# Week2_Capstone_Project_Final

June 4, 2020

### 0.1 # Capstone Project: The Battle of Neighborhoods

## San Francisco Police Department Incident Reports: 2018 to Present(01/06/2020) ### Author : Jithin Prakash Kolamkolly ***

Police Department is collating data about the criminal incidents happening in and arounf San Francisco, that data is published in 'The office of the chief Data Officer – City and County of San Francisco' (https://data.sfgov.org/Public-Safety/Police-Department-Incident-Reports-2018-to-Present/wg3w-h783) website. This data is directly pulled from the website to do modelling and to analyze the data

**CRISP DM** methodology is used in analysis and prediction

- Collecting Data
- Explore and Understand Data
- Data Preparation and pre-processing
- Modelling
- Evaluation and testing

### 0.2 #### Importing Libraries

[0]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

# 1 1. Collect Data

**Importing Data from website (https://data.sfgov.org/api/views/wg3w-h783/rows.csv)**

[0]:
```
!wget -q -O 'Police_incidents.csv' https://data.sfgov.org/api/views/wg3w-h783/
 ↪rows.csv
print('Data fetched from website')
```

```
Data fetched from website
```

---

# 2 2. Explore and Understand Data

Reading CSV to Data Frame and performing operations

[0]:
```
#Reading from the CSV file to the data frame
df_Police = pd.read_csv('Police_incidents.csv')
#Printing the shape of the Raw Data
print('\nRaw data has %d Rows and %d Columns\n'% df_Police.shape)
#Displaying the first 5 rows
df_Police.head()
```

```
Raw data has 351980 Rows and 36 Columns
```

[0]:
```
        Incident Datetime  ... Areas of Vulnerability, 2016
0  2019/05/01 01:00:00 AM  ...                          1.0
1  2019/06/22 07:45:00 AM  ...                          2.0
2  2019/06/03 04:16:00 PM  ...                          2.0
3  2018/11/16 04:34:00 PM  ...                          2.0
4  2019/05/27 02:25:00 AM  ...                          1.0

[5 rows x 36 columns]
```

[0]:
```
#Check for Duplicate Entries
df_Police.duplicated().sum()
```

[0]: 0

[0]:
```
#Check the details using describe method
df_Police.describe()
```

[0]:
```
       Incident Year  ...  Areas of Vulnerability, 2016
count  351980.000000  ...                 333307.000000
mean     2018.703597  ...                      1.549637
std         0.699340  ...                      0.497531
min      2018.000000  ...                      1.000000
25%      2018.000000  ...                      1.000000
50%      2019.000000  ...                      2.000000
75%      2019.000000  ...                      2.000000
max      2020.000000  ...                      2.000000

[8 rows x 20 columns]
```

---

# 3 3. Data Preparation and Pre-processing

Refer the documentation for the Column Header / Feature / Attribute description

[0]:
```
#Dropping the data that is not needed for the analysis
df_Police.drop(['Report Datetime','Row ID','Incident ID','Incident Number',
```

```
                'CAD Number','Report Type Code','Report Type␣
↪Description','Filed Online','Incident Code',
                'Incident Subcategory','CNN','Supervisor District','point','SF␣
↪Find Neighborhoods','Current Police Districts',
                'Current Supervisor Districts','Analysis Neighborhoods','HSOC␣
↪Zones as of 2018-06-05','OWED Public Spaces',
                'Central Market/Tenderloin Boundary Polygon - Updated','Parks␣
↪Alliance CPSI (27+TL sites)',
                'ESNCAG - Boundary File','Areas of Vulnerability,␣
↪2016'],axis=1,inplace=True)

#Printing the shape of the Processed Data
print('\nColumn Drop - Data has %d Rows and %d Columns'% df_Police.shape)

#Dropping the Rows that has NaN or Null values
df_Police.dropna(subset = ['Incident Datetime','Incident Date','Incident␣
↪Time','Incident Year','Incident Day of Week',
                           'Incident Category','Incident␣
↪Description','Resolution','Intersection',
                           'Police District','Analysis␣
↪Neighborhood','Latitude','Longitude'],inplace=True,axis=0)

# Adding Month Column on to the DataFrame
df_Police.insert(3,'Incident Month',df_Police['Incident Date'].apply(lambda x:␣
↪str(x.split("/")[1])))

#Printing the shape of the Processed Data
print('\nRow Drop - Data has %d Rows and %d Columns\n'% df_Police.shape)

df_Police.head()
```

```
Column Drop - Data has 351980 Rows and 13 Columns

Row Drop - Data has 333185 Rows and 14 Columns
```

[0]:

```
         Incident Datetime Incident Date  ...   Latitude   Longitude
0  2019/05/01 01:00:00 AM    2019/05/01  ...  37.762569 -122.499627
1  2019/06/22 07:45:00 AM    2019/06/22  ...  37.780535 -122.408161
2  2019/06/03 04:16:00 PM    2019/06/03  ...  37.721600 -122.390745
3  2018/11/16 04:34:00 PM    2018/11/16  ...  37.794860 -122.404876
4  2019/05/27 02:25:00 AM    2019/05/27  ...  37.797716 -122.430559

[5 rows x 14 columns]
```

