Harrisburg University of Science & Technology

Project Deliverable #3

CISC 601-90- O-2020/Fall – Data Engineering and Mining

Submitted to Prof. Khaled Iskandarani, Prof. Ying Lin

Submitted By:

Noopur Mishra

SID:# 273784

Submission Date: 11/21/2020

Background

Data used: 311 Data - Allegheny County / City of Pittsburgh / Western PA Regional Data Center

Data Source: https://catalog.data.gov/dataset/311-data-in-development

The dataset includes the 311 calls detail for Allegheny County, PA from 2015 to 2020. 311 is a non-emergency phone number, people can call to find information about services and to raise complaints as well like graffiti, road damage, etc. The data is also useful to provide wealth of insight of how cities are run and their functional improvement for future.

Motivation: In my quest of finding the dataset, which could function well with the algorithm and mining method learned in the course. I stumbled upon this dataset which will provide a great supervised learning solution using powerful classification algorithms such as Random Forest Algorithm and Time series analysis with ARIMA model.

Tasks:

1: Data Analysis and Visualization

2: **Time Series Analysis** : To predict the number of 311 Request raised on each date based on the data available using ARIMA model

3: **Random Forest Classifier**: Predict the values of column NEIGHBORHOOD based on the information contained in the other columns using Random Forest Classifier Algorithm.

This data analysis is based on the hypothesis that it could be used by the county to optimally allocate their resource in cost efficient way. I analyzed the call volume and requests created based on their origin, day of the week, day, day of the month, month of the year and further I used ARIMA model to predict when can we expect higher/lower volume of request. Furthermore, Random Forest algorithm is used to predict the neighborhood using the coordinates of the area for which request is raised. With the help of prediction, County can decide on which neighborhood needs more attention based on their issues.

Data Dictionary

_ID : Automatically generated request ID (Unique Key)(Numeric)
REQUEST_ID : Request ID as maintained in the 311-software database(Unique Key)(Numeric)
CREATED_ON : Date and time the request was created
REQUEST_TYPE : Type of request generated
REQUEST_ORIGIN : Source used to create request
STATUS : Status of the 311 Request(0 = New, 1 = Closed, 3 = Open)(Numeric)
DEPARTMENT : Department assigned the request
NEIGHBORHOOD : Neighborhood where the request is located (when applicable)
COUNCIL_DISTRICT : Council district where the request is located(Numeric)
WARD : Ward where the request is located(Numeric)
TRACT : Tract number of the request's location(Numeric)
PUBLIC_WORKS_DIVISION : Public Works Division where the request is located(Numeric)
PLI_DIVISION : Permits, Licenses & Inspection Division where the request is located(Numeric)
POLICE_ZONE : Police Zone where the request is located(Numeric)
FIRE_ZONE : Fire Zone where incident is located(Numeric)
X : X Coordinate for the request location
Y : Y Coordinate for the request location
GEO_ACCURACY : Category of specificity for the coordinate data, based on request type(Certain request types will contain approximated coordinates to protect the privacy of residents).

Methods

Preprocessing Steps: This dataset has initially 460196 rows and 17 columns, after removing the null valu es, it has 420409 rows and 17 columns. First, I have manipulated the data in the column 'CREATED_ON', which contains the timestamp of the request. The other columns have been created and added to the sa me dataset to get the exact date , day, month, year, etc. This information will be critical to analyze the c all volume on each day, month, year, etc. as well as to predict the future requests.

```
# Extract the information like month, day, and etc from the CREATED_ON

df.loc[:,'CREATED_ON'] = pd.to_datetime(df['CREATED_ON'])

df['MONTH'] = df['CREATED_ON'].dt.month_name()
df['DAY'] = df['CREATED_ON'].dt.day
df['YEAR'] = df['CREATED_ON'].dt.year

df['WEEKDAY'] = df['CREATED_ON'].dt.day_name()
df['HOUR'] = df['CREATED_ON'].dt.hour
df['CREATED_DATE'] = df['CREATED_ON'].dt.date
```

The other anomaly was with the column 'STATUS' that represents whether the request is open, close, or new. it should contain three unique values(0,1,and 3). The outliers values were removed.

```
In [9]: df['STATUS'].value_counts()

Out[9]: 1    355025
        3     39597
        0     25735
        4        52
        Name: STATUS, dtype: int64
```

Data Visualization

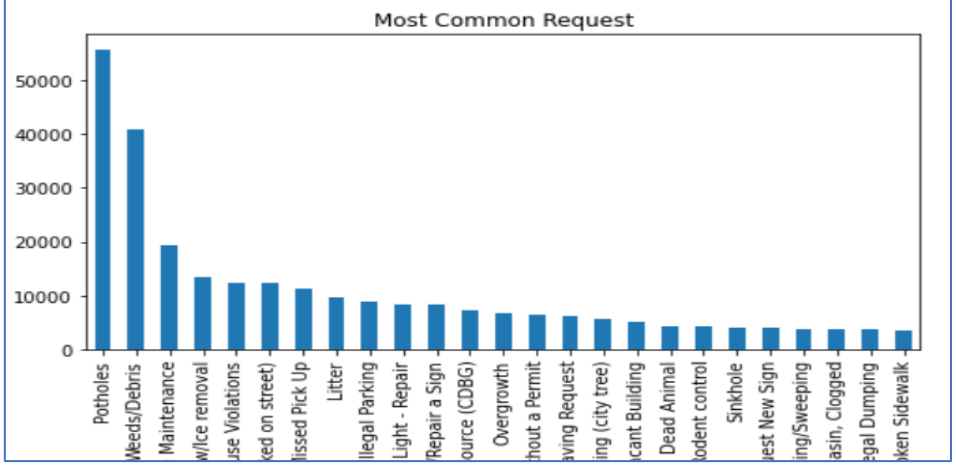
With the help of plotly and matplotlib, many useful observations have inferred.

