Project Name:

Taxi Trajectory Prediction

Description:

The taxi industry is evolving rapidly. New competitors and technologies are changing the way traditional taxi services do business. While this evolution has created new efficiencies, it has also created new problems.

One major shift is the widespread adoption of electronic dispatch systems that have replaced the VHF-radio dispatch systems of times past. These mobile data terminals are installed in each vehicle and typically provide information on GPS localization and taximeter state. Electronic dispatch systems make it easy to see where a taxi has been, but not necessarily where it is going. In most cases, taxi drivers operating with an electronic dispatch system do not indicate the final destination of their current ride.

Another recent change is the switch from broadcast-based (one to many) radio messages for service dispatching to unicast-based (one to one) messages. With unicast-messages, the dispatcher needs to correctly identify which taxi they should dispatch to a pick up location. Since taxis using electronic dispatch systems do not usually enter their drop off location, it is extremely difficult for dispatchers to know which taxi to contact.

Business Problem:

To improve the efficiency of electronic taxi dispatching systems it is important to predict the final destination of a taxi while it is in service. Particularly during periods of high demand, there is often a taxi whose current ride will end near or exactly at a requested pick up location from a new rider. If a dispatcher knew approximately where their taxi drivers would be ending their current rides, they would be able to identify which taxi to assign to each pickup request.

The spatial trajectory of an occupied taxi could provide some hints as to where it is going. Similarly, given the taxi id, to predict its final destination based on the regularity of pre-hired services. In a significant number of taxi rides (approximately 25%), the taxi has been called through the taxi call-center, and the passenger’s telephone id can be used to narrow the destination prediction based on historical ride data connected to their telephone id.

To overcome this problem, built a predictive framework that is able to infer the final destination of taxi rides in Porto, Portugal based on their (initial) partial trajectories. The output of such a framework must be the final trip's destination (WGS84 coordinates).

Import the necessary libraries

In [1]:
```
%%time
# Import Libraries
import numpy as np
import pandas as pd
import datetime, zipfile
import re
from itertools import cycle
import warnings
warnings.filterwarnings('ignore')
```

```python
# Data preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.multioutput import MultiOutputRegressor
from sklearn.impute import KNNImputer


#Visualization
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(style="ticks", color_codes=True)
import folium
import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots

#parameter Optimization
from sklearn.model_selection import GridSearchCV

# Validation
from sklearn.metrics import r2_score,mean_squared_error
```

```
CPU times: total: 4.55 s
Wall time: 7.87 s
```

In [2]:

```python
%%time
import os

for dirname, _, filenames in os.walk('D:\Projects'):
    for filename in filenames:
        print(os.path.join(dirname,filename))
```

```
D:\Projects\Taxi_Trajectory\Output_test.csv
D:\Projects\Taxi_Trajectory\taxi-trajectory-prediction-i.ipynb
D:\Projects\Taxi_Trajectory\test.csv.zip
D:\Projects\Taxi_Trajectory\train.csv.zip
CPU times: total: 0 ns
Wall time: 1.99 ms
```

In [3]:

```python
%%time
# Reading the data
zip_file= zipfile.ZipFile("D:\Projects\Taxi_Trajectory/train.csv.zip")
df_train = pd.read_csv(zip_file.open("train.csv"))

zip_file = zipfile.ZipFile("D:\Projects\Taxi_Trajectory/test.csv.zip")
df_test = pd.read_csv(zip_file.open("test.csv"))
```

```
CPU times: total: 28.6 s
Wall time: 29.2 s
```

Data overview

TRIP_ID: (String) It contains an unique identifier for each trip;

CALL_TYPE: (char) It identifies the way used to demand this service. It may contain one of three possible values: 'A' if this trip was dispatched from the central; 'B' if this trip was demanded directly to a taxi driver on a specific stand; 'C' otherwise (i.e. a trip demanded on a random street).

ORIGIN_CALL: (integer) It contains an unique identifier for each phone number which was used to demand, at least, one service. It identifies the trip's customer if CALL_TYPE='A'. Otherwise, it assumes a NULL value;

ORIGIN_STAND: (integer): It contains an unique identifier for the taxi stand. It identifies the starting point of the trip if CALL_TYPE='B'. Otherwise, it assumes a NULL value;

TAXI_ID: (integer): It contains an unique identifier for the taxi driver that performed each trip;

TIMESTAMP: (integer) Unix Timestamp (in seconds). It identifies the trip's start;

DAYTYPE: (char) It identifies the daytype of the trip's start. It assumes one of three possible values: 'B' if this trip started on a holiday or any other special day (i.e. extending holidays, floating holidays, etc.); 'C' if the trip started on a day before a type-B day; 'A' otherwise (i.e. a normal day, workday or weekend).

MISSING_DATA: (Boolean) It is FALSE when the GPS data stream is complete and TRUE whenever one (or more) locations are missing

POLYLINE: (String): It contains a list of GPS coordinates (i.e. WGS84 format) mapped as a string. The beginning and the end of the string are identified with brackets (i.e. [and], respectively). Each pair of coordinates is also identified by the same brackets as [LONGITUDE, LATITUDE]. This list contains one pair of coordinates for each 15 seconds of trip. The last list item corresponds to the trip's destination while the first one represents its start;

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

Out[4]:

	TRIP_ID	CALL_TYPE	ORIGIN_CALL	ORIGIN_STAND	TAXI_ID	TIMESTAMP	DAY_TYPE	MISSI
0	1372636858620000589	C	NaN	NaN	20000589	1372636858	A	
1	1372637303620000596	B	NaN	7.0	20000596	1372637303	A	
2	1372636951620000320	C	NaN	NaN	20000320	1372636951	A	
3	1372636854620000520	C	NaN	NaN	20000520	1372636854	A	
4	1372637091620000337	C	NaN	NaN	20000337	1372637091	A	

In [5]:
```
print("Train shape:",df_train.shape)
print("Test shape:",df_test.shape)
```

```
Train shape: (1710670, 9)
Test shape: (320, 9)
```

Exploratory Data Analysis (EDA)

In [6]:
```
# Time data preprocessing
df_train['TIMESTAMP'] = [float(time) for time in df_train['TIMESTAMP']]
df_train['Date_Time'] = [datetime.datetime.fromtimestamp(time, datetime.timezone.utc) for tim
```

In [7]:
```
%%time
df_train["Year"] = df_train["Date_Time"].dt.isocalendar().year
df_train["Month"] = df_train["Date_Time"].dt.month
df_train["Day"] = df_train["Date_Time"].dt.day
df_train["Hour"] = df_train["Date_Time"].dt.hour
```

```python
df_train["Minute"] = df_train["Date_Time"].dt.minute
df_train["Weekday"] = df_train["Date_Time"].dt.weekday
```

```
CPU times: total: 1.25 s
Wall time: 1.25 s
```

In [8]:
```python
df_train.head()
```

Out[8]:

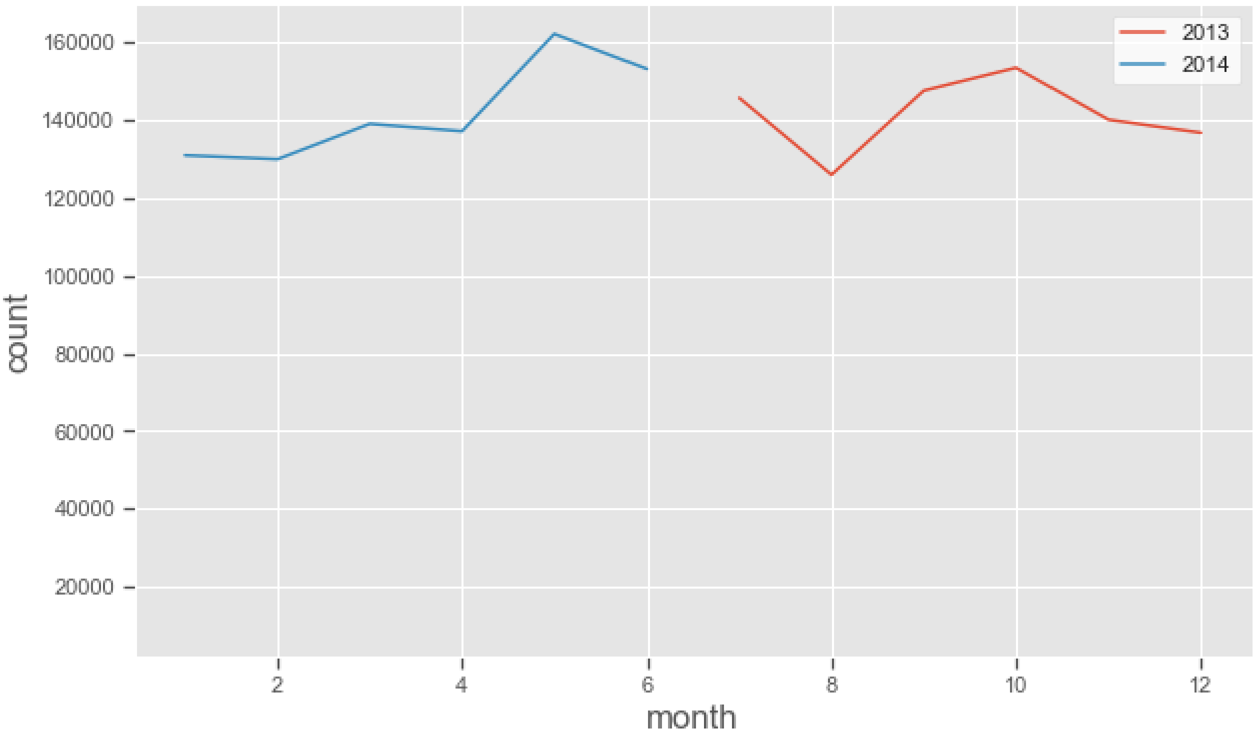
	TRIP_ID	CALL_TYPE	ORIGIN_CALL	ORIGIN_STAND	TAXI_ID	TIMESTAMP	DAY_TYPE	MISS
0	1372636858620000589	C	NaN	NaN	20000589	1.372637e+09	A	
1	1372637303620000596	B	NaN	7.0	20000596	1.372637e+09	A	
2	1372636951620000320	C	NaN	NaN	20000320	1.372637e+09	A	
3	1372636854620000520	C	NaN	NaN	20000520	1.372637e+09	A	
4	1372637091620000337	C	NaN	NaN	20000337	1.372637e+09	A	

In [9]:
```python
# Visualization
pivot =  pd.pivot_table(df_train, index='Month', columns='Year', values= 'TRIP_ID', aggfunc='
pivot
```

Out[9]:

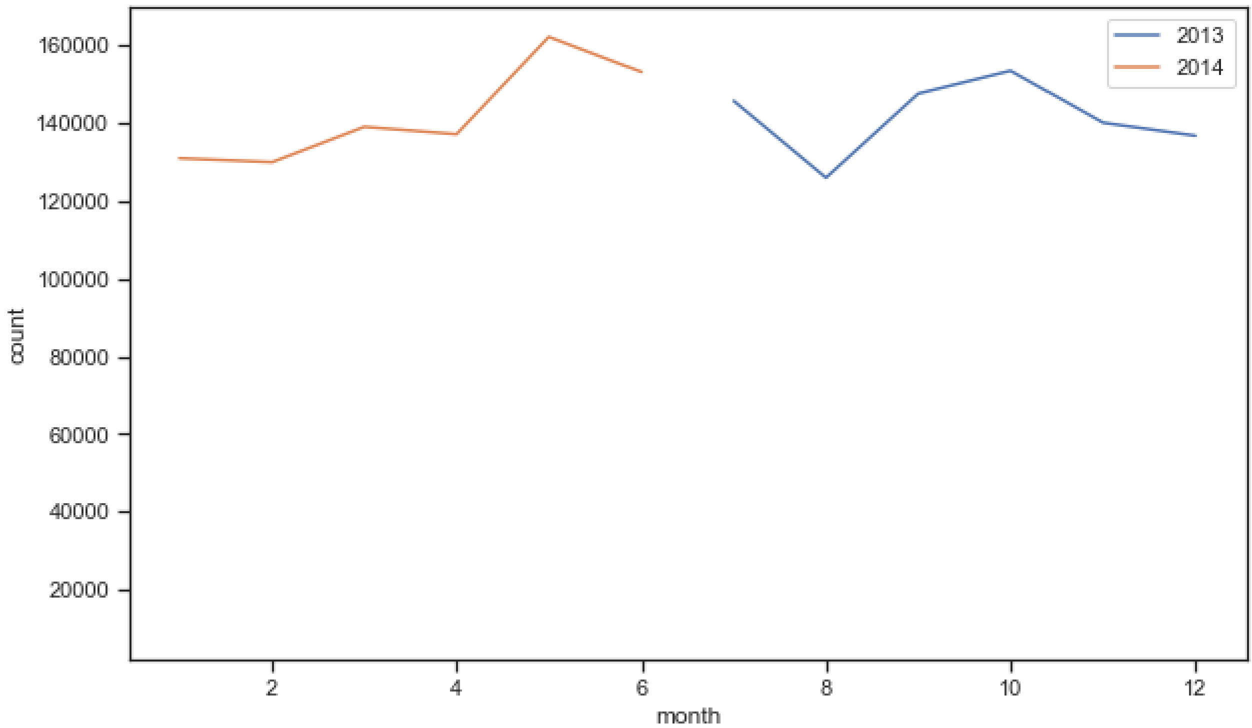
Year	Month	2013	2014
0	1	NaN	130875.0
1	2	NaN	129872.0
2	3	NaN	138969.0
3	4	NaN	137061.0
4	5	NaN	162107.0
5	6	NaN	153095.0
6	7	145640.0	NaN
7	8	125861.0	NaN
8	9	147514.0	NaN
9	10	153394.0	NaN
10	11	140024.0	NaN
11	12	136672.0	9586.0

In [10]:
```python
# Visualization monthwise
with plt.style.context('ggplot'):
    plt.figure(figsize=(10,6))
    plt.rcParams["font.size"]=14
    plt.plot(pivot['Month'], pivot[2013], label = "2013")
    plt.plot(pivot['Month'], pivot[2014], label = "2014")
    plt.xlabel("month")
    plt.ylabel("count")
    plt.legend(facecolor="white", loc='best')
```



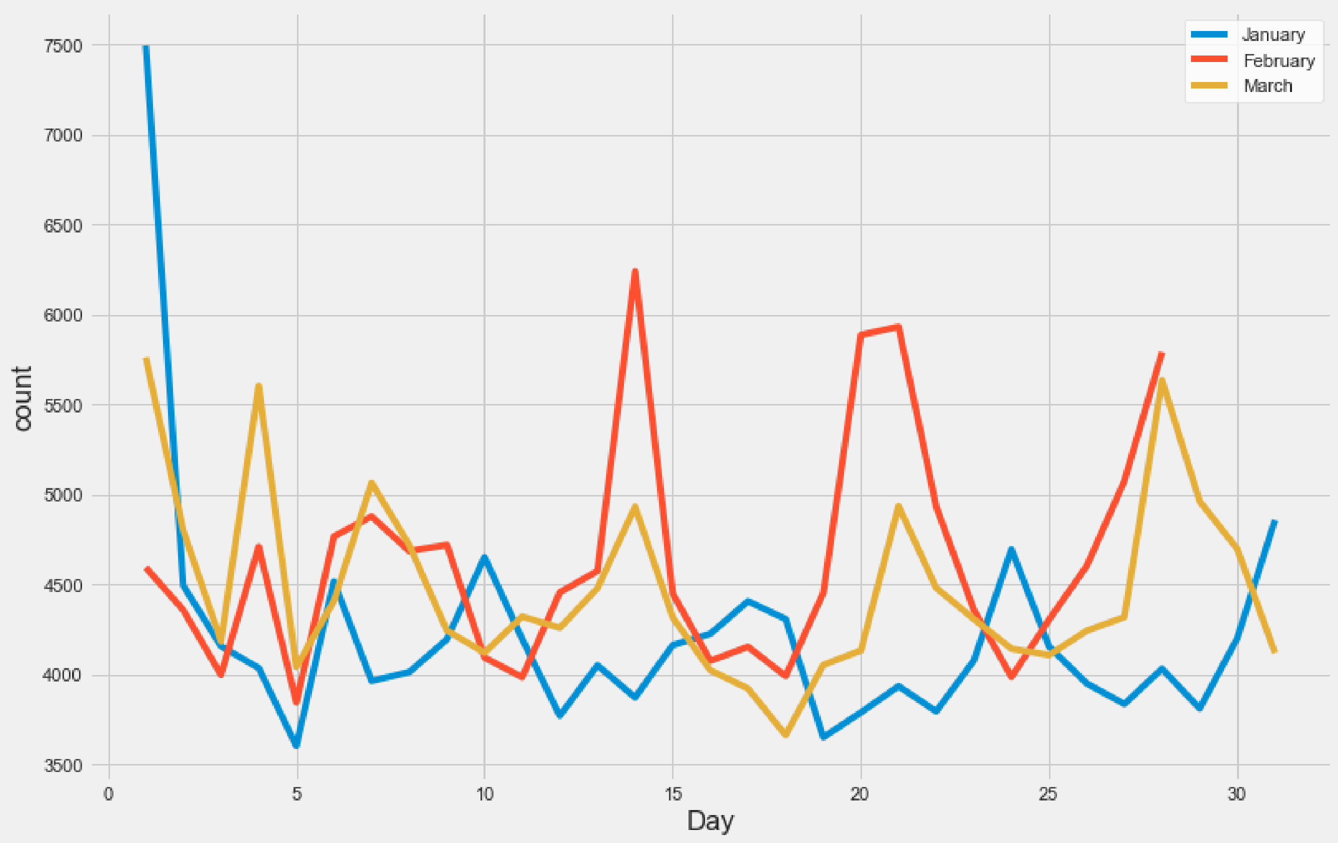

```
In [11]: #comparing with normal visualization
plt.figure(figsize=(10,6))
plt.rcParams["font.size"]=14
plt.plot(pivot['Month'], pivot[2013], label = "2013")
plt.plot(pivot['Month'], pivot[2014], label = "2014")
plt.xlabel("month")
plt.ylabel("count")
plt.legend(facecolor="white", loc='best')
```

```
Out[11]: <matplotlib.legend.Legend at 0x284dcc14eb0>
```