**Crimes per Category**

Lets check the number of crimes by category

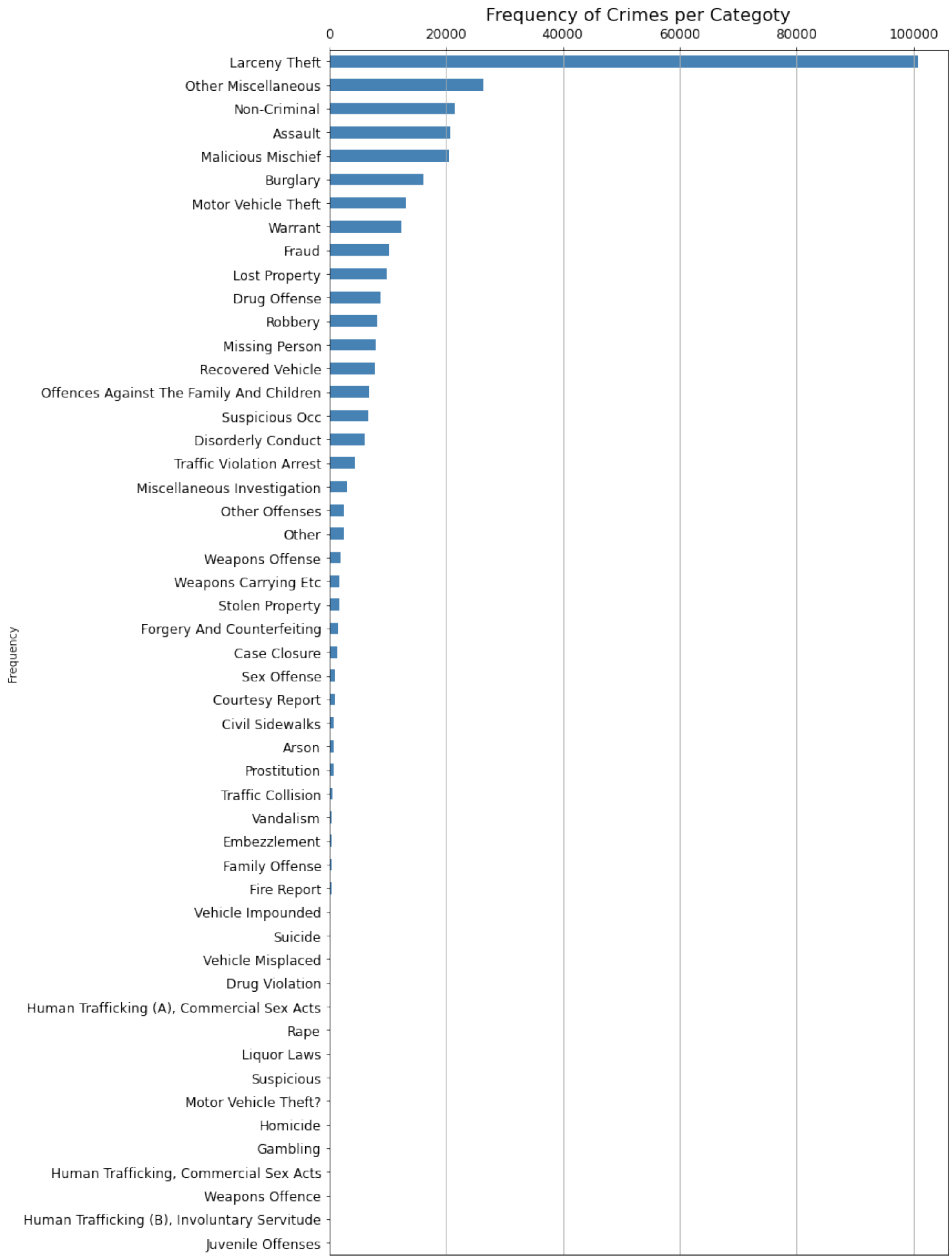
[0]:
```
df_category = pd.DataFrame(df_Police['Incident Category'].value_counts())
df_category=df_category.reset_index().rename(columns={'index' :'Incident␣
↪Category','Incident Category':'Incident Count'})
df_category.head(15)
```

[0]:

| | Incident Category | Incident Count |
|---|---|---|
| 0 | Larceny Theft | 100845 |
| 1 | Other Miscellaneous | 26326 |
| 2 | Non-Criminal | 21453 |
| 3 | Assault | 20712 |
| 4 | Malicious Mischief | 20516 |
| 5 | Burglary | 16200 |
| 6 | Motor Vehicle Theft | 13056 |
| 7 | Warrant | 12387 |
| 8 | Fraud | 10246 |
| 9 | Lost Property | 9955 |
| 10 | Drug Offense | 8792 |
| 11 | Robbery | 8244 |
| 12 | Missing Person | 8009 |
| 13 | Recovered Vehicle | 7753 |
| 14 | Offences Against The Family And Children | 6815 |

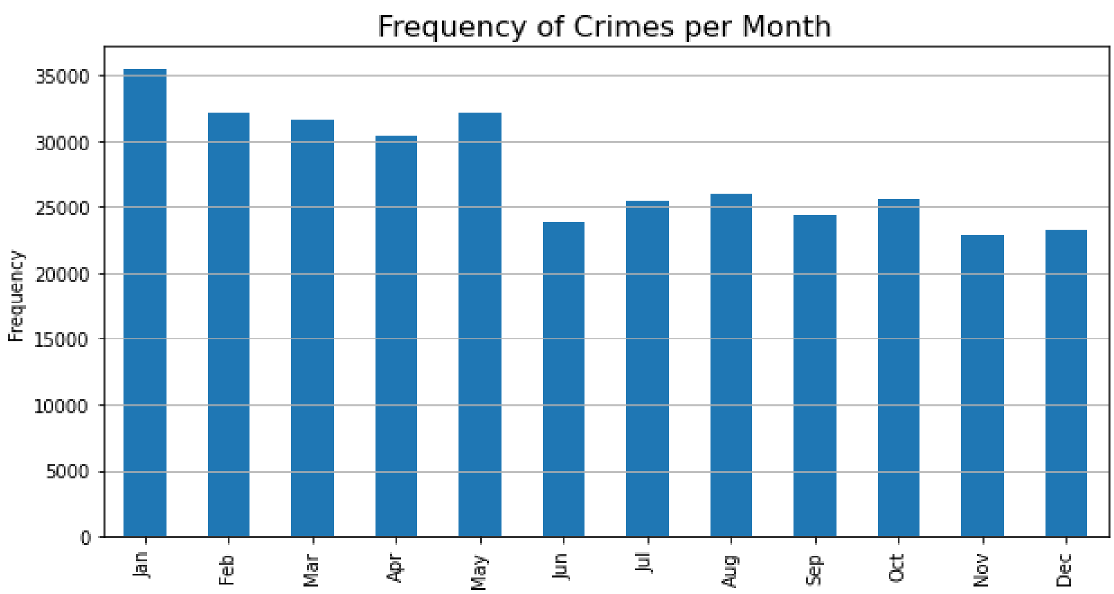
Plotting the data to see the frequency of all crime categories

