3	0.0	56.0	5.8	2	0	0	0	

5 rows × 37 columns

In [30]:
```
# Month Dummies

df2['Month'] = df2['Date'].dt.month
```

In [31]:
```
df2 = pd.get_dummies(data = df2, columns = ['Month'])
df2.head()
```

Out[31]:

	Date	air_temp	precip	humidity	wind_spd	rented_bike_count	Hour_1	Hour_2	Hour_3	Hc
0	2020-01-01	40.1	0.0	62.0	7.1	14	1	0	0	
1	2020-01-01	39.7	0.0	65.0	2.0	9	0	1	0	
2	2020-01-01	39.4	0.0	63.0	0.1	6	0	0	1	
3	2020-01-01	38.8	0.0	59.0	0.1	2	0	0	0	
4	2020-01-01	38.3	0.0	56.0	5.8	2	0	0	0	

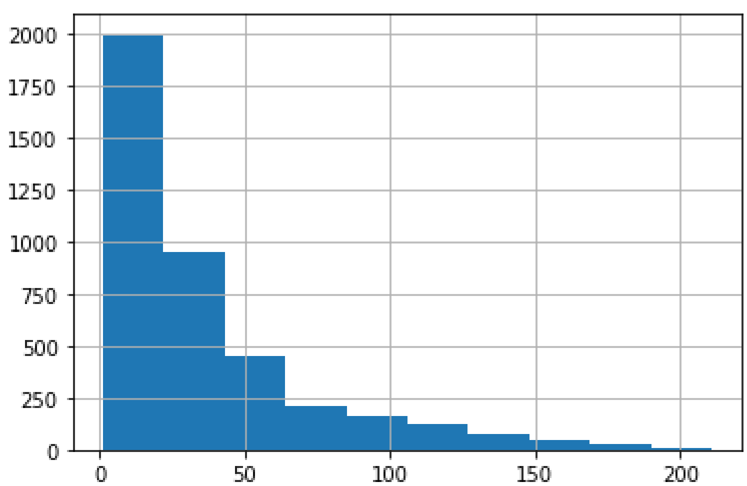
5 rows × 42 columns

1.2 Frequency of the target variable

Distribution of Bike Rental

In [32]:
```
df2['rented_bike_count'].hist()
```

Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1f734790>

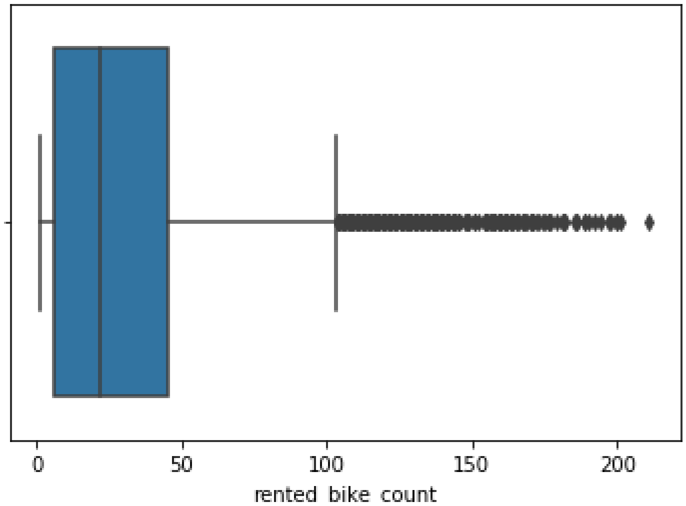


The histogram plot shows the frequency of rented bike count at each hour, and most frequent number is below 50. The distribution is skewed to the right. This result which was within our expectation for Jersey City area.

Distribution of Bike Rental by Hour

Outlier

In [33]:
```
ax = sns.boxplot(x=df2["rented_bike_count"])
```



Boxplot displays the position of the median value of rented_bike_count, which is around 20.