1. The Most common reasons for people to make complaint are 'Potholes' and Weeds/Debris.

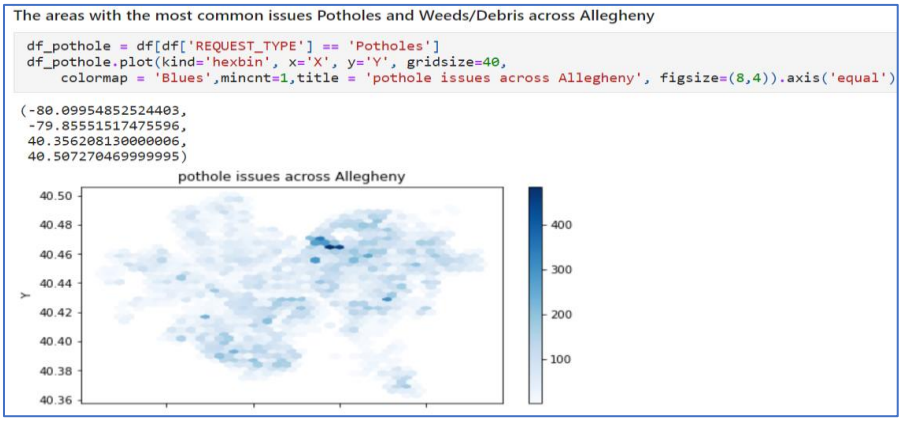
```
# Request Type

(df['REQUEST_TYPE'].value_counts()).head(25).plot(kind='bar',
                    figsize=(8,4), title = 'Most Common Request')
```

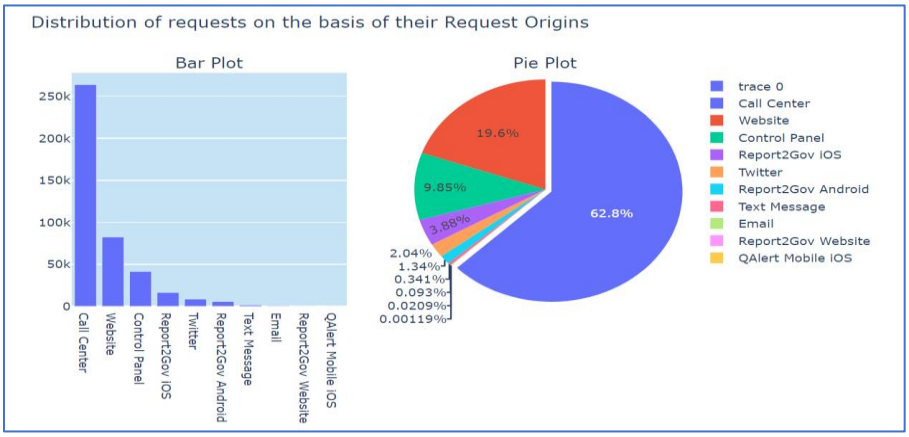
```
<matplotlib.axes._subplots.AxesSubplot at 0x15f9250>
```



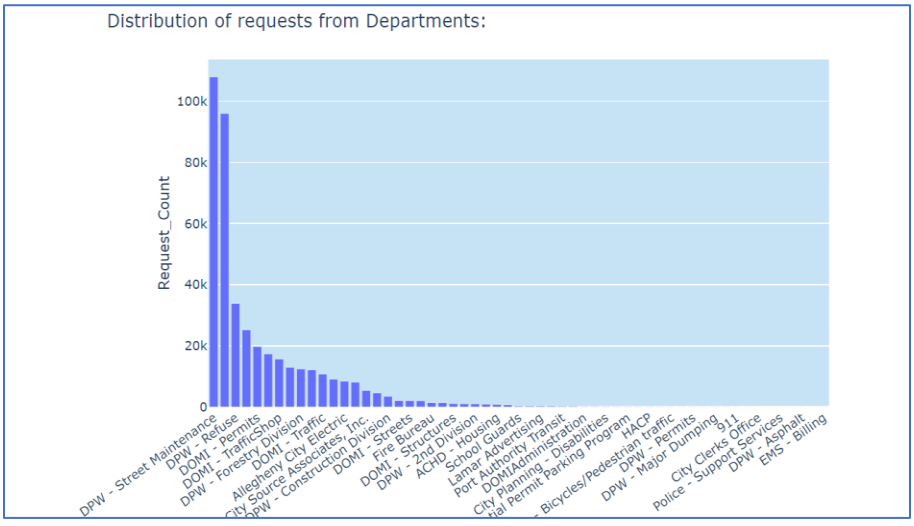
2. With the help of hexbin plot, we can check which area is majorly affected with pothole issue



