CSCI - 6409 - The Process of Data Science - Summer 2022

Project

Name	Banner Number
Guturu Rama Mohan Vishnu	B00871849
Aditya Mahale	B00867619
Smriti Mishra	B00904799
Shiva Shankar Pandillapalli	B00880049
Benny Daniel	B00899629

In [1]:
```python
# Library imports
import pandas as pd
import warnings
import numpy as np
from random import randint

# Visualization libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Geographical visualization library
import folium
from folium.plugins import HeatMap
from folium.plugins import HeatMapWithTime
from folium import plugins

from sklearn import set_config
from sklearn.feature_selection import SelectKBest
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score,mean_squared_error

import torch
import torch.nn as nn
```

In [2]:
```
# Prerequisites
# Ignoring Warnings
warnings.filterwarnings('ignore')
set_config(print_changed_only=False)
# Set figure size to 20 X 10
plt.figure(figsize=(20, 10))

# Set SNS plots figure size to 15 X 15
sns.set(rc={'figure.figsize': (15, 15)})

# Set options to avoid truncation when displaying a dataframe
pd.set_option("display.max_rows", None)
pd.set_option("display.max_columns", None)

# Set floating point numbers to be displayed with 2 decimal places
pd.set_option('display.float_format', '{:.5f}'.format)

# Setting plot font parameters
font_label = {'family': 'serif','color':  'darkred','weight': 'normal','s
ize': 16,}
font_title = {'family': 'serif','color':  'darkred','weight': 'bold','siz
e': 22,}
```

```
<Figure size 1440x720 with 0 Axes>
```

1. Data Exploration and preprocessing

Reference: CSCI 6409: Assignment-1, Aditya Mahale, Harshit Lakhani (dal.brightspace.com)

In [3]:
```
df = pd.read_csv("Tornadoes_SPC_1950to2015.csv")
```

1.1 Data Quality Report

1.1.a. Generate data quality reports for the continuous and the categorical features of the data set

In [4]:
```
# Summarizing the data types and the number of records available in each column
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60114 entries, 0 to 60113
Data columns (total 22 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   om      60114 non-null  int64  
 1   yr      60114 non-null  int64  
 2   mo      60114 non-null  int64  
 3   dy      60114 non-null  int64  
 4   date    60114 non-null  object 
 5   time    60114 non-null  object 
 6   tz      60114 non-null  int64  
 7   st      60114 non-null  object 
 8   stf     60114 non-null  int64  
 9   stn     60114 non-null  int64  
 10  mag     60114 non-null  int64  
 11  inj     60114 non-null  int64  
 12  fat     60114 non-null  int64  
 13  loss    60114 non-null  float64
 14  closs   60114 non-null  float64
 15  slat    60114 non-null  float64
 16  slon    60114 non-null  float64
 17  elat    60114 non-null  float64
 18  elon    60114 non-null  float64
 19  len     60114 non-null  float64
 20  wid     60114 non-null  int64  
 21  fc      60114 non-null  int64  
dtypes: float64(7), int64(12), object(3)
memory usage: 10.1+ MB
```

- There are 15 columns and none of them have any missing values.
- Two features are of type object. It cannot be used in preprocessing data and model training. So, we will convert those types to the relevant data types using the in built convert_dtypes method.

In [5]:
```
df = df.convert_dtypes()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60114 entries, 0 to 60113
Data columns (total 22 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   om      60114 non-null  Int64  
 1   yr      60114 non-null  Int64  
 2   mo      60114 non-null  Int64  
 3   dy      60114 non-null  Int64  
 4   date    60114 non-null  string 
 5   time    60114 non-null  string 
 6   tz      60114 non-null  Int64  
 7   st      60114 non-null  string 
 8   stf     60114 non-null  Int64  
 9   stn     60114 non-null  Int64  
 10  mag     60114 non-null  Int64  
 11  inj     60114 non-null  Int64  
 12  fat     60114 non-null  Int64  
 13  loss    60114 non-null  Float64
 14  closs   60114 non-null  Float64
 15  slat    60114 non-null  Float64
 16  slon    60114 non-null  Float64
 17  elat    60114 non-null  Float64
 18  elon    60114 non-null  Float64
 19  len     60114 non-null  Float64
 20  wid     60114 non-null  Int64  
 21  fc      60114 non-null  Int64  
dtypes: Float64(7), Int64(12), string(3)
memory usage: 11.2 MB
```

In [6]:
```
# Displaying the first row of the data to better understand the data inst
ance
df.loc[0]
```

Out[6]:
```
om                1
yr             1950
mo                1
dy                3
date       1/3/1950
time       11:00:00
tz                3
st               MO
stf              29
stn               1
mag               3
inj               3
fat               0
loss        6.00000
closs       0.00000
slat       38.77000
slon      -90.22000
elat       38.83000
elon      -90.03000
len         9.50000
wid             150
fc                0
Name: 0, dtype: object
```

In [7]:
```
# Custom function to create the continuous feature report
def build_continuous_features_report(data_df):

    """Build tabular report for continuous features"""

    stats = {
        "Count": len,
        "Miss %": lambda df: df.isna().sum() / len(df) * 100,
        "Card.": lambda df: df.nunique(),
        "Min": lambda df: df.min(),
        "1st Qrt.": lambda df: df.quantile(0.25),
        "Mean": lambda df: df.mean(),
        "Median": lambda df: df.median(),
        "3rd Qrt": lambda df: df.quantile(0.75),
        "Max": lambda df: df.max(),
        "Std. Dev.": lambda df: df.std(),
    }

    contin_feat_names = data_df.select_dtypes("number").columns
    continuous_data_df = data_df[contin_feat_names]

    report_df = pd.DataFrame(index=contin_feat_names, columns=stats.keys
())

    for stat_name, fn in stats.items():
        # NOTE: ignore warnings for empty features
        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=RuntimeWarning)
            report_df[stat_name] = fn(continuous_data_df)

    return report_df
```

In [8]:
```
# Preliminary statistics for the continuous features
build_continuous_features_report(df)
```

Out[8]:

	Count	Miss %	Card.	Min	1st Qrt.	Mean	Median	3rd Qrt
om	60114	0.00000	7422	1.00000	248.00000	41119.37575	509.00000	845.00000
yr	60114	0.00000	66	1950.00000	1974.00000	1987.97006	1991.00000	2003.00000
mo	60114	0.00000	12	1.00000	4.00000	5.97244	6.00000	7.00000
dy	60114	0.00000	31	1.00000	8.00000	15.87637	16.00000	24.00000
tz	60114	0.00000	4	0.00000	3.00000	3.00110	3.00000	3.00000
stf	60114	0.00000	52	1.00000	18.00000	29.41992	29.00000	45.00000
stn	60114	0.00000	233	0.00000	4.00000	26.47623	15.00000	35.00000
mag	60114	0.00000	6	0.00000	0.00000	0.79615	1.00000	1.00000
inj	60114	0.00000	206	0.00000	0.00000	1.56130	0.00000	0.00000
fat	60114	0.00000	48	0.00000	0.00000	0.09687	0.00000	0.00000
loss	60114	0.00000	472	0.00000	0.00000	2.15931	0.10000	4.00000
closs	60114	0.00000	47	0.00000	0.00000	0.00213	0.00000	0.00000
slat	60114	0.00000	2319	18.13000	33.24000	37.15521	37.08500	40.97000
slon	60114	0.00000	4234	-163.53000	-98.60000	-92.96113	-93.95000	-86.87000
elat	60114	0.00000	2291	0.00000	0.00000	20.95625	31.20000	38.15000
elon	60114	0.00000	3850	-163.53000	-94.25000	-51.90289	-81.76500	0.00000
len	60114	0.00000	2132	0.00000	0.10000	3.48072	0.60000	3.00000
wid	60114	0.00000	323	0.00000	13.00000	98.45460	40.00000	100.00000
fc	60114	0.00000	2	0.00000	0.00000	0.03099	0.00000	0.00000

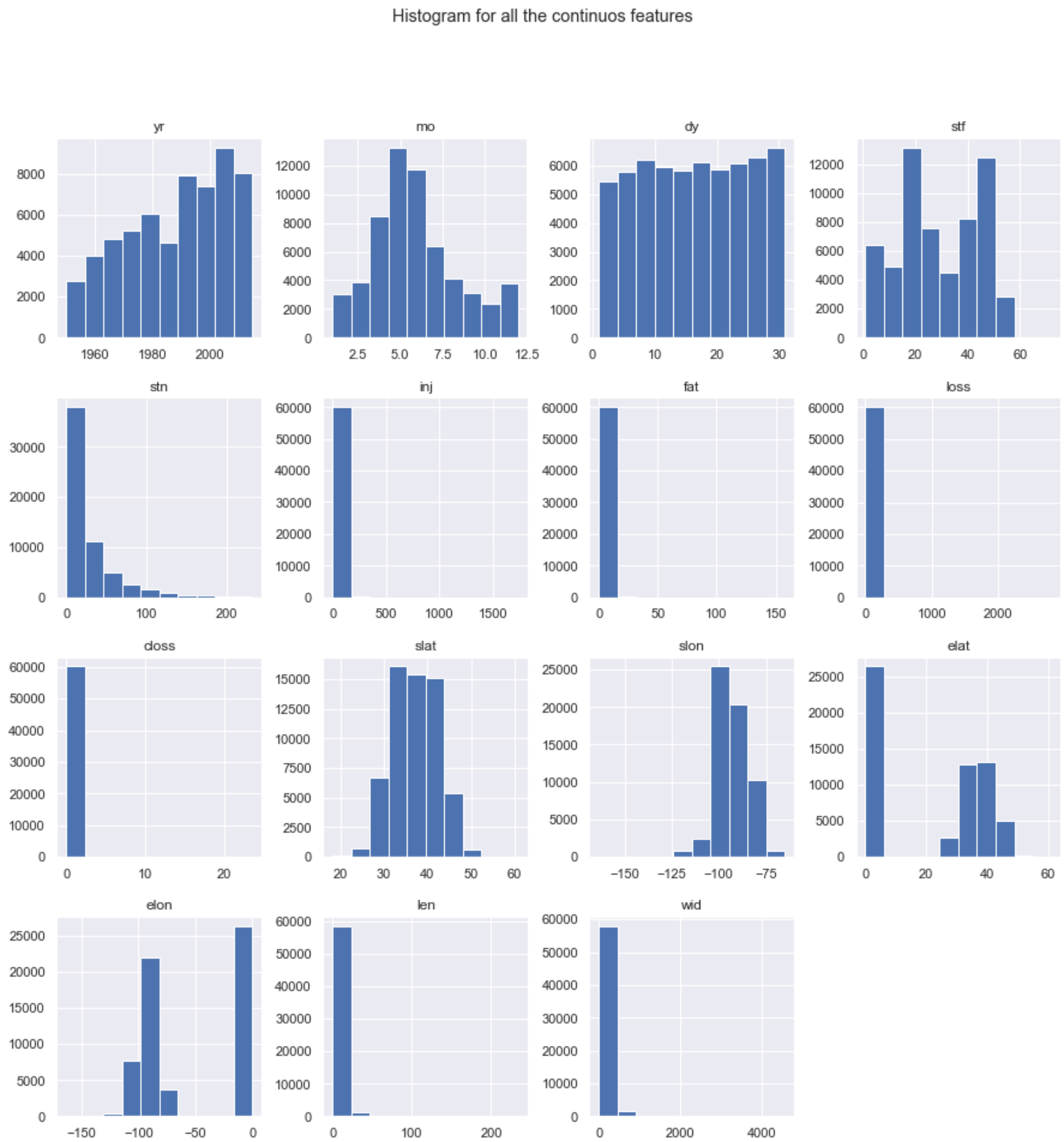
Initial Observations

- We can see that in the numerical data columns, there are no missing values
- We can say that the columns tz, mag,and fc are categorical features as because the cardinality is less than 10

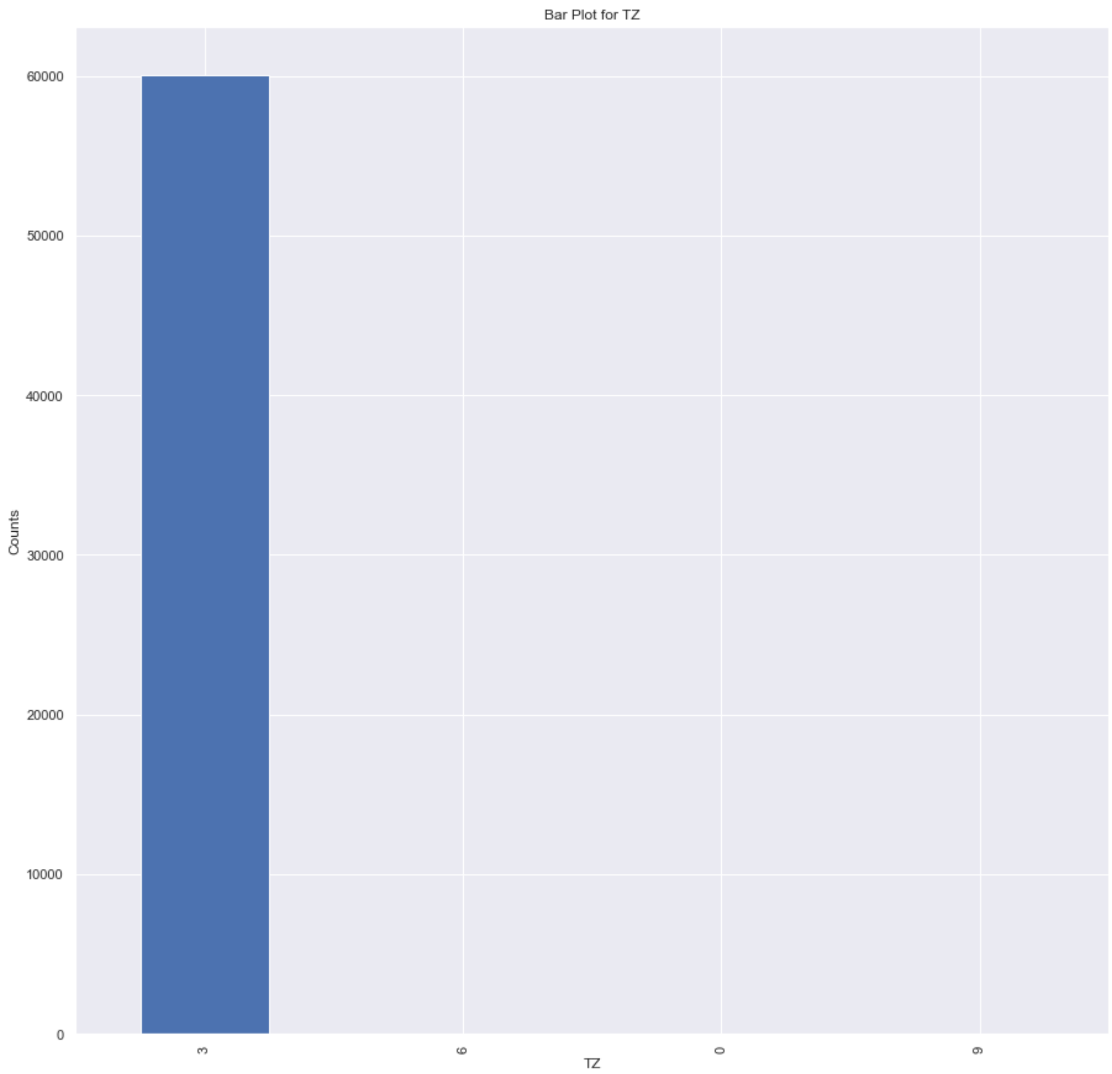
In [9]:
```
# Custom function to create the categorical feature report
def build_categorical_features_report(data_df):

    """Build tabular report for categorical features"""

    def _mode(df):
        return df.apply(lambda ft: ft.mode().to_list()).T

    def _mode_freq(df):
        return df.apply(lambda ft: ft.value_counts()[ft.mode()].sum())

    def _second_mode(df):
        return df.apply(lambda ft: ft[~ft.isin(ft.mode())].mode().to_list
())

    def _second_mode_freq(df):
        return df.apply(
            lambda ft: ft[~ft.isin(ft.mode())]
            .value_counts()[ft[~ft.isin(ft.mode())].mode()]
            .sum()
        )

    stats = {
        "Count": len,
        "Miss %": lambda df: df.isna().sum() / len(df) * 100,
        "Card.": lambda df: df.nunique(),
        "Mode": _mode,
        "Mode Freq": _mode_freq,
        "Mode %": lambda df: _mode_freq(df) / len(df) * 100,
        "2nd Mode": _second_mode,
        "2nd Mode Freq": _second_mode_freq,
        "2nd Mode %": lambda df: _second_mode_freq(df) / len(df) * 100,
    }

    cat_feat_names = data_df.select_dtypes(exclude="number").columns
    continuous_data_df = data_df[cat_feat_names]

    report_df = pd.DataFrame(index=cat_feat_names, columns=stats.keys())

    for stat_name, fn in stats.items():
        # NOTE: ignore warnings for empty features
        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=RuntimeWarning)
            report_df[stat_name] = fn(continuous_data_df)

    return report_df
```

In [10]:

```
# Visualization

# Histograms of continuous features
# Choosing only numerical columns and columns where cardinality is greater than or equal to ten

df.hist(column=['yr', 'mo', 'dy', 'stf', 'stn', 'inj', 'fat', 'loss', 'closs', 'slat', 'slon', 'elat', 'elon', 'len', 'wid']);
plt.suptitle("Histogram for all the continuos features")
plt.show()
```

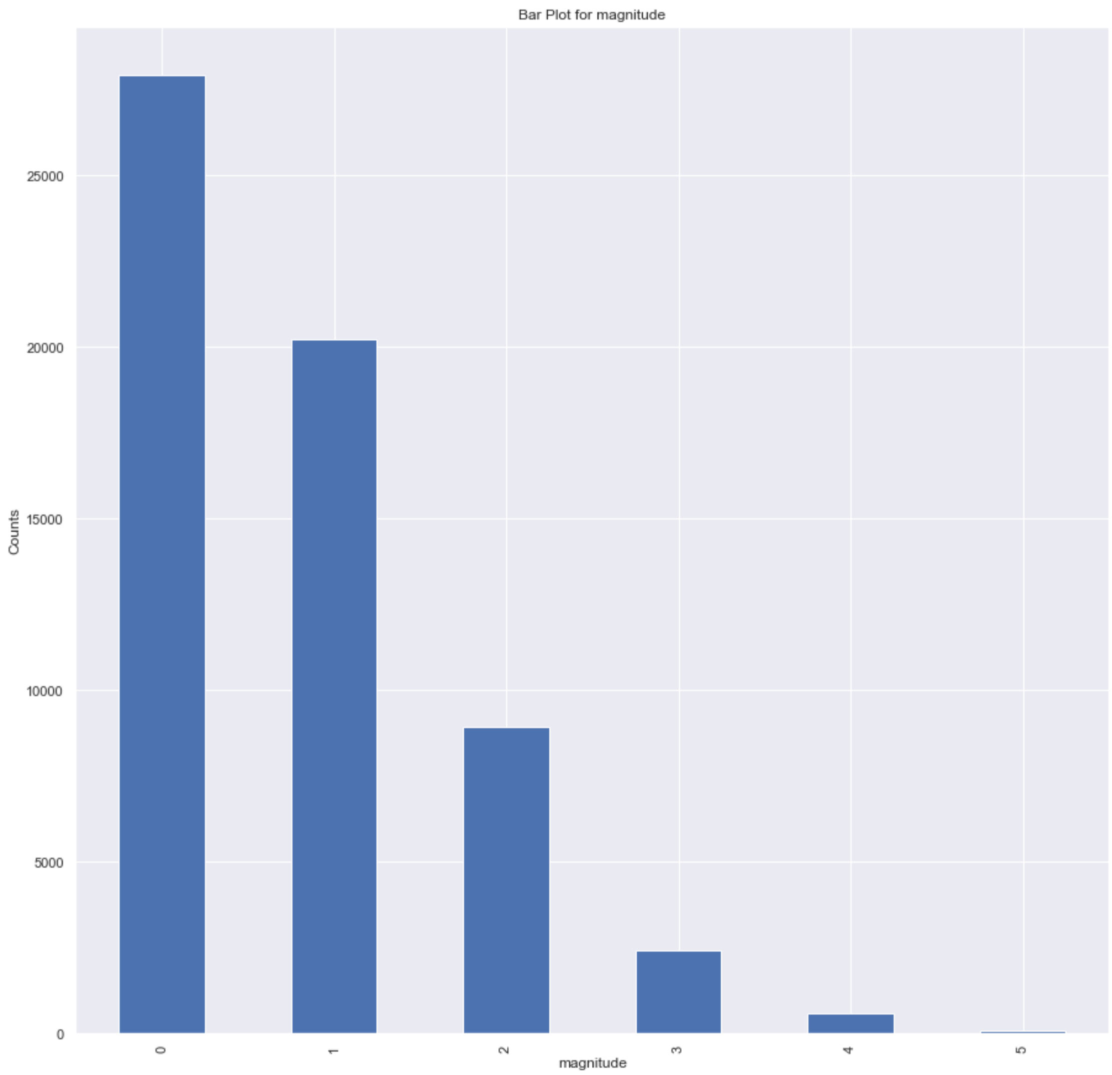


In [11]:
```python
# Categorical Features Visualizations

# Time Zone feature bar plot
df['tz'].value_counts().plot.bar();
plt.xlabel('TZ')
plt.ylabel('Counts')
plt.title('Bar Plot for TZ')
plt.show()
```

