```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\ranit\anaconda3\lib\site-packages (2.0.3)
Requirement already satisfied: numpy in c:\users\ranit\anaconda3\lib\site-packages (from xgboost) (1.20.3)
Requirement already satisfied: scipy in c:\users\ranit\anaconda3\lib\site-packages (from xgboost) (1.10.1)

[notice] A new release of pip is available: 22.0.4 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
!pip install lightgbm
```

```
Requirement already satisfied: lightgbm in c:\users\ranit\anaconda3\lib\site-packages (4.3.0)
Requirement already satisfied: numpy in c:\users\ranit\anaconda3\lib\site-packages (from lightgbm) (1.20.3)
Requirement already satisfied: scipy in c:\users\ranit\anaconda3\lib\site-packages (from lightgbm) (1.10.1)

[notice] A new release of pip is available: 22.0.4 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
!pip install statsmodels
```

```
Requirement already satisfied: statsmodels in c:\users\ranit\anaconda3\lib\site-packages (0.12.2)
Requirement already satisfied: numpy>=1.15 in c:\users\ranit\anaconda3\lib\site-packages (from statsmodels) (1.20.3)
Requirement already satisfied: scipy>=1.1 in c:\users\ranit\anaconda3\lib\site-packages (from statsmodels) (1.10.1)
Requirement already satisfied: pandas>=0.21 in c:\users\ranit\anaconda3\lib\site-packages (from statsmodels) (1.4.4)
Requirement already satisfied: patsy>=0.5 in c:\users\ranit\anaconda3\lib\site-packages (from statsmodels) (0.5.2)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\ranit\anaconda3\lib\site-packages (from pandas>=0.21->statsmodels) (2.
Requirement already satisfied: pytz>=2020.1 in c:\users\ranit\anaconda3\lib\site-packages (from pandas>=0.21->statsmodels) (2023.3.post1
Requirement already satisfied: six in c:\users\ranit\anaconda3\lib\site-packages (from patsy>=0.5->statsmodels) (1.16.0)

[notice] A new release of pip is available: 22.0.4 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
import  scipy.signal.signaltools

def _centered(arr, newsize):
    # Return the center newsize portion of the array.
    newsize = np.asarray(newsize)
    currsize = np.array(arr.shape)
    startind = (currsize - newsize) // 2
    endind = startind + newsize
    myslice = [slice(startind[k], endind[k]) for k in range(len(endind))]
    return arr[tuple(myslice)]

scipy.signal.signaltools._centered = _centered
```

```
import sklearn
import lightgbm as lgb
import pandas as pd
from pylab import rcParams
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import r2_score
from sklearn.model_selection import GridSearchCV
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_absolute_error as mae
```

```
C:\Users\ranit\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:7: FutureWarning: pandas.Int64Index is deprecated and will
  from pandas import (to_datetime, Int64Index, DatetimeIndex, Period,
C:\Users\ranit\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:7: FutureWarning: pandas.Float64Index is deprecated and wil
  from pandas import (to_datetime, Int64Index, DatetimeIndex, Period,
```

```
rcParams['figure.figsize']=12,5
```

Goal:Analyze the weather data and find insights that could help predict visitation patterns at a park.

We are tackling a multivariate time series forecasting problem. Our approach will concentrate on feature engineering with weather data to identify seasonal patterns and incorporate features based on it (e.g., a seasonal feature). Additionally, we will include lag values since today's number of visitors is likely dependent on yesterday's count (not independent). We will also address null values and manage the large number of features by removing them accordingly.

FORECASTING MODELS WILL FOCUS ON USING TREE BASED REGRESSION WITH BOOSTING AND ENSEMBLES AND A METRIC OF Mean absolute error for performance.

MAE is better suited for forecasting evaluations when robustness to outliers is paramount, as it treats all errors equally without squaring them. Its simplicity and interpretability make it a preferred choice when communicating forecasting performance, especially in scenarios where outliers can heavily skew results or when equal weighting of errors is desired.

Double-click (or enter) to edit

```
park_visits=pd.read_csv('park_visitation.csv') #read csv's
weather_data=pd.read_csv('weather_data.csv')
```

DATASET STRUCTURE ANALYSIS

```
weather_data_cols=set(weather_data.columns) #exploring weather_data columns
len(weather_data)
```

```
1096
```

```
weather_data['TEMPERATURE_HEAT_INDEX_24HR_DEP'] #lot of empty nulls in this feature.lets explore this null issue more
```

```
0      NaN
1      NaN
2      NaN
3      NaN
4      NaN
        ..
1091   NaN
1092   NaN
1093   NaN
1094   NaN
1095   NaN
Name: TEMPERATURE_HEAT_INDEX_24HR_DEP, Length: 1096, dtype: float64
```

```
#rename weather data date column from DATE_CALENDAR TO DATE
weather_data.rename(columns={'DATE_CALENDAR': 'DATE'}, inplace=True)
weather_data.head()
```

	DATE	CLOUD_BASE_HEIGHT_24HR_DEP	CLOUD_BASE_HEIGHT_AVG	CLOUD_BASE_HEIGHT_MAX	CLOU
0	2021-03-13	-1376	1063.0	1063.0	
1	2021-03-14	255	1318.0	2482.0	
2	2021-03-15	7448	8767.0	11406.0	
3	2021-03-16	-6705	2061.0	5232.0	
4	2021-03-17	639	2700.0	7604.0	

5 rows × 167 columns

```
list(weather_data.columns) #LOTS OF FEATURES !!
```

```
['DATE',
 'CLOUD_BASE_HEIGHT_24HR_DEP',
 'CLOUD_BASE_HEIGHT_AVG',
 'CLOUD_BASE_HEIGHT_MAX',
 'CLOUD_BASE_HEIGHT_MIN',
 'CLOUD_COVER_24HR_DEP',
 'CLOUD_COVER_AVG',
 'CLOUD_COVER_MAX',
 'CLOUD_COVER_MIN',
 'CLOUD_COVER_PERC_24HR_DEP',
 'CLOUD_COVER_PERC_AVG',
 'CLOUD_COVER_PERC_MAX',
 'CLOUD_COVER_PERC_MIN',
 'DEGREE_DAYS_COOLING',
 'DEGREE_DAYS_EFFECTIVE',
 'DEGREE_DAYS_FREEZING',
 'DEGREE_DAYS_GROWING',
 'DEGREE_DAYS_HEATING',
 'EVAPOTRANSPIRATION_LWE_TOTAL',
 'HAS_FREEZING_RAIN',
 'FREEZING_RAIN_LWE_TOTAL',
 'FREEZING_RAIN_LWE_RATE_AVG',
 'FREEZING_RAIN_LWE_RATE_MAX',
 'FREEZING_RAIN_LWE_RATE_MIN',
 'HUMIDITY_RELATIVE_24HR_DEP',
 'HUMIDITY_RELATIVE_AVG',
 'HUMIDITY_RELATIVE_MAX',
 'HUMIDITY_RELATIVE_MIN',
 'HAS_ICE',
 'ICE_LWE_TOTAL',
 'ICE_LWE_RATE_AVG',
 'ICE_LWE_RATE_MAX',
 'ICE_LWE_RATE_MIN',
 'INDEX_UV_24HR_DEP',
 'INDEX_UV_AVG',
 'INDEX_UV_MAX',
 'INDEX_UV_MIN',
 'MINUTES_OF_FREEZING_RAIN_TOTAL',
 'MINUTES_OF_ICE_TOTAL',
 'MINUTES_OF_PRECIPITATION_TOTAL',
 'MINUTES_OF_RAIN_TOTAL',
 'MINUTES_OF_SLEET_TOTAL',
 'MINUTES_OF_SNOW_TOTAL',
 'MINUTES_OF_SUN_TOTAL',
 'MOISTURE_SOIL_AVG',
 'MOISTURE_SOIL_MAX',
 'MOISTURE_SOIL_MIN',
 'HAS_PRECIPITATION',
 'PRECIPITATION_INTENSITY_MAX',
 'PRECIPITATION_LWE_TOTAL',
 'PRECIPITATION_LWE_RATE_AVG',
 'PRECIPITATION_LWE_RATE_MAX',
 'PRECIPITATION_LWE_RATE_MIN',
 'PRECIPITATION_TYPE_PREDOMINANT',
 'PRECIPITATION_TYPE_DESC_PREDOMINANT',
 'PRESSURE_24HR_DEP',
 'PRESSURE_AVG',
 'PRESSURE_MAX',
```

```
object_columns=weather_data[list(weather_data.select_dtypes(include=['object']).columns)] #all columns of type object
len(object_columns)
```

```
1096
```

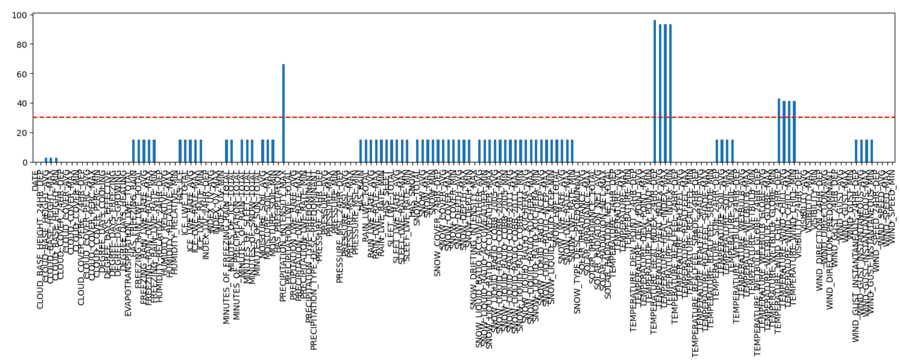
```
object_columns.isna().sum() #checking NULLS IN OBJECT COLUMNS
```

```
DATE                                  0
HAS_FREEZING_RAIN                   164
HAS_ICE                             164
PRECIPITATION_INTENSITY_MAX         727
PRECIPITATION_TYPE_DESC_PREDOMINANT   0
HAS_RAIN                            164
HAS_SLEET                           164
SNOW_DRIFTING_INTENSITY_MAX           0
SNOW_TYPE_DESC_PREDOMINANT            0
dtype: int64
```

