Wildfire Prediction

Use Machine Learning to Predict Wildfire in Particular Area

Taniya Adhikari

March 1, 2022

1 INTRODUCTION

1.1 Objective

Identify the most important factors that cause wildfires and build a model to predict wildfire events accurately using MODIS fire products data, historical weather, and soil quality data.

1.2 Background

Every year, wildfires destroy massive number of homes and acres of lands and properties around the world. Wildfires are getting dangerously longer and severer each year. Wildfires have burned an average of 7 million acres yearly since 2000 in the U.S. Climate change has been a main factor in increasing wildfires in U.S. There are various reports from United Nations on rising temperatures across the globe are contributing to wildfires by increasing drought, decreasing precipitation, soil moisture, and the presence of trees and other potential fuel. By exploring these factors that cause wildfires, allows experts to take early action to minimize the risks and threats it poses. Though, climate change is the key factor, human related factors such as debris burning, campfire and arson is also a contributing to rising wildfires. It is estimated 80% of fires are caused by combination of these factors.

Wildfires are getting uncontrollable over the years, cause many fatalities and it is costly to contain it. In 2021, Dixie fire was one of the deadliest wildfires that happened in California area with 463,000 acres of land burned, which destroyed hundreds of buildings and threatened many lives. In 2020, Bay Area fire killed 35 people and burned almost 1 million acres of land. The history of wildfires goes all the way back to 1871 when Peshtigo Fire in Wisconsin burned 1.2 million acres and killed 1152 people.

In this project, I will be exploring MODIS Collection 6 Active Fire Data provided by NASA. MODIS fire products identify thermal anomalies, such as volcanoes or vegetation fires. It can identify 50% of fires accurately by capturing relevant fire pixels and its' accuracy increases with the size of fire. Thus, fire pixels are not necessarily true fires. Along with this data I will also be using historical daily climate summaries, other meteorological indicators also provided by NASA LaRC. Lastly, I will also use Soil quality data. My goal is to create a model that can be used to do early fire detection using combination of variables from these datasets and detect true fires. For this project, I will be narrowing the research to just California State fire data.

1.3 Business Problem

Firefighting resource is crucial in containing wildfires and should be allocated correctly when needed. MODIS historical fire pixel data contains useful spatial information that can be used to identify patterns over time. Meteorological indicators can also provide information regarding weather conditions that causes wildfire. Soil quality indicators give information regarding drought possibilities. Though, MODIS can identify large fires with higher accuracy, it does have too many false alarms. It is expensive to use resources for false alarms. Thus, accurate prediction of fire is crucial to minimize the cost and for early preparation. For this project, I would like to answer following questions: **Do weather and soil conditions has any significance in wildfires? Can I identify true fires more accurately by combining weather and soil data to fire pixels data? Can I identify the fire prone area before fire happens, using historical fire pixels data combined with weather indicators and soil quality indicator?**

2 METHODS

2.1 Data Collection and Data Understanding

For this project, there was not any single existing dataset that had all the factors combined. I will be explaining the process of collecting different input variables for the model. I used multiple datasets to create a single dataset with as many factors as needed for this project. Following are the list of data:

California Fire Geo Data

To predict fire events, I had to map California fires correctly with weather, fire pixels and soil data for the given date. Furthermore, I must make sure that start dates of all fires are as correct as possible. I used three different resources to cross-check start dates of the known fire events in California. Below is the list of datasets:

- California Fire Perimeters, shape files: https://gis.data.ca.gov/
- Wildland Fire Open data: https://data-nifc.opendata.arcgis.com/
- Cal Fire open data: https://www.fire.ca.gov/

Furthermore, I used geopandas library to merge Wildland Fire data (Point geometry) and Cal Fire data (Point geometry) to California Fire Perimeters (Polygon geometry) by finding nearest point to polygon. I used burned areas (in acres) as a threshold to make sure that fire points are not mapped too far*. (See Appendix A.1 for Fire events Data Preparation Code).* I identified 3803 fires that happened in last 10 years (2011-2020). Key variables in this dataset are **Fire Date and Fire location.**

Note: This is the most important step. The most challenging task was to get the fire dates corrected. None of the datasets had the accurate start date information, some fire events had dates that were way off than when it happened, and the start dates among all datasets were not the same for some fire events. To get this part corrected, I had to hard code some dates manually by doing lot of internet research such as finding news articles for that year and month and look for fire events using the "Name" variable. *(See Appendix A.1 for Fire Events Preparation Code).*

MODIS Collection 6 Data

MODIS Collection 6 Data comes as granules in hdf5 files. Though, I initially attempted to get the data by using NASA earth data. However, those files were large, and my device could not process it. Fortunately, Kaggle.com had complete MODIS Data from 2010-2020. Below is the list of MODIS data: *(Appendix A.2 MODIS Data Preparation).*

- Fire Pixel data: https://earthdata.nasa.gov/
- MODIS CSV file: https://www.kaggle.com/

After cleaning the date set, I identified 114,599 fire pixels for California Area between 2011-2020. Key variables for this dataset are:

Brightness T21 and T31	Measure of photons at a wavelength received. measured in Kelvin
FRP	Measures the rate of radiant heat output from a fire.
Scan and Track	1 km fire pixels but MODIS pixels get bigger toward the edge of scan. Scan and track are used to measure actual size of pixel
Fire Pixel Latitude	Centre of 1 km fire pixels but not the actual location.
Date	Date to accurately map it with fire events dataset.
Confidence	It ranges between 0 to 100%. It intends to help user to understand the quality of fire pixels. Whether they are high risk or low risk. They can be binned in to 3 classes: low, nominal or high
Day or Night:	Time of the day when the pixel was taken
Hot Spot Type	Area type: Vegetation, Volcanoes, Other Static Land Source and Offshore
Month	The fire event month

Soil and Meteorological Data

Soil quality data and Meteorological data was collected from Kaggle.com as well. Soil and Meteorological datasets were big files, thus required a separate code. Altogether, there were 11,353,524 data points with 52 variables. After cleaning, combining and filtering for 10 year data I ended up with 277,628 data points with 25 variables. *(Appendix A.3: Soil and Meteorological Data Preparation).*

- Meteorological data: https://power.larc.nasa.gov/ , https://www.kaggle.com/
- Soil Dataset: https://www.fao.org/soils-portal/data-hub/soil-maps-and-databases/harmonized-world-soil-database-v12/en/
- Daily weather summary for California State Summary: https://www.noaa.gov/

This data was cleaned so there was not much to do except filtering the years and remove redundant variables, that had .80 or more correlation. Key variables are **Latitude, Longitude, elevation, slope, soil quality, oxygen availability, rain-fed cultivated land, precipitation, windspeed Humidity and temperature.**

2.2 Exploratory Data Analysis

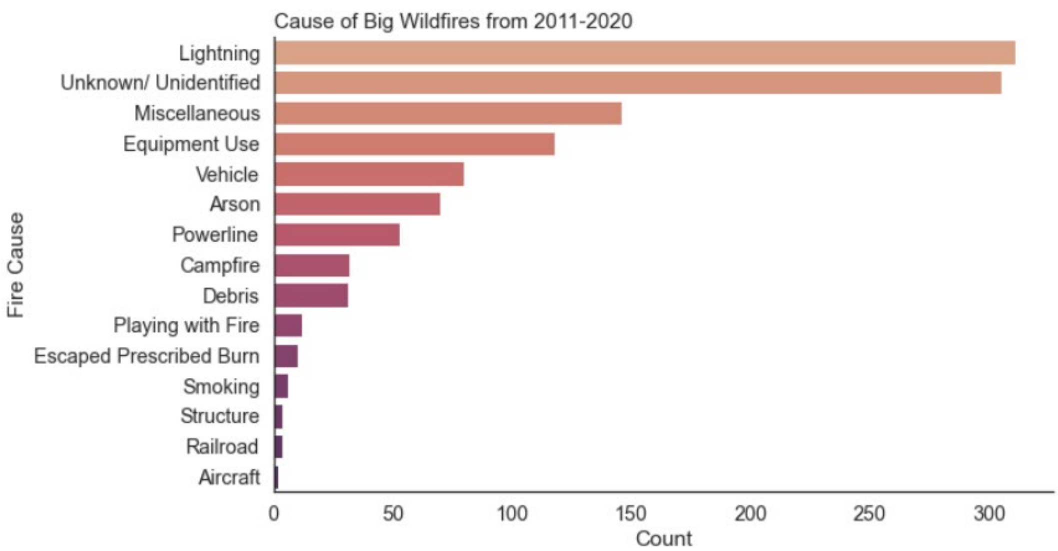
Fire Events Data Analysis

These are the key results I found:

There were **3803 fire events within 10 years of dataset for California** location. These are some of the important things I found when did a preliminary analysis:

Figure 2.2.1 Causes of Big Wildfires Between 2011-2020

Lightning was the number one reason to cause fire events that were bigger than 100 acres. *Note: 100 acres are considered small events and usually do not cause minimal harm.*



Class A	**0 < area ≤ 0.25 Acres**
Class B	**0.26 ≤ area ≤ 9.99**
Class C	**10 ≤ area ≤ 99.9**
Class D	**100 ≤ area ≤ 299**
Class E	**300 ≤ area ≤ 999**
Class F	**1000 ≤ area ≤ 4999**
Class G	**5000 ≤**

Other reasons were Miscellaneous, equipment use, vehicle arson etc., indicating that human related behavior also contributes to big fires.

Table 2.2.1 There are seven classes of fires based on area burned:

Figure 2.2.2 Wildfire Occurrence by Class

Class C wildfire had the highest occurrence during 2011-2020

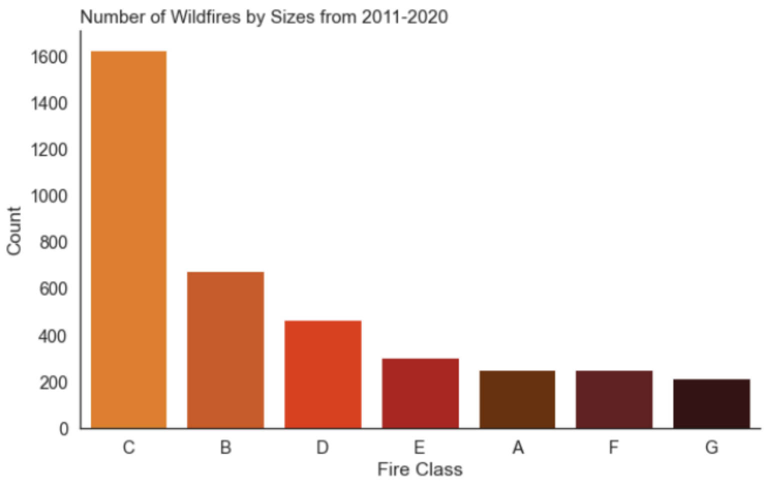


Figure 2.2.3 Top Reasons for Big Wildfires

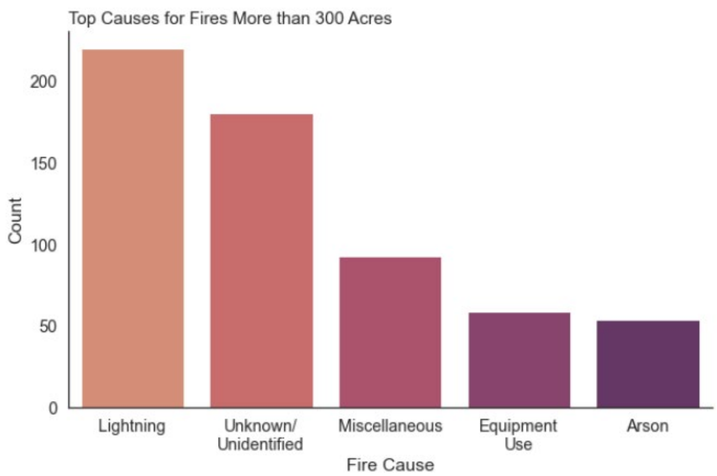


Figure 2.2.4 Total Fires Between 2011-2020

2017 had the highest number of wildfires in past 10 years.

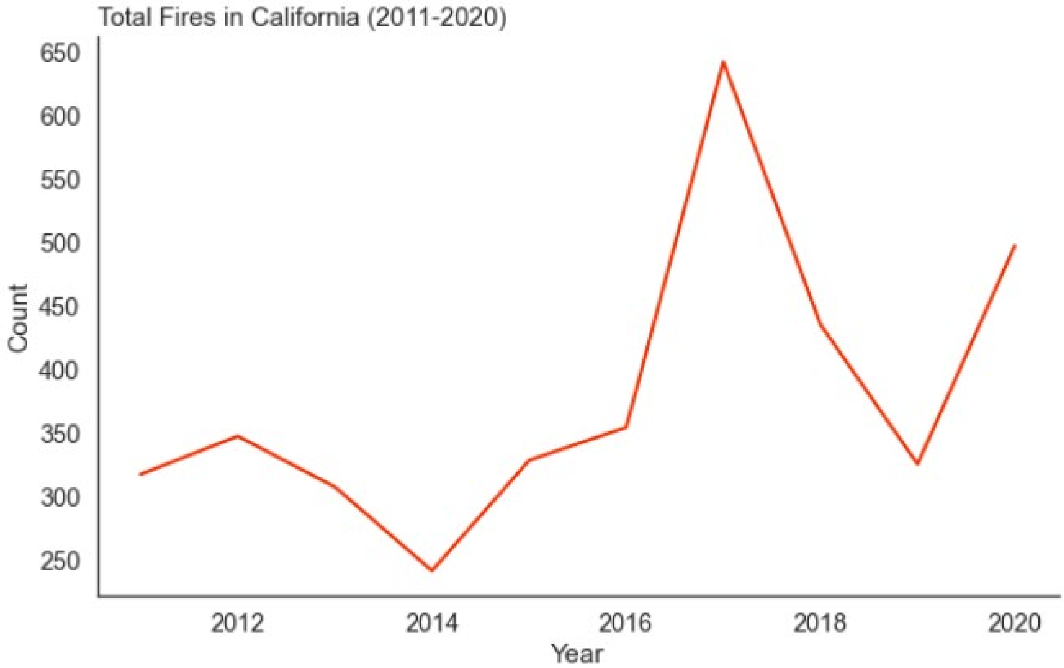


Figure 2.2.5 Wildfires Occurrence by Months

July month had the highest number of wildfires

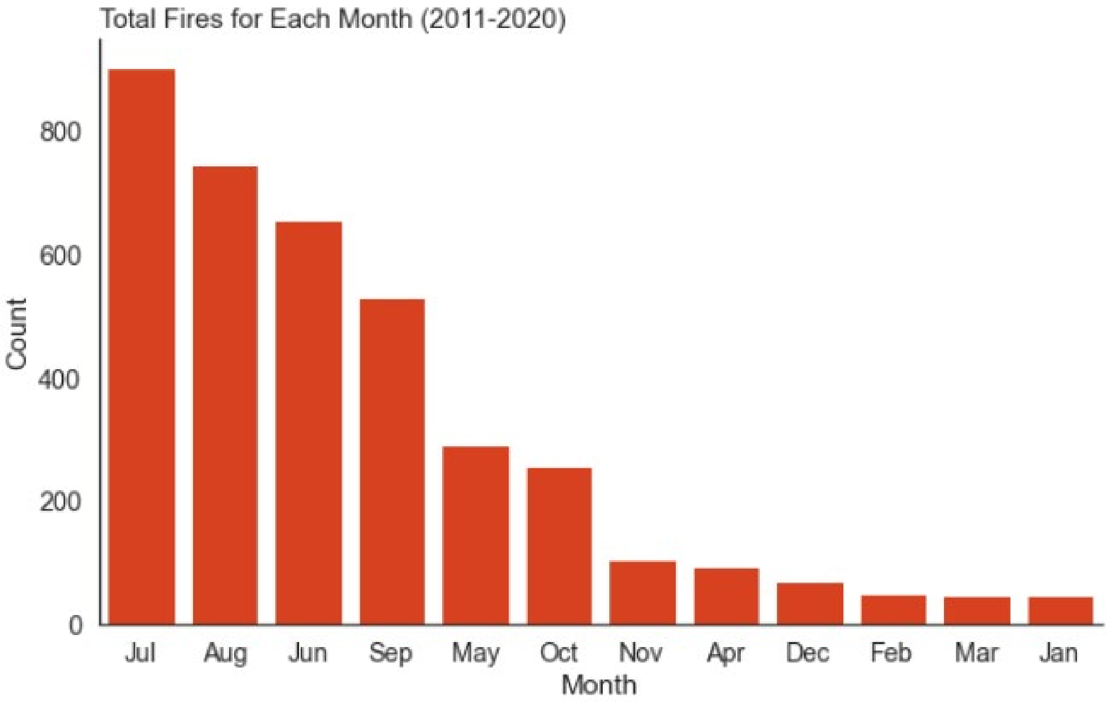
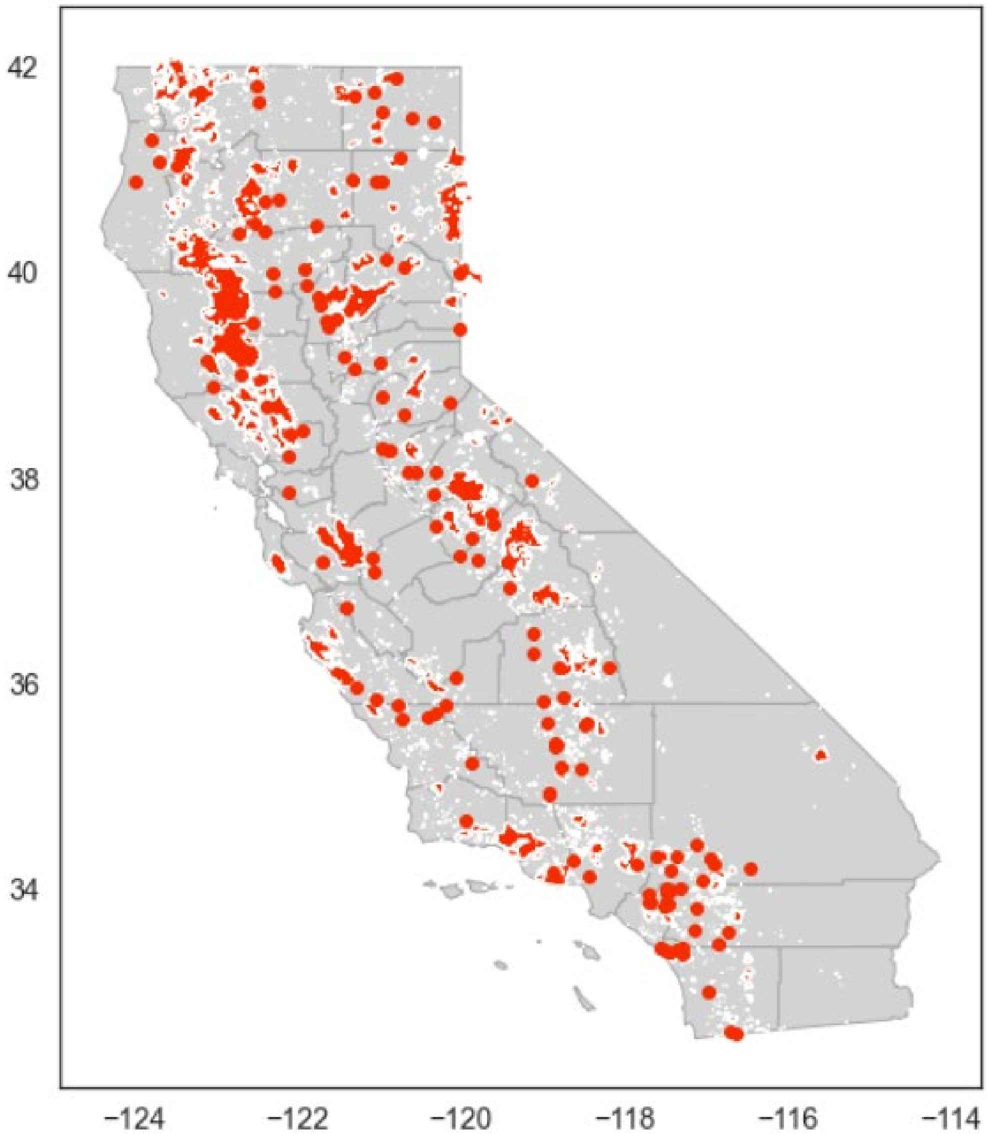


Figure 2.2.6: California Fires by Region.



Modis Collection 6 Data Analysis

These were the following analysis results:

For numerical data, Scan and track had the highest correlation 0f .99, so I took that variable out. Although Brightness T21 and T31 seems to be the same thing, but they have weak relationship, so I kept both variables.

Figure 2.2.7 Distribution of Fire Pixels Attributes

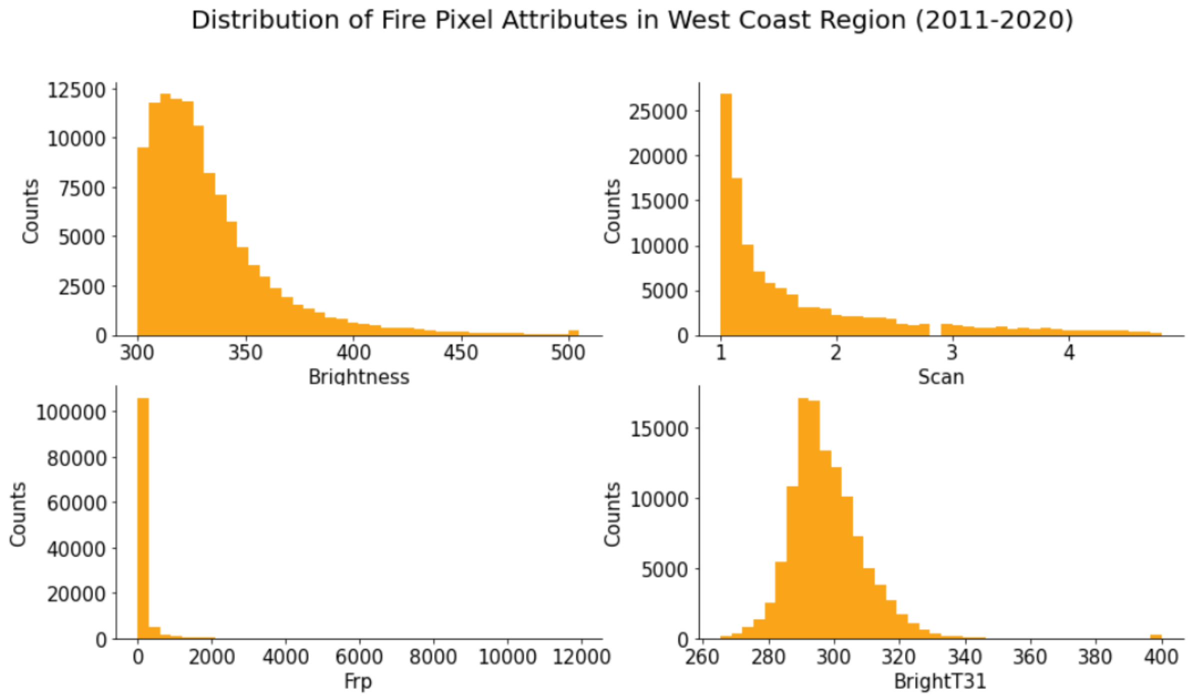


Figure 2.2.8 Fir Pixels Mapped

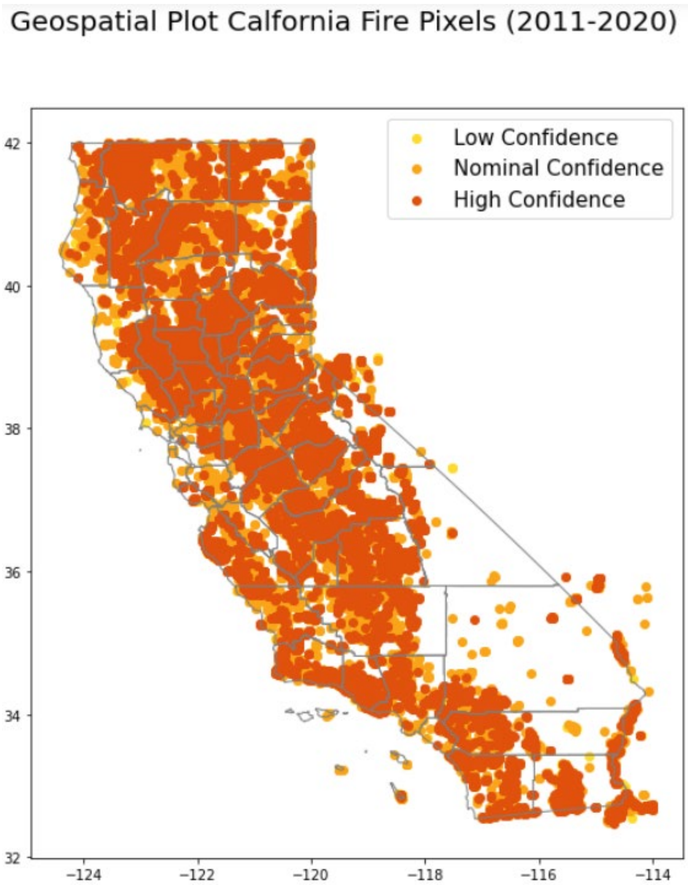
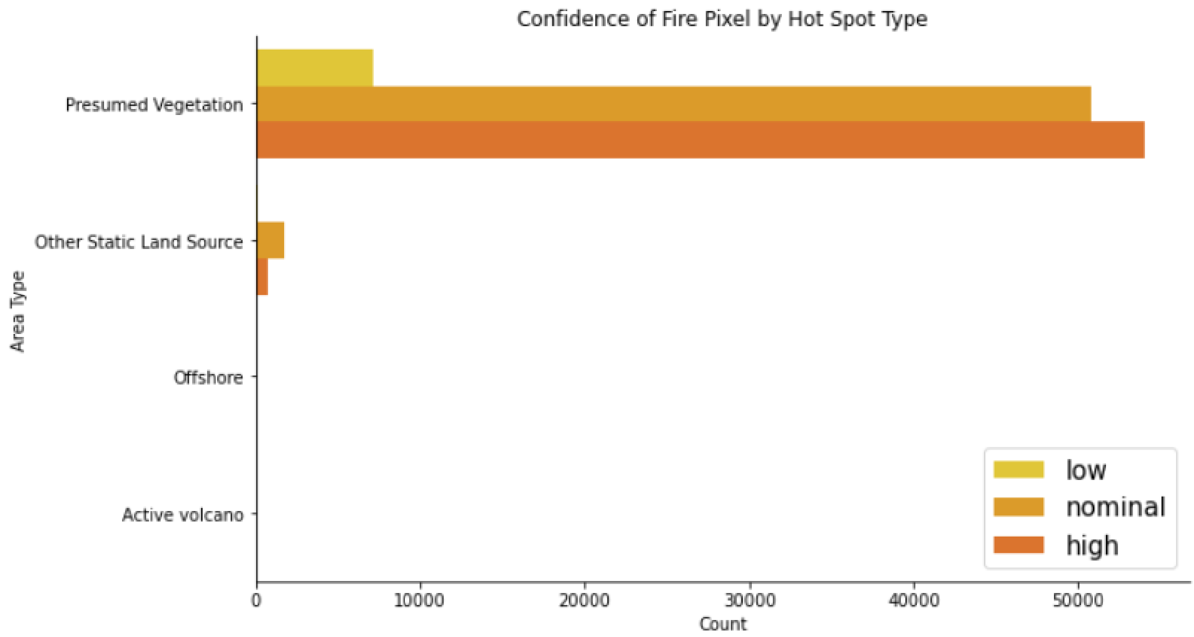
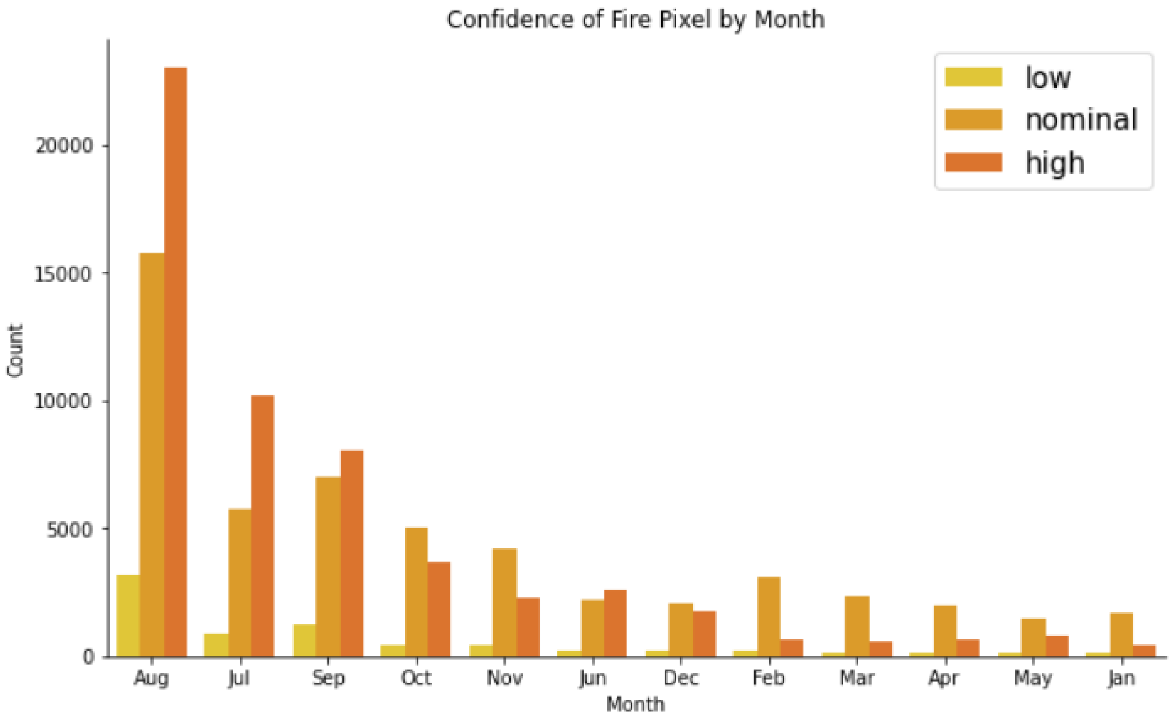


Figure 2.2.9 Fire Pixels by Type of Hot Spot Area

Fire Pixels data indicates Vegetation area is more prone to high-risk wildfires



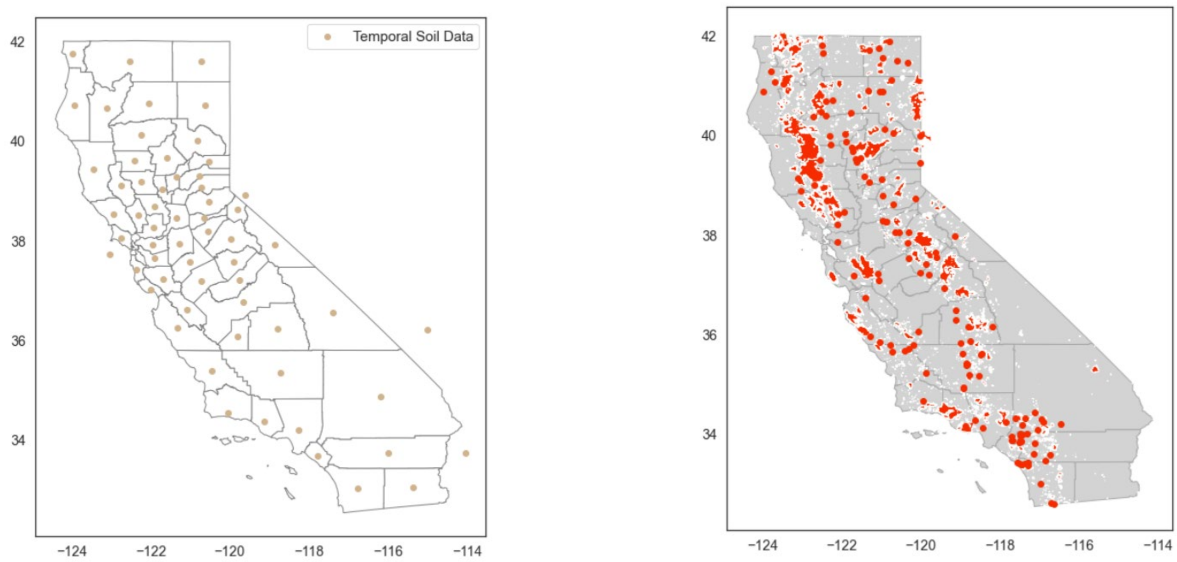
Fire Pixels data shows most fire pixels were taken in August, as opposed to **July fire months in Figure 2.2.5**



Soil and Meteorological Data Analysis

Here the few things I found. For Soil, data it was around the same region where most fire happened.

Figure 2.2.11 Geospatial Plot of Temporal Soil Data



For Meteorological Data. Here are the key findings:

Figure 2.2.12 Comparison of Precipitation, Temperature and Windspeed.

June, July and August is when I have highest temperature and lowest precipitation, in Figure 2.2.6, I found that Most fires occur in June, July and August, with July being the month with highest fire occurance.