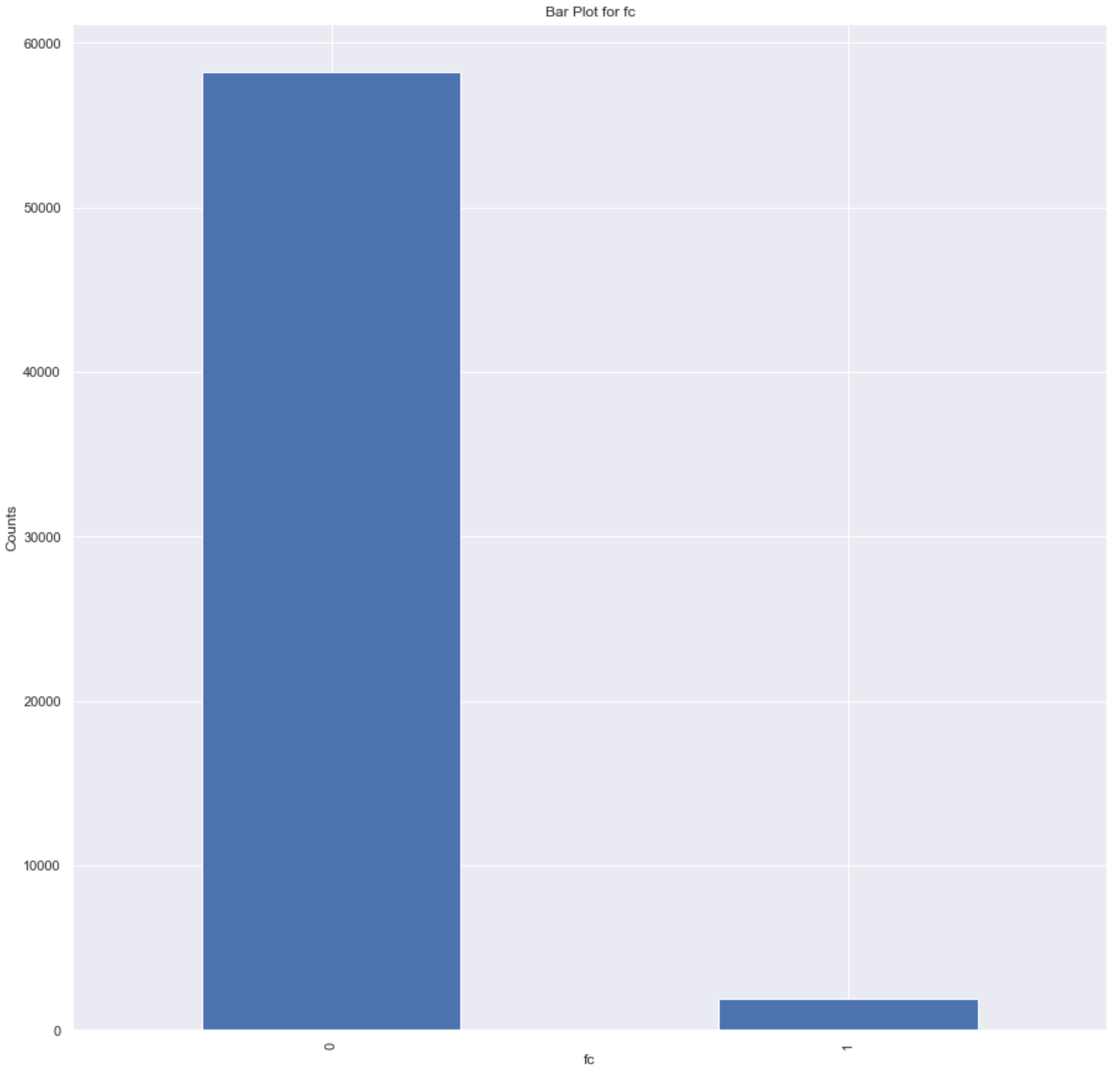


In [12]:

```
# Categorical Features Visualizations

# Magnitude feature bar plot
df['mag'].value_counts().plot.bar();
plt.xlabel('magnitude')
plt.ylabel('Counts')
plt.title('Bar Plot for magnitude')
plt.show()
```

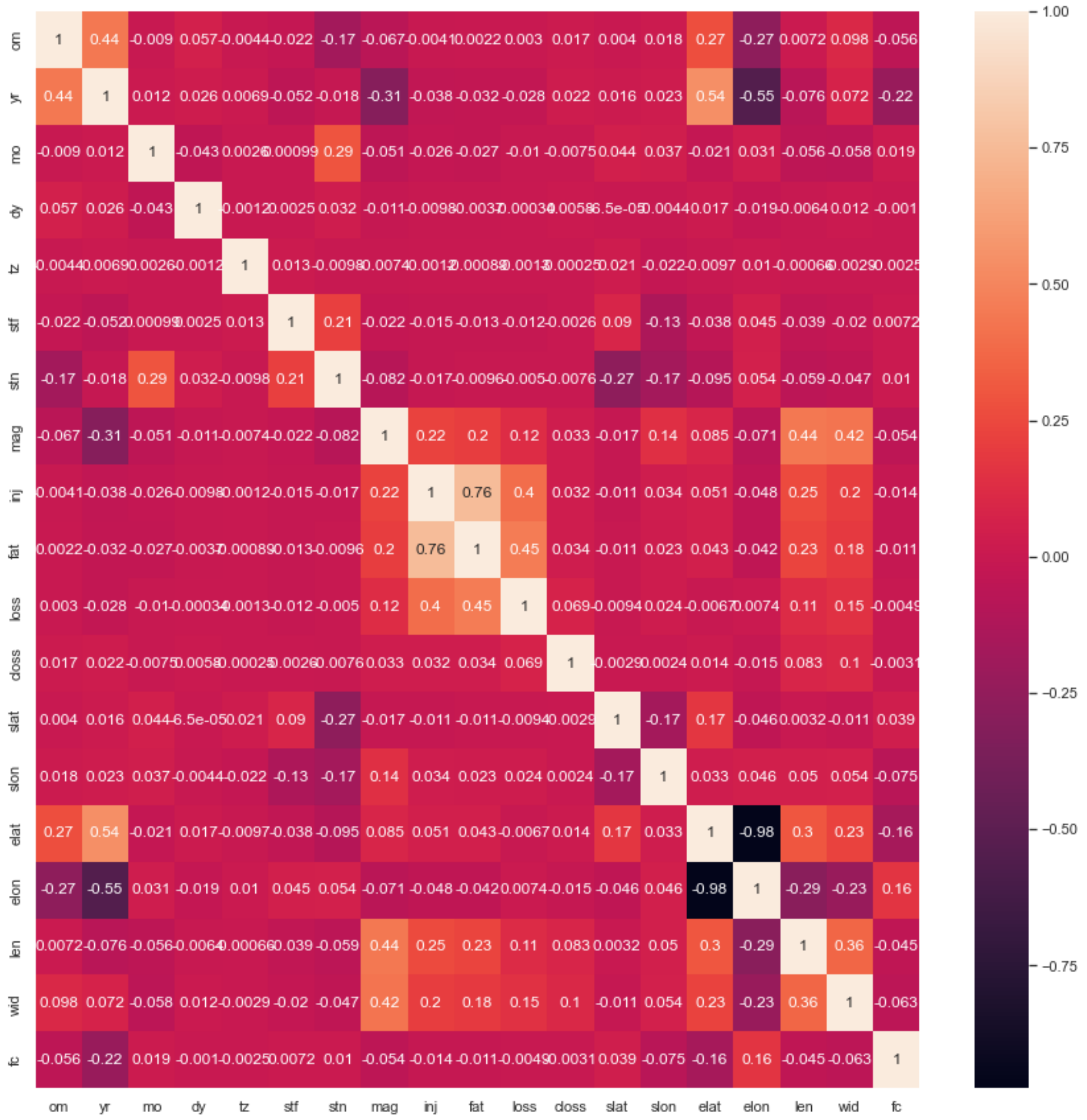


In [13]:
```
# Categorical Features Visualizations

# Magnitude feature bar plot
df['fc'].value_counts().plot.bar();
plt.xlabel('fc')
plt.ylabel('Counts')
plt.title('Bar Plot for fc')
plt.show()
```



Heatmap Correlation

```python
In [14]: sns.heatmap(df.corr(), annot=True);
```



2. Preprocessing data

Dropping columns that do not add any information to the model training

The "om" column just denotes tornado number. It does not add anything to the data

```python
In [15]: df.drop(['om'], axis = 1, inplace = True)
```

The "date" column is redundant since there are separate columns for year, month and day.

```python
In [16]: df.drop(['date'], axis = 1, inplace = True)
```

The timezone(tz) column does not add any new information to the dataset since location information is already represented by other columns

In [17]:
```
df.drop(['tz'], axis = 1, inplace = True)
```

The "stf" and "stn" columns are redundant since the "st" column already represents that data

In [18]:
```
df.drop(['stf', 'stn'], axis = 1, inplace = True)
```

In [19]:
```
df.drop(['fc'], axis = 1, inplace = True)
```

In [20]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60114 entries, 0 to 60113
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   yr      60114 non-null  Int64  
 1   mo      60114 non-null  Int64  
 2   dy      60114 non-null  Int64  
 3   time    60114 non-null  string 
 4   st      60114 non-null  string 
 5   mag     60114 non-null  Int64  
 6   inj     60114 non-null  Int64  
 7   fat     60114 non-null  Int64  
 8   loss    60114 non-null  Float64
 9   closs   60114 non-null  Float64
 10  slat    60114 non-null  Float64
 11  slon    60114 non-null  Float64
 12  elat    60114 non-null  Float64
 13  elon    60114 non-null  Float64
 14  len     60114 non-null  Float64
 15  wid     60114 non-null  Int64  
dtypes: Float64(7), Int64(7), string(2)
memory usage: 8.1 MB
```

Replacing zero values in the elat and elon columns with the starting values

In [21]:
```
df.head()
```

Out[21]:

	yr	mo	dy	time	st	mag	inj	fat	loss	closs	slat	slon	
0	1950	1	3	11:00:00	MO	3	3	0	6.00000	0.00000	38.77000	-90.22000	38.8
1	1950	1	3	11:55:00	IL	3	3	0	5.00000	0.00000	39.10000	-89.30000	39.1
2	1950	1	3	16:00:00	OH	1	1	0	4.00000	0.00000	40.88000	-84.58000	0.0
3	1950	1	13	5:25:00	AR	3	1	1	3.00000	0.00000	34.40000	-94.37000	0.0
4	1950	1	25	19:30:00	MO	2	5	0	5.00000	0.00000	37.60000	-90.68000	37.6

In [22]:
```python
df.loc[df['elat'] == 0, 'elat'] = df['slat']
df.loc[df['elon'] == 0, 'elon'] = df['slon']
```

In [23]:
```python
df.head()
```

Out[23]:

	yr	mo	dy	time	st	mag	inj	fat	loss	closs	slat	slon	
0	1950	1	3	11:00:00	MO	3	3	0	6.00000	0.00000	38.77000	-90.22000	38.8
1	1950	1	3	11:55:00	IL	3	3	0	5.00000	0.00000	39.10000	-89.30000	39.1
2	1950	1	3	16:00:00	OH	1	1	0	4.00000	0.00000	40.88000	-84.58000	40.8
3	1950	1	13	5:25:00	AR	3	1	1	3.00000	0.00000	34.40000	-94.37000	34.4
4	1950	1	25	19:30:00	MO	2	5	0	5.00000	0.00000	37.60000	-90.68000	37.6

In [24]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60114 entries, 0 to 60113
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   yr      60114 non-null  Int64
 1   mo      60114 non-null  Int64
 2   dy      60114 non-null  Int64
 3   time    60114 non-null  string
 4   st      60114 non-null  string
 5   mag     60114 non-null  Int64
 6   inj     60114 non-null  Int64
 7   fat     60114 non-null  Int64
 8   loss    60114 non-null  Float64
 9   closs   60114 non-null  Float64
 10  slat    60114 non-null  Float64
 11  slon    60114 non-null  Float64
 12  elat    60114 non-null  Float64
 13  elon    60114 non-null  Float64
 14  len     60114 non-null  Float64
 15  wid     60114 non-null  Int64
dtypes: Float64(7), Int64(7), string(2)
memory usage: 8.1 MB
```

3. Building the Model to predict the features like starting latitude and longitude, ending latitude and longitude of a tornado

After going through the lectures, the prescribed text book and an article [7] mentioned in the reference, We decided to use three main evaluation metrics for the regression model we have planned to build.

1. Mean Squared Error
2. Root Mean Squared Error
3. Mean Absolute Error
We discuss in brief below about each of the metric mentioned above.

1. Mean Squared Error

The mean or average of the squared differences between predicted and expected target values in a dataset is used to calculate the MSE.

Basically, the lower the MSE the better is the prediction.

```
MSE = 1/n∑(actual-predicted)^2
```

The units of the MSE are squared units.

Mean Squared Error (MSE) is a popular error metric for regression problems for several reasons, one of which is the squaring concept, which has the effect of inflating large errors and thus has the effect of “punishing” models more for larger errors.

2. Root Mean Squared Error

The root mean square error is the residuals' standard deviation, where the residuals are the measure of how far the data points are from the regression line . In other words, it indicates how concentrated the data is around the line of the best fit.

```
RMSE = √ 1/n∑(actual-predicted)^2 = √ MSE
```

The units of the RMSE are the same as the original units of the target value.

This is another popular error metric for regression as generally regression prediction models are frequently trained using MSE loss, and their performance is assessed and reported using RMSE.

3. Mean Absolute Error

The mean absolute error is the average difference between the observations (true values) and model output (predictions).

The changes in MAE, in contrast to the RMSE, are linear i.e the MAE does not give distinct sorts of errors more or less weight; instead, the scores rise linearly as the amount of error increases.

```
MAE = 1/n∑[abs(actual-predicted)]
```

We decided to use these three regression measures indicated above for the assignment due to its aforementioned behaviour.

Starting latitude and Statring Longitude

In this section, we will try to predict the starting and ending latitude and longitudes of the tornado.

Although this purely depends on weather conditions, In this section we tried to predict the tornado location based on the other parameters available in this dataset.

The preciseness of the location could be enhanced by adding a few other weather related features.

First we would like to see where the tornadoes are most frequently seen in the states and then we would like to start working from there.

In [25]:
```
df['ones'] = np.ones(len(df))
df.groupby(['st'])['ones'].sum().sort_values(ascending=False)
```

Out[25]:
```
st
TX    8484.00000
KS    4027.00000
OK    3658.00000
FL    3233.00000
NE    2758.00000
IA    2404.00000
IL    2349.00000
MO    2154.00000
CO    2071.00000
MS    2034.00000
AL    1979.00000
LA    1858.00000
SD    1745.00000
AR    1715.00000
MN    1708.00000
GA    1483.00000
ND    1483.00000
IN    1391.00000
WI    1309.00000
NC    1239.00000
TN    1145.00000
OH    1014.00000
MI    1004.00000
SC     942.00000
KY     900.00000
PA     752.00000
VA     675.00000
WY     651.00000
NM     561.00000
CA     423.00000
NY     422.00000
MT     406.00000
MD     346.00000
AZ     241.00000
ID     206.00000
MA     158.00000
NJ     142.00000
WV     128.00000
ME     124.00000
UT     123.00000
WA     112.00000
OR     105.00000
CT      94.00000
NH      88.00000
NV      86.00000
DE      60.00000
VT      44.00000
HI      41.00000
PR      24.00000
RI      10.00000
AK       4.00000
DC       1.00000
Name: ones, dtype: float64
```

As we can see from the above list, we see that texas has got the highest number of tornadoes seen in the entire dataset which is 8484.

so we will be trying to predict the tornado position as that data is quite rich.

Once we are successfull in doing so, we could run the same model for the entire dataset.

The following code creates a new dataframe separately for Texas state.

In [26]:
```
txdf = df[df['st']=='TX']
txdf = txdf.drop('ones',axis=1)
```

In [27]:
```
txdf.groupby('st')['slat'].nunique()
```

Out[27]:
```
st
TX    843
Name: slat, dtype: int64
```

The following is the head of the dataframe which shows all the columns and the data inside them giving us a basic understanding where to start

In [28]:
```
txdf.head()
```

Out[28]:

	yr	mo	dy	time	st	mag	inj	fat	loss	closs	slat	slon	
6	1950	1	26	18:00:00	TX	2	2	0	0.00000	0.00000	26.88000	-98.12000	26.8
7	1950	2	11	13:10:00	TX	2	0	0	4.00000	0.00000	29.42000	-95.25000	29.5
8	1950	2	11	13:50:00	TX	3	12	1	4.00000	0.00000	29.67000	-95.05000	29.8
9	1950	2	11	21:00:00	TX	2	5	0	5.00000	0.00000	32.35000	-95.20000	32.4
10	1950	2	11	23:55:00	TX	2	6	0	5.00000	0.00000	32.98000	-94.63000	33.0

from the below code we can see the most common locations in the entire dataset of 8484 records.

In [29]:
```
df['slat'].nunique()
```

Out[29]:
```
2319
```

In [30]:
```
df['slon'].nunique()
```

Out[30]:
```
4234
```

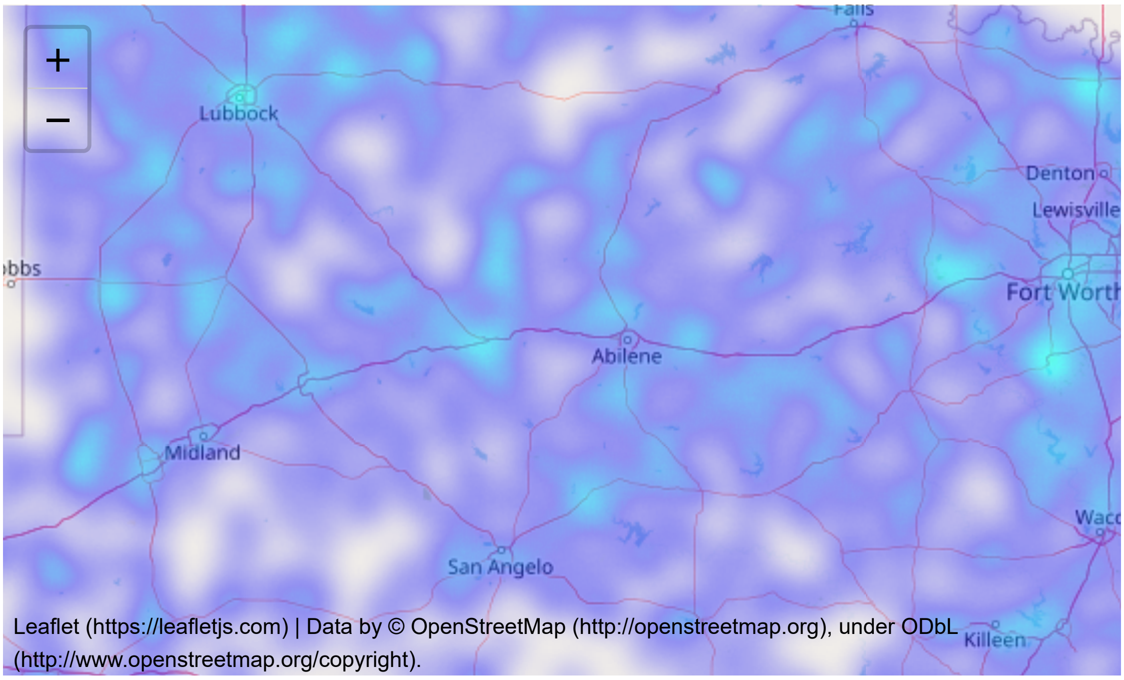
From the above, we can see that there are 2319 unique latititude locations and 4234 unique longitude locations of the start of a tornado

The following code visualizes the records to figure out the hot spots of the tornadoes in the map based on the geolocations we have with us in the dataset.

```
In [31]: latlonlist = []
         for i in range(len(txdf)):
             latlonlist.append([txdf['slat'].iloc[i],txdf['slon'].iloc[i]])
```

```
In [32]: map = folium.Map(location=[31.2412024,-97.7553315], zoom_start=7)
         HeatMap(latlonlist,min_opacity=0.3, radius=10).add_to(map)
         map
```

Out[32]:



From the above map, we can see that the tornadoes are most frequently seen in cities like Houston, Baumont, FortWorth, San Antonio, Corpus, Lubbock.