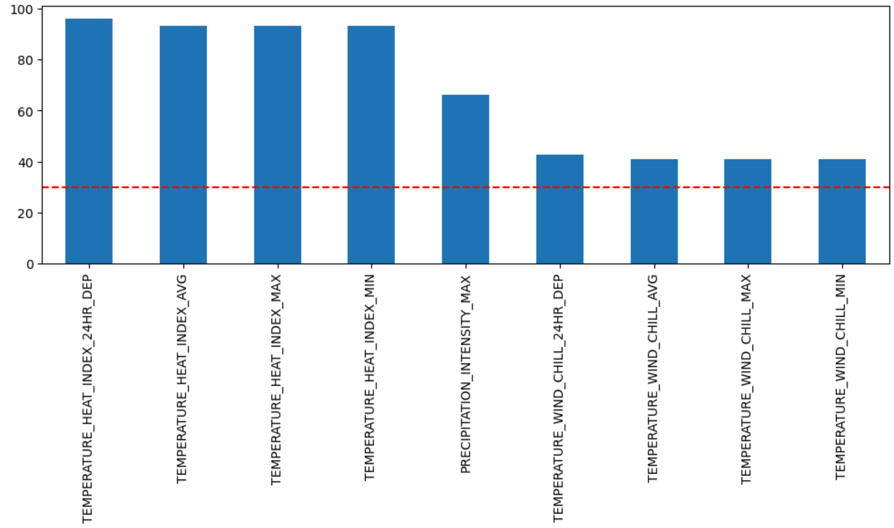
DEALING WITH NAN VALUES

THE DIAGRAM BELOW SHOWS ALL THE FEATURES WITH COLUMNS HAVING MORE THAN 30% NULL VALUES.WE USE A NULL THRESHOLD TO ERADICATE THESE COLUMNS.OUR NULL THRESHOLD IS SET AT 30%

```
empty_values_per_column=(weather_data.isna().sum()/len(weather_data))*100 #eradicate columns with empty cells more than 30%
plt.figure(figsize=(15,6))  # Adjust figsize as needed
empty_values_per_column.plot(kind='bar')
plt.axhline(y=30, color='r', linestyle='--') #30% threshold
plt.tight_layout()
```




```
#plotting the columns with empty values over 30% .
empty_vals_top_9=empty_values_per_column.nlargest(9) #see that the temperature _heat _index and wind_chills are missing.might be defective s
plt.figure(figsize=(10,6))  # Adjust figsize as needed
empty_vals_top_9.plot(kind='bar',)
plt.axhline(y=30, color='r', linestyle='--') #30% threshold
plt.tight_layout()
```




```
#dropping these columns
weather_data.drop(columns=list(empty_vals_top_9.index), inplace=True)
```

```
num_columns=weather_data[list(weather_data.select_dtypes(include=['number']).columns)] #exploring number columns
num_columns
```

	CLOUD_BASE_HEIGHT_24HR_DEP	CLOUD_BASE_HEIGHT_AVG	CLOUD_BASE_HEIGHT_MAX	CLOUD_B
0	-1376	1063.0	1063.0	
1	255	1318.0	2482.0	
2	7448	8767.0	11406.0	
3	-6705	2061.0	5232.0	
4	639	2700.0	7604.0	
...	...	...	...	
1091	-499	3288.0	6096.0	
1092	-2364	924.0	7162.0	
1093	951	1875.0	9448.0	
1094	5515	7391.0	8686.0	
1095	4801	NaN	NaN	

1096 rows × 148 columns

```
weather_data.head(10) #exploring weather_data
```

	DATE	CLOUD_BASE_HEIGHT_24HR_DEP	CLOUD_BASE_HEIGHT_AVG	CLOUD_BASE_HEIGHT_MAX	CLOU
0	2021-03-13	-1376	1063.0	1063.0	
1	2021-03-14	255	1318.0	2482.0	
2	2021-03-15	7448	8767.0	11406.0	
3	2021-03-16	-6705	2061.0	5232.0	
4	2021-03-17	639	2700.0	7604.0	
5	2021-03-18	774	3475.0	7198.0	
6	2021-03-19	3927	7402.0	7402.0	
7	2021-03-20	4789	NaN	NaN	
8	2021-03-21	0	NaN	NaN	
9	2021-03-22	-4037	8154.0	9551.0	

10 rows × 158 columns

```
weather_data.isna().sum().nlargest(35)    #lots of null values in these columns
```

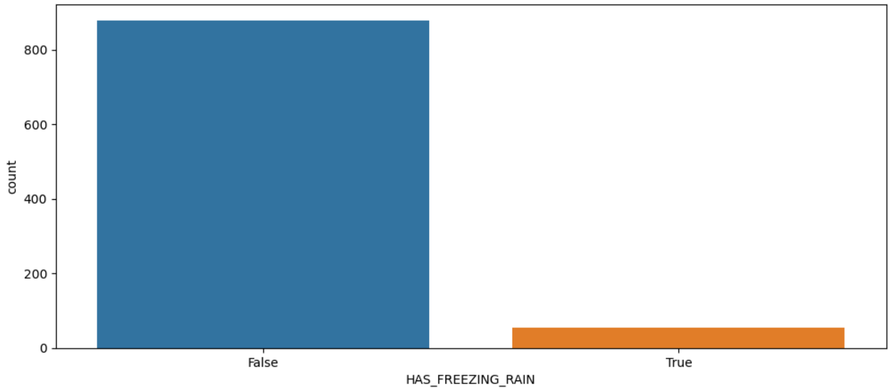
```
HAS_FREEZING_RAIN               164
FREEZING_RAIN_LWE_TOTAL         164
FREEZING_RAIN_LWE_RATE_AVG      164
FREEZING_RAIN_LWE_RATE_MAX      164
FREEZING_RAIN_LWE_RATE_MIN      164
HAS_ICE                         164
ICE_LWE_TOTAL                   164
ICE_LWE_RATE_AVG                164
ICE_LWE_RATE_MAX                164
ICE_LWE_RATE_MIN                164
MINUTES_OF_FREEZING_RAIN_TOTAL  164
MINUTES_OF_ICE_TOTAL            164
MINUTES_OF_RAIN_TOTAL           164
MINUTES_OF_SLEET_TOTAL          164
MINUTES_OF_SNOW_TOTAL           164
MOISTURE_SOIL_AVG               164
MOISTURE_SOIL_MAX               164
MOISTURE_SOIL_MIN               164
HAS_RAIN                        164
RAIN_LWE_TOTAL                  164
RAIN_LWE_RATE_AVG               164
RAIN_LWE_RATE_MAX               164
RAIN_LWE_RATE_MIN               164
HAS_SLEET                       164
SLEET_LWE_TOTAL                 164
SLEET_LWE_RATE_AVG              164
SLEET_LWE_RATE_MAX              164
SLEET_LWE_RATE_MIN              164
SNOW_TOTAL                      164
SNOW_AVG                        164
SNOW_MAX                        164
SNOW_MIN                        164
SNOW_COVER_24HR_DEP             164
SNOW_COVER_AVG                  164
SNOW_COVER_MAX                  164
dtype: int64
```

```
weather_data['HAS_FREEZING_RAIN'].value_counts() #freezing rain data very skewed.this is the case for several features
```

```
False    878
True      54
Name: HAS_FREEZING_RAIN, dtype: int64
```

```
sns.countplot(x='HAS_FREEZING_RAIN', data=weather_data) #this represents real life conditions since it doesnt have freezing rain so often
```

```
<AxesSubplot:xlabel='HAS_FREEZING_RAIN', ylabel='count'>
```




```
corr_matrix =weather_data.corr().abs() #exploring correlations .too many features for heatmaps just yet
corr_matrix['CLOUD_BASE_HEIGHT_AVG']
```

```
CLOUD_BASE_HEIGHT_24HR_DEP     0.422040
CLOUD_BASE_HEIGHT_AVG          1.000000
CLOUD_BASE_HEIGHT_MAX          0.737705
CLOUD_BASE_HEIGHT_MIN          0.810733
CLOUD_COVER_24HR_DEP           0.068738
                                 ...
WIND_GUST_INSTANTANEOUS_MIN    0.143329
WIND_SPEED_24HR_DEP            0.077989
WIND_SPEED_AVG                 0.171400
WIND_SPEED_MAX                 0.143450
WIND_SPEED_MIN                 0.172095
Name: CLOUD_BASE_HEIGHT_AVG, Length: 150, dtype: float64
```