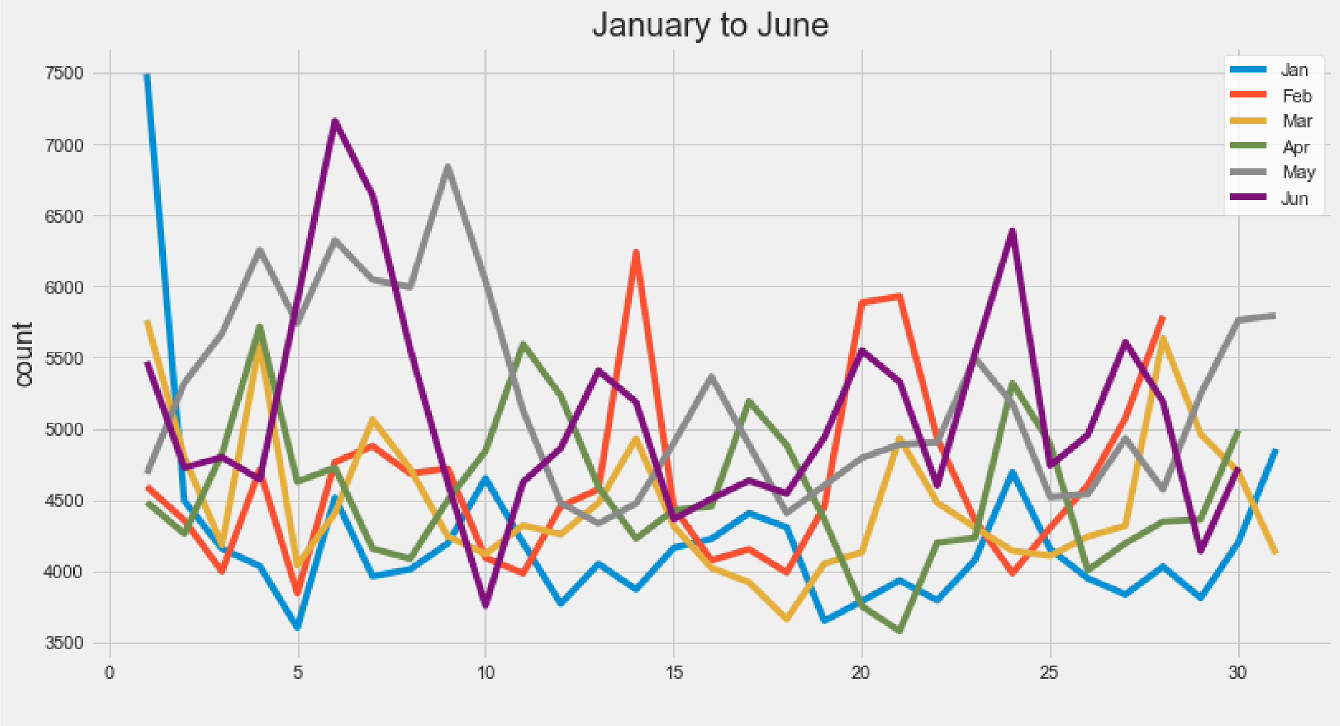


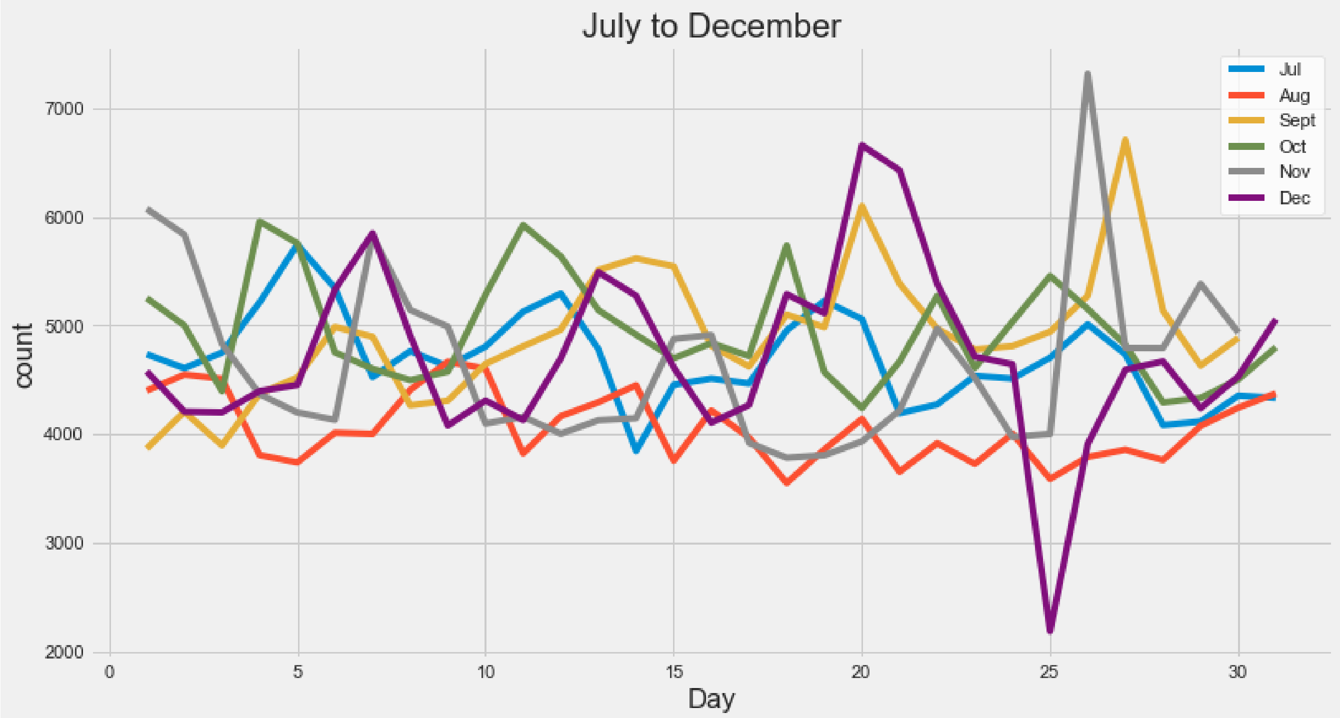

```
In [12]: # Visualization daywise
pivot= pd.pivot_table(df_train, index='Day', columns='Month', values= 'TRIP_ID', aggfunc='cou

with plt.style.context('fivethirtyeight'):
    plt.figure(figsize=(12,8))
    plt.rcParams["font.size"]=14
    plt.plot(pivot['Day'], pivot[1], label = "January")
    plt.plot(pivot['Day'], pivot[2], label = "February")
    plt.plot(pivot['Day'], pivot[3], label = "March")
```

```
    plt.xlabel("Day")
    plt.ylabel("count")
    plt.legend(facecolor="white", loc='best')
```



In [13]:
```
with plt.style.context('fivethirtyeight'):
    plt.figure(figsize=(12,14))
    plt.rcParams["font.size"]=14
    month = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sept', 'Oct', 'Nov', 'D
    plt.subplot(2,1,1)
    for i in range(1,7):
        plt.plot(pivot['Day'], pivot[i], label =month[i-1])
    plt.ylabel("count")
    plt.title("January to June")
    plt.legend(facecolor="white", loc='best')
    plt.subplot(2,1,2)
    for i in range(7,13):
        plt.plot(pivot['Day'], pivot[i], label =month[i-1])
    plt.xlabel("Day")
    plt.ylabel("count")
    plt.title("July to December")
    plt.legend(facecolor="white", loc='best')
```