Since the numbers are huge, we will not be concentrating on one location.

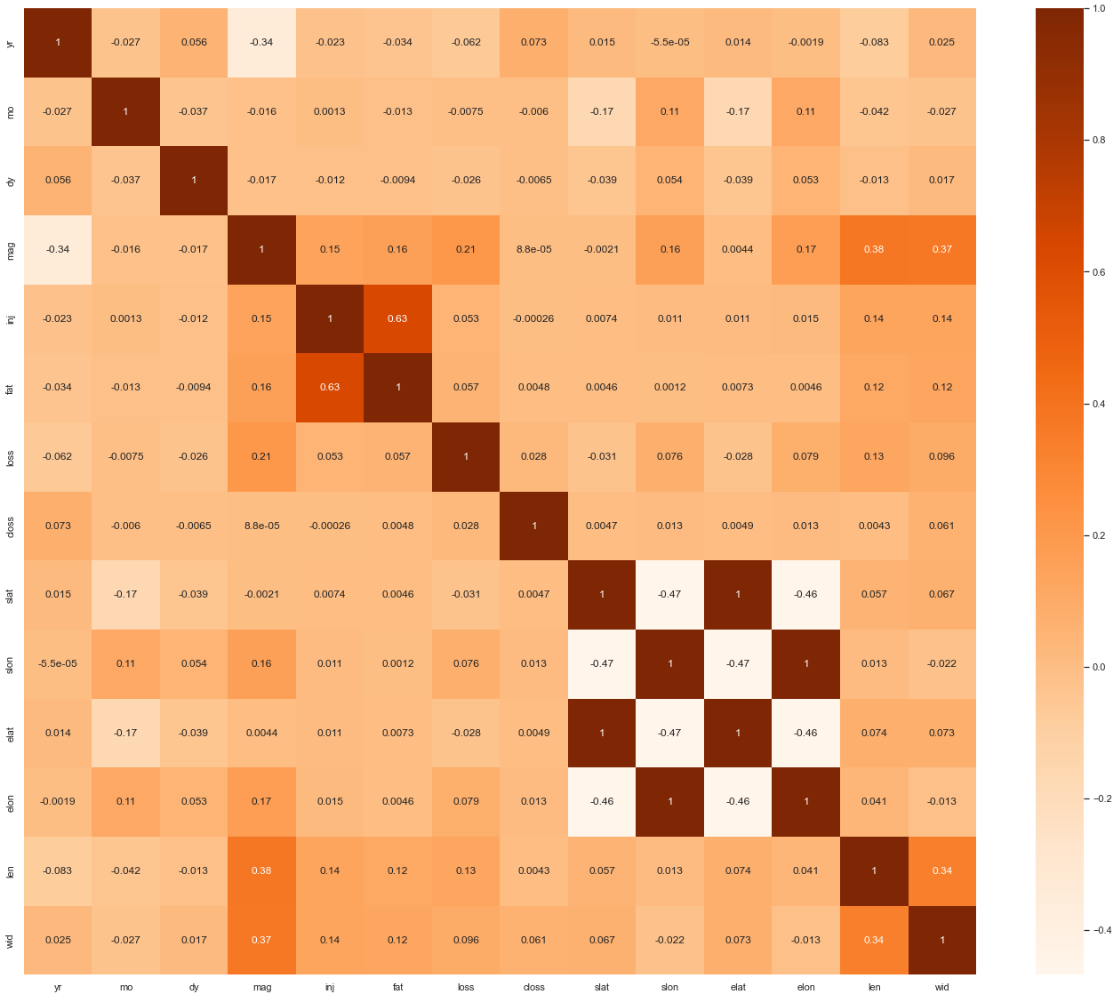
For now we will be working on the entire state's data to see how the predictions comes

```
In [33]: # map = folium.Map(location=[31.2412024,-97.7553315], zoom_start=7)
         # for x in latlonlist:
         #     folium.CircleMarker([x[0], x[1]], radius=0.5, color='orange', fill=
         True, fill_opacity=0.7, fill_color='orange').add_to(map)
         # # HeatMap(latlonlist,min_opacity=0.2, radius=10).add_to(map)
         # map
```

To figure out the best features, correlation graph helps us which is shown below.

```
In [34]: sns.set(rc={'figure.figsize':(24,20)})
         dataCorrelation = txdf.corr()
         sns.heatmap(dataCorrelation, cmap="Oranges", annot=True)
```

```
Out[34]: <AxesSubplot:>
```



From the above we can see there is no strong relationship of slat and slon features with any of the other features in the dataset.

Hence, we would like to try all the features in the dataset to figure out the slat and slon.

Although this is completely based on weather conditions, we are trying to predict the slat and slon features using ensembles to see if it would work.

We are doing this simply to see if ensembles could help us predict the features with the remaining data.

The following K-Best Features method is writtten to extract the best features as all of our features are continuous except the Date fields.

In [35]:
```python
def getKFeatures(data, kValue, target):
    predictors = list(data.describe(include=['number']).columns)
    predictors.remove('slat')
    predictors.remove('slon')
    predictors.remove('elat')
    predictors.remove('elon')
    try:
        predictors.remove(target)
    except:
        pass
    bestFeatureSelection = SelectKBest(k=kValue)
    # print(data[predictors].head(),"\n\n\n",data[target].head())
    # print(type(data[target]))
    _ = bestFeatureSelection.fit_transform(data[predictors],data[target])
    bools = bestFeatureSelection.get_support()
    selected = []
    for i in range(len(predictors)):
        if bools[i]==True:
            selected.append(predictors[i])
    return selected
```

In the above code we remove the slat, slon, elat and elon features as these are the features we would like to predict.

Now we would like to see the top features picked by the KBest features starting k-value from 1 till all the features.

In [36]:
```python
len(txdf.columns)
```

Out[36]: 16

In [37]:
```python
for i in range(1,len(txdf.columns)-5):
    print(getKFeatures(txdf, i, ['slat']))
```

```
['yr']
['yr', 'closs']
['yr', 'mo', 'closs']
['yr', 'mo', 'loss', 'closs']
['yr', 'mo', 'dy', 'loss', 'closs']
['yr', 'mo', 'dy', 'mag', 'loss', 'closs']
['yr', 'mo', 'dy', 'mag', 'loss', 'closs', 'wid']
['yr', 'mo', 'dy', 'mag', 'inj', 'loss', 'closs', 'wid']
['yr', 'mo', 'dy', 'mag', 'inj', 'loss', 'closs', 'len', 'wid']
['yr', 'mo', 'dy', 'mag', 'inj', 'fat', 'loss', 'closs', 'len', 'wid']
```

After doing some trial and erros on the dataset with all the above features, we have observed that when the K-value is 6, the features selected like Year, Month, Crop loss, Starting longitude, ending latitude, ending longitude were showing better results to predict the slat feature.

In [38]:
```python
selected = getKFeatures(txdf, 7, ['slat'])
print(selected)
```

```
['yr', 'mo', 'dy', 'mag', 'loss', 'closs', 'wid']
```

In [39]:
```
selected = getKFeatures(txdf, 6, ['slon'])
print(selected)
```

```
['yr', 'mo', 'dy', 'mag', 'loss', 'closs']
```

In [40]:
```
selected = getKFeatures(txdf, 8, ['slon'])
print(selected)
```

```
['yr', 'mo', 'dy', 'mag', 'loss', 'closs', 'len', 'wid']
```

In [41]:
```
class RegressionNeuralNet(nn.Module):
    def __init__(self, inputDimensions, outputDimensions):
        nn.Module.__init__(self)
        # self.hiddenLayers = nn.Sequential(nn.Linear(inputDimensions,64),nn.Linear(64,32),nn.Linear(32,8),nn.Linear(8,outputDimensions))
        # self.hiddenLayers = nn.Sequential(nn.Linear(inputDimensions,32),nn.Linear(32,8),nn.Linear(16,outputDimensions))
        # self.hiddenLayers = nn.Sequential(nn.Linear(inputDimensions,16),nn.Linear(16,outputDimensions))
        self.hiddenLayers = nn.Sequential(nn.Linear(inputDimensions,inputDimensions), nn.ReLU(), nn.Linear(inputDimensions,inputDimensions), nn.ReLU(), nn.Linear(inputDimensions,inputDimensions), nn.ReLU(), nn.Linear(inputDimensions,round(inputDimensions/2)), nn.Linear(round(inputDimensions/2),outputDimensions))


    def forward(self,trainX):
        # print(trainX.size(),trainX)
        return self.hiddenLayers(trainX)
```

In [42]:
```
current_device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
current_device.type
```

Out[42]:
```
'cuda'
```

In [43]:
```
predictors = txdf[selected].copy()
target = txdf[['slat','slon']].copy()
xTrain, xTest, yTrain, yTest = train_test_split(predictors, target, test_size=0.3, shuffle = True)
```

In [44]:
```
def prepareDataset(df,inputFeatures, outputFeatures):
    my_data = list()
    for i in range(len(df)):
        my_data.append((torch.FloatTensor(list(df[inputFeatures].iloc[i])),torch.FloatTensor(list(df[outputFeatures].iloc[i]))))
    return my_data
```

In [45]:
```
my_data = prepareDataset(txdf,selected,['slat','slon'])
```

In [46]:
```
def getTrainTestBatches(my_data, ssize, bsize):
    train_data, test_data = torch.utils.data.random_split(my_data, [round
(len(my_data)*(1-ssize)), round(len(my_data)*ssize)])

    train_batches = torch.utils.data.DataLoader(train_data, batch_size=bs
ize, shuffle=True) #shuffling data to get random samples instead of seque
nce
    # print(len(train_batches))

    test_batches = torch.utils.data.DataLoader(test_data, batch_size=bsiz
e, shuffle=True)
    return train_batches, test_batches
```

In [47]:
```
train_batches, test_batches = getTrainTestBatches(my_data, 0.7, 32)
```

In [48]:
```
print(len(test_batches)*32,len(train_batches)*32)
```

```
5952 2560
```

In [49]:
```
# simple plot to plot between iterations and the cost
def plotData(title, xlabel, ylabel,x, y, color, marker):
    # plt.scatter(x,loss,color='orange')
    plt.plot(x,y,color=color, marker=marker, linestyle='dashed',linewidth
=2, markersize=6)
    plt.scatter(x,y)
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.show()
```

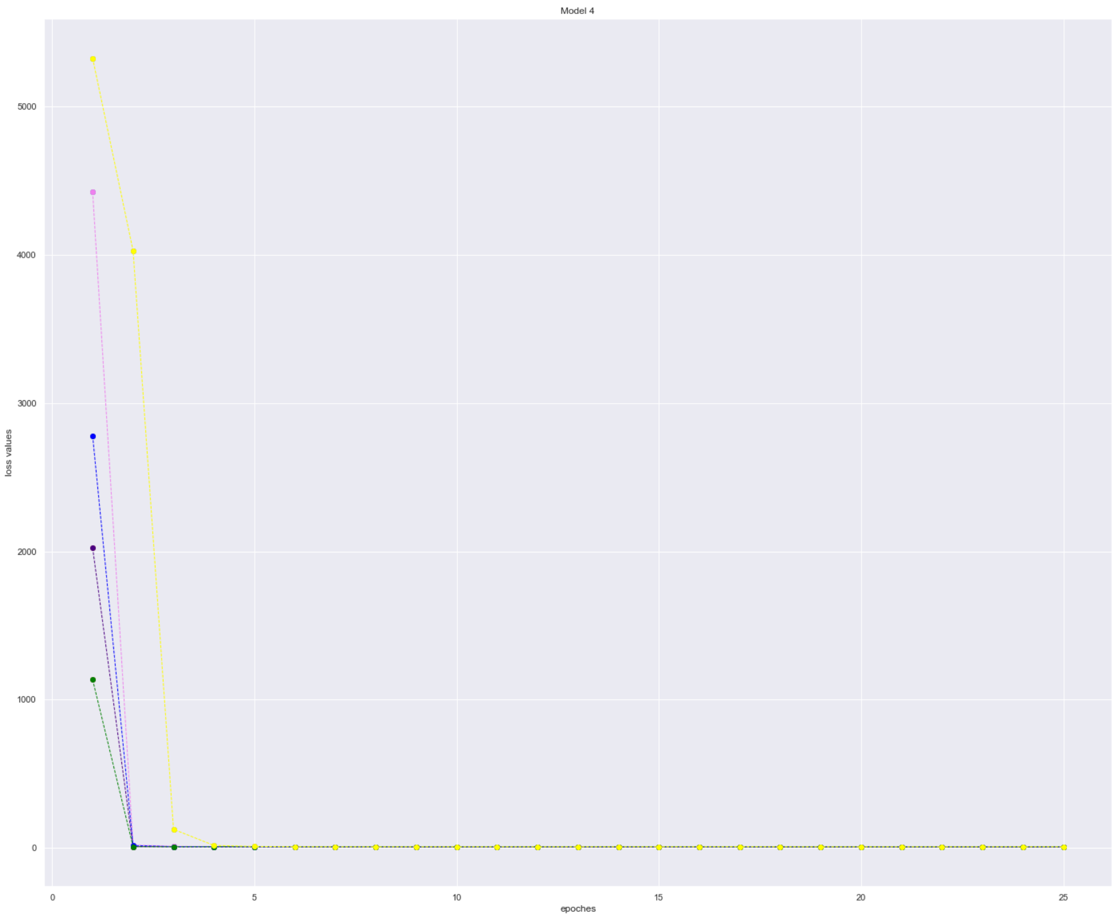
In [50]:

```
def trainTestMyModel(train_batches,epochs,lr, i):
    colors = ['violet','indigo','blue','green', 'yellow', 'orange', 'red
']
    lossValues = []
    gradScaler = torch.cuda.amp.GradScaler()
    lossFunction = nn.MSELoss()
    myModel = RegressionNeuralNet(len(selected),2).to(current_device)
    optimizer = torch.optim.SGD(myModel.parameters(), lr=lr, momentum=0.
9)
    # optimizer = torch.optim.Adam(myModel.parameters(), lr=0.001)
    for epoch in range(epochs):
        current_loss = []
        current_loss.clear()
        for itr,(x_true, y_true) in enumerate(train_batches):
            xTrue = (x_true.requires_grad_()).to(current_device)
            yTrue = (y_true).to(current_device)
            optimizer.zero_grad()
            yPred = myModel(xTrue)
            loss = lossFunction(yPred,yTrue)
            gradScaler.scale(loss).backward()
            gradScaler.unscale_(optimizer)
            torch.nn.utils.clip_grad_norm_(myModel.parameters(), max_norm
=1.0)
            gradScaler.step(optimizer)
            gradScaler.update()
            current_loss.append(loss.item())
        lossValues.append(sum(current_loss)/len(current_loss))
    plt.scatter(range(1,len(lossValues)+1),lossValues)
    plt.plot(range(1,len(lossValues)+1),lossValues,color=colors[i], marke
r="o", linestyle='dashed',linewidth=1, markersize=6)
    plt.title("Model "+str(i))
    plt.xlabel("epoches")
    plt.ylabel("loss values")
    # plotData("Cost with respect to models at each epoch","Cost","Epoche
s",range(1,len(lossValues)+1),lossValues,'orange','+')
    return myModel
```

In [51]:
```
def testModelsPredictAccuracy(models,test_batches):
    accs = []
    ypred = []
    ytrue = []
    final = []
    temp = []
    for i in range(len(models)):
        ypred.append([])
    for itr,(x_true, y_true) in enumerate(test_batches):
        temp.clear()
        ytrue = ytrue + y_true.detach().numpy().tolist()
        x_true = (x_true.requires_grad_()).to(current_device)
        for i in range(len(models)):
            yPred = models[i](x_true)
            ypred[i] = ypred[i] + yPred.cpu().detach().numpy().tolist()
            temp.append(yPred)
        z = np.zeros((len(y_true),2))
        for i in range(len(models)):
            z = np.add(z,temp[i].cpu().detach().numpy())
        z = z/len(models)
        accs = accs + ((z/y_true.cpu().detach().numpy())*100).tolist()
        final = final + z.tolist()
        # print(z)
        # print(accs)
        # return
    return accs, ypred, ytrue, final
```

In [52]:
```
models= []
def buildNModelsAndTest(n,train_batches, test_batches,epochs,lr):
    models.clear()
    for i in range(n):
        models.append(trainTestMyModel(train_batches,epochs,lr,i))
    return testModelsPredictAccuracy(models,test_batches)
```

In [53]:
```python
a,b,c,d = buildNModelsAndTest(5,train_batches,test_batches,25,0.001)
```



The above scatter plot shows the training cost of the models trained. different colors shows different models costs.

In [54]:
```python
# a,b,c,d = buildNModelsAndTest(5,train_batches,test_batches,25,0.001)
```

In [55]:
```python
print(len(a),len(b[0]),len(c),len(d))
```

```
5939 5939 5939 5939
```

This is how accurate the predictions were. we divided the predicted value with the actual value and these are the accuracies.

In [56]:
```python
i=randint(0,len(a)-1)
j = randint(0,len(b)-1)
print("Accuracy : ",a[i],"\nPredicted Values by model ["+str(j+1)+"] : ",
b[j][i],"\nActual Value: ",c[i],"\nFinal Value (mean of all models): ",d
[i],"\n")
```

```
Accuracy :  [98.92674052955857, 98.09966131497904]
Predicted Values by model [4] :  [32.29779815673828, -98.04853057861328]
Actual Value:  [32.220001220703125, -100.0199966430664]
Final Value (mean of all models):  [31.874197006225586, -98.119277954101
56]
```

The following is the root mean square error on the final predictions and the true values.

In [57]:
```
from sklearn.metrics import *
print("R Squared Error : ",r2_score(c, d))
print("Mean Squared Error : ",mean_squared_error(c, d))
print("Mean Absolute Error :",mean_absolute_error(c, d))
```

```
R Squared Error :  -0.0855635740099513
Mean Squared Error :  6.543493211116807
Mean Absolute Error : 2.0983884304295204
```

From the above evaluation metrics, we can see that the model is not performing too bad considering the features we are predicting based on the input features.

Although the values are not quite good, but as per our understanding on the data and the dataset, we fee that R-Squared value is good.

The Mean Squared Error value is not in the expected range, because the values are too high as they are in the ranges of 90's and 100's and 110's.

The Mean absolute error gives us a good understanding how the model is predicting the values compared to the actual values.

all the above metrics are calculated with respect to the mean value of the predicted values and the actual values.

We also believe training the model with more layers might increase the accuracy, however we are positive that the difference will not be too much.

Ending latitude and longitude

The following line of code prepares the dataset for ending latitude and longitude as the values to be predicted.

