Visualization of predicted data

In [1]:
```
# Import pandas, numpy, matplotlib, seaborn libraries
import pandas as pd
from pandas.api.types import is_numeric_dtype
import datetime
import pickle

# path = './RandomForest/'
path = ''
```

Make Predictions on Test Data

In [2]:
```
def predictRandomForest(rf, test_features):
    try:

        # Use the forest's predict method on the test data
        predictions = rf.predict(test_features).tolist()
        return predictions

    except Exception as e:
        print("Error in predictRandomForest", e)
```

Get dateRange dataframe for prediction

In [3]:
```
# ML Models needs time parameters for input [i.e. test_features] in specific format :
# dateInput = [date hour minute year month1 month2 month3 ... month12]

# get_TimeInputs() generates a dataframe congruent with ML model input format for next 5 hours by 30 minute interval
w.r.t. datetimeIn
# DayOfWeek is optionally included in returned dataframe. It is required for getBike() function.

def get_TimeInputs(ts = datetime.datetime.now(), dayOfWeek = False, tStampRet = False, perHrs = 5, intervals = 10):
    try:
        tStampRetVals=[]
        data_entry_timestamp_In = ts
        ts = pd.DataFrame({'ts}, columns = ['ts'])
        data_entry_timestamp_In = pd.Timestamp(data_entry_timestamp_In)
        data_entry_timestamp = data_entry_timestamp_In

        # Logic to find nearest value divisible by 10
        minutes = data_entry_timestamp_In.minute
        minutes = minutes + (30-(minutes%30))

        data_entry_timestamp = data_entry_timestamp.replace(minute = minutes) if (minutes != 60)\
                                else  data_entry_timestamp.round('H')

        # obtain datetime range spacing 30 minutes for next "perHrs" hours by spacing of "intervals" minutes per ho
ur
        dateRange = pd.DataFrame(pd.date_range(data_entry_timestamp, periods=perHrs*int(60/intervals), freq='
{}Min'.format(intervals)), columns = ['ts'])
        dateRange = pd.concat([ts, dateRange]).reset_index(drop = True)
        dateRange['ts'] = pd.to_datetime(dateRange['ts'])

        # Sort datetime into dataframe
        dateRange['date'] = dateRange.ts.dt.day
        dateRange['hour'] = dateRange.ts.dt.hour
        dateRange['minute'] = dateRange.ts.dt.minute
        dateRange['year'] = dateRange.ts.dt.year
        dateRange['month'] = dateRange.ts.dt.month
        dateRange['dayOfWeek'] = dateRange.ts.dt.weekday

        # prepare onhot encodes for year and month
        mask = dateRange.year == 2020
        column_name = 'year'
        dateRange.loc[mask, column_name] = 1

        # Months is trained with month data onehot encoded for all 12 months; hence similar data
        # is introduced inorganically dataframe
        for m in range(1,13):
            mask = dateRange.month == m
            column_name = 'month'+str(m)
            dateRange[column_name] = 0
            dateRange.loc[mask, column_name] = 1

        dateRange = dateRange.drop(['month'], axis=1)

        if dayOfWeek:
            for d in range(7):
                mask = dateRange.dayOfWeek == d
                column_name = 'dayOfWeek'+str(d)
                dateRange[column_name] = 0
                dateRange.loc[mask, column_name] = 1

        dateRange = dateRange.drop(['dayOfWeek'], axis=1)

        if tStampRet:
            tStampRetVal = dateRange.ts.dt.strftime("%I:%M %p")

        dateRange = dateRange.drop(['ts'], axis=1)

        return dateRange, tStampRetVal

    except Exception as e:
        print("Error in get_TimeInputs", e)
```

In [4]:
```
get_TimeInputs()
```

Out[4]:
```
    date  hour  minute  year  month1  month2  month3  month4  month5  month6  \
0     16    17      30     1       0       0       0       1       0       0
1     16    17      30     1       0       0       0       1       0       0
2     16    18       0     1       0       0       0       1       0       0
3     16    18      30     1       0       0       0       1       0       0
4     16    19       0     1       0       0       0       1       0       0
5     16    19      30     1       0       0       0       1       0       0
6     16    20       0     1       0       0       0       1       0       0
7     16    20      30     1       0       0       0       1       0       0
8     16    21       0     1       0       0       0       1       0       0
9     16    21      30     1       0       0       0       1       0       0
10    16    22       0     1       0       0       0       1       0       0

    month7  month8  month9  month10  month11  month12
0        0       0       0        0        0        0
1        0       0       0        0        0        0
2        0       0       0        0        0        0
3        0       0       0        0        0        0
4        0       0       0        0        0        0
5        0       0       0        0        0        0
6        0       0       0        0        0        0
7        0       0       0        0        0        0
8        0       0       0        0        0        0
9        0       0       0        0        0        0
10       0       0       0        0        0        0 ,
[])
```