DEALING WITH MULTICOLLINEARITY

Multicollinearity occurs when independent variables in a regression model are highly correlated with each other. This can lead to issues in the interpretation of coefficients and can inflate the standard errors of the coefficients, making them unreliable. To address multicollinearity, create a correlation map between features to identify highly correlated columns, then remove one of each correlated pair. This helps reduce redundancy and ensures the robustness of the regression model, preserving the independence of predictors while maintaining model interpretability and performance.

```
threshold = 0.75  #set multicollinearity threshold to 75%
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
to_drop = [column for column in upper.columns if any(upper[column] > threshold)]
weather_data_filtered=weather_data.drop(to_drop, axis=1)
```

```
list(weather_data.columns) #CURRENT WEATHER COLUMNS
```

```
['DATE',
 'CLOUD_BASE_HEIGHT_24HR_DEP',
 'CLOUD_BASE_HEIGHT_AVG',
 'CLOUD_BASE_HEIGHT_MAX',
 'CLOUD_BASE_HEIGHT_MIN',
 'CLOUD_COVER_24HR_DEP',
 'CLOUD_COVER_AVG',
 'CLOUD_COVER_MAX',
 'CLOUD_COVER_MIN',
 'CLOUD_COVER_PERC_24HR_DEP',
 'CLOUD_COVER_PERC_AVG',
 'CLOUD_COVER_PERC_MAX',
```

```
'CLOUD_COVER_PERC_MIN',
'DEGREE_DAYS_COOLING',
'DEGREE_DAYS_EFFECTIVE',
'DEGREE_DAYS_FREEZING',
'DEGREE_DAYS_GROWING',
'DEGREE_DAYS_HEATING',
'EVAPOTRANSPIRATION_LWE_TOTAL',
'HAS_FREEZING_RAIN',
'FREEZING_RAIN_LWE_TOTAL',
'FREEZING_RAIN_LWE_RATE_AVG',
'FREEZING_RAIN_LWE_RATE_MAX',
'FREEZING_RAIN_LWE_RATE_MIN',
'HUMIDITY_RELATIVE_24HR_DEP',
'HUMIDITY_RELATIVE_AVG',
'HUMIDITY_RELATIVE_MAX',
'HUMIDITY_RELATIVE_MIN',
'HAS_ICE',
'ICE_LWE_TOTAL',
'ICE_LWE_RATE_AVG',
'ICE_LWE_RATE_MAX',
'ICE_LWE_RATE_MIN',
'INDEX_UV_24HR_DEP',
'INDEX_UV_AVG',
'INDEX_UV_MAX',
'INDEX_UV_MIN',
'MINUTES_OF_FREEZING_RAIN_TOTAL',
'MINUTES_OF_ICE_TOTAL',
'MINUTES_OF_PRECIPITATION_TOTAL',
'MINUTES_OF_RAIN_TOTAL',
'MINUTES_OF_SLEET_TOTAL',
'MINUTES_OF_SNOW_TOTAL',
'MINUTES_OF_SUN_TOTAL',
'MOISTURE_SOIL_AVG',
'MOISTURE_SOIL_MAX',
'MOISTURE_SOIL_MIN',
'HAS_PRECIPITATION',
'PRECIPITATION_LWE_TOTAL',
'PRECIPITATION_LWE_RATE_AVG',
'PRECIPITATION_LWE_RATE_MAX',
'PRECIPITATION_LWE_RATE_MIN',
'PRECIPITATION_TYPE_PREDOMINANT',
'PRECIPITATION_TYPE_DESC_PREDOMINANT',
'PRESSURE_24HR_DEP',
'PRESSURE_AVG',
'PRESSURE_MAX',
'PRESSURE_MIN',
```

```
weather_data_filtered.head(10) #current dataframe
```

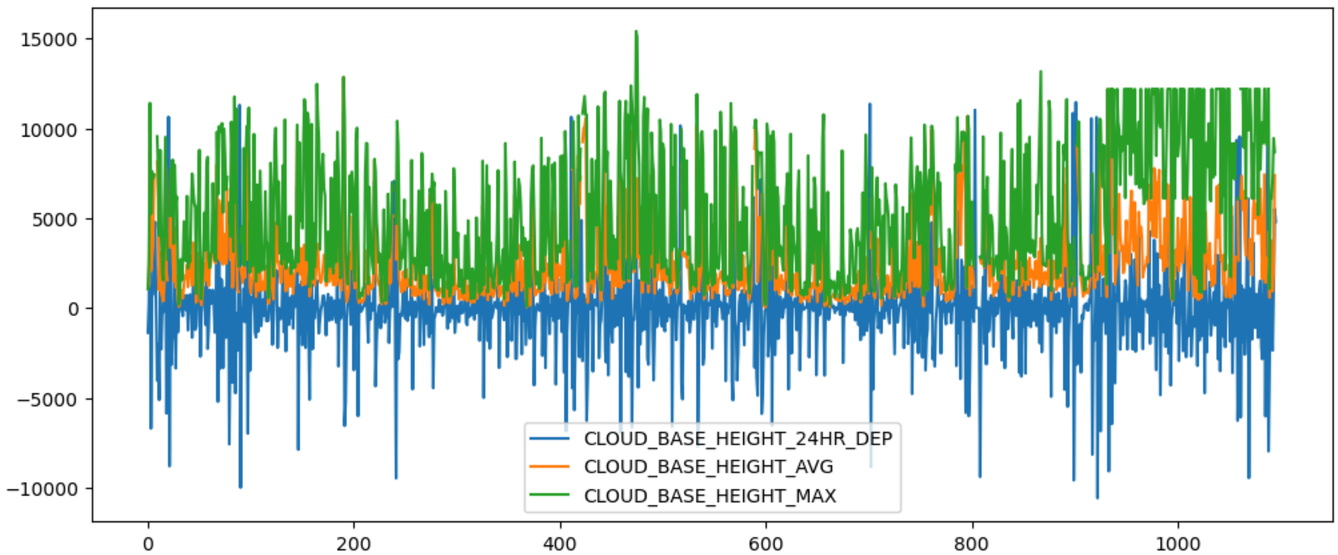
	DATE	CLOUD_BASE_HEIGHT_24HR_DEP	CLOUD_BASE_HEIGHT_AVG	CLOUD_BASE_HEIGHT_MAX	CLOUD_COVER_24HR_DEP	CLOUD_COVER_AVG	CLOUD_COVER_M/
0	2021-03-13	-1376	1063.0	1063.0	-0.01	0.18	0.:
1	2021-03-14	255	1318.0	2482.0	-0.03	0.15	0.8
2	2021-03-15	7448	8767.0	11406.0	0.18	0.33	1.(
3	2021-03-16	-6705	2061.0	5232.0	0.55	0.88	1.(
4	2021-03-17	639	2700.0	7604.0	-0.07	0.81	1.(
5	2021-03-18	774	3475.0	7198.0	0.17	0.98	1.(
6	2021-03-19	3927	7402.0	7402.0	-0.98	0.00	0.(
7	2021-03-20	4789	NaN	NaN	0.01	0.01	0.(
8	2021-03-21	0	NaN	NaN	-0.01	0.00	0.(
9	2021-03-22	-4037	8154.0	9551.0	0.36	0.36	0.!

10 rows × 61 columns

```
# another feature engineering CLOUD_BASE.We see that cloud_base_height_avg is the just the average of cloud_bases
```

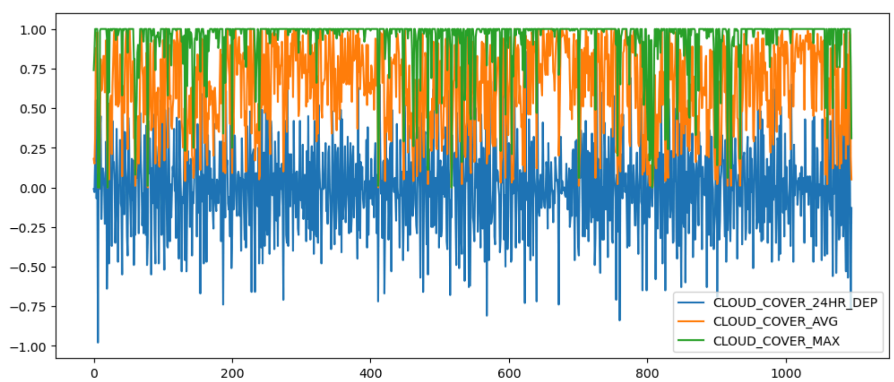
```
#so we can drop theother cloud_base features and focus on one.There are similar other features I show below
def plot_min_max_avg(df,min_name,avg_name,max_name):

    plt.plot(df[min_name], label=min_name)
    plt.plot(weather_data_filtered[avg_name], label=avg_name)
    plt.plot(weather_data_filtered[max_name], label=max_name)
    plt.legend()
```

```
plot_min_max_avg(weather_data_filtered,'CLOUD_BASE_HEIGHT_24HR_DEP','CLOUD_BASE_HEIGHT_AVG','CLOUD_BASE_HEIGHT_MAX')
```




```
weather_data_filtered.drop(columns=['CLOUD_BASE_HEIGHT_24HR_DEP','CLOUD_BASE_HEIGHT_MAX'],inplace=True) #dropping similar features
```