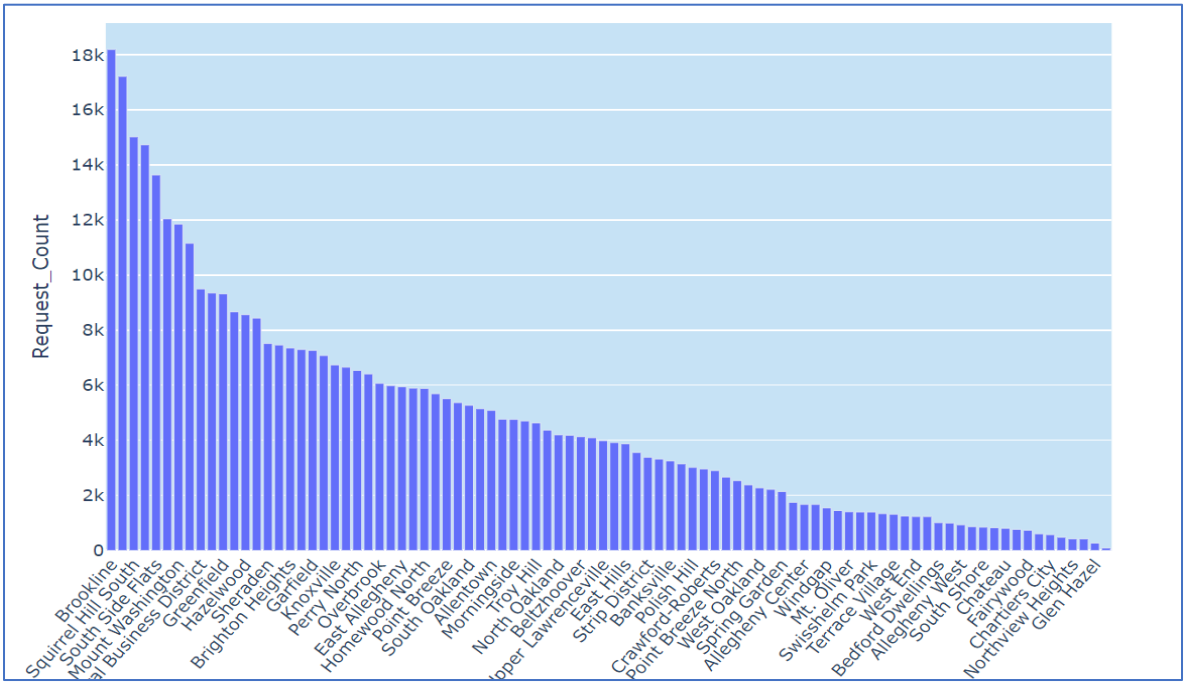
3. Government has provided multiple resource to raise a 311 request like call center, website, control panel, Twitter, etc. we can easily find out that which way is more preferred by the people to make calls. It is evident that 62.8% people like to use the call center. The services like QAlert Mobile iOS can be stopped completely.



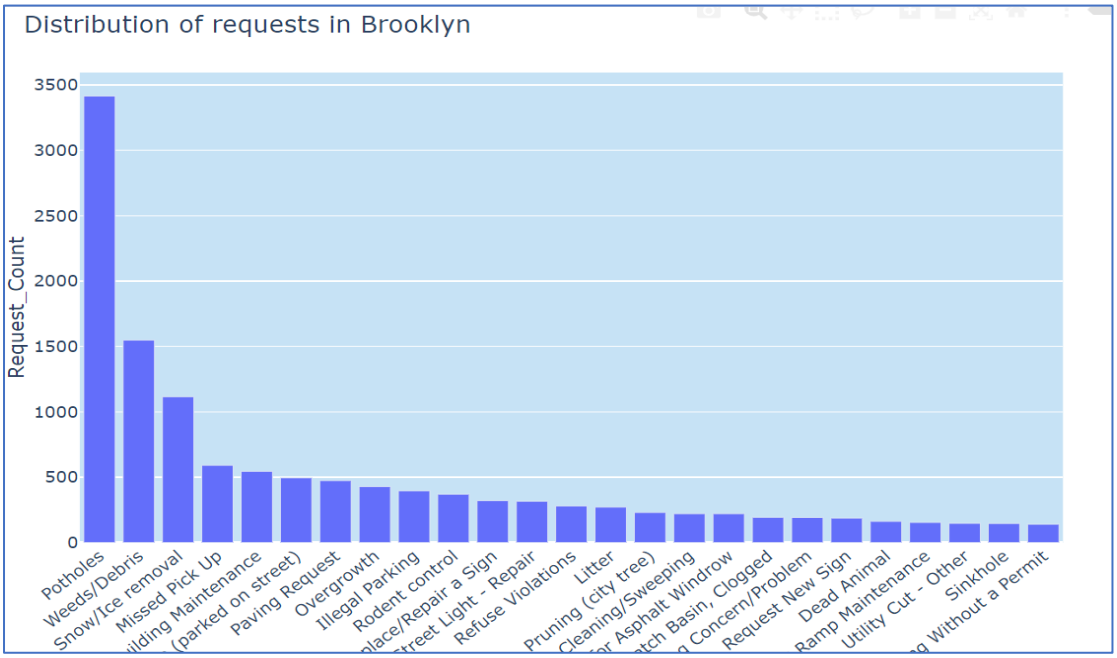
4. Because of the major pothole issue, DPW-street maintenance department has maximum number of requests for them.



5. We can also find out that which neighborhood is getting higher requests than the other. This information would be extremely useful for city planning. Brookline has highest number of requests raised.



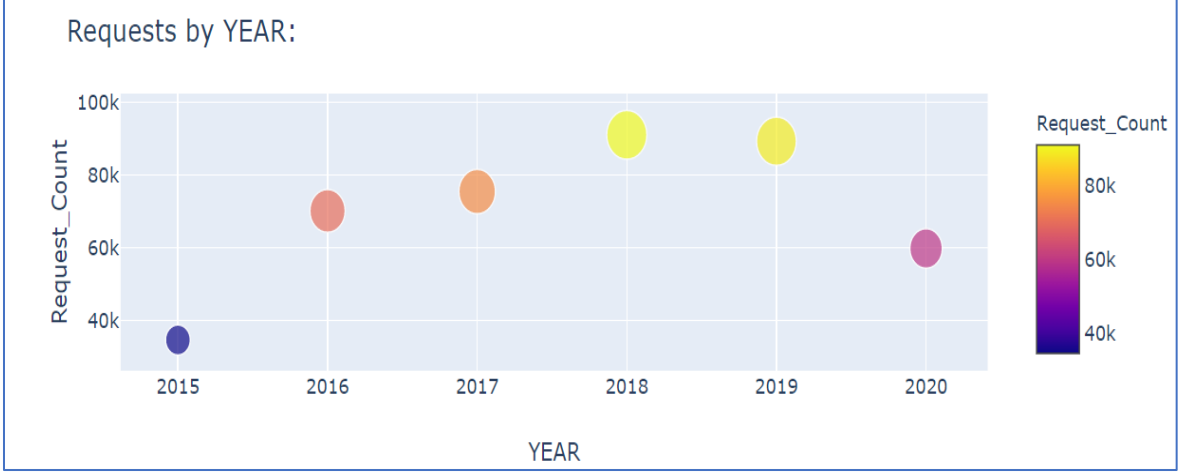
Potholes are the top reason for people to make 311 call in Brookline.

