In [296]:
```
# Environment Setup
from pyspark import SparkContext, SQLContext
from pyspark.sql import functions
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, DoubleType, TimestampType
from pyspark.ml import Pipeline
from pyspark.ml.regression import DecisionTreeRegressor, GBTRegressor, RandomForestRegressor
from pyspark.ml.feature import VectorAssembler, VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator
import pandas as pd
import datetime
import folium
from folium.plugins import HeatMapWithTime
import calendar
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
```

In [111]:
```
SC = SparkContext('local', 'citibike')
sqlContext = SQLContext(SC)

# Stream citibike data
df = sqlContext.read.format('com.databricks.spark.csv').options(header='true', inferschema='true').load('201904-citibike-tripdata.csv')
df_May = sqlContext.read.format('com.databricks.spark.csv').options(header='true', inferschema='true').load('201905-citibike-tripdata.csv')
df_Jun = sqlContext.read.format('com.databricks.spark.csv').options(header='true', inferschema='true').load('201906-citibike-tripdata.csv')
df_Jul = sqlContext.read.format('com.databricks.spark.csv').options(header='true', inferschema='true').load('201907-citibike-tripdata.csv')
df = df.union(df_May)
df = df.union(df_Jun)
df = df.union(df_Jul)

df_519 = df.filter(df['start station id'] == 519)

hour_window = functions.window(functions.col('starttime'), '1 hour', '1 hour').start.alias('starttime')
hour_number = df_519.groupBy(hour_window).count()
hour_sort = hour_number.sort('starttime')

# Stream weather data and concatenate with citibike data
schema_weather = StructType([
    StructField('starttime', StringType(), True),
    StructField('SPD', DoubleType(), True),
    StructField('GUS', DoubleType(), True),
    StructField('SKC', DoubleType(), True),
    StructField('VSB', DoubleType(), True),
    StructField('TEMP', DoubleType(), True),
    StructField('DEWP', DoubleType(), True),
    StructField('SLP', DoubleType(), True),
    StructField('ALT', DoubleType(), True),
    StructField('STP', DoubleType(), True),
    StructField('PCP01', DoubleType(), True)
])

df_weather = sqlContext.read.format('com.databricks.spark.csv').options(header='true').schema(schema_weather).load('processed_weather.csv')
df_join = df_weather.join(hour_sort, ['starttime'], 'left')
df_sort = df_join.sort('starttime')
df_sort = df_sort.withColumn('starttime', df_sort['starttime'].cast(TimestampType()))
df_filled = df_sort.fillna(0, subset='count')

# Add features 'hour' and 'is_weekend'
df_hour = df_filled.withColumn('hour', functions.hour('starttime'))
df_final = df_hour.withColumn('is_weekend', functions.dayofweek('starttime').isin([1, 7]).cast('int'))
df_final.show()
```

```
+-------------------+-----+----+---+---+----+----+-----+-----+-----+-----+----------+
|          starttime|SPD| GUS|SKC|VSB|TEMP|DEWP|  SLP|  ALT|  STP|PCP01|count|hour|is_weekend|
+-------------------+-----+----+---+---+----+----+-----+-----+-----+-----+-----+----+----------+
|2019-04-01 00:00:00| 5.0|21.0|0.0|10.0|40.0|20.0|1014.4|29.98|1009.5|  0.0|    5|   0|         0|
|2019-04-01 01:00:00|13.0|22.0|1.0|10.0|38.0|18.0|1015.1| 30.0|1010.1|  0.0|    0|   1|         0|
|2019-04-01 02:00:00|10.0|18.0|0.0|10.0|36.0|17.0|1015.5|30.01|1010.5|  0.0|    0|   2|         0|
|2019-04-01 03:00:00|15.0|18.0|0.0|10.0|35.0|17.0|1016.1|30.03|1011.2|  0.0|    0|   3|         0|
|2019-04-01 04:00:00| 9.0| 0.0|0.0|10.0|35.0|16.0|1017.2|30.06|1012.2|  0.0|    0|   4|         0|
|2019-04-01 05:00:00| 8.0|20.0|0.0|10.0|34.0|16.0|1016.2|30.09|1013.2|  0.0|    3|   5|         0|
|2019-04-01 06:00:00| 8.0|24.0|0.0|10.0|33.0|15.0|1019.1|30.12|1014.2|  0.0|   16|   6|         0|
|2019-04-01 07:00:00|10.0|22.0|0.0|10.0|33.0|15.0|1020.4|30.16|1015.5|  0.0|   39|   7|         0|
|2019-04-01 08:00:00| 8.0|17.0|0.0|10.0|35.0|16.0|1020.9|30.17|1015.9|  0.0|   63|   8|         0|
|2019-04-01 09:00:00| 8.0| 0.0|0.0|10.0|37.0|16.0|1021.7|30.19|1016.6|  0.0|   38|   9|         0|
|2019-04-01 10:00:00| 7.0|24.0|0.0|10.0|39.0|16.0|1022.5|30.22|1017.6|  0.0|   18|  10|         0|
|2019-04-01 11:00:00| 8.0|24.0|0.0|10.0|40.0|16.0|1022.9|30.23|1017.9|  0.0|    7|  11|         0|
|2019-04-01 12:00:00|11.0|21.0|0.0|10.0|42.0|16.0|1022.2|30.24|1018.2|  0.0|   12|  12|         0|
|2019-04-01 13:00:00| 3.0| 0.0|0.0|10.0|44.0|13.0|1022.2|30.24|1018.2|  0.0|   16|  13|         0|
|2019-04-01 14:00:00| 0.0| 0.0|0.0|10.0|45.0|12.0|1022.3|30.24|1018.2|  0.0|   14|  14|         0|
|2019-04-01 15:00:00|10.0|20.0|0.0|10.0|45.0|10.0|1023.4|30.25|1018.6|  0.0|   22|  15|         0|
|2019-04-01 16:00:00|15.0|18.0|0.0|10.0|45.0| 9.0|1023.7|30.26|1018.9|  0.0|   32|  16|         0|
|2019-04-01 17:00:00|11.0|21.0|0.0|10.0|45.0| 8.0|1024.7|30.28|1019.6|  0.0|   86|  17|         0|
|2019-04-01 18:00:00| 5.0| 0.0|0.0|10.0|44.0| 9.0|1025.5|30.31|1020.6|  0.0|  133|  18|         0|
|2019-04-01 19:00:00| 6.0| 0.0|0.0|10.0|43.0| 8.0|1026.3|30.33|1021.3|  0.0|   40|  19|         0|
+-------------------+-----+----+---+---+----+----+-----+-----+-----+-----+-----+----+----------+
only showing top 20 rows
```