[0]:
```
#df_category.sort_values('Incident Category',ascending=True,inplace=True)

ax = df_category.plot(kind='barh',figsize=(10,20),color='steelblue')
ax.set_title('Frequency of Crimes per Categoty',fontsize=16)
ax.set_yticklabels( df_category['Incident Category'],fontsize=10)
ax.set_ylabel('Frequency')
ax.tick_params(labelsize=12)

ax.invert_yaxis()
ax.xaxis.tick_top()

ax.xaxis.grid(True)
ax.get_legend().remove()

#plt.savefig('plt_1.png',bbox_inches='tight')
```




**Time Crime Incident - Year/Month/Time/Day analysis Month**

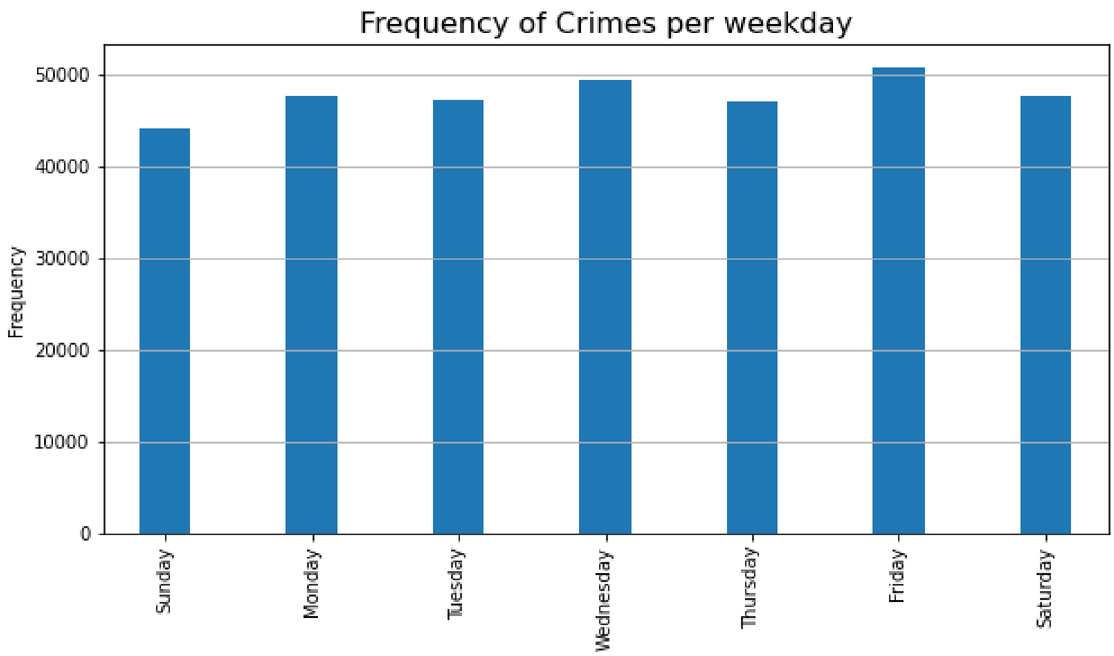
[0]:
```
import calendar

df_month = pd.DataFrame(df_Police['Incident Month'].value_counts()).
↪reset_index()
df_month.sort_values('index',ascending=True,inplace=True)

ax = df_month.plot(kind='bar',figsize=(10,5))
ax.set_title('Frequency of Crimes per Month',fontsize=16)
ax.set_xticklabels( df_month['index'].apply(lambda x: calendar.
↪month_abbr[int(x)]),fontsize=10)
ax.set_ylabel('Frequency')
ax.tick_params(labelsize=10)

ax.yaxis.grid(True)
ax.get_legend().remove()
```