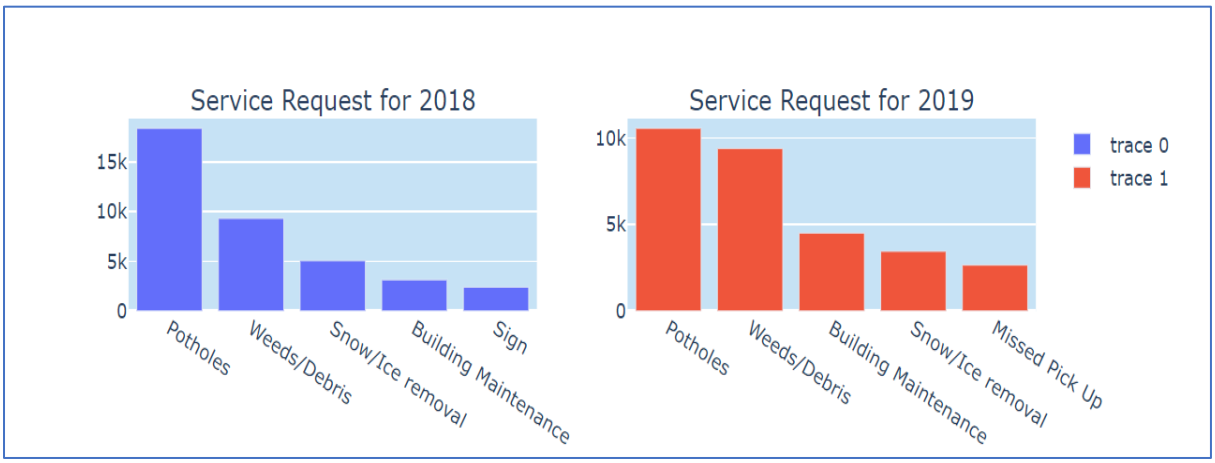


This dataset has columns for police zone, ward, council, etc. , which can provide a lot of useful information about the whole city.

6. The request raised from 2015 to 2020: The maximum number of requests were raised in the year of 2018 and 2019. Potholes and weeds/debris were the reason for high amount of call.

```
df_year = df['YEAR'].value_counts().reset_index()
df_year.columns=['YEAR', 'Request_Count']
df_year.sort_values(by='Request_Count', inplace=True)
fig = px.scatter(df_year, 'YEAR', 'Request_Count', size='Request_Count', color='Request_Count')
fig.update_layout(title='Requests by YEAR:', height=300)
fig.show()
```