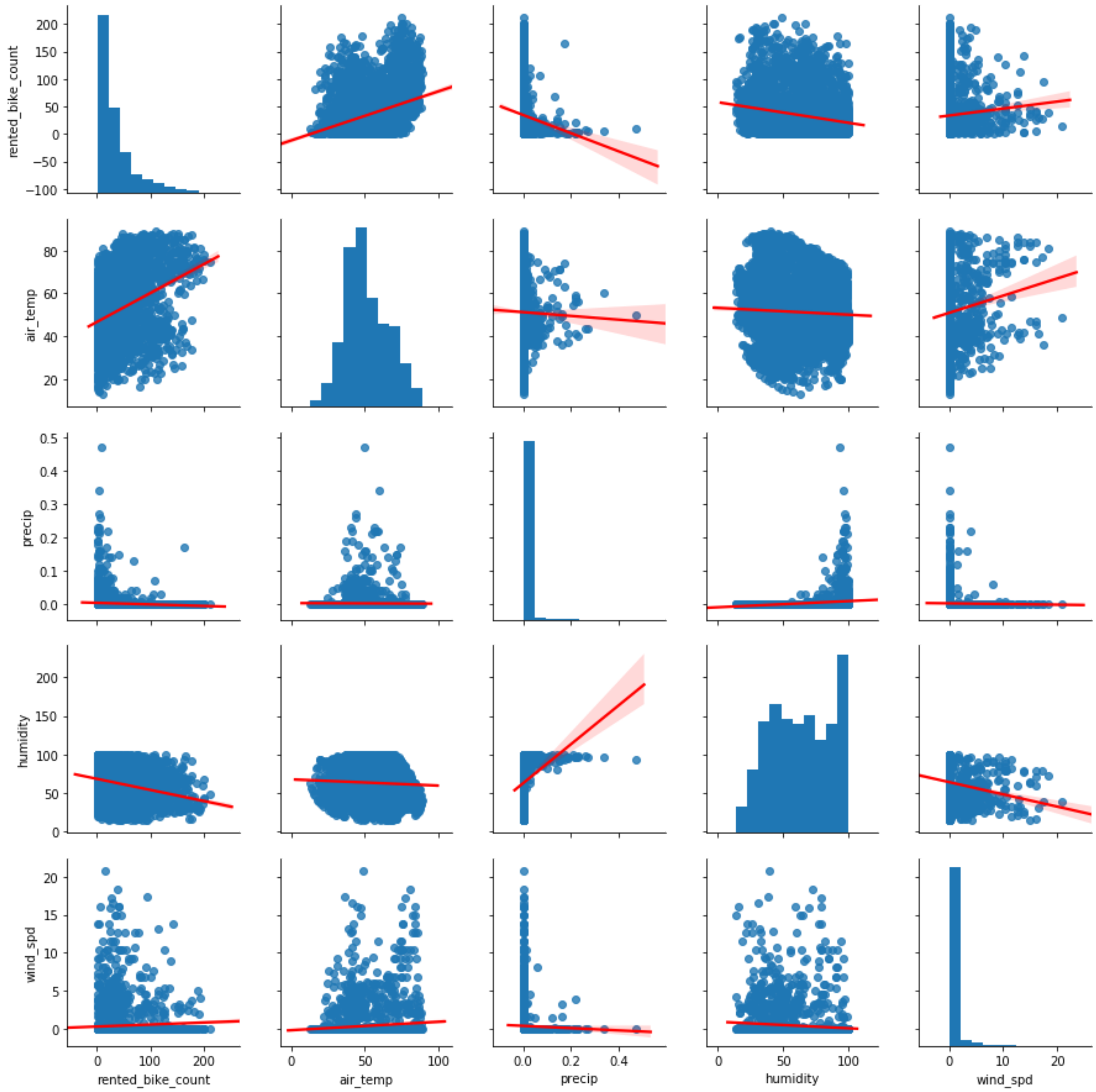
There is a long tail in the data distribution.

And, there are many outliers displayed above the upper whisker that are marked with circles.

1.3 Relationship between variables

In [34]:
```
sns.pairplot(df2[['rented_bike_count', 'air_temp', 'precip', 'humidity', 'wind
_spd']],
             kind = 'reg',
             plot_kws={'line_kws':{'color':'red'}})
```

Out[34]: <seaborn.axisgrid.PairGrid at 0x1c1bf9fbd0>



This pairplot is meant for the team to observe any relationships among the variables. A best fit line is included to indicate the relationship between the two variables.

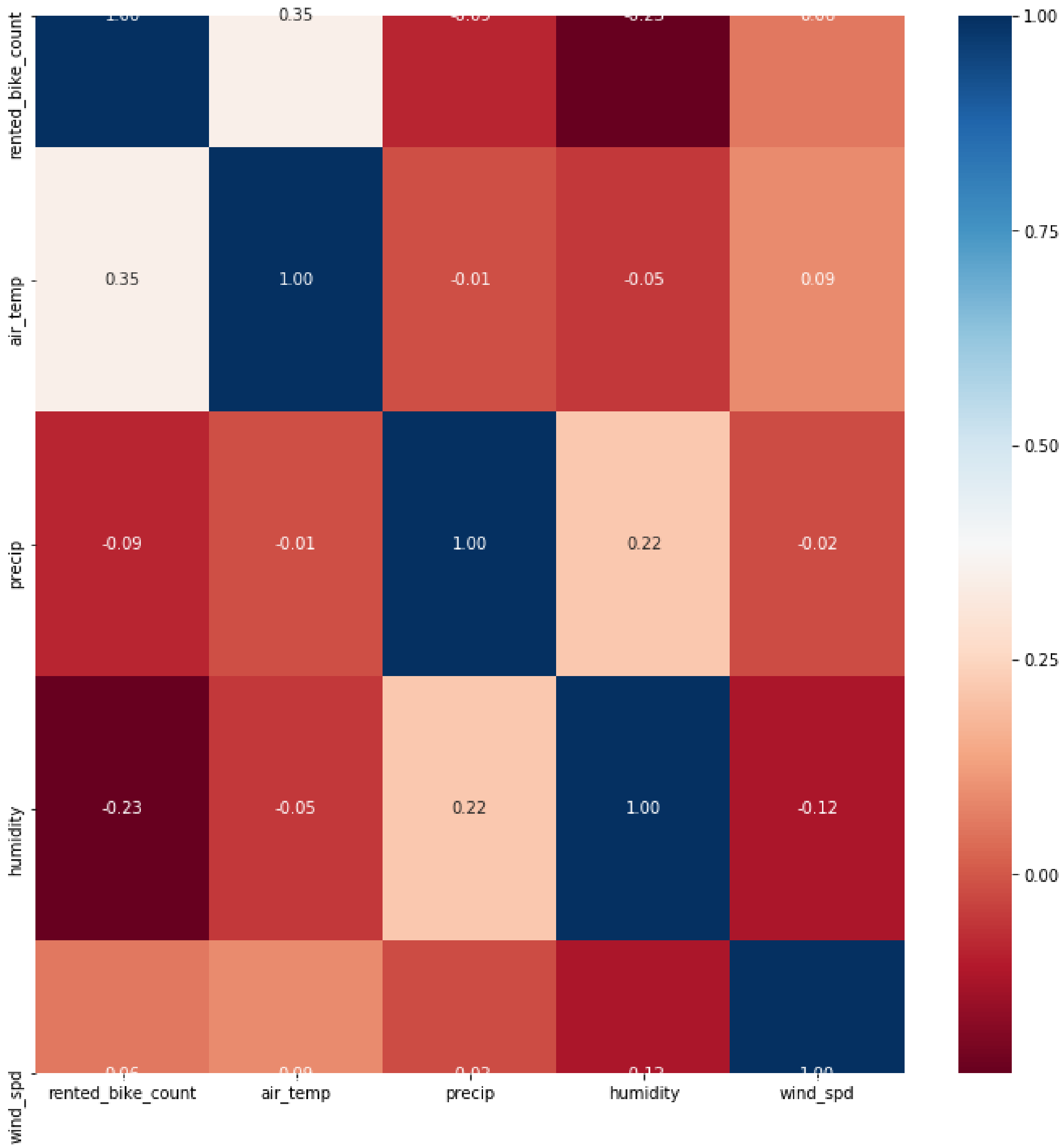
- A positive relationship is observed between rented bike count and tempeuration.
- A positive relationship is observed between precipitation and humidity.
- A negative relationship is observed between precipitation and rented bike count.