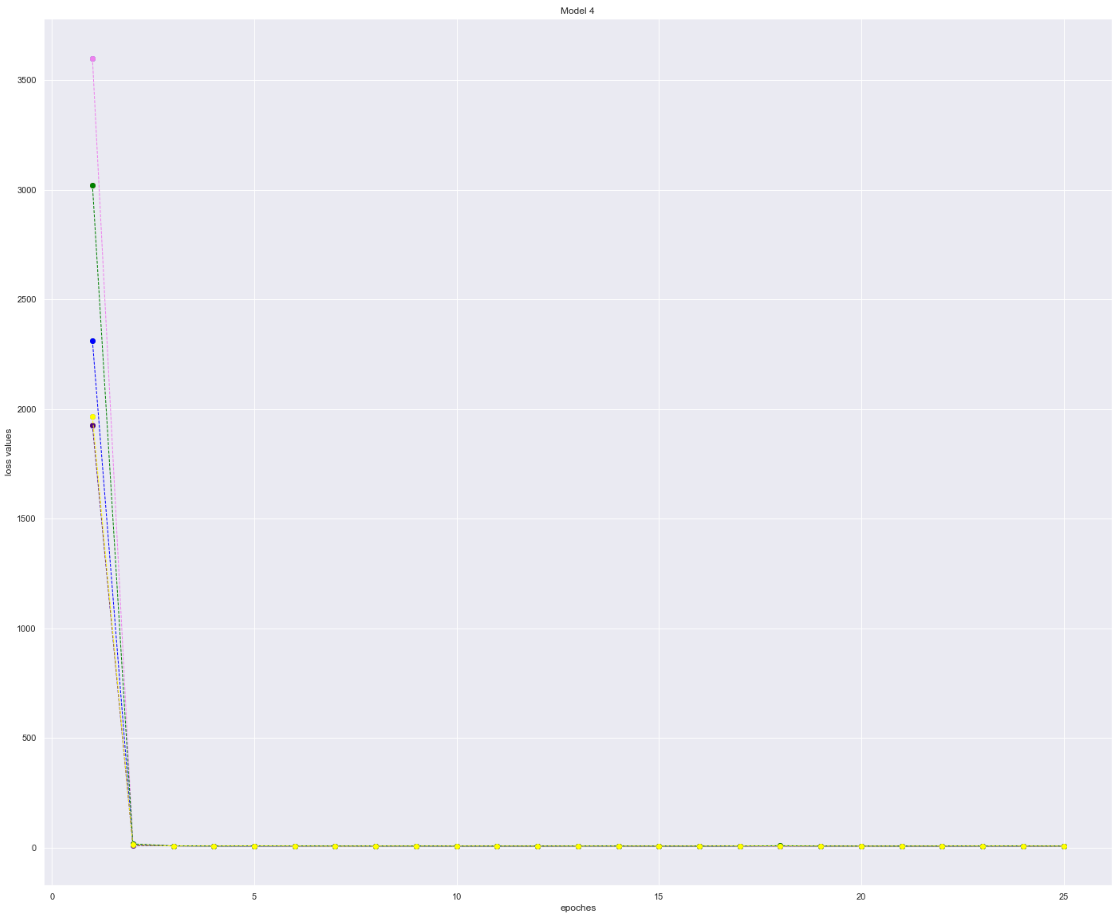
In [58]:
```
my_elat_elon_data = prepareDataset(txdf,selected,['elat','elon'])
```

Now that the data is ready, we can prepare the batches of both training and testing datasets as shown below.

In [59]:
```
ending_train_batches, ending_test_batches = getTrainTestBatches(my_elat_e
lon_data, 0.7, 32)
```

Now that the training and testing batches are ready with batch size of 32. We can train and predict the values as shown below

```
In [60]: ea,eb,ec,ed = buildNModelsAndTest(5,ending_train_batches,ending_test_batc
hes,25,0.001)
```



This is how accurate the predictions were. we divided the predicted value with the actual value and these are the accuracies.

```
In [61]: i=randint(0,len(ea)-1)
j = randint(0,len(eb))
print("Accuracy : ",ea[i],"\nPredicted Values by model ["+str(j+1)+"] :
",eb[j][i],"\nActual Value: ",ec[i],"\nFinal Value (mean of all models):
",ed[i],"\n")
```

```
Accuracy :  [100.9581924843858, 102.83961548496632]
Predicted Values by model [3] :  [31.77961540222168, -96.7448501586914]
Actual Value:  [31.18000030517578, -94.80000305175781]
Final Value (mean of all models):  [31.478764724731445, -97.491958618164
06]
```

The following is the root mean square error on the final predictions and the true values.

In [62]:
```
from sklearn.metrics import *
print("R Squared Error : ",r2_score(ec, ed))
print("Mean Squared Error : ",mean_squared_error(ec, ed))
print("Mean Absolute Error :",mean_absolute_error(ec, ed))
```

```
R Squared Error :  -0.08208154520503408
Mean Squared Error :  6.593897034582552
Mean Absolute Error : 2.1218613368450443
```

From the above evaluation metrics, we can see that the model is not performing too bad considering the features we are predicting based on the input features.

Although the values are not quite good, but as per our understanding on the data and the dataset, we fee that R-Squared value is good.

The Mean Squared Error value is not in the expected range, because the values are too high as they are in the ranges of 90's and 100's and 110's.

The Mean absolute error gives us a good understanding how the model is predicting the values compared to the actual values.

all the above metrics are calculated with respect to the mean value of the predicted values and the actual values.

We also believe training the model with more layers might increase the accuracy, however we are positive that the difference will not be too much.

Starting and Ending latitude and longitude prediction on the whole dataset.

As shown above, we can use the same methods to predict the locations of the tornadoes of the entire dataset.

In [63]:
```
selected = getKFeatures(df, 8, 'slat')
```

In [64]:
```
selected
```

Out[64]:
```
['yr', 'mo', 'dy', 'mag', 'fat', 'loss', 'len', 'wid']
```

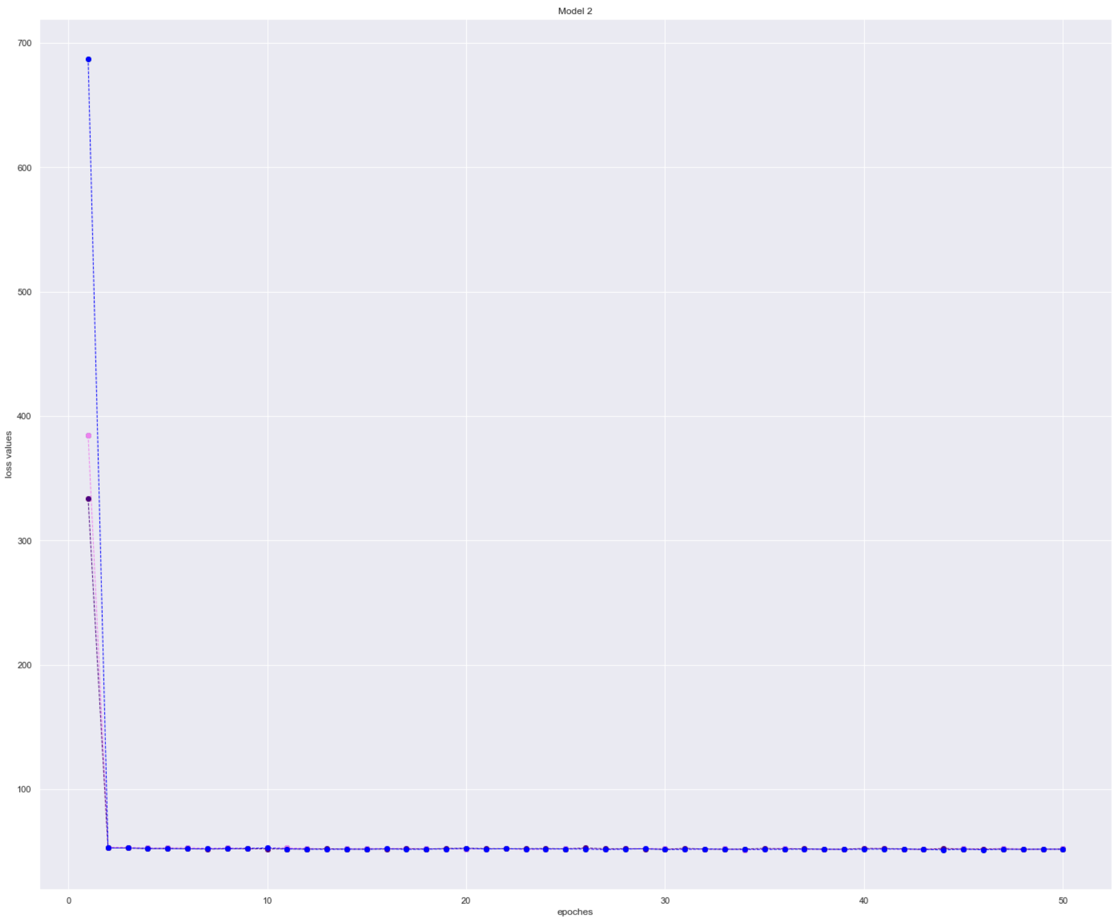
In [65]:
```
starting_whole_data = prepareDataset(df,selected,['slat','slon'])
```

In [66]:
```
selected = getKFeatures(df, 8, 'elat')
```

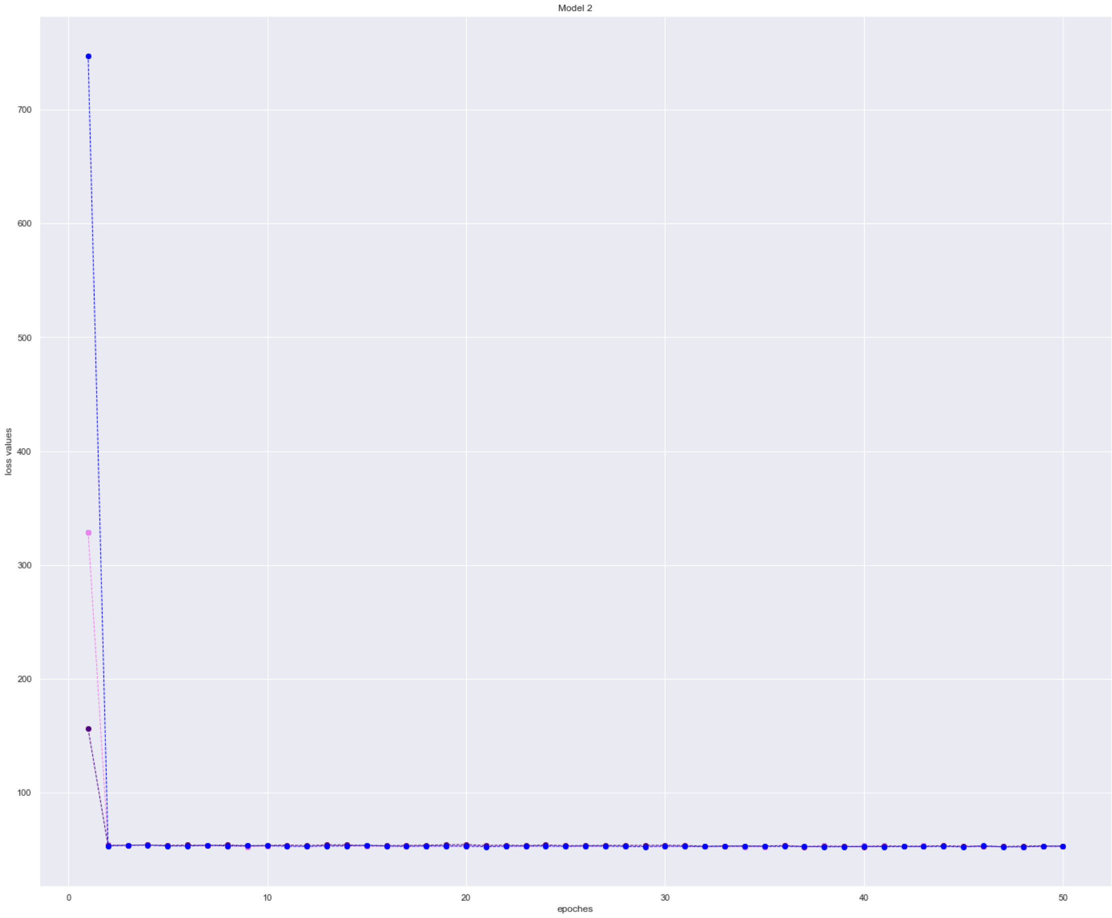
In [67]:
```
ending_whole_data = prepareDataset(df,selected,['elat','elon'])
```

In [68]:
```
sw_train_batches, sw_test_batches = getTrainTestBatches(starting_whole_da
ta, 0.7, 32)
ew_train_batches, ew_test_batches = getTrainTestBatches(ending_whole_dat
a, 0.7, 32)
```

In [69]:
```
swa,swb,swc,swd = buildNModelsAndTest(3,sw_train_batches,sw_test_batches,
50,0.001)
```



In [70]:
```
ewa,ewb,ewc,ewd = buildNModelsAndTest(3,ew_train_batches,ew_test_batches,
50,0.001)
```



In [71]:
```
from sklearn.metrics import *
print("R Squared Error : ",r2_score(swc, swd))
print("Mean Squared Error : ",mean_squared_error(swc, swd))
print("Mean Absolute Error :",mean_absolute_error(swc, swd))
```

```
R Squared Error :  0.002450518179562766
Mean Squared Error :  50.92637604838342
Mean Absolute Error : 5.565257935099395
```

In [72]:
```
from sklearn.metrics import *
print("R Squared Error : ",r2_score(ewc, ewd))
print("Mean Squared Error : ",mean_squared_error(ewc, ewd))
print("Mean Absolute Error :",mean_absolute_error(ewc, ewd))
```

```
R Squared Error :  -0.003370628507374074
Mean Squared Error :  51.10743646102567
Mean Absolute Error : 5.619795906033794
```

```
In [73]: i=randint(0,len(swa)-1)
j = randint(0,len(swb)-1)
print("Accuracy : ",swa[i],"\nPredicted Values by model ["+str(j+1)+"] : ",swb[j][i],"\nActual Value: ",swc[i],"\nFinal Value (mean of all models): ",swd[i],"\n")
```

```
Accuracy :  [96.1109956930709, 98.90034663332648]
Predicted Values by model [3] :  [38.512962341308594, -92.53645324707031]
Actual Value:  [38.779998779296875, -93.44999694824219]
Final Value (mean of all models):  [37.27184295654297, -92.42237091064453]
```

```
In [74]: i=randint(0,len(ewa)-1)
j = randint(0,len(ewb)-1)
print("Accuracy : ",ewa[i],"\nPredicted Values by model ["+str(j+1)+"] : ",ewb[j][i],"\nActual Value: ",ewc[i],"\nFinal Value (mean of all models): ",ewd[i],"\n")
```

```
Accuracy :  [88.36502695015544, 96.73623365467621]
Predicted Values by model [3] :  [35.62492370605469, -90.0240249633789]
Actual Value:  [41.72999954223633, -95.41999816894531]
Final Value (mean of all models):  [36.874725341796875, -92.3057123819987]
```

From the above, Although the model is not performing the best on the whole dataset as we explained earlier that the prediction is mostly dependent on the weather conditions, However, we can see that the model is performing quite well than anticipated. the MSE and MAE are not too high for the model built.

Crop Loss Model

The following code shows the length of the records we have and the information in them

```
In [75]: print(len(df))==None and df.head()
```

```
60114
```

Out[75]:

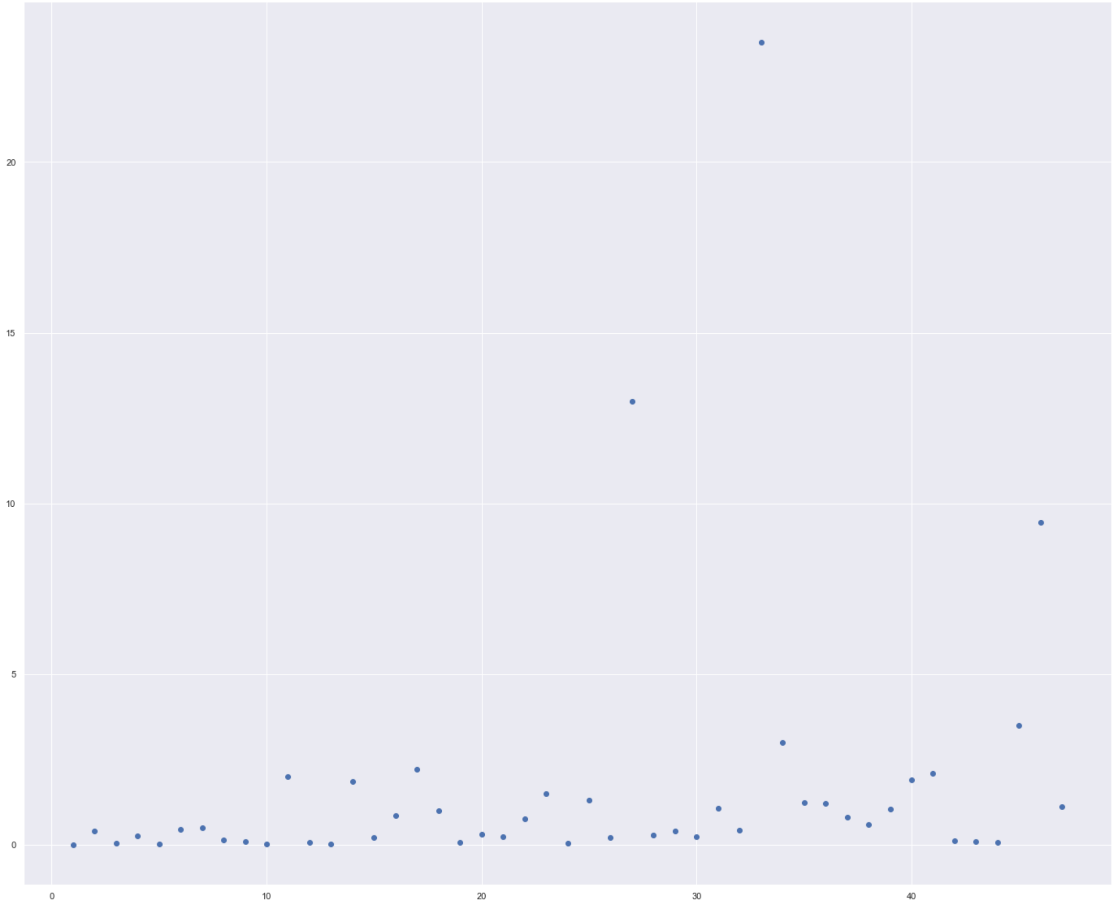
	yr	mo	dy	time	st	mag	inj	fat	loss	closs	slat	slon	
0	1950	1	3	11:00:00	MO	3	3	0	6.00000	0.00000	38.77000	-90.22000	38.8
1	1950	1	3	11:55:00	IL	3	3	0	5.00000	0.00000	39.10000	-89.30000	39.1
2	1950	1	3	16:00:00	OH	1	1	0	4.00000	0.00000	40.88000	-84.58000	40.8
3	1950	1	13	5:25:00	AR	3	1	1	3.00000	0.00000	34.40000	-94.37000	34.4
4	1950	1	25	19:30:00	MO	2	5	0	5.00000	0.00000	37.60000	-90.68000	37.6

The following are the unique values in the closs feature as shown below

In [76]:
```
uv = list(df['closs'].unique())
```

In [77]:
```
plt.scatter(range(1,len(uv)+1),uv)
```

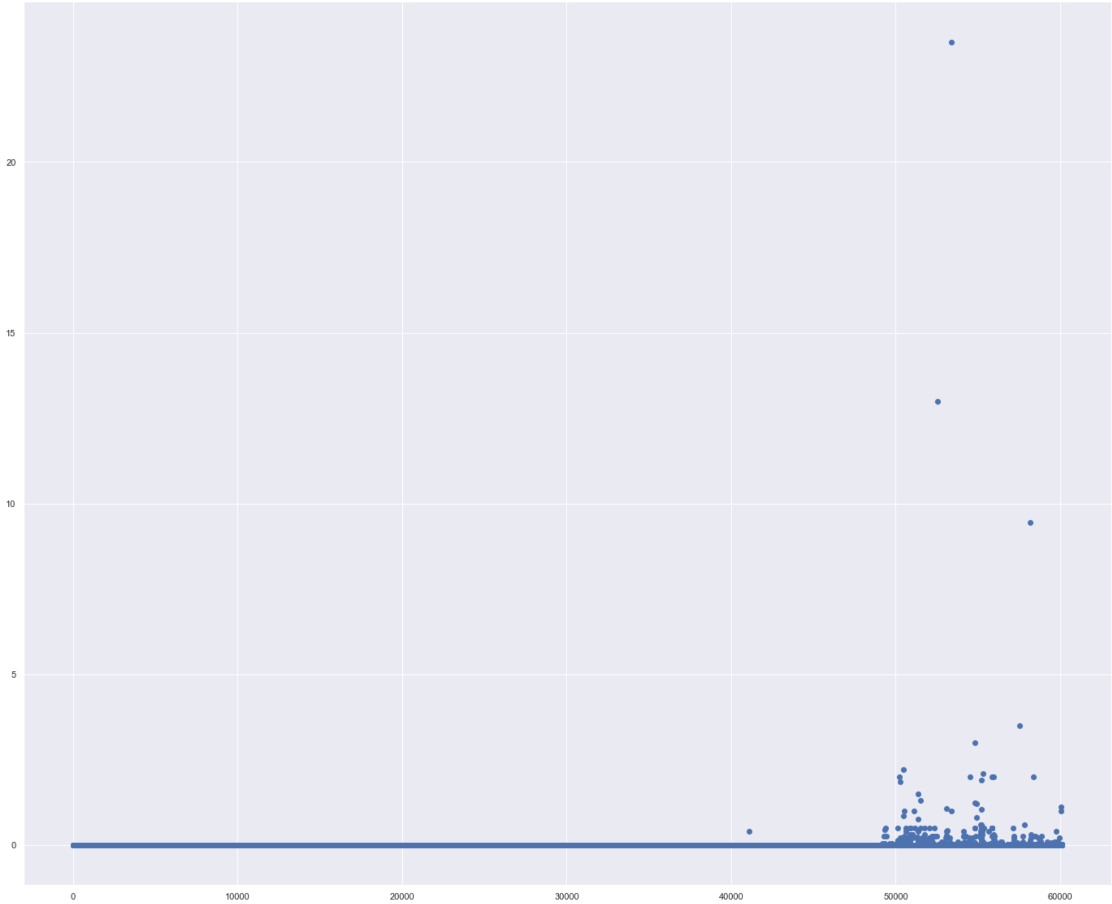
Out[77]:
```
<matplotlib.collections.PathCollection at 0x174db939508>
```



Since there are many unique values, We believe its not a good idea to treat this feature as a classification feature. Although its a numerical features, we wanted to see if we could make this a classification instead of regression. From the above graph, its clear that it is not possible to do that.