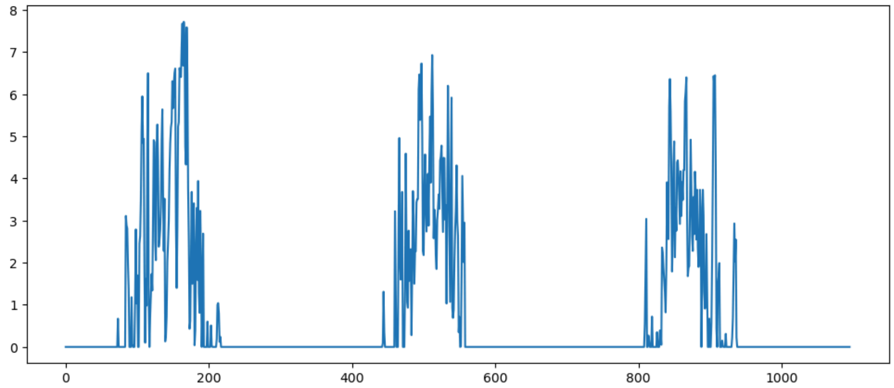
```
plot_min_max_avg(weather_data_filtered,'CLOUD_COVER_24HR_DEP','CLOUD_COVER_AVG','CLOUD_COVER_MAX')#dropping more similar features
```




```
weather_data_filtered.drop(columns=['CLOUD_COVER_24HR_DEP','CLOUD_COVER_MAX','CLOUD_COVER_MIN'],inplace=True)#dropping similar columns
```

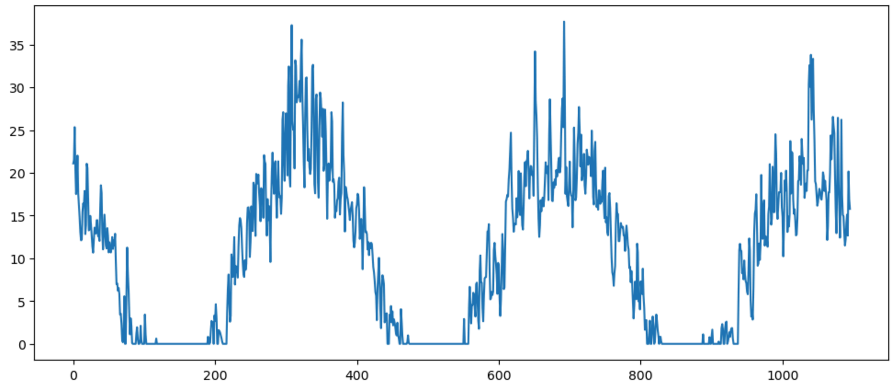
```
weather_data_filtered['DEGREE_DAYS_COOLING'].plot()#we see a seasonal trend in some of the features
```

<AxesSubplot:>




```
weather_data_filtered['DEGREE_DAYS_EFFECTIVE'].plot()#seasonal trend
```

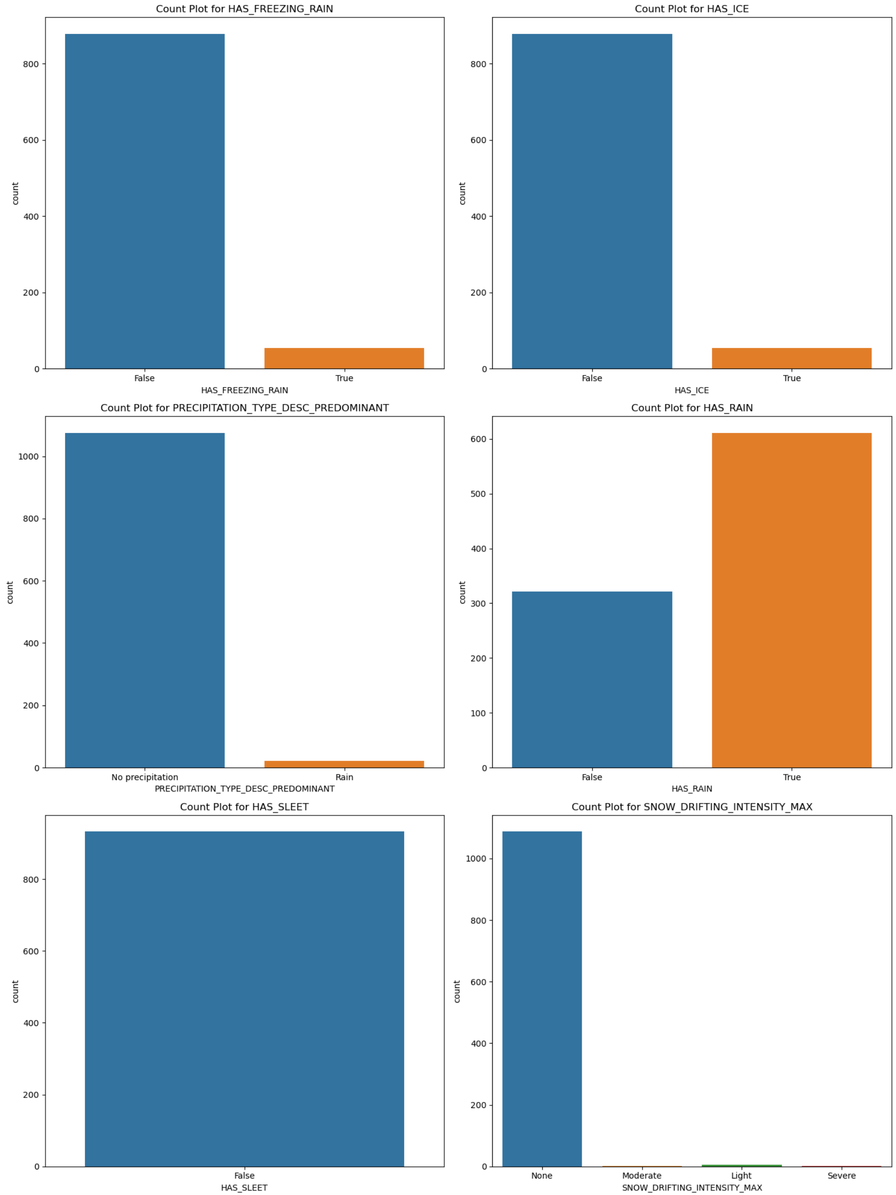
<AxesSubplot:>



CATEGORICAL COUNTPLOTS VISUALIZATION

WE SEE THE SKEWNESS FOR A LOT OF OUR FEATURES.THIS IS A REPRESENTATIVE OF THE REAL WEATHER NOT UNDERSAMPLING SINCE MOST PLACES DONOT HAVE FREEZING RAINS AND ICE FOR MOST OF THE YEAR AND HENCE NOT AN ISSUE.HOWEVER, THERE ARE MULTIPLE FEATURES REPRESENTING SIMILAR THINGS. FREEZING RAIN AND HAS_ICE HAVE SIMILAR VALUES BECAUSE THEY MOSTLY OCCUR DURING THE SAME TIME AND ARE VERY CORRELATED.WE REMOVE SOME OF THEM.HAS_SLEET HAS ONLY ONE VALUE AND HENCE COMPLETELY USELESS FOR OUR ANALYSIS

```
#PLOTTING CATEGORICAL VARIABLES COUNT PLOTS
object_columns=weather_data_filtered[list(weather_data.select_dtypes(include=['object']).columns)]

fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(15, 20))

# Loop through each column and create count plot
for i, column in enumerate(object_columns.columns[1:-1]):
    sns.countplot(data=object_columns, x=column, ax=axes[i//2, i%2])
    axes[i//2, i%2].set_title(f'Count Plot for {column}')

plt.tight_layout()
plt.show()
```


Count Plot for HAS_FREEZING_RAIN
count
HAS_FREEZING_RAIN
False
True
Count Plot for HAS_ICE
count
HAS_ICE
False
True
Count Plot for PRECIPITATION_TYPE_DESC_PREDOMINANT
count
PRECIPITATION_TYPE_DESC_PREDOMINANT
No precipitation
Rain
Count Plot for HAS_RAIN
count
HAS_RAIN
False
True
Count Plot for HAS_SLEET
count
HAS_SLEET
False
Count Plot for SNOW_DRIFTING_INTENSITY_MAX
count
SNOW_DRIFTING_INTENSITY_MAX
None
Moderate
Light
Severe

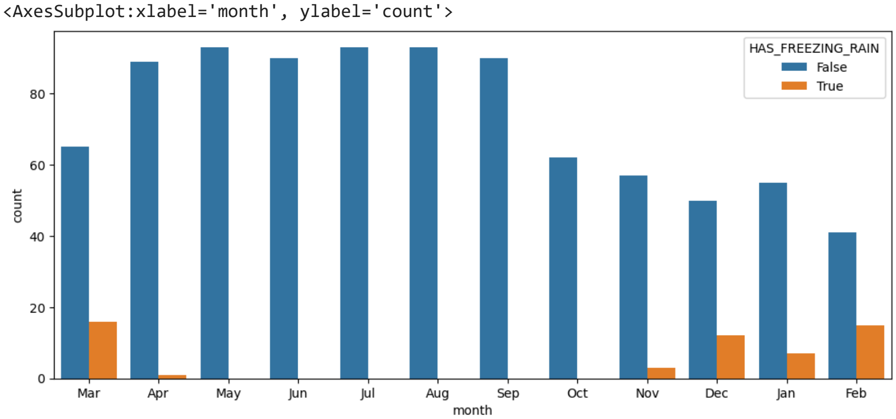
```
object_columns.columns[1:] #ALL THE CATEGORICAL COLUMNS
```

```
Index(['HAS_FREEZING_RAIN', 'HAS_ICE', 'PRECIPITATION_TYPE_DESC_PREDOMINANT',
       'HAS_RAIN', 'HAS_SLEET', 'SNOW_DRIFTING_INTENSITY_MAX',
       'SNOW_TYPE_DESC_PREDOMINANT'],
      dtype='object')
```

```
#VISUALIZING SOME OF THESE VARIABLES WITH THE MONTH.CREATE THE MONTH FEATURE MANUALLY
weather_data_filtered['month']=pd.DatetimeIndex(weather_data_filtered['DATE']).month
month_mapping = {1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr', 5: 'May', 6: 'Jun',
                 7: 'Jul', 8: 'Aug', 9: 'Sep', 10: 'Oct', 11: 'Nov', 12: 'Dec'}
weather_data_filtered['month'] = weather_data_filtered['month'].apply(lambda x: month_mapping.get(x, x))
```