1.3.1 correlation matrix

In [35]:
```
corrMatrix = df1[['rented_bike_count', 'air_temp', 'precip', 'humidity', 'wind
_spd']].corr()
print (corrMatrix)
```

```
                   rented_bike_count  air_temp    precip  humidity  wind_spd
rented_bike_count           1.000000  0.347552 -0.087462 -0.232072  0.057973
air_temp                    0.347552  1.000000 -0.012018 -0.050992  0.094466
precip                     -0.087462 -0.012018  1.000000  0.215798 -0.017339
humidity                   -0.232072 -0.050992  0.215798  1.000000 -0.117016
wind_spd                    0.057973  0.094466 -0.017339 -0.117016  1.000000
```

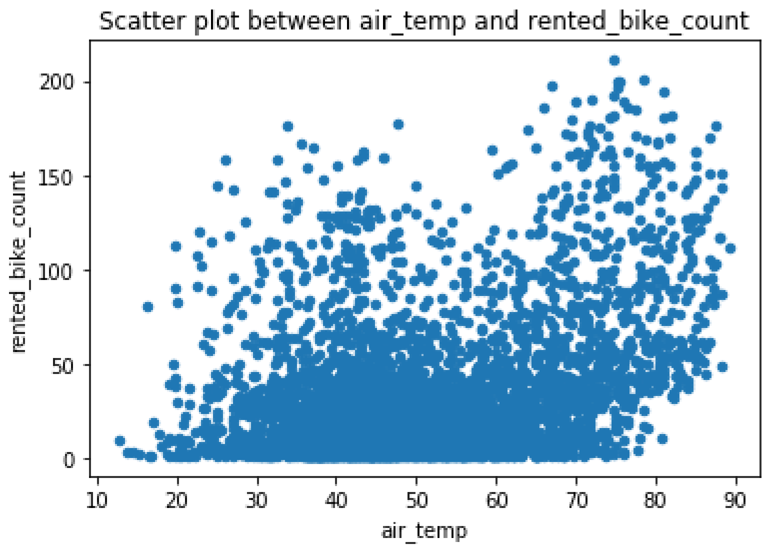
In [36]:
```
fig, ax = plt.subplots()
fig.set_size_inches(12, 12)
sns.heatmap(corrMatrix, annot=True, fmt=".2f", cmap="RdBu", ax=ax)
plt.show()
```



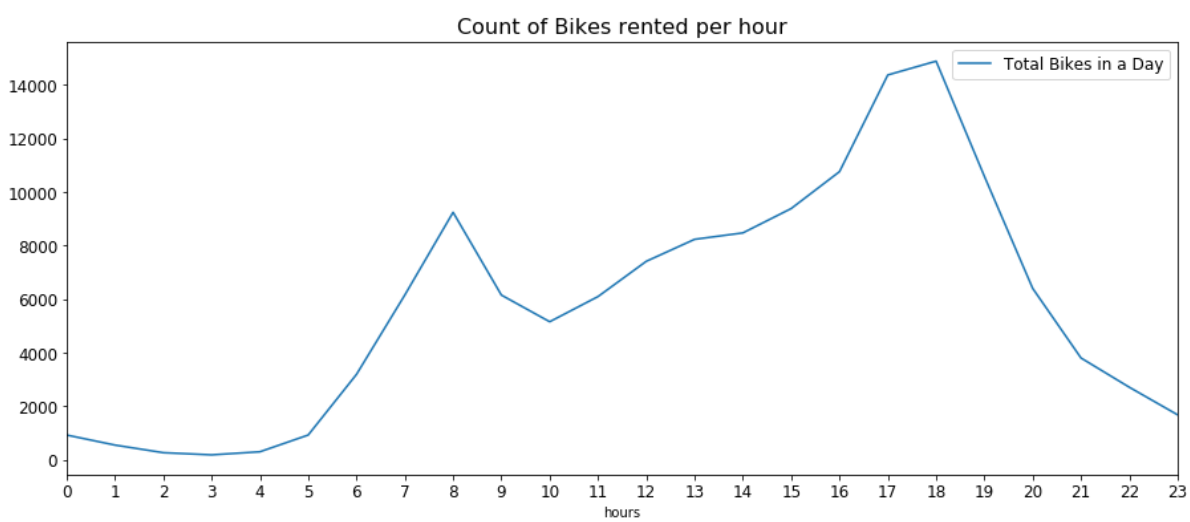
1.3.2 scatterplots between important variables

In [37]:
```
# Scatter plot between air_temp and rented_bike_count
df2.plot.scatter(x='air_temp', y='rented_bike_count', title= "Scatter plot bet
ween air_temp and rented_bike_count")
```

Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x1c227f1d50>



In [38]:
```
# Plotting - Bikes per Day
plt.figure(figsize=(15,6))

df.groupby('Hour').size().plot(label = 'Total Bikes in a Day')
plt.title('Count of Bikes rented per hour', fontsize=16)
plt.xlabel('hours')
plt.xticks(np.arange(0,24))
plt.legend(prop={'size':12})
plt.tick_params(labelsize=12)
```



The 'number of trips' on the road is highest at 8 AM & between 5 - 6 PM which are rush hours

The 'number of trips' on the road is at its lowest around 3 - 4 AM when everybody is at home asleep.