In [78]:
```
plt.scatter(range(1,len(df)+1),df['closs'])
```

Out[78]: <matplotlib.collections.PathCollection at 0x174d7954d08>



From the above plot, its clear that the all the crop loss is happened very recently in the dataset as the records shows the crop loss starting from approximately 48000 records which were ver recent records.

In [79]:
```
dfs = df.sort_values('yr')
```

In [80]:
```
dfsc = dfs[dfs['closs']>0]
```

In [81]:
```
len(dfsc)
```

Out[81]: 483

In [82]:
```
dfsc = dfsc.sort_values('closs', ascending=False)
```

```
In [83]: dfsc.head()
```

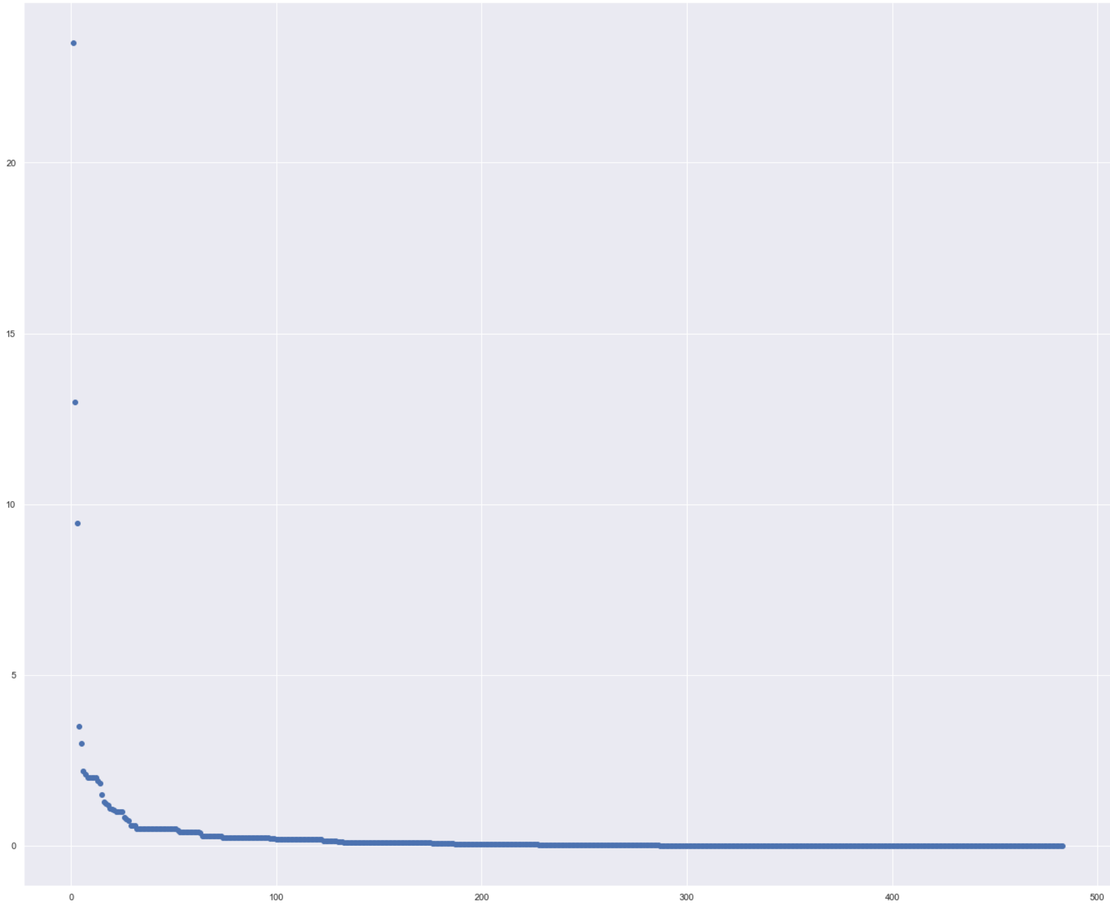
Out[83]:

	yr	mo	dy	time	st	mag	inj	fat	loss	closs	slat	sl
53374	2010	4	24	10:09:00	LA	4	146	10	386.04000	23.52000	32.40000	-91.300
52558	2009	5	8	8:35:00	MO	2	2	0	2.55000	13.00000	37.00000	-91.820
58186	2014	4	28	14:51:00	MS	4	84	10	116.70000	9.45000	32.88000	-89.430
57509	2013	5	27	18:04:00	KS	3	1	0	1.80000	3.50000	39.86000	-98.540
54833	2011	4	15	11:20:00	AL	1	0	0	0.10000	3.00000	31.36000	-87.890

The above information shows that crop loss happened very recently as the years are shown for each loss in the above table.

```
In [84]: plt.scatter(range(1,len(dfsc)+1),dfsc['closs'])
```

Out[84]: <matplotlib.collections.PathCollection at 0x174ea45b748>



From both the above graphs we can see that only the recent years have the crop loss as shown in the above table.

now that we know that year is an import feature that effects the crop loss, let us figure out the other features that are effecting this crop loss featue using k-best features method from sklearn as shown below.

```
In [85]: selected = getKFeatures(df, 8, 'closs')
```

```
In [86]: selected
```

```
Out[86]: ['yr', 'mo', 'mag', 'inj', 'fat', 'loss', 'len', 'wid']
```

As we can see that year is in the front of the list which is followed by the magnitude which is obvious for us to understand how its effect the crop loss.

All the other features are also important in predicting the crop loss caused by the tornado.

now that we have the features, we can train a model on these and see how they help us in predicting our target feature.

The following code shows the creation of the training and testing datasets.

```
In [87]: xTrain, xTest, yTrain, yTest = train_test_split(df[selected], df['closs
         '], test_size=0.3, shuffle = True)
```

Now that we have the data, we can train the model and get the predictions on the crop loss as shown below

```
In [88]: rfRegressor = RandomForestRegressor()
         rfRegressor.fit(xTrain,yTrain)
         yPred = rfRegressor.predict(xTest)
```

Now that we have the predictions ready, lets see some samples from the dataset selected at random and see how much closer they were predicted as shown below.

```
In [89]: yTest = yTest.tolist()
         while True==True:
             i=randint(0,len(yPred)-1)
             if yTest[i]!=0:
                 print(yPred[i],yTest[i])
                 break
```

```
0.0001 0.01
```

From the above we can see that the model is predicting the values properly and the error seems to be minimal. Lets now check how the model is performingn with the choosen evaluation metrics for the regression problem as shown below.

```
In [90]: from sklearn.metrics import *
         print("R Squared Error : ",r2_score(ec, ed))
         print("Mean Squared Error : ",mean_squared_error(ec, ed))
         print("Mean Absolute Error :",mean_absolute_error(ec, ed))
```

```
R Squared Error :  -0.08208154520503408
Mean Squared Error :  6.593897034582552
Mean Absolute Error : 2.1218613368450443
```

From the above evaluation metrics, we can see that the model is not performing too bad considering the features we are predicting based on the input features.

Although the values are not quite good, but as per our understanding on the data and the dataset, we fee that R-Squared value is good.

The Mean Squared Error value is not in the expected range, because the values are too high as they are in the ranges of 90's and 100's and 110's.

The Mean absolute error gives us a good understanding how the model is predicting the values compared to the actual values.

all the above metrics are calculated with respect to the mean value of the predicted values and the actual values.

We also believe training the model with more layers might increase the accuracy, however we are positive that the difference will not be too much.

We are building multiple models to predict/classify different target variables. The sections below show the model creation and evaluation for different target variables such as starting latitude, starting longitude, ending latitude, ending longitude, loss, crop loss, length, width and magnitude of the tornado.

4. Building the Model to classify magnitude

Now that we have the cleaned and preprocessed data, we can start building the model and train and test it according to our requirement to predict the magnitude of the tornado based on the other features present in the dataset.

We will use nunique and unique function to get the unique values across the magnitude to column to understand it better before commencing the model building.

In [91]:
```
df['mag'].nunique()
```

Out[91]:
```
6
```

In [92]:
```
df['mag'].unique()
```

Out[92]:
```
<IntegerArray>
[3, 1, 2, 4, 0, 5]
Length: 6, dtype: Int64
```

From the above we can see that the magnitude feature have only 6 attributes in the feature. Hence, We believe we can convert this numerical feature in to a categorical feature and use it predict the values accordingly.

so, to perform these operations we would like to take a copy of the actual dataset and then try on it as shown below.

In [93]:
```
cdf = df.copy()
```

now that we have a copy of the dataframe, we can start converting our numeric magnitude feature to a categorical feature as we wanted earlier which is shown below.

In [94]:
```
cdf = cdf.astype({'mag':'string'})
```

Now that the feature is conveted, we can now select the features that are having high influence on this feature and then use them to predict the value.

the following are the list of columns in the dataset.

In [95]:
```
cols = list(cdf.columns)
```

In [96]:
```
cols
```

Out[96]:
```
['yr',
 'mo',
 'dy',
 'time',
 'st',
 'mag',
 'inj',
 'fat',
 'loss',
 'closs',
 'slat',
 'slon',
 'elat',
 'elon',
 'len',
 'wid',
 'ones']
```

Since we will be predicting the magnitude feature, we don't want that to be in the input features to the model. Hence, we drop it as shown below.

In [97]:
```
cols.remove('mag')
```

In [98]:
```
cols
```

Out[98]:
```
['yr',
 'mo',
 'dy',
 'time',
 'st',
 'inj',
 'fat',
 'loss',
 'closs',
 'slat',
 'slon',
 'elat',
 'elon',
 'len',
 'wid',
 'ones']
```

Now that we have the dataset and the columns that we need for the prediction, we want to know which columns are best to predict the magnitude features. Since the whoe dataset contains most of the features as numeric, we believe that we can use select I-best features method from sklearn that will help us with the best features as shown below.

In [99]:
```
def getKFeatures(data,kValue, target):
    predictors = list(data.describe(include=['number']).columns)
    predictors.remove(target)
    bestFeatureSelection = SelectKBest(k=kValue)
    _ = bestFeatureSelection.fit_transform(data[predictors],data[target])
    bools = bestFeatureSelection.get_support()
    selected = []
    for i in range(len(predictors)):
        if bools[i]==True:
            selected.append(predictors[i])
    return selected
```

In [100]:
```
getKFeatures(df,5, 'mag')
```

Out[100]:
```
['yr', 'inj', 'fat', 'len', 'wid']
```

In [101]:
```
getKFeatures(df,7, 'mag')
```

Out[101]:
```
['yr', 'inj', 'fat', 'loss', 'elon', 'len', 'wid']
```

In [102]:
```
getKFeatures(df,9, 'mag')
```

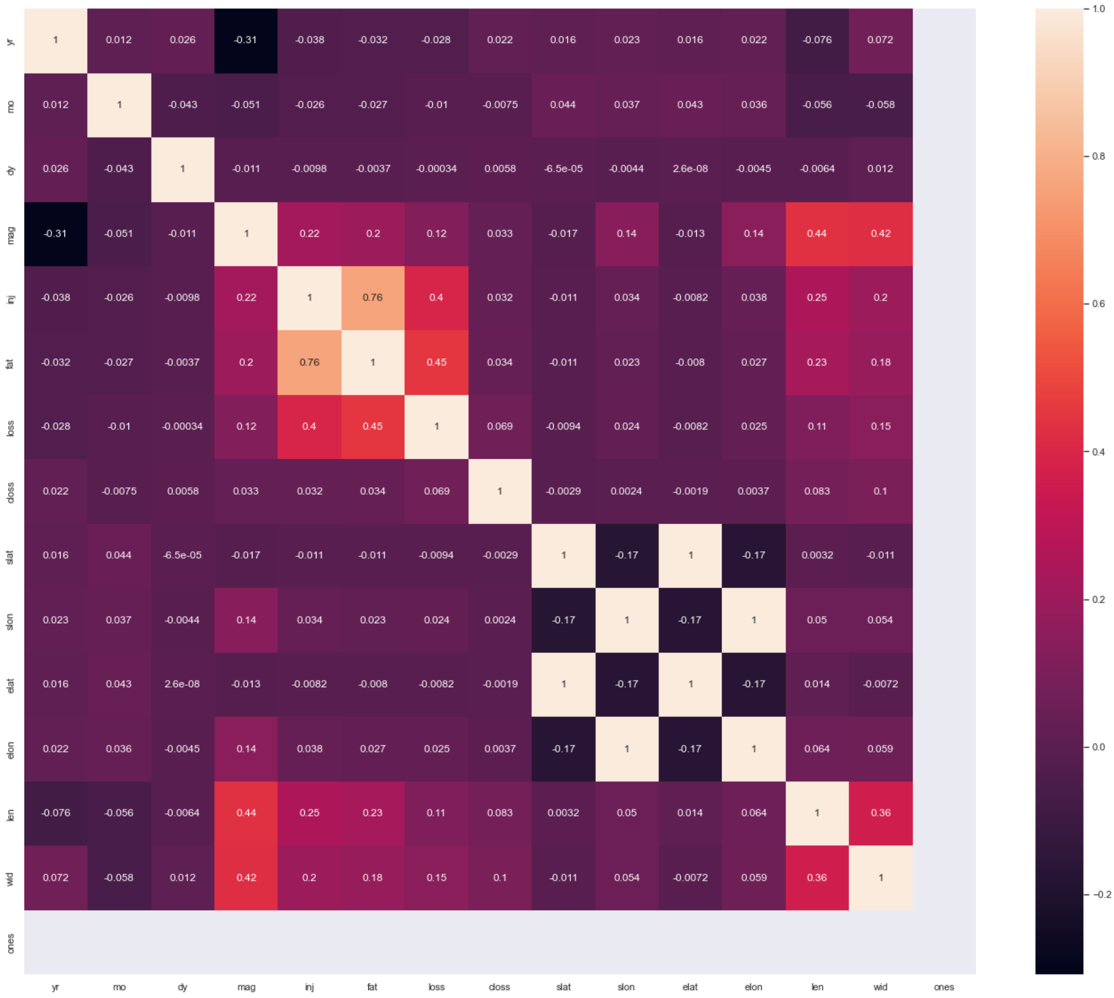
Out[102]:
```
['yr', 'mo', 'inj', 'fat', 'loss', 'slon', 'elon', 'len', 'wid']
```

As shown above, the above shown features are the best features for predicting the magnitude of the tornado.

From the above features, we can notice that len and wid are the features that were given repeatedly.

The following coorelatino graph also shows the relation between the magnitude feature and the other features of the dataset clearly.

In [103]:
```
sns.heatmap(df.corr(), annot=True);
```



From the above figure we can see that there is a high correlation between wid,len, fat, loss and fat etc features. Hencew we believe all these featuers will help us in predicting the magnitude feature using ensemble methods.

Using train_test_split() from the data science library scikit-learn for spliting the dataset into subsets. It is essential as it minimize the potential for bias in the process.

Now that we have the features ready, we can start building our train and test datasets as shown below.

In [104]:
```
x_train, x_test, y_train, y_test= train_test_split(cdf[cols], cdf['mag'],
train_size=0.7, shuffle=True, random_state=1)
```

When preparing a big dataset for training, it is critical to select the optimal features. We can use the SelectKBest technique to chooses features based on the k highest score. After selecting the featues, we used RandomForestClassifier to build our model.

In [105]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
def buildModelAndRun(cols):
    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        rf_model = RandomForestClassifier(n_estimators=200,max_depth=15,
criterion="entropy",warm_start=True,max_features="sqrt")
        rf_model.fit(x_train[cols],y_train)
        y_pred = rf_model.predict(x_test[cols])
        print(" Accuracy : ",accuracy_score(y_test,y_pred)*100)
        return y_pred, rf_model
```

In [106]:
```
_ , _ = buildModelAndRun(getKFeatures(df,13, 'mag'))
```

```
 Accuracy :  70.44080953701136
```

The following method is a newer version of the previous verion which is changed to perform the hyper paramter tunining on the model.

In [107]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
def buildModelAndRun(cols,est,md):
    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        rf_model = RandomForestClassifier(n_estimators=est, max_depth=md,
criterion="entropy", warm_start=True, max_features="sqrt")
        rf_model.fit(x_train[cols],y_train)
        y_pred = rf_model.predict(x_test[cols])
        print(" Accuracy : ",accuracy_score(y_test,y_pred)*100)
        return y_pred, rf_model
```

The following shows the hyperparameter tuning of the model with different estimators values and depth of the trees in the forest.

In [108]:
```
_ , _ = buildModelAndRun(getKFeatures(df,13, 'mag'),est=150,md=15) # esti
mators = 150
```

```
 Accuracy :  70.40199611865816
```

In [109]:
```
_ , _ = buildModelAndRun(getKFeatures(df,13, 'mag'),est=300,md=13) # esti
mators = 300
```

```
 Accuracy :  70.01386193512614
```

In [110]:
```
_ , _ = buildModelAndRun(getKFeatures(df,13, 'mag'),est=130,md=30) # esti
mators = 130
```

```
 Accuracy :  70.8788466869975
```

In [111]:
```
_ , _ = buildModelAndRun(getKFeatures(df,13, 'mag'),est=35,md=15) # estim
ators = 130
```

```
Accuracy :  69.80870529525922
```

In [112]:
```
_ , _ = buildModelAndRun(getKFeatures(df,13, 'mag'),est=600,md=100) # est
imators = 600
```

```
Accuracy :  71.03964513446077
```

From All the above runs we figured out that maximum accuracy we could achieve is 71.144.

5. Building the Model to predict the length of tornado

Please note that a tornado with magnitude zero doesn' imply an absence of Tornadoes. An EF0 tornado is the weakest tornado on the Enhanced Fujita Scale. An EF0 will have wind speeds between 65 and 85 mph (105 and 137 km/h).