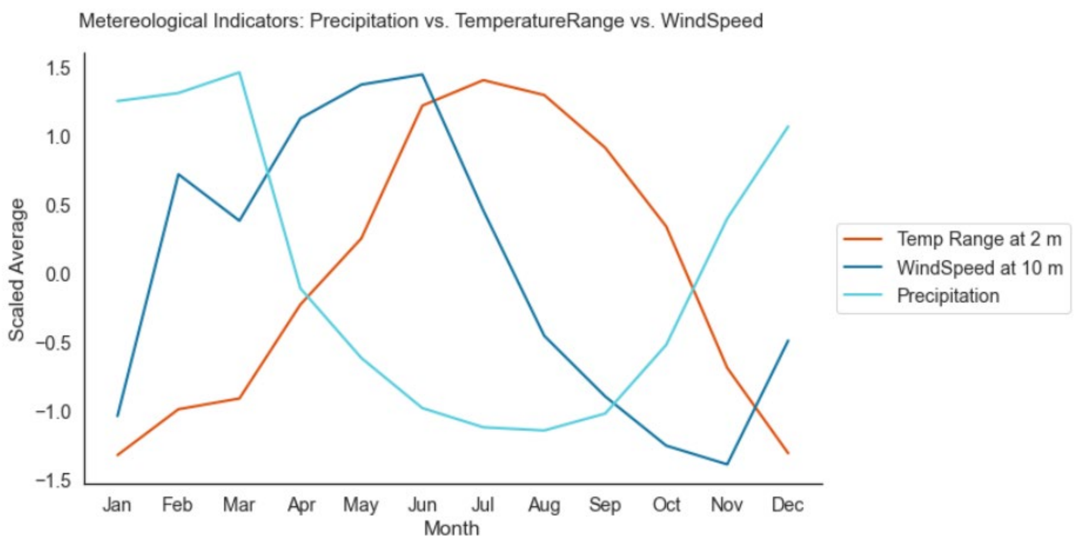
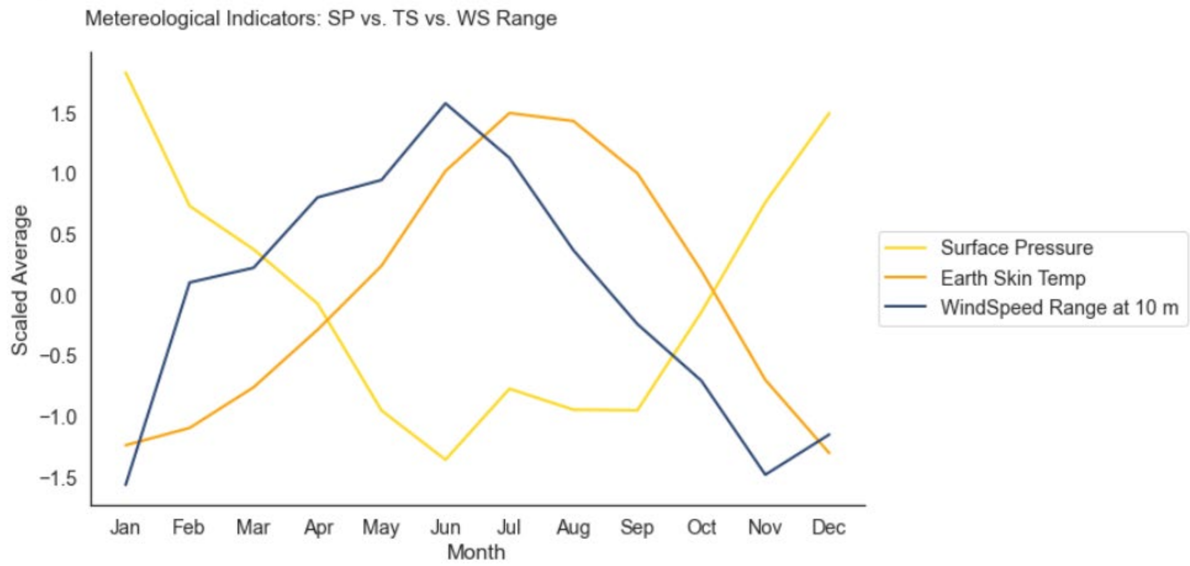
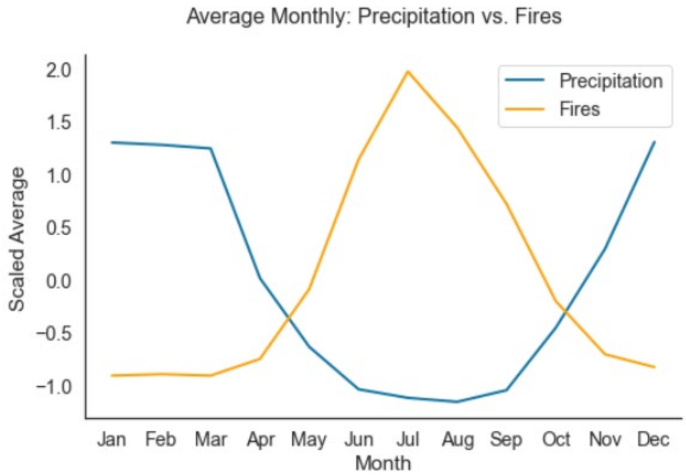
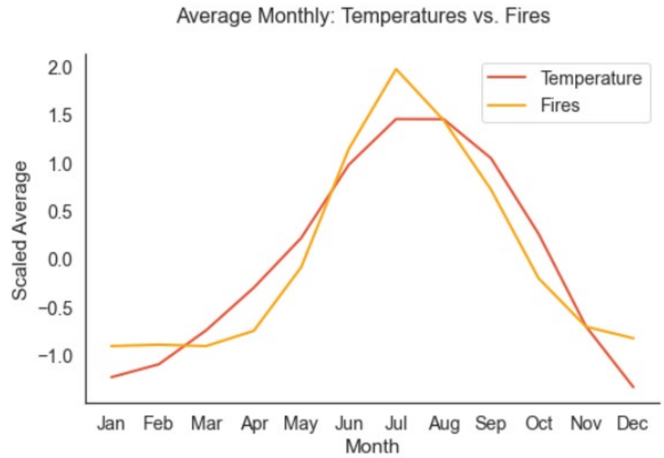


Figure 2.2.13 Comparison of Surface Pressure, Earth Skin Temperature, and

These also shows similar indication as the Figure 2.2.12







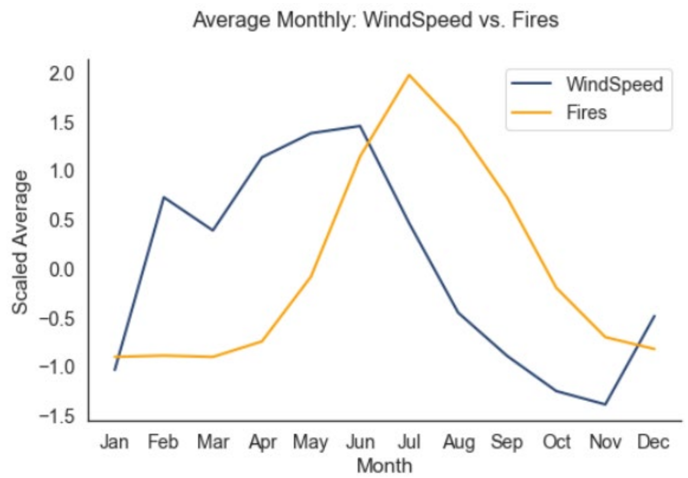
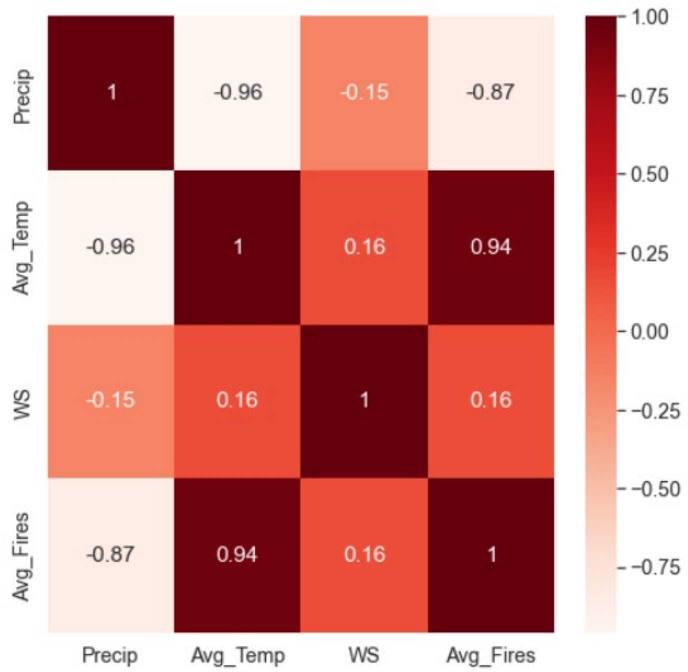


Figure 2.2.15: Heat Map to show correlation



2.3 Data Merging

To prepare data for analysis and predict fire events correctly, I must first make sure that all fire dates are as correct as possible. California Fire Perimeter dataset is a shapefile, which had fire area as polygon with latitude and longitude coordinates of each point. All the above datasets came with coordinates information (latitude and longitude). After cleaning the fires data (handing missing values, fixing dates, identifying unique fires), I narrowed all the data for only California coordinates and filtered datasets for fires only between year 2011-2020. Furthermore, I converted fire pixels, weather, and soil data into geospatial data frame using `Geographic 2D CRS: EPSG:4326`. Then, I converted all of them including fire data into `Projected CRS: EPSG:3310`, so I can merge them by nearest point location (Note: This is crucial to get the minimum distance correctly in m or km, otherwise distance in degree unit holds no meaning. EPSG:3310 converts geographic coordinates into meters).

Steps into Data Preparation

1. I combined daily temperatures data to MODIS pixels data by finding the nearest geo point on the given date of the pixel.

2. I combined soil and meteorological data the same way (all these data were daily data, thus merging them was not as complex).

3. After merging the above datasets, I merged fire data with combined dataset based on fire pixels geo point to nearest fire polygon for the same year and month (Note: I did not merge by exact date, because fire pixels can also be true fire event after it already happened. Fires can go on for weeks and sometimes months depending on the fire size). After merging all the data, I performed few other analyses and made few assumptions to make sure fires are mapped correctly *(See Section 2.7 Assumptions)*.

After merging the data, I found 113912 fire pixels were mapped and 687 was un-mapped. I ended up with total 63 input variables. To find out which fires were accurately mapped, I did separate analysis using assumptions from Section 10.1 Fires that were mapped outside of its burned area distance, were marked as false alarms. All the fires that were mapped correctly was considered labelled data and all the ones that weren't marked correctly were considered as un-labelled data. For Data Merging *See Appendix A.4 Data Merging Code.*

2.4 Data Modeling

Total data points were: 45186 labelled and 57935 un-labelled data and for California. Below is the plot of True and False Fire Pixels from MODIS Data.

Figure 1.7.1 Geospatial analysis of labelled and un-labelled MODIS fire pixels data.

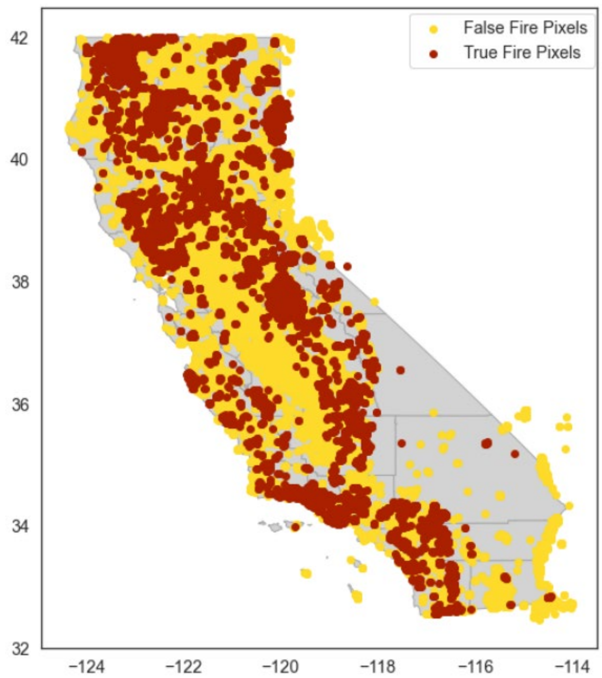
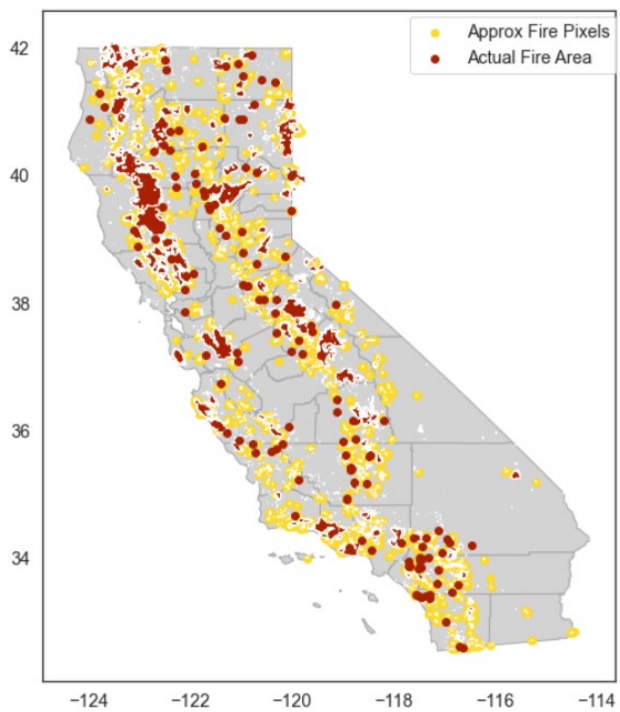


Figure 1.7.2 Geospatial analysis of Actual Fire Area and True Mapped Fire pixels.



Using above visualization, it is safe to assume that that fire pixels are mapped correctly to fire events. Before data modeling step, one last thing I did is check for multi-collinearity since, some variables were duplicated due to data collection from multiple sources. There was total **25 variables.**

Preliminary Data Modeling

For Data modeling, I needed two-part analysis, because I had two things I wanted to answer, can I identify true fire pixels using current data? and can I do early prediction of wildfire using previous days data?

For the first part of the problem statement requires simple classification, I split the data into train and test set using sklearn package and stratified the data, so I don't have unbalanced classes. 70% of data were in train set and 30 % in test. For the binary classification, I chose two algorithms: **Random Forest Classifier and Support Vector Machines with RBF kernel (Non-Linear data model).** For SVM I also did one-hot-coding for categorical data and scaled input variables. *(See Appendix A.5 Data Modeling Part 1)***.** I also did cross validation to check for model accuracy.

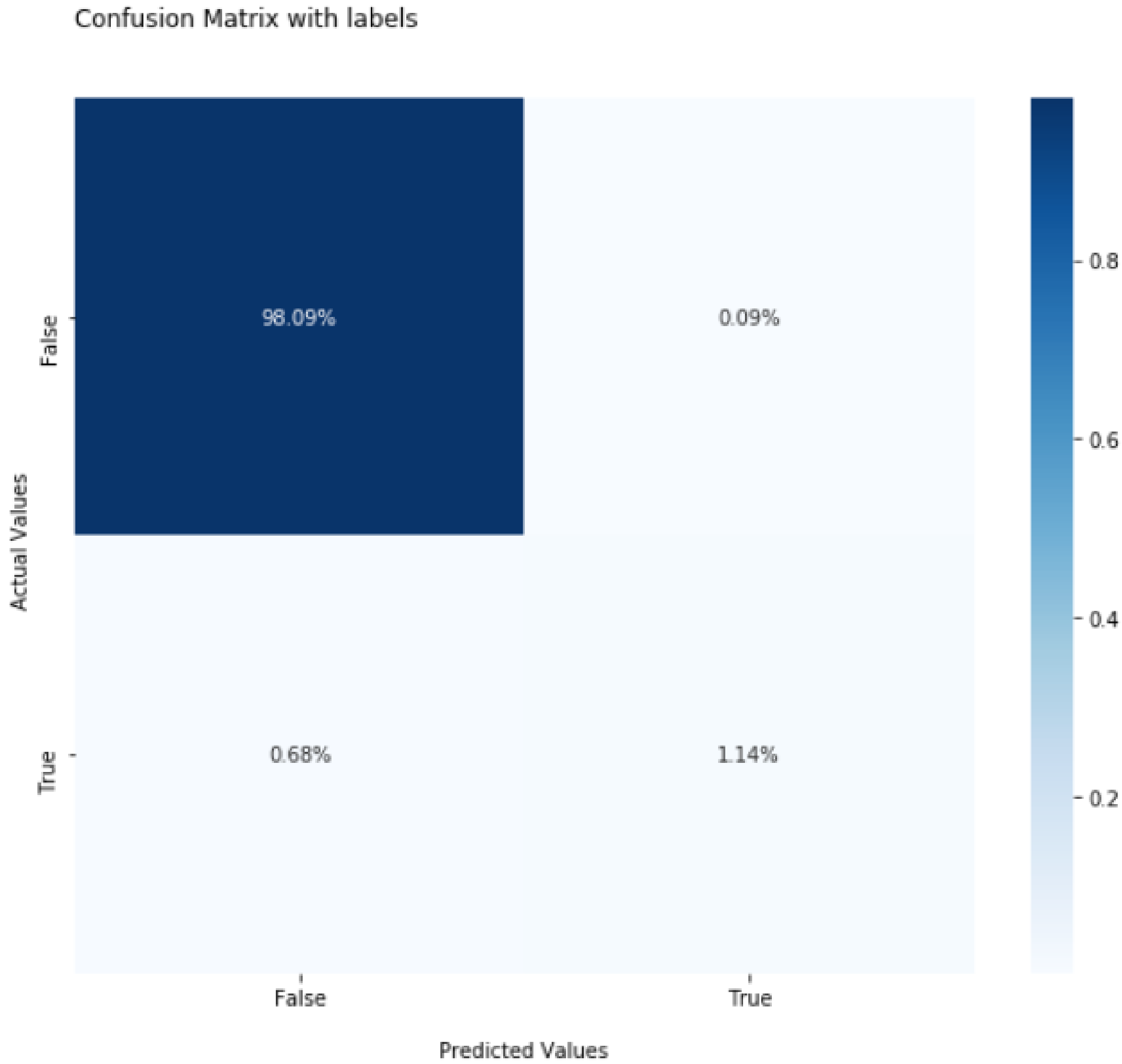
Note: In final analysis I will be doing model selection between these two models

For the second part, I needed to do **Time Series Classification**. For time series classification, idea was to use last 3-7 days data to do early fire detection. Whether the fire will happen or not? I chose **LSTM RNN Model** for this approach. I did some feature engineering to transform the data, by shifting and prepending data from previous days, turned data into 3 dimensions. I split the data into train, test, and validation set. and build the model with 4 layers. *(See Appendix A.6 Data Modeling Part 2)*

Note: In final Analysis I will be doing feature selection to see if the accuracy score increase.

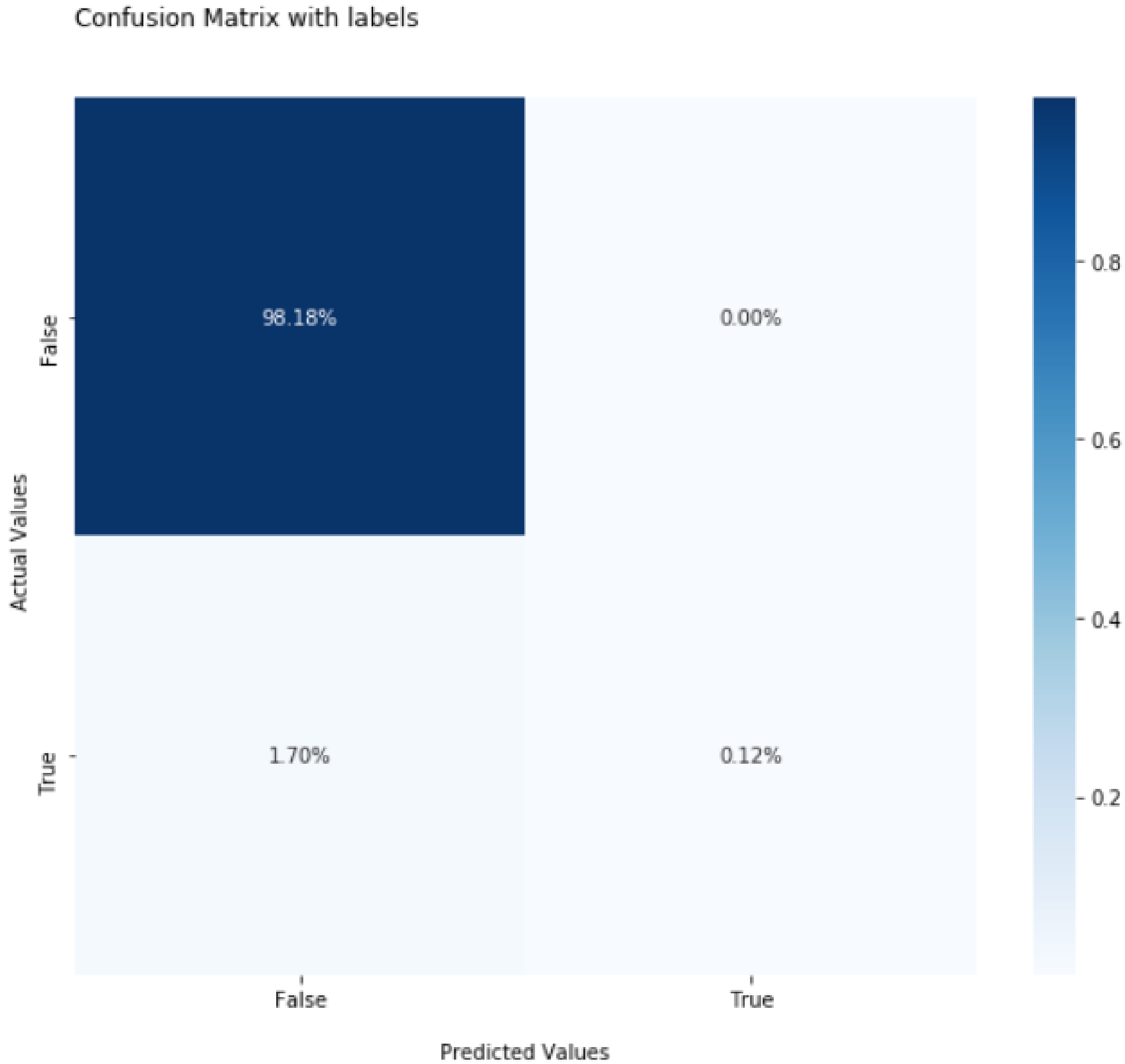
2.5 Preliminary Analysis

```
Accuracy score: 99.23%
Cross validation Accuracy score: 98.81%
Cross validation Precision score: 92.69%
Cross validation Recall score: 37.87%
Cross validation F1 score: 53.74%
```



These are the results from Random Forest Classification. Random Forest shows 99.23% accuracy with high precision score of 92.86%, recall score is 37.87% which is low and F1 score is 53.74%

```
Accuracy score: 98.3%
Cross validation Accuracy score: 98.26%
Cross validation Precision score: 86.29%
Cross validation Recall score: 5.43%
Cross validation F1 score: 10.18%
```



For SVM model Accuracy score was slight lower than Random Forest 98.3%, precision is 86.29%, recall score is 5.43% and F1 score 10.18%

For Time Series here are the results

I did analysis with 3 days, 5 days, and 7 days data.

3 days LSTM model

Before Feature Selection

```
Accuracy score: 64.36%
Precision score: 22.19%
Recall score: 16.74%
```

5 days LSTM Model

Before Feature Selection

```
Accuracy score: 67.08%
Precision score: 28.86%
Recall score: 21.23%
```

7 days LSTM Model

Before Feature Selection

```
Accuracy score: 64.6%
Precision score: 21.7%
Recall score: 15.71%
```

Testing score **using 5 days data shows a slightly higher accuracy with 67.08% and precision 28.86%.** I only attached results form 5 days because that showed highest score. I also run the model for 3 days and 7 days data, but the accuracy score was too low. *See Appendix A.6 Data Modelling Part 2 for full code and results*

2.6 Conclusion

Do weather and soil conditions have any significance in wildfires?

Yes, in Figure **2.2.14** Ihighest temperature months were between Jun, July, and August when the highest number of fires happened. Not only that precipitation also had negative relationship to number of wildfires. Not having enough precipitation can contribute to drought which can result into wildfires.

Can I identify true fires more accurately by combining weather and soil data to fire pixels data?

Yes, as of now MODIS accuracy is about 50% and its' prediction gets accurate as the fire becomes bigger. Whereas the random forest model increases the accuracy when combined with temperature and soil data. However, I would like to mention recall score was below 50% meaning there are high number of false negative.

Can I identify the fire prone area before fire happens, using historical fire pixels data combined with weather indicators and soil quality indicator?

Maybe, this is the problem of time series classification I use previous 5 days to see if I can predict whether it is a true fire or not. Model shows 67.08% accuracy. It is not as high but better than 50 %. However, I have not done feature selection yet, for the final paper, I will be including those results. *(Note: Please see the final paper for results)*

2.7 Assumptions

MODIS data collects pixels and its' location. Each location is 1 km away from the centroid of the pixel. It is never exact location of the fire, from what I know there could be two fires and it could belong one or the other. In addition, MODIS also detects smoke, if there are big wildfires, MODIS can detect a pixel as far as the smoke is going. For simplicity when I mapped wildfires with fire pixels, I mapped them by nearest point to polygon merge. I assumed that any fire that is mapped and is within their burned area, I marked it as labelled and true fire data. For example, if the fire is 10 km away and total burned area is 10 km sq (2417 acres) then I marked it as true fire pixel.

2.8 Limitations and Challenges

There were many Challenges and Limitations:

1. Data Collection took significant amount of time. Downloading data from NASA database and interpreting data was difficult and time consuming, ultimately, I ended up choosing easier route and found a database that was already clean.

2. Not having the exact date of fire event or having dates that are not correctly or missing dates that had end of the year date as its start date. Finding and cleaning

that data took a huge amount of time and work, even though I got the data from official state websites. Data was not managed and had too many missing values.

3. Fire size was measured using acres or km square, however fire distance was 1 dimensional, so mapping it without assumption was difficult.
4. Some fires were mapped 3000-4000 km when the fire size was only 3-4 km. Identifying them and marking them as un-labelled data took significant amount of time.

2.9 Future Applications

For future applications, I do believe the model can be better with the right data and clean data. The most important thing is to have correct date fore fire. I also think including data from other states can make the performance better.

2.10 Recommendations

I mapped fires by year and month, but to further build the model I believe fire should be mapped exactly on the date to do early prediction. Perhaps, LSTM was not a right approach, there is a random Forest time series classification model, that may perform better.

2.11 Implementation

This can be implemented along with MODIS to accurately predict true fire pixels, so fire fighting resources can be allocated correctly. As mentioned earlier MODIS detects 50% of the fires accurately, and only large fires are accurately detected, but adding external factors such as weather and soil quality can increase the accuracy,

2.12 Ethical Assessment

Because wildfires are dangerous and can be a threat to human life. It is crucial to choose right data and correct modelling approach. For this project, I collected data from credible resources such as NASA Earth Data. For modelling.

3 REFERENCES

1. *15 worst wildfires in US history: Earth.org - past: Present: Future*. Earth.Org. (2021, August 30). Retrieved February 16, 2022, from https://earth.org/worst-wildfires-in-us-history/

2. Ares. (2020, February 9). *California wildfires (2013-2020)*. Kaggle. Retrieved February 17, 2022, from https://www.kaggle.com/ananthu017/california-wildfire-incidents-20132020

3. California Department of Forestry and Fire Protection (CAL FIRE). (n.d.). *Incidents overview*. Cal Fire Department of Forestry and Fire Protection. Retrieved February 17, 2022, from https://www.fire.ca.gov/incidents/

4. *California fire perimeters (all)*. California State Geoportal. (n.d.). Retrieved February 17, 2022, from https://gis.data.ca.gov/maps/CALFIRE-Forestry::california-fire-perimeters-all-1/about

5. Chen, F., Niu, S., Tong, X., Zhao, J., Sun, Y., & He, T. (2014, August 27). *The impact of precipitation regimes on forest fires in Yunnan Province, Southwest China*. The Scientific World Journal. Retrieved February 17, 2022, from https://www.hindawi.com/journals/tswj/2014/326782/

6. Eswaramoorthy, V. D. (n.d.). *Increasing the stStatistical Significance For Modis Active Fire Hotspots in Portugal Using One-Class Support Vector Machines*. Retrieved February 17, 2022, from https://cartographymaster.eu/wp-content/theses/2017_Eswaramoorthy_Thesis.pdf

7. *Harmonized World Soil Database v 1.2*. FAO SOILS PORTAL. (n.d.). Retrieved February 17, 2022, from https://www.fao.org/soils-portal/data-hub/soil-maps-and-databases/harmonized-world-soil-database-v12/en/

8. NASA. (2022, February 17). *MCD14DL*. NASA. Retrieved February 17, 2022, from https://earthdata.nasa.gov/earth-observation-data/near-real-time/firms/c6-mcd14dl#ed-firms-attributes

9. NASA. (2022, February 17). *Wildfires*. NASA. Retrieved February 17, 2022, from https://earthdata.nasa.gov/learn/toolkits/wildfires

10. NASA. (n.d.). *NASA Power*. NASA. Retrieved February 17, 2022, from https://power.larc.nasa.gov/

11. *The national burn rate is going up*. Face the Facts USA. (2013, August 26). Retrieved February 17, 2022, from https://facethefactsusa.org/facts/the-national-burn-rate-is-going-up--literally/

12. National Centers for Environmental Information (NCEI). (n.d.). *Climate Data Online Search*. Search | Climate Data Online (CDO) | National Climatic Data Center (NCDC). Retrieved February 17, 2022, from https://www.ncdc.noaa.gov/cdo-web/search

13. *WFIGS - Wildland Fire Locations Full History*. National Interagency Fire Center. (n.d.). Retrieved February 17, 2022, from https://data-nifc.opendata.arcgis.com/datasets/wfigs-wildland-fire-locations-full-history/explore

14. *Wildfires and climate change*. Center for Climate and Energy Solutions. (2021, July 22). Retrieved February 17, 2022, from https://www.c2es.org/content/wildfires-and-climate-change/

15. Ye, A. (2020, October 3). *NASA Wildfire Satellite Data*. Kaggle. Retrieved February 17, 2022, from https://www.kaggle.com/washingtongold/wildfire-satellite-data

16. Zaitseff, S. (2019, January 18). *Classification of time series with LSTM RNN*. Kaggle. Retrieved February 17, 2022, from https://www.kaggle.com/szaitseff/classification-of-time-series-with-lstm-rnn

4 APPENDICES

Please see the attached documents

A.1 Fire events Data Preparation

A.2 MODIS Collection 6 Data Preparation

A.3 Soil and Meteorological Data Preparation

A.4 Data Merging Code

A.5 Data Modelling Part 1 – Simple Classification

A.6 Data Modelling Part 2 – Time Series Classification

Appendix A.1 Fire Events Data Preparation code

Following code provides the steps that were taken to prepare the fire events data with correct dates and location. We identified 3803 wildfire events happened between 2011-2020. Out of which 779 fires were bigger than 300 acres.

In [1]:
```
import datetime as dt
from pathlib import Path
import math
import os
import sqlite3
import json
import geopandas as gpd
import pygeos
import pyproj
import shapely
import shapely.ops as ops
from shapely.geometry import Point, Polygon
from shapely.geometry.polygon import Polygon
from functools import partial

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split


from sklearn import svm
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_m
atrix

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, f_classif, mutual_info_classif
from functools import partial


from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings('ignore')
```

Data Collection of all 3 Fire events Dataset

1a. Load the dataset 1: All California Fire Dataset - https://gis.data.ca.gov/ (https://gis.data.ca.gov/)

In [2]:
```
ca_fires_df = gpd.read_file("Data/California_Fire_Perimeters_(all).geojson")
ca_fires_df.head(2)
```

Out[2]:

	OBJECTID	YEAR_	STATE	AGENCY	UNIT_ID	FIRE_NAME	INC_NUM	ALARM_DATE	
0	21440	2020	CA	CDF	NEU	NELSON	00013212	2020-06-18T00:00:00+00:00	23
1	21441	2020	CA	CDF	NEU	AMORUSO	00011799	2020-06-01T00:00:00+00:00	04

In [3]:
```
ca_fires_df.shape
```

Out[3]:
```
(21318, 19)
```

In [4]:
```
ca_fires_df.crs
```

Out[4]:
```
<Geographic 2D CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: World.
- bounds: (-180.0, -90.0, 180.0, 90.0)
Datum: World Geodetic System 1984 ensemble
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

1b. Load the dataset 1: Supplementary Data for Fire Incident Location - https://data-nifc.opendata.arcgis.com/ (https://data-nifc.opendata.arcgis.com/)

In [5]:
```
fire_location = gpd.read_file("Data/WFIGS_-_Wildland_Fire_Locations_Full_History.geojson")
print(fire_location.shape)
fire_location.head(2)
```

(208013, 95)

Out[5]:

	OBJECTID	ABCDMisc	ADSPermissionState	CalculatedAcres	ContainmentDateTime	Controll
0	1	None	CERTIFIED	50.64	2020-08-06T23:13:07+00:00	06T23:13:
1	2	None	DEFAULT	NaN	None	

2 rows × 95 columns

1c. Load the dataset 1: Supplementary California Fire Dataset - https://www.fire.ca.gov/ (https://www.fire.ca.gov/)

In [6]:
```
fires_df = pd.read_csv('Data/California_Fire_Incidents.csv')
print(fires_df.shape)
```

(1636, 40)

USA Shape File https://www.census.gov/geographies/mapping-files/time-series/geo/carto-boundary-file.html (https://www.census.gov/geographies/mapping-files/time-series/geo/carto-boundary-file.html)

In [7]:
```
USA = gpd.read_file("Data/County/cb_2018_us_county_500k.shp")
USA.head()
```

Out[7]:

	STATEFP	COUNTYFP	COUNTYNS	AFFGEOID	GEOID	NAME	LSAD	ALAND	AV
0	21	007	00516850	0500000US21007	21007	Ballard	06	639387454	69
1	21	017	00516855	0500000US21017	21017	Bourbon	06	750439351	4
2	21	031	00516862	0500000US21031	21031	Butler	06	1103571974	13
3	21	065	00516879	0500000US21065	21065	Estill	06	655509930	6
4	21	069	00516881	0500000US21069	21069	Fleming	06	902727151	7

Data Preliminary Analysis

In [8]:
```
# check for missing value
def percentMissing(df):

    df_numeric = df.select_dtypes(include=[np.number])
    numeric_cols = df_numeric.columns.values

    # % of missing data
    for col in df.columns:
        # create missing indicator for features with missing data
        missing = df[col].isnull()
        pct_missing = np.mean(missing)*100
        #if pct_missing >60:
        print('{} - {}%'.format(col, round(pct_missing)))
        num_missing = np.sum(missing)
```

In [9]:
```
# Checking data type
def Datatype(df):
    # shape and data types of the data
    print("There are {} rows and {} columns".format(df.shape[0], df.shape[1]))
    print(df.dtypes)

    # select numeric columns
    df_numeric = df.select_dtypes(include=[np.number])
    numeric_cols = df_numeric.columns.values
    print(numeric_cols)

    # select non numeric columns
    df_non_numeric = df.select_dtypes(exclude=[np.number])
    non_numeric_cols = df_non_numeric.columns.values
    print(non_numeric_cols)
```

Data Exploration: All California Fire Dataset

All Fire Incident Geopoint data to map fire with right date and time for the first discovery or reported fire incident

In [10]:
```
fire_location = fire_location[(fire_location['InitialLatitude']<= 42) & (fire_
location ['InitialLatitude'] >= 32)]
fire_location = fire_location[(fire_location['InitialLongitude']<= -114) & (fi
re_location['InitialLongitude'] >= -126)]
```

In [11]:
```
fire_location = fire_location[['InitialLatitude','InitialLongitude','FireDisco
veryDateTime','ContainmentDateTime',
                               'ControlDateTime','FireOutDateTime','POOState',
'FireCause','GACC','IncidentName',
                               'LocalIncidentIdentifier','UniqueFireIdentifie
r','WFDSSDecisionStatus','geometry']]
```

In [12]:
```
import string
import re
def get_year(text):
    pattern = r'[^A-Za-z ]'
    if re.search("[^0-9]", text):
        year = text[0:4]
    else:
        None
    return year
```

In [13]:
```
fire_location['IncidentYear'] = fire_location['UniqueFireIdentifier'].apply(la
mbda x: get_year(x))
```

In [14]:
```
fire_location['IncidentYear'] = fire_location['IncidentYear'].astype(int)
```

In [15]:
```
fire_location = fire_location[(fire_location['IncidentYear'] >=2010) & (fire_location['POOState']=='US-CA')]
fire_location = fire_location[fire_location['IncidentYear'] <=2020]
```

In [16]:
```
fire_location = fire_location.drop(['ContainmentDateTime', 'ControlDateTime',
                                    'FireOutDateTime', 'WFDSSDecisionStatus'],
axis = 1)
```

In [17]:
```
fire_location.crs
```

Out[17]:
```
<Geographic 2D CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: World.
- bounds: (-180.0, -90.0, 180.0, 90.0)
Datum: World Geodetic System 1984 ensemble
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

In [18]:
```
fire_location['DiscoveryDate'] = fire_location['FireDiscoveryDateTime'].astype('datetime64[ns]')
fire_location['DiscoveryDate']= fire_location['DiscoveryDate'].dt.strftime('%Y-%m-%d')
fire_location['DiscoveryDate'] = fire_location['DiscoveryDate'].astype('datetime64[ns]')
fire_location['DiscoveryYear'] = fire_location['DiscoveryDate'].dt.year
fire_location['DiscoveryMonth'] = fire_location['DiscoveryDate'].dt.month
fire_location['DiscoveryDay'] = fire_location['DiscoveryDate'].dt.day
```

In [19]:
```
fire_location = fire_location[['UniqueFireIdentifier', 'IncidentYear', 'IncidentName', 'DiscoveryDate',
                               'DiscoveryYear','DiscoveryMonth', 'DiscoveryDay', 'geometry']]
```