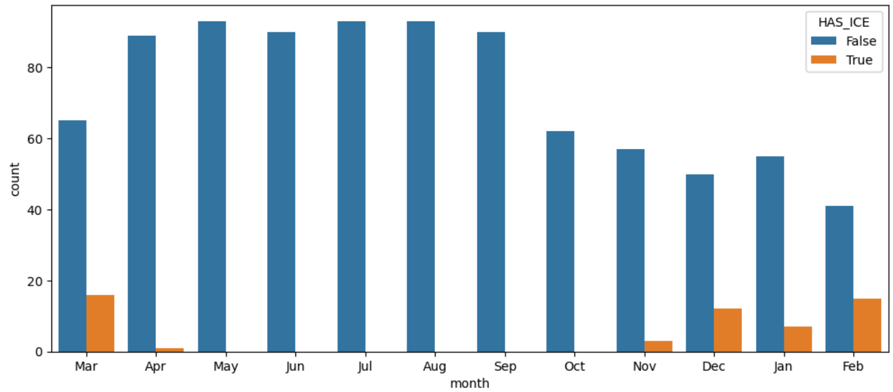
```
#Shows the occurence of freezing rain during the year

sns.countplot(data=weather_data_filtered, x='month', hue='HAS_FREEZING_RAIN')
```

```
<AxesSubplot:xlabel='month', ylabel='count'>
```



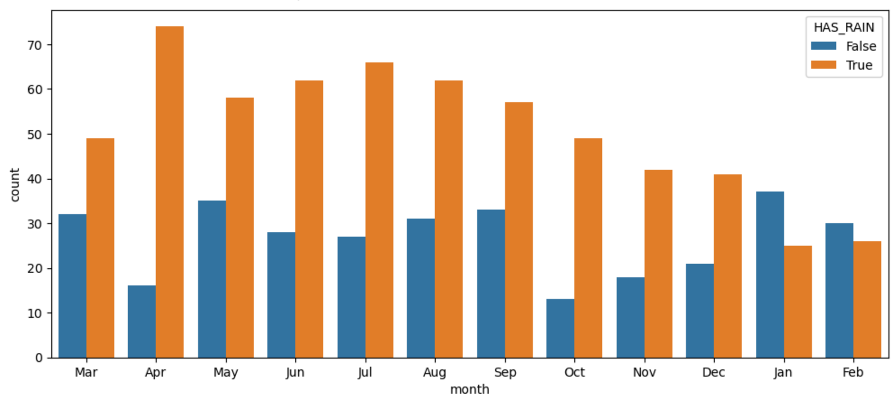
```
#Shows the occurence of has ice during the year.EXACTLY THE SAME AS FREEZING RAIN.remove one of them!

sns.countplot(data=weather_data_filtered, x='month', hue='HAS_ICE')
```

```
<AxesSubplot:xlabel='month', ylabel='count'>
```




```
#Data shows that the place gets a lot of rain during the year.This can be important for mapping the environmental factors
sns.countplot(data=weather_data_filtered, x='month', hue='HAS_RAIN')
```

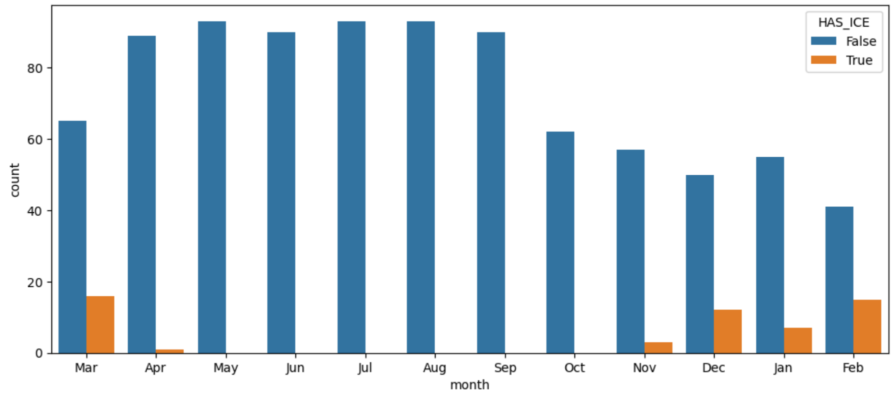
```
<AxesSubplot:xlabel='month', ylabel='count'>
```



We see that the plots for freezing rain and ice are pretty much the same!

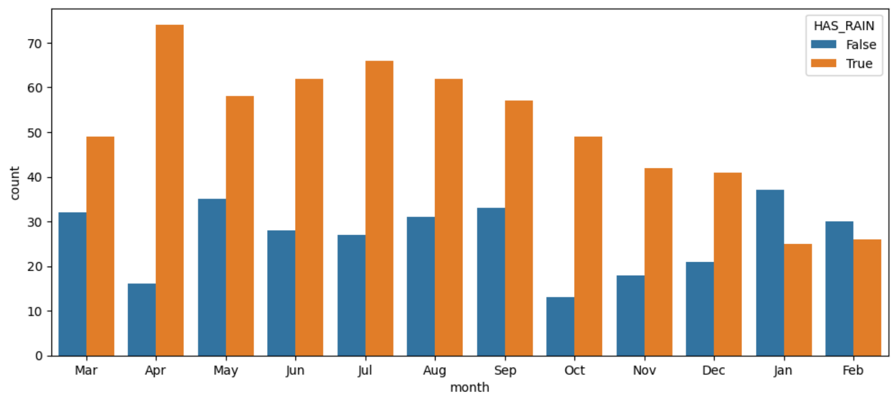
```
sns.countplot(data=weather_data_filtered, x='month', hue='HAS_ICE')
```

```
<AxesSubplot:xlabel='month', ylabel='count'>
```




```
sns.countplot(data=weather_data_filtered, x='month', hue='HAS_RAIN')
```

```
<AxesSubplot:xlabel='month', ylabel='count'>
```




```
#dropping all the correlated columns or columns with no information gain.
weather_data_filtered.drop(columns=['HAS_SLEET','PRECIPITATION_TYPE_DESC_PREDOMINANT','HAS_FREEZING_RAIN','SNOW_DRIFTING_INTENSITY_MAX'],inp
```

```
#perfect example of one column with no information gain since its only 0'S.
weather_data_filtered['FREEZING_RAIN_LWE_RATE_MIN'].value_counts()
```

```
0.0    932
Name: FREEZING_RAIN_LWE_RATE_MIN, dtype: int64
```

ONE HOT ENCODING CATEGORICAL VARIABLES

```
#converting categorical variables into one hot encoded forms
object_columns = weather_data_filtered.select_dtypes(include=['object']).drop(columns=['month','DATE'])

weather_data_encoded = pd.get_dummies(weather_data_filtered, columns=object_columns.columns)
```

```
weather_data_encoded.head(10)
```

	DATE	CLOUD_BASE_HEIGHT_AVG	CLOUD_COVER_AVG	DEGREE_DAYS_COOLING	DEGREE_DAYS_EFFEC
0	2021-03-13	1063.0	0.18	0.0	
1	2021-03-14	1318.0	0.15	0.0	
2	2021-03-15	8767.0	0.33	0.0	
3	2021-03-16	2061.0	0.88	0.0	
4	2021-03-17	2700.0	0.81	0.0	
5	2021-03-18	3475.0	0.98	0.0	
6	2021-03-19	7402.0	0.00	0.0	
7	2021-03-20	NaN	0.01	0.0	
8	2021-03-21	NaN	0.00	0.0	
9	2021-03-22	8154.0	0.36	0.0	

10 rows × 55 columns

```
weather_data_cols-set(weather_data.columns) #the current colu
```

```
{'DATE_CALENDAR',
 'PRECIPITATION_INTENSITY_MAX',
 'TEMPERATURE_HEAT_INDEX_24HR_DEP',
 'TEMPERATURE_HEAT_INDEX_AVG',
 'TEMPERATURE_HEAT_INDEX_MAX',
 'TEMPERATURE_HEAT_INDEX_MIN',
 'TEMPERATURE_WIND_CHILL_24HR_DEP',
 'TEMPERATURE_WIND_CHILL_AVG',
 'TEMPERATURE_WIND_CHILL_MAX',
 'TEMPERATURE_WIND_CHILL_MIN'}
```

```
weather_data.corr()
```