**Day**

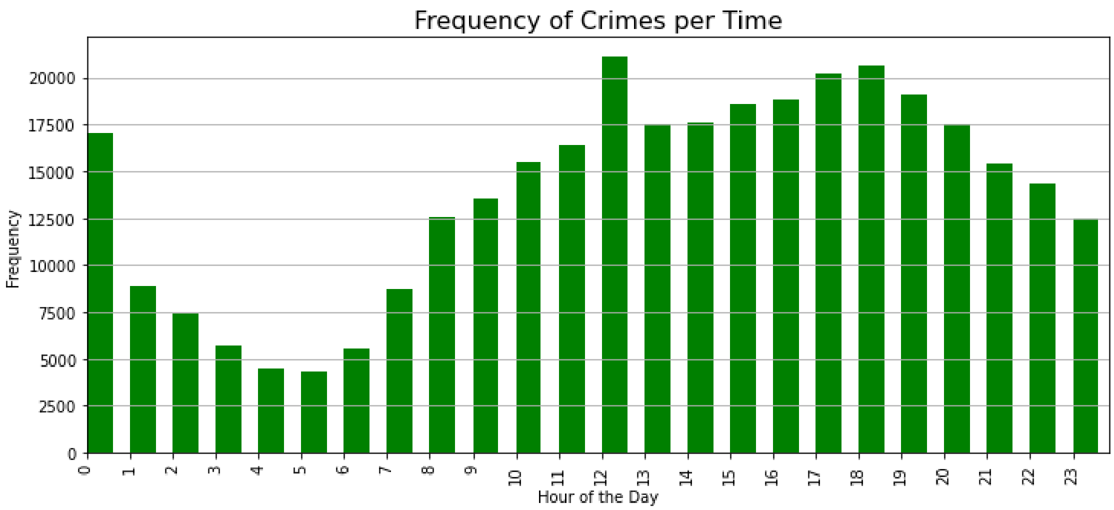
[0]:
```
day_seq = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',␣
↪'Saturday']
df_day = pd.DataFrame(df_Police['Incident Day of Week'].value_counts()).
↪reset_index()
df_day.sort_values('index',ascending=True,inplace=True)
df_day['index'] = pd.Categorical(df_day['index'], categories=day_seq,␣
↪ordered=True)
df_day.sort_values('index',inplace=True)

ax = df_day.plot(kind='bar',figsize=(10,5),width=.35)
```

```
ax.set_title('Frequency of Crimes per weekday',fontsize=16)
ax.set_xticklabels( df_day['index'],fontsize=10)
ax.set_ylabel('Frequency')
ax.tick_params(labelsize=10)

ax.yaxis.grid(True)
ax.get_legend().remove()
```



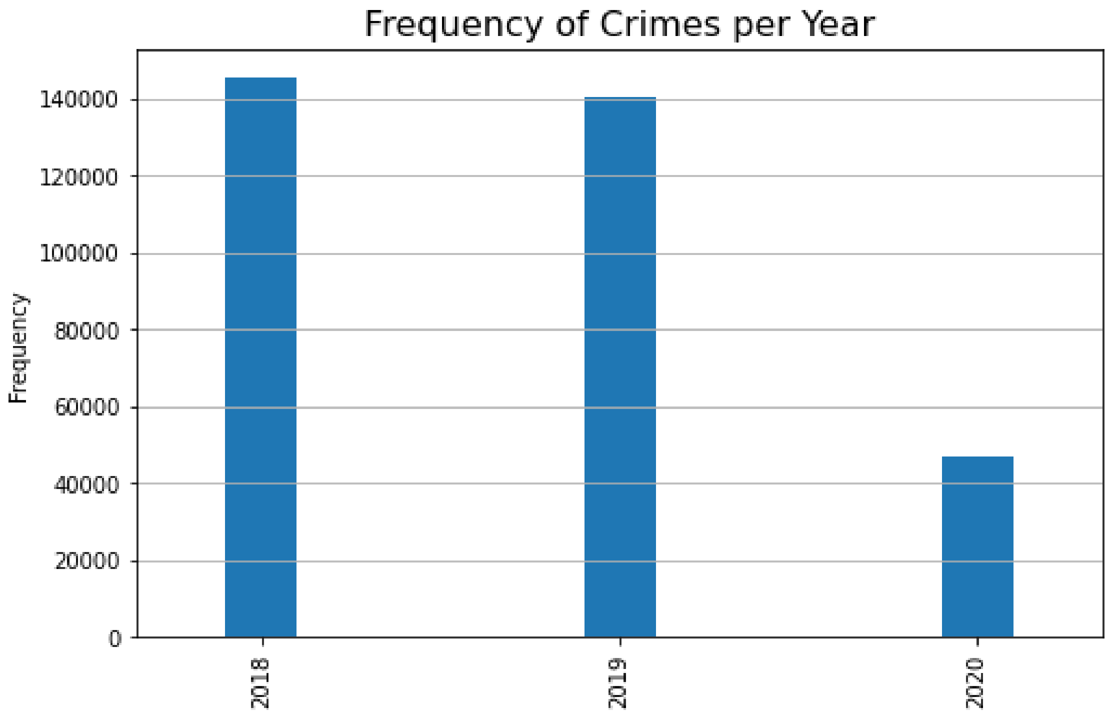

### Time of the Day

[0]:
```
df_time = pd.DataFrame(pd.to_datetime(df_Police['Incident Time'],␣
 ↪errors='coerce').dt.hour)
df_time = pd.DataFrame(df_time['Incident Time'].value_counts()).reset_index().
 ↪sort_values('index')

ax = df_time.plot(kind='bar',figsize=(12,5),color='g',width = 1.2)
ax.set_title('Frequency of Crimes per Time',fontsize=16)
ax.set_xticklabels( df_time['index'],fontsize=10)
ax.tick_params(labelsize=10)
ax.set_xlabel('Hour of the Day',fontsize=10)
ax.set_ylabel('Frequency',fontsize=10)
ax.set_xlim(0)
ax.yaxis.grid(True)
ax.get_legend().remove()
```