In [22]:
```
fire_location.head(2)
```

Out[22]:

	UniqueFireIdentifier	IncidentYear	IncidentName	DiscoveryDate	DiscoveryYear	Discover
79385	2011-NVCNC-000020	2011	Washoe	2011-01-26	2011	
128205	2014-CALBOR-001660	2014	Casitas	2014-06-17	2014	

In [23]:
```
fire_location = fire_location.sort_values(['DiscoveryDate'], ascending=True)
```

In [24]:
```
percentMissing(fire_location)
```

```
UniqueFireIdentifier - 0%
IncidentYear - 0%
IncidentName - 0%
DiscoveryDate - 0%
DiscoveryYear - 0%
DiscoveryMonth - 0%
DiscoveryDay - 0%
geometry - 0%
```

Check for Correct year for fire incidents and delete duplicates based on coordinates and date

In [25]:
```
fire_location["Disc_minus_ID"] = fire_location["IncidentYear"] - fire_location
["DiscoveryYear"]
fire_location[fire_location["Disc_minus_ID"]!=0]
```

Out[25]:

	UniqueFireIdentifier	IncidentYear	IncidentName	DiscoveryDate	DiscoveryYear	DiscoveryI
53158	2016-CACNF-002667	2016	MUTUAL AID DE LUZ	2015-08-09	2015	

In [26]:
```
fire_location = fire_location.drop(['Disc_minus_ID'], axis = 1)
```

In [27]:
```
fire_location= fire_location[~fire_location.duplicated(['geometry', 'Discovery
Date'], keep='first')]
```

In [28]:
```
fire_location['DiscoveryYear'].describe()
```

Out[28]:
```
count    23849.000000
mean      2018.729129
std          1.439810
min       2011.000000
25%       2018.000000
50%       2019.000000
75%       2020.000000
max       2020.000000
Name: DiscoveryYear, dtype: float64
```

All Geospatial data for fire area perimeter

In [29]:
```
Datatype(ca_fires_df)
```

```
There are 21318 rows and 19 columns
OBJECTID          int64
YEAR_            object
STATE            object
AGENCY           object
UNIT_ID          object
FIRE_NAME        object
INC_NUM          object
ALARM_DATE       object
CONT_DATE        object
CAUSE           float64
COMMENTS         object
REPORT_AC       float64
GIS_ACRES       float64
C_METHOD        float64
OBJECTIVE       float64
FIRE_NUM         object
SHAPE_Length    float64
SHAPE_Area      float64
geometry       geometry
dtype: object
['OBJECTID' 'CAUSE' 'REPORT_AC' 'GIS_ACRES' 'C_METHOD' 'OBJECTIVE'
 'SHAPE_Length' 'SHAPE_Area']
['YEAR_' 'STATE' 'AGENCY' 'UNIT_ID' 'FIRE_NAME' 'INC_NUM' 'ALARM_DATE'
 'CONT_DATE' 'COMMENTS' 'FIRE_NUM' 'geometry']
```

In [30]:
```
ca_fires_df['ALARM_DATE'] = pd.to_datetime(ca_fires_df['ALARM_DATE'], errors =
'coerce')
ca_fires_df['ALARM_DATE'] = ca_fires_df['ALARM_DATE'].astype('datetime64[ns]')
ca_fires_df['FireDate']= ca_fires_df['ALARM_DATE'].dt.strftime('%Y-%m-%d')
ca_fires_df['FireDate'] = ca_fires_df['FireDate'].astype('datetime64[ns]')
ca_fires_df['FireYear'] = ca_fires_df['FireDate'].dt.year
ca_fires_df['FireMonth'] = ca_fires_df['FireDate'].dt.month
ca_fires_df['FireDay'] = ca_fires_df['FireDate'].dt.day

ca_fires_df['CONT_DATE'] = pd.to_datetime(ca_fires_df['CONT_DATE'], errors =
'coerce')
ca_fires_df['ContDate']= ca_fires_df['CONT_DATE'].dt.strftime('%Y-%m-%d')
```

In [31]:
```
ca_fires_df = ca_fires_df[(ca_fires_df['FireYear']>= 2011) & (ca_fires_df['STA
TE']== 'CA')]
ca_fires_df.shape
```

Out[31]:
```
(3677, 24)
```

In [32]:
```
ca_fires_df = ca_fires_df.sort_values('FireDate')
```

In [33]:
```
# These columns are unnecessary
# information, agency who was incharge is not needed,
# INC NUM is unique, C_method is how it was tracked.

ca_fires_df = ca_fires_df.drop(['YEAR_', 'C_METHOD', 'AGENCY', 'INC_NUM', 'ALARM_DATE',
                                'CONT_DATE','FIRE_NUM', 'COMMENTS', 'OBJECTIVE', 'SHAPE_Area',
                                'SHAPE_Length', 'REPORT_AC'], axis = 1)
```

In [34]:
```
ca_fires_df = ca_fires_df.rename(columns={'OBJECTID': 'ObjectID', 'CAUSE': 'FireCause',
                                          'GIS_ACRES': 'TotalAcres','STATE':'State', 'UNIT_ID':'UnitID',
                                          'FIRE_NAME':'Name', 'ContDate': 'ContainmentDate'})
```

In [35]:
```
percentMissing(ca_fires_df)
```

```
ObjectID - 0%
State - 0%
UnitID - 1%
Name - 0%
FireCause - 1%
TotalAcres - 0%
geometry - 0%
FireDate - 0%
FireYear - 0%
FireMonth - 0%
FireDay - 0%
ContainmentDate - 1%
```

Check for duplicate values based on geometry and date

In [36]:
```
ca_fires_df = ca_fires_df.sort_values(['FireDate'], ascending=True)
```

In [37]:
```
ca_fires_df = ca_fires_df[~ca_fires_df.duplicated(['geometry', 'FireDate'], keep='first')]
```

In [38]:
```
ca_fires_df = ca_fires_df[~ca_fires_df.duplicated(['geometry', 'TotalAcres'], keep='first')]
```

In [39]:
```
ca_fires_df.shape
```

Out[39]:
```
(3673, 12)
```

All California Incidents with name and coordinates. This data is used as supplement data to check for fire date inaccuracies

In [40]:
```
fires_df['Started'] = fires_df['Started'].astype('datetime64[ns]')
fires_df['CaDate']= fires_df['Started'].dt.strftime('%Y-%m-%d')
fires_df['Extinguished'] = fires_df['Extinguished'].astype('datetime64[ns]')
fires_df['ExitDate']= fires_df['Extinguished'].dt.strftime('%Y-%m-%d')
```

In [41]:
```
fires_df = fires_df[(fires_df['Latitude']<= 42) & (fires_df['Latitude'] >= 32
)]
fires_df = fires_df[(fires_df['Longitude']<= -114) & (fires_df['Longitude'] >=
-126)]
```

In [42]:
```
fires_df['CaDate'] = fires_df['CaDate'].astype('datetime64[ns]')
fires_df['CaYear'] = fires_df['CaDate'].dt.year
fires_df['CaMonth'] = fires_df['CaDate'].dt.month
fires_df['CaDay'] = fires_df['CaDate'].dt.day
```

In [43]:
```
fires_df = fires_df[['Name', 'Latitude', 'Longitude', 'CaDate','CaYear', 'CaMo
nth', 'CaDay', 'ExitDate', 'AcresBurned',
                     'ArchiveYear', 'Counties', 'UniqueId']]
fires_df.shape
```

Out[43]:
```
(1465, 12)
```

In [44]:
```
percentMissing(fires_df)
```

```
Name - 0%
Latitude - 0%
Longitude - 0%
CaDate - 0%
CaYear - 0%
CaMonth - 0%
CaDay - 0%
ExitDate - 4%
AcresBurned - 0%
ArchiveYear - 0%
Counties - 0%
UniqueId - 0%
```

Check for Correct year for fire incidents and delete duplicates based on coordinates and date

In [45]:
```
fires_df["Archive_minus_Year"] = fires_df["ArchiveYear"] - fires_df["CaYear"]
fires_df[fires_df["Archive_minus_Year"]!=0]
```

Out[45]:

	Name	Latitude	Longitude	CaDate	CaYear	CaMonth	CaDay	ExitDate	AcresBurned	A
1019	Taglio Fire	37.21812	-121.07761	1969-12-31	1969	12	31	2018-01-09	12.0	
1261	Bridge Fire	38.07135	-122.76751	1969-12-31	1969	12	31	2019-01-04	45.0	

In [46]:
```
idx = fires_df[fires_df["Archive_minus_Year"]!=0].index.tolist()

# Taglio fire was in May 17, 2017
fires_df.at[idx[0],'CaDate']= '2017-05-17'
fires_df.at[idx[0],'ExitDate']= '2017-05-17'
fires_df.at[idx[0],'CaYear']= 2017
fires_df.at[idx[0],'CaMonth']= 5
fires_df.at[idx[0],'CaDay']= 17


# Taglio fire was in May 17, 2017
fires_df.at[idx[1],'CaDate']= '2018-08-08'
fires_df.at[idx[1],'ExitDate']= '2018-08-09'
fires_df.at[idx[1],'CaYear']= 2018
fires_df.at[idx[1],'CaMonth']= 8
fires_df.at[idx[1],'CaDay']= 8
```

In [47]:
```
fires_df["Archive_minus_Year"] = fires_df["ArchiveYear"] - fires_df["CaYear"]
fires_df[fires_df["Archive_minus_Year"]!=0]
```

Out[47]:

Name	Latitude	Longitude	CaDate	CaYear	CaMonth	CaDay	ExitDate	AcresBurned	Archive

In [48]:
```
geometry = [Point(xy) for xy in zip(fires_df['Longitude'], fires_df['Latitude'
])]
geometry[:3]
crs = {'init': "EPSG:4326"}
fires_df1 = gpd.GeoDataFrame(fires_df, crs=crs, geometry=geometry)
fires_df1.head(2)
```

Out[48]:

	Name	Latitude	Longitude	CaDate	CaYear	CaMonth	CaDay	ExitDate	AcresBurne
0	Rim Fire	37.857000	-120.086000	2013-08-17	2013	8	17	2013-09-06	257314.
1	Powerhouse Fire	34.585595	-118.423176	2013-05-30	2013	5	30	2013-06-08	30274.

Check for duplicate values based on geometry and date

In [49]:
```
fires_df1 = fires_df1.sort_values(['CaDate'], ascending=True)
```

In [50]:
```
fires_df1 = fires_df1[~fires_df1.duplicated(['geometry', 'CaDate'], keep='firs
t')]
```

In [51]:
```
fires_df1 = fires_df1[~fires_df1.duplicated(['Name','geometry', 'AcresBurned'
], keep='first')]
```

In [52]:
```
fires_df1 = fires_df1.sort_values('CaDate')
```

In [53]:
```
fires_df1 = fires_df1.drop(["Archive_minus_Year"], axis = 1)
```

In [54]:
```
fires_df1["ArchiveYear"].describe()
```

Out[54]:
```
count    1436.000000
mean     2016.804318
std         1.800893
min      2013.000000
25%      2016.000000
50%      2017.000000
75%      2018.000000
max      2019.000000
Name: ArchiveYear, dtype: float64
```

Making Copies for the record

In [55]:
```
FireLocation = fire_location.copy()
FirePolygon = ca_fires_df.copy()
FireList = fires_df1.copy()
```

In [56]:
```
print(FirePolygon.crs)
print(FireLocation.crs)
print(FireList.crs)
```

```
epsg:4326
epsg:4326
+init=epsg:4326 +type=crs
```

In [57]:
```
FireLocation.crs
```

Out[57]:
```
<Geographic 2D CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: World.
- bounds: (-180.0, -90.0, 180.0, 90.0)
Datum: World Geodetic System 1984 ensemble
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

In [58]:
```
FireList.crs
```

Out[58]:
```
<Geographic 2D CRS: +init=epsg:4326 +type=crs>
Name: WGS 84
Axis Info [ellipsoidal]:
- lon[east]: Longitude (degree)
- lat[north]: Latitude (degree)
Area of Use:
- name: World.
- bounds: (-180.0, -90.0, 180.0, 90.0)
Datum: World Geodetic System 1984 ensemble
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

Merging Fire Perimeter with Fire Location by date.

Projecting the crs to from WGS84 to NAD83 so we can compute distances between points correctly in meters or kilometers.

In [59]:
```
FireLocation = FireLocation.to_crs({'init': "EPSG:3310"})
FirePolygon = FirePolygon.to_crs({'init': "EPSG:3310"})
FireList = FireList.to_crs({'init': "EPSG:3310"})
```

In [60]:
```
FireLocation['Fire'] = 0
FireList['Fire'] = 0
```

In [61]:
```
def get_nearestpoints(df1, df1day, df2, df2day, dist):
    """
        This Function merges dataframe for selected day by finding nearest poi
nts
        for each day and creates mini dfs for each day of month
    """
    days = list(range(1, 32))
    dfs = []
    for day in days:
        df = df1[df1[df1day] == day]
        df3 = df2
        m_df = gpd.sjoin_nearest(df, df3, how='left', max_distance = 10,distan
ce_col=dist)
        m_df[dist] = m_df[dist].apply(lambda x: x/1000)
        d = pd.DataFrame(m_df)
        dfs.append(d)

    dfs = pd.concat(dfs)
    dfs = dfs.drop(["Fire"], axis = 1)
    return dfs
```

In [62]:
```
def merge_data(data1, df1year, df1month, df1day, data2, df2year, df2month,df2d
ay, dist, year):
    """
        This Function filters dataframe by year and months and calls for day d
fs,
        append it and then converts it into pandas df.
    """
    months = list(range(1, 13))
    dfs = []
    for month in months:
        df1 = data1[(data1[df1year] == year) & (data1[df1month] == month)]
        df2 = data2[(data2[df2year] == year) & (data2[df2month] == month)]

        if 'Fire' in df2.columns:
            df = gpd.sjoin_nearest(df1, df2, how='left', distance_col=dist)
            df[dist] = df[dist].apply(lambda x: x/1000)
            d = pd.DataFrame(df)
            dfs.append(d)
        else:
            df = get_nearestpoints(df1, df1day, df2, df2day, dist)
            dfs.append(df)

    dfs = pd.concat(dfs)
    return dfs
```

In [63]:
```
def get_data(df1, df1year, df1month, df1day, df2, df2year, df2month, df2day, d
ist):
    """
        This calls_ for all dataframes and combine it and create one dataset f
or
        fire data, so we can use the combined information to find the estimate
d dates
    """
    years = list(range(2011, 2021))
    dataframesList = []
    for year in years:
        data = merge_data(df1, df1year, df1month, df1day, df2, df2year, df2mon
th, df2day, dist, year)
        dataframesList.append(data)

    df = gpd.GeoDataFrame(pd.concat(dataframesList), crs=crs)
    try:
        df.drop('index_right', axis=1, inplace=True)
    except ValueError:
        # ignore if there are no index columns
        pass

    print(df.shape)

    return df
```

In [64]:
```
nearestfire1 = get_data(FirePolygon, 'FireYear', 'FireMonth', 'FireDay',
                        FireLocation, 'DiscoveryYear', 'DiscoveryMonth', 'Disco
veryDay', 'locationdist')
```

```
(3838, 21)
```

Handling Missing Values and Duplicates. Fixing Bad Data

Analysis of Fire Start Dates Missing Values using Fire Location Data

In [65]:
```
nearestfire1 = nearestfire1.sort_values('locationdist', ascending=True)
nearestfire1.head(2)
```

Out[65]:

	ObjectID	State	UnitID	Name	FireCause	TotalAcres	geometry	FireDate	FireYear
252	21694	CA	SHF	FLAT	10.0	62.040543	MULTIPOLYGON (((-289859.162 314145.109, -28986...	2020-06-30	2020.0
375	21819	CA	SKU	NOYES 1-14	1.0	8.120131	MULTIPOLYGON (((-224232.716 382534.303, -22423...	2020-07-28	2020.0

2 rows × 21 columns

In [66]:
```
def get_duration(df):
    df['LocDate_minus_FireDate'] = (df["DiscoveryDate"] - df["FireDate"]).dt.d
ays
    df = df.sort_values('LocDate_minus_FireDate', ascending=True)
    return df
```

In [67]:
```
nearestfire1 = get_duration(nearestfire1)
```

In [68]:
```
# dropping duplicates based on earliest discovery date and location distance =
= 0 for fires bigger than 100 acres
nearestfire1 = nearestfire1[~((nearestfire1.index.duplicated(keep='first')) &
                            (nearestfire1['LocDate_minus_FireDate']<=0) &
                            (nearestfire1['TotalAcres']>100) & (nearestfire1[
'locationdist']==0))]

# dropping duplicates based on earliest discovery date and location distance =
= 0 for fires less than 100 acres
nearestfire1 = nearestfire1[~((nearestfire1.index.duplicated(keep='first')) &
                              (nearestfire1['LocDate_minus_FireDate']<=0) &
                              (nearestfire1['TotalAcres']<100) & (nearestfire1
['locationdist']==0))]

# dropping it because it is not mapped correctly
nearestfire1 = nearestfire1[~((nearestfire1.index.duplicated(keep='first')) &
                            (nearestfire1['LocDate_minus_FireDate']>=0) &
                            (nearestfire1['TotalAcres']<100) & (nearestfire1[
'locationdist']==0))]

nearestfire1 = nearestfire1[~((nearestfire1.index.duplicated(keep='first')) &
                            (nearestfire1['LocDate_minus_FireDate']>=0) &
                            (nearestfire1['TotalAcres']>100) & (nearestfire1[
'locationdist']==0))]
```

In [69]:
```
nearestfire1 = get_duration(nearestfire1)
nearestfire1 = nearestfire1.sort_values('locationdist', ascending=True)
```

In [70]:
```
nearestfire1 = nearestfire1[~nearestfire1.index.duplicated(keep='first')]
nearestfire1 = get_duration(nearestfire1)
nearestfire1 = nearestfire1.sort_values('locationdist', ascending=True)
```

In [71]:
```
nearestfire1[(nearestfire1['LocDate_minus_FireDate'] <0) &
             (nearestfire1['DiscoveryDate'].notnull()) &
             (nearestfire1['locationdist'] ==0)]

## replacing fire date with discovery date.
nearestfire1.loc[(nearestfire1['LocDate_minus_FireDate'] <0) &
                 (nearestfire1['DiscoveryDate'].notnull()) &
                 (nearestfire1['locationdist'] ==0), 'FireDate'] = nearestfire
1['DiscoveryDate']
```

In [72]:
```
# replacing discovery dates for any fire which is less than kilometer away and
starts earlier than reported fire date
nearestfire1[(nearestfire1['LocDate_minus_FireDate'] <0) &
             (nearestfire1['DiscoveryDate'].notnull()) &
             (nearestfire1['locationdist'] >0) &
             (nearestfire1['locationdist'] <1) & (nearestfire1['TotalAcres']>1
00)]

nearestfire1.loc[(nearestfire1['LocDate_minus_FireDate'] <0) &
                 (nearestfire1['DiscoveryDate'].notnull()) &
                 (nearestfire1['locationdist'] >0) &
                 (nearestfire1['locationdist'] <1) & (nearestfire1['TotalAcre
s']>100), 'FireDate'] = nearestfire1['DiscoveryDate']
```

In [73]:
```
# replacing discovery dates for any fire which is less than kilometer away and
starts earlier than reported fire date
nearestfire1[(nearestfire1['LocDate_minus_FireDate'] <0) &
             (nearestfire1['DiscoveryDate'].notnull()) &
             (nearestfire1['locationdist'] >0) &
             (nearestfire1['locationdist'] <0.2) & (nearestfire1['TotalAcres']
<100)]

nearestfire1.loc[(nearestfire1['LocDate_minus_FireDate'] <0) &
                 (nearestfire1['DiscoveryDate'].notnull()) &
                 (nearestfire1['locationdist'] >0) &
                 (nearestfire1['locationdist'] <0.2) & (nearestfire1['TotalAcr
es']<100), 'FireDate'] = nearestfire1['DiscoveryDate']
```

In [74]:
```
nearestfire1 = get_duration(nearestfire1)
nearestfire1 = nearestfire1.sort_values('locationdist', ascending=True)
```

In [75]:
```
# replacing discovery dates for any fire which is less than kilometer away and
starts earlier than reported fire date
nearestfire1[(nearestfire1['LocDate_minus_FireDate'] <=-1) &
             (nearestfire1['DiscoveryDate'].notnull()) &
             (nearestfire1['LocDate_minus_FireDate'] >=-2) &
             (nearestfire1['locationdist'] <0.5)]

nearestfire1.loc[(nearestfire1['LocDate_minus_FireDate'] <=-1) &
             (nearestfire1['DiscoveryDate'].notnull()) &
             (nearestfire1['LocDate_minus_FireDate'] >=-2) &
             (nearestfire1['locationdist'] <0.5), 'FireDate'] = nearestfire1[
'DiscoveryDate']
```

In [76]:
```
nearestfire1 = get_duration(nearestfire1)
nearestfire1 = nearestfire1.sort_values('locationdist', ascending=True)
```

In [77]:
```
# replacing discovery dates for any fire which is less than kilometer away and
starts earlier than reported fire date
nearestfire1.loc[(nearestfire1['LocDate_minus_FireDate'] <=-1) &
                 (nearestfire1['DiscoveryDate'].notnull()) &
                 (nearestfire1['LocDate_minus_FireDate'] >=-2) &
                 (nearestfire1['locationdist'] <1),'FireDate'] = nearestfire1[
'DiscoveryDate']
```

In [78]:
```
nearestfire1 = get_duration(nearestfire1)
nearestfire1 = nearestfire1.sort_values('locationdist', ascending=True)
```

In [79]:
```
# replacing discovery dates for any fire which is less than 2 kilometer away a
nd
# starts earlier than reported fire date, but shares the same name
nearestfire1[(nearestfire1['LocDate_minus_FireDate'] <=-1) &
             (nearestfire1['DiscoveryDate'].notnull()) &
             (nearestfire1['LocDate_minus_FireDate'] >=-2) &
             (nearestfire1['locationdist'] <2)]

nearestfire1.loc[(nearestfire1['LocDate_minus_FireDate'] <=-1) &
             (nearestfire1['DiscoveryDate'].notnull()) &
             (nearestfire1['LocDate_minus_FireDate'] >=-2) &
             (nearestfire1['locationdist'] <2),'FireDate'] = nearestfire1['Dis
coveryDate']

nearestfire1 = get_duration(nearestfire1)
nearestfire1 = nearestfire1.sort_values('locationdist', ascending=True)
```

In [80]:
```
# replacing discovery dates for any fire which is less than 3 kilometer away a
nd
# starts earlier than reported fire date, but shares the same name
nearestfire1[(nearestfire1['LocDate_minus_FireDate'] <=-1) &
             (nearestfire1['DiscoveryDate'].notnull()) &
             (nearestfire1['LocDate_minus_FireDate'] >=-3) &
             (nearestfire1['locationdist'] <=3)]

nearestfire1.loc[(nearestfire1['LocDate_minus_FireDate'] <=-1) &
             (nearestfire1['DiscoveryDate'].notnull()) &
             (nearestfire1['LocDate_minus_FireDate'] >=-3) &
             (nearestfire1['locationdist'] <3),'FireDate'] = nearestfire1['Dis
coveryDate']

nearestfire1 = get_duration(nearestfire1)
nearestfire1 = nearestfire1.sort_values('locationdist', ascending=True)
```

In [81]:
```
# replacing discovery dates for any fire which is less than 10 kilometer away
 and
# starts earlier than reported fire date, but shares the same name and has hig
her totalacres
nearestfire1[(nearestfire1['LocDate_minus_FireDate'] <=-1) &
             (nearestfire1['DiscoveryDate'].notnull()) &
             (nearestfire1['LocDate_minus_FireDate'] >=-10) &
             (nearestfire1['locationdist'] <=3) &
             (nearestfire1['TotalAcres']>=50)]

nearestfire1.loc[(nearestfire1['LocDate_minus_FireDate'] <=-1) &
             (nearestfire1['DiscoveryDate'].notnull()) &
             (nearestfire1['LocDate_minus_FireDate'] >=-10) &
             (nearestfire1['locationdist'] <=3) &
             (nearestfire1['TotalAcres']>=50),'FireDate'] = nearestfire1['Disc
overyDate']

nearestfire1 = get_duration(nearestfire1)
nearestfire1 = nearestfire1.sort_values('locationdist', ascending=True)
```

In [82]:
```
# replacing discovery dates for any fire which is less than 10 kilometer away
 and
# starts earlier than reported fire date, but shares the same name and has 100
0+ totalacres
nearestfire1[(nearestfire1['LocDate_minus_FireDate'] <=-1) &
             (nearestfire1['DiscoveryDate'].notnull()) &
             (nearestfire1['LocDate_minus_FireDate'] >=-20) &
             (nearestfire1['locationdist'] <=3) &
             (nearestfire1['TotalAcres']>=1000)]

nearestfire1.loc[(nearestfire1['LocDate_minus_FireDate'] <=-1) &
             (nearestfire1['DiscoveryDate'].notnull()) &
             (nearestfire1['LocDate_minus_FireDate'] >=-20) &
             (nearestfire1['locationdist'] <=3) &
             (nearestfire1['TotalAcres']>=1000),'FireDate'] = nearestfire1['Di
scoveryDate']

nearestfire1 = get_duration(nearestfire1)
nearestfire1 = nearestfire1.sort_values('locationdist', ascending=True)
```

In [83]:
```
Fires_df1 = nearestfire1[['ObjectID', 'UnitID', 'FireCause', 'TotalAcres', 'ge
ometry', 'FireDate', 'FireYear',
                          'FireMonth', 'FireDay', 'Name']]

Fires_df1[Fires_df1.duplicated(['geometry'], keep=False)]
```

Out[83]:

ObjectID	UnitID	FireCause	TotalAcres	geometry	FireDate	FireYear	FireMonth	FireDay	Na

In [84]:
```
Fires_df1.shape
```

Out[84]: (3670, 10)

In [85]:
```
## get left over fire location data using UniqueIdentifier to later merge into
fire data.
UniqueFireIdentifier1 = nearestfire1[(nearestfire1['LocDate_minus_FireDate'] <
=-10) &
                              (nearestfire1['DiscoveryDate'].notnull()) &
                              (nearestfire1['locationdist'] >10)]
UniqueFireIdentifier1 = UniqueFireIdentifier1[~UniqueFireIdentifier1.duplicate
d(['UniqueFireIdentifier'], keep='first')]
UniqueFireIdentifier1 = UniqueFireIdentifier1[['UniqueFireIdentifier']]
```

Analysis of Fire Start Dates using State Fire List

In [86]:
```
nearestfire2 = get_data(Fires_df1, 'FireYear', 'FireMonth', 'FireDay',
                        FireList, 'CaYear', 'CaMonth', 'CaDay', 'firedist')
```

(3675, 24)

In [87]:
```
def get_duration(df):
    df['CaDate_minus_FireDate'] = (df["CaDate"] - df["FireDate"]).dt.days
    df['Area_diff'] = (df["AcresBurned"] - df["TotalAcres"])
    df = df.sort_values('CaDate_minus_FireDate', ascending=True)

    return df
```

In [88]:
```
nearestfire2 = get_duration(nearestfire2)
nearestfire2 = nearestfire2.sort_values('firedist', ascending=True)
```

In [89]:
```
# dropping duplicates based on earliest discovery date and Location distance =
= 0 for fires bigger than 100 acres
nearestfire2 = nearestfire2[~((nearestfire2.index.duplicated(keep='first')) &
                              (nearestfire2['CaDate_minus_FireDate']<=0) &
                              (nearestfire2['TotalAcres']<100))]
```

In [90]:
```
nearestfire2 = get_duration(nearestfire2)
nearestfire2 = nearestfire2.sort_values('firedist', ascending=True)
```

In [91]:
```
# dropping it because it is not mapped correctly
nearestfire2 = nearestfire2[~((nearestfire2.index.duplicated(keep='first')) &
                              (nearestfire2['CaDate_minus_FireDate']>=0) &
                              (nearestfire2['TotalAcres']>100))]

nearestfire2 = get_duration(nearestfire2)
nearestfire2 = nearestfire2.sort_values('firedist', ascending=True)
```

In [92]:
```
nearestfire2[nearestfire2.index.duplicated(keep=False)]
```

Out[92]:

ObjectID	UnitID	FireCause	TotalAcres	geometry	FireDate	FireYear	FireMonth	FireDay	Na

0 rows × 26 columns

In [93]:
```
## No dates that are listed earlier for state fire list. We will use the origi
nal fire date.
nearestfire2[(nearestfire2['CaDate_minus_FireDate'] >-10) &
             (nearestfire2['CaDate_minus_FireDate'] <=-1) &
             (nearestfire2['CaDate'].notnull()) &
             (nearestfire2['firedist'] <3)].sort_values('CaDate_minus_FireDat
e', ascending=True)

## No dates that are listed earlier for state fire list. We will use the origi
nal fire date.
nearestfire2.loc[(nearestfire2['CaDate_minus_FireDate'] >-10) &
                 (nearestfire2['CaDate_minus_FireDate'] <=-1) &
                 (nearestfire2['CaDate'].notnull()) &
                 (nearestfire2['firedist'] <3), 'FireDate'] = nearestfire2['Ca
Date']

nearestfire2 = get_duration(nearestfire2)
nearestfire2 = nearestfire2.sort_values('firedist', ascending=True)
```

In [94]:
```
UniqueId =  nearestfire2[(nearestfire2['CaDate_minus_FireDate'] >-20) &
                          (nearestfire2['CaDate_minus_FireDate'] <=-1) &
                          (nearestfire2['CaDate'].notnull()) &
                          (nearestfire2['firedist'] >2)].sort_values('firedist'
, ascending=True)
UniqueId = UniqueId[~UniqueId.duplicated(['UniqueId'], keep='first')]
UniqueId = UniqueId[['UniqueId']]
```

In [95]:
```
nearestfire2.shape
```

Out[95]: (3670, 26)

In [96]:
```
Fires_df2 = nearestfire2[['ObjectID', 'UnitID', 'FireCause', 'TotalAcres', 'ge
ometry', 'FireDate', 'FireYear',
                             'FireMonth', 'FireDay', 'Name_left']]

Fires_df2[Fires_df2.duplicated(['geometry'], keep=False)]
```

Out[96]:

	ObjectID	UnitID	FireCause	TotalAcres	geometry	FireDate	FireYear	FireMonth	FireDay	Na

In [97]:
```
UniqueFireIdentifier1 = UniqueFireIdentifier1['UniqueFireIdentifier'].tolist()
UniqueId = UniqueId['UniqueId'].tolist()
```

In [98]:
```
Fires_df4 = FireList[FireList['UniqueId'].isin(UniqueId)]
Fires_df4 = Fires_df4[['geometry','CaDate', 'CaYear', 'CaMonth', 'CaDay', 'Acr
esBurned', 'Name', 'UniqueId']]
```

In [99]:
```
Fires_df4.shape
```

Out[99]: (334, 8)

What is the Top Cause for Wildfire?