In [14]:

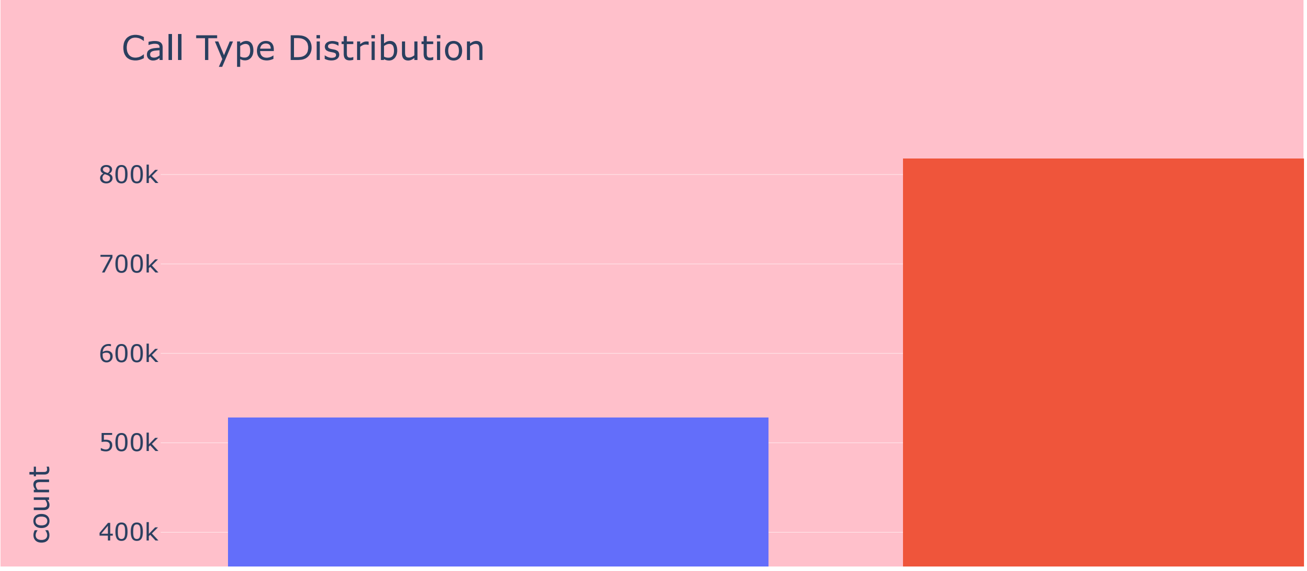
```
df_train['CALL_TYPE'].value_counts()
```

Out[14]:

```
B    817881
C    528019
A    364770
Name: CALL_TYPE, dtype: int64
```

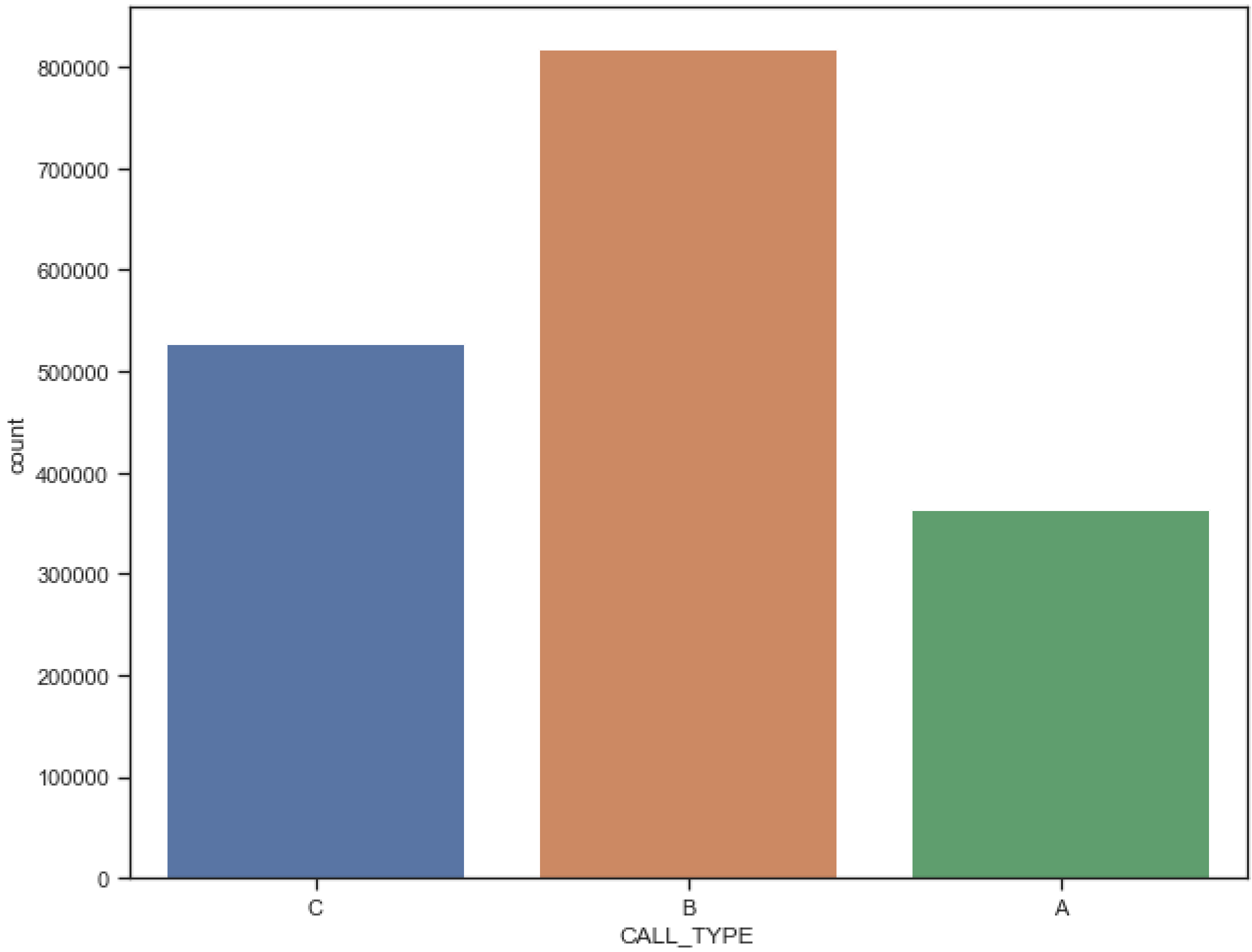
In [15]:

```
# Call type Visualization
fig = px.histogram(df_train, x='CALL_TYPE',color=df_train['CALL_TYPE'])
fig.update_layout(title_text='Call Type Distribution', bargap=0.2, paper_bgcolor="pink", plot
fig.show()
```