In [250]:
```
# Split the dataset into training set, test set and realtime renewing set
df_model = df_final.drop('starttime')
list_model = df_model.collect()
training_list = list_model[0: 1464]
test_list = list_model[1464: 2184]
realtime_list = list_model[2184: 2924]
```

In [251]:
```
spark = SparkSession.builder.appName('citibike').getOrCreate()
training_df = spark.createDataFrame(training_list)
test_df = spark.createDataFrame(test_list)
realtime_df = spark.createDataFrame(realtime_list)

training_df.cache()
test_df.cache()
realtime_df.cache()
```

Out[251]:
```
DataFrame[SPD: double, GUS: double, SKC: double, VSB: double, TEMP: double, DEWP: double, SLP: double, ALT: double, STP: double, PCP01: double, count: bigint, hour: bigint, is_weekend: bigint]
```

In [218]:
```
# Build the pipeline of decision tree and train the model
features_columns = training_df.columns
features_columns.remove('count')
features_assembling = VectorAssembler(inputCols = features_columns, outputCol = 'features_assembled')
features_indexing = VectorIndexer(inputCol = 'features_assembled', outputCol = 'features', maxCategories = 25, handleInvalid = 'skip')
dt = DecisionTreeRegressor(featuresCol = 'features', labelCol = 'count', seed = 2021)

dt_pipeline = Pipeline(stages = [features_assembling, features_indexing, dt])
dt_model = dt_pipeline.fit(training_df)
dt_prediction = dt_model.transform(test_df)
evaluator = RegressionEvaluator(labelCol = 'count', predictionCol = 'prediction', metricName = 'rmse')
dt_rmse = evaluator.evaluate(dt_prediction)
print('Decision Tree Regressor', dt_rmse)
```

```
Decision Tree Regressor: 11.890417631985933
```

In [216]:
```
# Build the pipeline of gradient boost tree and train the model
gbt = GBTRegressor(featuresCol = 'features', labelCol = 'count')
gbt_pipeline = Pipeline(stages = [features_assembling, features_indexing, gbt])
gbt_model = gbt_pipeline.fit(training_df)
gbt_prediction = gbt_model.transform(test_df)
gbt_rmse = evaluator.evaluate(gbt_prediction)
print('Gradient Boost Tree Regression', gbt_rmse)
```

```
Gradient Boost Tree Regression: 13.682482439199084
```