Reference: [Facts-just-for-kids (https://www.factsjustforkids.com/weather-facts/tornado-facts-for-kids/enhanced-fujita-scale/ef0-tornado/)](https://www.factsjustforkids.com/weather-facts/tornado-facts-for-kids/enhanced-fujita-scale/ef0-tornado/)

In [113]:
```
df['mag'].unique()
```

Out[113]:
```
<IntegerArray>
[3, 1, 2, 4, 0, 5]
Length: 6, dtype: Int64
```

In [114]:
```
(df['len']==0).sum()
```

Out[114]:
```
123
```

The column 'len' has 123 instances which have a value of zero. For each instance we are going to replace the zeroes with the mean length for the corresponding magnitude of the instance.

In [115]:
```
df[df['len']==0].head(10)
```

Out[115]:

	yr	mo	dy	time	st	mag	inj	fat	loss	closs	slat	slon
34416	1995	5	1	14:45:00	PR	0	0	0	0.00000	0.00000	18.41000	-66.11000
39500	1999	4	5	13:28:00	NE	0	0	0	0.01000	0.00000	40.02000	-95.63000
39501	1999	4	5	13:32:00	NE	0	0	0	0.00000	0.00000	40.08000	-95.73000
39523	1999	4	8	13:08:00	IA	0	0	0	0.00000	0.00000	40.60000	-95.52000
39605	1999	4	21	18:20:00	NE	0	0	0	0.00000	0.00000	40.83000	-96.38000
39709	1999	5	3	20:15:00	KS	0	0	0	0.00000	0.00000	37.65000	-97.02000
39711	1999	5	3	20:24:00	NE	0	0	0	0.00000	0.00000	42.63000	-98.08000
39716	1999	5	3	20:32:00	NE	0	0	0	0.00000	0.00000	42.82000	-98.12000
39801	1999	5	9	15:00:00	WA	0	0	0	0.00000	0.00000	46.67000	-120.62000
39852	1999	5	16	17:28:00	KS	0	0	0	0.00000	0.00000	38.48000	-97.17000

In [116]:
```
df.groupby('mag')['len'].agg([np.mean])
```

Out[116]:

	mean
mag	
0	1.02036
1	3.19796
2	6.95947
3	14.96380
4	27.56085
5	39.00780

We have only two types of magnitudes with length - 0 , Mag=0 and Mag=1. Let's replace these with the mean values of length.

In [117]:
```
df['len'] = np.where((df['len'] == 0) & (df['mag'] == 0), 1.02036, df['le
n'])
```

In [118]:
```
df[df['len']==0]
```

Out[118]:

	yr	mo	dy	time	st	mag	inj	fat	loss	closs	slat	slon
40277	1999	7	9	16:00:00	OH	1	0	0	0.10000	0.00000	41.08000	-82.40000
43939	2003	4	30	17:42:00	IL	1	0	0	0.50000	0.00000	41.15000	-90.75000
46266	2004	8	12	14:25:00	NC	1	0	0	0.00000	0.00000	36.52000	-79.53000
46348	2004	8	26	21:28:00	IA	1	0	0	0.01000	0.00000	42.07000	-91.62000

In [119]:
```
df['len'] = np.where((df['len'] == 0) & (df['mag'] == 1), 3.19796, df['le
n'])
```

Let's Replace zeroes for the column - "wid"/width as well

The column 'len' has 473 instances which have a value of zero. For each instance we are going to replace the zeroes with the mean width for the corresponding magnitude of the instance.

In [120]:
```
df[df['wid']==0].head(10)
```

Out[120]:

	yr	mo	dy	time	st	mag	inj	fat	loss	closs	slat	slon
39176	1999	1	1	13:00:00	TX	1	7	0	0.12000	0.00000	30.75000	-95.53000
39177	1999	1	1	13:23:00	TX	0	1	0	0.01000	0.00000	30.85000	-95.42000
39179	1999	1	1	13:53:00	TX	0	0	0	0.01000	0.00000	30.72000	-95.37000
39186	1999	1	1	22:26:00	LA	2	1	0	1.00000	0.00000	32.38000	-93.80000
39187	1999	1	1	22:33:00	TX	1	0	0	0.07000	0.00000	30.10000	-95.23000
39188	1999	1	1	22:36:00	LA	1	0	0	0.01000	0.00000	32.42000	-93.62000
39192	1999	1	2	0:20:00	TX	1	0	0	0.04000	0.00000	29.52000	-94.48000
39193	1999	1	2	0:26:00	TX	0	0	0	0.03000	0.00000	29.53000	-94.48000
39212	1999	1	2	11:40:00	FL	0	0	0	0.00000	0.00000	30.72000	-86.87000
39215	1999	1	2	15:30:00	FL	0	0	0	0.03000	0.00000	30.10000	-85.22000

In [121]:
```
df[df['wid']==0]['mag'].unique()
```

Out[121]:
```
<IntegerArray>
[1, 0, 2, 3]
Length: 4, dtype: Int64
```

In [122]:
```
df.groupby('mag')['wid'].agg([np.mean])
```

Out[122]:

mag	mean
0	41.56138
1	95.52292
2	175.65890
3	363.32007
4	588.64425
5	839.06780

We have only 4 types of magnitudes with width - 0 , Mag=0, Mag=1, Mag=2, Mag=3. Let's replace these with the mean values of width.

In [123]:
```
df['wid'] = np.where((df['wid'] == 0) & (df['mag'] == 0), 41.56138, df['w
id'])
```

In [124]:
```
df['wid'] = np.where((df['wid'] == 0) & (df['mag'] == 1), 95.52292, df['w
id'])
```

In [125]:
```
df['wid'] = np.where((df['wid'] == 0) & (df['mag'] == 2), 175.65890, df['
wid'])
```

In [126]:
```
df['wid'] = np.where((df['wid'] == 0) & (df['mag'] == 3), 363.32007, df['
wid'])
```

In [127]:
```
df[df['wid']==0]
```

Out[127]:

yr	mo	dy	time	st	mag	inj	fat	loss	closs	slat	slon	elat	elon	len	wid	ones

Exploratory Data Analysis

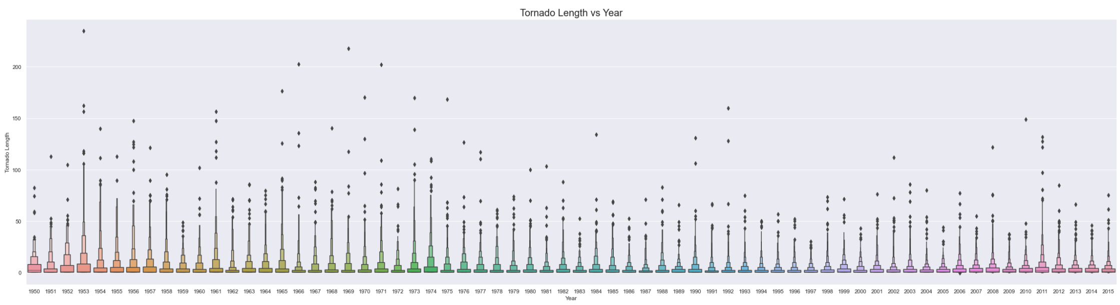
Tornado Length vs Year

Boxen Plot between Tornado Length and Year

We have used a Boxen plot since Boxen plots provide a better representation of distribution of data than boxplots, especially for large datasets.

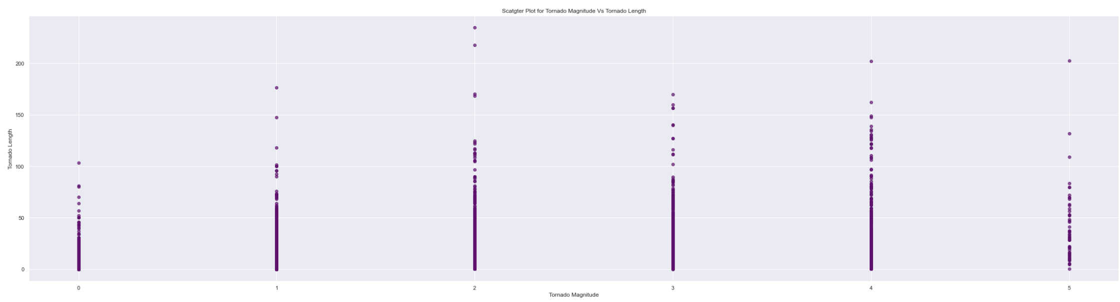
Reference: [Seaborn Boxenplot (https://seaborn.pydata.org/generated/seaborn.boxenplot.html)](https://seaborn.pydata.org/generated/seaborn.boxenplot.html)

In [128]:
```
plt.rcParams['figure.figsize']=(40, 10)
plt.style.use('seaborn-dark-palette')
sns.boxenplot(df['yr'],df['len'])
plt.xlabel('Year')
plt.ylabel('Tornado Length')
plt.title('Tornado Length vs Year',fontsize=20)
plt.show()
```



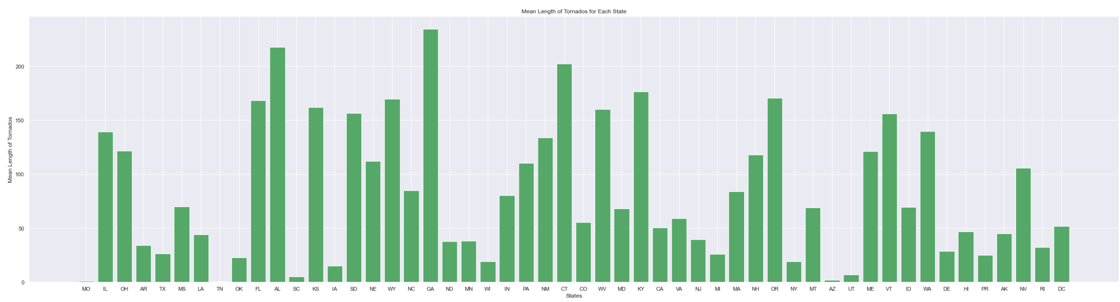
```
In [129]: colors = np.random.rand(3)
          x=df['mag']
          y=df['len']
          plt.scatter(x, y, c=colors, alpha=0.7)
          plt.xlabel('Tornado Magnitude')
          plt.ylabel('Tornado Length')
          plt.title('Scatgter Plot for Tornado Magnitude Vs Tornado Length')
          plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which sho
uld be avoided as value-mapping will have precedence in case its length
matches with *x* & *y*.  Please use the *color* keyword-argument or prov
ide a 2D array with a single row if you intend to specify the same RGB o
r RGBA value for all points.
```



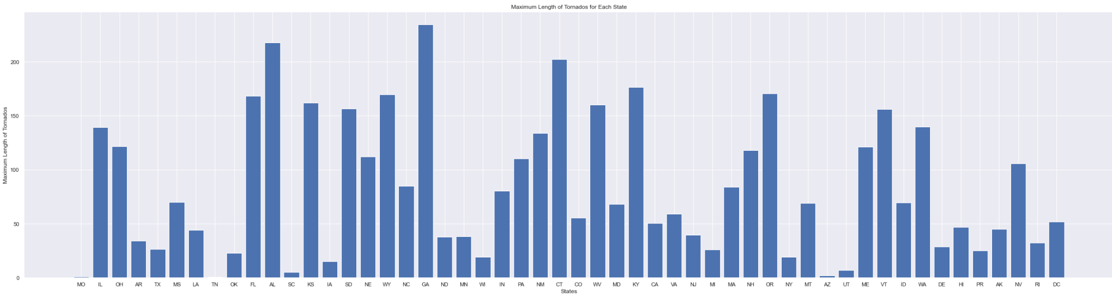
Mean Length of Tornadoes in each State

```
In [130]: plt.style.use('seaborn-dark-palette')
          plt.bar(df['st'].unique(),df.groupby("st")["len"].max(),color='g')
          plt.xlabel('States')
          plt.ylabel('Mean Length of Tornados')
          plt.title('Mean Length of Tornados for Each State')
          plt.show()
```



Maximum Length of Tornadoes in each State

In [131]:
```
plt.style.use('seaborn-dark-palette')
plt.bar(df['st'].unique(),df.groupby("st")["len"].max(),color='b')
plt.xlabel('States')
plt.ylabel('Maximum Length of Tornados')
plt.title('Maximum Length of Tornados for Each State')
plt.show()
```

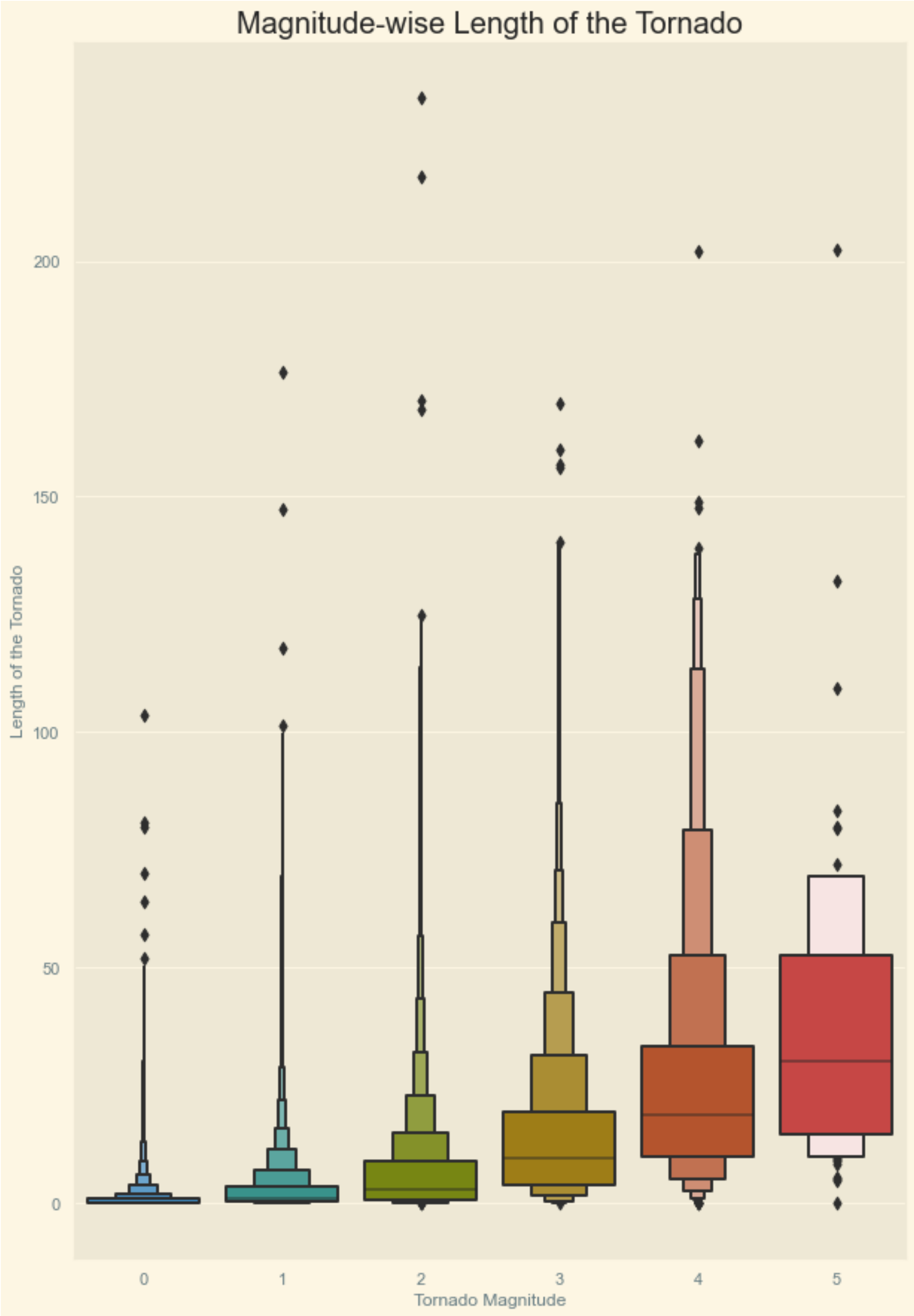


Magnitude-wise Length of the Tornado

In [132]:
```
df = df.convert_dtypes()
```

In [133]:

```
plt.rcParams['figure.figsize']=(10, 15)
plt.style.use('Solarize_Light2')
sns.boxenplot(df['mag'],df['len'])
plt.xlabel('Tornado Magnitude')
plt.ylabel('Length of the Tornado')
plt.title('Magnitude-wise Length of the Tornado',fontsize=20)
plt.show()
```



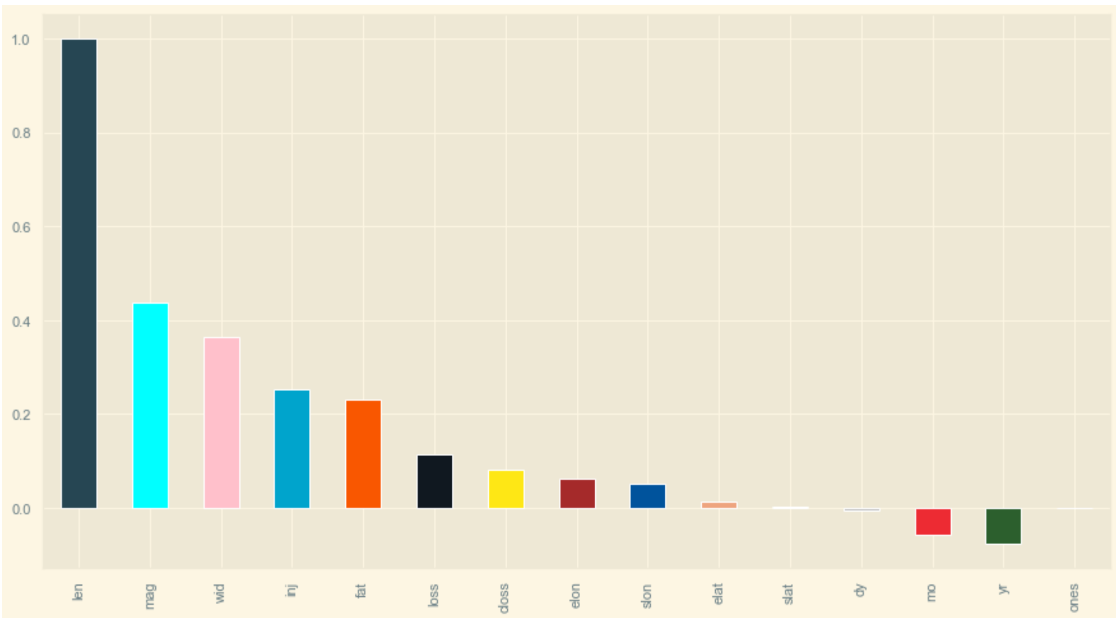
```
In [134]: corr = df.corr()
corr = (corr)
a4_dims=(20,10)
fig, ax = plt.subplots(figsize=a4_dims)
sns.heatmap(corr,annot=True,fmt='f',xticklabels=corr.columns.values,yticklabels=corr.columns.values)
```

Out[134]: <AxesSubplot:>

Correlation Between the Feature length and other Features

```
In [135]: plt.figure(figsize=(15,8))
colors = ['#264653','cyan', 'pink','#00A4CCFF', '#F95700FF','#101820FF', '#FEE715FF', 'brown','#00539CFF', '#EEA47FFF','gold', 'silver','#ED2B33FF', '#2C5F2D']
df.corr()['len'].sort_values(ascending = False).plot(kind = 'bar', color = colors)
```

Out[135]: <AxesSubplot:>



Encoding Values of States

In [136]:
```
df1= pd.get_dummies(df, columns=['st'])
```

Dropping Unnecessary Columns

In [137]:
```
Y=df1.len.astype(float)
X=df1.drop(['len','dy','yr','time','ones'],axis=1).astype(float)
print(type(X))
X.head(5)
```

```
<class 'pandas.core.frame.DataFrame'>
```