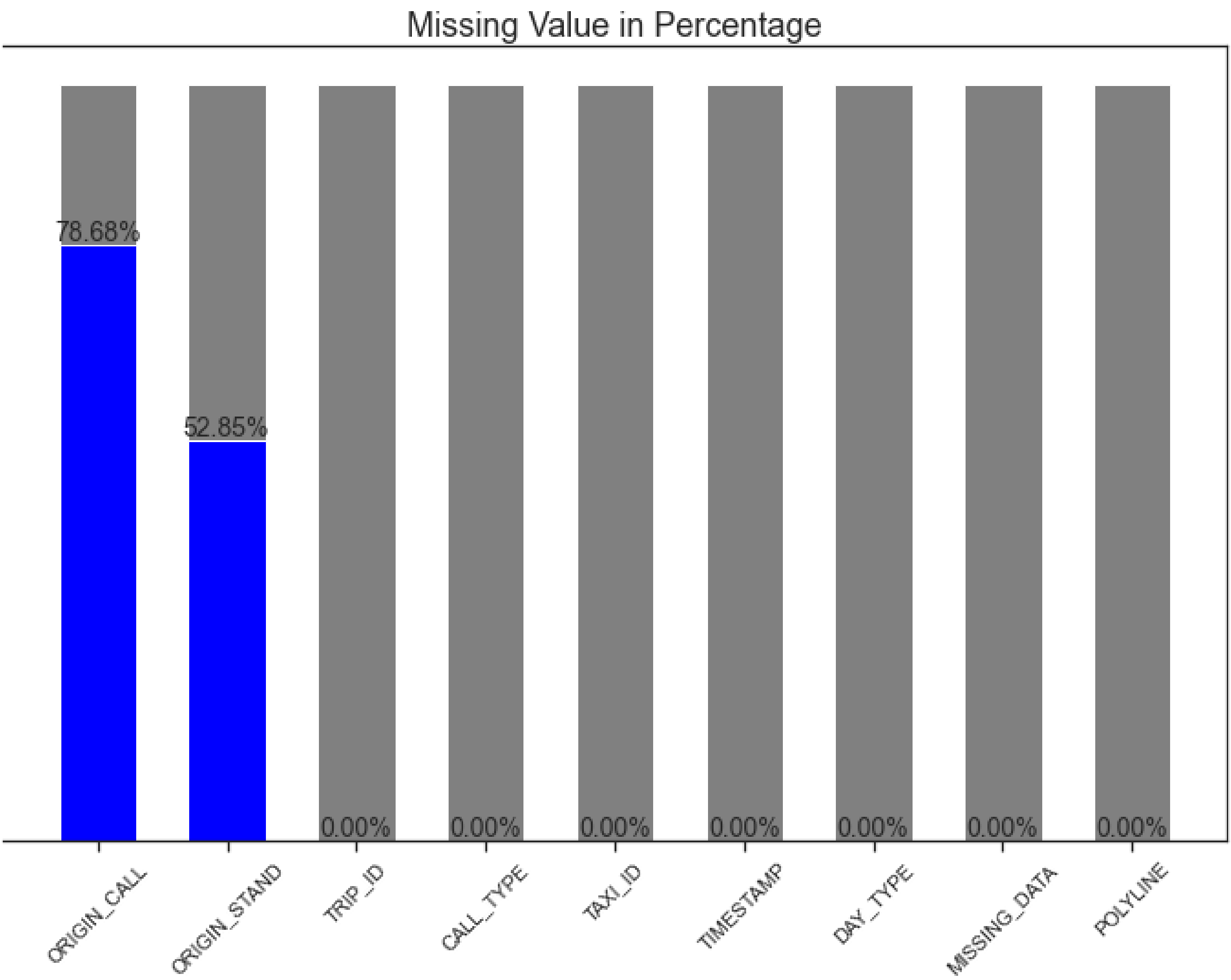


In [16]:
```
plt.figure(figsize=(10,8))
sns.countplot(data=df_train, x=df_train['CALL_TYPE'])
```

Out[16]: <AxesSubplot:xlabel='CALL_TYPE', ylabel='count'>



In [17]:
```python
# Missing Value
df_train_nan = (df_train.isna().sum().sort_values(ascending=False)/len(df_train)*100)[:9]

fig, ax = plt.subplots(1,1,figsize=(12,8))
ax.bar(df_train_nan.index, 100, color="grey", width=0.6)
bar= ax.bar(df_train_nan.index, df_train_nan, color="blue", width=0.6)
ax.bar_label(bar, fmt='%.02f%%')
ax.spines.left.set_visible(False)
ax.set_yticks([])
ax.set_title('Missing Value in Percentage',fontsize=18)
plt.xticks(rotation=45)
plt.show()
```



Data Preprocessing

In [18]:
```python
%%time
# First Longitude and Latitude
Lon_1st = []
for i in range(0, len(df_train['POLYLINE'])):
    if df_train['POLYLINE'][i]=='[]':
        k=0
        Lon_1st.append(k)
    else:
        k=re.sub(r"[[|[|]|]|]]", "", df_train['POLYLINE'][i]).split(",")[0]
        Lon_1st.append(k)
df_train['Lon_1st'] = Lon_1st

Lat_1st=[]
for i in range(0, len(df_train['POLYLINE'])):
    if df_train['POLYLINE'][i]=='[]':
        k=0
        Lat_1st.append(k)
    else:
        k=re.sub(r"[[|[|]|]|]]","",df_train['POLYLINE'][i]).split(",")[1]
```

```
        Lat_1st.append(k)
df_train['Lat_1st']= Lat_1st
```

```
CPU times: total: 3min 23s
Wall time: 3min 23s
```

In [19]:
```
%%time
# Last Longitude and Latitude
Lon_last = []
for i in range(0, len(df_train['POLYLINE'])):
    if df_train['POLYLINE'][i]=='[]':
        k=0
        Lon_last.append(k)
    else:
        k=re.sub(r"[[|[|]|]|]]", "", df_train['POLYLINE'][i]).split(",")[-2]
        Lon_last.append(k)
df_train['Lon_last'] = Lon_last

Lat_last=[]
for i in range(0, len(df_train['POLYLINE'])):
    if df_train['POLYLINE'][i]=='[]':
        k=0
        Lat_last.append(k)
    else:
        k=re.sub(r"[[|[|]|]|]]","",df_train['POLYLINE'][i]).split(",")[-1]
        Lat_last.append(k)
df_train['Lat_last']= Lat_last
```

```
CPU times: total: 3min 30s
Wall time: 3min 30s
```

In [20]:
```
# Removing zeros related to longitude & latitude
df_train = df_train.query("Lon_last !=0")
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1704769 entries, 0 to 1710669
Data columns (total 20 columns):
 #   Column        Dtype
---  ------        -----
 0   TRIP_ID       int64
 1   CALL_TYPE     object
 2   ORIGIN_CALL   float64
 3   ORIGIN_STAND  float64
 4   TAXI_ID       int64
 5   TIMESTAMP     float64
 6   DAY_TYPE      object
 7   MISSING_DATA  bool
 8   POLYLINE      object
 9   Date_Time     datetime64[ns, UTC]
 10  Year          UInt32
 11  Month         int64
 12  Day           int64
 13  Hour          int64
 14  Minute        int64
 15  Weekday       int64
 16  Lon_1st       object
 17  Lat_1st       object
 18  Lon_last      object
 19  Lat_last      object
dtypes: UInt32(1), bool(1), datetime64[ns, UTC](1), float64(3), int64(7), object(7)
memory usage: 256.9+ MB
```

In [21]:
```
df_train['Lon_1st'] = [float(i)for i in df_train['Lon_1st']]
df_train['Lat_1st'] = [float(i)for i in df_train['Lat_1st']]
df_train['Lon_last'] = [float(i)for i in df_train['Lon_last']]
df_train['Lat_last'] = [float(i)for i in df_train['Lat_last']]
```

In [22]:
```
#Visualization for first 5000 data
```

```python
mapping_1st= pd.DataFrame({"Date":df_train.head(5000)["Date_Time"].values,
                           "Lat": df_train.head(5000)["Lat_1st"].values,
                           "Lon":df_train.head(5000)["Lon_1st"].values})
mapping_last = pd.DataFrame({"Date":df_train.head(5000)["Date_Time"].values,
                             "Lat":df_train.head(5000)["Lat_last"].values,
                             "Lon":df_train.head(5000)["Lon_last"].values})
```

In [23]:

```python
print(mapping_1st)
print("***************")
mapping_last
```

```
                     Date        Lat       Lon
0    2013-07-01 00:00:58  41.141412 -8.618643
1    2013-07-01 00:08:23  41.159826 -8.639847
2    2013-07-01 00:02:31  41.140359 -8.612964
3    2013-07-01 00:00:54  41.151951 -8.574678
4    2013-07-01 00:04:51  41.180490 -8.645994
...                  ...        ...       ...
4995 2013-07-02 05:51:16  41.175396 -8.627769
4996 2013-07-01 13:56:52  41.154282 -8.649405
4997 2013-07-02 05:16:31  41.154408 -8.613306
4998 2013-07-01 18:15:57  41.155533 -8.590626
4999 2013-07-02 06:25:36  41.160393 -8.653653

[5000 rows x 3 columns]
***************
```

Out[23]:

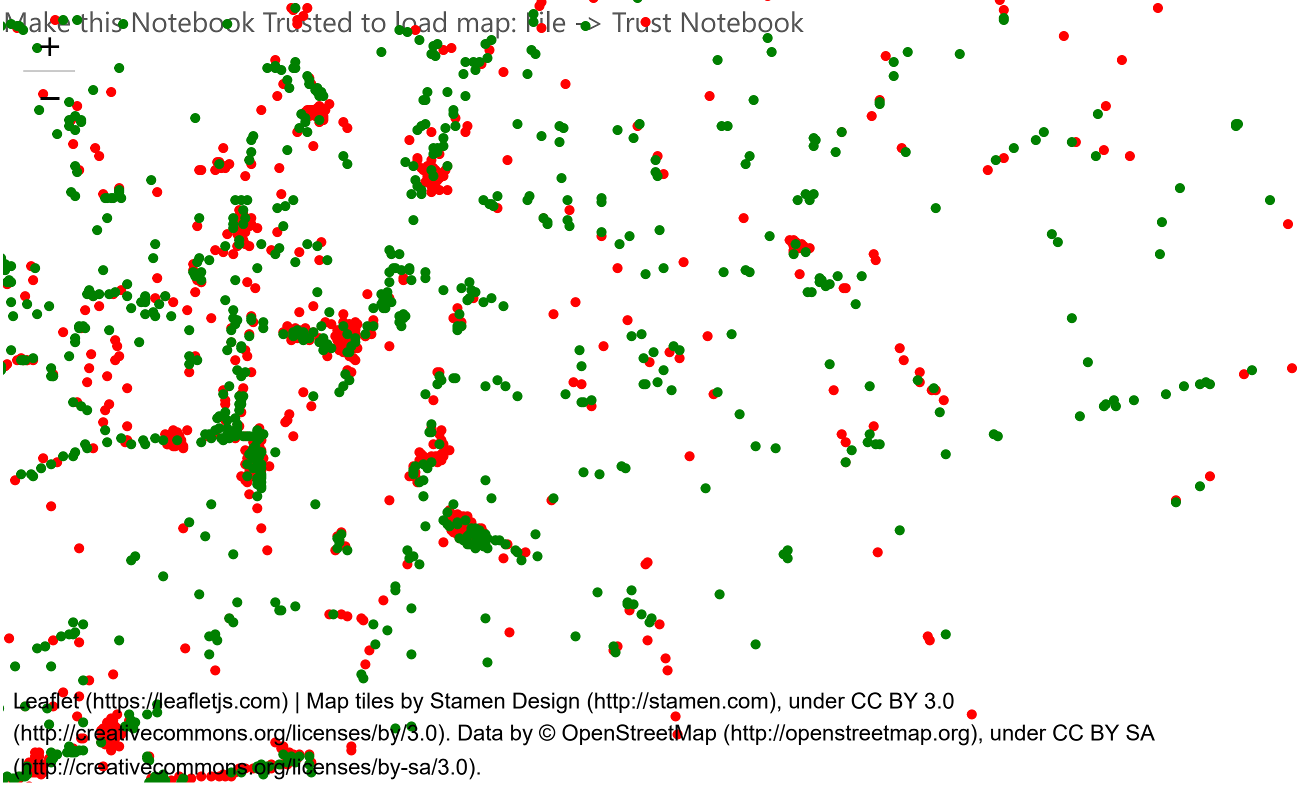
	Date	Lat	Lon
0	2013-07-01 00:00:58	41.154489	-8.630838
1	2013-07-01 00:08:23	41.170671	-8.665740
2	2013-07-01 00:02:31	41.140530	-8.615970
3	2013-07-01 00:00:54	41.142915	-8.607996
4	2013-07-01 00:04:51	41.178087	-8.687268
...	...	...	...
4995	2013-07-02 05:51:16	41.181498	-8.663490
4996	2013-07-01 13:56:52	41.168871	-8.635950
4997	2013-07-02 05:16:31	41.158593	-8.641044
4998	2013-07-01 18:15:57	41.166153	-8.659674
4999	2013-07-02 06:25:36	41.129658	-8.620137

5000 rows × 3 columns

In [24]:

```python
fol_map = folium.Map(location=[41.141412,-8.590324], tiles='Stamen Terrain', zoom_start=15)
for i,r in mapping_1st.iterrows():
    folium.CircleMarker(location=[r["Lat"],r["Lon"]], radius=0.5, color="red").add_to(fol_map

for i,r in mapping_last.iterrows():
    folium.CircleMarker(location=[r["Lat"],r["Lon"]], radius=0.5, color="green").add_to(fol_m

fol_map
```

Out[24]:



In [25]:

```
# Test Dataset Preprocessing
# First Longitude and Latitude
Lon_1st = []
for i in range(0, len(df_test['POLYLINE'])):
    if df_test['POLYLINE'][i]=='[]':
        k=0
        Lon_1st.append(k)
    else:
        k=re.sub(r"[[|[|]|]|]]", "", df_test['POLYLINE'][i]).split(",")[0]
        Lon_1st.append(k)
df_test['Lon_1st'] = Lon_1st

Lat_1st=[]
for i in range(0, len(df_test['POLYLINE'])):
    if df_test['POLYLINE'][i]=='[]':
        k=0
        Lat_1st.append(k)
    else:
        k=re.sub(r"[[|[|]|]|]]","",df_test['POLYLINE'][i]).split(",")[1]
        Lat_1st.append(k)
df_test['Lat_1st']= Lat_1st
```

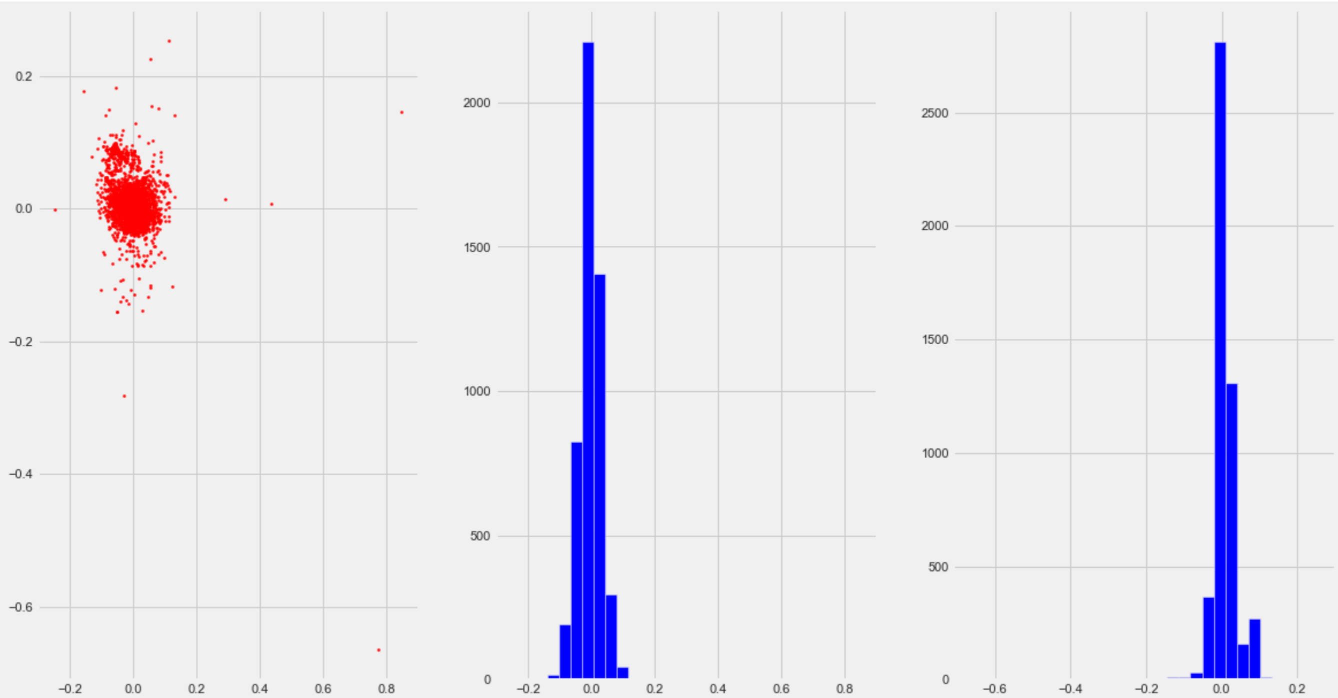
In [26]:

```
# Last Longitude and Latitude
Lon_last = []
for i in range(0, len(df_test['POLYLINE'])):
    if df_test['POLYLINE'][i]=='[]':
        k=0
        Lon_last.append(k)
    else:
        k=re.sub(r"[[[]|]|]", "", df_test['POLYLINE'][i]).split(",")[-2]
        Lon_last.append(k)

df_test['Lon_last'] = Lon_last

Lat_last=[]
for i in range(0, len(df_test['POLYLINE'])):
    if df_test['POLYLINE'][i]=='[]':
        k=0
        Lat_last.append(k)
    else:
        k=re.sub(r"[[[]|]|]", "", df_test['POLYLINE'][i]).split(",")[-1]
        Lat_last.append(k)
df_test['Lat_last']= Lat_last
```