**Year**

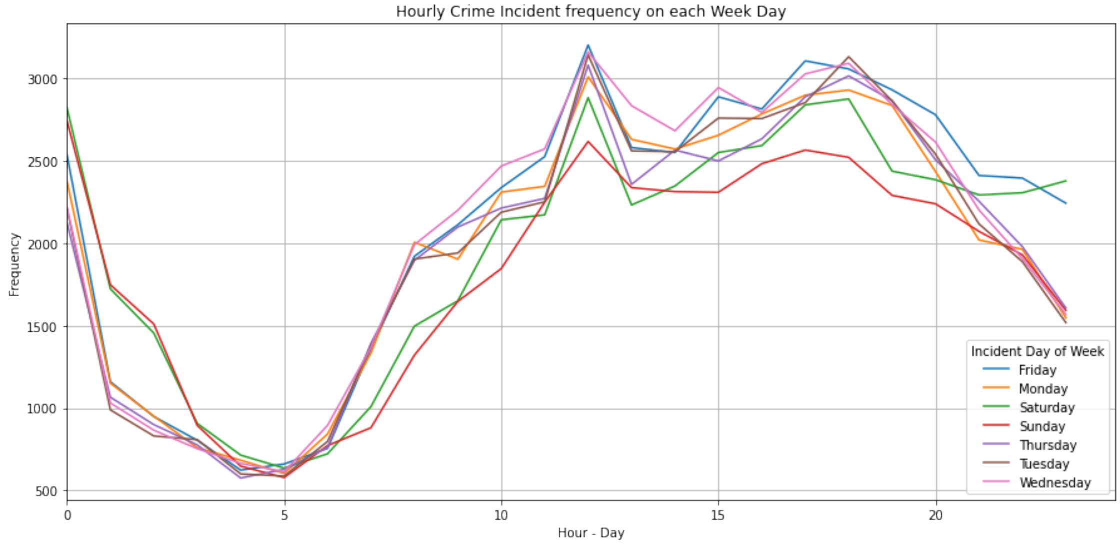
[0]:
```
df_year = pd.DataFrame(df_Police['Incident Year'].value_counts())

ax = df_year.plot(kind='bar',figsize=(8,5),width=0.2,align='center')
ax.set_title('Frequency of Crimes per Year',fontsize=16)
ax.set_xticklabels( df_year.index,fontsize=10)
#ax.set_xlabel('Year',fontsize=10)
ax.set_ylabel('Frequency')
ax.tick_params(labelsize=10)

ax.yaxis.grid(True)
ax.get_legend().remove()
```




**Crime Frequency on Different WeekDays**

[0]:
```
df_time_Day = df_Police[['Incident Time','Incident Day of Week']]
df_time = pd.DataFrame(pd.to_datetime(df_Police['Incident Time'],␣
 ↪errors='coerce').dt.hour).rename(columns={'Incident Time':'TimeH'})
df_time_Day = pd.concat([df_time_Day,df_time],axis=1)
```

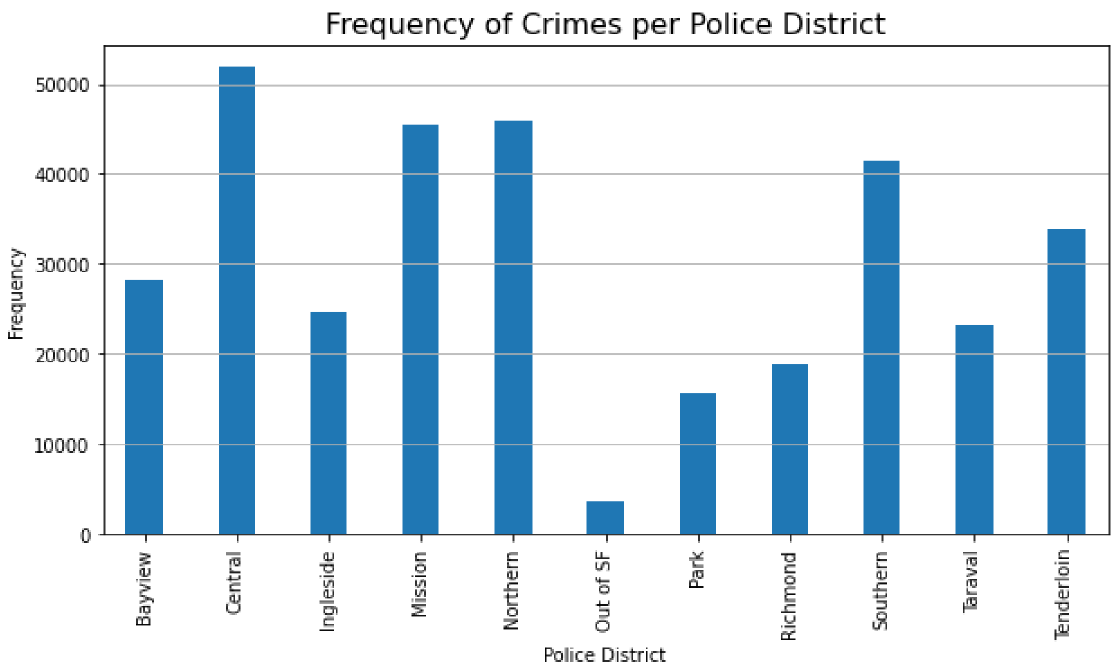
[0]:
```
df_time_Day_group= df_time_Day.groupby(['TimeH','Incident Day of Week']).
 ↪count()['Incident Time'].unstack()
fig, ax = plt.subplots(figsize=(15,7))
df_time_Day_group.plot(ax=ax)
ax.set_title('Hourly Crime Incident frequency on each Week Day')
ax.set_xlabel('Hour - Day')
ax.set_ylabel('Frequency')
ax.set_xlim(0)
ax.grid()
```




---

**Crime per Police District**

[0]:
```
df_district =pd.DataFrame(df_Police['Police District'].value_counts()).
 ↪reset_index()
df_district.sort_values('index',inplace=True)
df_district

ax = df_district.plot(kind='bar',figsize=(10,5),width=0.4)
ax.set_title('Frequency of Crimes per Police District',fontsize=16)
ax.set_xticklabels( df_district['index'],fontsize=10)
ax.set_xlabel('Police District',fontsize=10)
ax.set_ylabel('Frequency')
ax.tick_params(labelsize=10)

ax.yaxis.grid(True)
ax.get_legend().remove()
```