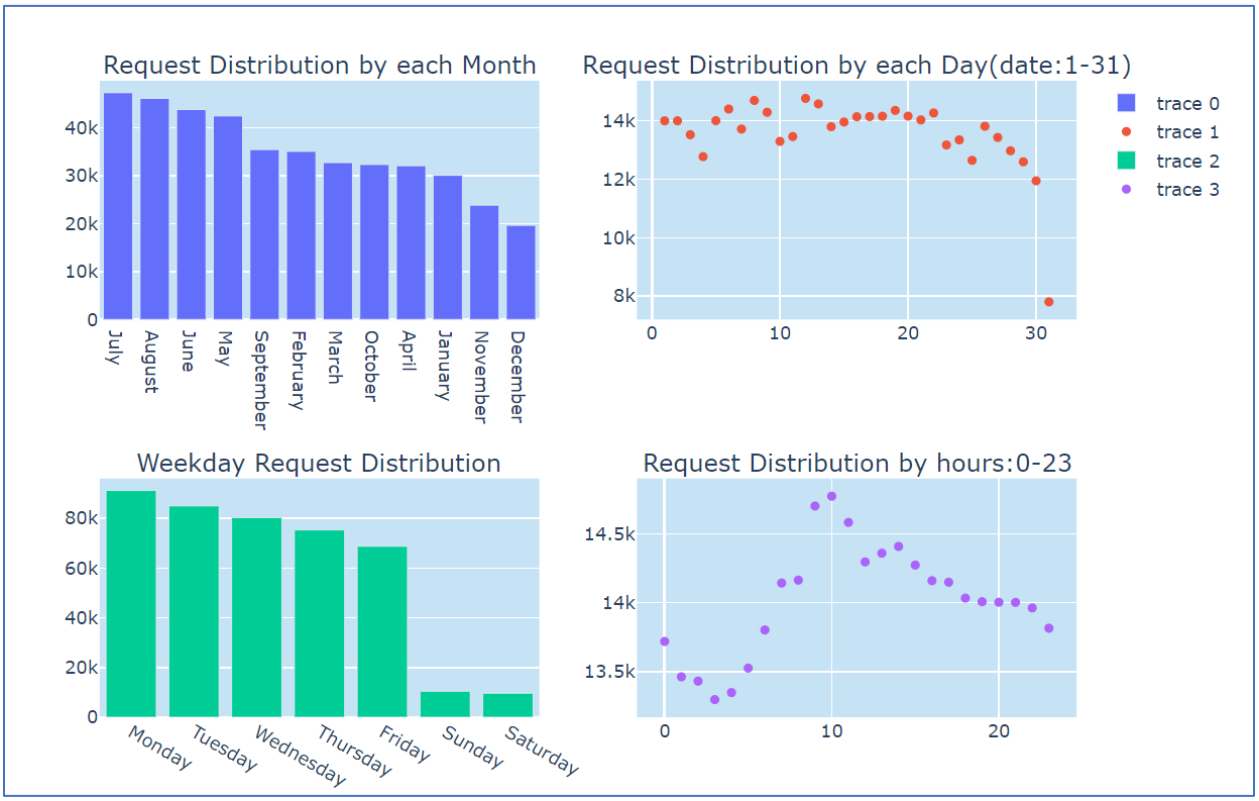




7. The requests distribution by month, day, weekday, and hours:

- July, August, June, and May have maximum number of requests raised, November and December have the minimum number of requests raised.
- Day 31 and 30 have the minimum number of requests raised, after '13'th day of any month, there is a downward trend in number of requests.
- The Minimum number of requests are raised on Saturday and Sunday.
- Number of requests go up as the day begins till the noon(11 am) and then go down significantly till midnight.

This information can be useful to allocate the resources efficiently.



Time series Analysis: To predict the number of requests raised on each day from the data available in other columns.

A time series is a sequence taken at successive points in time. Time Series Forecasting refers to the prediction of values from the identified patterns and analyzing the past data. To process the time series data, dataset's index should be a date field and the other column which will be numerical or categorical values that can be predicted. Dataset has been converted compatible for time series analysis that contains the date and total number of requests raised on the date. ARIMA model is used to predict the request's count for future dates.

```
print(df_time.size)
df_time.head()
```

```
2006
```

CREATED_DATE	Request_Count
2015-02-20	1
2015-04-20	264
2015-04-21	259
2015-04-22	243
2015-04-23	211

ARIMA : Autoregressive Integrated Moving Average (ARIMA) model is a generalization of an autoregressive moving average (ARMA) model. Both models are fitted to time series data either to better understand the data or to predict future points in the series (forecasting). ARIMA models are applied for stationary data, where an initial differencing step (corresponding to the "integrated" part of the model) can be applied one or more times to eliminate the non-stationarity.

The ARIMA model : ARIMA (p ,d, q), where p and q are the levels of AR and MA, respectively.

p is the number of autoregressive terms.

d is the number of nonseasonal differences needed for stationarity.

q is the number of moving averages.

The requirement for ARIMA model is that data should be stationary that means statistical properties such as mean, variance, and standard deviation remain mostly the same over time. There are few methods like statistics summary and Augmented Dickey-Fuller test available as well to test whether the data is stationary or not.

Statistics Summary: We can calculate the moving average(mean), standard deviation, and moving variance and see if it varies with time.

```
#Statistics Summary

# Split the dataset into three parts and calculate statistics
one, two, three = np.split(
        df_time['Request_Count'].sample(
        frac=1), [int(.25*len(df_time['Request_Count'])),
        int(.75*len(df_time['Request_Count']))])


mean1, mean2, mean3 = one.mean(), two.mean(), three.mean()
var1, var2, var3 = one.var(), two.var(), three.var()
std1, std2, std3 = one.std(), two.std(), three.std()
print("Moving average(mean) : ")
print( mean1, mean2, mean3)
print("Moving variance : ")
print(var1, var2, var3)
print("Standard deviation : ")
print(std1, std2, std3)
```

```
Moving average(mean) :
206.14570858283435 208.6729810568295 214.699203187251
Moving variance :
21819.672726546905 20702.36600577509 21939.200757846855
Standard deviation :
147.71483583766022 143.88316790290338 148.11887373946257
```

Augmented Dickey-Fuller test (ADF): This is one of the statistical tests for checking stationarity.
Null Hypothesis (H0): If accepted, it suggests the time series has a unit root, meaning it is non-stationary. It has some time dependent structure. p-value > 0.05: Accept H0, the data has a unit root and is non-stationary

Alternate Hypothesis (H1): The null hypothesis is rejected; it suggests the time series does not have a unit root, meaning it is stationary. p-value ≤ 0.05: Reject H0. the data does not have a unit root and is stationary

```
# Testing for stationarity with Augmented Dickey-Fuller test (ADF)
from statsmodels.tsa.stattools import adfuller

#Augmented Dickey-Fuller test (ADF) function
def ad_test(req_count):
    df_time_ad = adfuller(req_count)
    print("ADF = ", df_time_ad[0])
    print("P-Value = ", df_time_ad[1])
    print("Lags = ", df_time_ad[2])
    print("Number of Observations used for ADF regression and critical values calculation = ", df_time_ad[3])
    print("Critical Values = ")
    for key, val in df_time_ad[4].items():
        print("\t",key, ": ", val)
    if df_time_ad[1] <= 0.05:
        print("Reject H0. the data does not have a unit root and is stationary")
    else:
        print("Accept H0, the data has a unit root and is non-stationary")

# pass the column to ADF function
ad_test(df_time['Request_Count'])
```

```
ADF =  -3.8325157928038323
P-Value =  0.0025916171946846725
Lags =  22
Number of Observations used for ADF regression and critical values calculation =  1983
Critical Values =
         1% :  -3.4336519592295947
         5% :  -2.862998620943585
         10% :  -2.567546508593341
Reject H0. the data does not have a unit root and is stationary
```

we observe that P-Value is less than 0.05. So we can reject the null hypothesis. Time series is stationary.