In [217]:
```
# Build the pipeline of randomforest regressor
rf = RandomForestRegressor(numTrees=10, maxDepth=5, seed=2021, featuresCol='features', labelCol='count')
rf_pipeline = Pipeline(stages = [features_assembling, features_indexing, rf])
rf_model = rf_pipeline.fit(training_df)
rf_prediction = rf_model.transform(test_df)
rf_rmse = evaluator.evaluate(rf_prediction)
print('Random Forest Regressor', rf_rmse)
```

```
Random Forest Regressor: 13.276845143846367
```

In [265]:
```
# Realtime streaming for Decision Tree Regressor
realtime_list = realtime_df.collect()
newtrain_list = training_df.collect()
july_prediction_list = []
for i in range(int(realtime_df.count()/12)):
    realtime_point = realtime_list[i*12:i*12+12]
    realtime_dataframe = spark.createDataFrame(realtime_point)
    prediction = dt_model.transform(realtime_dataframe)
    july_prediction_list = july_prediction_list + prediction.collect()
    newtrain_list = newtrain_list + realtime_point
    newtrain_df = spark.createDataFrame(newtrain_list)
    dt_model = dt_pipeline.fit(newtrain_df)

july_prediction_df = spark.createDataFrame(july_prediction_list)
df_reg = july_prediction_df.select('*').toPandas()
df_reg.to_csv('df_regression.csv')
```

In [266]:
```
# Realtime streaming for GBTR
realtime_list = realtime_df.collect()
newtrain_list = training_df.collect()
july_prediction_list = []
for i in range(int(realtime_df.count()/12)):
    realtime_point = realtime_list[i*12:i*12+12]
    realtime_dataframe = spark.createDataFrame(realtime_point)
    prediction = gbt_model.transform(realtime_dataframe)
    july_prediction_list = july_prediction_list + prediction.collect()
    newtrain_list = newtrain_list + realtime_point
    newtrain_df = spark.createDataFrame(newtrain_list)
    gbt_model = gbt_pipeline.fit(newtrain_df)

july_prediction_df = spark.createDataFrame(july_prediction_list)
df_gbtr = july_prediction_df.select('*').toPandas()
df_gbtr.to_csv('df_gbtr.csv')
```

In [267]:
```
# Realtime streaming for Random Forest
realtime_list = realtime_df.collect()
newtrain_list = training_df.collect()
july_prediction_list = []
for i in range(int(realtime_df.count()/12)):
    realtime_point = realtime_list[i*12:i*12+12]
    realtime_dataframe = spark.createDataFrame(realtime_point)
    prediction = rf_model.transform(realtime_dataframe)
    july_prediction_list = july_prediction_list + prediction.collect()
    newtrain_list = newtrain_list + realtime_point
    newtrain_df = spark.createDataFrame(newtrain_list)
    rf_model = rf_pipeline.fit(newtrain_df)

july_prediction_df = spark.createDataFrame(july_prediction_list)
df_rf = july_prediction_df.select('*').toPandas()
df_rf.to_csv('df_rf.csv')
```

In [271]:
```
# Load data and preprocess datetime
citibike = pd.read_csv('201907-citibike-tripdata.csv')
citibike['starttime'] = citibike['starttime'].str[:-5]
citibike['stoptime'] = citibike['stoptime'].str[:-5]
citibike['starttime'] = pd.to_datetime(citibike['starttime'])
citibike['stoptime'] = pd.to_datetime(citibike['stoptime'])
citibike = citibike.set_index('starttime')
```

In [280]:
```
# Create list to store lists of locations day by day, hour by hour
df_hour_list = []
day_list = list(set(citibike.index.date))
day_list.sort()
time_index = []
for day in day_list:
    for hour in range(0, 24):
        time_index.append(datetime.datetime.combine(day, datetime.time(hour)))
for day in day_list:
    for hour in range(0,24):
        citibike_day = citibike.loc[citibike.index.date == day,['start station latitude', 'start station longitude']]
        citibike_hour = citibike_day.loc[citibike_day.index.hour == hour,['start station latitude', 'start station longitude']].groupby(['start station latitude', 'start station longitude']).size().reset_index().values.tolist()
        citibike_demand = citibike_day.loc[citibike_day.index.hour == hour,['start station latitude', 'start station longitude']].groupby(['start station latitude', 'start station longitude']).size()
        demand_max = citibike_demand.values.max()
        demand_scaled = citibike_demand.values/demand_max
        for k in range(len(citibike_hour)):
            citibike_hour[k].append(demand_scaled[k])
        df_hour_list.append(citibike_hour)
```