2. Create a baseline model

Define X and y

In [39]:
```
# define X and y set
X = df2.drop(columns = ["Date",
                        "rented_bike_count"
                       ])
y = df2['rented_bike_count']
```

Fit into training and testing sets

In [40]:
```
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size = 0.3, r
andom_state = 1)
print('Traning set:', train_X.shape, 'Validation set:', valid_X.shape)
```

```
Traning set: (2836, 40) Validation set: (1216, 40)
```

Prediction Results

In [41]:
```
def pred_results(model, valid_X, valid_y):
    pred = model.predict(valid_X)

    result = pd.DataFrame({'Predicted': pred, 'Actual': valid_y, 'Residual': v
alid_y - pred})
    return result
```

Regression Summaries

In [42]:
```
def regressionReport(model, valid_X, valid_y):
    pred = model.predict(valid_X)
    regressionSummary(valid_y, pred)
    print(f"\n R square : {r2_score(valid_y, pred)}")
```

Linear Regression

In [43]:
```
lr = LinearRegression()
lr.fit(train_X, train_y)
```

Out[43]:
```
LinearRegression()
```

In [44]:
```
print('Intercept', lr.intercept_)
print(pd.DataFrame({'Predictor': X.columns, 'Coefficient': lr.coef_}))
```

```
Intercept -29.325762161444892
    Predictor  Coefficient
0    air_temp     0.944084
1      precip  -103.966477
2    humidity    -0.162074
3    wind_spd    -1.141940
4      Hour_1    -2.059208
5      Hour_2    -3.963304
6      Hour_3    -3.729258
7      Hour_4    -1.899462
8      Hour_5     4.413634
9      Hour_6    17.790684
10     Hour_7    34.141320
11     Hour_8    48.485715
12     Hour_9    27.901127
13    Hour_10    21.558796
14    Hour_11    23.783334
15    Hour_12    28.668301
16    Hour_13    35.165610
17    Hour_14    29.872302
18    Hour_15    37.177706
19    Hour_16    44.084140
20    Hour_17    65.154394
21    Hour_18    69.412436
22    Hour_19    46.413352
23    Hour_20    26.056892
24    Hour_21    12.313790
25    Hour_22     7.451958
26    Hour_23     3.615741
27  Weekday_0    -2.640411
28  Weekday_1    -0.786898
29  Weekday_2    -0.248778
30  Weekday_3     0.345379
31  Weekday_4    -2.016831
32  Weekday_5     1.281186
33  Weekday_6     4.066353
34    Month_1    16.908825
35    Month_2    13.342479
36    Month_3    -4.419598
37    Month_4   -20.694833
38    Month_5    -6.222084
39    Month_6     1.085210
```

In [45]:
```
regressionReport(lr, valid_X, valid_y)
```

```
Regression statistics

                      Mean Error (ME) : -0.4255
       Root Mean Squared Error (RMSE) : 25.5642
            Mean Absolute Error (MAE) : 17.5779
          Mean Percentage Error (MPE) : -30.4648
Mean Absolute Percentage Error (MAPE) : 185.9833

 R square : 0.5400646757616663
```

Interpretation

- The **R^2** is 0.54, meaning the linear regression model fits 54% of the data. We considered this an acceptable score so far given the characteristic of our data containing two different patterns (pre-COVID and post-COVID).
- **Mean Error** is -0.4255 so average prediction tends to lower than the actual data.
- A **Root Mean Squared Error** of 25 is an unsatisfactory result given that most of the target variables fall under a range of 0 to 100.
- **Mean Absolute Error** is around 17 units of bike, so the average predicted distance from the true value is 17. Although this is not the best nor worst scenario, we would like to improve the performace and reduce the MAE.
- Our linear regression **Mean Percentage Error** is -30.4648. Negative sign indicates that the predicted value is less than the actual value, and the number is how big the error is. So on average, the predictions are 30% less than the actual bike counts.
- Our **Mean Absolute Percentage Error** is 185.9833, meaning our prediction is off by **186%**, or on average, the forecast's distance from the true value is 186% of the true value. Since smaller MAPE value indicate a better fit, 186% is not a pleasant number.