---

#### 3.0.1 Populate a word-cloud to understand the most occuring crime types

[50]:
```
#Install Word-Cloud if not available
!conda install -c conda-forge wordcloud --yes

from wordcloud import WordCloud
from wordcloud import STOPWORDS

stopWords = set(STOPWORDS)
stopWords.add('Found')

wc = WordCloud(background_color = 'white', max_words=20000, width = 1000,␣
↪height =700, stopwords=stopWords)
wc.generate(str(df_Police['Incident Description']))
plt.figure(figsize=(15,8))
plt.imshow(wc,interpolation='bilinear')
plt.axis('off')
plt.title('Description of the Crime\n', fontsize = 15)

plt.show()
```

```
/bin/bash: conda: command not found
```


---

# 4 4. Modelling

**Pre-Processing**

Filtering relevant features to X and Y arrays

[0]:
```
X = df_Police[['Incident Datetime','Police District','Latitude','Longitude']].
 ↪values
y = df_Police['Incident Category'].values
print(X[:5])
print(y[:5])
```

```
[['2019/05/01 01:00:00 AM' 'Taraval' 37.76256939715695
  -122.49962745519908]
 ['2019/06/22 07:45:00 AM' 'Southern' 37.7805353858225
  -122.40816079455212]
 ['2019/06/03 04:16:00 PM' 'Bayview' 37.72159985216247
  -122.39074534279013]
 ['2018/11/16 04:34:00 PM' 'Central' 37.79485953222834
```

```
  -122.40487561154785]
 ['2019/05/27 02:25:00 AM' 'Northern' 37.79771621229674
  -122.43055896140595]]
['Offences Against The Family And Children' 'Non-Criminal'
 'Missing Person' 'Offences Against The Family And Children' 'Assault']
```

Pre processing the Data - Features in X

[0]:
```python
from sklearn.preprocessing import LabelEncoder

Label_Encoder = LabelEncoder()
X[:,0]=Label_Encoder.fit_transform(X[:,0])
Label_Encoder.fit(['Bayview','Central','Ingleside','Mission','Northern','Out of␣
 ↪SF','Park','Richmond','Southern','Taraval','Tenderloin'])
X[:,1]=Label_Encoder.fit_transform(X[:,1])
```

Splitting Test and Train Data

[0]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=3)
print ('Training Data size for X :{}\nTesting Data size for X :{}\n'.format(␣
 ↪X_train.shape,X_test.shape))
print ('Training Data size for y :{}\nTesting Data size for y :{}\n'.format(␣
 ↪y_train.shape,y_test.shape))
```

```
Training Data size for X :(233229, 4)
Testing Data size for X :(99956, 4)

Training Data size for y :(233229,)
Testing Data size for y :(99956,)
```

**K-Nearest Neighbour ___**

[0]:
```python
#Finding K
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.metrics import jaccard_similarity_score

Ks=10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))

for n in range(1,Ks):
    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)
    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])
```

```
mean_acc
print( "The best accuracy was with", mean_acc.max(), "with k=", mean_acc.
 ↪argmax()+1)
```

```
The best accuracy was with 0.24718876305574453 with k= 9
```

It is found that the accuracy is higher = 0.25 when k is 9
Prediction

[0]:
```
neigh = KNeighborsClassifier(n_neighbors = 9).fit(X_train,y_train)
yhat_KNN = neigh.predict(X_test)
```

[0]:
```
print("Train set Accuracy : ", metrics.accuracy_score(y_train, neigh.
 ↪predict(X_train)))
print("Test set Accuracy : ", metrics.accuracy_score(y_test, yhat_KNN))
print("F1  Accuracy : ", metrics.f1_score(y_test, yhat_KNN, average='weighted'))
print("Jaccard Similarity Score : ", jaccard_similarity_score(y_test, yhat_KNN))
```

```
Train set Accuracy :  0.354685738051443
Test set Accuracy :  0.24718876305574453
F1  Accuracy :  0.17138718298316208
Jaccard Similarity Score :  0.24718876305574453

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:664:
FutureWarning: jaccard_similarity_score has been deprecated and replaced with
jaccard_score. It will be removed in version 0.23. This implementation has
surprising behavior for binary and multiclass classification tasks.
  FutureWarning)
```

**Decision Tree** ***

[0]:
```
from sklearn.tree import DecisionTreeClassifier
cat_Tree = DecisionTreeClassifier(criterion="gini", max_depth = 80)

cat_Tree.fit(X_train,y_train)
```

[0]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=80, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Prediction

[0]:
```
pred_Tree = cat_Tree.predict(X_test)
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_test, pred_Tree))
print("Jaccard Similarity Score : ", jaccard_similarity_score(y_test,␣
 ↪pred_Tree))
```

```
DecisionTrees's Accuracy:  0.24465764936572093
Jaccard Similarity Score :  0.24465764936572093
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:664:
FutureWarning: jaccard_similarity_score has been deprecated and replaced with
jaccard_score. It will be removed in version 0.23. This implementation has
surprising behavior for binary and multiclass classification tasks.
  FutureWarning)
```

[0]:
```
#X = df_Police[].values

'''
from sklearn.externals.six import StringIO
import pydotplus
import matplotlib.image as mpimg
from sklearn import tree
%matplotlib inline

dot_data = StringIO()
filename = "cat_tree.png"
featureNames = ['Incident Datetime','Police District','Latitude','Longitude']
targetNames = df_Police["Incident Category"].unique().tolist()
out=tree.export_graphviz(cat_Tree,feature_names=featureNames,␣
 ↪out_file=dot_data, class_names= np.unique(y_train), filled=True,
                         special_characters=True,rotate=False)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png(filename)
img = mpimg.imread(filename)
plt.figure(figsize=(100, 200))
plt.imshow(img,interpolation='nearest')