In [1]:
```
# Add trip events to the map
time_index = [str(x) for x in time_index]
map_time = folium.Map(location = [40.7470, -73.9955], tiles = 'CartoDB Positron', zoom_start = 13)

HeatMapWithTime(df_hour_list, index = time_index, auto_play = True, max_opacity = 0.5, gradient={0.2: 'cornflowerblue', 0.4: 'royalblue', 0.75: 'mediumblue', 1: 'blue'}).add_to(map_time)
```

In [291]:
```
# load citibike data and preprocess datetime
citibike_07_519 = pd.read_csv('citibike_07_519.csv')
citibike_07_519 = citibike_07_519.drop(columns=['Unnamed: 0'], axis=1)

citibike_07_519['starttime'] = citibike_07_519['starttime'].str[:-5]
citibike_07_519['stoptime'] = citibike_07_519['stoptime'].str[:-5]
citibike_07_519['starttime'] = pd.to_datetime(citibike_07_519['starttime'])
citibike_07_519['stoptime'] = pd.to_datetime(citibike_07_519['stoptime'])

citibike_07_519 = citibike_07_519.set_index('starttime', drop=True)

# extract date from starttime
day_list = list(set(citibike_07_519.index.date))
day_list.sort()

# combine date and time
time_index = []
for day in day_list:
    for hour in range(0,24):
        time_index.append(datetime.datetime.combine(day, datetime.time(hour)))

# extract weekday from time_index
time_text = [x.date() for x in time_index]
weekday_text = [calendar.day_name[y.weekday()] for y in time_text]

# create list to store demand hour by hour
df_hour = []
for day in day_list:
    for hour in range(0,24):
        citibike_07_519_day = citibike_07_519.loc[citibike_07_519.index.date == day, ['start station latitude', 'start station longitude']]
        citibike_07_519_demand = citibike_07_519_day.loc[citibike_07_519_day.index.hour == hour]
        df_hour.append(citibike_07_519_demand.shape[0])

# load regression, gbtr, rf result
df_reg = pd.read_csv('df_regression.csv')
reg_pre = df_reg['prediction']
df_gbtr = pd.read_csv('df_gbtr.csv')
gbtr_pre = df_gbtr['prediction']
df_rf = pd.read_csv('df_rf.csv')
rf_pre = df_rf['prediction']
```

In [292]:
```
# load weather data
weather_data = pd.read_csv('original_weather.csv')

# extract weather data hour by hour
weather_data['starttime'] = pd.to_datetime(weather_data['starttime'])
weather_data.set_index('starttime', inplace=True)
weather_data_5 = weather_data[weather_data.index.month == 5]

# extract temperature and wind speed data
temp_data = weather_data_5['TEMP']
speed_data = weather_data_5['SPD']
```

In [293]:
```
# create animation of original demands and predicted ones
fig, ax = plt.subplots()
ax.set_xlim(0, 24)
ax.set_ylim(0, 200)
ax.set_xticks(range(0, 24))
ax.set_xlabel('time(h)')
ax.set_ylabel('demands')
line, = ax.plot([], [])

plotlays, plotcols, labels = [4], ["cornflowerblue", "turquoise", "sienna", "moccasin"], ["original", "tree_reg", "gbtr", "random forest"]
lines = []
for index in range(4):
    lobj = ax.plot([], [], color=plotcols[index], label=labels[index], linewidth=2)[0]
    ax.legend(loc='upper right')
    lines.append(lobj)

def init():
    for line in lines:
        line.set_data([], [])
    return lines

x_data = [0] * 24
y_data = [0] * 24
x_data2 = [0] * 24
y_data2 = [0] * 24
x_data3 = [0] * 24
y_data3 = [0] * 24
x_data4 = [0] * 24
y_data4 = [0] * 24

def animate(i):
    x_data[i%24] = i%24
    y_data[i%24] = df_hour[i]
    x_data2[i%24] = i%24
    y_data2[i%24] = reg_pre[i]
    x_data3[i%24] = i%24
    y_data3[i%24] = gbtr_pre[i]
    x_data4[i%24] = i%24
    y_data4[i%24] = rf_pre[i]
    xlist = [x_data, x_data2, x_data3, x_data4]
    ylist = [y_data, y_data2, y_data3, y_data4]
    ax.set_title("{}, Temperature: {}, Wind speed: {}".format(weekday_text[i], temp_data[i], speed_data[i]))

    for lnum, line in enumerate(lines):
        line.set_data(xlist[lnum], ylist[lnum])
    return lines

animation = FuncAnimation(fig, animate, init_func=init, frames=np.arange(0, 732, 1), interval=150)
HTML(animation.to_html5_video())
```