Logistics Regression

In [46]:
```
log = LogisticRegression()
log.fit(train_X, train_y)
```

```
/Users/luqinyuan/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_mod
el/_logistic.py:764: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[46]:
```
LogisticRegression()
```

In [47]:
```
regressionReport(log, valid_X, valid_y)
```

```
Regression statistics

                      Mean Error (ME) : 23.3125
       Root Mean Squared Error (RMSE) : 44.3706
            Mean Absolute Error (MAE) : 29.0674
          Mean Percentage Error (MPE) : 38.0909
Mean Absolute Percentage Error (MAPE) : 103.0535

 R square : -0.38555357036974214
```

Interpretation

- The **R^2** is -0.29, meaning that a horizontal line (R^2 of 0) is going fitting our data even better. Model rendered unusable.
- **Mean Error** is 23 so average error is about 23 counts of bike, which is not ideal considering we have a a range of 1 to 100 bike rentals for majority of the data.
- A **Root Mean Squared Error** of 42 is an unsatisfactory result, again, because given that most of the target variables fall under a range of 0 to 100.
- **Mean Absolute Error** is around 27 units of bike, so the average predicted distance from the true value is 27. This is an unsatisfactory result for the same reasons.
- Our linear regression **Mean Percentage Error** is 28.5826. Positive sign indicates that the predicted value is more than the actual value, and the number is how big the error is. So on average, the predictions are 28% more than the actual bike counts.
- Our **Mean Absolute Percentage Error** is 102, meaning our prediction is off by **102%**, or on average, the forecast's distance from the true value is 102% of the true value. Since smaller MAPE value indicate a better fit, 102% is still not a pleasant number, because it is over 100.

Logistics Regression model is not well performed.

Naive Bayes

In [48]:
```
nb = GaussianNB()
nb.fit(train_X, train_y)
```

Out[48]:
```
GaussianNB()
```

In [49]:
```
regressionReport(nb, valid_X, valid_y)
```

```
Regression statistics

                      Mean Error (ME) : -43.1538
       Root Mean Squared Error (RMSE) : 59.2873
            Mean Absolute Error (MAE) : 47.1735
          Mean Percentage Error (MPE) : -512.6460
Mean Absolute Percentage Error (MAPE) : 521.2045

 R square : -1.4737484769331277
```

Interpretation

- The **R^2** is -1.88, meaning that a horizontal line (R^2 of 0) is going fitting our data even better. Model rendered unusable, and even worst than the previous logistics model.
- **Mean Error** is -48.5724 so average error is about 48 counts of bike, which is not ideal considering we have a a range of 1 to 100 bike rentals for majority of the data.
- A **Root Mean Squared Error** of 63.9789 is an unsatisfactory result, again, because given that most of the target variables fall under a range of 0 to 100.
- **Mean Absolute Error** is around 51.4375 units of bike, so the average predicted distance from the true value is 51.4375. This is an unsatisfactory result for the same reasons.
- Our linear regression **Mean Percentage Error** is -552.639. Negative sign indicates that the predicted value is less than the actual value, and the number is how big the error is. So on average, the predictions are 552% less than the actual bike counts.
- Our **Mean Absolute Percentage Error** is 560.0885, meaning our prediction is off by **560%**, or on average, the forecast's distance from the true value is 560% of the true value. Since smaller MAPE value indicate a better fit, 560% is still not a pleasant number, because it is over 100.

Naive Bayes Regression model is not well performed. R square, MPE and MAPE are TOO large

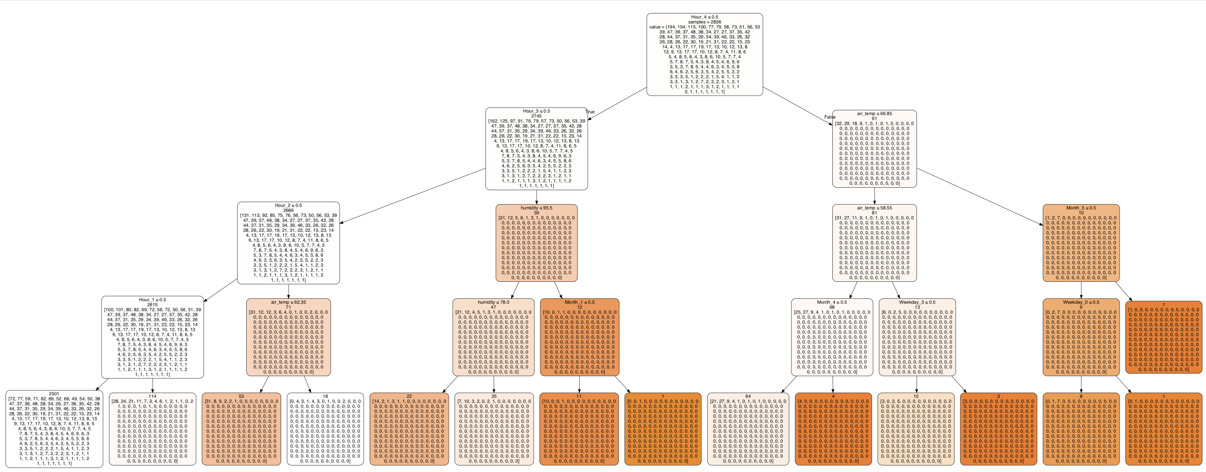
Decision Tree

In [50]:
```
dt = DecisionTreeClassifier(max_depth = 4)
dt.fit(train_X, train_y)
```

Out[50]: DecisionTreeClassifier(max_depth=4)

In [51]:
```
plotDecisionTree(dt, feature_names = X.columns)
```

Out[51]:



In [52]:
```
regressionReport(dt, valid_X, valid_y)
```

```
Regression statistics

                      Mean Error (ME) : 31.8339
       Root Mean Squared Error (RMSE) : 49.3106
            Mean Absolute Error (MAE) : 31.9161
          Mean Percentage Error (MPE) : 72.7203
Mean Absolute Percentage Error (MAPE) : 80.4505