Looking at the above results the mean and variance of all the three parts are very similar from the second part and third part. It shows the indication of stationary time series data. we observe that P-Value is less than 0.05. So, we can reject the null hypothesis. Time series is stationary.

Forecasting

To predict the request count, we need to find the ARIMA order to build the model. To find the order for ARIMA model. Auto_arima module can be used to get the model.

```
from pmdarima import auto_arima

# Find out the order of ARIMA model
stepwise_fit = auto_arima(df_time['Request_Count'], trace=True, suppress_warnings=True)

stepwise_fit.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=24857.089, Time=3.86 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=26044.881, Time=0.09 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=26021.996, Time=0.18 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=25473.638, Time=0.96 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=26042.882, Time=0.04 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=25083.125, Time=1.90 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=25000.087, Time=1.87 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=24985.952, Time=4.51 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=24917.298, Time=6.64 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=25277.265, Time=2.36 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=24940.496, Time=3.09 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=24990.420, Time=3.76 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=24736.148, Time=7.87 sec
 ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=24007.738, Time=8.06 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=24940.953, Time=6.85 sec
 ARIMA(5,1,3)(0,0,0)[0] intercept   : AIC=23958.651, Time=9.93 sec
 ARIMA(5,1,2)(0,0,0)[0] intercept   : AIC=23889.273, Time=8.97 sec
 ARIMA(5,1,1)(0,0,0)[0] intercept   : AIC=24237.621, Time=4.49 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=24744.984, Time=4.41 sec
 ARIMA(5,1,2)(0,0,0)[0]             : AIC=23887.386, Time=4.11 sec
 ARIMA(4,1,2)(0,0,0)[0]             : AIC=24939.004, Time=4.08 sec
 ARIMA(5,1,1)(0,0,0)[0]             : AIC=24235.618, Time=2.42 sec
 ARIMA(5,1,3)(0,0,0)[0]             : AIC=23955.906, Time=5.31 sec
 ARIMA(4,1,1)(0,0,0)[0]             : AIC=24742.996, Time=1.73 sec
 ARIMA(4,1,3)(0,0,0)[0]             : AIC=23992.076, Time=4.65 sec

Best model:  ARIMA(5,1,2)(0,0,0)[0]
Total fit time: 102.208 seconds
```

The order for our dataset is ARIMA(5,1,2).

After getting the order, we can train the model on training dataset:

```
# import ARIMA
from statsmodels.tsa.arima_model import ARIMA

import warnings
warnings.filterwarnings("ignore")

# Train the model on training dataset
model = ARIMA(df_time_train['Request_Count'], order=(5,1,2))
model = model.fit()
```

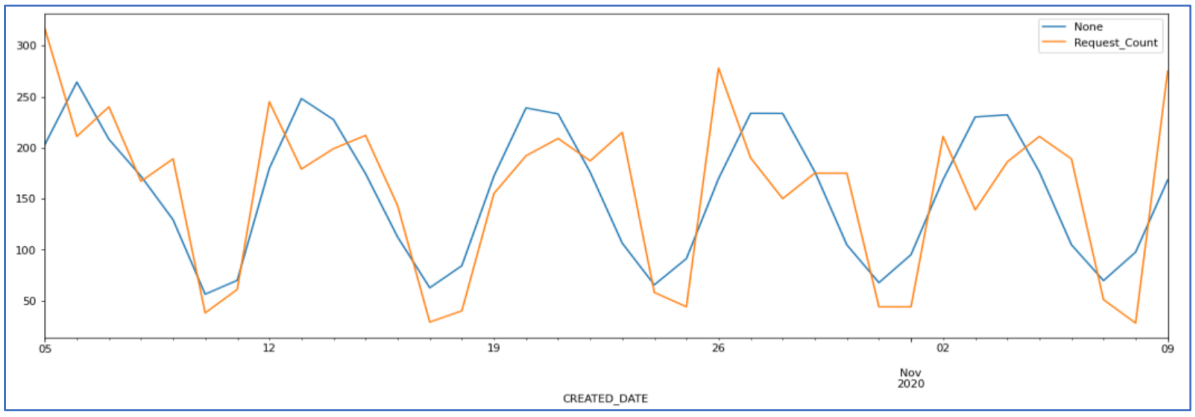
```
# Perform predict on training set

start = len(df_time_train)
end = len(df_time_train) + len(df_time_test) -1
forecast = model.predict(start=start, end=end, typ='levels')
forecast.index=df_time.index[start:end+1]
print(forecast)

forecast.plot(legend=True)
df_time_test['Request_Count'].plot(legend=True, figsize=(18,6))
```

Output: These are the predicted values and let us compare it with the original value.

```
CREATED_DATE
2020-10-05    202.744973
2020-10-06    264.253106
2020-10-07    208.190207
2020-10-08    172.037963
2020-10-09    129.322663
2020-10-10     56.299084
2020-10-11     69.955606
2020-10-12    179.620479
2020-10-13    248.098620
2020-10-14    227.453364
2020-10-15    174.435793
2020-10-16    112.300511
2020-10-17     62.616233
2020-10-18     84.242068
2020-10-19    172.078961
2020-10-20    238.925702
2020-10-21    233.074205
2020-10-22    175.954943
2020-10-23    106.338428
2020-10-24     65.516806
2020-10-25     91.243577
2020-10-26    169.584690
2020-10-27    233.609602
2020-10-28    233.530173
2020-10-29    176.438316
2020-10-30    104.728026
2020-10-31     67.638934
2020-11-01     95.081989
2020-11-02    168.729319
2020-11-03    230.149930
2020-11-04    232.158009
2020-11-05    176.231404
2020-11-06    104.796569
2020-11-07     69.652332
2020-11-08     97.577311
2020-11-09    168.360186
dtype: float64
```



The same model can be used to predict the request count for future dates as well.

```
# model the whole data set
model_df = ARIMA(df_time['Request_Count'], order=(5,1,2))
model_f= model_df.fit()

# Predict the future dates

future_dates = pd.date_range(start='2020-11-09', end='2020-12-09')
forecast_df = model_f.predict(start=len(df_time), end=len(df_time)+ 30, typ='levels').rename('ARIMA prediction for 311 Requests for next 30 days')
forecast_df.index = future_dates
print(forecast_df)

forecast_df.plot(legend=True, figsize=(18,6))
```

Random Forest Classifier: To predict the NEIGHBORHOOD based on the data present in other columns. Random forest is a supervised learning algorithm which is used for both classification as well as regression. It creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution.