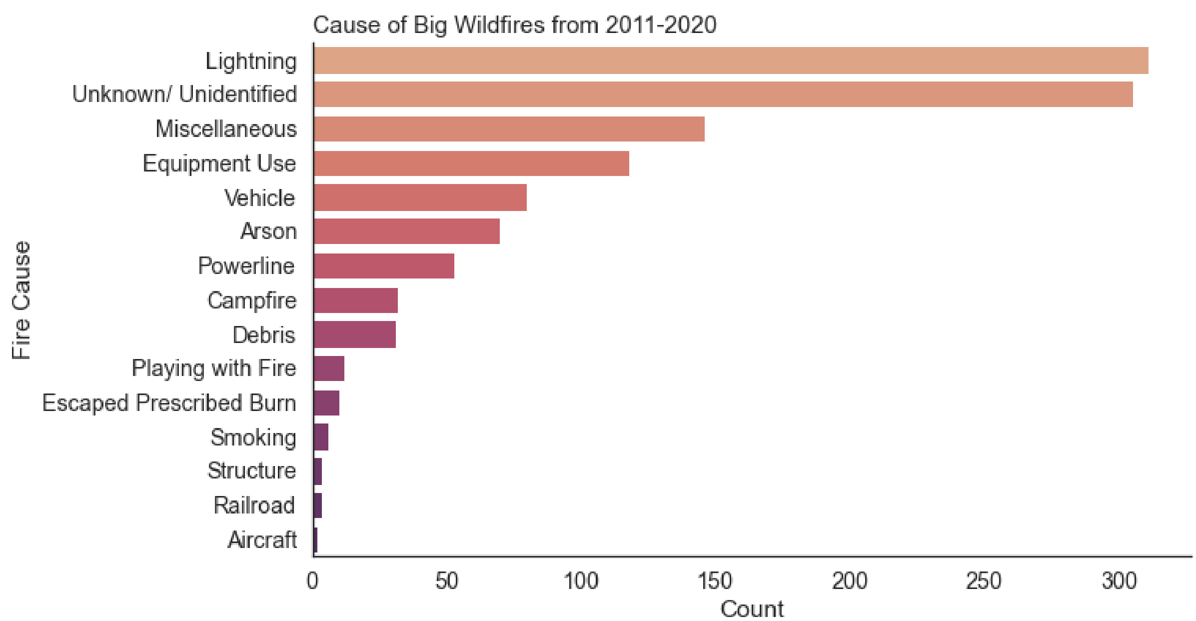
In [100]:
```
Fires_df2['FireYear'].describe()
```

Out[100]:
```
count    3670.000000
mean     2015.915804
std         2.875493
min      2011.000000
25%      2013.000000
50%      2016.000000
75%      2018.000000
max      2020.000000
Name: FireYear, dtype: float64
```

In [101]:
```
Fires_df2 = Fires_df2[Fires_df2['FireYear'] <=2020]
```

In [102]:
```
Fires_df2["FireCause"].replace({1: "Lightning", 2: "Equipment Use", 3:"Smokin
g",
                                4:"Campfire", 5:"Debris", 6:"Railroad", 7:"Ars
on", 8:"Playing with Fire",
                                9:"Miscellaneous",10:"Vehicle", 11:"Powerline"
, 14:"Unknown/ Unidentified",
                                15: 'Structure', 16: 'Aircraft', 18: 'Escaped
 Prescribed Burn', 19: "Illegal Alien Campfire"}, inplace=True)
```

In [103]:
```
plt.rcParams['figure.figsize'] = [10,6]
sns.set(font_scale = 1.3)
sns.set_style("white")
df2 = Fires_df2[Fires_df2['TotalAcres'] >99]
ax = sns.countplot(y=df2['FireCause'],data=df2, palette="flare",
                   order=df2['FireCause'].value_counts().index)

ax.set_title("Cause of Big Wildfires from 2011-2020",fontsize = 15, loc='left'
)
ax.set_xlabel("Count")
ax.set_ylabel("Fire Cause")
sns.despine()

plt.show()
```



In [104]:
```
Fires_df2 = Fires_df2.drop(['ObjectID', 'UnitID'], axis = 1)
```

In [105]:
```
Fires_df2.rename(columns={"Name_left":"Name"}, inplace=True)
Fires_df2['UniqueId'] = "0B"
```

In [106]:
```
Fires_df4.rename(columns={"CaDate":"FireDate", "CaYear":"FireYear",
                          "CaMonth":"FireMonth", "CaDay":"FireDay", "AcresBurn
ed":"TotalAcres"}, inplace=True)
```

In [107]:
```
Fires_df4[Fires_df4.duplicated(['geometry'], keep=False)]
```

Out[107]:

geometry	FireDate	FireYear	FireMonth	FireDay	TotalAcres	Name	UniqueId

In [108]:
```
Fires_df = Fires_df2.append(Fires_df4)
```

In [109]:
```
Fires_df = Fires_df[~(Fires_df['TotalAcres'] == 0)]
```

In [110]:
```
Fires_df = Fires_df.sort_index()
```

In [111]:
```
Fires_df['FireYear'].describe()
```

Out[111]:
```
count    3995.000000
mean     2015.954693
std         2.810346
min      2011.000000
25%      2013.500000
50%      2017.000000
75%      2018.000000
max      2020.000000
Name: FireYear, dtype: float64
```

In [112]:
```
Fires_df = Fires_df
```

In [113]:
```
Fires_df[['TotalAcres']].describe()
```

Out[113]:

	TotalAcres
count	3.988000e+03
mean	2.983130e+03
std	2.536381e+04
min	1.356887e-03
25%	1.060945e+01
50%	3.700000e+01
75%	2.032367e+02
max	1.032699e+06

Check for duplicates by Name

In [114]:
```
import string
import re

def text_w_punc(text):
    pattern = r'[^A-Za-z ]'
    if re.search("[^0-9]", text):
        pass
    else:
        regex = re.compile(pattern)
        text = regex.sub(' ', text)
    return text

def no_whitespace(string):
    string = string.replace('fire','')
    return string.strip()

def string_lower(df, col):
    # converting all text to lowercase
    df[col] = df[col].str.lower()
    return df
```

In [115]:
```
Fires_df['Name'] = Fires_df['Name'].astype(str)
Fires_df['Name'] = Fires_df['Name'].apply(no_whitespace)
Fires_df['Name'] = Fires_df['Name'].apply(lambda x: "".join([i for i in x if i
not in string.punctuation]))
Fires_df['Name'] = Fires_df['Name'].apply(lambda x: text_w_punc(x))
Fires_df = string_lower(Fires_df, "Name")
Fires_df['Name'] = Fires_df['Name'].apply(no_whitespace)
```

In [116]:
```
Fires_df.shape
```

Out[116]:
```
(3995, 9)
```

In [117]:
```
f = Fires_df[(Fires_df.duplicated(['Name', 'FireDate'], keep=False)) &
             (Fires_df['Name']!="")].sort_values(['Name','FireDate', 'TotalAcr
es'], ascending=False)
print(f.shape)

Fires_df = Fires_df[~((Fires_df.duplicated(['Name', 'FireDate'], keep=False))
&
                      (Fires_df['Name']!=""))]
print(Fires_df.shape)
```

```
(415, 9)
(3580, 9)
```

In [118]:
```
multipolygon =f[f['UniqueId']=="0B"]
f = f[~(f['UniqueId']=="0B")]
```

In [119]:
```
Fires_df = Fires_df.append(multipolygon)
```

In [120]:
```
Fires_df.shape
```
Out[120]: (3803, 9)

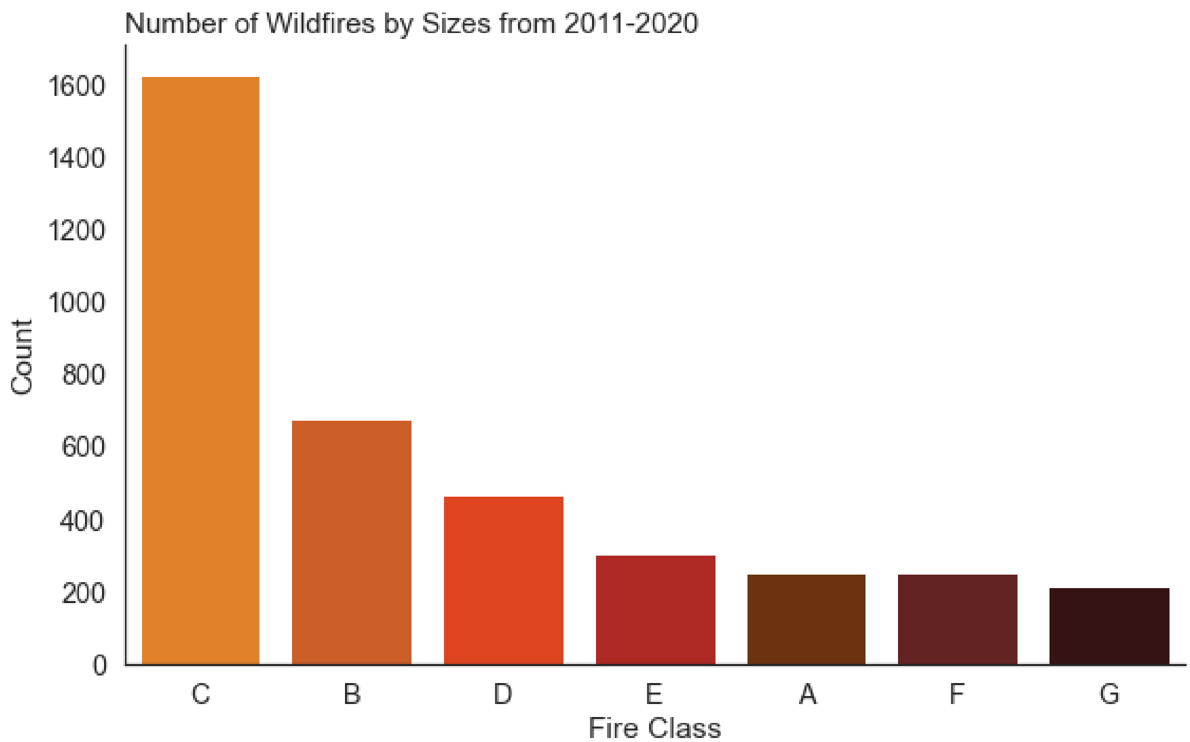
In [121]:
```
Fires_df[Fires_df.duplicated(['geometry'], keep=False)]
```
Out[121]:

FireCause	TotalAcres	geometry	FireDate	FireYear	FireMonth	FireDay	Name	UniqueId

What fire class is more common?

In [122]:
```
## Fire Class Binning
# A=greater than 0 but less than or equal to 0.25 acres
# B=0.26-9.9 acres, C=10.0-99.9 acres, D=100-299 acres
# E=300 to 999 acres, F=1000 to 4999 acres, and G=5000+ acres)

#binning method for confidence of fire.
bins = [0,.25,9.9,99.9,299,999,4999,1032699.0]
labels = ['A', 'B','C', 'D', 'E', 'F', 'G']
Fires_df['FireSize'] = pd.cut(Fires_df['TotalAcres'], bins=bins, labels=labels
)
Fires_df['FireSize']= Fires_df['FireSize'].fillna('A')
```

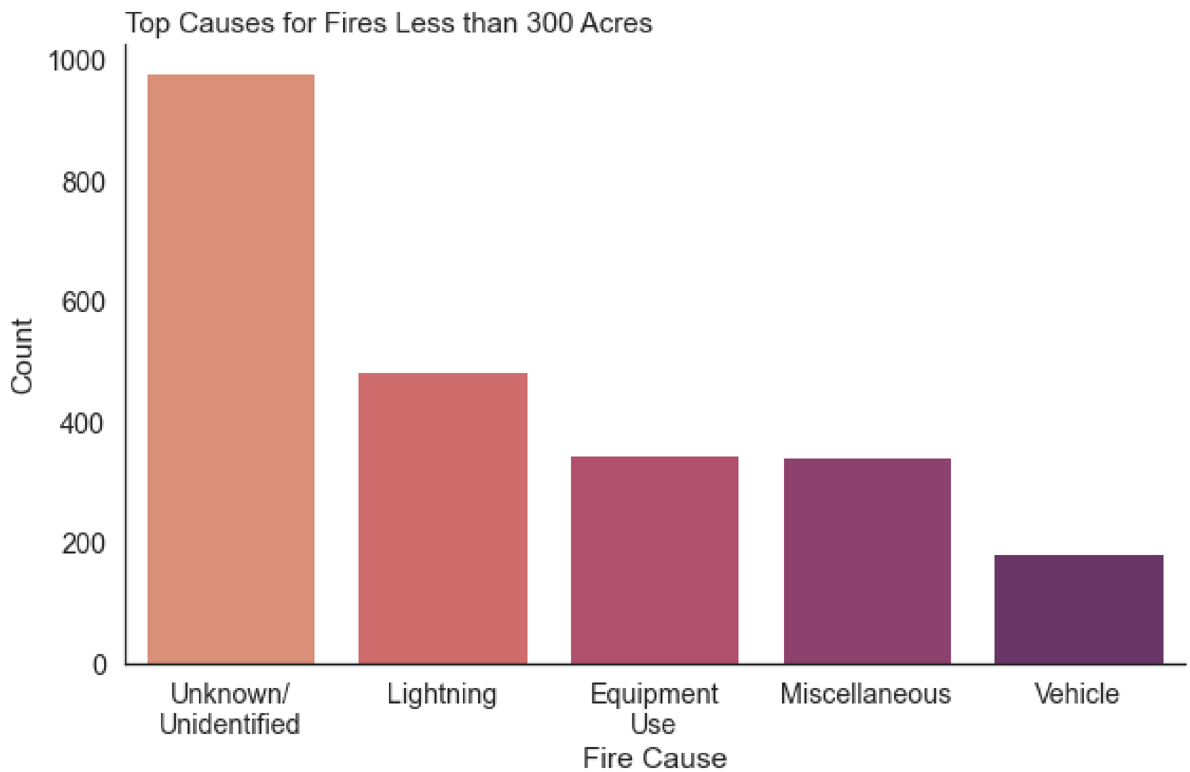
In [123]:
```
## Fire Class Binning
# A=greater than 0 but less than or equal to 0.25 acres
# B=0.26-9.9 acres, C=10.0-99.9 acres, D=100-299 acres
# E=300 to 999 acres, F=1000 to 4999 acres, and G=5000+ acres)

plt.rcParams['figure.figsize'] = [10,6]
colors = ["#FF800D", "#E8540C","#FF3000", "#C5140C", "#7A3000", "#6E1818", "#3
B0D0D"]
sns.set_palette(sns.color_palette(colors))
sns.set(font_scale = 1.3)
sns.set_style("white")


ax = sns.countplot(x=Fires_df['FireSize'],data=Fires_df, palette=sns.color_pal
ette(colors),
                   order=Fires_df['FireSize'].value_counts().index)

ax.set_title("Number of Wildfires by Sizes from 2011-2020",fontsize = 15, loc=
'left')
ax.set_xlabel("Fire Class")
ax.set_ylabel("Count")
sns.despine()

plt.show()
```



In [124]:
```
import textwrap
def wrap_labels(ax, width, break_long_words=False):
    labels = []
    for label in ax.get_xticklabels():
        text = label.get_text()
        labels.append(textwrap.fill(text, width=width,
                      break_long_words=break_long_words))
    ax.set_xticklabels(labels, rotation=0)

def plot_cause(df, title):


    plt.rcParams['figure.figsize'] = [10,6]
    sns.set(font_scale = 1.3)
    sns.set_style("white")

    ax = sns.countplot(x=df['FireCause'],data=df, palette="flare",
                       order=df['FireCause'].value_counts().iloc[:5].index)
    ax.set_title(title,fontsize = 15, loc='left')
    ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
    #ax.tick_params(axis='x', labelrotation=90)
    wrap_labels(ax, 5)
    ax.figure
    ax.set_xlabel("Fire Cause")
    ax.set_ylabel("Count")
    sns.despine()

    plt.show()
```

What are the Top Causes for Small Wildfires (less than 300 Acres)?

In [125]:
```
df = Fires_df[(Fires_df['TotalAcres'] <300)]
plot_cause(df, "Top Causes for Fires Less than 300 Acres")
```

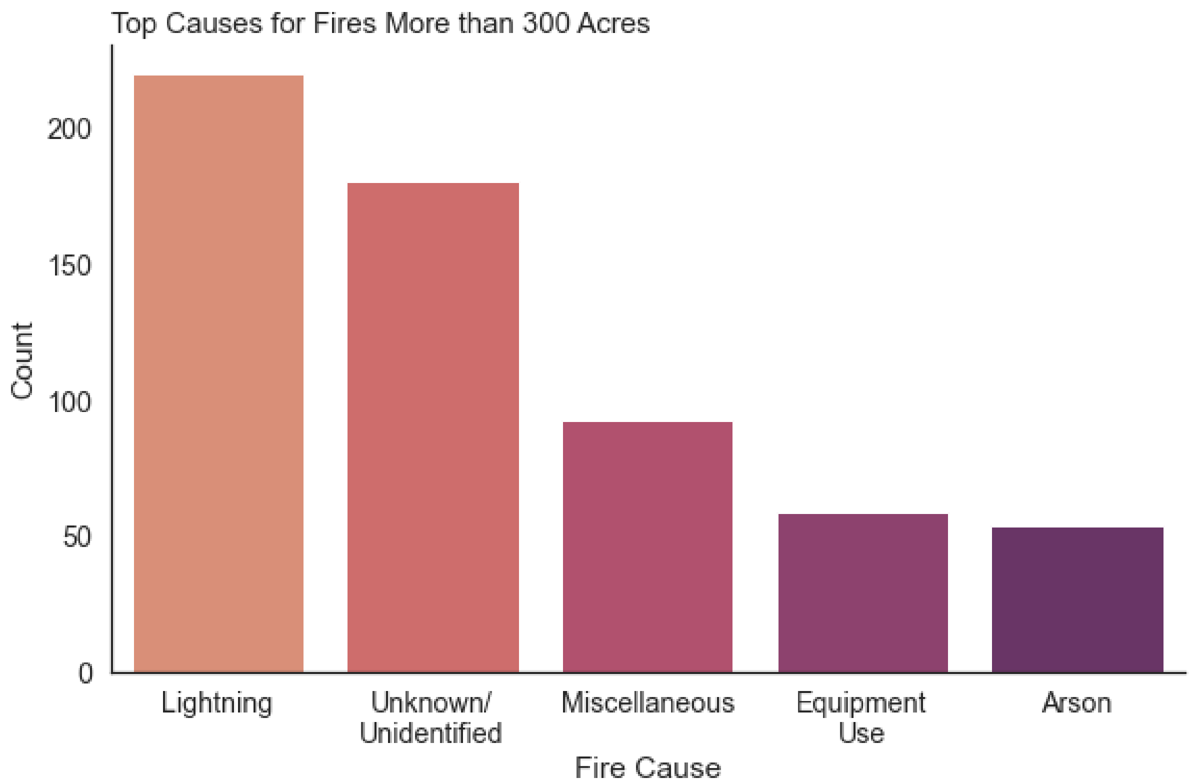


In [126]:
```
print("Number of Small fires: {}".format(df.shape[0]))
```

```
Number of Small fires: 3017
```

What are the Top Causes for Large Wildfires (greater than or 300 Acres)?

In [127]:
```
df = Fires_df[(Fires_df['TotalAcres'] >=300)]
plot_cause(df, "Top Causes for Fires More than 300 Acres")
```



In [128]:
```
print("Number of Large fires: {}".format(df.shape[0]))
```

```
Number of Large fires: 779
```

In [129]:
```
Datatype(Fires_df)
```

```
There are 3803 rows and 10 columns
FireCause              object
TotalAcres            float64
geometry             geometry
FireDate       datetime64[ns]
FireYear              float64
FireMonth             float64
FireDay               float64
Name                   object
UniqueId               object
FireSize             category
dtype: object
['TotalAcres' 'FireYear' 'FireMonth' 'FireDay']
['FireCause' 'geometry' 'FireDate' 'Name' 'UniqueId' 'FireSize']
```

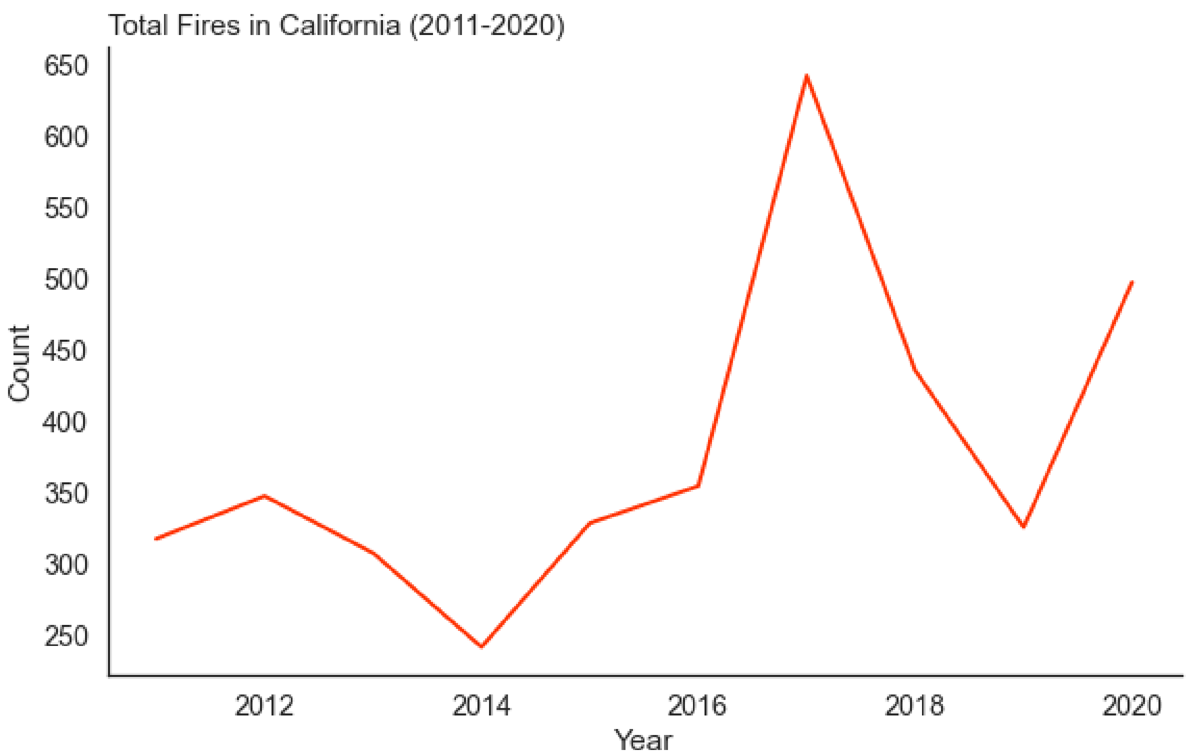
In [130]:
```
geo_fires_df = Fires_df
geo_fires_df[geo_fires_df.duplicated(['geometry'], keep=False)]
```

Out[130]:

FireCause	TotalAcres	geometry	FireDate	FireYear	FireMonth	FireDay	Name	UniqueId	Fir

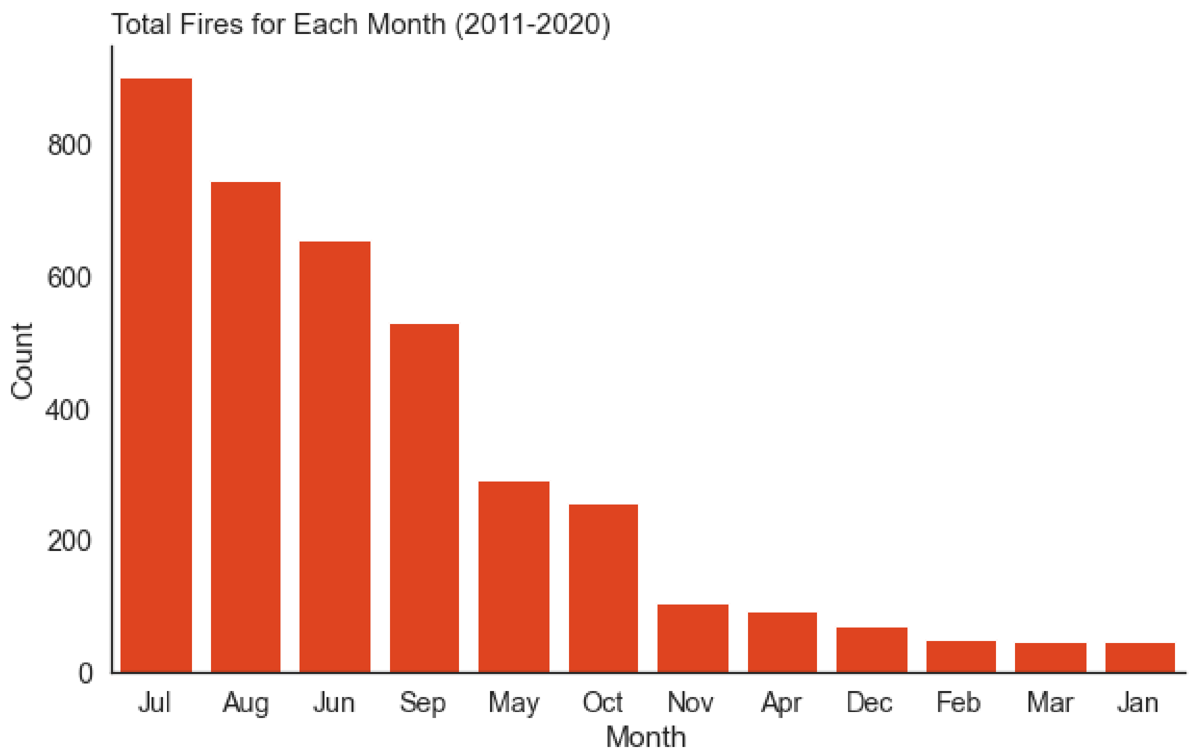
2017 had the highest number of wildfire

In [131]:
```
#ca_fires_df = ca_fires_df[ca_fires_df["ReportedAcres"] >= .25]

fire_count = pd.DataFrame(geo_fires_df['FireYear'].value_counts(sort=False))

plt.rcParams['figure.figsize'] = [10,6]
ax = sns.lineplot(data=fire_count, x=fire_count.index, y="FireYear", color="#F
F3000", linewidth = 2)


ax.set_title("Total Fires in California (2011-2020)",fontsize = 15,loc='left')
ax.set_xlabel("Year")
ax.set_ylabel("Count")
sns.despine()
plt.show()
```



Which Month had the highest fire counts over time?

In [132]:
```
sns.set(font_scale = 1.3)
sns.set_style("white")
df1 = geo_fires_df.replace({'FireMonth' : {1: 'Jan', 2 : 'Feb', 3 : 'Mar', 4:
'Apr',
                                           5: 'May', 6: 'Jun', 7: 'Jul', 8: 'Au
g',
                                           9: 'Sep', 10: 'Oct', 11: 'Nov', 12:
'Dec'}})
ax = sns.countplot(df1['FireMonth'],data=df1, color="#FF3000",
                   order=df1['FireMonth'].value_counts().index)

ax.set_title("Total Fires for Each Month (2011-2020)",fontsize = 15, loc='lef
t')
ax.set_xlabel("Month")
ax.set_ylabel("Count")
sns.despine()

plt.show()
```

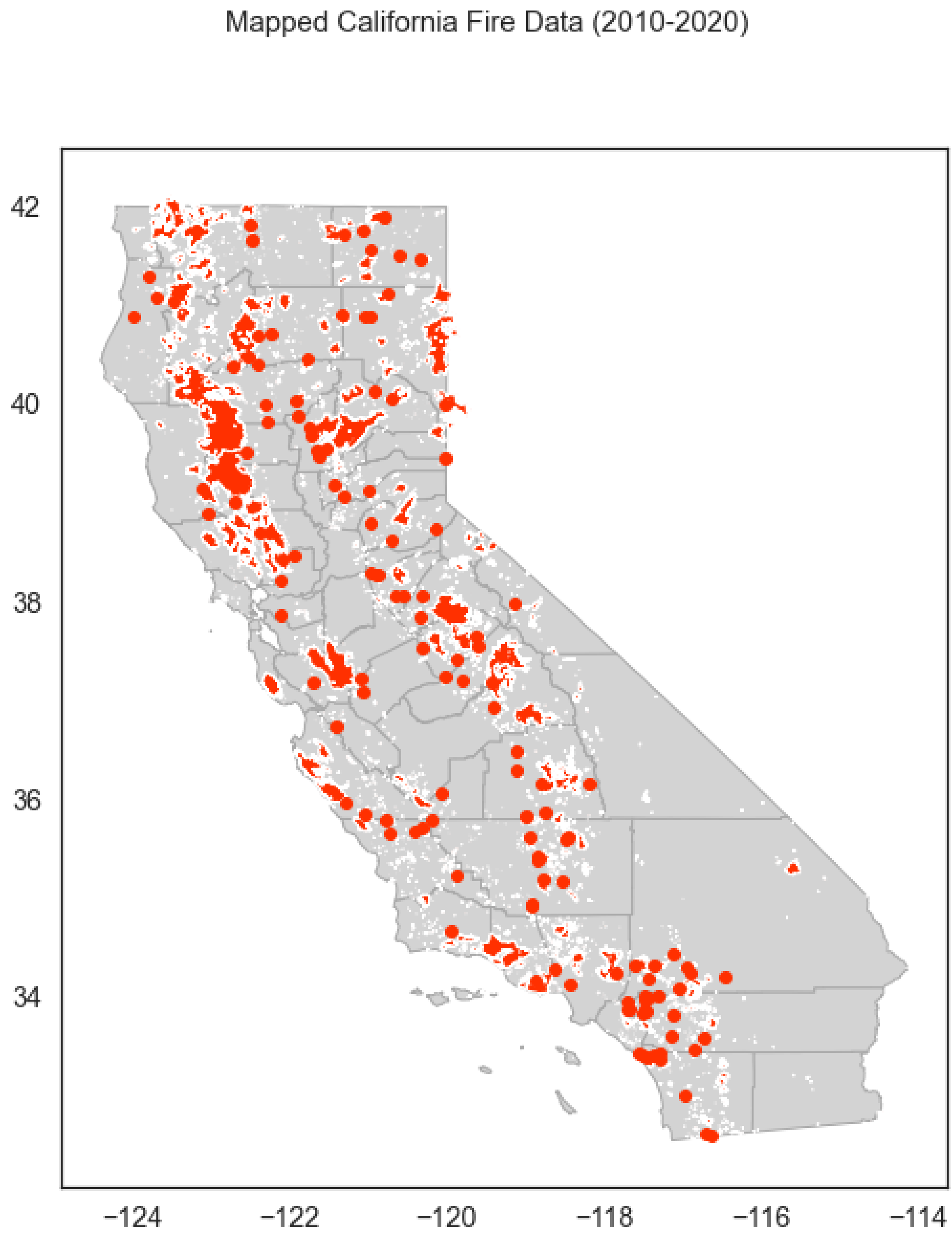


In [133]:
```
plot_gf1 = geo_fires_df.to_crs({'init': "EPSG:4326"})
```

What area of California is more wildfire prone area?

In [134]:
```
fig, ax = plt.subplots(figsize = (10,10))
fig.suptitle('Mapped California Fire Data (2010-2020)', fontsize=15)
plt.xticks([-124, -122, -120, -118, -116, -114])

USA[USA.STATEFP == '06'].plot(ax = ax, edgecolor="darkgrey", facecolor='lightg
rey')
plot_gf1.plot(ax=ax, color="#FF3000")
```

Out[134]: <AxesSubplot:>



In [135]:
```
geo_fires_df.crs
```

Out[135]:
```
<Derived Projected CRS: EPSG:3310>
Name: NAD83 / California Albers
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- name: United States (USA) - California.
- bounds: (-124.45, 32.53, -114.12, 42.01)
Coordinate Operation:
- name: California Albers
- method: Albers Equal Area
Datum: North American Datum 1983
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

In [136]:
```
def add(date, num):
    end_date = date + dt.timedelta(days=num)
    return end_date

def substract(date, num):
    end_date = date - dt.timedelta(days=num)
    return end_date
```

In [137]:
```
Dates = geo_fires_df[['FireDate']]
Dates = Dates.drop_duplicates()
```

In [138]:
```
Dates = Dates['FireDate'].tolist()
```

In [139]:
```
years = list(range(2010, 2021))
months = list(range(1, 13))
days = list(range(1, 32))
```

In [140]:
```
fires_copy = geo_fires_df
```

In [142]:
```
fires_copy.shape
```

Out[142]:
```
(3803, 10)
```

In [ ]:
```
```

Appendix A.2 MODIS Data Preparation code

In [2]:

```
import datetime as dt
from pathlib import Path
import math
import os
import sqlite3
import json
import geopandas as gpd
import pygeos
import pyproj
import shapely
import shapely.ops as ops
from shapely.geometry import Point, Polygon
from shapely.geometry.polygon import Polygon
from functools import partial

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split


from sklearn import svm
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confu

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, f_classif, mutual_info_class
from functools import partial


from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings('ignore')
```

Data Collection

In [3]:

```
## Map file
```

In [4]:
```
USA = gpd.read_file("Data/County/cb_2018_us_county_500k.shp")
USA.head()
```

Out[4]:

	STATEFP	COUNTYFP	COUNTYNS	AFFGEOID	GEOID	NAME	LSAD	ALAND
0	21	007	00516850	0500000US21007	21007	Ballard	06	639387454
1	21	017	00516855	0500000US21017	21017	Bourbon	06	750439351
2	21	031	00516862	0500000US21031	21031	Butler	06	1103571974
3	21	065	00516879	0500000US21065	21065	Estill	06	655509930
4	21	069	00516881	0500000US21069	21069	Fleming	06	902727151

2b. Load the dataset 2: NASA Active Fire Data - [https://earthdata.nasa.gov/ (https://earthdata.nasa.gov/)](https://earthdata.nasa.gov/)

In [5]:
```
current_dir = Path(os.getcwd()).absolute()
```

In [6]:

```
old_nasa_df = pd.read_csv('Data/fire_archive_M6_156000.csv') # archive da
new_nasa_df = pd.read_csv('Data/fire_nrt_M6_156000.csv') # new data

nasa_df = pd.concat([old_nasa_df, new_nasa_df]) #concatenate old and new
print(nasa_df.shape)
nasa_df.tail(4)
```

```
(1248606, 15)
```

Out[6]:

	latitude	longitude	brightness	scan	track	acq_date	acq_time	satellite	instrument
66330	36.293	-118.664	329.6	1.1	1.0	2020-09-27	1855	Terra	MODIS
66331	40.043	-123.080	337.5	1.2	1.1	2020-09-27	1855	Terra	MODIS
66332	45.726	-118.308	305.5	1.0	1.0	2020-09-27	1855	Terra	MODIS
66333	37.262	-119.443	319.4	1.0	1.0	2020-09-27	1855	Terra	MODIS

Data Preliminary Analysis

In [8]:

```
# check for missing value
def percentMissing(df):

    df_numeric = df.select_dtypes(include=[np.number])
    numeric_cols = df_numeric.columns.values

    # % of missing data
    for col in df.columns:
        # create missing indicator for features with missing data
        missing = df[col].isnull()
        pct_missing = np.mean(missing)*100
        #if pct_missing >60:
        print('{} - {}%'.format(col, round(pct_missing)))
        num_missing = np.sum(missing)
```

In [9]:

```
# Checking data type
def Datatype(df):
    # shape and data types of the data
    print("There are {} rows and {} columns".format(df.shape[0], df.shap
    print(df.dtypes)

    # select numeric columns
    df_numeric = df.select_dtypes(include=[np.number])
    numeric_cols = df_numeric.columns.values
    print(numeric_cols)

    # select non numeric columns
    df_non_numeric = df.select_dtypes(exclude=[np.number])
    non_numeric_cols = df_non_numeric.columns.values
    print(non_numeric_cols)
```

Data Exploration: MODIS Collection 6 Active Fire Data

In [10]:

```
Datatype(nasa_df)
```

```
There are 1248606 rows and 15 columns
latitude      float64
longitude     float64
brightness    float64
scan          float64
track         float64
acq_date       object
acq_time        int64
satellite      object
instrument     object
confidence      int64
version        object
bright_t31    float64
frp           float64
daynight       object
type          float64
dtype: object
['latitude' 'longitude' 'brightness' 'scan' 'track' 'acq_time'
 'confidence' 'bright_t31' 'frp' 'type']
['acq_date' 'satellite' 'instrument' 'version' 'daynight']
```

In [11]:

```
# Adding new month and day variables
nasa_df['acq_date'] = pd.to_datetime(nasa_df['acq_date'])
nasa_df.rename(columns={"acq_date":"ActiveDate"}, inplace=True)

nasa_df['ActiveYear'] = nasa_df['ActiveDate'].dt.year
nasa_df['ActiveMonth'] = nasa_df['ActiveDate'].dt.month
nasa_df['ActiveDay'] = nasa_df['ActiveDate'].dt.day
```

In [12]:
```
#binning method for confidence of fire.
bins = [0, 30,80,100]
labels = ['low', 'nominal','high']
nasa_df['ConfidenceBinned'] = pd.cut(nasa_df['confidence'], bins=bins, la
nasa_df['ConfidenceBinned']= nasa_df['ConfidenceBinned'].fillna('low')
```

In [13]:
```
# dropping version and instrument variable because it just tells us what
nasa_df = nasa_df.drop(['instrument', 'version', 'acq_time'], axis = 1)
nasa_df = nasa_df.rename(columns={'brightness': 'Brightness', 'scan': 'Sc
                                  'track': 'Track', 'longitude': 'NasaLon
                                  'satellite': 'Satellite', 'confidence'
                                  'bright_t31':'BrightT31', 'frp': 'Frp',
                                  'type': 'HotSpotType','latitude': 'Nasa

nasa_df.shape
```

Out[13]: (1248606, 16)

In [14]:
```
nasa_df.dropna(inplace=True)
```

In [15]:
```
ca_nasa_df = nasa_df[(nasa_df['NasaLatitude']<= 42) & (nasa_df['NasaLati
ca_nasa_df = ca_nasa_df[(ca_nasa_df['NasaLongitude']<= -114) & (ca_nasa_d
```

In [16]:
```
x = ca_nasa_df[ca_nasa_df['NasaLatitude']<= 42]
y = ca_nasa_df[(ca_nasa_df['NasaLatitude'] >= 42) & (ca_nasa_df['NasaLati
y = y[y['NasaLongitude'] <=-117.1]
```

In [17]:
```
ca_nasa_df = ca_nasa_df[ca_nasa_df['ActiveYear'] >=2011]
```

In [18]:
```
ca_nasa_df.shape
```

Out[18]: (114599, 16)

In [19]:
```
percentMissing(ca_nasa_df)
```