	CLOUD_BASE_HEIGHT_24HR_DEP	CLOUD_BASE_HEIGHT_AVG	CLOUD
CLOUD_BASE_HEIGHT_24HR_DEP	1.000000	0.422040	
CLOUD_BASE_HEIGHT_AVG	0.422040	1.000000	
CLOUD_BASE_HEIGHT_MAX	0.268863	0.737705	
CLOUD_BASE_HEIGHT_MIN	0.347250	0.810733	
CLOUD_COVER_24HR_DEP	-0.271993	0.068738	
...	...	...	
WIND_GUST_INSTANTANEOUS_MIN	0.005563	-0.143329	
WIND_SPEED_24HR_DEP	-0.151455	-0.077989	
WIND_SPEED_AVG	-0.048618	-0.171400	
WIND_SPEED_MAX	-0.078692	-0.143450	
WIND_SPEED_MIN	-0.009569	-0.172095	

150 rows × 150 columns

```
weather_data_encoded.isna().sum().nlargest(30)
```

```
FREEZING_RAIN_LWE_TOTAL            164
FREEZING_RAIN_LWE_RATE_MIN         164
ICE_LWE_RATE_MIN                   164
```

```
MINUTES_OF_SLEET_TOTAL              164
MINUTES_OF_SNOW_TOTAL               164
MOISTURE_SOIL_MIN                   164
RAIN_LWE_RATE_MIN                   164
SLEET_LWE_TOTAL                     164
SLEET_LWE_RATE_AVG                  164
SLEET_LWE_RATE_MAX                  164
SLEET_LWE_RATE_MIN                  164
SNOW_MIN                            164
SNOW_COVER_24HR_DEP                 164
SNOW_COVER_AVG                      164
SNOW_COVER_MAX                      164
SNOW_COVER_MIN                      164
SNOW_DEPTH_AVG                      164
SNOW_DEPTH_MAX                      164
SNOW_DEPTH_MIN                      164
SNOW_LIQUID_RATIO_ACCUWEATHER_AVG   164
SNOW_LIQUID_RATIO_ACCUWEATHER_MIN   164
SNOW_LWE_RATE_MIN                   164
TEMPERATURE_SOIL_24HR_DEP           164
CLOUD_BASE_HEIGHT_AVG                28
DATE                                  0
CLOUD_COVER_AVG                       0
DEGREE_DAYS_COOLING                   0
DEGREE_DAYS_EFFECTIVE                 0
DEGREE_DAYS_FREEZING                  0
EVAPOTRANSPIRATION_LWE_TOTAL          0
dtype: int64
```

```
weather_data['HAS_SLEET'].value_counts() #useless hence remove all sleets.gives away area does not have sleet usually
```

```
False    932
Name: HAS_SLEET, dtype: int64
```

```
#exploring some of the nan values in these columns.shows percent empty in columns.
empty_values_per_column[(empty_values_per_column<20)&(empty_values_per_column>5)]
```

```
HAS_FREEZING_RAIN                  14.963504
FREEZING_RAIN_LWE_TOTAL            14.963504
FREEZING_RAIN_LWE_RATE_AVG         14.963504
FREEZING_RAIN_LWE_RATE_MAX         14.963504
FREEZING_RAIN_LWE_RATE_MIN         14.963504
                                     ...
TEMPERATURE_SOIL_MIN               14.963504
WIND_GUST_INSTANTANEOUS_24HR_DEP   14.963504
WIND_GUST_INSTANTANEOUS_AVG        14.963504
WIND_GUST_INSTANTANEOUS_MAX        14.963504
WIND_GUST_INSTANTANEOUS_MIN        14.963504
Length: 66, dtype: float64
```

EXPLORING PARK VISITOR TIME SERIES DATA

```
#using park_visits data
park_visits.head()
```

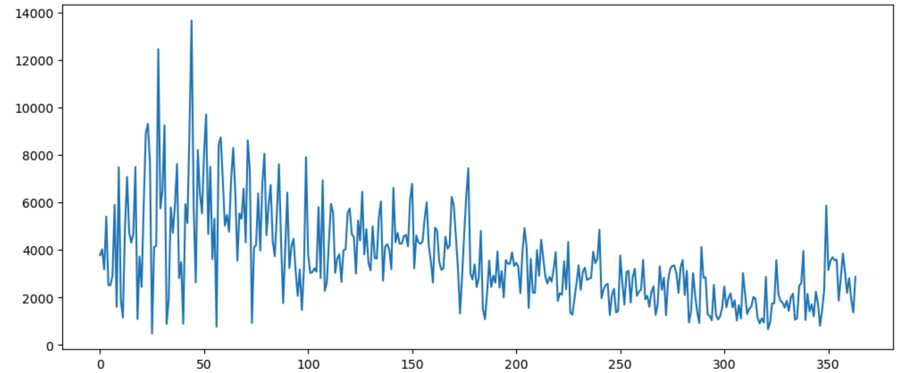
	DATE	ESTIMATED_VISITS
0	2021-04-01	3781
1	2021-04-02	4024
2	2021-04-03	3189
3	2021-04-04	5407
4	2021-04-05	2519

```
park_visits['DATE'] = pd.to_datetime(park_visits['DATE']) #converting dates
park_2 = park_visits.set_index('DATE').asfreq('D')
```

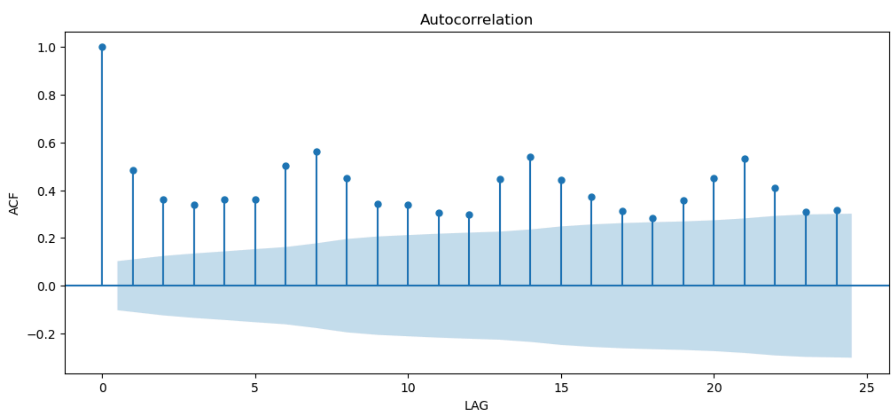
```
park_2['ESTIMATED_VISITS'].fillna(park_2['ESTIMATED_VISITS'].mean(),inplace=True) #imputing with mean
```

```
plt.plot(park_visits['ESTIMATED_VISITS']) #plotting the estimated visits.We see some seasonality in this plot.
```

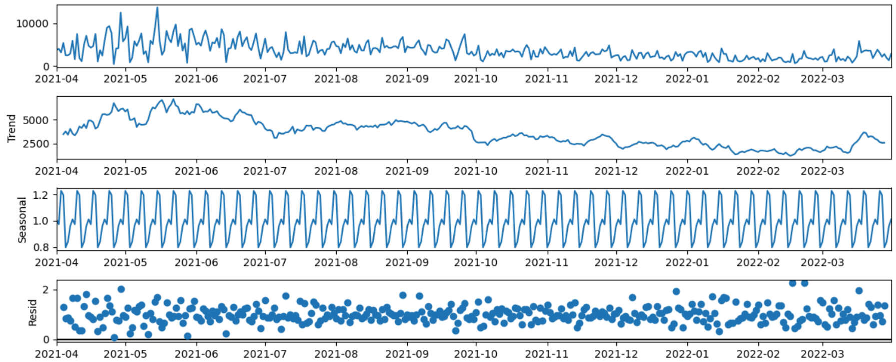
```
[<matplotlib.lines.Line2D at 0x1f21d8ab730>]
```




```
# This function is used to visualize the autocorrelation of a time series, which represents the correlation
# between a series and a lagged version of itself at different time lags.
#The plot helps in understanding the presence of patterns or dependencies within the time series data.
fig = plot_acf(park_visits['ESTIMATED_VISITS'], lags=24)
plt.xlabel('LAG')
plt.ylabel('ACF')
plt.show()
```




```
#USING ETS TO DECOMPOSE OUR TIME SERIES INTO TREND ,SEASONAL AND RESIDUAL PORTIONS.SHOWS CLEAR PRESENCE OF SEASONS.
ets = seasonal_decompose(park_2, model='multiplicative')
ets.plot();
```



```
#The code conducts an Augmented Dickey-Fuller test on the 'ESTIMATED_VISITS' time series data to assess its stationarity.
#It prints the p-value associated with the test, indicating the likelihood of the data being non-stationary; if the p-value is below a signi
result = adfuller(park_visits['ESTIMATED_VISITS'])
print('p-value: %f' % result[1])
```

```
p-value: 0.612862
```

```
#ADDING A SEASON FEATURE TO OUR DATASET SINCE IT SHOWS STRONG SEASONALITY
park_visits['month'] = park_visits['DATE'].dt.month
```