Out[137]:

	mo	mag	inj	fat	loss	closs	slat	slon	elat	
0	1.00000	3.00000	3.00000	0.00000	6.00000	0.00000	38.77000	-90.22000	38.83000	-90
1	1.00000	3.00000	3.00000	0.00000	5.00000	0.00000	39.10000	-89.30000	39.12000	-89
2	1.00000	1.00000	1.00000	0.00000	4.00000	0.00000	40.88000	-84.58000	40.88000	-84
3	1.00000	3.00000	1.00000	1.00000	3.00000	0.00000	34.40000	-94.37000	34.40000	-94
4	1.00000	2.00000	5.00000	0.00000	5.00000	0.00000	37.60000	-90.68000	37.63000	-90

In [138]:
```
Y.head(10)
```

Out[138]:
```
0     9.50000
1     3.60000
2     0.10000
3     0.60000
4     2.30000
5     0.10000
6     4.70000
7     9.90000
8    12.00000
9     4.60000
Name: len, dtype: float64
```

Normalization of Input Features

In [139]:
```
X = (X-X.min()) / (X.max()-X.min())
X.head(5)
```

Out[139]:

	mo	mag	inj	fat	loss	closs	slat	slon	elat	elon
0	0.00000	0.60000	0.00172	0.00000	0.00214	0.00000	0.48123	0.74328	0.48263	0.74521
1	0.00000	0.60000	0.00172	0.00000	0.00179	0.00000	0.48893	0.75261	0.48939	0.75332
2	0.00000	0.20000	0.00057	0.00000	0.00143	0.00000	0.53043	0.80047	0.53043	0.80047
3	0.00000	0.60000	0.00057	0.00633	0.00107	0.00000	0.37934	0.70121	0.37934	0.70121
4	0.00000	0.40000	0.00287	0.00000	0.00179	0.00000	0.45395	0.73862	0.45465	0.73892

Performing Feature Selection using the Mutual Info Regression Score Function

In [140]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.feature_selection import mutual_info_regression

selector = SelectKBest(mutual_info_regression, k=9)
selector.fit(X, Y)
# Get columns to keep and create new dataframe with the K best columns
cols = selector.get_support(indices=True)
X = X.iloc[:,cols]

print(X.shape)
print("\n")
print(X.columns)
print("\n")
print(Y.shape)
```

```
(60114, 9)


Index(['mo', 'mag', 'inj', 'loss', 'slat', 'slon', 'elat', 'elon', 'wid
'], dtype='object')


(60114,)
```

Splitting Data into Train and Test

In [141]:
```
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)
x_train.shape,x_test.shape,y_train.shape, y_test.shape
```

Out[141]:
```
((42079, 9), (18035, 9), (42079,), (18035,))
```

Linear Regression Model

Reference: [Linear Regression (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)

In [142]:
```
from sklearn.linear_model import LinearRegression
LR = LinearRegression()
# fitting the training data
history = LR.fit(x_train,y_train)
y_prediction =  LR.predict(x_test)
```

Evaluation of the Linear Regression Model

Reference[Metrics and scoring (https://scikit-learn.org/stable/modules/model_evaluation.html)](https://scikit-learn.org/stable/modules/model_evaluation.html)

In [143]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
r2 = r2_score(y_test, y_prediction)
mse = mean_squared_error(y_test,y_prediction,squared=False)
mae = mean_absolute_error(y_test, y_prediction)
print('r2 Score: ',r2)
print('mean_sqrd_error: ',mse)
print('mean absolute error: ',mae)
```

```
r2 Score:  0.8987999814112838
mean_sqrd_error:  2.679952731035056
mean absolute error:  1.072856917362297
```

In [144]:
```
from sklearn.model_selection import learning_curve
train_sizes = [1, 100, 500, 2000, 5000, 10000, 20000, 30000]

train_sizes, train_scores, validation_scores = learning_curve(
estimator = LinearRegression(),
X = X,
y = Y, train_sizes = train_sizes, cv = 2,
scoring = 'neg_mean_squared_error')
```

In [145]:
```
train_scores_mean = -train_scores.mean(axis = 1)
validation_scores_mean = -validation_scores.mean(axis = 1) #Changed the s
ign of the mean validation scores
print('Mean training scores\n\n', pd.Series(train_scores_mean, index = tr
ain_sizes))
print('\n', '-' * 20) # separator
print('\nMean validation scores\n\n',pd.Series(validation_scores_mean, in
dex = train_sizes))
```

```
Mean training scores

 1       -0.00000
100      3.12596
500      5.39347
2000     8.21369
5000     7.15664
10000    8.76693
20000    8.62437
30000    8.18880
dtype: float64

 --------------------

Mean validation scores

 1          98.38414
100      89242.77398
500         10.59818
2000        12.38529
5000        13.44658
10000        9.71966
20000        9.85214
30000       10.28731
dtype: float64
```

In [146]:
```
plt.style.use('seaborn')
plt.plot(train_sizes, train_scores_mean, label = 'Training error')
plt.plot(train_sizes, validation_scores_mean, label = 'Validation error')
plt.ylabel('MSE', fontsize = 14)
plt.xlabel('Training set size', fontsize = 14)
plt.title('Learning curves for our linear regression model', fontsize = 1
8, y = 1.03)
plt.legend()
plt.ylim(0,100)
```

Out[146]: (0.0, 100.0)

Development of a Bagging Ensemble using a Linear Regressor as a Base Estimator for Prediction of Tornado Lengths

Reference: [sklearn.ensemble.BaggingRegressor (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html)](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html)

```
In [147]: # evaluate bagging ensemble for regression
from numpy import mean
from numpy import std
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.ensemble import BaggingRegressor
# define dataset
X_Input_Features = X
Y_Target_Feature = Y
# define the model
model = BaggingRegressor(LinearRegression())
# evaluate the model
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X_Input_Features, Y_Target_Feature, sco
ring='neg_mean_absolute_error', cv=cv, n_jobs=-1, error_score='raise')
# report performance
print('MAE: %.3f (%.3f)' % (-mean(n_scores), std(n_scores)))
```

```
MAE: 1.080 (0.032)
```

Decision Tree Regressor for Predicting Length of Tornadoes

```
In [148]: # Fitting Decision Tree Regression to the dataset
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)

# ccp_alpha=0.0, criterion='squared_error', max_depth=6,
#                       max_features=None, max_leaf_nodes=None,
#                       min_impurity_decrease=0.0, min_samples_leaf=1,
#                       min_samples_split=2, min_weight_fraction_leaf=0.
0,
#                       random_state=None, splitter='best'
```

```
Out[148]: DecisionTreeRegressor(ccp_alpha=0.0, criterion='squared_error', max_dept
h=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      random_state=None, splitter='best')
```

```
In [149]: y_pred = regressor.predict(x_test)
```

```
In [150]: result = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
```

In [151]:
```
from sklearn import metrics
from sklearn.metrics import r2_score
print("Mean Absolute Error:", metrics.mean_absolute_error(y_test, y_pre
d))
print("Mean Squared Error:" , metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error:", np.sqrt(metrics.mean_squared_error(y_te
st, y_pred)))
print("R Squared Score is:", r2_score(y_test, y_pred))
```

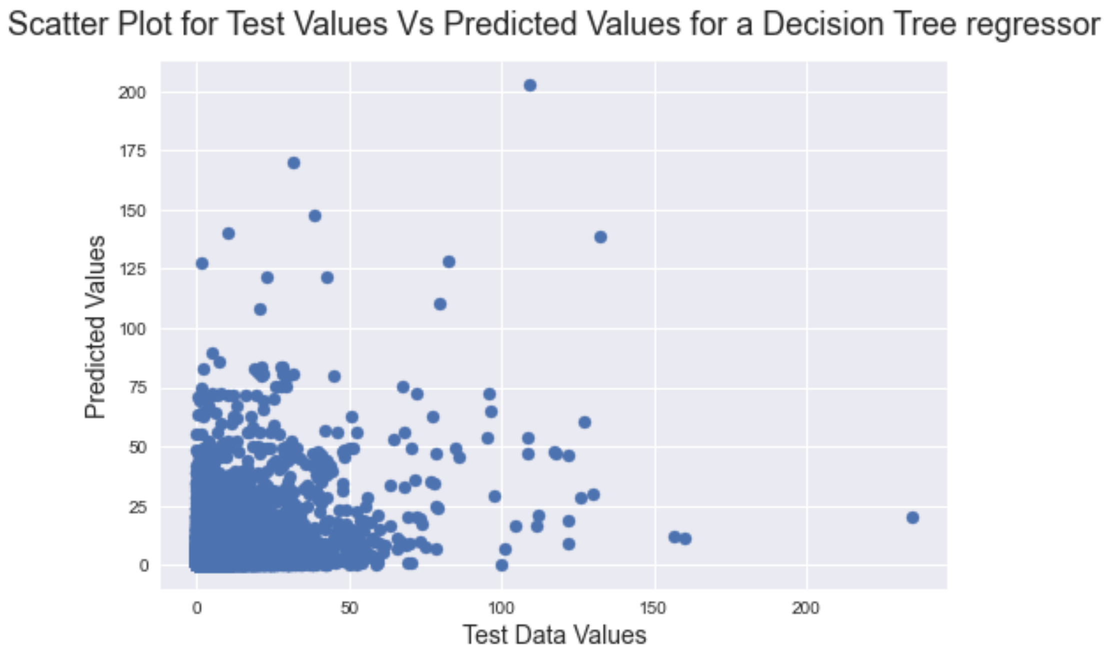
```
Mean Absolute Error: 3.5438506969780987
Mean Squared Error: 76.62383208237547
Root Mean Squared Error: 8.753503988825017
R Squared Score is: -0.07966790698197501
```

In [152]:
```
regressor.score(x_test,y_test)
```

Out[152]:
```
-0.07966790698197501
```

In [153]:
```
plt.scatter(y_test,y_pred)
plt.ylabel('Predicted Values', fontsize = 14)
plt.xlabel('Test Data Values', fontsize = 14)
plt.title('Scatter Plot for Test Values Vs Predicted Values for a Decisio
n Tree regressor', fontsize = 18, y = 1.03)
```

Out[153]:
```
Text(0.5, 1.03, 'Scatter Plot for Test Values Vs Predicted Values for a
Decision Tree regressor')
```



Random Forest Regressor

In [154]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
```

In [155]:
```
rf.fit(x_train, y_train)
# make a single prediction
ypred_rf = rf.predict(x_test)
```

In [156]:
```
errors = abs(ypred_rf - y_test)
# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 2))
```

```
Mean Absolute Error: 2.88
```

6. Building the Model to predict loss

Extracting only starting latitude, starting longitude and loss in a separate dataframe

In [157]:
```
df_tornado_l = df[['slat', 'slon', 'loss']]
```

Normalizing loss feature between 0 and 1 to satisfy folium requirements

In [158]:
```
a = df_tornado_l["loss"]
norm_a = (a - a.min())/(a.max() - a.min())
df_tornado_l["loss_norm"] = norm_a
df_tornado_l.drop('loss', axis=1, inplace=True)
df_tornado_l.head()
```

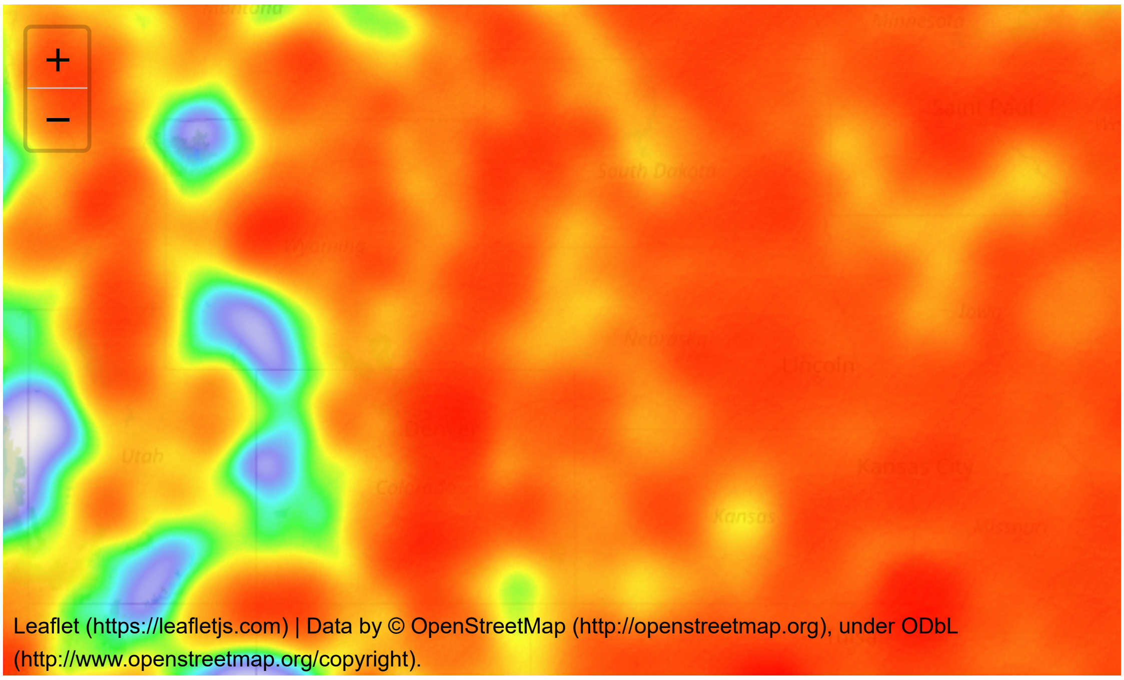
Out[158]:

	slat	slon	loss_norm
0	38.77000	-90.22000	0.00214
1	39.10000	-89.30000	0.00179
2	40.88000	-84.58000	0.00143
3	34.40000	-94.37000	0.00107
4	37.60000	-90.68000	0.00179

Plotting geographical heatmap using folium

In [159]:
```python
mapObj = folium.Map(location=[df_tornado_l["slat"].mean(), df_tornado_l["slon"].mean()], zoom_start=4.7)

HeatMap(df_tornado_l).add_to(mapObj)
mapObj
```

Out[159]:



Visualizing Overall Crop Loss in USA

In [160]:
```python
df_tornado_cl = df[['slat', 'slon', 'closs']]
```

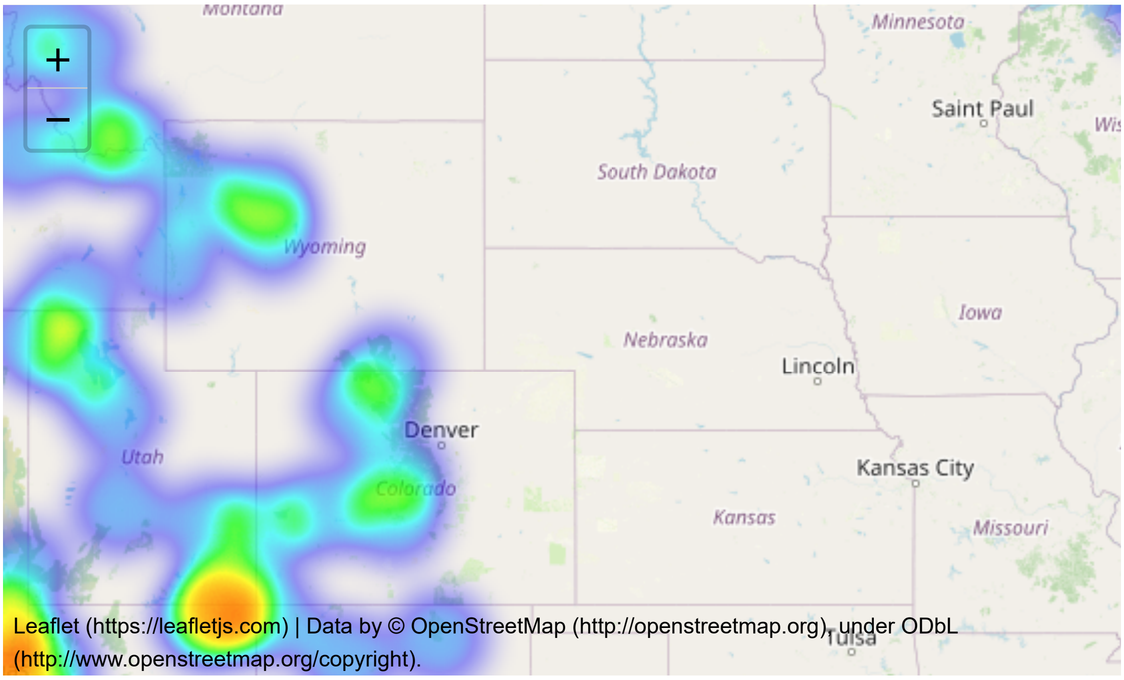
In [161]:
```python
# Re-scale Loss column for satisfying folium requirement

a = df_tornado_cl["closs"]
norm_a = (a - a.min())/(a.max() - a.min())
df_tornado_cl["closs_norm"] = norm_a
df_tornado_cl.drop('closs', axis=1, inplace=True)
df_tornado_cl.head()
```

Out[161]:

	slat	slon	closs_norm
0	38.77000	-90.22000	0.00000
1	39.10000	-89.30000	0.00000
2	40.88000	-84.58000	0.00000
3	34.40000	-94.37000	0.00000
4	37.60000	-90.68000	0.00000

In [162]:
```
# Plotting geographical heatmap using folium
mapObj = folium.Map(location=[df_tornado_cl["slat"].mean(), df_tornado_cl
["slon"].mean()], zoom_start=4.7)