'''
```

[0]:
```
'\nfrom sklearn.externals.six import StringIO\nimport pydotplus\nimport
matplotlib.image as mpimg\nfrom sklearn import tree\n%matplotlib inline
\n\ndot_data = StringIO()\nfilename = "cat_tree.png"\nfeatureNames = [\'Incident
Datetime\',\'Police District\',\'Latitude\',\'Longitude\']\ntargetNames =
df_Police["Incident Category"].unique().tolist()\nout=tree.export_graphviz(cat_T
ree,feature_names=featureNames, out_file=dot_data, class_names=
np.unique(y_train), filled=True,  \n
special_characters=True,rotate=False)  \ngraph =
pydotplus.graph_from_dot_data(dot_data.getvalue())
\ngraph.write_png(filename)\nimg =
mpimg.imread(filename)\nplt.figure(figsize=(100,
200))\nplt.imshow(img,interpolation=\'nearest\')\n\n'
```

---

**Logistic Regression** ***

[0]:
```python
print(X_train.shape,y_train.shape)
print(X_test.shape,y_test.shape)
```

```
(233229, 4) (233229,)
(99956, 4) (99956,)
```

[0]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
```

[0]:
```python
LR = LogisticRegression(C=0.01, solver='sag').fit(X_train,y_train)
yhat_LR = LR.predict(X_test)
yhat_prob = LR.predict_proba(X_test)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_sag.py:330:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  "the coef_ did not converge", ConvergenceWarning)
```

[0]:
```python
print("Train set Accuracy : ", metrics.accuracy_score(y_train, LR.
 ↪predict(X_train)))
print("Test set Accuracy : ", metrics.accuracy_score(y_test, yhat_LR))
print("Regression F1  Accuracy : ", metrics.f1_score(y_test, yhat_LR,␣
 ↪average='weighted'))
print("Log Loss : ", log_loss(y_test, yhat_prob))
print("Jaccard Similarity Score : ", jaccard_similarity_score(y_test, yhat_LR))
```

```
Train set Accuracy :  0.3032513109433218
Test set Accuracy :  0.301312577534115
Regression F1  Accuracy :  0.13953491412845728
Log Loss :  2.817651886914775
Jaccard Similarity Score :  0.301312577534115
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:664:
FutureWarning: jaccard_similarity_score has been deprecated and replaced with
jaccard_score. It will be removed in version 0.23. This implementation has
surprising behavior for binary and multiclass classification tasks.
  FutureWarning)
```

**Support Vector Machine** ***

[0]:
```python
# from sklearn import svm
# clf=svm.SVC(kernel='rbf')
# clf.fit(X_train,y_train)
# yhat_svm = clf.predict(X_test)
```

[0]:
```
# print("Train set Accuracy : ", metrics.accuracy_score(y_train, clf.
 ↪predict(X_train)))
# print("Test set Accuracy : ", metrics.accuracy_score(y_test, yhat_svm))
# print("Regression F1  Accuracy : ", metrics.f1_score(y_test, yhat_svm,␣
 ↪average='weighted'))
# print("Jaccard Similarity Score : ", jaccard_similarity_score(y_test,␣
 ↪yhat_svm))
```

---

## 4.1 Plot the points on Map

---

In order to plot the incident map in the neighbourhood, we can make use of the latitude longitude data

[0]:
```
#!conda install -c conda-forge geopy --yes # uncomment this line if you haven't␣
 ↪completed the Foursquare API lab
from geopy.geocoders import Nominatim
#!conda install -c conda-forge folium=0.5.0 --yes # uncomment this line if you␣
 ↪haven't completed the Foursquare API lab
import folium
```

[0]:
```
df_Police_Short = df_Police.head(200)
```

[0]:
```
address = 'San Francisco'

geolocator = Nominatim(user_agent="ny_explorer")
location = geolocator.geocode(address)
latitude = location.latitude
longitude = location.longitude
print('The geograpical coordinate of San Francisco City are {}, {}.'.
 ↪format(latitude, longitude))
```

```
The geograpical coordinate of San Francisco City are 37.7790262, -122.4199061.
```

[36]:
```
# add markers to map
# instantiate a feature group for the incidents in the dataframe
map_SF = folium.Map(location=[latitude, longitude], zoom_start=12)

incidents = folium.map.FeatureGroup()

# loop through the 100 crimes and add each to the incidents feature group
for lat, lng, in zip(df_Police_Short.Latitude, df_Police_Short.Longitude):
    incidents.add_child(
        folium.CircleMarker(
            [lat, lng],
            radius=5, # define how big you want the circle markers to be
```

```
            color='yellow',
            fill=True,
            fill_color='blue',
            fill_opacity=0.6
        )
    )

# add pop-up text to each marker on the map
latitudes = list(df_Police_Short.Latitude)
longitudes = list(df_Police_Short.Longitude)
labels = list(df_Police_Short['Incident Category'])

for lat, lng, label in zip(latitudes, longitudes, labels):
    folium.Marker([lat, lng], popup=label).add_to(map_SF)

# add incidents to map
map_SF.add_child(incidents)
```

[36]: <folium.folium.Map at 0x7fbae9937cf8>

[37]:
```
from folium import plugins

# let's start again with a clean copy of the map of San Francisco
map_SF = folium.Map(location = [latitude, longitude], zoom_start = 12)

# instantiate a mark cluster object for the incidents in the dataframe
incidents = plugins.MarkerCluster().add_to(map_SF)

# loop through the dataframe and add each data point to the mark cluster
for lat, lng, label, in zip(df_Police_Short.Latitude, df_Police_Short.
 ↪Longitude, df_Police_Short['Incident Category']):
    folium.Marker(
        location=[lat, lng],
        icon=None,
        popup=label,
    ).add_to(incidents)

# display map
map_SF
```

[37]: <folium.folium.Map at 0x7fbae9904668>

### 4.2 --------------

## 5 Using Fourquare to visualize businesses venues

We will make calls to the Foursquare API for different purposes. You will construct a URL to send a request to the API to search for a specific type of venues, to explore a particular business venue, to explore a Foursquare user, to explore a geographical location, and to get trending venues around a location. Also, you will learn how to use the visualization library, Folium, to visualize the results.