 R square : -0.7112501036272325
```

Interpretation

- The **R^2** is -0.711, meaning that a horizontal line (R^2 of 0) is going fitting our data even better. Model rendered unusable, and even worst than the previous logistics model.
- **Mean Error** is 31.8322 so average error is about 31 counts of bike, which is not ideal considering we have a a range of 1 to 100 bike rentals for majority of the data.
- A **Root Mean Squared Error** of 49.3106 is an unsatisfactory result, again, because given that most of the target variables fall under a range of 0 to 100.
- **Mean Absolute Error** is around 31.9178 units of bike, so the average predicted distance from the true value is 31.9178. This is an unsatisfactory result for the same reasons.
- Our linear regression **Mean Percentage Error** is 72.5558. Positive sign indicates that the predicted value is more than the actual value, and the number is how big the error is. So on average, the predictions are 72.558% less than the actual bike counts.
- Our **Mean Absolute Percentage Error** is 80.6150, meaning our prediction is off by 80%, or on average, the forecast's distance from the true value is 80% of the true value. Since smaller MAPE value indicate a better fit, 80% is still not a pleasant number, because it is over 100.

Decision Tree Regression model is not well performed.

Random Forest

In [53]:
```
rf = RandomForestClassifier(n_estimators = 1000, max_depth=5, random_state=0)
rf.fit(train_X, train_y)
```

Out[53]:
```
RandomForestClassifier(max_depth=5, n_estimators=1000, random_state=0)
```

In [54]:
```
regressionReport(rf, valid_X, valid_y)
```

```
Regression statistics

                      Mean Error (ME) : 32.4663
       Root Mean Squared Error (RMSE) : 49.7169
            Mean Absolute Error (MAE) : 32.5255
          Mean Percentage Error (MPE) : 81.6469
Mean Absolute Percentage Error (MAPE) : 83.3319

 R square : -0.7395671376863304
```

Interpretation

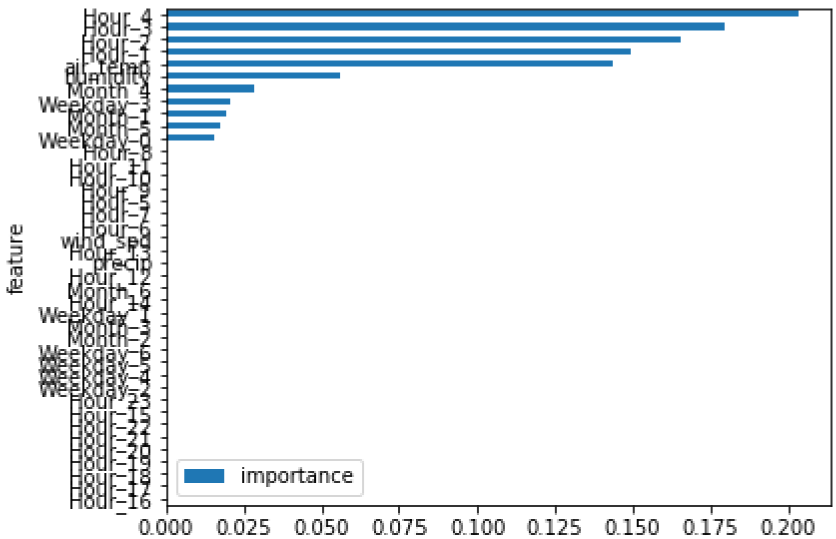
- The **R^2** is -0.74, meaning that a horizontal line (R^2 of 0) is going fitting our data even better. Model rendered unusable, and even worst than the previous logistics model.
- **Mean Error** is 32.1957 so average error is about 32 counts of bike, which is not ideal considering we have a a range of 1 to 100 bike rentals for majority of the data.
- A **Root Mean Squared Error** of 49.6544 is an unsatisfactory result, again, because given that most of the target variables fall under a range of 0 to 100.
- **Mean Absolute Error** is around 32 units of bike, so the average predicted distance from the true value is 32. This is an unsatisfactory result for the same reasons.
- Our linear regression **Mean Percentage Error** is 79.8488. Positive sign indicates that the predicted value is more than the actual value, and the number is how big the error is. So on average, the predictions are 80% less than the actual bike counts.
- Our **Mean Absolute Percentage Error** is 83, meaning our prediction is off by **83%**, or on average, the forecast's distance from the true value is 83% of the true value. Since smaller MAPE value indicate a better fit, 83% is still not a pleasant number, because it is over 100.

RandomForest Regression model is not well performed. The score satisitics for Random Forest are very similar to the ones of Decision Tree.

It appears that linear regression was our best fit model to the dataset because it was the only model without a negative R square.

3. Select only the most important features

In [55]:
```
var_imp = pd.DataFrame({'feature': train_X.columns, 'importance': dt.feature_importances_}).sort_values('importance',ascending=True)
var_imp

ax = var_imp.plot(kind='barh', x='feature', legend=True)


plt.tight_layout()
plt.show()
```



In [56]:
```
print(pd.DataFrame(list(zip(valid_X.columns,dt.feature_importances_))).sort_values(1, ascending=False))
```