For the maximum accuracy of the algorithm, we need to find out that which column will be helpful in predicting the target variable. These columns will not play an important role to predict the target variable.

```
# drop irrelavent columns
cols_to_drop = ['CREATED_ON', 'REQUEST_ID', 'DEPARTMENT', 'REQUEST_TYPE', 'REQUEST_ORIGIN', 'REQUEST_ID','HOUR', 'DAY','STATUS',
                'FIRE_ZONE', 'CREATED_DATE', 'WEEKDAY', 'MONTH','YEAR','GEO_ACCURACY']
df = df.drop(cols_to_drop, axis = 1)
```

Our target variable is object type, for random classifier algorithm to work, all the columns should be numeric type. We can use label encoder to convert the value into numeric and decode it later.

```
# All the columns are numeric type except target variable, no need for dummies.
# We need to encode the target variable into a numeric value

from sklearn.preprocessing import LabelEncoder

lab = LabelEncoder()
df['NEIGHBORHOOD'] = lab.fit_transform(df['NEIGHBORHOOD'])

keys = lab.classes_
values = lab.transform(lab.classes_)
dict_keys = dict(zip(keys, values))
print(dict_keys)
```

```
{'Allegheny Center': 0, 'Allegheny West': 1, 'Allentown': 2, 'Arlington': 3, 'Arlington Heights': 4, 'Banksville': 5, 'Bedford Dwellings': 6, 'Beechview': 7, 'Beltzhoover': 8, 'Bloomfield': 9, 'Bluff': 10, 'Bon Air': 11, 'Brighton Heights': 12, 'Brookline': 13, 'California-Kirkbride': 14, 'Carrick': 15, 'Central Business District': 16, 'Central Lawrenceville': 17, 'Central Northside': 18, 'Central Oakland': 19, 'Chartiers City': 20, 'Chateau': 21, 'Crafton Heights': 22, 'Crawford-Roberts': 23, 'Duquesne Heights': 24, 'East Allegheny': 25, 'East Carnegie': 26, 'East Hills': 27, 'East Liberty': 28, 'Elliott': 29, 'Esplen': 30, 'Fairywood': 31, 'Fineview': 32, 'Friendship': 33, 'Garfield': 34, 'Glen Hazel': 35, 'Greenfield': 36, 'Hays': 37, 'Hazelwood': 38, 'Highland Park': 39, 'Homewood North': 40, 'Homewood South': 41, 'Homewood West': 42, 'Knoxville': 43, 'Larimer': 44, 'Lincoln Place': 45, 'Lincoln-Lemington-Belmar': 46, 'Lower Lawrenceville': 47, 'Manchester': 48, 'Marshall-Shadeland': 49, 'Middle Hill': 50, 'Morningside': 51, 'Mount Washington': 52, 'Mt. Oliver': 53, 'New Homestead': 54, 'North Oakland': 55, 'North Shore': 56, 'Northview Heights': 57, 'Oakwood': 58, 'Overbrook': 59, 'Perry North': 60, 'Perry South': 61, 'Point Breeze': 62, 'Point Breeze North': 63, 'Polish Hill': 64, 'Regent Square': 65, 'Ridgemont': 66, 'Shadyside': 67, 'Sheraden': 68, 'South Oakland': 69, 'South Shore': 70, 'South Side Flats': 71, 'South Side Slopes': 72, 'Spring Garden': 73, 'Spring Hill-City View': 74, 'Squirrel Hill North': 75, 'Squirrel Hill South': 76, 'St. Clair': 77, 'Stanton Heights': 78, 'Strip District': 79, 'Summer Hill': 80, 'Swisshelm Park': 81, 'Terrace Village': 82, 'Troy Hill': 83, 'Upper Hill': 84, 'Upper Lawrenceville': 85, 'West End': 86, 'West Oakland': 87, 'Westwood': 88, 'Windgap': 89}
```

After splitting the dataset into training and testing dataset, we can create the model and fit our training data:

Training the RandomForestClassifier Algorithm

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=10, random_state=30)
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

#the values in y_test and y_pred should be same for the best result
```

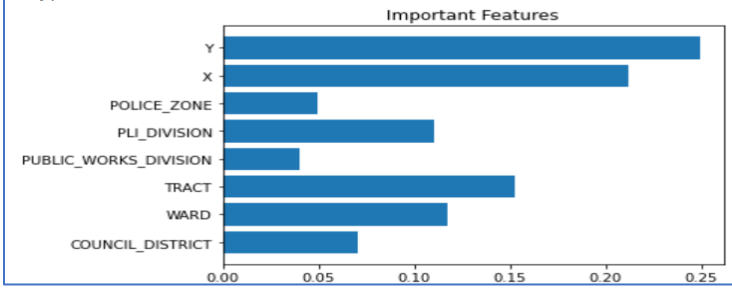
We can check which columns will be important to predict the neighborhood, Y and X are the important features to decide on NEIGHBORHOOD