```
NasaLatitude - 0%
NasaLongitude - 0%
Brightness - 0%
Scan - 0%
Track - 0%
ActiveDate - 0%
Satellite - 0%
Confidence - 0%
BrightT31 - 0%
Frp - 0%
DayNight - 0%
HotSpotType - 0%
ActiveYear - 0%
ActiveMonth - 0%
ActiveDay - 0%
ConfidenceBinned - 0%
```

In [20]:
```
# check for duplicates in coordinates
duplicate = ca_nasa_df[ca_nasa_df.duplicated(['NasaLatitude', 'NasaLongi
duplicate
```

Out[20]:

	NasaLatitude	NasaLongitude	Brightness	Scan	Track	ActiveDate	Satellite	Confiden
587108	36.8276	-118.8827	309.6	2.6	1.5	2015-08-20	Terra	
589099	36.8276	-118.8827	406.3	1.7	1.3	2015-08-20	Aqua	1
701970	36.3855	-121.7702	306.1	3.5	1.8	2016-07-30	Terra	
702125	36.3855	-121.7702	373.7	1.0	1.0	2016-07-30	Terra	1
982192	40.7501	-122.5168	328.5	1.1	1.0	2018-08-06	Terra	1
982434	40.7501	-122.5168	321.9	1.0	1.0	2018-08-06	Aqua	1

They are not really duplicates and got detected by two separate Satellite

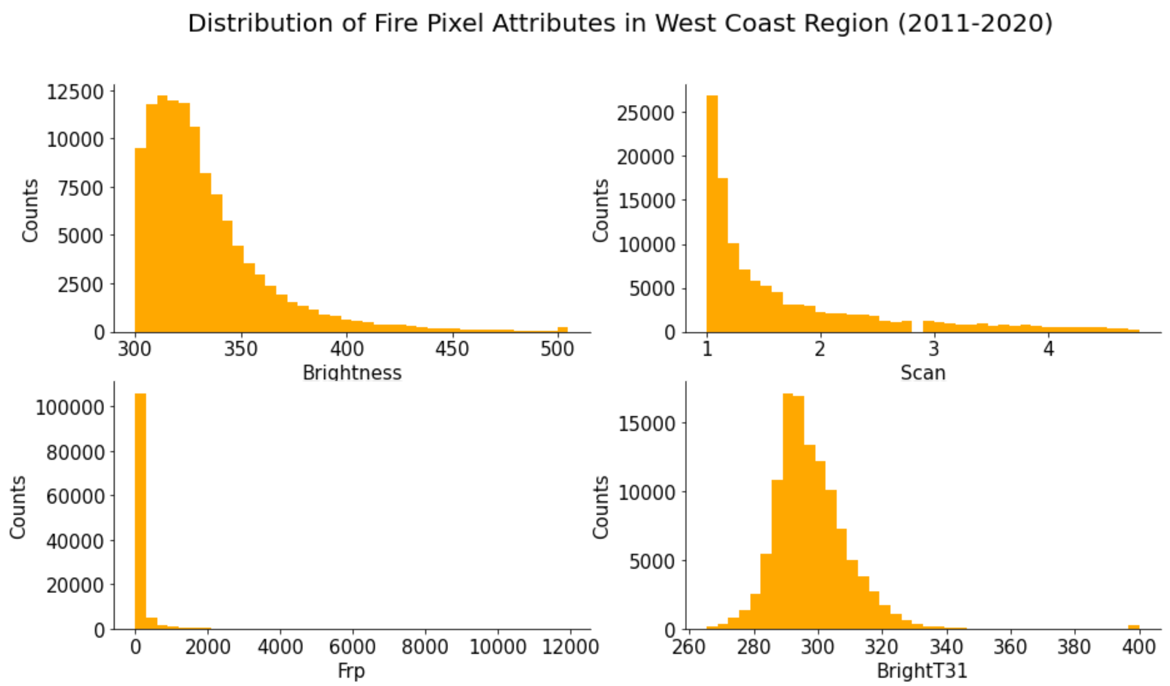
In [21]:
```
ca_nasa_df.shape
```

Out[21]: (114599, 16)

In [22]:
```
# Histograms
def histogram(xaxes, yaxes, df, x, y, nrows, color):
    plt.rcParams['figure.figsize'] = (x, y)

    fig, axes = plt.subplots(nrows = nrows, ncols = 2)
    fig.suptitle('Distribution of Fire Pixel Attributes in West Coast Reg

    # draw histograms in for loop
    axes = axes.ravel()
    for idx, ax in enumerate(axes):
        # drops NaN values
        ax.hist(df[num_features[idx]].dropna(), bins=40, color= color)
        ax.set_xlabel(xaxes[idx], fontsize=15)
        ax.set_ylabel(yaxes[idx], fontsize=15)
        ax.tick_params(axis='both', labelsize=15)
        right_side = ax.spines["right"]
        right_side.set_visible(False)
        top = ax.spines["top"]
        top.set_visible(False)

    plt.show()
```

In [23]:

```
# Specify the features of interest
num_features = ['Brightness', 'Scan', 'Frp', 'BrightT31']
xaxes = num_features
yaxes = ['Counts', 'Counts', 'Counts', 'Counts']
histogram(xaxes, yaxes, ca_nasa_df, 15,8, 2, "#ffa800")
```



In [24]:

```
geometry = [Point(xy) for xy in zip(ca_nasa_df['NasaLongitude'], ca_nasa_
geometry[:3]
```

Out[24]:

```
[<shapely.geometry.point.Point at 0x1f833064220>,
 <shapely.geometry.point.Point at 0x1f82365e740>,
 <shapely.geometry.point.Point at 0x1f81d7ff370>]
```

In [25]:

```
crs = {'init': "EPSG:4326"}
geo_nasa_df = gpd.GeoDataFrame(ca_nasa_df, crs=crs, geometry=geometry)
geo_nasa_df.head(2)
```

Out[25]:

	NasaLatitude	NasaLongitude	Brightness	Scan	Track	ActiveDate	Satellite	Confidenc
26628	34.6033	-118.3848	305.8	1.2	1.1	2011-01-06	Terra	5
26638	39.8467	-121.5210	308.2	1.1	1.0	2011-01-07	Terra	7

In [26]:

```
geo_nasa_df = geo_nasa_df.to_crs({'init': "EPSG:3310"})
```

In [27]:

```
y = ca_nasa_df[ca_nasa_df['NasaLongitude'] <=-119.5]
z = ca_nasa_df[(ca_nasa_df['NasaLatitude'] <= 39.5) & (ca_nasa_df['NasaLa
w = z[(z['NasaLongitude'] >=-119.5) & (z['NasaLongitude'] <=-116)]
v = ca_nasa_df[ca_nasa_df['NasaLatitude'] <= 36.5]

plot_df = pd.concat([y,z,w,v])
plot_df = plot_df[~((plot_df['NasaLatitude']>39) & (plot_df['NasaLongitu
plot_df = plot_df[~((plot_df['NasaLatitude']>38) & (plot_df['NasaLongitu
plot_df = plot_df[~((plot_df['NasaLatitude']>37) & (plot_df['NasaLongitu
plot_df = plot_df[~((plot_df['NasaLatitude']>36) & (plot_df['NasaLongitu


geometry = [Point(xy) for xy in zip(plot_df['NasaLongitude'], plot_df['Na
geometry[:3]
plot_df = gpd.GeoDataFrame(plot_df, crs=crs, geometry=geometry)
```

In [28]:

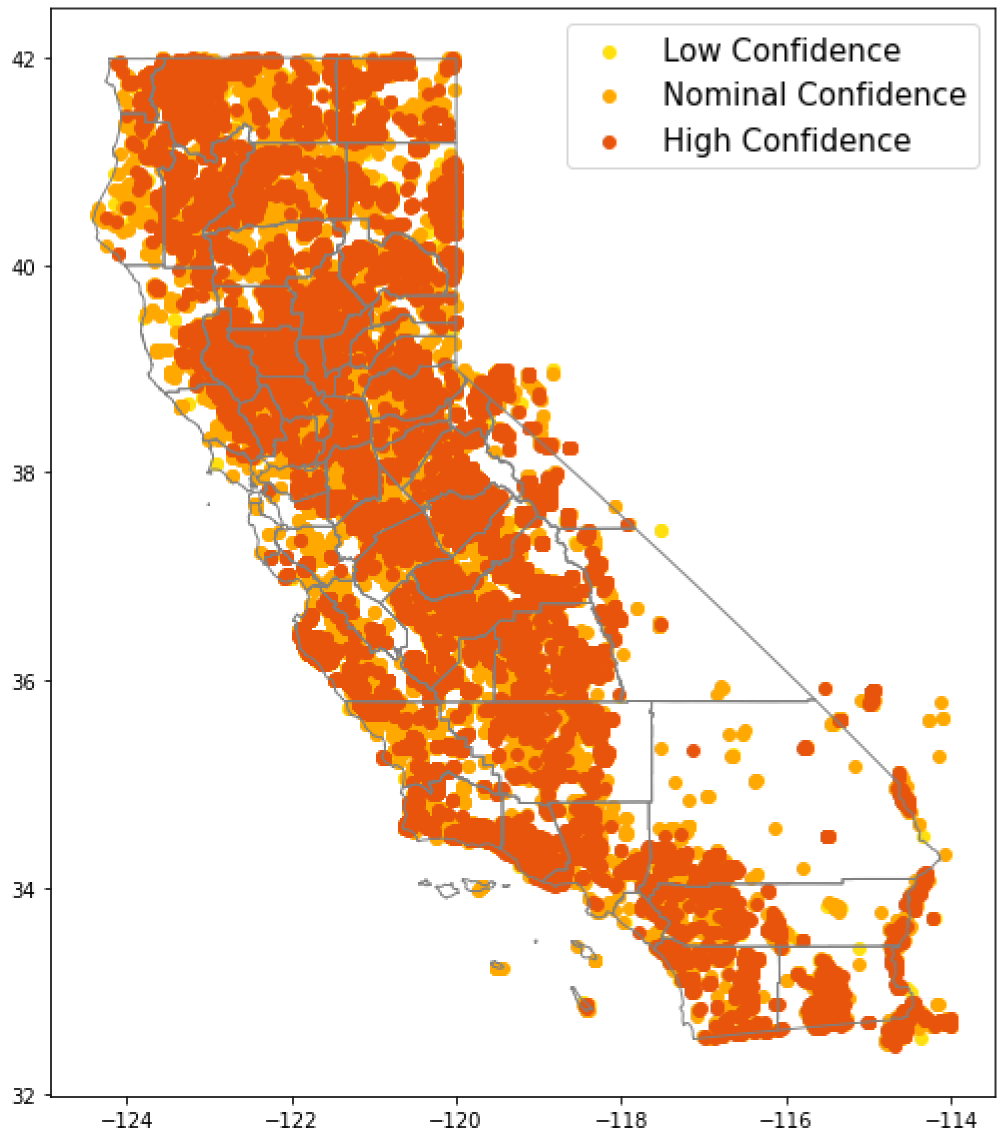
```
stcode = ['06']
```

In [29]:

```
fig, ax = plt.subplots(figsize = (10, 10))
fig.suptitle('Geospatial Plot Calfornia Fire Pixels (2011-2020)', fontsi
 
plot_df[plot_df['ConfidenceBinned'] == "low"].plot(ax=ax, color="#FFDF0D
plot_df[plot_df['ConfidenceBinned'] == "nominal"].plot(ax=ax, color="#FF/
plot_df[plot_df['ConfidenceBinned'] == "high"].plot(ax=ax, color="#E8540(
USA[USA['STATEFP'].isin(stcode)].plot(ax=ax, edgecolor="grey", facecolor
plt.legend(prop={'size':15})
```

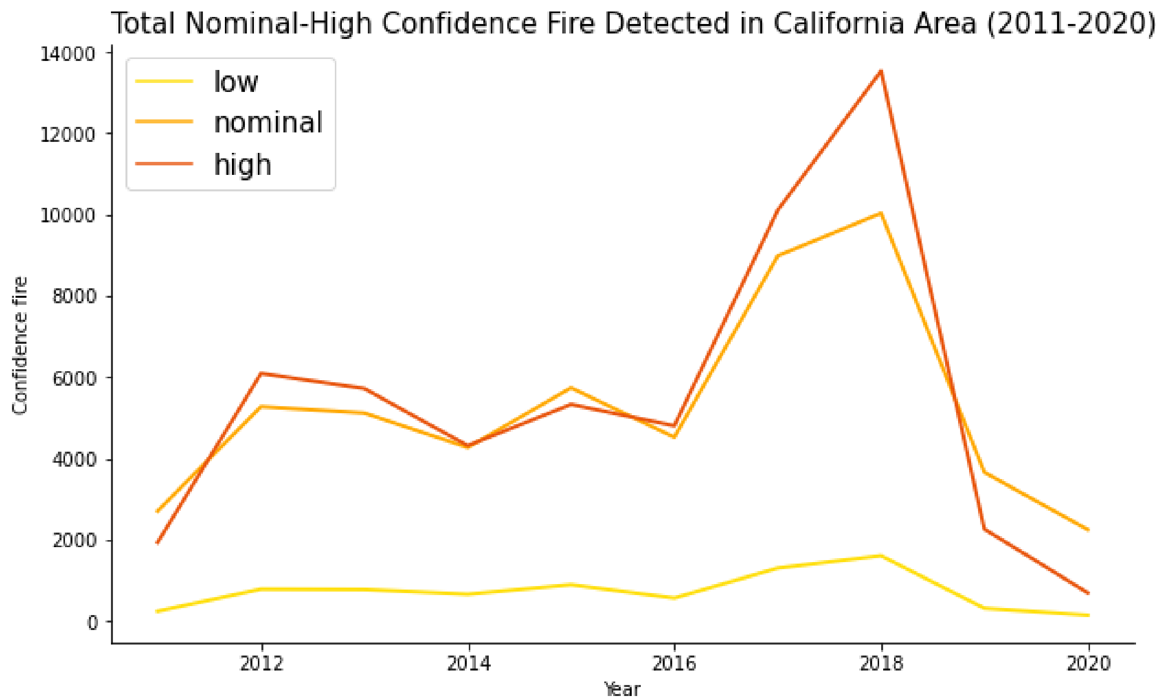
Out[29]: <matplotlib.legend.Legend at 0x1f832f50820>

Geospatial Plot Calfornia Fire Pixels (2011-2020)



In [30]:

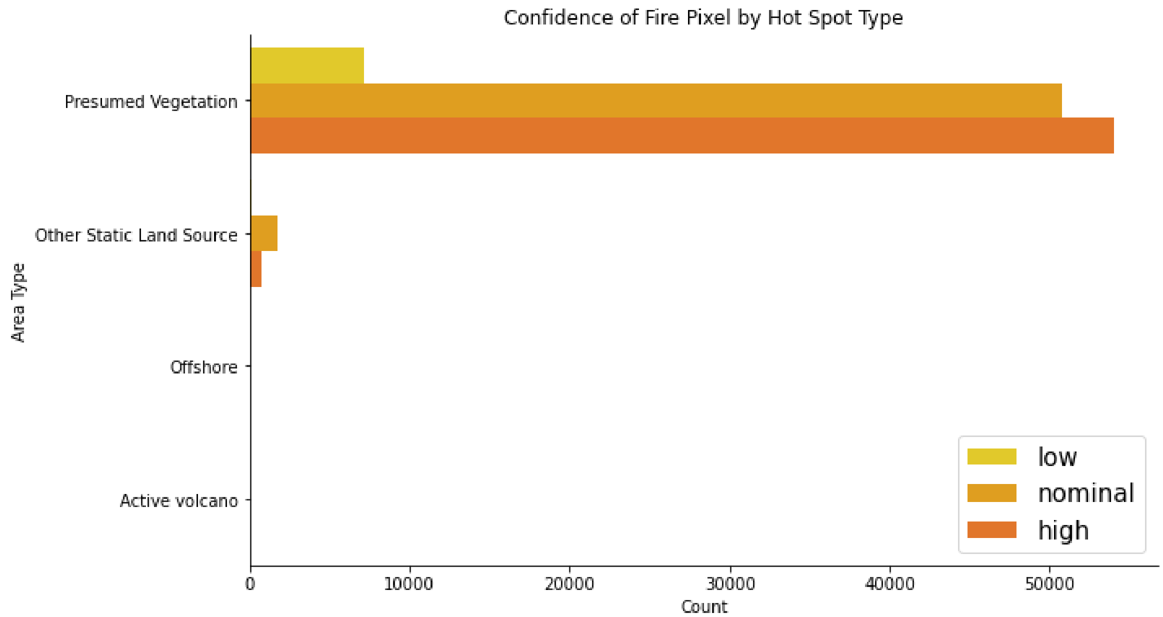
```
confidence_count = pd.DataFrame(geo_nasa_df[['ConfidenceBinned', 'ActiveY
confidence_count.columns.values[2] = 'count'
colors = ["#FFDF0D", "#FFA800","#E8540C"]
plt.rcParams['figure.figsize'] = [10,6]
ax = sns.lineplot(data=confidence_count, x="ActiveYear", y='count', hue=
                  palette=sns.color_palette(colors), linewidth=2)

ax.set_title("Total Nominal-High Confidence Fire Detected in California A
ax.set_xlabel("Year")
ax.set_ylabel("Confidence fire")
sns.despine()
plt.legend(prop={'size':15}, loc='upper left')
plt.show()

```



What Hotspot Type has the highest confidence for Fire?

In [31]:

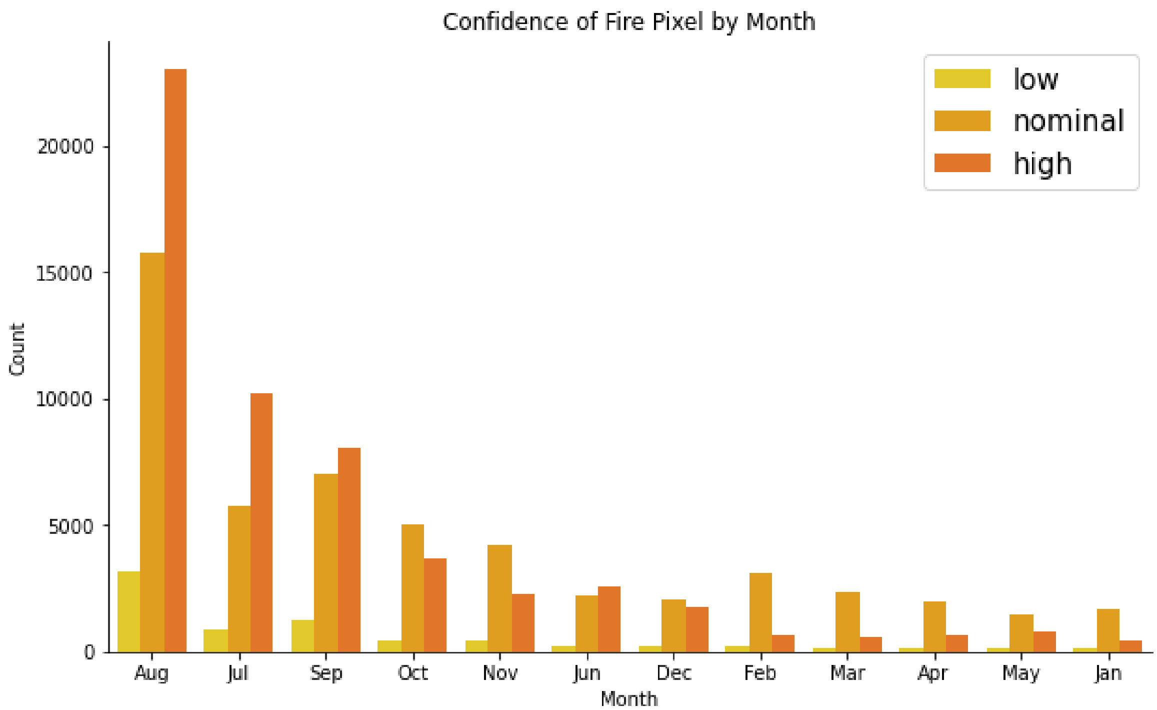
```
colors = ["#FFDF0D", "#FFA800","#FF710D"]
df1 = geo_nasa_df.replace({'HotSpotType' : {3 : 'Offshore', 2 : 'Other St
                                            1 : "Active volcano", 0 : "Presu
sns.set_palette(sns.color_palette(colors))
plt.rcParams['figure.figsize'] = [10,6]
sns.countplot(y=df1['HotSpotType'], data=df1, hue=df1['ConfidenceBinned']
                                     xlabel = "Count", ylabel = "Area Ty

sns.despine()
plt.legend(prop={'size':15}, loc='lower right')
plt.show()
```



What Month highest fires were detected?

In [32]:

```
colors = ["#FFDF0D", "#FFA800","#FF710D"]
df1 = geo_nasa_df.replace({'ActiveMonth' : {1: 'Jan', 2 : 'Feb', 3 : 'Ma
                                             5: 'May', 6: 'Jun', 7: 'Jul', 8
                                             9: 'Sep', 10: 'Oct', 11: 'Nov',
sns.set_palette(sns.color_palette(colors))
plt.rcParams['figure.figsize'] = [10,6]
sns.countplot(x=df1['ActiveMonth'], data=df1, hue=df1['ConfidenceBinned'
              order=df1['ActiveMonth'].value_counts().index).set(title =
                                      xlabel = "Month", ylabel = "Count")

sns.despine()
plt.legend(prop={'size':15}, loc='upper right')
plt.show()
```



Appendix A3: Soil and Metereological Data Preparation

In [1]:

```
import datetime as dt
from pathlib import Path
import math
import os
import sqlite3
import json
import geopandas as gpd
import pygeos
import pyproj
import shapely
import shapely.ops as ops
from shapely.geometry import Point, Polygon
from shapely.geometry.polygon import Polygon
from functools import partial

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split


from sklearn import svm
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confu

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, f_classif, mutual_info_class
from functools import partial


from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings('ignore')
```

Data Collection

In [2]:

```
Drought1 = pd.read_csv('Data/DroughtData2010-2011.csv') # archive data
Drought2 = pd.read_csv('Data/DroughtData2012-2020.csv') # new data Droug
Drought3 = pd.read_csv('Data/DroughtData2000.csv')
```

In [3]:
```
soil_df = pd.read_csv('Data/soil_data.csv') # new data
```

In [4]:
```
# Checking data type
def Datatype(df):
    # shape and data types of the data
    print("There are {} rows and {} columns".format(df.shape[0], df.shap
    print(df.dtypes)

    # select numeric columns
    df_numeric = df.select_dtypes(include=[np.number])
    numeric_cols = df_numeric.columns.values
    print(numeric_cols)

    # select non numeric columns
    df_non_numeric = df.select_dtypes(exclude=[np.number])
    non_numeric_cols = df_non_numeric.columns.values
    print(non_numeric_cols)
```

1c. Data Preliminary Analysis: Drought Data (2010-2020)

In [5]:
```
# Adding new month and day variables
Drought3['date'] = pd.to_datetime(Drought3['date'])
Drought3.rename(columns={"date":"DroughtDate"}, inplace=True)

Drought3['DroughtYear'] = Drought3['DroughtDate'].dt.year
Drought3['DroughtMonth'] = Drought3['DroughtDate'].dt.month
Drought3['DroughtDay'] = Drought3['DroughtDate'].dt.day
```

In [6]:
```
df1 = Drought3[Drought3['DroughtYear'] >=2011]
```

In [7]:
```
df1['DroughtYear'] = df1['DroughtYear'].astype(int)
df1['DroughtMonth'] = df1['DroughtMonth'].astype(int)
df1['DroughtDay'] = df1['DroughtDay'].astype(int)
```

In [8]:
```
drought_df = pd.concat([Drought1, Drought2]) #concatenate old and new da
print(drought_df.shape)
drought_df.tail(4)
```

(4540788, 21)

Out[8]:

	fips	date	PRECTOT	PS	QV2M	T2M	T2MDEW	T2MWET	T2M_MAX	T2M_M
2271944	56043	2020-12-28	0.00	83.04	1.82	-7.31	-12.06	-9.68	-1.48	-11.
2271945	56043	2020-12-29	0.00	82.78	1.87	-7.38	-11.79	-9.59	-0.88	-11.
2271946	56043	2020-12-30	0.01	82.87	1.57	-6.40	-13.94	-10.17	1.33	-12.
2271947	56043	2020-12-31	0.00	82.82	2.13	-3.83	-10.12	-6.98	2.16	-8.

4 rows × 21 columns

In [9]:
```
# Adding new month and day variables
drought_df['date'] = pd.to_datetime(drought_df['date'])
drought_df.rename(columns={"date":"DroughtDate"}, inplace=True)

drought_df['DroughtYear'] = drought_df['DroughtDate'].dt.year
drought_df['DroughtMonth'] = drought_df['DroughtDate'].dt.month
drought_df['DroughtDay'] = drought_df['DroughtDate'].dt.day
```

In [10]:
```
drought_df['DroughtYear'] = drought_df['DroughtYear'].astype(int)
drought_df['DroughtMonth'] = drought_df['DroughtMonth'].astype(int)
drought_df['DroughtDay'] = drought_df['DroughtDay'].astype(int)
```

In [11]:
```
df = pd.concat([drought_df, df1]) #concatenate old and new data
print(df.shape)
df.tail(4)
```

(11353524, 24)

Out[11]:

	fips	DroughtDate	PRECTOT	PS	QV2M	T2M	T2MDEW	T2MWET	T2M_MAX
19300676	56043	2016-12-28	0.02	83.33	1.41	-8.71	-14.10	-13.84	-2.49
19300677	56043	2016-12-29	0.00	83.75	1.59	-7.96	-13.30	-13.03	0.42
19300678	56043	2016-12-30	1.22	82.49	2.63	-2.94	-7.40	-7.33	3.76
19300679	56043	2016-12-31	0.44	82.19	1.75	-7.56	-11.98	-11.82	-0.95

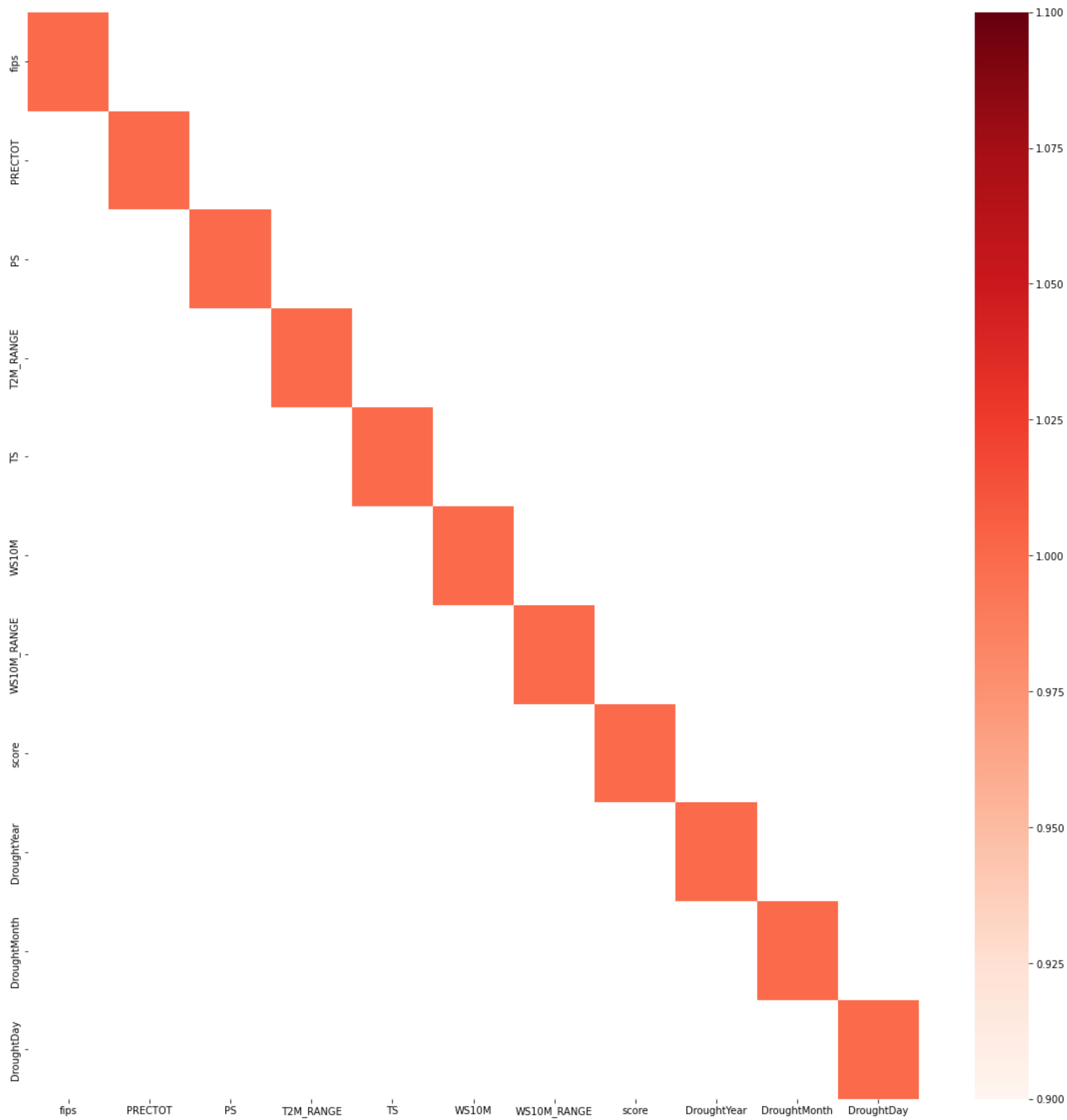
4 rows × 24 columns

In [28]:
```
df1 = df.drop(['T2MDEW', 'T2M_MAX', 'T2MWET', 'T2M_MIN', 'WS10M_MAX','WS
               'WS50M', 'WS50M_RANGE', 'QV2M', 'T2M'], axis = 1)
```

In [29]:

```
plt.figure(figsize=(20,20))
corr = df1.corr()
kot = corr[corr>=.75]
sns.heatmap(kot, cmap="Reds")
```

Out[29]: <AxesSubplot:>



In [12]:

```
soil_df.head(2)
```

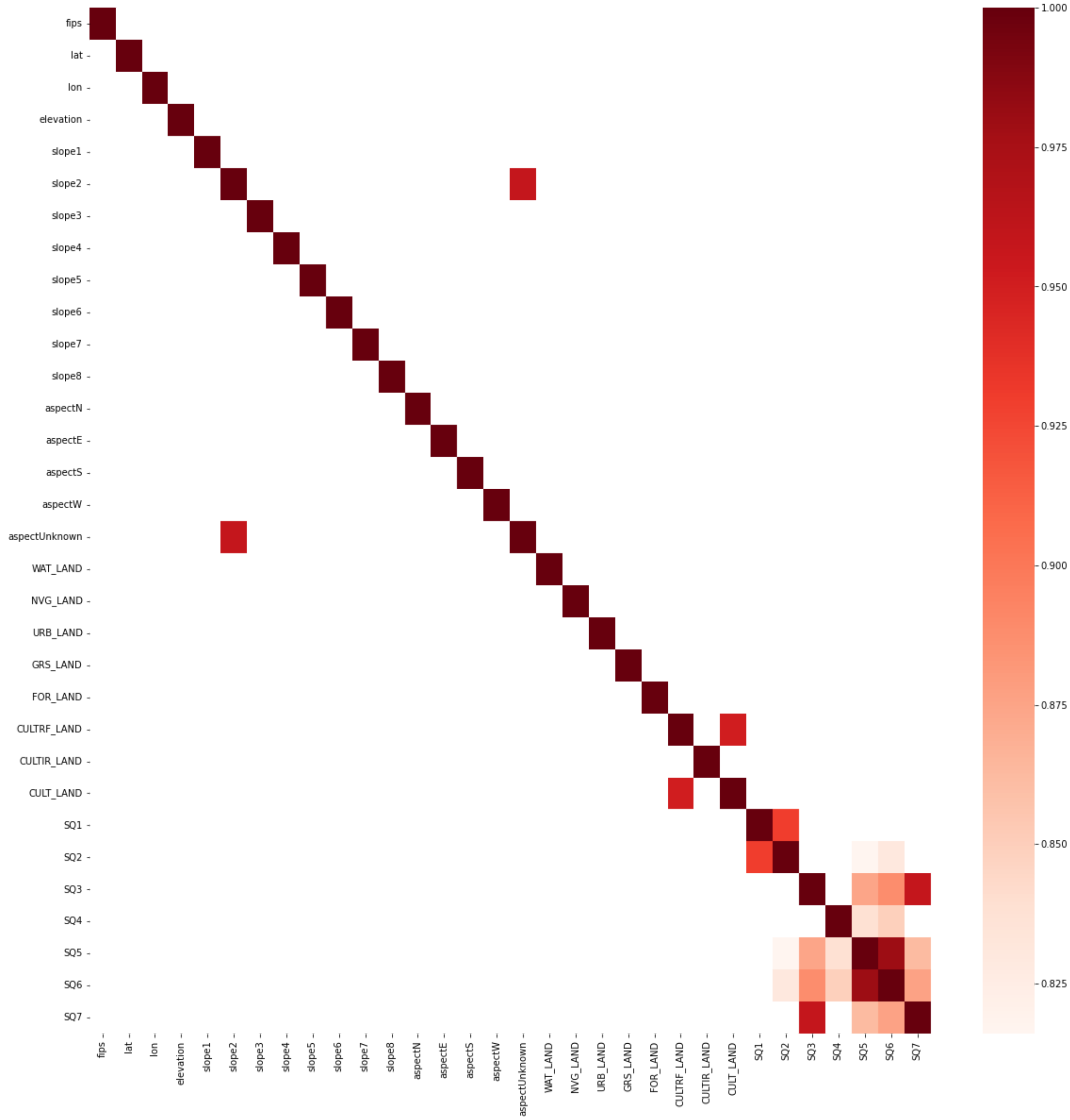
Out[12]:

	fips	lat	lon	elevation	slope1	slope2	slope3	slope4	slope5	slope6	...
0	1001	32.536382	-86.644490	63	0.0419	0.2788	0.2984	0.2497	0.1142	0.0170	...
1	1005	31.870670	-85.405456	146	0.0158	0.1868	0.5441	0.2424	0.0106	0.0003	...

2 rows × 32 columns

In [15]:
```python
plt.figure(figsize=(20,20))
corr = soil_df.corr()
kot = corr[corr>=.8]
sns.heatmap(kot, cmap="Reds")

```

Out[15]: <AxesSubplot:>

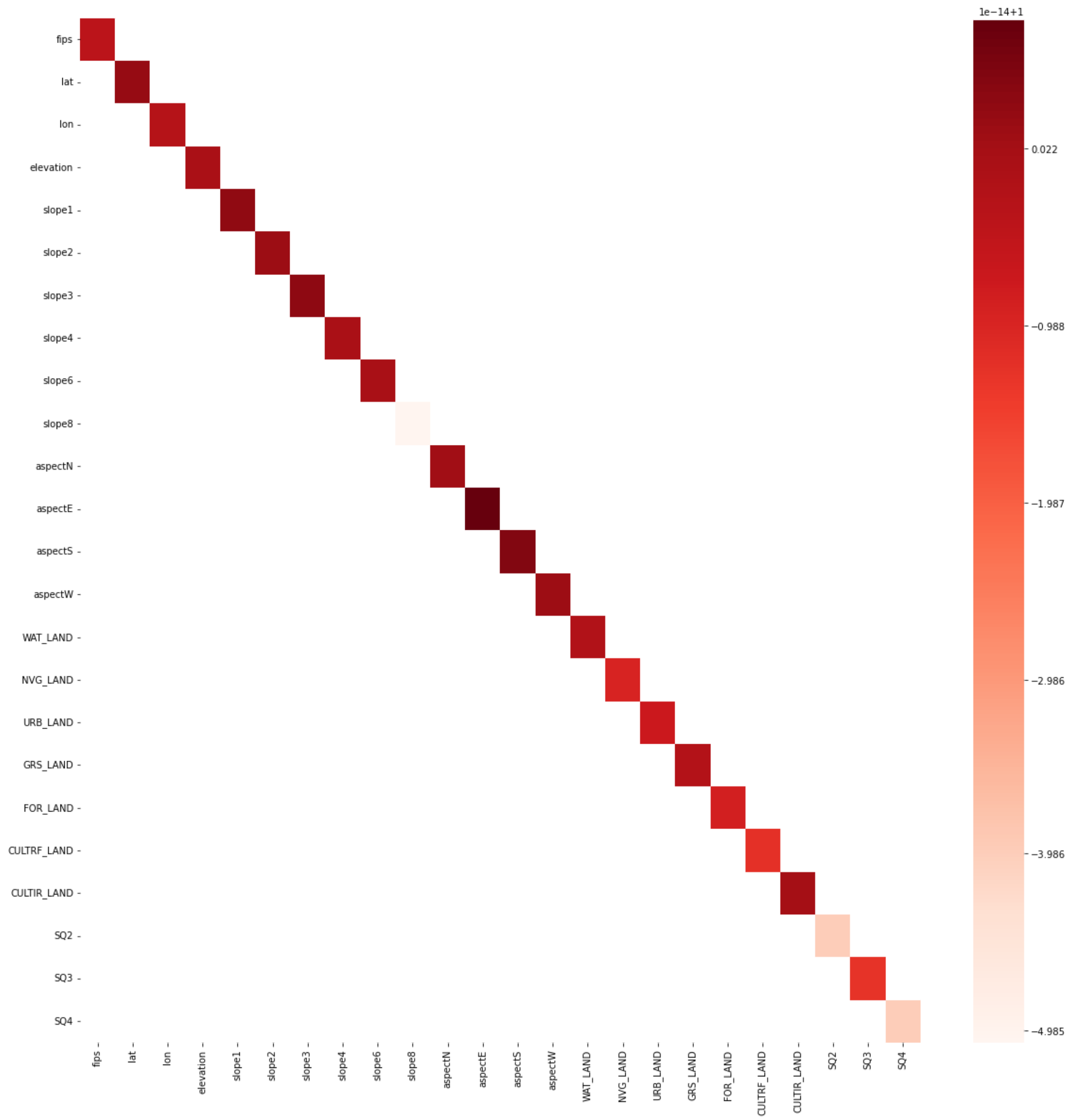


In [20]:
```python
df2 = soil_df.drop(['aspectUnknown', 'CULT_LAND', 'SQ1', 'SQ5', 'SQ6', '
```

In [21]:

```
plt.figure(figsize=(20,20))
corr = df2.corr()
kot = corr[corr>=.75]
sns.heatmap(kot, cmap="Reds")
```

Out[21]: <AxesSubplot:>



In [30]:

```
drought_soil = pd.merge(df1, df2, how='left', on=['fips'])
```

In [31]:

```
drought_soil = drought_soil[(drought_soil['lat']<= 42) & (drought_soil['
drought_soil = drought_soil[(drought_soil['lon']<= -114) & (drought_soil
```

In [32]:

```
drought_soil = drought_soil[(drought_soil['DroughtYear'] >=2011) & droug
```

In [33]:
```
drought_soil['DroughtYear'].describe()
```

Out[33]:
```
count    277628.000000
mean       2015.500411
std           2.872668
min        2011.000000
25%        2013.000000
50%        2016.000000
75%        2018.000000
max        2020.000000
Name: DroughtYear, dtype: float64
```

In [34]:
```
drought_soil.to_csv('Data/Drought_Soil.csv')
```

In [ ]:
```

```

Appendix A3: Soil and Metereological Data EDA

In [1]:

```
import datetime as dt
from pathlib import Path
import math
import os
import sqlite3
import json
import geopandas as gpd
import pygeos
import pyproj
import shapely
import shapely.ops as ops
from shapely.geometry import Point, Polygon
from shapely.geometry.polygon import Polygon
from functools import partial

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split


from sklearn import svm
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confu

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, f_classif, mutual_info_class
from functools import partial


from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings('ignore')
```

Data Collection: Soil and Metereological

USA Shape File [https://www.census.gov/geographies/mapping-files/time-series/geo/carto-boundary-file.html (https://www.census.gov/geographies/mapping-files/time-series/geo/carto-boundary-file.html)](https://www.census.gov/geographies/mapping-files/time-series/geo/carto-boundary-file.html)

In [2]:

```
USA = gpd.read_file("Data/County/cb_2018_us_county_500k.shp")
USA.head()
```

Out[2]:

	STATEFP	COUNTYFP	COUNTYNS	AFFGEOID	GEOID	NAME	LSAD	ALAND
0	21	007	00516850	0500000US21007	21007	Ballard	06	639387454
1	21	017	00516855	0500000US21017	21017	Bourbon	06	750439351
2	21	031	00516862	0500000US21031	21031	Butler	06	1103571974
3	21	065	00516879	0500000US21065	21065	Estill	06	655509930
4	21	069	00516881	0500000US21069	21069	Fleming	06	902727151

3a. Load the dataset 3: US Drought Data - https://power.larc.nasa.gov/ (https://power.larc.nasa.gov/)

In [5]:

```
drought_soil = pd.read_csv('Data/Drought_Soil.csv')
```

3b. Daily weather summary data https://www.noaa.gov/ (https://www.noaa.gov/)

In [3]:

```
# import necessary libraries
import glob

# use glob to get all the csv files
# in the folder
current_dir = Path(os.getcwd()).absolute()
data_dir = current_dir.joinpath('Data')
weather_dir = data_dir.joinpath('weather')
csv_files = glob.glob(os.path.join(weather_dir, "*.csv"))


# loop over the list of csv files
df_list = []
for f in csv_files:

    # read the csv file
    df = pd.read_csv(f)
    df_list.append(df)
```

In [4]:

```
ca_daily_df = []
for i in df_list:
    df = pd.DataFrame(i)
    df = df.dropna()
    ca_daily_df.append(df)
ca_daily_df = pd.concat(ca_daily_df)
```

Data Preliminary Analysis

In [6]:

```
# check for missing value
def percentMissing(df):

    df_numeric = df.select_dtypes(include=[np.number])
    numeric_cols = df_numeric.columns.values

    # % of missing data
    for col in df.columns:
        # create missing indicator for features with missing data
        missing = df[col].isnull()
        pct_missing = np.mean(missing)*100
        #if pct_missing >60:
        print('{} - {}%'.format(col, round(pct_missing)))
        num_missing = np.sum(missing)
```

In [7]:
```
# Checking data type
def Datatype(df):
    # shape and data types of the data
    print("There are {} rows and {} columns".format(df.shape[0], df.shap
    print(df.dtypes)

    # select numeric columns
    df_numeric = df.select_dtypes(include=[np.number])
    numeric_cols = df_numeric.columns.values
    print(numeric_cols)

    # select non numeric columns
    df_non_numeric = df.select_dtypes(exclude=[np.number])
    non_numeric_cols = df_non_numeric.columns.values
    print(non_numeric_cols)
```

Data Exploration

In [8]:
```
drought_soil.head(4)
drought_soil = drought_soil.drop(['Unnamed: 0', 'score'], axis = 1)
```

In [9]:
```
drought_soil['DroughtYear'].describe()
```

Out[9]:
```
count    277628.000000
mean       2015.500411
std           2.872668
min        2011.000000
25%        2013.000000
50%        2016.000000
75%        2018.000000
max        2020.000000
Name: DroughtYear, dtype: float64
```

In [14]:
```
y = drought_soil[drought_soil['lon'] <=-119.5]
z = drought_soil[(drought_soil['lat'] <= 39.5) & (drought_soil['lat'] >=
w = z[(z['lon'] >=-119.5) & (z['lon'] <=-116)]
v = drought_soil[drought_soil['lat'] <= 36.5]
crs = {'init': "EPSG:4326"}
plot_df0 = pd.concat([y,z,w,v])
plot_df0 = plot_df0[~((plot_df0['lat']>39) & (plot_df0['lon']>-120))]
plot_df0 = plot_df0[~((plot_df0['lat']>37.6) & (plot_df0['lon']>-118.5))

geometry = [Point(xy) for xy in zip(plot_df0['lon'], plot_df0['lat'])]
geometry[:3]
geo_soil_df = gpd.GeoDataFrame(plot_df0, crs=crs, geometry=geometry)
```

In [15]:
```
soil_df = geo_soil_df[['fips','lat','lon','elevation', 'slope1','slope2',
                       'slope4','slope6','slope8','aspectN','aspectE', '
                       'aspectW','WAT_LAND','NVG_LAND','URB_LAND','GRS_LA
                       'CULTRF_LAND','CULTIR_LAND','SQ2','SQ3', 'SQ4', '
soil_df.shape
```

Out[15]: (361647, 25)

In [16]:
```
soil_df = soil_df[~soil_df.duplicated(['fips'], keep='first')]
soil_df.shape
```

Out[16]: (61, 25)

In [17]:
```
soil_df[['lat','lon','elevation', 'slope1','slope2','slope3','slope4','s
```

Out[17]:

	lat	lon	elevation	slope1	slope2	slope3	slope4	sl
count	61.000000	61.000000	61.000000	61.000000	61.000000	61.000000	61.000000	61.00
mean	37.761497	-120.526784	643.426230	0.041205	0.187852	0.117167	0.118982	0.24
std	2.183253	2.200278	678.960148	0.076777	0.267800	0.112690	0.102348	0.18
min	33.023604	-123.980998	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
25%	36.561977	-122.007205	106.000000	0.000400	0.005300	0.029800	0.044400	0.02
50%	38.021451	-120.773446	444.000000	0.001600	0.023200	0.082700	0.096400	0.27
75%	39.177739	-119.749852	825.000000	0.044500	0.322700	0.175900	0.184900	0.40
max	41.749903	-114.038793	2630.000000	0.343100	0.745400	0.561200	0.483300	0.55

In [18]:
```
soil_df[['aspectN','aspectE', 'aspectS', 'aspectW']].describe()
```

Out[18]:

	aspectN	aspectE	aspectS	aspectW
count	61.000000	61.000000	61.000000	61.000000
mean	0.151067	0.165772	0.192116	0.230715
std	0.093424	0.096897	0.117666	0.137050
min	0.000000	0.000000	0.000000	0.000000
25%	0.064200	0.089500	0.070300	0.128400
50%	0.163600	0.173000	0.201800	0.238000
75%	0.240400	0.226800	0.289000	0.317900
max	0.348700	0.397800	0.473800	0.560200

In [19]:

```
1 soil_df[['WAT_LAND','NVG_LAND','URB_LAND','GRS_LAND', 'FOR_LAND', 'CULTRI
```

Out[19]:

	WAT_LAND	NVG_LAND	URB_LAND	GRS_LAND	FOR_LAND	CULTRF_LAND	CULTIR_
count	61.000000	61.000000	61.000000	61.000000	61.000000	61.000000	61.(
mean	1.033192	7.304989	2.745081	18.985231	47.190747	1.722027	16.1
std	5.806408	19.514630	13.599561	14.832669	31.989077	2.404207	28.7
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.(
25%	0.000000	0.000000	0.000000	8.869439	18.155117	0.000000	0.(
50%	0.000000	0.000000	0.055821	15.587294	49.677391	0.272293	0.4
75%	0.000000	2.083883	0.429140	29.065670	77.980881	3.188647	15.7
max	44.035000	78.871132	99.955193	58.796833	90.971321	9.187908	99.9

In [20]:

```
1 soil_df[['SQ2','SQ3', 'SQ4']].describe() #### these are discrete variabl
```

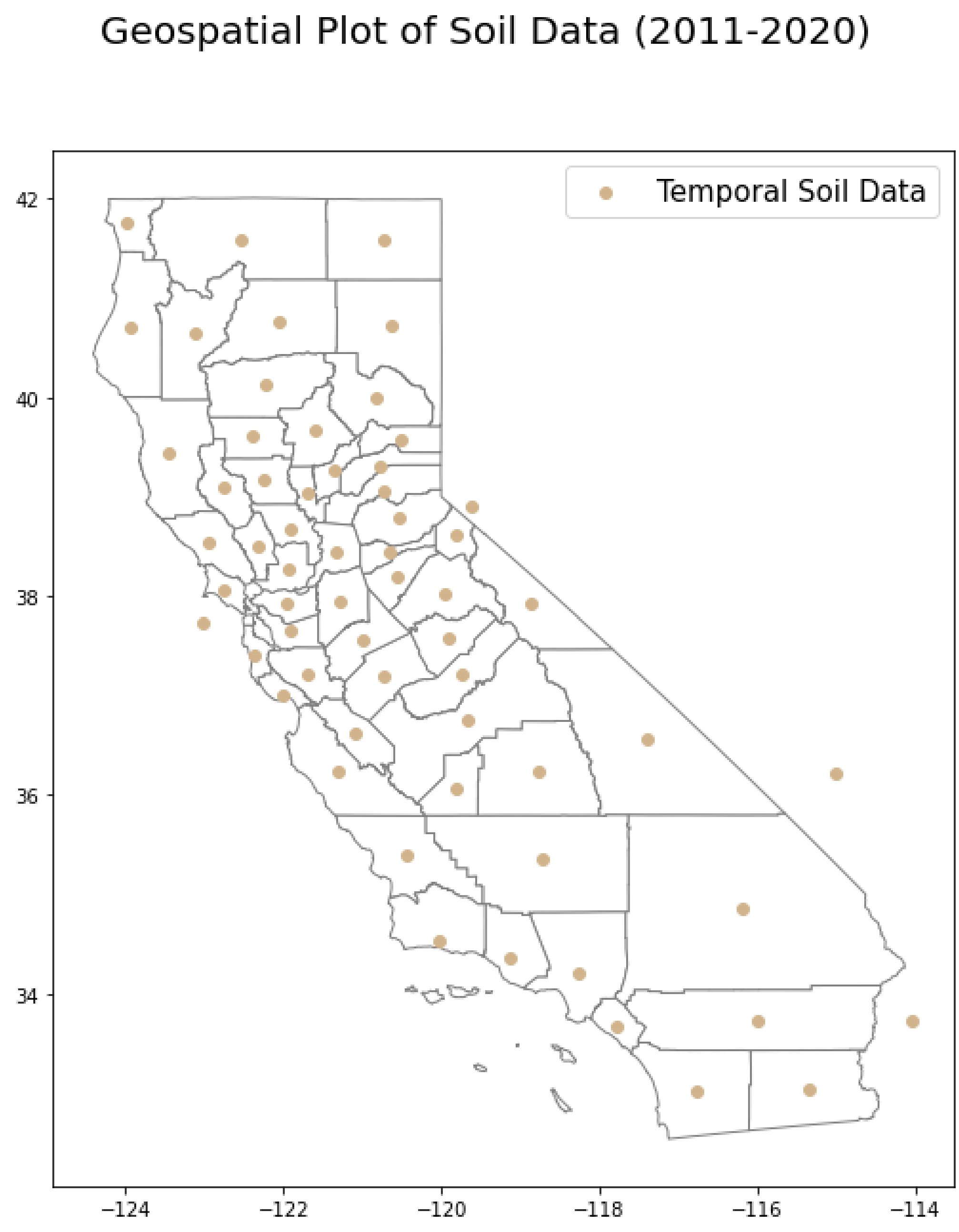
Out[20]:

	SQ2	SQ3	SQ4
count	61.000000	61.000000	61.000000
mean	1.360656	1.573770	1.163934
std	1.155174	1.257763	1.113258
min	0.000000	0.000000	0.000000
25%	1.000000	1.000000	1.000000
50%	1.000000	1.000000	1.000000
75%	1.000000	2.000000	1.000000
max	7.000000	7.000000	7.000000

In [21]:

```
fig, ax = plt.subplots(figsize = (10, 10))
fig.suptitle('Geospatial Plot of Soil Data (2011-2020)', fontsize=20)

USA[USA['STATEFP'] == '06'].plot(ax=ax, edgecolor="grey", facecolor="whi
soil_df.plot(ax=ax, color="tan", label="Temporal Soil Data")
plt.legend(prop={'size':15})
```

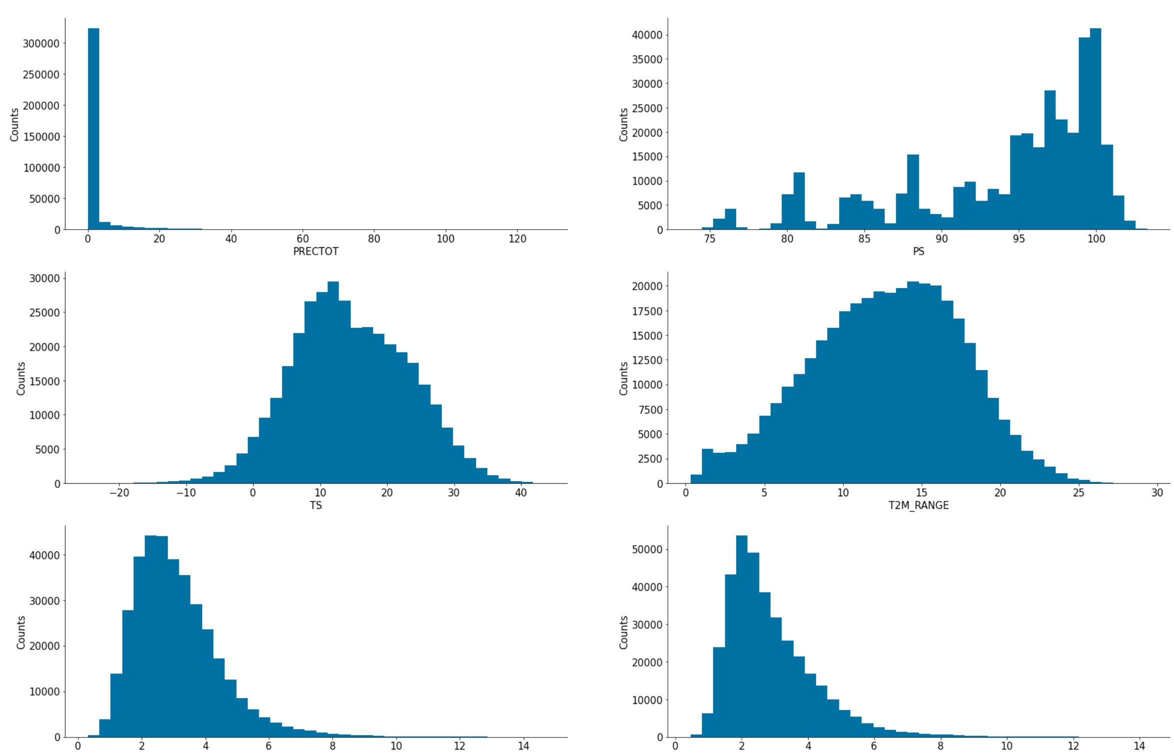
Out[21]: <matplotlib.legend.Legend at 0x1f95eeace20>



In [22]:
```
# Histograms
def histogram(xaxes, df, x, y, nrows, color):
    plt.rcParams['figure.figsize'] = (x, y)

    fig, axes = plt.subplots(nrows = nrows, ncols = 2)
    fig.suptitle('Distribution of Meteorological Indicators in West Coast

    # draw histograms in for loop
    axes = axes.ravel()
    for idx, ax in enumerate(axes):
        # drops NaN values
        ax.hist(df[num_features[idx]].dropna(), bins=40, color= color)
        ax.set_xlabel(xaxes[idx], fontsize=15)
        ax.set_ylabel('Counts', fontsize=15)
        ax.tick_params(axis='both', labelsize=15)
        right_side = ax.spines["right"]
        right_side.set_visible(False)
        top = ax.spines["top"]
        top.set_visible(False)

    plt.show()
```

In [23]:
```
# Specify the features of interest
num_features = ['PRECTOT', 'PS','TS', 'T2M_RANGE', 'WS10M_RANGE', 'WS10M
xaxes = num_features
histogram(xaxes, geo_soil_df, 30,20, 3, "#0071A3")
```

Distribution of Meteorological Indicators in West Coast Region (2011-2020)



In [24]:

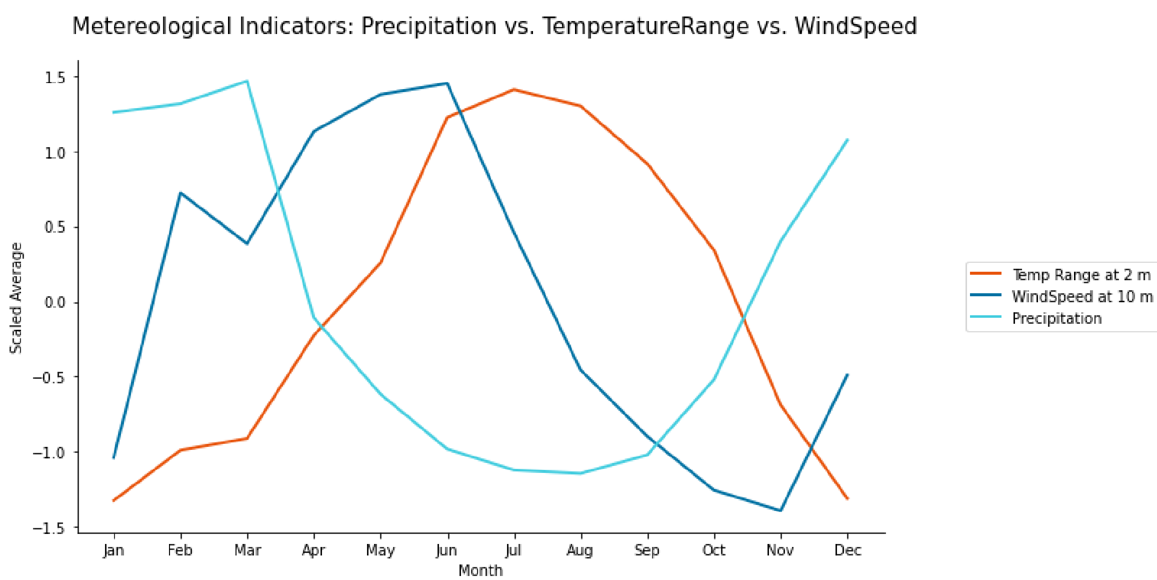
```
grouped = geo_soil_df.groupby(by=['DroughtMonth'] ,as_index=False).agg({






df = grouped[["PRECTOT",'PS', 'TS', 'T2M_RANGE','WS10M_RANGE', 'WS10M']]
Month_column = grouped[['DroughtMonth']]
```

In [25]:

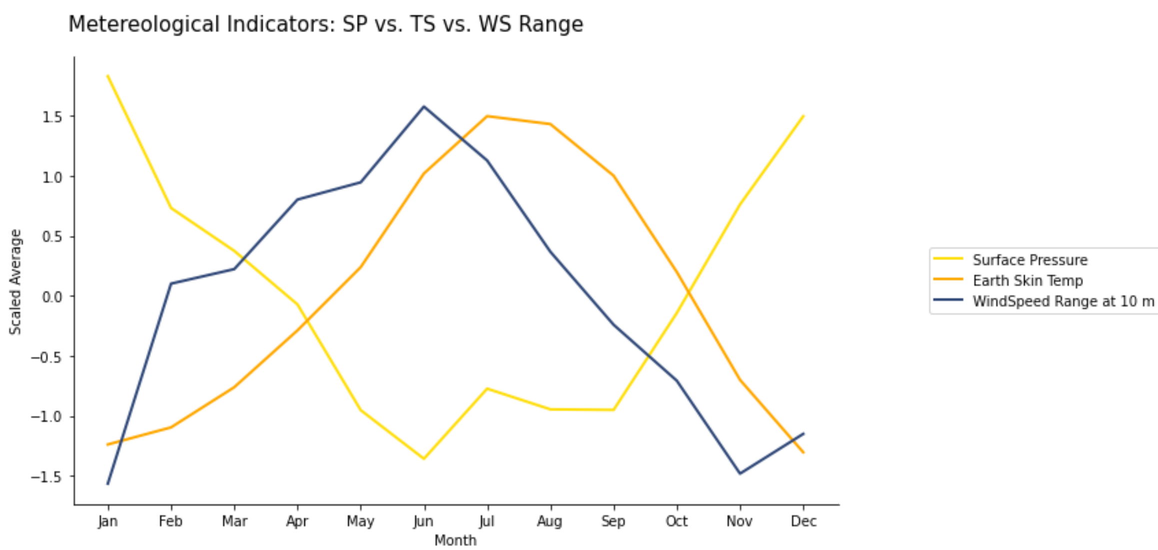
```
sc = StandardScaler()
df_std = sc.fit_transform(df)
df_std = pd.DataFrame(df_std, columns=["PRECTOT",'PS', 'TS', 'T2M_RANGE',
df_std['DroughtMonth'] = Month_column['DroughtMonth']
```

In [26]:

```
df_std = df_std.replace({'DroughtMonth' : {1: 'Jan', 2 : 'Feb', 3 : 'Mar
                                           5: 'May', 6: 'Jun', 7: 'Jul',
                                           9: 'Sep', 10: 'Oct', 11: 'Nov
fig, ax = plt.subplots(figsize = (10,6))
fig.suptitle('Metereeological Indicators: Precipitation vs. TemperatureRa
             x=0.12, y=.95, horizontalalignment='left', verticalalignmen
plot3 = sns.lineplot(data=df_std, x="DroughtMonth", y='T2M_RANGE', color
                     ax=ax, label="Temp Range at 2 m")
plot4 = sns.lineplot(data=df_std, x="DroughtMonth", y='WS10M', color="#0
                     ax=ax, label="WindSpeed at 10 m")
plot6 = sns.lineplot(data=df_std, x="DroughtMonth", y="PRECTOT", color="
                     ax=ax, label="Precipitation")

ax.set_xticks([0,1,2,3,4,5,6,7, 8,9, 10,11])
ax.set_xlabel("Month")
ax.set_ylabel("Scaled Average")

plt.legend(loc="right", bbox_to_anchor=(1.35, 0.5))
sns.despine()
plt.show()
```



In [27]:
```
df_std = df_std.replace({'DroughtMonth' : {1: 'Jan', 2 : 'Feb', 3 : 'Mar
                                          5: 'May', 6: 'Jun', 7: 'Jul',
                                          9: 'Sep', 10: 'Oct', 11: 'Nov
fig, ax = plt.subplots(figsize = (10,6))
fig.suptitle('Metereological Indicators: SP vs. TS vs. WS Range',
             x=0.12, y=.95, horizontalalignment='left', verticalalignmen
plot1 = sns.lineplot(data=df_std, x="DroughtMonth", y='PS', color="#FFDF(
                     ax=ax, label="Surface Pressure")
plot2 = sns.lineplot(data=df_std, x="DroughtMonth", y='TS', color="#FFA8(
                     ax=ax, label="Earth Skin Temp")
plot5 = sns.lineplot(data=df_std, x="DroughtMonth", y='WS10M_RANGE', colo
                     ax=ax, label="WindSpeed Range at 10 m")

ax.set_xticks([0,1,2,3,4,5,6,7, 8,9, 10,11])
ax.set_xlabel("Month")
ax.set_ylabel("Scaled Average")
plt.legend(loc="right", bbox_to_anchor=(1.43, 0.5))
sns.despine()
plt.show()
```



1d. Data Preliminary Analysis: Daily Summaries (2010 -2020)

In [28]:
```
ca_daily_df['DATE'] = pd.to_datetime(ca_daily_df['DATE'])
ca_daily_df['TempYear'] = ca_daily_df['DATE'].dt.year
ca_daily_df['TempMonth'] = ca_daily_df['DATE'].dt.month
ca_daily_df['TempDay'] = ca_daily_df['DATE'].dt.day
```

In [29]:
```
ca_daily_df = ca_daily_df.rename(columns={'STATION': 'StationCode', 'NAM
                                          'LATITUDE': 'StationLatitude',
                                          'ELEVATION': 'Elevation', 'DAT
                                          'TMAX': 'Max_Temp', 'TMIN': 'M
```

In [30]:
```
ca_daily_df['Avg_Temp'] = (ca_daily_df['Min_Temp'] + ca_daily_df['Max_Te
ca_daily_df.shape
```

Out[30]: (1314180, 13)

In [31]:
```
percentMissing(ca_daily_df)
```

```
StationCode - 0%
StationName - 0%
StationLatitude - 3%
StationLongitude - 3%
Elevation - 3%
TempDate - 0%
Precip - 0%
Max_Temp - 0%
Min_Temp - 0%
TempYear - 0%
TempMonth - 0%
TempDay - 0%
Avg_Temp - 0%
```

Deleting duplicates

In [32]:
```
ca_daily_df.shape
```

Out[32]:
```
(1314180, 13)
```

In [33]:
```
ca_daily_df = ca_daily_df.sort_values(["Max_Temp", "Min_Temp"], ascendin
```

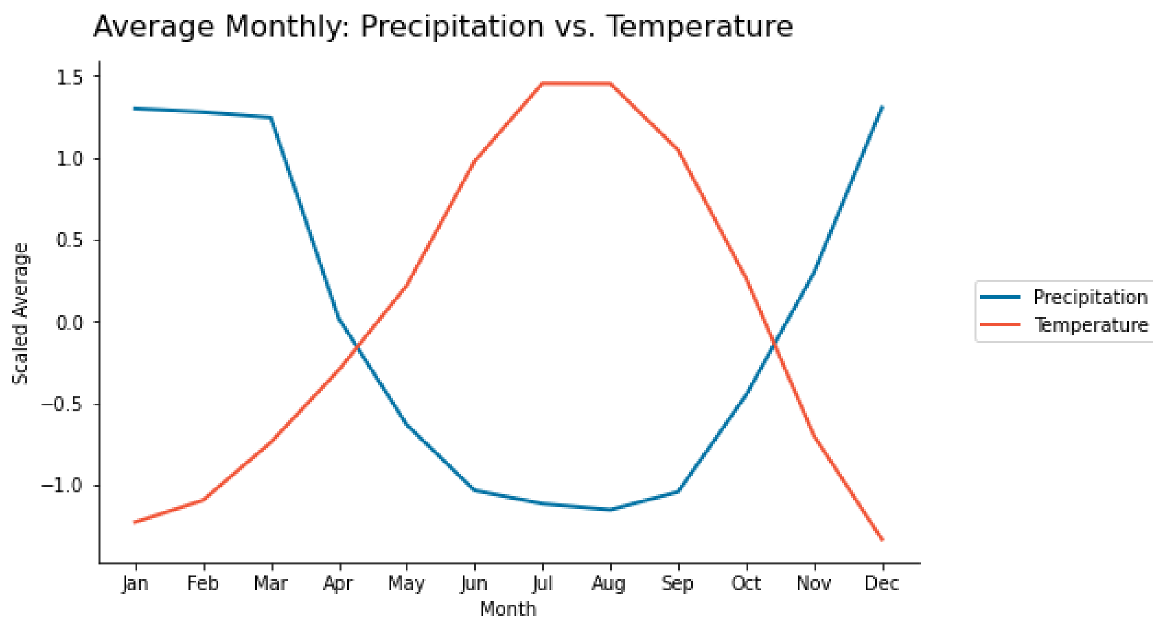
In [34]:
```
# check for duplicates in coordinates
ca_daily_df = ca_daily_df[~ca_daily_df.duplicated(['StationLatitude', 'S
ca_daily_df.shape
```

Out[34]:
```
(1269731, 13)
```

In [35]:
```
grouped1 = ca_daily_df.groupby(by=['TempMonth'] ,as_index=False).agg({'P
df1 = grouped1[["Precip", "Avg_Temp"]]
Month_column1 = grouped1[['TempMonth']]
```

In [36]:
```
sc = StandardScaler()
df_std1 = sc.fit_transform(df1)
df_std1 = pd.DataFrame(df_std1, columns=["Precip", "Avg_Temp"])
df_std1['TempMonth'] = Month_column1['TempMonth']
```

In [37]:
```
df_std1 = df_std1.replace({'TempMonth' : {1: 'Jan', 2 : 'Feb', 3 : 'Mar',
                                          5: 'May', 6: 'Jun', 7: 'Jul', 8:
                                          9: 'Sep', 10: 'Oct', 11: 'Nov',
fig, ax = plt.subplots(figsize = (8,5))
fig.suptitle('Average Monthly: Precipitation vs. Temperature',
             x=0.12, y=.95, horizontalalignment='left', verticalalignmen
plot1 = sns.lineplot(data=df_std1, x="TempMonth", y='Precip', color="#00
                     ax=ax, label="Precipitation")
plot2 = sns.lineplot(data=df_std1, x="TempMonth", y='Avg_Temp', color="#
                     ax=ax, label="Temperature")

ax.set_xticks([0,1,2,3,4,5,6,7, 8,9, 10,11])
ax.set_xlabel("Month")
ax.set_ylabel("Scaled Average")
plt.legend(loc="right", bbox_to_anchor=(1.3, 0.5))
sns.despine()
plt.show()
```



In [38]:
```
geometry = [Point(xy) for xy in zip(ca_daily_df['StationLongitude'], ca_
geometry[:3]
geo_daily_df = gpd.GeoDataFrame(ca_daily_df, crs=crs, geometry=geometry)
geo_daily_df.head(2)
```

Out[38]:

	StationCode	StationName	StationLatitude	StationLongitude	Elevation	TempDate	Pr
99763	USC00042319	DEATH VALLEY NATIONAL PARK, CA US	36.46263	-116.86720	-59.1	2020-08-16	
96964	USW00093115	IMPERIAL BEACH REAM FIELD NAS, CA US	32.56797	-117.11715	7.2	2010-12-08	

In [39]:
```
geo_daily_df = geo_daily_df.to_crs({'init': "EPSG:3310"})
```

In [40]:
```
geo_daily_df['Precip'].describe()
```

Out[40]:
```
count    1.269731e+06
mean     6.134752e-02
std      2.677502e-01
min      0.000000e+00
25%      0.000000e+00
50%      0.000000e+00
75%      0.000000e+00
max      1.218000e+01
Name: Precip, dtype: float64
```

Plotting Average Monthly Fires with Metereological Indicators

In [41]:
```
grouped2 = geo_fires_df['FireMonth'].value_counts().to_frame()
grouped2 = grouped2.sort_index()
grouped2.reset_index(level=0, inplace=True)
grouped2['AvgFires'] = grouped2['FireMonth'].apply(lambda x: x/10)
grouped2['FireMonth'] = grouped2['index']
grouped2 = grouped2.iloc[: , 1:]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [41], in <module>
----> 1 grouped2 = geo_fires_df['FireMonth'].value_counts().to_frame()
      2 grouped2 = grouped2.sort_index()
      3 grouped2.reset_index(level=0, inplace=True)

NameError: name 'geo_fires_df' is not defined
```

In [ ]:
```
df2 = grouped2[["AvgFires"]]
Month_column2 = grouped2[['FireMonth']]
```

In [ ]:
```
sc = StandardScaler()
df_std2 = sc.fit_transform(df2)
df_std2 = pd.DataFrame(df_std2, columns=["AvgFires"])
df_std2['FireMonth'] = Month_column2['FireMonth']
```

In [ ]:

```
df_std1 = df_std1.replace({'TempMonth' : {1: 'Jan', 2 : 'Feb', 3 : 'Mar'
                                        5: 'May', 6: 'Jun', 7: 'Jul', 8:
                                        9: 'Sep', 10: 'Oct', 11: 'Nov',
df_std2 = df_std2.replace({'FireMonth' : {1: 'Jan', 2 : 'Feb', 3 : 'Mar'
                                        5: 'May', 6: 'Jun', 7: 'Jul', 8:
                                        9: 'Sep', 10: 'Oct', 11: 'Nov',
fig, ax = plt.subplots(figsize = (8, 5))
fig.suptitle('Average Monthly: Precipitation vs. Fires', fontsize=16)
plot1 = sns.lineplot(data=df_std1, x="TempMonth", y='Precip', color="#00
                     ax=ax, label="Precipitation")
plot4 = sns.lineplot(data=df_std2, x="FireMonth", y='AvgFires', color="#
                     ax=ax, label="Fires")
ax.set_xticks([0,1,2,3,4,5,6,7, 8,9, 10,11])
ax.set_xlabel("Month")
ax.set_ylabel("Scaled Average")
plt.legend(loc="upper right")
sns.despine()
plt.show()
```

In [ ]:

```
df_std1 = df_std1.replace({'TempMonth' : {1: 'Jan', 2 : 'Feb', 3 : 'Mar'
                                        5: 'May', 6: 'Jun', 7: 'Jul', 8:
                                        9: 'Sep', 10: 'Oct', 11: 'Nov',
df_std2 = df_std2.replace({'FireMonth' : {1: 'Jan', 2 : 'Feb', 3 : 'Mar'
                                        5: 'May', 6: 'Jun', 7: 'Jul', 8:
                                        9: 'Sep', 10: 'Oct', 11: 'Nov',
fig, ax = plt.subplots(figsize = (8,5))
fig.suptitle('Average Monthly: Temperatures vs. Fires', fontsize=16)
plot2 = sns.lineplot(data=df_std1, x="TempMonth", y='Avg_Temp', color="#
                     ax=ax, label="Temperature")
plot4 = sns.lineplot(data=df_std2, x="FireMonth", y='AvgFires', color="#
                     ax=ax, label="Fires")
ax.set_xticks([0,1,2,3,4,5,6,7, 8,9, 10,11])
ax.set_xlabel("Month")
ax.set_ylabel("Scaled Average")
plt.legend(loc="upper right")
sns.despine()
plt.show()
```

In [ ]:

```
df_std2 = df_std2.replace({'FireMonth' : {1: 'Jan', 2 : 'Feb', 3 : 'Mar'
                                        5: 'May', 6: 'Jun', 7: 'Jul', 8:
                                        9: 'Sep', 10: 'Oct', 11: 'Nov',
fig, ax = plt.subplots(figsize = (8, 5))
fig.suptitle('Average Monthly: WindSpeed vs. Fires', fontsize=16)
plot3 = sns.lineplot(data=df_std, x="DroughtMonth", y='WS10M', color="#3
                     ax=ax, label="WindSpeed")
plot4 = sns.lineplot(data=df_std2, x="FireMonth", y='AvgFires', color="#
                     ax=ax, label="Fires")
ax.set_xticks([0,1,2,3,4,5,6,7, 8,9, 10,11])
ax.set_xlabel("Month")
ax.set_ylabel("Scaled Average")
plt.legend(loc="upper right")
sns.despine()
plt.show()
```

In [ ]:

```
r_df = df_std1[['Precip', 'Avg_Temp']]
r_df['WS'] = df_std['WS10M']
r_df['Avg_Fires'] = df_std2["AvgFires"]
```

In [ ]:

```
corrmat = r_df.corr()
top = corrmat.index
plt.figure(figsize=(8,8))

                      #plot heat map
g=sns.heatmap(r_df[top].corr(),annot=True,cmap="Reds")
```

In [ ]:

```

```

Appendix A4 - Data Merging Code

In [1]:

```
import datetime as dt
from pathlib import Path
import math
import os
import sqlite3
import json
import geopandas as gpd
import pygeos
import pyproj
import shapely
import shapely.ops as ops
from shapely.geometry import Point, Polygon
from shapely.geometry.polygon import Polygon
from functools import partial

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split


from sklearn import svm
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confu

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, f_classif, mutual_info_class
from functools import partial


from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings('ignore')
```

Dataset 1

In [250]:

```
print(geo_daily_df.crs)
print(geo_nasa_df.crs)
print(geo_fires_df.crs)
print(geo_soil_df.crs)
```

```
+init=epsg:3310 +type=crs
+init=epsg:3310 +type=crs
+init=epsg:3310 +type=crs
+init=epsg:4326 +type=crs
```

First combine all weather and Soil data with nasa dataset

In [251]:

```
def get_nearestpoint(df1, df1day, df2, df2day, days, dist):
    """
        This Function merges dataframe for selected day by finding neares
        for each day and creates mini dfs for each day of month
    """

    dfs = []
    for day in days:
        df = df1[df1[df1day] == day]
        df3 = df2[df2[df2day] == day]
        m_df = gpd.sjoin_nearest(df, df3, how='left', distance_col=dist)
        m_df[dist] = m_df[dist].apply(lambda x: x/1000)
        d = pd.DataFrame(m_df)
        dfs.append(d)

    dfs = pd.concat(dfs)
    return dfs
```

In [252]:

```
def merge_data(data1, df1year, df1month, df1day, data2, df2year, df2montl
    """
        This Function filters dataframe by year and months and calls for
        append it and then converts it into pandas df.
    """
    dfs = []
    for month in months:
        df1 = data1[(data1[df1year] == year) & (data1[df1month] == month
        df2 = data2[(data2[df2year] == year) & (data2[df2month] == month
        df = get_nearestpoint(df1, df1day, df2, df2day, days, dist)
        dfs.append(df)

    dfs = pd.concat(dfs)
    return dfs
```

In [253]:
```
def get_data(df1, df1year, df1month,df1day, df2, df2year, df2month,df2day
    """
        This calls for all dataframes and combine it and create one datas
        nasa data and daily temperatures
    """
    years = list(range(2010, 2021))
    months = list(range(1, 13))
    days = list(range(1, 32))

    dataframesList = []
    for year in years:
        data = merge_data(df1, df1year, df1month,df1day, df2, df2year, d

        dataframesList.append(data)

    df = gpd.GeoDataFrame(pd.concat(dataframesList), crs=crs)
    try:
        df.drop('index_right', axis=1, inplace=True)
    except ValueError:
        # ignore if there are no index columns
        pass

    print(df.shape)

    return df

```

Combine California fire data with nasa DF

In [254]:
```
pixel_temp = get_data(geo_nasa_df, 'ActiveYear', 'ActiveMonth', 'ActiveD
                      geo_daily_df, 'TempYear', 'TempMonth', 'TempDay',
```

(114599, 31)

In [255]:
```
pixel_soil = get_data(pixel_temp, 'ActiveYear', 'ActiveMonth', 'ActiveDay
                      geo_soil_df, 'DroughtYear', 'DroughtMonth', 'Drougl
```

(119665, 66)

Dropping Duplicates

In [256]:
```
pixel_soil = pixel_soil[~(pixel_soil.index.duplicated(keep='first'))]
```

In [257]:
```
pixel_fire= get_data(pixel_soil, 'ActiveYear', 'ActiveMonth', 'ActiveDay
                     geo_fires_df, 'FireYear','FireMonth', 'FireDay', 'f
```

(114599, 76)

In [258]:
```
merged_df =pixel_fire
```

In [259]:
```
# create a set for important variables and target variables.
# Delete repetitive dates, Station name, station codes are not needed.
# Name of the fire is also irrelevant and acres are also not
# needed because they are attributes that are logged after the fire event
merged_df = merged_df.drop(['Satellite', 'StationCode', 'StationName', 'S
                            'TempDate', 'TempYear', 'TempMonth', 'TempDay
                            'DroughtYear', 'DroughtMonth', 'DroughtDay',
                            'FireYear', 'Name', 'UniqueId'], axis = 1)
```

Perform Spatial Analysis

In [260]:
```
labeled_data = merged_df[merged_df['fire_dist'].notnull()]
unlabeled_data = merged_df[merged_df['fire_dist'].isnull()] # taking pixe
```

In [261]:
```
print(labeled_data.shape)
print(unlabeled_data.shape)
```

```
(82493, 57)
(32106, 57)
```

In [262]:
```
labeled_data[['fire_dist', 'Station_dist', 'Drought_dist']].describe()
```

Out[262]:

	fire_dist	Station_dist	Drought_dist
count	82493.000000	82493.000000	82493.000000
mean	299.182305	50.681572	310.481150
std	230.135528	85.893810	158.940170
min	0.000000	0.051470	0.260367
25%	122.804595	14.653380	202.725861
50%	237.723699	23.041096	299.132089
75%	430.907748	36.686614	442.452498
max	1259.457214	518.527597	805.499278

The maximum number of TotalAcres burned is 410202 Acres and roughly 1660 km^2, meaning the maximum distance a potential fire goes in any one direction is approximately around 830 km and min 1 km. This could've been a threshold to filter our data out for fire_dist to fire pixel if we were only trying to detect the fire pixel is true fire or not in any given day of the fire (considering fire can go on for weeks). However, the main purpose of this project is to build a model that can be used for early detection of the wildfire, so the hazard can be prevented from spreading. Not all fire pixels are true fire pixels and some times it is a false alarm. So we will filter for pixels that are within than 1 km from the true fire event, 1 km is used as a threshold because MODIS location coordinates are center of 1km fire pixel but not necessarily the actual location of the fire as one or more fires can be detected within the 1km pixel. Randomly sampled data from unlabeled data will be used as false alarms because no fire event was mapped to those dates.

In [263]:
```
class1 = labeled_data[labeled_data['fire_dist'] <=1]
```

In [264]:
```
class1[['fire_dist', 'Station_dist']].describe()
```

Out[264]:

	fire_dist	Station_dist
count	2087.000000	2087.000000
mean	0.218470	20.609494
std	0.285376	12.244417
min	0.000000	0.177639
25%	0.000000	11.653871
50%	0.030367	19.253845
75%	0.410110	27.747893
max	0.999613	71.508454

Station distance is sort of irrelevant as it is maximum 71 km away from the fire pixel, which is usually within county limit, and drastic weather changes are highly unlikely for such close approximation.

In [265]:
```
# class2 = unlabeled_data.sample(frac=.08)
unlabeled_fires = labeled_data[labeled_data['fire_dist'] >1]
```

In [957]:
```
#class2 = class1.append(class2)
```

In [266]:
```
print(class1.shape)
print(unlabeled_fires.shape)
print(unlabeled_data.shape)
```

```
(2087, 57)
(80406, 57)
(32106, 57)
```

Check for duplicates after merging

In [267]:
```
class1[class1.index.duplicated(keep=False)]
```

Out[267]:

NasaLatitude	NasaLongitude	Brightness	Scan	Track	ActiveDate	Confidence	BrightT31

0 rows × 57 columns

In [268]:
```
unlabeled_fires[unlabeled_fires.index.duplicated(keep=False)]
```

Out[268]:

NasaLatitude	NasaLongitude	Brightness	Scan	Track	ActiveDate	Confidence	BrightT31

0 rows × 57 columns

In [269]:
```
unlabeled_data[unlabeled_data.index.duplicated(keep=False)]
```

Out[269]:

NasaLatitude	NasaLongitude	Brightness	Scan	Track	ActiveDate	Confidence	BrightT31

0 rows × 57 columns

In [270]:
```
class1["Target"] = 1
unlabeled_fires["Target"] = 0
unlabeled_data["Target"] = 0
class2 = unlabeled_fires.append(unlabeled_data)
```

In [273]:
```
fire_data = class1.append(class2)
```

In [274]:
```
fire_data.shape
```

Out[274]: (114599, 58)

In [275]:
```
fire_data = fire_data.replace({'DayNight': {'D':1, 'N':0}, 'ConfidenceBi
```

In [276]:
```
features = fire_data.drop(['ActiveYear', 'ActiveMonth', 'ActiveDay','Sta
                           'lat','lon','Drought_dist', 'FireDay', 'fire_
                           'ActiveDate', 'Confidence', 'geometry', 'Fire
features.shape
```

Out[276]: (114599, 43)

In [277]:
```
fire_data.to_csv('Data/clean_dataset_preliminary.csv')
```

Final Analysis: Secondary and Optional Approach for Modeling

For this analysis, we will be going back to data preparation part and change the data little bit. For the first approach we took out all the noise data and trained the model with fire pixels that were mapped to the true fire events within the 1 km of the fire event on the day of fire alarm and with the unlabeled fire pixels that were not mapped with the fire events. However, this approach has some biases:

1. We do not know how much of the unlabeled data (fire pixels that was not mapped to the exact date of fire event) are true fire pixels after the start of fire.

2. All the fire pixels that were about more than 1 km away could also be true fire events on a different day for different fire event, because fire can go on for weeks. For example, for class C fire which is less than 100 acres (0.400639 km^2) can go minimum 0.400639 km and maximum 1 km distance for fire, any fire pixels in that range of distance and duration range between the day of actual fire and the fire containment date can be considered true fire event, anything outside of that distance threshold can be a false alarm, however anything outside of duration but within the 1 km distance means it is an ongoing fire. Note: The containment date signifies that control line has been completed around the fire, and any associated spot fires, which can reasonably be expected to stop the fire's spread, but the fire can continue going for months.

There are few assumptions I will be making when creating a transformed dataset only for this project purpose.

Dataset 2

In [278]:

```
class1.head() ## True Fire pixels on the day of fire and with distance w
```

Out[278]:

	NasaLatitude	NasaLongitude	Brightness	Scan	Track	ActiveDate	Confidence	Bright
39687	36.8878	-118.2121	315.9	1.3	1.1	2011-03-02	74	28
39705	36.8816	-118.2201	305.3	1.5	1.2	2011-03-02	50	29
77188	37.3252	-118.5696	330.4	3.8	1.8	2011-05-25	0	28
77769	35.6501	-118.3755	341.0	2.8	1.6	2011-05-27	88	29
77770	35.6372	-118.3795	320.3	2.8	1.6	2011-05-27	40	30

5 rows × 58 columns

In [279]:

```
def get_firepoints(data1, df1year, df1month, df1day, data2, df2year, df2r
    """
        This Function filters dataframe by year and months and calls for
        append it and then converts it into pandas df.
    """
    months = list(range(1, 13))
    dfs = []
    for month in months:
        df1 = data1[(data1[df1year] == year) & (data1[df1month] == month
        df2 = data2[(data2[df2year] == year) & (data2[df2month] == month

        df = gpd.sjoin_nearest(df1, df2, how='left', distance_col=dist)
        df[dist] = df[dist].apply(lambda x: x/1000)
        d = pd.DataFrame(df)
        dfs.append(d)

    dfs = pd.concat(dfs)
    return dfs
```

In [280]:

```
def get_fire_data(df1, df1year, df1month, df1day, df2, df2year, df2month,
    """
        This calls_ for all dataframes and combine it and create one data
        fire data, so we can use the combined information to find the est
    """
    years = list(range(2011, 2021))
    dataframesList = []
    for year in years:
        data = get_firepoints(df1, df1year, df1month, df1day, df2, df2yea
        dataframesList.append(data)

    df = gpd.GeoDataFrame(pd.concat(dataframesList), crs=crs)
    try:
        df.drop('index_right', axis=1, inplace=True)
    except ValueError:
        # ignore if there are no index columns
        pass

    print(df.shape)

    return df
```

In [281]:

```
pixel_soil.head(2)
```

Out[281]:

	NasaLatitude	NasaLongitude	Brightness	Scan	Track	ActiveDate	Satellite	Confidenc
26628	34.6033	-118.3848	305.8	1.2	1.1	2011-01-06	Terra	5
26638	39.8467	-121.5210	308.2	1.1	1.0	2011-01-07	Terra	7

2 rows × 66 columns

In [282]:

```
geo_fires_df = geo_fires_df[geo_fires_df['TotalAcres'].notnull()]
```

In [283]:
```
geo_fires_df['TotalAcres_sq_km'] = geo_fires_df['TotalAcres'].apply(lamb
```

In [284]:
```
print(geo_fires_df.shape)
geo_fires_df.head(2)
```

```
(3795, 11)
```

Out[284]:

	FireCause	TotalAcres	geometry	FireDate	FireYear	FireMonth	FireDay	Name	U
0	Powerline	109.60250	MULTIPOLYGON (((-116842.172 97942.739, -116837...	2020-06-18	2020.0	6.0	18.0	nelson	
1	Equipment Use	685.58502	MULTIPOLYGON (((-117329.343 90212.620, -117322...	2020-06-01	2020.0	6.0	1.0	amoruso	

In [285]:
```
pixel_fire2= get_fire_data(pixel_soil, 'ActiveYear', 'ActiveMonth', 'Act
                           geo_fires_df, 'FireYear','FireMonth', 'FireDay
```

```
(114599, 77)
```

In [286]:
```
merged_df2 = pixel_fire2
```

In [287]:
```
# create a set for important variables and target variables.
# Delete repetitive dates, Station name, station codes are not needed.
# Name of the fire is also irrelevant and acres are also not
# needed because they are attributes that are logged after the fire event
merged_df2 = merged_df2.drop(['Satellite', 'StationCode', 'StationName',
                              'TempDate', 'TempYear', 'TempMonth', 'TempDay
                              'DroughtYear', 'DroughtMonth', 'DroughtDay',
```

In [288]:
```
mapped_fire = merged_df2[merged_df2['fire_dist'].notnull()]
unmapped_fire = merged_df2[merged_df2['fire_dist'].isnull()] # taking pi
```

In [289]:
```
print(mapped_fire.shape)
print(unmapped_fire.shape)
```

```
(113912, 61)
(687, 61)
```

In [290]:
```
mapped_fire['fire_dist'].describe()
```

Out[290]:
```
count    113912.000000
mean         79.608633
std         127.814990
min           0.000000
25%           0.000000
50%          29.523428
75%          87.483079
max        1151.519722
Name: fire_dist, dtype: float64
```

In [291]:
```
def get_duration(df):
    df['Active_minus_FireDate'] = (df["ActiveDate"] - df["FireDate"]).dt
    df['Area_diff'] = (df["TotalAcres_sq_km"] - df["fire_dist"])
    df = df.sort_values('Active_minus_FireDate', ascending=True)

    return df
```

In [292]:
```
mapped_fire = get_duration(mapped_fire)
```

In [293]:
```
# fire distance bigger than total Acres means fire is outside of range a
# We used 32 km (10 miles) as a threshold for the area. Any fire outside
all_false = mapped_fire[(mapped_fire['TotalAcres_sq_km'] <=100) &
                        (mapped_fire['fire_dist'] >100) & (mapped_fire['
```

In [294]:
```
all_fire = mapped_fire[~((mapped_fire['TotalAcres_sq_km'] <=100) &
                         (mapped_fire['fire_dist'] >100) &
                         (mapped_fire['Area_diff'] <0))].sort_values('fi
```

In [295]:
```
f = all_fire[(all_fire['TotalAcres_sq_km'] <200) &
             (all_fire['fire_dist'] >200) &
             (all_fire['Area_diff'] <0)]

all_false = all_false.append(f)
```

In [296]:
```
all_fire = all_fire[~((all_fire['TotalAcres_sq_km'] <200) &
                      (all_fire['fire_dist'] >200) &
                      (all_fire['Area_diff'] <0))]
```

In [297]:
```
f = all_fire[(all_fire['TotalAcres_sq_km'] <400) &
             (all_fire['fire_dist'] >400) &
             (all_fire['Area_diff'] <0)]

all_false = all_false.append(f)
```

In [298]:
```
all_fire = all_fire[~((all_fire['TotalAcres_sq_km'] <400) &
                      (all_fire['fire_dist'] >400) &
                      (all_fire['Area_diff'] <0))]
```

In [299]:
```
f = all_fire[(all_fire['TotalAcres_sq_km'] <5) &
             (all_fire['fire_dist'] >5.99) &
             (all_fire['Area_diff'] <0)]

all_false = all_false.append(f)
```

In [300]:
```
all_fire = all_fire[~((all_fire['TotalAcres_sq_km'] <5) &
                      (all_fire['fire_dist'] >5.99) &
                      (all_fire['Area_diff'] <0))]
```

In [301]:
```
f = all_fire[(all_fire['Area_diff'] <0)]
all_false = all_false.append(f)
all_false = all_false.append(unmapped_fire)
```

In [302]:
```
all_fire = all_fire[~(all_fire['Area_diff'] <0)]
```

Removing duplicates from all the merging

In [303]:
```
False_pixels = all_false[~all_false.index.duplicated(keep='first')]
False_pixels = get_duration(False_pixels)
False_pixels.shape
```

Out[303]: (67863, 63)

In [304]:
```
True_pixels = all_fire[~all_fire.index.duplicated(keep='first')]
True_pixels = get_duration(True_pixels)
True_pixels.shape
```

Out[304]: (46736, 63)

Filtering for California Pixels only

In [305]:
```
x = False_pixels[~((False_pixels['NasaLongitude'] >-119.8) & (False_pixe
y = x[~((x['NasaLongitude'] >-119) & (x['NasaLatitude'] >38))]
False_pixels = y[~((y['NasaLongitude'] >-118) & (y['NasaLatitude'] >35.9
False_pixels.shape
```

Out[305]: (57935, 63)

In [306]:
```
geometry1 = [Point(xy) for xy in zip(False_pixels['NasaLongitude'], False
geometry1[:3]
plot_df1 = gpd.GeoDataFrame(False_pixels, crs=crs, geometry=geometry1)
```

In [307]:
```
True_pixels = True_pixels[~((True_pixels['NasaLongitude'] >-119.8) & (Tru
True_pixels.shape
```

Out[307]: (45186, 63)

In [308]:

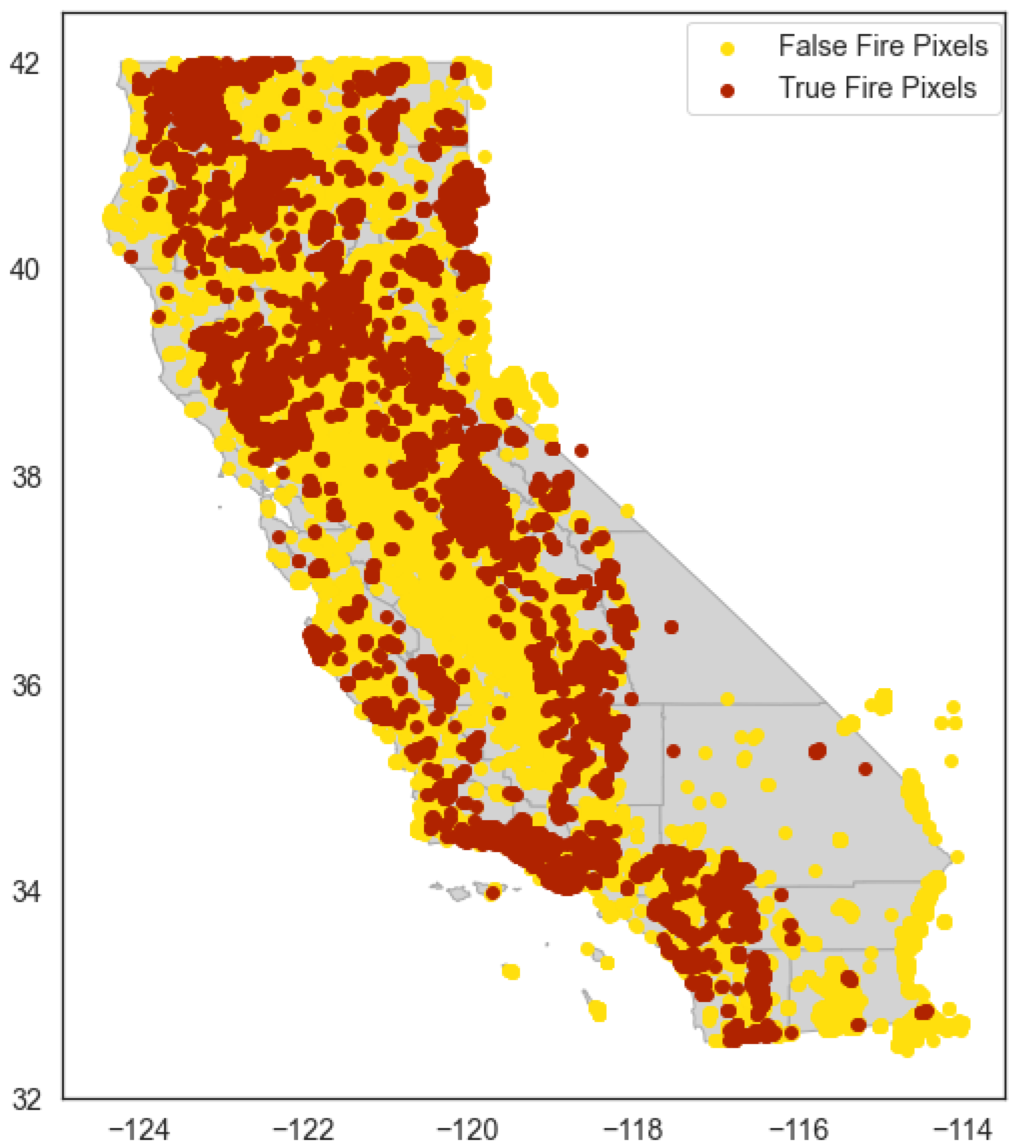
```
geometry2 = [Point(xy) for xy in zip(True_pixels['NasaLongitude'], True_
geometry2[:3]
plot_df2 = gpd.GeoDataFrame(True_pixels, crs=crs, geometry=geometry2)
```

Plotting all Labeled and Unlabeled Fires

In [309]:

```
fig, ax = plt.subplots(figsize = (10,10))
fig.suptitle('Mapped All Fire Pixels: labeled and unlabeled (2011-2020)',
plt.yticks([32, 34, 36, 38, 40, 42])
plt.xticks([-124, -122, -120, -118, -116, -114])

plt0 = USA[USA.STATEFP == '06'].plot(ax = ax, edgecolor="darkgrey", face
plt1 = plot_df1.plot(ax=ax, color="#FFDF0D", label="False Fire Pixels")
plt2 = plot_df2.plot(ax=ax, color="#B02300", label="True Fire Pixels")

handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc=(0.65,0.8))
plt.show()
```

Mapped All Fire Pixels: labeled and unlabeled (2011-2020)



In [310]:

```
plot_gf1 = geo_fires_df.to_crs({'init': "EPSG:4326"})
```

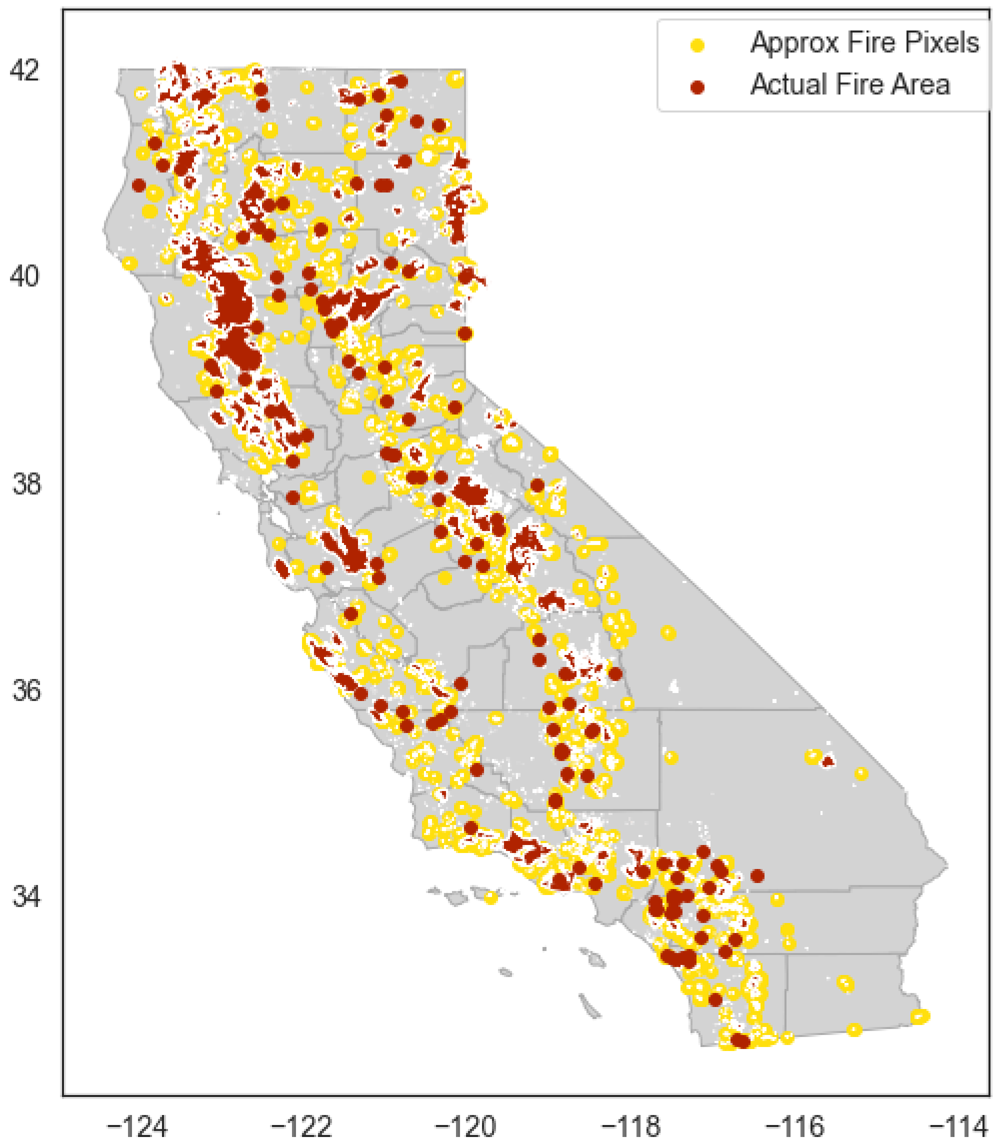
In [311]:

```
plot_df2 = plot_df2[(plot_df2['Active_minus_FireDate'] >=-1) & (plot_df2
```

In [312]:

```
fig, ax = plt.subplots(figsize = (10,10))
fig.suptitle('Mapped Labeled Actual Fires with Fire Pixel (2011-2020)',
plt.yticks([32, 34, 36, 38, 40, 42])
plt.xticks([-124, -122, -120, -118, -116, -114])

plt0 = USA[USA.STATEFP == '06'].plot(ax = ax, edgecolor="darkgrey", face
plt1 = plot_df2.plot(ax=ax, color="#FFDF0D", label="Approx Fire Pixels")
plt2 = plot_gf1.plot(ax=ax, color="#B02300", label ="Actual Fire Area")
handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, loc=(0.63,0.8))

plt.show()
```

Mapped Labeled Actual Fires with Fire Pixel (2011-2020)



In [313]:
```
dta0 = False_pixels.drop(['fire_dist','TotalAcres_sq_km', 'Active_minus_I

dta1 = True_pixels.drop(['fire_dist','TotalAcres_sq_km', 'Active_minus_F:
```

In [314]:
```
print(dta0.shape)
print(dta1.shape)
```

```
(57935, 59)
(45186, 59)
```

In [315]:
```
dta1["Target"] = 1
dta0["Target"] = 0
```

In [316]:
```
new_data = dta1.append(dta0)
```

In [317]:
```
new_data = new_data.replace({'DayNight': {'D':1, 'N':0}})
new_data = new_data.reset_index(drop=True)
```

In [318]:
```
new_data = new_data.sort_values(['ActiveDate'])

new_data['geometry'] = list(zip(new_data['NasaLongitude'], new_data['Nasa
```

In [319]:
```
new_df = new_data.set_index(['ActiveDate', 'geometry'])
```

In [320]:
```
new_df.to_csv('Data/clean_dataset1.csv')
```

In [ ]:
```

```

Appendix A.5 Data Modeling Preliminary - Random Forest and SVM - Classification Model

In [1]:

```
import datetime as dt
from pathlib import Path
import math
import os
import json

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split


from sklearn import svm
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confu

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, f_classif, mutual_info_class
from functools import partial

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_
from scipy import stats
from sklearn.preprocessing import StandardScaler

# Import Keras
from keras.models import Sequential
from keras.layers import Dense, LSTM, BatchNormalization
from keras.optimizers import Adam
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
from keras.regularizers import l2
from time import time
import pickle

import warnings
warnings.filterwarnings('ignore')
```

```
Using TensorFlow backend.
```

In [2]:
```
fires_data = pd.read_csv("Data/clean_dataset_preliminary.csv")
fires_data.drop(columns=fires_data.columns[0], axis=1, inplace=True)
fires_data.head(2)
```

Out[2]:

	NasaLatitude	NasaLongitude	Brightness	Scan	Track	ActiveDate	Confidence	BrightT31
0	36.8878	-118.2121	315.9	1.3	1.1	2011-03-02	74	280.6
1	36.8816	-118.2201	305.3	1.5	1.2	2011-03-02	50	291.6

2 rows × 58 columns

In [4]:
```
features = fires_data.drop(['ActiveYear', 'ActiveMonth', 'ActiveDay','Sta
                            'lat','lon','Drought_dist', 'FireDay', 'fire_
                            'ActiveDate', 'ConfidenceBinned', 'geometry',
features.shape
```

Out[4]: (114599, 42)

In [5]:
```
target = fires_data['Target']
target.shape
```

Out[5]: (114599,)

In [6]:
```
# Checking data type
def Datatype(df):
    # shape and data types of the data
    print("There are {} rows and {} columns".format(df.shape[0], df.shape
    print(df.dtypes)

    # select numeric columns
    df_numeric = df.select_dtypes(include=[np.number])
    numeric_cols = df_numeric.columns.values
    print(numeric_cols)

    # select non numeric columns
    df_non_numeric = df.select_dtypes(exclude=[np.number])
    non_numeric_cols = df_non_numeric.columns.values
    print(non_numeric_cols)
```

In [7]:
```
cor_matrix = features.corr().abs()
upper_tri = cor_matrix.where(np.triu(np.ones(cor_matrix.shape),k=1).asty

to_drop = [column for column in upper_tri.columns if any(upper_tri[colum
print(to_drop)
```

```
['Track', 'Avg_Temp', 'elevation', 'slope2', 'slope3', 'slope6', 'slope8',
'aspectN', 'aspectE', 'aspectS', 'aspectW', 'NVG_LAND', 'FOR_LAND', 'CULTRF
_LAND', 'CULTIR_LAND', 'SQ2', 'SQ3']
```

In [8]:

```
features = features.drop(to_drop, axis = 1)
```

In [10]:

```
Datatype(features)
```

```
There are 114599 rows and 25 columns
NasaLatitude     float64
NasaLongitude    float64
Brightness       float64
Scan             float64
Confidence         int64
BrightT31        float64
Frp              float64
DayNight           int64
HotSpotType      float64
Elevation        float64
Precip           float64
Max_Temp         float64
Min_Temp         float64
PRECTOT          float64
PS               float64
T2M_RANGE        float64
TS               float64
WS10M            float64
WS10M_RANGE      float64
slope1           float64
slope4           float64
WAT_LAND         float64
URB_LAND         float64
GRS_LAND         float64
SQ4                int64
dtype: object
['NasaLatitude' 'NasaLongitude' 'Brightness' 'Scan' 'Confidence'
 'BrightT31' 'Frp' 'DayNight' 'HotSpotType' 'Elevation' 'Precip'
 'Max_Temp' 'Min_Temp' 'PRECTOT' 'PS' 'T2M_RANGE' 'TS' 'WS10M'
 'WS10M_RANGE' 'slope1' 'slope4' 'WAT_LAND' 'URB_LAND' 'GRS_LAND' 'SQ4']
[]
```

Preliminary Modeling

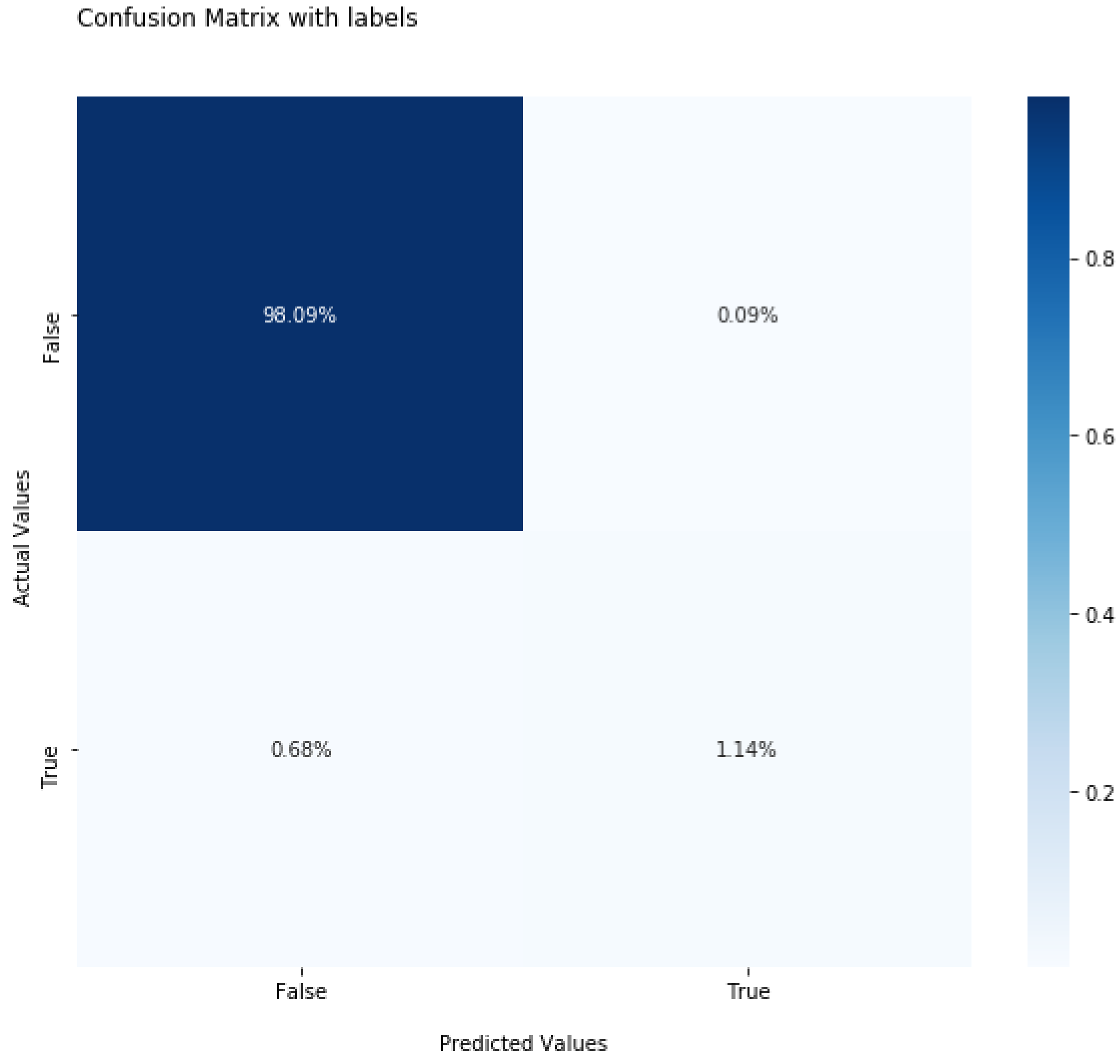
In [11]:
```
def results(classifier, x_train, y_train, x_test, y_test):

    model = classifier.fit(x_train, y_train)

    y_pred = classifier.predict(x_test)