HeatMap(df_tornado_cl).add_to(mapObj)
mapObj
```

Out[162]:



Visualizing only March and April month data in 2011

In [163]:
```
mask = ((df["yr"] == 2011) & (df["mo"] >= 1) & (df["mo"] <= 12))
df_2011_mar_apr = df.loc[mask]
```

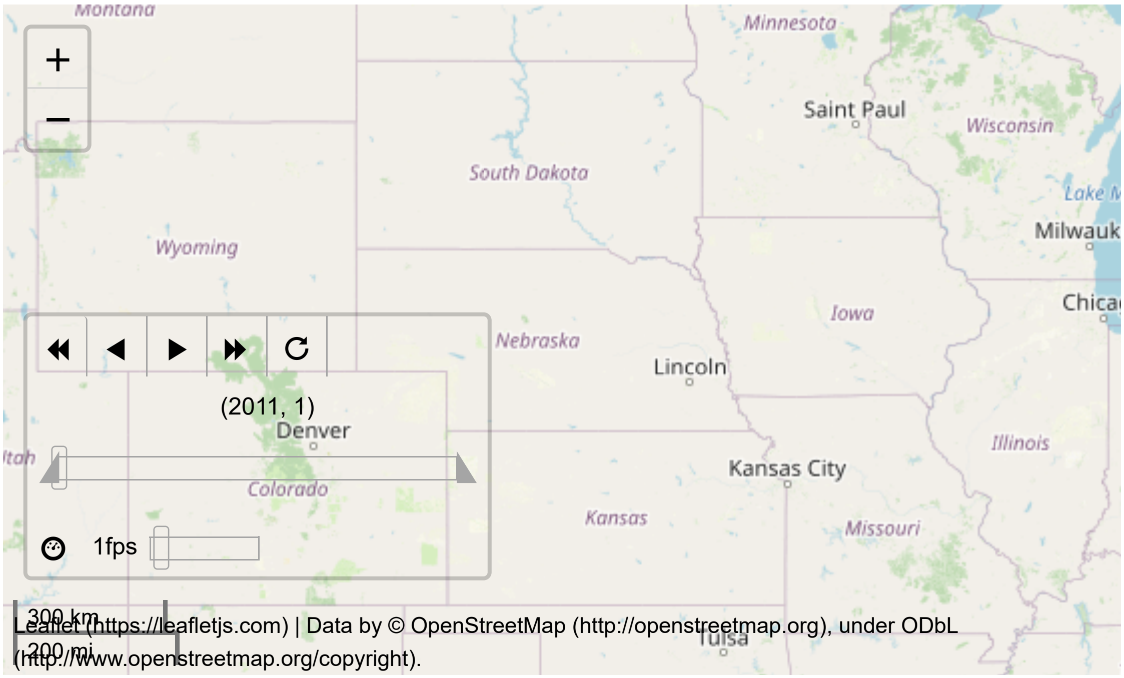
In [164]:
```
map = folium.Map(location=[df_2011_mar_apr["slat"].mean(), df_2011_mar_ap
r["slon"].mean()], zoom_start=5, control_scale=True)
```

In [165]:
```
# Loss
# The data is grouped by date

lat_long_list = []
group_index = []
for index, group in df_2011_mar_apr.groupby(["yr", "mo"]):
    temp = []
    group_index.append(str(index))
    for lat, long, frp in zip(group["slat"],group["slon"], group["los
s"]):
        temp.append([lat, long, frp])
    lat_long_list.append(temp)
```

Visualizing tornado affected regions in year 2011

In [166]:
```python
# HeatMap with time

timeslider = plugins.HeatMapWithTime(lat_long_list, index=group_index).ad
d_to(map)
map
```

Out[166]:

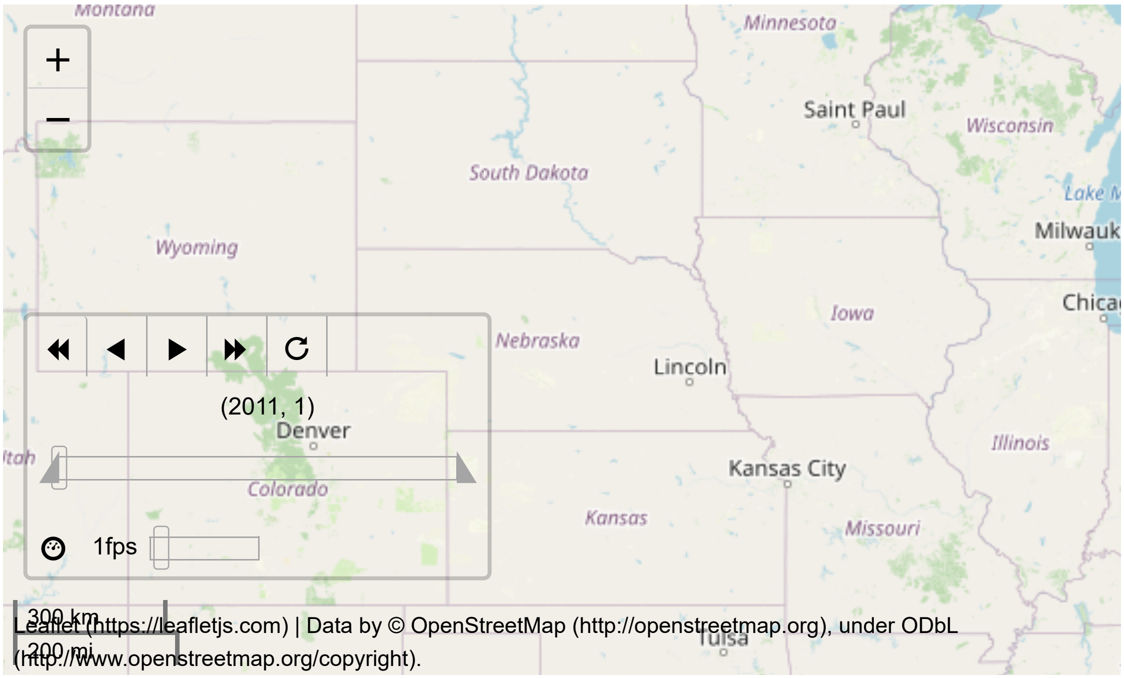


In [167]:
```python
map_closs = folium.Map(location=[df_2011_mar_apr["slat"].mean(), df_2011_
mar_apr["slon"].mean()], zoom_start=5, control_scale=True)
```

In [168]:
```python
# Crop Loss
# The data is grouped by date

lat_long_list = []
group_index = []
for index, group in df_2011_mar_apr.groupby(["yr", "mo"]):
    temp = []
    group_index.append(str(index))
    for lat, long, frp in zip(group["slat"],group["slon"], group["clos
s"]):
        temp.append([lat, long, frp])
    lat_long_list.append(temp)
```

In [169]:
```
# HeatMap with time

timeslider = plugins.HeatMapWithTime(lat_long_list, index=group_index).ad
d_to(map_closs)
map_closs
```

Out[169]:



Defining target and feature vectors

In [170]:
```
y_loss = df['loss']
X_loss = df.drop(['loss', 'time','st'], axis = 1)
```

In [171]:
```
X_loss.head()
```

Out[171]:

	yr	mo	dy	mag	inj	fat	closs	slat	slon	elat	elon	len
0	1950	1	3	3	3	0	0.00000	38.77000	-90.22000	38.83000	-90.03000	9.50000
1	1950	1	3	3	3	0	0.00000	39.10000	-89.30000	39.12000	-89.23000	3.60000
2	1950	1	3	1	1	0	0.00000	40.88000	-84.58000	40.88000	-84.58000	0.10000
3	1950	1	13	3	1	1	0.00000	34.40000	-94.37000	34.40000	-94.37000	0.60000
4	1950	1	25	2	5	0	0.00000	37.60000	-90.68000	37.63000	-90.65000	2.30000

Use a feature selection method to select the features to build a model.

In [172]:
```
# Use select K best method to get the best features to predict the output variable
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_regression


selector = SelectKBest(mutual_info_regression, k=7)
selector.fit_transform(X_loss, y_loss)
# Get columns to keep and create new dataframe with those only
cols = selector.get_support(indices=True)
X_loss = X_loss.iloc[:,cols]
X_loss.head()
```

Out[172]:

	yr	mag	inj	slon	elon	len	wid
0	1950	3	3	-90.22000	-90.03000	9.50000	150.00000
1	1950	3	3	-89.30000	-89.23000	3.60000	130.00000
2	1950	1	1	-84.58000	-84.58000	0.10000	10.00000
3	1950	3	1	-94.37000	-94.37000	0.60000	17.00000
4	1950	2	5	-90.68000	-90.65000	2.30000	300.00000

Baseline model to predict Loss

Using linear regression as the baseline model to predict loss

In [173]:
```
# here we are split the data into training (80%) and testing (20%) sets.
X_loss = X_loss.astype("float64")
train_X_loss, test_X_loss, train_y_loss, test_y_loss = train_test_split(X_loss, y_loss, test_size = 0.2, random_state = 123)
```

In [174]:
```
from sklearn.linear_model import LinearRegression
```

In [175]:
```
# Linear Regression Model
model = LinearRegression()
model.fit(train_X_loss, train_y_loss)
```

Out[175]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                 normalize='deprecated', positive=False)
```

In [176]:
```
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import mean_absolute_error as MAE
from sklearn.metrics import r2_score as R2
```

In [177]:
```
# Predict the model
pred = model.predict(test_X_loss)
# MAE, RMSE, R2 Computation
mae = MAE(test_y_loss, pred)
rmse = np.sqrt(MSE(test_y_loss, pred))
rsquare = R2(test_y_loss, pred)
print("MAE : % f" %(mae))
print("RMSE : % f" %(rmse))
print("R-Square : % f" %(rsquare))
```

```
MAE :  2.668155
RMSE :  23.524695
R-Square : -0.242527
```

Final Candidate Model to predict Loss

In [178]:
```
import xgboost as xg
from tqdm import tqdm
```

XGB

Initialy we have choose random hyperparameter values for the XGB regressor n_estimators is the max number of weak learners max_depth is the Xgb tree depth / level eta is the learning rate

In [179]:
```
xgb_r = xg.XGBRegressor(n_estimators = 35, seed = 123, max_depth=10, eta=
0.25)
```

In [180]:
```
# model training
eval_set=[(test_X_loss, test_y_loss)]
xgb_r.fit(train_X_loss, train_y_loss, eval_set=eval_set)
```

```
[0]	validation_0-rmse:19.54018
[1]	validation_0-rmse:18.45669
[2]	validation_0-rmse:18.61738
[3]	validation_0-rmse:18.94251
[4]	validation_0-rmse:19.29595
[5]	validation_0-rmse:19.55626
[6]	validation_0-rmse:19.80719
[7]	validation_0-rmse:19.66839
[8]	validation_0-rmse:19.90871
[9]	validation_0-rmse:19.83474
[10]	validation_0-rmse:19.80601
[11]	validation_0-rmse:19.95830
[12]	validation_0-rmse:20.06718
[13]	validation_0-rmse:20.19701
[14]	validation_0-rmse:20.23221
[15]	validation_0-rmse:20.25854
[16]	validation_0-rmse:20.28433
[17]	validation_0-rmse:20.32226
[18]	validation_0-rmse:20.35603
[19]	validation_0-rmse:20.39904
[20]	validation_0-rmse:20.44260
[21]	validation_0-rmse:20.45726
[22]	validation_0-rmse:20.47119
[23]	validation_0-rmse:20.48580
[24]	validation_0-rmse:20.51024
[25]	validation_0-rmse:20.51797
[26]	validation_0-rmse:20.51994
[27]	validation_0-rmse:20.52788
[28]	validation_0-rmse:20.53511
[29]	validation_0-rmse:20.54539
[30]	validation_0-rmse:20.54901
[31]	validation_0-rmse:20.55323
[32]	validation_0-rmse:20.55356
[33]	validation_0-rmse:20.55633
[34]	validation_0-rmse:20.56024
```

Out[180]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=
False,
             eta=0.25, gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.25, max_delta_s
tep=0,
             max_depth=10, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=35, n_jobs=16,
             num_parallel_tree=1, objective='reg:squarederror',
             predictor='auto', random_state=123, reg_alpha=0, reg_lambda
=1,
             scale_pos_weight=1, seed=123, subsample=1, tree_method='exa
ct',
             validate_parameters=1, ...)
```

```
In [181]: pred = xgb_r.predict(test_X_loss)
          # MAE, RMSE, RSquare Computation
          mae = MAE(test_y_loss, pred)
          rmse = np.sqrt(MSE(test_y_loss, pred))
          r2 = R2(test_y_loss, pred)
          print("MAE : % f" %(mae))
          print("RMSE: % f" %(rmse))
          print("R2 : % f" %(r2))
```

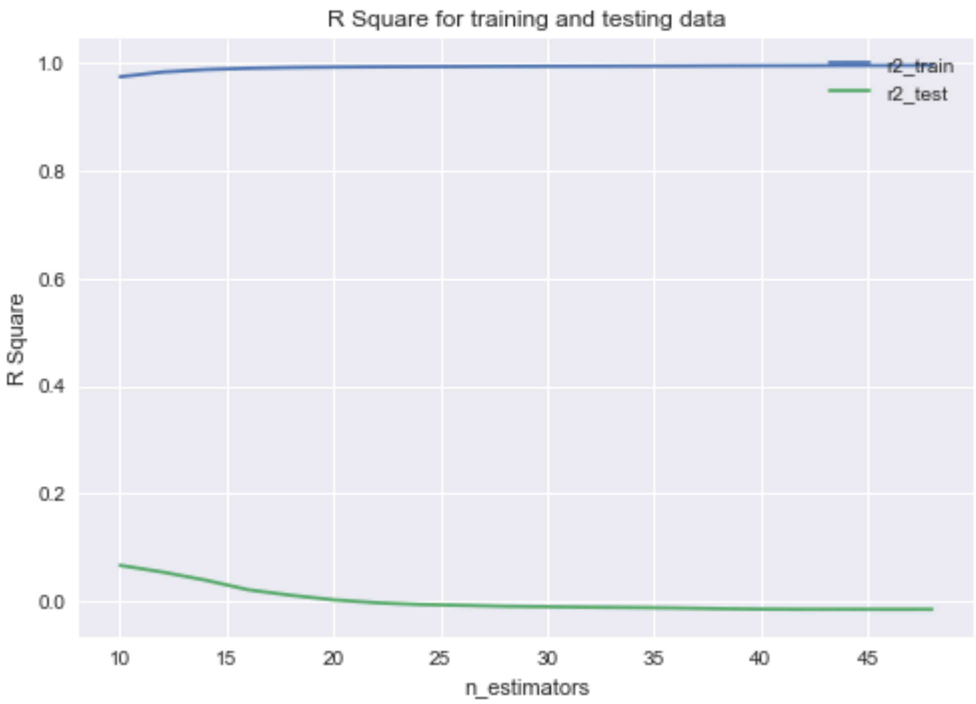
```
MAE :  1.530191
RMSE:  20.560246
R2 :  0.050895
```

Performing hyperparameter tuning to find the estimators

```
In [182]: # Training on XGB Regressor
          def xgb_train_model(n):
              model = xg.XGBRegressor(objective ='reg:linear',n_estimators = n, see
          d = 123, max_depth=10, verbosity = 0)
              model.fit(train_X_loss, train_y_loss)
              return model

          # Prediction model
          def xgb_prediction(model, X):
              pred = model.predict(X)
              return pred


          r2_train = []
          r2_test = []

          for i in tqdm(range(10, 50, 2)):
              model = xgb_train_model(i)
              train_pred = xgb_prediction(model, train_X_loss)
              test_pred = xgb_prediction(model, test_X_loss)
              r2_train.append(R2(train_y_loss, train_pred))
              r2_test.append(R2(test_y_loss, test_pred))
```

```
100%|
| 20/20 [00:07<00:00,  2.59it/s]
```

In [183]:
```
plt.plot(range(10, 50, 2), r2_train, label='r2_train')
plt.plot(range(10, 50, 2), r2_test, label='r2_test')
plt.legend(loc='upper right')
plt.xlabel("n_estimators")
plt.ylabel("R Square")
plt.title("R Square for training and testing data")
plt.show()
```



In [184]:
```
xgb_model = xg.XGBRegressor(objective ='reg:linear',n_estimators = 20, se
ed = 123, max_depth=10, verbosity = 0)
xgb_r.fit(train_X_loss, train_y_loss)

xgb_final_prediction = xgb_r.predict(test_X_loss)

xgb_rsquare = R2(test_y_loss, xgb_final_prediction)

print("R-Square for final XGB regressor model: % f" %(xgb_rsquare))
```

```
R-Square for final XGB regressor model:  0.050895
```

References:

[1] developer, “EF0 Tornado,” Facts Just for Parents, Teachers and Students, Jul. 19, 2021. https://www.factsjustforkids.com/weather-facts/tornado-facts-for-kids/enhanced-fujita-scale/ef0-tornado/ (https://www.factsjustforkids.com/weather-facts/tornado-facts-for-kids/enhanced-fujita-scale/ef0-tornado/) (accessed Jul. 07, 2022).

[2] A.Mahale, H.Lakhani, "Assignment 1." Dalhousie University, [Online], 2022. [Accessed 05-Jul-2022]

[3] “Storm Prediction Center Severe Weather GIS (SVRGIS) Page,” www.spc.noaa.gov. http://www.spc.noaa.gov/gis/svrgis/ (http://www.spc.noaa.gov/gis/svrgis/) (accessed Jul. 02, 2022).

[4] “sklearn.linear_model.LinearRegression — scikit-learn 0.22 documentation,” Scikit-learn.org, Feb. 01, 2019. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html) (accessed Jul. 10, 2022).

[5] “sklearn.ensemble.BaggingRegressor — scikit-learn 0.23.2 documentation,” scikit-learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html) (accessed Jul. 05, 2022).

[6] “3.3. Metrics and scoring: quantifying the quality of predictions — scikit-learn 0.22.1 documentation,” scikit-learn.org. https://scikit-learn.org/stable/modules/model_evaluation.html (https://scikit-learn.org/stable/modules/model_evaluation.html) (accessed Jul. 10, 2022).

[7] “seaborn.boxenplot — seaborn 0.11.2 documentation,” seaborn.pydata.org. https://seaborn.pydata.org/generated/seaborn.boxenplot.html (https://seaborn.pydata.org/generated/seaborn.boxenplot.html) (accessed Jul. 09, 2022).

[8] P. Org, "SGD — PyTorch 1.12 documentation", Pytorch.org, 2022. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.optim.SGD.html (https://pytorch.org/docs/stable/generated/torch.optim.SGD.html). [Accessed: 03- Aug- 2022]

[9] scikit-learn. developers, "3.3. Metrics and scoring: quantifying the quality of predictions", scikit-learn, 2022. [Online]. Available: https://scikit-learn.org/stable/modules/model_evaluation.html (https://scikit-learn.org/stable/modules/model_evaluation.html). [Accessed: 03- Aug- 2022]

[10] J. Brownlee, "Regression Metrics for Machine Learning", Machine Learning Mastery, 2022. [Online]. Available: https://machinelearningmastery.com/regression-metrics-for-machine-learning/ (https://machinelearningmastery.com/regression-metrics-for-machine-learning/). [Accessed: 03- Aug- 2022]

[11] “Guide to Prevent Overfitting in Neural Networks,” Analytics Vidhya, Jun. 12, 2021. https://www.analyticsvidhya.com/blog/2021/06/complete-guide-to-prevent-overfitting-in-neural-networks-part-1/ (https://www.analyticsvidhya.com/blog/2021/06/complete-guide-to-prevent-overfitting-in-neural-networks-part-1/) (accessed Aug. 03, 2022).

[12] s. developers, "Plotting Cross-Validated Predictions", scikit-learn, 2022. [Online]. Available: https://scikit-learn.org/stable/auto_examples/model_selection/plot_cv_predict.html#sphx-glr-auto-examples-model-selection-plot-cv-predict-py (https://scikit-learn.org/stable/auto_examples/model_selection/plot_cv_predict.html#sphx-glr-auto-examples-model-selection-plot-cv-predict-py). [Accessed: 03- Aug- 2022]

[13] s. developers, "API Reference", scikit-learn, 2022. [Online]. Available: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics). [Accessed: 03- Aug- 2022]

[14] p. org, "PyTorch", Pytorch.org, 2022. [Online]. Available: https://pytorch.org/get-started/locally/ (https://pytorch.org/get-started/locally/). [Accessed: 03- Aug- 2022]

[15] P. Mortensen, "How can I group by month from a date field using Python and Pandas?", Stack Overflow, 2022. [Online]. Available: https://stackoverflow.com/questions/44908383/how-can-i-group-by-month-from-a-date-field-using-python-and-pandas (https://stackoverflow.com/questions/44908383/how-can-i-group-by-month-from-a-date-field-using-python-and-pandas). [Accessed: 03- Aug- 2022]

[16] p. org, "PyTorch documentation — PyTorch 1.12 documentation", Pytorch.org, 2022. [Online]. Available: https://pytorch.org/docs/stable/index.html (https://pytorch.org/docs/stable/index.html). [Accessed: 03- Aug- 2022]

[17] s. developers, "sklearn.ensemble.RandomForestClassifier", scikit-learn, 2022. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html). [Accessed: 03- Aug- 2022]

[18] N. Focus, "pandas - Python Data Analysis Library", Pandas.pydata.org, 2022. [Online]. Available: https://pandas.pydata.org/ (https://pandas.pydata.org/). [Accessed: 03- Aug- 2022]

[19] N. Developers, "NumPy: the absolute basics for beginners — NumPy v1.23 Manual", Numpy.org, 2022. [Online]. Available: https://numpy.org/doc/stable/user/absolute_beginners.html (https://numpy.org/doc/stable/user/absolute_beginners.html). [Accessed: 03- Aug- 2022]

[20] P. Org, "random — Generate pseudo-random numbers — Python 3.10.6 documentation", Docs.python.org, 2022. [Online]. Available: https://docs.python.org/3/library/random.html (https://docs.python.org/3/library/random.html). [Accessed: 03- Aug- 2022]

[21] M. Waksom, "seaborn: statistical data visualization — seaborn 0.11.2 documentation", Seaborn.pydata.org, 2022. [Online]. Available: https://seaborn.pydata.org/ (https://seaborn.pydata.org/). [Accessed: 03- Aug- 2022]

[22] R. Story, "Quickstart — Folium 0.12.1 documentation", Python-visualization.github.io, 2022. [Online]. Available: https://python-visualization.github.io/folium/quickstart.html (https://python-visualization.github.io/folium/quickstart.html). [Accessed: 03- Aug- 2022]

[23] S. Developers, "1.13. Feature selection", scikit-learn, 2022. [Online]. Available: https://scikit-learn.org/stable/modules/feature_selection.html (https://scikit-learn.org/stable/modules/feature_selection.html). [Accessed: 03- Aug- 2022]

[24] S. Developers, "sklearn.model_selection.train_test_split", scikit-learn, 2022. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html). [Accessed: 03- Aug- 2022]

[25] S. Developers, "sklearn.ensemble.RandomForestRegressor", scikit-learn, 2022. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html). [Accessed: 03- Aug- 2022]