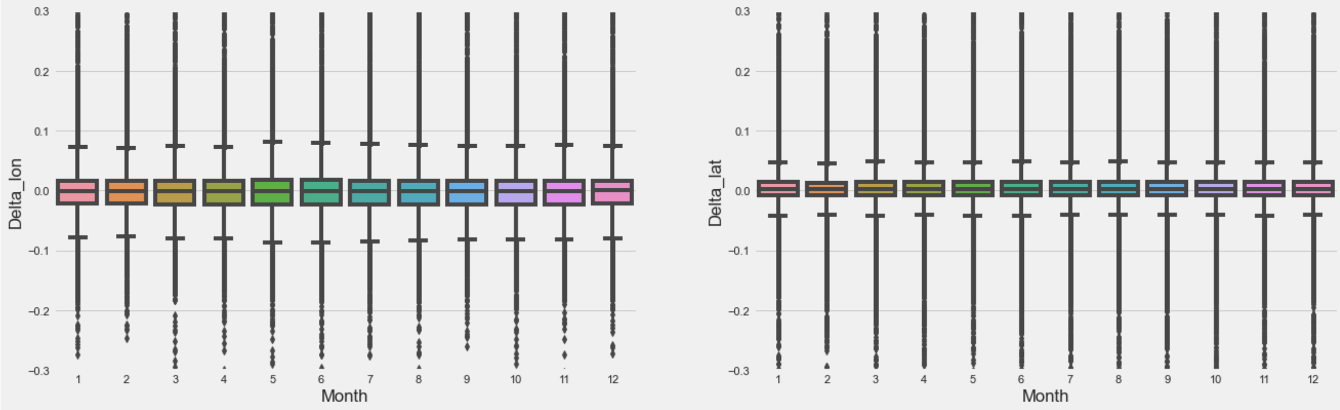
In [27]:
```
df_test['Lon_1st'] = [float(i)for i in df_test['Lon_1st']]
df_test['Lat_1st'] = [float(i)for i in df_test['Lat_1st']]
df_test['Lon_last'] = [float(i)for i in df_test['Lon_last']]
df_test['Lat_last'] = [float(i)for i in df_test['Lat_last']]
```

In [28]:
```
# Create Delta Parameter
df_train["Delta_lon"]= df_train["Lon_last"]-df_train["Lon_1st"]
df_train["Delta_lat"]= df_train["Lat_last"]-df_train["Lat_1st"]

df_test["Delta_lon"]= df_test["Lon_last"]-df_test["Lon_1st"]
df_test["Delta_lat"]= df_test["Lat_last"]-df_test["Lat_1st"]
```

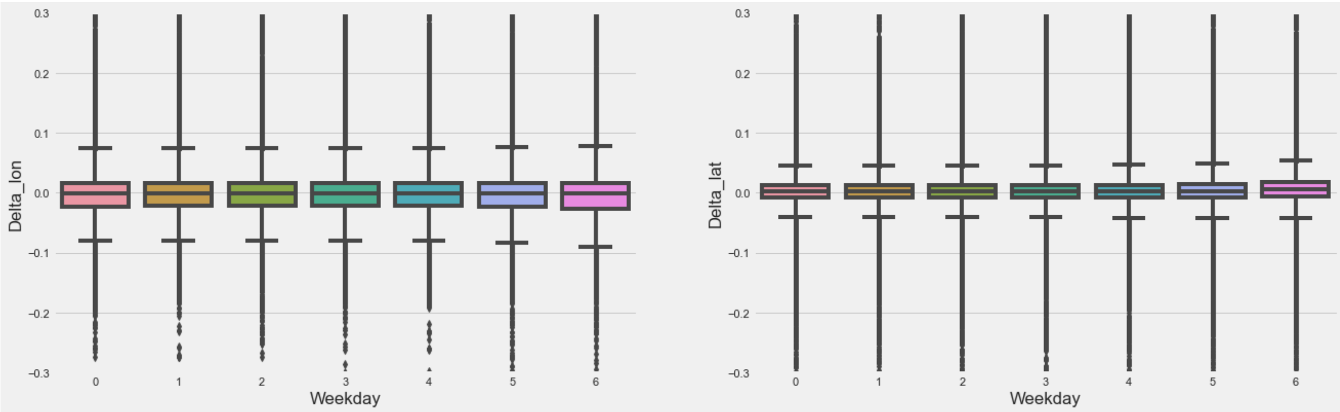
In [29]:
```
sample = df_train.head(5000)

with plt.style.context("fivethirtyeight"):
    fig, ax = plt.subplots(1,3,figsize=(18,10))

    ax[0].scatter(sample["Delta_lon"], sample["Delta_lat"], color="red",s=4 )

    ax[1].hist(sample["Delta_lon"], bins=30, color="blue")


    ax[2].hist(sample["Delta_lat"], bins=30, color="blue")
```



In [30]:
```
with plt.style.context("fivethirtyeight"):
    fig, ax = plt.subplots(1,2, figsize=(20,6))
    # delta_lon
    sns.boxplot("Month", "Delta_lon", data=df_train, ax=ax[0])
    ax[0].set_ylim([-0.3,0.3])
    sns.boxplot("Month", "Delta_lat", data=df_train, ax=ax[1])
    ax[1].set_ylim([-0.3,0.3])
```



In [31]:
```
with plt.style.context("fivethirtyeight"):
    fig, ax = plt.subplots(1,2, figsize=(20,6))
    # delta_lon
    sns.boxplot("Weekday", "Delta_lon", data=df_train, ax=ax[0])
    ax[0].set_ylim([-0.3,0.3])
    sns.boxplot("Weekday", "Delta_lat", data=df_train, ax=ax[1])
    ax[1].set_ylim([-0.3,0.3])
```



In [32]:
```
# Outliers is dropped
df_train_ml = df_train.copy()
df_train_ml = df_train_ml.query("Delta_lon <=0.2 & Delta_lon >= -0.2 & Delta_lat <=0.2 & Delt
```

In [33]:
```
print(df_train_ml['DAY_TYPE'].value_counts())
# Single type, hence drop
df_train_ml.drop(['DAY_TYPE'], axis=1, inplace=True)
```

```
A    1699677
Name: DAY_TYPE, dtype: int64
```

In [34]:
```
map_calltype = {"A":1,'B':2,'C':3}
df_train_ml['CALL_TYPE'] = df_train_ml['CALL_TYPE'].map(map_calltype)
```

In [35]:
```
df_train_ml['MISSING_DATA'].value_counts()
```

Out[35]:
```
False    1699668
True           9
Name: MISSING_DATA, dtype: int64
```

df_train_ml['ORIGIN_STAND']=

df_train_ml['ORIGIN_STAND'].fillna(value=df_train_ml["ORIGIN_STAND"].median())

df_train_ml['ORIGIN_CALL']= df_train_ml['ORIGIN_CALL'].fillna(value=df_train_ml["ORIGIN_CALL"].mode()[0])

In [36]:
```
df_train_ml['ORIGIN_STAND']=[str(i) for i in df_train_ml['ORIGIN_STAND']]
df_train_ml['ORIGIN_CALL']=[str(i) for i in df_train_ml['ORIGIN_CALL']]
```

In [37]:
```
def origin_stand(x):
    if x['ORIGIN_STAND']== 'nan':
        res=0
```

```python
    else:
        res=1
    return res
df_train_ml['ORIGIN_STAND'] = df_train_ml.apply(origin_stand, axis=1)

def origin_call(x):
    if x['ORIGIN_CALL']== 'nan':
        res=0
    else:
        res=1
    return res
df_train_ml['ORIGIN_CALL'] = df_train_ml.apply(origin_call, axis=1)
```

In [38]:
```python
df_train_ml.isna().any().any()
```

Out[38]:
```
False
```

In [39]:
```python
def missing_flag(x):
    if x["MISSING_DATA"] == False:
        res=0
    else:
        res=1
    return res

df_train_ml["MISSING_DATA"] = df_train_ml.apply(missing_flag,axis=1)
```

In [40]:
```python
df_train_ml = df_train_ml.sample(6000)
```

In [41]:
```python
X = df_train_ml[["CALL_TYPE", 'ORIGIN_CALL', 'ORIGIN_STAND', 'MISSING_DATA', 'Lon_1st', 'Lat_
y = df_train_ml[["Lon_last","Lat_last"]]
```

In [42]:
```python
df_test_ml = df_test.copy()
```

In [43]:
```python
df_test_ml.drop(columns='DAY_TYPE', inplace=True, axis=1)
```

In [44]:
```python
map_calltype = {"A":1,'B':2,'C':3}
df_test_ml['CALL_TYPE'] = df_test_ml['CALL_TYPE'].map(map_calltype)
```

In [45]:
```python
df_test_ml['ORIGIN_STAND']=[str(i) for i in df_test_ml['ORIGIN_STAND']]
df_test_ml['ORIGIN_CALL']=[str(i) for i in df_test_ml['ORIGIN_CALL']]
```