```
# Feature Importance

feature_list = list(x.columns)
feature_imp = pd.Series(model.feature_importances_, index=feature_list).sort_values(ascending=False)

plt.barh(feature_list, model.feature_importances_)
plt.title('Important Features')
feature_imp
```

```
Y                       0.248891
X                       0.211554
TRACT                   0.152663
WARD                    0.116990
PLI_DIVISION            0.110323
COUNCIL_DISTRICT        0.070165
POLICE_ZONE             0.049409
PUBLIC_WORKS_DIVISION   0.040005
dtype: float64
```



Output: We can add the predicted value to test dataset to compare with the actual value of Neighborhood**.**

```
# we need to inverse the encoder to see the actual value of neighborhood
y_pred = lab.inverse_transform(y_pred)
y_test = lab.inverse_transform(y_test)
```

```
# Add the column to test data setwith predicted and actual data
x_test['Predicted Neighborhood'] = y_pred
x_test['Actual Neighborhood'] = y_test
```

```
x_test.head()
```

:T	WARD	TRACT	PUBLIC_WORKS_DIVISION	PLI_DIVISION	POLICE_ZONE	X	Y	Predicted Neighborhood	Actual Neighborhood
.0	14.0	4.200314e+10	3.0	14.0	4.0	-79.920938	40.441697	Squirrel Hill North	Squirrel Hill North
.0	32.0	4.200332e+10	3.0	32.0	3.0	-79.996296	40.394474	Carrick	Carrick
.0	29.0	4.200329e+10	3.0	29.0	3.0	-79.984615	40.383630	Carrick	Carrick
.0	24.0	4.200324e+10	1.0	24.0	1.0	-79.991754	40.460291	Spring Garden	Spring Garden
.0	17.0	4.200317e+10	3.0	17.0	3.0	-79.990909	40.426104	South Side Slopes	South Side Slopes

```
# Save the Final file with predicted and actual data.
x_test.to_csv('final_result.csv')
```

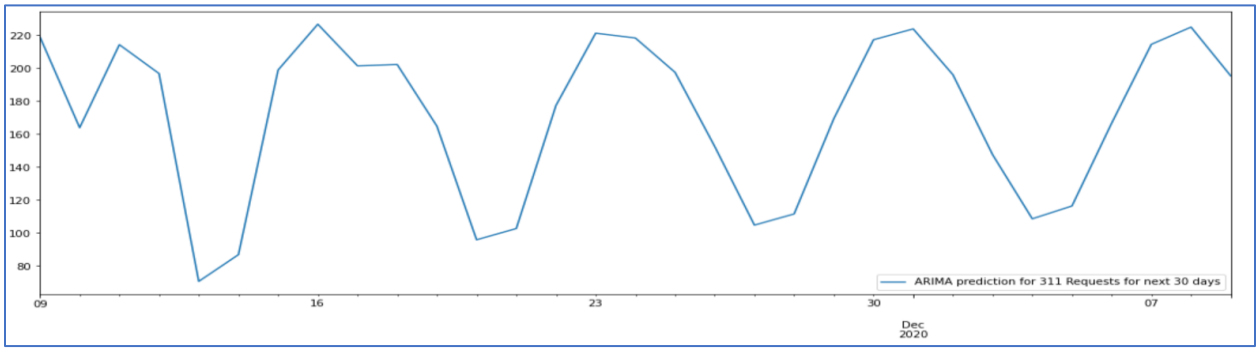
Results

In the Analysis of using ARIMA and Random forest classifier algorithms, the performance of the algorithm is based on the accuracy and precision. The final goal is to achieve a model with high accuracy. That totally depends the quality of data. Data cleaning and removal of noise will lead the data to perform well with these models.

Time Series Analysis(ARIMA) To predict the number of calls	To predict the number of requests raised on each day ARIMA model is used.
	Accuracy : root mean square error: 57.57 with mean 157.
	The prediction of future 30 days is also added to the notebook.
	AIC: 23887.386
	Best Model: (5,1,2)
Random Forest Classifier To predict the neighborhood	To predict the neighborhood based on other information
	Precision = 0.99
	Recall = 0.99
	f1-score = 0.99
	Accuracy= 0.9967567481840962

Discussion and Conclusion

As per the results, both the algorithms worked well with the dataset. The predicted values of call for each day is satisfactory. However, I would like to test the dataset with other time series forecasting methods such as Vector autoregression or Simple exponential smoothing as well. The results we obtained from data visualization are extremely useful to allocate the resources efficiently by only looking at the requests distribution throughout the month, day, year, or hour data. This is the graph for predicted calls by ARIMA model.



Random forest classifier algorithm for predicting the categorical column 'NEIGHBORHOOD' worked wonderfully with 99% accuracy. The challenge was model did not work well with the other features. It would be a futuristic approach to build the model which can predict the 'NEIGHBORHOOD' based on the request type. This model is using X,Y coordinates, and other features to make prediction. While using the column REQUEST_TYPE with 323 unique values as the random forest algorithm only works with numerical columns, we need to change these columns to dummy columns containing the numerical value. That means creating 323 dummy columns in the dataset. That could be a challenge on space complexity.

References:

1. https://catalog.data.gov/dataset/311-data-in-development
2. https://www.govtech.com/dc/articles/What-is-311.html
3. http://eds.b.ebscohost.com.proxy-harrisburg.klnpa.org/eds/pdfviewer/pdfviewer?vid=2&sid=321bd056-0a9b-4c1a-886c-0e11ea75fb04%40pdc-v-sessmgr05
4. https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average
5. https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_classification_algorithms_random_forest.htm
6. https://en.wikipedia.org/wiki/Random_forest
7. https://en.wikipedia.org/wiki/Time_series