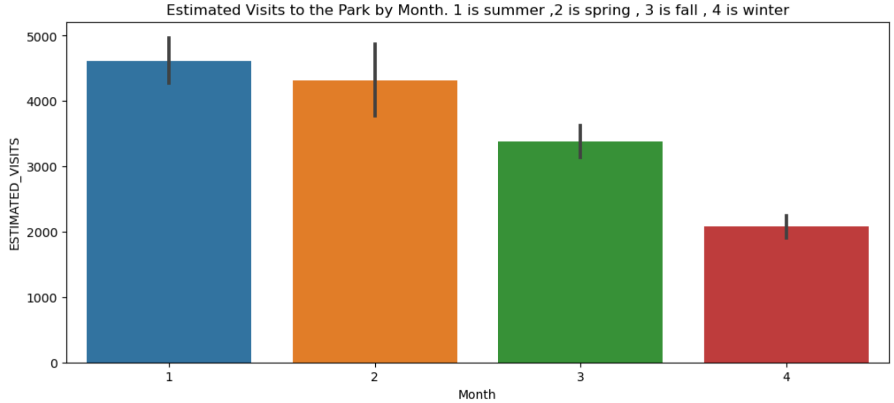
```
def find_season(val): #finds the season based on month number
    if val in [1, 2, 12]:
        return 4 #winter
    elif val in [3, 4, 5]:
        return 2 #spring
    elif val in [6, 7, 8]:
        return 1 #summer
    elif val in [9, 10, 11]:
        return 3 #fall
```

```
#Added a season feature
park_visits['month_val']=park_visits['month'].apply(find_season)
park_visits.head()
```

	DATE	ESTIMATED_VISITS	month	month_val
0	2021-04-01	3781	4	2
1	2021-04-02	4024	4	2
2	2021-04-03	3189	4	2
3	2021-04-04	5407	4	2
4	2021-04-05	2519	4	2

```
#we see that the highest number of visitors are during the summer and spring months

sns.barplot(data=park_visits,x='month_val',y='ESTIMATED_VISITS')
plt.xlabel('Month')
plt.title('Estimated Visits to the Park by Month. 1 is summer ,2 is spring , 3 is fall , 4 is winter')
```

```
Text(0.5, 1.0, 'Estimated Visits to the Park by Month. 1 is summer ,2 is spring , 3 is
fall , 4 is winter')
```




```
len(park_visits)
```

```
364
```

```
#dropping the column.not needed now
park_visits['DATE'].max()
park_visits.drop('month', axis=1, inplace=True)
```

```
#converting date
weather_data_encoded['DATE']=pd.to_datetime(weather_data['DATE'])
```

```
#merging the 2 dataframes together .left join on park_visits
merged_df = pd.merge(park_visits, weather_data_encoded, on='DATE', how='left')
merged_df.head()
```

	DATE	ESTIMATED_VISITS	month_val	CLOUD_BASE_HEIGHT_AVG	CLOUD_COVER_AVG	DEGREE_DA
0	2021-04-01	3781	2	1552.0	0.30	
1	2021-04-02	4024	2	NaN	0.00	
2	2021-04-03	3189	2	3385.0	0.53	
3	2021-04-04	5407	2	7547.0	0.05	
4	2021-04-05	2519	2	4407.0	0.46	

5 rows × 57 columns

```
merged_df['month_val'].value_counts()
```

```
1    92
2    91
3    91
4    90
Name: month_val, dtype: int64
```

USING DECISION TREE FOR FEATURE ENGINEERING TO CALCULATE FEATURE IMPORTANCES

In regression, decision trees provide information on feature importance by assessing how much each feature reduces the variance of the target variable across the splits. Features that result in larger reductions in variance are considered more important, as they contribute more to explaining the variance in the target variable.

```
clf = DecisionTreeClassifier()
merged_df=merged_df.dropna()
X=merged_df.drop(['ESTIMATED_VISITS','DATE',],axis=1)
Y=merged_df['ESTIMATED_VISITS']
```

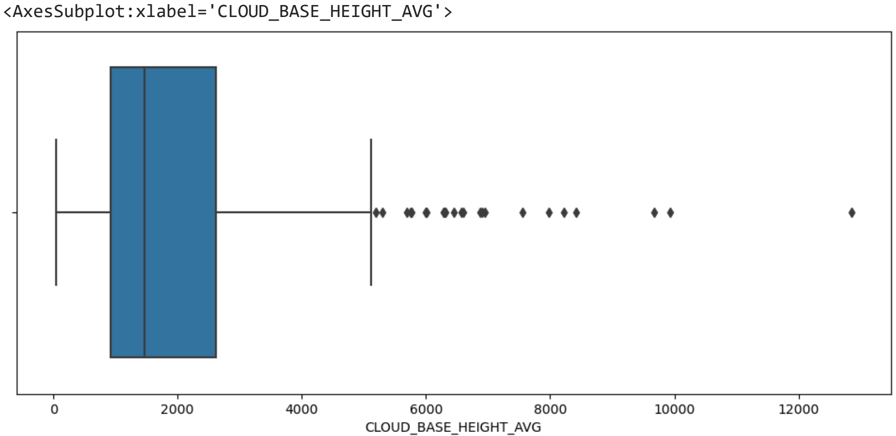
```
X.head()
```

	month_val	CLOUD_BASE_HEIGHT_AVG	CLOUD_COVER_AVG	DEGREE_DAYS_COOLING	DEGREE_DAYS_I
0	2	1552.0	0.30	0.0	
2	2	3385.0	0.53	0.0	
3	2	7547.0	0.05	0.0	
4	2	4407.0	0.46	0.0	
5	2	3108.0	0.55	0.0	

5 rows × 55 columns

```
#Tree-based models such as Decision Trees, Random Forests, and Gradient Boosted Trees can be robust against outliers to some extent
sns.boxplot(X['CLOUD_BASE_HEIGHT_AVG']) #SEE THE PRESENCE OUTLIERS.
```

```
C:\Users\ranit\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
  warnings.warn(
<AxesSubplot:xlabel='CLOUD_BASE_HEIGHT_AVG'>
```




```
#creating a lag variable since we saw that there is some autocorrelation in our time series
merged_df['lagged_estimated_visits']=merged_df['ESTIMATED_VISITS'].shift(1)
```

```
merged_df.head()
```

	DATE	ESTIMATED_VISITS	month_val	CLOUD_BASE_HEIGHT_AVG	CLOUD_COVER_AVG	DEGREE_DA
0	2021-04-01	3781	2	1552.0	0.30	
2	2021-04-03	3189	2	3385.0	0.53	
3	2021-04-04	5407	2	7547.0	0.05	
4	2021-04-05	2519	2	4407.0	0.46	
5	2021-04-06	2515	2	3108.0	0.55	

5 rows × 58 columns

```
#fill empty rows with means.We cannot leave time series values empty or we will need to resample.
merged_df['lagged_estimated_visits'].fillna(merged_df['lagged_estimated_visits'].mean(), inplace=True)
```

	month_val	CLOUD_BASE_HEIGHT_AVG	CLOUD_COVER_AVG	DEGREE_DAYS_COOLING	DEGREE_DAYS_I
0	2	1552.0	0.30	0.0	
2	2	3385.0	0.53	0.0	
3	2	7547.0	0.05	0.0	
4	2	4407.0	0.46	0.0	
5	2	3108.0	0.55	0.0	

5 rows × 55 columns

```
X.drop(['month'],axis=1,inplace=True)
```

```
#using decision tree for feature importances
clf = DecisionTreeClassifier()
merged_df=merged_df.dropna()

clf.fit(X, Y)
```

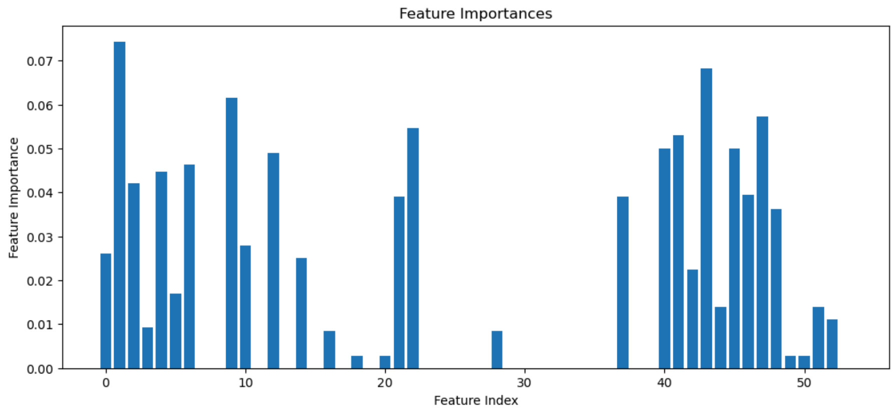
```
DecisionTreeClassifier()
```

```
#getting the top features based on importances
feature_importances = clf.feature_importances_
top_indices = np.argsort(feature_importances)[::-1][feature_importances > 0.02]

top_indices
```

```
array([ 1, 43,  9, 22, 45,  6,  4, 46, 37, 51, 44, 17, 38, 36, 35, 34, 31,
       19, 30, 29], dtype=int64)
```