In [46]:
```python
df_test_ml['ORIGIN_STAND'] = df_test_ml.apply(origin_stand, axis=1)
df_test_ml['ORIGIN_CALL'] = df_test_ml.apply(origin_call, axis=1)
```

In [47]:
```python
df_test_ml["MISSING_DATA"] = df_test_ml.apply(missing_flag,axis=1)
```

In [48]:
```python
df_test_ml.head()
```

Out[48]:

	TRIP_ID	CALL_TYPE	ORIGIN_CALL	ORIGIN_STAND	TAXI_ID	TIMESTAMP	MISSING_DATA	PO
0	T1	2	0	1	20000542	1408039037	0	[[-8.585676,41.1 [-8.585712,41.1
1	T2	2	0	1	20000108	1408038611	0	[[-8.610876,41. [-8.610858,41.1
2	T3	2	0	1	20000370	1408038568	0	[[-8.585739,41.1 [-8.58573,41.1
3	T4	2	0	1	20000492	1408039090	0	[[-8.613963,41.1 [-8.614125,41.1
4	T5	2	0	1	20000621	1408039177	0	[[-8.619903,41.1 [-8.619894,41.1

In [49]:

```
X_test = df_test_ml[['CALL_TYPE', 'ORIGIN_CALL', 'ORIGIN_STAND', 'MISSING_DATA', 'Lon_1st', '
```

In [50]:

```
# Train test split
X_train, x_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state=1)
```

In [51]:

```
X.shape, X_train.shape, x_test.shape
```

Out[51]:

```
((6000, 8), (4200, 8), (1800, 8))
```

Model Building

In [52]:

```
%%time
# Random Forest
RF = MultiOutputRegressor(RandomForestRegressor(n_estimators=100, random_state=1))
RF= RF.fit(X_train,y_train)
y_train_pred = RF.predict(X_train)
y_test_pred = RF.predict(x_test)
df_test_pred = RF.predict(X_test)

print("Mean Squarred Error - Train: {}".format(mean_squared_error(y_train,y_train_pred)))
print("Mean Squarred Error - Test: {}".format(mean_squared_error(y_test,y_test_pred)))
print("R2 Score- Train-{}". format(r2_score(y_train,y_train_pred)))
print("R2 Score- Test-{}". format(r2_score(y_test,y_test_pred)))
```

```
Mean Squarred Error - Train: 1.3854704909786454e-06
Mean Squarred Error - Test: 0.0002800426091761335
R2 Score- Train-0.9979665793193333
R2 Score- Test-0.8271781673447849
CPU times: total: 3.3 s
Wall time: 3.49 s
```

In [53]:

```
%%time
# Gradient Boosting
GB = MultiOutputRegressor(GradientBoostingRegressor(random_state=1))
GB=GB.fit(X_train,y_train)
y_train_pred_gb= GB.predict(X_train)
y_test_pred_gb = GB.predict(x_test)
df_test_pred_gb = GB.predict(X_test)

print("Mean Squarred Error - Train: {}".format(mean_squared_error(y_train,y_train_pred_gb)))
print("Mean Squarred Error - Test: {}".format(mean_squared_error(y_test,y_test_pred_gb)))
print("R2 Score- Train-{}". format(r2_score(y_train,y_train_pred_gb)))
print("R2 Score- Test-{}". format(r2_score(y_test,y_test_pred_gb)))
```

```
Mean Squared Error - Train: 2.340490315535844e-06
Mean Squared Error - Test: 0.0002720444288615266
R2 Score- Train-0.997113487397032
R2 Score- Test-0.8348337300534678
CPU times: total: 1.09 s
Wall time: 1.12 s
```

In [54]:
```
%%time
#LINEAR REGRESSION
LR= MultiOutputRegressor(LinearRegression())

lr = LR.fit(X_train,y_train)
y_train_pred_lr= lr.predict(X_train)
y_test_pred_lr = lr.predict(x_test)
df_test_pred_lr = lr.predict(X_test)

print("Mean Squared Error - Train: {}".format(mean_squared_error(y_train,y_train_pred_lr)))
print("Mean Squared Error - Test: {}".format(mean_squared_error(y_test,y_test_pred_lr)))
print("R2 Score- Train-{}". format(r2_score(y_train,y_train_pred_lr)))
print("R2 Score- Test-{}". format(r2_score(y_test,y_test_pred_lr)))
```

```
Mean Squared Error - Train: 1.0099898863959645e-28
Mean Squared Error - Test: 1.01009256352966e-28
R2 Score- Train-1.0
R2 Score- Test-1.0
CPU times: total: 15.6 ms
Wall time: 57.8 ms
```

In [55]:
```
%%time
#knn
KNN = MultiOutputRegressor(KNeighborsRegressor())
knn=KNN.fit(X_train,y_train)
y_train_pred_knn= knn.predict(X_train)
y_test_pred_knn = knn.predict(x_test)
df_test_pred_knn = knn.predict(X_test)

print("Mean Squared Error - Train: {}".format(mean_squared_error(y_train,y_train_pred_knn)))
print("Mean Squared Error - Test: {}".format(mean_squared_error(y_test,y_test_pred_knn)))
print("R2 Score- Train-{}". format(r2_score(y_train,y_train_pred_knn)))
print("R2 Score- Test-{}". format(r2_score(y_test,y_test_pred_knn)))
```

```
Mean Squared Error - Train: 2.6634657096257176e-05
Mean Squared Error - Test: 0.00034993693776869936
R2 Score- Train-0.9629419526602927
R2 Score- Test-0.7617568290922252
CPU times: total: 281 ms
Wall time: 291 ms
```

In [56]:
```
%%time
#Decision Tree

DT= MultiOutputRegressor(LinearRegression())

dt = DT.fit(X_train,y_train)
y_train_pred_dt= dt.predict(X_train)
y_test_pred_dt = dt.predict(x_test)
df_test_pred_dt = dt.predict(X_test)

print("Mean Squared Error - Train: {}".format(mean_squared_error(y_train,y_train_pred_dt)))
print("Mean Squared Error - Test: {}".format(mean_squared_error(y_test,y_test_pred_dt)))
print("R2 Score- Train-{}". format(r2_score(y_train,y_train_pred_dt)))
print("R2 Score- Test-{}". format(r2_score(y_test,y_test_pred_dt)))
```

```
Mean Squared Error - Train: 1.0099898863959645e-28
Mean Squared Error - Test: 1.01009256352966e-28
R2 Score- Train-1.0
R2 Score- Test-1.0
CPU times: total: 31.2 ms
Wall time: 22.4 ms
```

In [57]:
```
%%time
from xgboost import XGBRegressor
# Gradient Boosting
XGB = MultiOutputRegressor(XGBRegressor(random_state=42, n_jobs=-1))
XGB=XGB.fit(X_train,y_train)
y_train_pred_xgb= XGB.predict(X_train)
y_test_pred_xgb = XGB.predict(x_test)
df_test_pred_xgb = XGB.predict(X_test)

print("Mean Squarred Error - Train: {}".format(mean_squared_error(y_train,y_train_pred_xgb)))
print("Mean Squarred Error - Test: {}".format(mean_squared_error(y_test,y_test_pred_xgb)))
print("R2 Score- Train-{}". format(r2_score(y_train,y_train_pred_xgb)))
print("R2 Score- Test-{}". format(r2_score(y_test,y_test_pred_xgb)))
```

```
Mean Squarred Error - Train: 4.0697319376010925e-07
Mean Squarred Error - Test: 0.00027266583334823006
R2 Score- Train-0.9994737318741662
R2 Score- Test-0.8327084055084006
CPU times: total: 4.81 s
Wall time: 929 ms
```

Test Data Prediction Output

In [58]:
```
submit_lat = df_test_pred_lr.T[1]
submit_lon = df_test_pred_lr.T[0]
```

In [59]:
```
submit_lat.shape
df_test_pred_gb.shape
y_test_pred_gb.shape, df_test_pred_gb.shape,X_test.shape
```

Out[59]:
```
((1800, 2), (320, 2), (320, 8))
```

In [60]:
```
submit = pd.DataFrame({"TRIP_ID":df_test_ml["TRIP_ID"], "LATITUDE":submit_lat, "LONGITUDE":su
```

In [61]:
```
submit.head()
```

Out[61]:

	TRIP_ID	LATITUDE	LONGITUDE
0	T1	41.146623	-8.584884
1	T2	41.163597	-8.601894
2	T3	41.167719	-8.574903
3	T4	41.140980	-8.614638
4	T5	41.148036	-8.619894

In [62]:
```
submit.to_csv("D:\Projects\Taxi_Trajectory/Output_test.csv", index=False)
```