```
            0         1
7      Hour_4  0.203350
6      Hour_3  0.179871
5      Hour_2  0.165682
4      Hour_1  0.149388
0    air_temp  0.143752
2    humidity  0.056280
37    Month_4  0.028461
30  Weekday_3  0.020813
34    Month_1  0.019447
38    Month_5  0.017216
27  Weekday_0  0.015739
36    Month_3  0.000000
35    Month_2  0.000000
23    Hour_20  0.000000
33  Weekday_6  0.000000
32  Weekday_5  0.000000
31  Weekday_4  0.000000
29  Weekday_2  0.000000
28  Weekday_1  0.000000
26    Hour_23  0.000000
25    Hour_22  0.000000
24    Hour_21  0.000000
20    Hour_17  0.000000
22    Hour_19  0.000000
13    Hour_10  0.000000
3    wind_spd  0.000000
8      Hour_5  0.000000
9      Hour_6  0.000000
10     Hour_7  0.000000
11     Hour_8  0.000000
12     Hour_9  0.000000
14    Hour_11  0.000000
21    Hour_18  0.000000
15    Hour_12  0.000000
16    Hour_13  0.000000
17    Hour_14  0.000000
18    Hour_15  0.000000
19    Hour_16  0.000000
1      precip  0.000000
39    Month_6  0.000000
```

Interpretation

It appears that the important features are:

- 4AM, 3AM, 2AM, 1AM because no one rides bike during these times.
- Air Temperature, which is expected because we observed a positive correlation between it and rented bike counts.
- Humidity, possibly related to precipitation.
- Months 1, 4, and 6 because represents three different time periods of COVID.

4. Parameter optimization

In [57]:
```
classifier = RandomForestClassifier(random_state=0)

grid_param = {
    'n_estimators':[100,200],
     'criterion': ['gini', 'entropy'],
    'bootstrap': [True, False]
}


grid = GridSearchCV(estimator=classifier,
                     param_grid=grid_param,
                     scoring='r2',
                     cv=2,
                     n_jobs=-1)

grid.fit(train_X, train_y)

print(grid.best_params_)

print(grid.best_score_)
```

```
/Users/luqinyuan/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_sele
ction/_split.py:672: UserWarning: The least populated class in y has only 1 m
embers, which is less than n_splits=2.
  % (min_groups, self.n_splits)), UserWarning)

{'bootstrap': False, 'criterion': 'gini', 'n_estimators': 200}
0.5112188102144959
```

5. Ensembles & Stacking

In [58]:
```
estimators = [('nb', GaussianNB()),
              ('dt', DecisionTreeClassifier(max_depth = 4)),
              ('log', LogisticRegression()))]
```

In [59]:
```
clf = StackingClassifier(estimators = estimators)
```

In [60]:
```
clf.fit(train_X, train_y)
```

```
/Users/luqinyuan/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_sele
ction/_split.py:672: UserWarning: The least populated class in y has only 1 m
embers, which is less than n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
/Users/luqinyuan/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_sele
ction/_validation.py:905: RuntimeWarning: Number of classes in training fold
(175) does not match total number of classes (178). Results may not be approp
riate for your use case. To fix this, use a cross-validation technique result
ing in properly stratified folds
  RuntimeWarning)
/Users/luqinyuan/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_sele
ction/_validation.py:905: RuntimeWarning: Number of classes in training fold
(171) does not match total number of classes (178). Results may not be approp
riate for your use case. To fix this, use a cross-validation technique result
ing in properly stratified folds
  RuntimeWarning)
/Users/luqinyuan/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_sele
ction/_validation.py:905: RuntimeWarning: Number of classes in training fold
(172) does not match total number of classes (178). Results may not be approp
riate for your use case. To fix this, use a cross-validation technique result
ing in properly stratified folds
  RuntimeWarning)
/Users/luqinyuan/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_sele
ction/_validation.py:905: RuntimeWarning: Number of classes in training fold
(173) does not match total number of classes (178). Results may not be approp
riate for your use case. To fix this, use a cross-validation technique result
ing in properly stratified folds
  RuntimeWarning)
/Users/luqinyuan/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_sele
ction/_validation.py:905: RuntimeWarning: Number of classes in training fold
(173) does not match total number of classes (178). Results may not be approp
riate for your use case. To fix this, use a cross-validation technique result
ing in properly stratified folds
  RuntimeWarning)
/Users/luqinyuan/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_sele
ction/_split.py:672: UserWarning: The least populated class in y has only 1 m
embers, which is less than n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
/Users/luqinyuan/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_sele
ction/_validation.py:905: RuntimeWarning: Number of classes in training fold
(175) does not match total number of classes (178). Results may not be approp
riate for your use case. To fix this, use a cross-validation technique result
ing in properly stratified folds
  RuntimeWarning)
/Users/luqinyuan/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_sele
ction/_validation.py:905: RuntimeWarning: Number of classes in training fold
(171) does not match total number of classes (178). Results may not be approp
riate for your use case. To fix this, use a cross-validation technique result
ing in properly stratified folds
  RuntimeWarning)
/Users/luqinyuan/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_sele
ction/_validation.py:905: RuntimeWarning: Number of classes in training fold
(172) does not match total number of classes (178). Results may not be approp
riate for your use case. To fix this, use a cross-validation technique result
ing in properly stratified folds
  RuntimeWarning)
/Users/luqinyuan/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_sele
```

```
ction/_validation.py:905: RuntimeWarning: Number of classes in training fold
(173) does not match total number of classes (178). Results may not be approp
riate for your use case. To fix this, use a cross-validation technique result
ing in properly stratified folds
  RuntimeWarning)
/Users/luqinyuan/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_sele
ction/_validation.py:905: RuntimeWarning: Number of classes in training fold
(173) does not match total number of classes (178). Results may not be approp
riate for your use case. To fix this, use a cross-validation technique result
ing in properly stratified folds
  RuntimeWarning)
```

Out[60]:
```
StackingClassifier(estimators=[('nb', GaussianNB()),
                               ('dt', DecisionTreeClassifier(max_depth=4)),
                                ('log', LogisticRegression()))])
```

In [61]:
```
regressionReport(clf, valid_X, valid_y)
```

```
Regression statistics

                      Mean Error (ME) : 21.1842
       Root Mean Squared Error (RMSE) : 44.5069
            Mean Absolute Error (MAE) : 26.1990
          Mean Percentage Error (MPE) : 9.3267
Mean Absolute Percentage Error (MAPE) : 83.8912