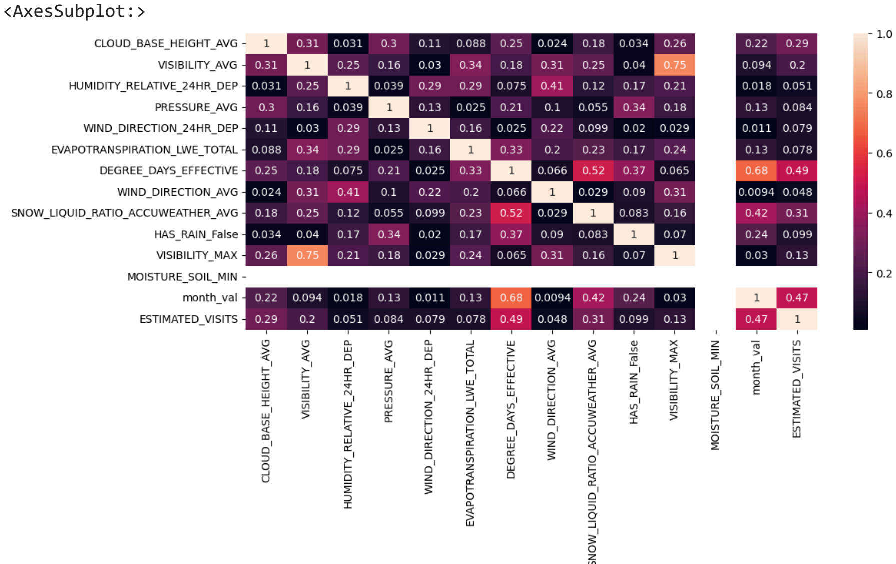
```
#all important columns.choosing just 12 of the features
important_cols=list(X.columns[top_indices])[:12]
```

```
plt.bar(range(len(feature_importances)), feature_importances)
plt.xlabel('Feature Index')
plt.ylabel('Feature Importance')
plt.title('Feature Importances')
plt.show()
```




```
#WE SEE THAT MONTH VAL WHICH SHOWS THE SEASON HAS A BIG CORRELATION WITH ESTIMATED VISITS
important_cols.append('month_val')
important_cols.append('ESTIMATED_VISITS')
sns.heatmap(merged_df[important_cols].corr().abs(),annot=True)
```

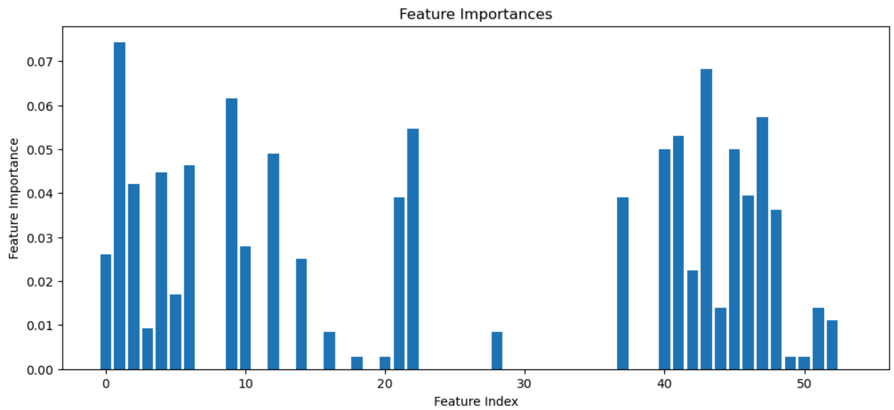
```
<AxesSubplot:>
```




```
removal=important_cols.pop()
removal
```

```
'ESTIMATED_VISITS'
```

```
plt.bar(range(len(feature_importances)), feature_importances)
plt.xlabel('Feature Index')
plt.ylabel('Feature Importance')
plt.title('Feature Importances')
plt.show()
```



MODELS USED BELOW ->DECISION TREE ,RANDOM FORESTS,XGBOOST,LGBM,XGBOOST WITH GRDISEARCHCV

Tree-based models, such as Random Forests and Gradient Boosting Machines (including XGBoost, LightGBM, and CatBoost), are advantageous for time series forecasting due to their ability to capture non-linear relationships, handle multicollinearity, provide feature importance, robustness to outliers and missing data, scalability, and interpretability. These models excel in capturing complex patterns inherent in time series data while remaining interpretable

```
X_a=merged_df[important_cols] #popping to remove estimated visits
Y_a=merged_df['ESTIMATED_VISITS']
```

```
len(Y_a)
```

```
359
```

```
X.head(10)
```

	month_val	CLOUD_BASE_HEIGHT_AVG	CLOUD_COVER_AVG	DEGREE_DAYS_COOLING	DEGREE_DAYS_
0	2	1552.0	0.30	0.0	
2	2	3385.0	0.53	0.0	
3	2	7547.0	0.05	0.0	
4	2	4407.0	0.46	0.0	
5	2	3108.0	0.55	0.0	
6	2	1476.0	0.58	0.0	
7	2	4506.0	0.21	0.0	
8	2	1147.0	0.64	0.0	
9	2	1782.0	0.80	0.0	
10	2	430.0	0.95	0.0	

10 rows × 54 columns

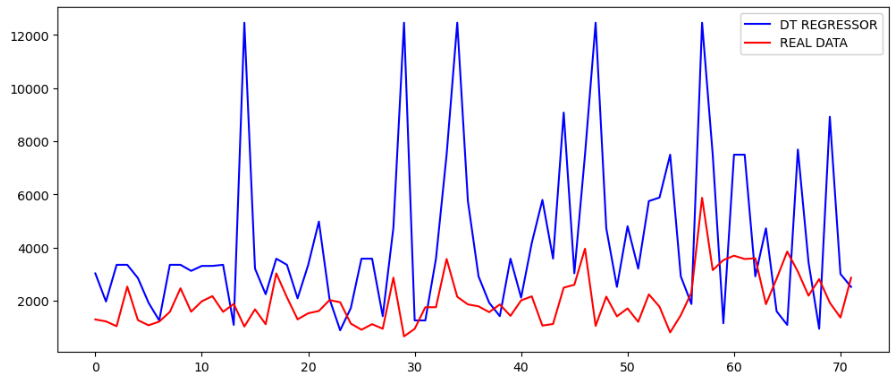
```
#creating a 80-20 split.Since this is a time series,we cannot have a random split but have a time frame instead.
half_80=int(len(X_a)*(0.8))
test_size=len(X_a)-(half_80)
X_train=X_a.iloc[:half_80]
X_test=X_a.iloc[half_80:]
Y_test=Y_a[X_test.index]
Y_train=Y_a[X_train.index]
X_test
```

	CLOUD_BASE_HEIGHT_AVG	VISIBILITY_AVG	HUMIDITY_RELATIVE_24HR_DEP	PRESSURE_AVG	W:
292	732.0	14.766	4.41	99918.48	
293	691.0	15.185	-2.77	101999.23	
294	289.0	14.680	3.92	102725.30	
295	527.0	14.654	-2.25	101285.77	
296	959.0	15.141	-1.03	100206.95	
...	...	...	...	...	
359	1040.0	15.751	-24.07	100004.41	
360	884.0	16.000	-3.75	100911.11	
361	1055.0	16.000	2.10	101479.34	
362	3604.0	15.338	18.00	100405.49	
363	949.0	15.378	4.96	98420.52	

72 rows × 13 columns

```
#DECISION TREE CLASSIFIER.OUR BASELINE ESTIMATOR
clf2 = DecisionTreeClassifier()
clf2.fit(X_train, Y_train)
y_pred_clf = clf2.predict(X_test)
plt.plot(y_pred_clf,label='DT REGRESSOR',color='blue')
plt.plot(Y_test.values,label='REAL DATA',color='red')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x1f221a754c0>
```

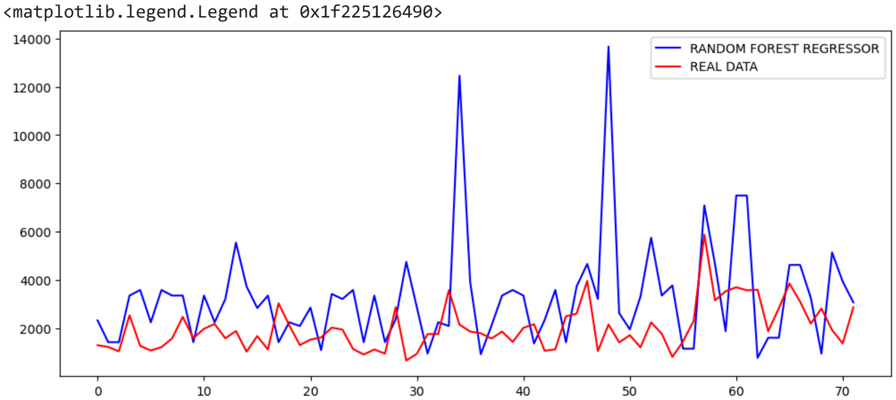



```
mae_ = mae(Y_test, y_pred_clf) #BASELINE MEAN AVERAGE ERROR OF 2455.04
mae_
```

```
2590.1111111111113
```

```
clf_rf = RandomForestClassifier() #RANDOM FOREST CLASSIFIER
clf_rf.fit(X_train, Y_train)
y_pred_rf = clf_rf.predict(X_test)
plt.plot(y_pred_rf,label='RANDOM FOREST REGRESSOR',color='blue')
plt.plot(Y_test.values,label='REAL DATA',color='red')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x1f225126490>
```




```
mae_4 = mae(Y_test, y_pred_rf) #RANDOM FOREST CLASSIFIER MASSIVE IMPROVEMENT ON DECISION TREE MAE.
mae_4
```

```
1756.1527777777778
```

```
#LIGHT GRADIENT BOOSTING->LightGBM works by using a histogram-based approach to bin continuous features,
#efficiently splitting nodes in trees based on the gradient of the loss function.
params = {'n_estimators': 400,'max_depth': 8}

model_lgb = lgb.LGBMRegressor()
model_lgb = model_lgb.fit(X_train, Y_train)
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000077 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 815
[LightGBM] [Info] Number of data points in the train set: 287, number of used features: 12
[LightGBM] [Info] Start training from score 3982.930314
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```