[38]:
```
from geopy.geocoders import Nominatim # module to convert an address into␣
↪latitude and longitude values
import requests # library to handle requests
import random # library for random number generation

# libraries for displaying images
from IPython.display import Image
from IPython.core.display import HTML

# tranforming json file into a pandas dataframe library
from pandas.io.json import json_normalize

#!conda install -c conda-forge folium=0.5.0 --yes
import folium # plotting library

print('Folium installed')
print('Libraries imported.')
```

```
Folium installed
Libraries imported.
```

[0]:
```
CLIENT_ID = 'INHYLUFFXJR2LZILTPSQJYV4JQYNEQUFZKNWMQ10CWHAEWJR' # your␣
↪Foursquare ID
CLIENT_SECRET = 'RH2ADKJBT5GL21C1UV5H44HRC3QTQPWBZFYIU5KODVDOA24M' # your␣
↪Foursquare Secret
VERSION = '20180605'
LIMIT=100
```

We will use the same data frame that we used to plot the geo-location data of crimes. For an example, lets find the Crime Incident of theft, which is

[40]:
```
df_Theft= df_Police_Short.loc[df_Police_Short['Incident Category'].str.
↪contains('Vehicle Theft')]
df_Theft.head(3)
```

[40]:
```
          Incident Datetime Incident Date  ...   Latitude   Longitude
47   2019/08/21 02:00:00 PM    2019/08/21  ...  37.769007 -122.438338
99   2018/11/11 09:20:00 AM    2018/11/11  ...  37.779459 -122.402377
117  2020/05/23 06:30:00 PM    2020/05/23  ...  37.807483 -122.413975

[3 rows x 14 columns]
```

**Explore the Area of Theft and venues nearby**

Lets assume that this location is a theft prone area as per the analysis. Lets try to explore the area and find any venues within 100m radius

[0]:
```
neighborhood_latitude = df_Theft.iloc[2][12]
neighborhood_longitude = df_Theft.iloc[2][13]
#Choesn Central Neighbouthood
```

[42]:
```
radius = 100 # define radius
url = 'https://api.foursquare.com/v2/venues/explore?
↪&client_id={}&client_secret={}&v={}&ll={},{}&radius={}&limit={}'.format(
    CLIENT_ID,
    CLIENT_SECRET,
    VERSION,
    neighborhood_latitude,
    neighborhood_longitude,
    radius,
    LIMIT)
url # display URL
```

[42]:
```
'https://api.foursquare.com/v2/venues/explore?&client_id=INHYLUFFXJR2LZILTPSQJYV
4JQYNEQUFZKNWMQ10CWHAEWJR&client_secret=RH2ADKJBT5GL21C1UV5H44HRC3QTQPWBZFYIU5KO
DVDOA24M&v=20180605&ll=37.80748251193778,-122.41397500878729&radius=100&limit=10
0'
```

[0]:
```
results = requests.get(url).json()
#results
```

[0]:
```
def get_category_type(row):
    try:
        categories_list = row['categories']
    except:
        categories_list = row['venue.categories']

    if len(categories_list) == 0:
        return None
    else:
        return categories_list[0]['name']
```

[45]:
```
venues = results['response']['groups'][0]['items']
nearby_venues = json_normalize(venues) # flatten JSON

# filter columns
filtered_columns = ['venue.name', 'venue.categories', 'venue.location.lat',␣
↪'venue.location.lng']
nearby_venues =nearby_venues.loc[:, filtered_columns]

# filter the category for each row
nearby_venues['venue.categories'] = nearby_venues.apply(get_category_type,␣
↪axis=1)

# clean columns
nearby_venues.columns = [col.split(".")[-1] for col in nearby_venues.columns]
```

```
nearby_venues.head()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: FutureWarning:
pandas.io.json.json_normalize is deprecated, use pandas.json_normalize instead
```

[45]:

| | name | categories | lat | lng |
|---|---|---|---|---|
| 0 | Hot Spud | Restaurant | 37.807800 | -122.413997 |
| 1 | Big Bus Tours | Tour Provider | 37.808323 | -122.414126 |
| 2 | Hotel Zephyr San Francisco | Hotel | 37.807763 | -122.413222 |
| 3 | Tower Tours San Francisco | Tour Provider | 37.807532 | -122.413749 |
| 4 | Alamo Rent A Car | Rental Car Location | 37.807722 | -122.414738 |

[48]:

```
venues_map = folium.Map(location=[neighborhood_latitude,␣
 ↪neighborhood_longitude], zoom_start=20) # generate map centred around the␣
 ↪Hotel

# add a red circle marker to represent the Hotel
folium.CircleMarker(
    [neighborhood_latitude, neighborhood_longitude],
    radius=10,
    color='red',
    popup='Crime Incident',
    fill = True,
    fill_color = 'red',
    fill_opacity = 0.6
).add_to(venues_map)

# add the Italian restaurants as blue circle markers
for lat, lng, label in zip(nearby_venues.lat, nearby_venues.lng, nearby_venues.
 ↪categories):
    folium.CircleMarker(
        [lat, lng],
        radius=5,
        color='blue',
        popup=label,
        fill = True,
        fill_color='blue',
        fill_opacity=0.6
    ).add_to(venues_map)

# display map
venues_map
```

[48]: `<folium.folium.Map at 0x7fbae93e6e10>`