Weather prediction

Support function which returns dataframe containing weather parameters for coming 5 hours by space of 30 minutes

In [5]:
```
# dataFrame of all default targetCols are returned when getWeather() is called by getBike().
# dataFrame of All targetCols is used as input to ML model for predicting station availability

def getWeather(dateTimeIn = datetime.datetime.now(),targetCols = ['main_temp','main_feels_like', 'main_pressure', 'm
ain_humidity', \
               'main_temp_min', 'main_temp_max', 'wind_speed', 'wind_deg'],tStampReturn = False):

    try:

        test_features,tStampRetVal = get_TimeInputs(dateTimeIn,tStampRet = tStampReturn)
        result = pd.DataFrame()

        for targetCol in targetCols:
            # load the model
            filename = path + 'targetC='+targetCol+'.pkl'
            rfWeatherModel_loaded = pickle.load(open(filename, 'rb'))

            predictions = predictRandomForest(rfWeatherModel_loaded,test_features)
            result[targetCol] = predictions

        for col in result.columns:
            if not is_numeric_dtype(result[col]):
                result[col] = result[col].astype('category')
            else:
                result[col] = result[col].astype('int')

        return result, tStampRetVal

    except Exception as e:
        print("Error in getWeather", e)
```

In [6]:
```
getWeather()
```

Out[6]:
```
(    main_temp  main_feels_like  main_pressure  main_humidity  main_temp_min  \
 0          8               1           1015             73              7
 1          8               1           1015             75              7
 2          8               1           1015             76              7
 3          8               2           1015             78              6
 4          8               2           1015             80              7
 5          8               1           1015             81              7
 6          8               2           1015             81              7
 7          8               1           1015             81              7
 8          8               7           1016             83              7
 9          8               3           1016             86              7
 10         8               3           1016             84              7

    main_temp_max  wind_speed  wind_deg
 0              9           6       210
 1              9           7       219
 2              9           7       210
 3              9           7       210
 4              9           7       210
 5              9           7       210
 6              9           7       219
 7              9           8       210
 8              9           6       210
 9              9           6       212
 10             8           6       212 ,
 [])
```

Get future bike station parameters

Function which returns array of station parameters for next 5 hours by 30 minute interval w.r.t. datetimeIn

In [7]:
```
# Function which returns array of station parameters for next 5 hours by 30 minute interval w.r.t. dateimeIn
def getBike(station = 42, dateTimeIn = datetime.datetime.now(), targetC = "bikes"):

    try:

        targetCols = {"bikes' : 'available_bikes', "stands" : 'available_bike_stands'}
        # dateTimeIn = datetime.datetime.fromtimestamp(dateTimeIn)

        weatherFeatures, tStampRetVal = getWeather(dateTimeIn)
        bikeFeatures,tStampRetVal = get_TimeInputs(ts = dateTimeIn, dayOfWeek = True, tStampRet= True)
        test_features = weatherFeatures.join(bikeFeatures).values.tolist()

        result = pd.DataFrame()

        # load the model
        filename = path + 'station'+str(station)+"_"+targetCols[targetC]+'.pkl'
        rfWeatherModel_loaded = pickle.load(open(filename, 'rb'))

        # Predict station availability for all input timestamps and predicted weather parameters
        predictions = predictRandomForest(rfWeatherModel_loaded,test_features)
        result[targetCols[targetC]] = predictions
        # Cast results as integer values
        result = result.astype(int)

        return [result[targetCols[targetC]].values.tolist(),list(tStampRetVal)]

    except Exception as e:
        print("Error in getBike", e)
```

In [8]:
```
getBike()
```

Out[8]:
```
[[24, 24, 15, 17, 18, 19, 19, 19, 19, 19, 19],
 ['05:16 PM',
  '05:30 PM',
  '06:00 PM',
  '06:30 PM',
  '07:00 PM',
  '07:30 PM',
  '08:00 PM',
  '08:30 PM',
  '09:00 PM',
  '09:30 PM',
  '10:00 PM']]
```