 R square : -0.39407987579943593
```

Even after stacking, our results were not satisfactory.

Interpretation

- The **R^2** is -0.3, meaning that a horizontal line (R^2 of 0) is going fitting our data even better. Model rendered unusable.
- **Mean Error** is 20.4704 so average error is about 27 counts of bike, which is not ideal considering we have a a range of 1 to 100 bike rentals for majority of the data.
- A **Root Mean Squared Error** of 43.0179 is an unsatisfactory result, again, because given that most of the target variables fall under a range of 0 to 100.
- **Mean Absolute Error** is 24.5526 units of bike, so the average predicted distance from the true value is 24.5526. This is an unsatisfactory result for the same reasons.
- Our linear regression **Mean Percentage Error** is 3.7611. Positive sign indicates that the predicted value is more than the actual value, and the number is how big the error is. So on average, the predictions are 3.7% more than the actual bike counts.
- Our **Mean Absolute Percentage Error** is 83.0684, meaning our prediction is off by **83%**, or on average, the forecast's distance from the true value is 83% of the true value. Since smaller MAPE value indicate a better fit, 83% is still not a pleasant number, because it is over 100.

Conclusions

Discussions, Insights, and Limitations

Although we did not achieve the desired results from this project, our teammates were able to gain the following insights, and learned from some limitations.

1. It is very important to have a clear definition of the project scope.

1. Before developing any codes, it is important to understand how will the data work to resolve the business problem raised. Because our team did not practice this in our mobile app topic prior, we ended up with two dataset holding information that won't contribute much to our problem. Furthermore, in the middle working on our new Citi Bike we chose another "poor" dataset for the purpose of using weather to predict bike usage, because we did not take into consideration of COVID19 impacting the ridership.

1. One of our biggest limitation throughout this project was **time**. Our group started off with a different topic on mobile apps. But that topic was ultamitely dropped because we chose two unapplicable datasets, and we'd also wish to be able to apply supervised machine learning for the group project. That left us with a very tight shedule to start from the beginning again, and accomplish the works of two phases together.

1. Since we had very serious time limitation, to save more time on exploratory data analysis and model building, we decided to choose the data of the first 6 months of 2020 in Jersey City. We were also concerned with the bulk of data, and not every teammate will have sufficient space on their machine to hold more than 6 months worth of Jersey City and weather data. Therefore, when we finished building models, the outcomes were not as good as we expected early.

1. When we started this project, we assumed an objective factor like weather would ultimately decide the usage of Citi Bike. However, we learned that a practical distribution plan needs to consider subjective factors, such as pandemics or economic instability, as factors that influence individual behavior.

1. Some ideas to why we think the models didn't work:

   - Not enough time, weather, and season variation.
   - The sudden ridership drops in April was "disruptive" to our prediction model because the independent variables couldn't explain this pattern change.

Future Work & Recommendation

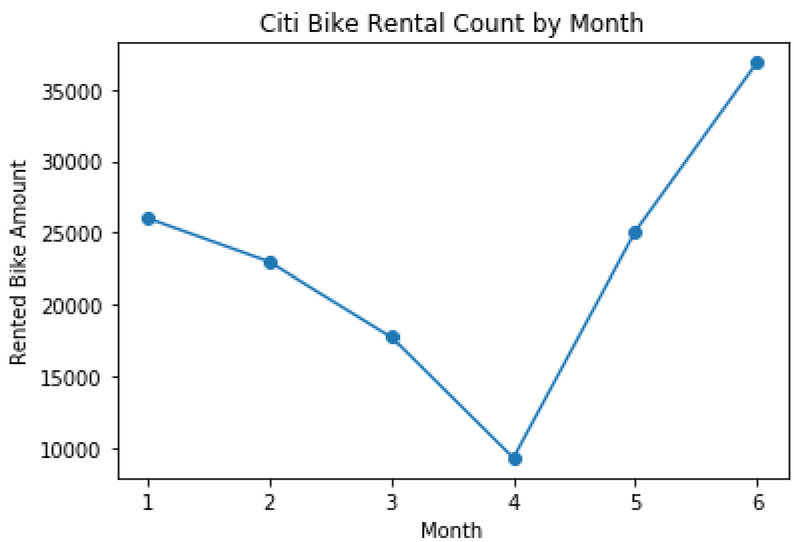
1. For future work, we would like to first consider performing a **time series** analysis to unveil any internal sturctures such as weekly trends, seasonal trends, monthly trends, and more... We limited ourselves within the realm of predictive modeling.

1. Another possible future work may be capturing a longer range of data. We shall include data for over at least **one full year** to capture enough time, seasonal, and weather variation to forecast bicycle demands. We only included January to June of 2020 which excluded fall seasons, and weather variations of later-half of the year.

1. We'd also like to look more deeply and explicitly into the changing ridership patterns **between pre-COVID19 and post-COVID19**. In our study, we were able to learn that riderships which used to mainly occur in weekday rush hours are now more focused in afternoon periods on weekends. Now that 2020 is almost over, our team can leverage on the dataset for the later half of the year to look at future pattern changes.

   - Did the post-COVID19 trend we observe persist, or did it followed a different path.
   - Was seasonal change (fall) able to affect any patterns?
   - Moreover, was the pattern change in June a seasonal change or affected by COVID19?

1. Our team observed poorly performed models. Due to the fact that we were trying to predict an unconventional trend in bike usage:



   We thought that out existing variables weren't sufficient to explain the pattern fully.
2. Related to the first and fourth point, our team would like to experiment more with other prediction models. For example, we can use ARIMA to perform time series analysis.