Support function which returns requested weather predictions as a list of lists

In [9]:
```
def retWeather(dateTimeIn = datetime.datetime.now(),targetCols =['main_temp']):
    try:

        result = []
        # dateTimeIn = datetime.datetime.fromtimestamp(dateTimeIn)
        weatherFeatures,tStampRet = getWeather(dateTimeIn = dateTimeIn,targetCols = targetCols, tStampReturn = True)
        for col in weatherFeatures.columns:
            result.append(weatherFeatures[col].values.tolist())
        result.append(tStampRet.values.tolist())
        return result

    except Exception as e:
        print("Error in retWeather", e)
```

In [10]:
```
retWeather()
```

Out[10]:
```
[[8, 8, 8, 8, 8, 8, 8, 8, 7, 8, 8],
 ['05:16 PM',
  '05:30 PM',
  '06:00 PM',
  '06:30 PM',
  '07:00 PM',
  '07:30 PM',
  '08:00 PM',
  '08:30 PM',
  '09:00 PM',
  '09:30 PM',
  '10:00 PM']]
```

Get future parameters for bike station and weather

Driver function retBikeWeather() returns list of lists consisting:
[[Bikes Soruce], [main weather Source],[main temperature Source],[Bike_stands Destination], [main weather Destination], [main_temperature Destination]]

In [11]:
```
# Driver function retBikeWeather() returns list of lists consisting:

# [[Bikes Soruce], [main weather Source],[main temperature Source],\
# [Bike_stands Destination], [main weather Destination], [main_temperature Destination]]

def retBikeWeather(stations = [42,43], dateTimeIP = [datetime.datetime.now(), datetime.datetime.now()], targetC = [
"bikes", "stands"]):
    try:

        result = []
        # dateTimeIn = datetime.datetime.fromtimestamp(dateTimeIn)
        for (station,dateTimeIn, targetC) in zip(stations, dateTimeIP, targetC):
            bData = getBike(station = station, dateTimeIn = dateTimeIn, targetC = targetC)
            result.append(bData)
            wData = retWeather(dateTimeIn = dateTimeIn,targetCols ='weather_main'})
            result.append(wData)
            tData = retWeather(dateTimeIn = dateTimeIn, targetCols ='main_temp'})
            result.append(tData)

        return result

    except Exception as e:
        print("Error in retBikeWeather", e)
```

In [12]:
```
retBikeWeather(stations = [41,41], dateTimeIP = [datetime.datetime.now(), datetime.datetime.now()], targetC = ["bike
s", "stands"])
```

Out[12]:
```
[[[18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18],
  ['05:16 PM',
   '05:30 PM',
   '06:00 PM',
   '06:30 PM',
   '07:00 PM',
   '07:30 PM',
   '08:00 PM',
   '08:30 PM',
   '09:00 PM',
   '09:30 PM',
   '10:00 PM']],
 [['Clouds',
   'Clouds',
   'Clouds',
   'Clouds',
   'Clouds',
   'Clouds',
   'Rain',
   'Clouds',
   'Clouds',
   'Clouds'],
  ['05:16 PM',
   '05:30 PM',
   '06:00 PM',
   '06:30 PM',
   '07:00 PM',
   '07:30 PM',
   '08:00 PM',
   '08:30 PM',
   '09:00 PM',
   '09:30 PM',
   '10:00 PM']],
 [[8, 8, 8, 8, 8, 8, 8, 8, 7, 8, 8],
  ['05:16 PM',
   '05:30 PM',
   '06:00 PM',
   '06:30 PM',
   '07:00 PM',
   '07:30 PM',
   '08:00 PM',
   '08:30 PM',
   '09:00 PM',
   '09:30 PM',
   '10:00 PM']],
 [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
  ['05:16 PM',
   '05:30 PM',
   '06:00 PM',
   '06:30 PM',
   '07:00 PM',
   '07:30 PM',
   '08:00 PM',
   '08:30 PM',
   '09:00 PM',
   '09:30 PM',
   '10:00 PM']],
 [[8, 8, 8, 8, 8, 8, 8, 8, 7, 8, 8],
  ['05:16 PM',
   '05:30 PM',
   '06:00 PM',
   '06:30 PM',
   '07:00 PM',
   '07:30 PM',
   '08:00 PM',
   '08:30 PM',
   '09:00 PM',
   '09:30 PM',
   '10:00 PM']]]
```