    #Checking the accuracy
    accuracy = round(accuracy_score(y_test, y_pred)*100,2)
    print("Accuracy score: {}{}".format(round(accuracy, 2), '%'))

    cv_accuracy_score = cross_val_score(model, x_test, y_test, cv=5, scor
    print("Cross validation Accuracy score: {}{}".format(round(cv_accura

    cv_precision_score = cross_val_score(model, x_test, y_test, cv=5, sc
    print("Cross validation Precision score: {}{}".format(round(cv_preci

    cv_recall_score = cross_val_score(model, x_test, y_test, cv=5, scori
    print("Cross validation Recall score: {}{}".format(round(cv_recall_s

    cv_f1_score = cross_val_score(model, x_test, y_test, cv=5, scoring="
    print("Cross validation F1 score: {}{}".format(round(cv_f1_score*100

    cf_matrix = confusion_matrix(y_test, y_pred)

    fig, ax = plt.subplots(figsize = (10,8))
    ax = sns.heatmap(cf_matrix/np.sum(cf_matrix), annot=True, fmt='.2%',

    ax.set_title('Confusion Matrix with labels\n\n', loc='left')
    ax.set_xlabel('\nPredicted Values')
    ax.set_ylabel('Actual Values ')
    ax.xaxis.set_ticklabels(['False','True'])
    ax.yaxis.set_ticklabels(['False','True'])
    plt.show()

    return model

```

Model 1 Random Forest Classification

In [13]:
```
# split the data
x_train, x_test, y_train, y_test = train_test_split(features, target,
                                                    test_size =0.30, random_
```

In [14]:
```
rfm = RandomForestClassifier()
model1 = results(rfm, x_train, y_train, x_test, y_test)
```

```
Accuracy score: 99.23%
Cross validation Accuracy score: 98.81%
Cross validation Precision score: 92.69%
Cross validation Recall score: 37.87%
Cross validation F1 score: 53.74%
```



Model 2 Support Vector Machines: RBF Kernel

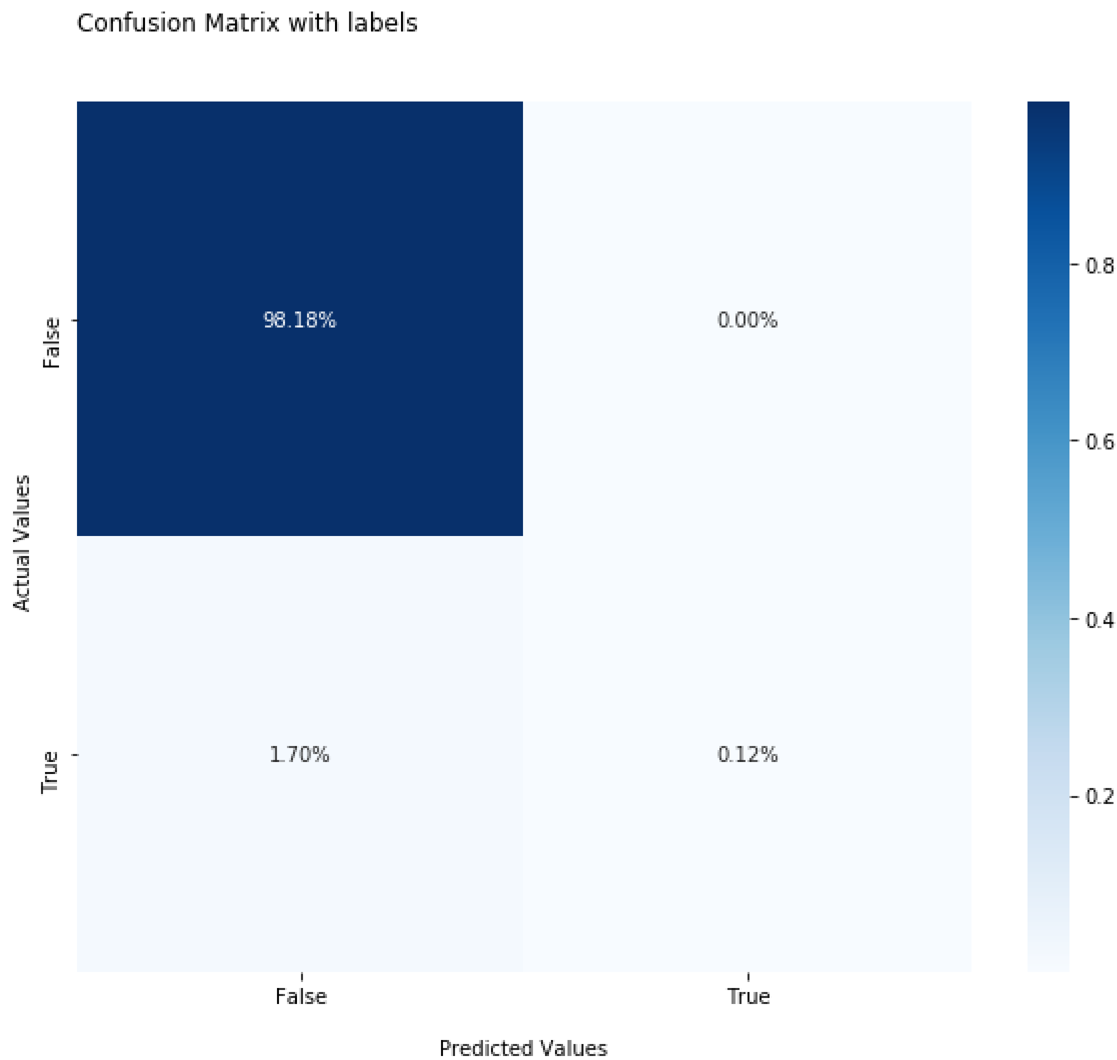
In [17]:
```
one_hot_encoded_data = pd.get_dummies(features, columns = ['DayNight', 'I
                                                           'WAT_LAND', 'U
```

In [18]:
```
# scaling the features
sc = StandardScaler()
x_train_std = sc.fit_transform(x_train)
x_test_std = sc.transform(x_test)
```

In [19]:

```
rbf = svm.SVC(kernel='rbf', C=1.0)
model2 = results(rbf, x_train_std, y_train, x_test_std, y_test)
```

```
Accuracy score: 98.3%
Cross validation Accuracy score: 98.26%
Cross validation Precision score: 86.29%
Cross validation Recall score: 5.43%
Cross validation F1 score: 10.18%
```



Appendix A.6 Data Modeling Preliminary LSTM RNN - Time Series Classification

In [3]:

```
import datetime as dt
from pathlib import Path
import math
import os
import json

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split


from sklearn import svm
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confu

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, f_classif, mutual_info_class
from functools import partial

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_
from scipy import stats
from sklearn.preprocessing import StandardScaler

# Import Keras
from keras.models import Sequential
from keras.layers import Dense, LSTM, BatchNormalization
from keras.optimizers import Adam
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
from keras.regularizers import l2
from time import time
import pickle

import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
# Checking data type
def Datatype(df):
    # shape and data types of the data
    print("There are {} rows and {} columns".format(df.shape[0], df.shap
    print(df.dtypes)

    # select numeric columns
    df_numeric = df.select_dtypes(include=[np.number])
    numeric_cols = df_numeric.columns.values
    print(numeric_cols)

    # select non numeric columns
    df_non_numeric = df.select_dtypes(exclude=[np.number])
    non_numeric_cols = df_non_numeric.columns.values
    print(non_numeric_cols)
```

Functions for Feature Engineering

In [18]:

```
def transformed_features(x, y, T):
    # scaling the features
    sc = StandardScaler()
    x_std = pd.DataFrame(sc.fit_transform(x.values),
                                  index=x.index,
                                  columns=x.columns)
    x_std, y_std= TimeSeries_data(x_std, y, T)
    return x_std, y_std
```

In [19]:

```
def TimeSeries_data (x, y, T):
    inputs, target = [], []
    for i in range(y.shape[0] - (T)):
        inputs.append(x.iloc[i:i+T].values)
        target.append(y.iloc[i + (T)])
    inputs, target = np.array(inputs), np.array(target).reshape(-1,1)
    return inputs, target
```

In [20]:

```
def lstm_transformed_data(features, target, Year_column, T):
    cor_matrix = features.corr().abs()
    upper_tri = cor_matrix.where(np.triu(np.ones(cor_matrix.shape),k=1).a
    to_drop = [column for column in upper_tri.columns if any(upper_tri[co
    print("Dropping these variables because of multicollinearity")
    print("\n")
    print(to_drop)

    features = features.drop(to_drop, axis=1)
    features['Year'] = Year_column['ActiveYear']
    features['Target'] = target['Target']

    # split time wise because of time series classlification
    train_set = features[features['Year'] <2017]
    validation_set = features[(features['Year'] >=2017) & (features['Year
    test_set = features[features['Year'] >2018]

    x_train = train_set.drop(['Target', 'Year'], axis = 1)
    y_train = train_set['Target']

    x_val = validation_set.drop(['Target', 'Year'], axis = 1)
    y_val = validation_set['Target']

    x_test = test_set.drop(['Target', 'Year'], axis = 1)
    y_test = test_set['Target']
    s1 = x_val.shape
    s2 = x_test.shape
    print("\nFeatures Before Prepending {} days of data:".format(T))
    print("Validation set: {}".format(s1))
    print("Test set: {}".format(s2))

    prepend_features1 = x_train.iloc[-(T):]
    prepend_features2 = x_val.iloc[-(T):]

    x_val = pd.concat([prepend_features1, x_val], axis=0)
    x_test = pd.concat([prepend_features2, x_test], axis=0)
    s3 = x_val.shape
    s4 = x_test.shape
    print("\nFeatures After Prepending {} days of data:".format(T))
    print("Validation set: {}".format(s3))
    print("Test set: {}".format(s4))

    x_train, y_train = transformed_features(x_train, y_train, T)
    x_val, y_val = transformed_features(x_val, y_val, T)
    x_test, y_test = transformed_features(x_test, y_test, T)
    s5 = x_train.shape
    s6 = x_val.shape
    s7 = x_test.shape

    print("\nFeatures After Transforming and scaling the data:")
    print("train set: {}".format(s5))
    print("Validation set: {}".format(s6))
    print("Test set: {}".format(s7))

    return x_train, y_train, x_val, y_val, x_test, y_test

```

Functions for Modeling

In [21]:

```
# Build the Model
def lstm_model(x_train, y_train, x_val, y_val, x_test, y_test, T,N, epoc
    model = Sequential()
    model.add(LSTM(input_shape=(T, N), units=4, activation='tanh',
                   recurrent_activation='hard_sigmoid', kernel_regulariz
                   recurrent_regularizer=l2(.01),
                   return_sequences=True, return_state=False))

    model.add(BatchNormalization())
    model.add(LSTM(units=4, activation='tanh', recurrent_activation='har
                   kernel_regularizer=l2(.01), recurrent_regularizer=l2(
                   return_sequences=True, return_state=False))

    model.add(BatchNormalization())
    model.add(LSTM(units=4, activation='tanh', recurrent_activation='har
                   kernel_regularizer=l2(0.1), recurrent_regularizer=l2(
                   return_sequences=False, return_state=False))

    model.add(BatchNormalization())
    model.add(Dense(units=1, activation='sigmoid'))
    # Compile the model with Adam optimizer
    model.compile(loss='binary_crossentropy',
                      metrics=['accuracy'],
                      optimizer=Adam(lr=.001))
    print(model.summary())
    lstm = results2(model, x_train, y_train, x_val, y_val, x_test, y_tes
    return lstm
```

In [22]:

```python
def results2(model, x_train, y_train, x_val, y_val, x_test, y_test, epoc

     # Define a learning rate decay method:
    lr_decay = ReduceLROnPlateau(monitor='loss',
                                 patience=1, verbose=0,
                                 factor=0.5, min_lr=1e-8)
    # Define Early Stopping:
    early_stop = EarlyStopping(monitor='val_acc', min_delta=0,
                                 patience=30, verbose=1, mode='auto',
                                 baseline=0, restore_best_weights=True)

    start = time()
    History = model.fit(x_train, y_train,
                        epochs=epoch,
                        batch_size=size,
                        validation_data=(x_val, y_val),
                        shuffle=True,verbose=0,
                        callbacks=[lr_decay, early_stop])
    print('-'*65)
    print(f'Training was completed in {time() - start:.2f} secs')
    print('-'*65)

    print('\n')
    print('Score for Model Testing')
    print(model.evaluate(x_test,y_test))


    history_dict = History.history
    acc = history_dict['accuracy']
    val_acc = history_dict['val_accuracy']
    loss_values = history_dict['loss']
    val_loss_values = history_dict['val_loss']
    epochs = range(1,len(acc) + 1)

    # Plotting metrics
    plt.plot(epochs, acc,  'bo', label = 'Training accuracy')
    plt.plot(epochs, val_acc, 'b', label = 'Validation accuracy')
    plt.title('Training and Validation Accuracy')
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.figure()
    plt.plot(epochs, loss_values,  'bo', label = 'Training Loss')
    plt.plot(epochs, val_loss_values, 'b', label = 'Validation loss')
    plt.title('Training and Validation Loss')
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.show()


    y_preds = model.predict_classes(x_test)
    accuracy = accuracy_score(y_test, y_preds)*100
    print("Accuracy score: {}{}".format(round(accuracy, 2), '%'))

    precision = precision_score(y_test, y_preds)*100
```

```
    print("Precision score: {}{}".format(round(precision, 2), "%"))

    recall = recall_score(y_test, y_preds)*100
    print("Recall score: {}{}".format(round(recall, 2), "%"))


    cf_matrix = confusion_matrix(y_test, y_preds)

    fig, ax = plt.subplots(figsize = (10,8))
    ax = sns.heatmap(cf_matrix/np.sum(cf_matrix), annot=True, fmt='.2%',

    ax.set_title('Confusion Matrix with labels\n\n', loc='left')
    ax.set_xlabel('\nPredicted Values')
    ax.set_ylabel('Actual Values ')
    ax.xaxis.set_ticklabels(['False','True'])
    ax.yaxis.set_ticklabels(['False','True'])
    plt.show()

    return model

```

In [23]:

```
def get_duration(df):
    df['Active_minus_FireDate'] = (df["ActiveDate"] - df["FireDate"]).dt
    df['Area_diff'] = (df["TotalAcres_sq_km"] - df["fire_dist"])
    df = df.sort_values('Active_minus_FireDate', ascending=True)

    return df
```

In [106]:

```
features2 = new_df.drop(['ActiveYear', 'ActiveDay','Station_dist', 'lat'
                          'elevation','ConfidenceBinned', 'Drought_dist'

print(features.shape)
Year_column2 = new_df[['ActiveYear']]
target2 = new_df[['Target']]
print(target.shape)
```

```
(103121, 41)
(103121, 1)
```

Model training for 3 day Sequential Data

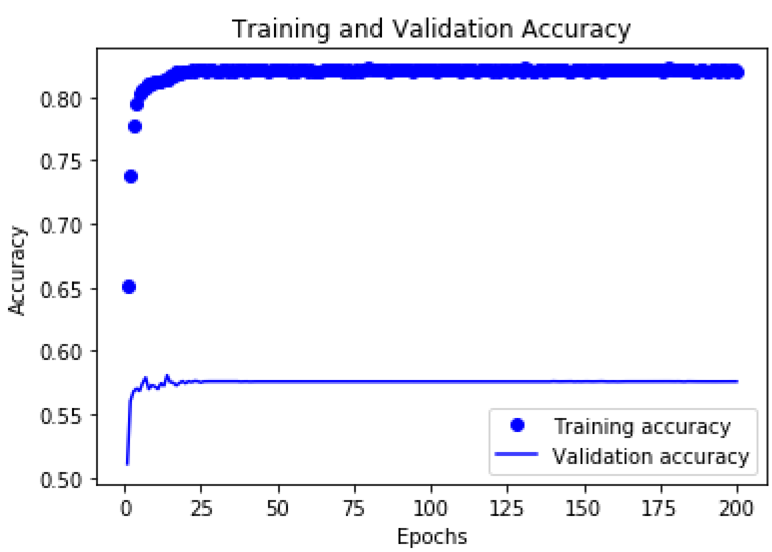
In [99]:

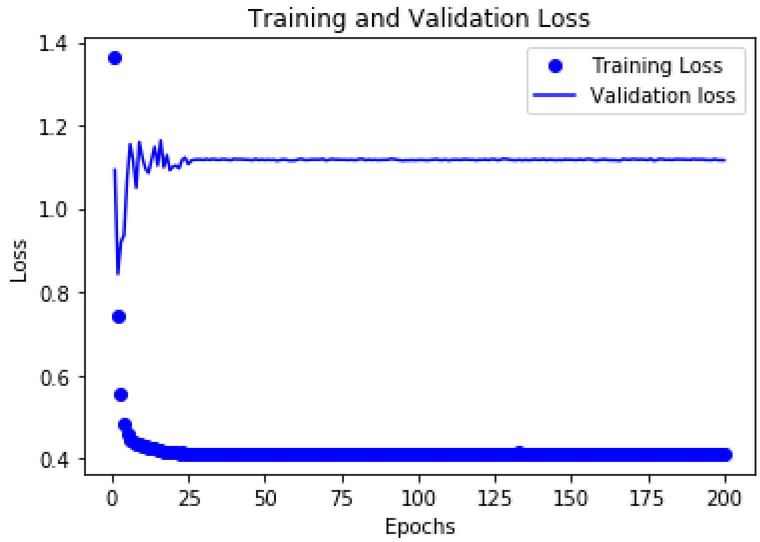
```
1 x_train, y_train, x_val, y_val, x_test, y_test = lstm_transformed_data(f
```

```
Dropping these variables because of multicollinearity


['NasaLongitude', 'Track', 'Avg_Temp', 'PS', 'TS', 'slope2', 'slope6', 'asp
ectE', 'aspectS', 'aspectW', 'CULTIR_LAND', 'SQ2', 'SQ3', 'SQ4']

Features Before Prepending 3 days of data:
Validation set: (40048, 27)
Test set: (8879, 27)

Features After Prepending 3 days of data:
Validation set: (40051, 27)
Test set: (8882, 27)

Features After Transforming and scaling the data:
train set: (54191, 3, 27)
Validation set: (40045, 3, 27)
Test set: (8876, 3, 27)
```

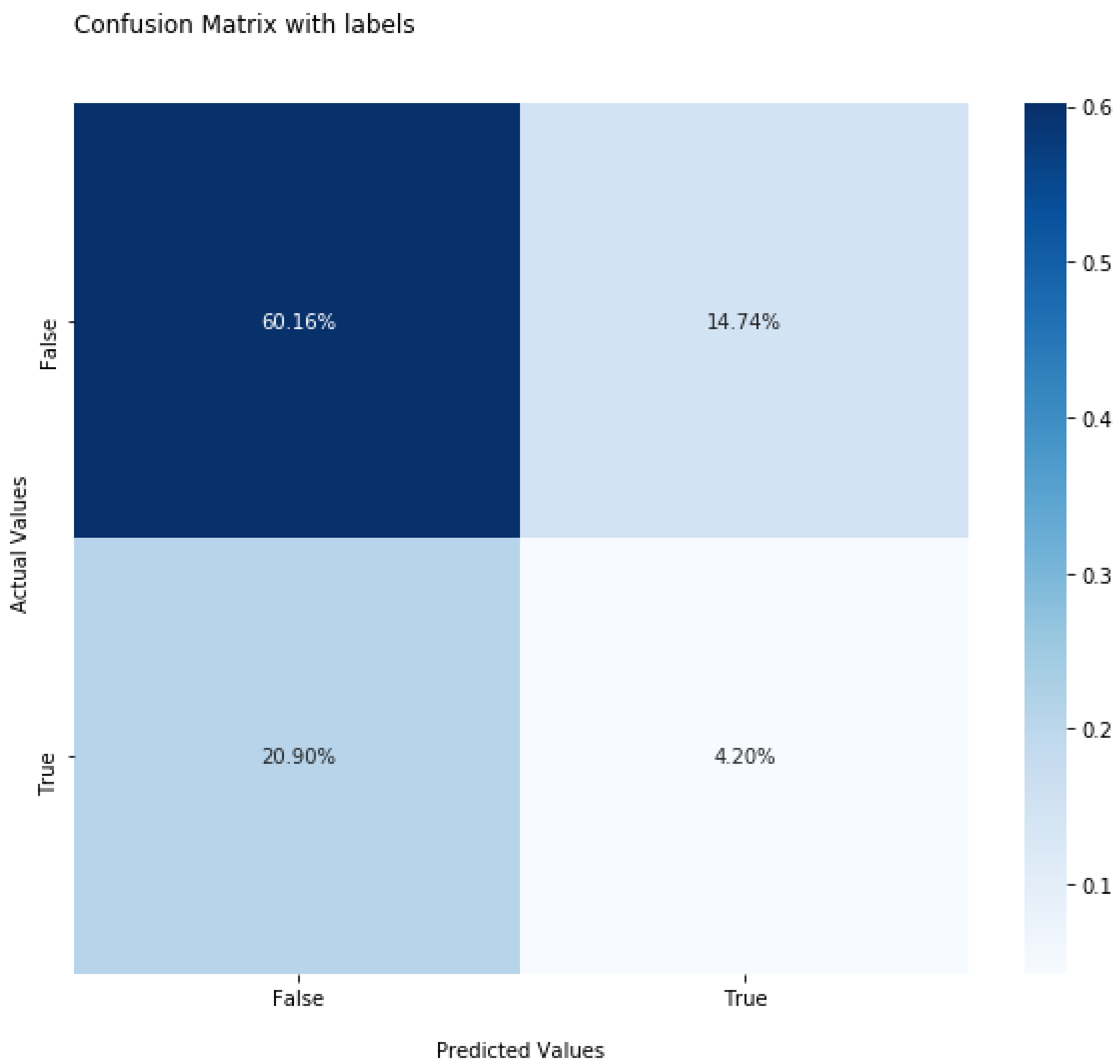
In [102]:
```
1 lstm1 = lstm_model(x_train, y_train, x_val, y_val, x_test, y_test,3,27,
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_7 (LSTM)                (None, 3, 4)              512
_________________________________________________________________
batch_normalization_7 (Batch (None, 3, 4)              16
_________________________________________________________________
lstm_8 (LSTM)                (None, 3, 4)              144
_________________________________________________________________
batch_normalization_8 (Batch (None, 3, 4)              16
_________________________________________________________________
lstm_9 (LSTM)                (None, 4)                 144
_________________________________________________________________
batch_normalization_9 (Batch (None, 4)                 16
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 5
=================================================================
Total params: 853
Trainable params: 829
Non-trainable params: 24
_________________________________________________________________
None
-----------------------------------------------------------------
Training was completed in 332.24 secs
-----------------------------------------------------------------


Score for Model Testing
8876/8876 [==============================] - 0s 43us/step
[0.845916429253353, 0.6436457633972168]
```






```
Accuracy score: 64.36%
Precision score: 22.19%
Recall score: 16.74%
```



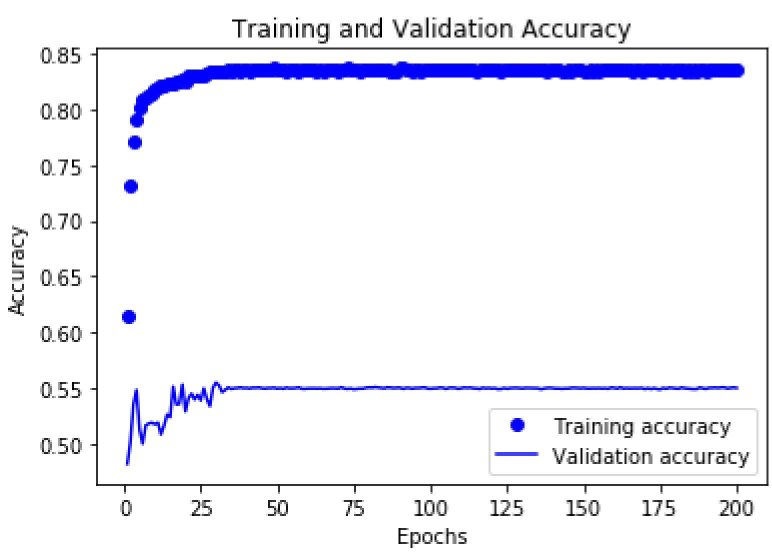
Model training for 5 day Sequential Data

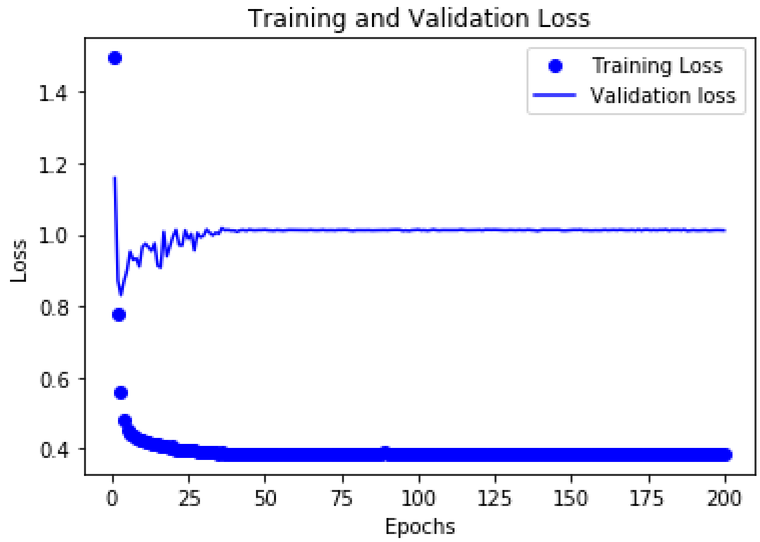
In [112]:
```
1 x_train, y_train, x_val, y_val, x_test, y_test = lstm_transformed_data(f
```

```
Dropping these variables because of multicollinearity


['NasaLongitude', 'Track', 'Avg_Temp', 'PS', 'TS', 'slope2', 'slope6', 'asp
ectE', 'aspectS', 'aspectW', 'CULTIR_LAND', 'SQ2', 'SQ3', 'SQ4']

Features Before Prepending 5 days of data:
Validation set: (40048, 27)
Test set: (8879, 27)

Features After Prepending 5 days of data:
Validation set: (40053, 27)
Test set: (8884, 27)

Features After Transforming and scaling the data:
train set: (54189, 5, 27)
Validation set: (40043, 5, 27)
Test set: (8874, 5, 27)
```

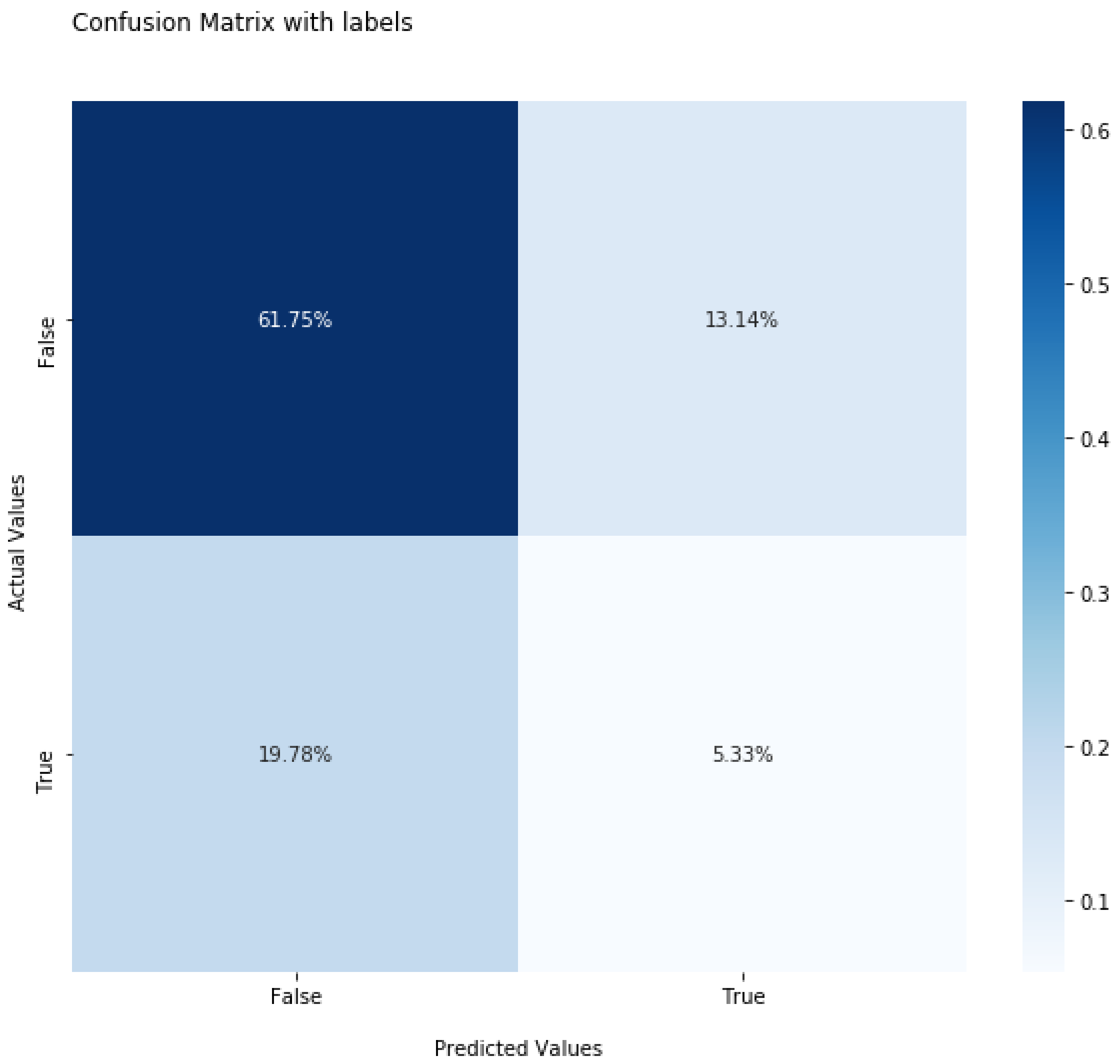
In [113]:
```
1 lstm2 = lstm_model(x_train, y_train, x_val, y_val, x_test, y_test,5,27, :
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_13 (LSTM)               (None, 5, 4)              512
_________________________________________________________________
batch_normalization_13 (Batc (None, 5, 4)              16
_________________________________________________________________
lstm_14 (LSTM)               (None, 5, 4)              144
_________________________________________________________________
batch_normalization_14 (Batc (None, 5, 4)              16
_________________________________________________________________
lstm_15 (LSTM)               (None, 4)                 144
_________________________________________________________________
batch_normalization_15 (Batc (None, 4)                 16
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 5
=================================================================
Total params: 853
Trainable params: 829
Non-trainable params: 24
_________________________________________________________________
None
-----------------------------------------------------------------
Training was completed in 497.81 secs
-----------------------------------------------------------------


Score for Model Testing
8874/8874 [==============================] - 0s 55us/step
[0.6974002486592293, 0.6708361506462097]
```






```
Accuracy score: 67.08%
Precision score: 28.86%
Recall score: 21.23%
```



Model training for 7 day Sequential Data

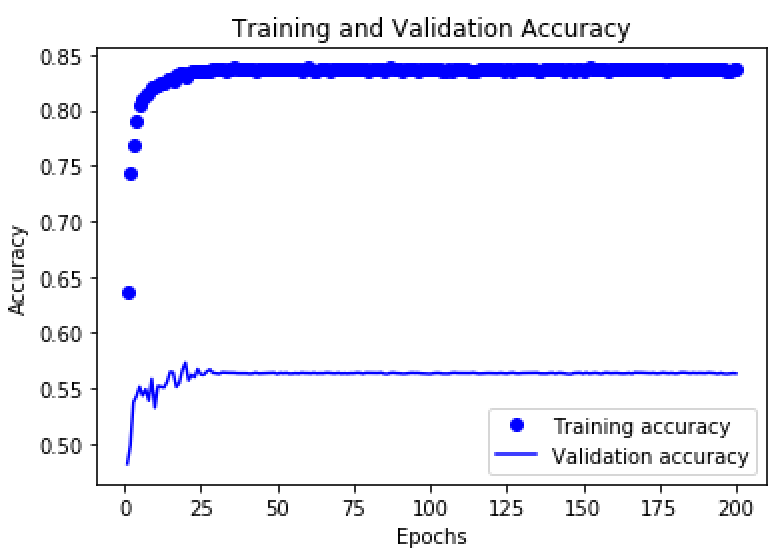
In [114]:
```
x_train, y_train, x_val, y_val, x_test, y_test = lstm_transformed_data(f
```

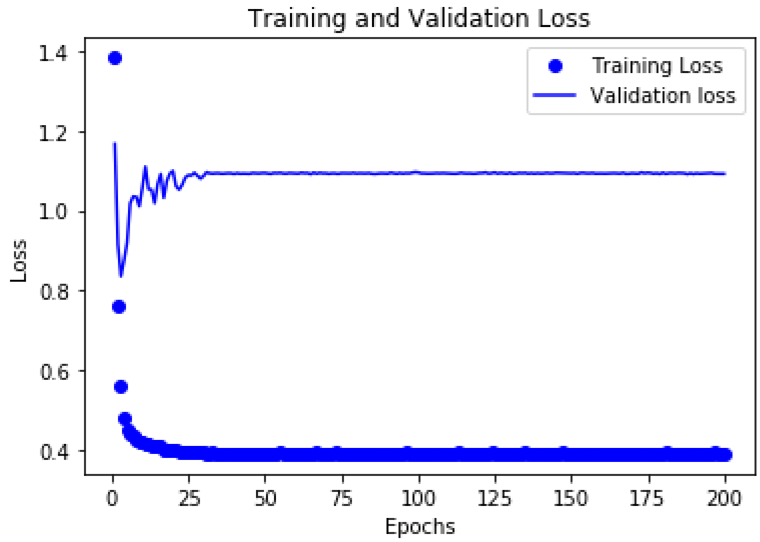
```
Dropping these variables because of multicollinearity


['NasaLongitude', 'Track', 'Avg_Temp', 'PS', 'TS', 'slope2', 'slope6', 'asp
ectE', 'aspectS', 'aspectW', 'CULTIR_LAND', 'SQ2', 'SQ3', 'SQ4']

Features Before Prepending 7 days of data:
Validation set: (40048, 27)
Test set: (8879, 27)

Features After Prepending 7 days of data:
Validation set: (40055, 27)
Test set: (8886, 27)

Features After Transforming and scaling the data:
train set: (54187, 7, 27)
Validation set: (40041, 7, 27)
Test set: (8872, 7, 27)
```

In [115]:

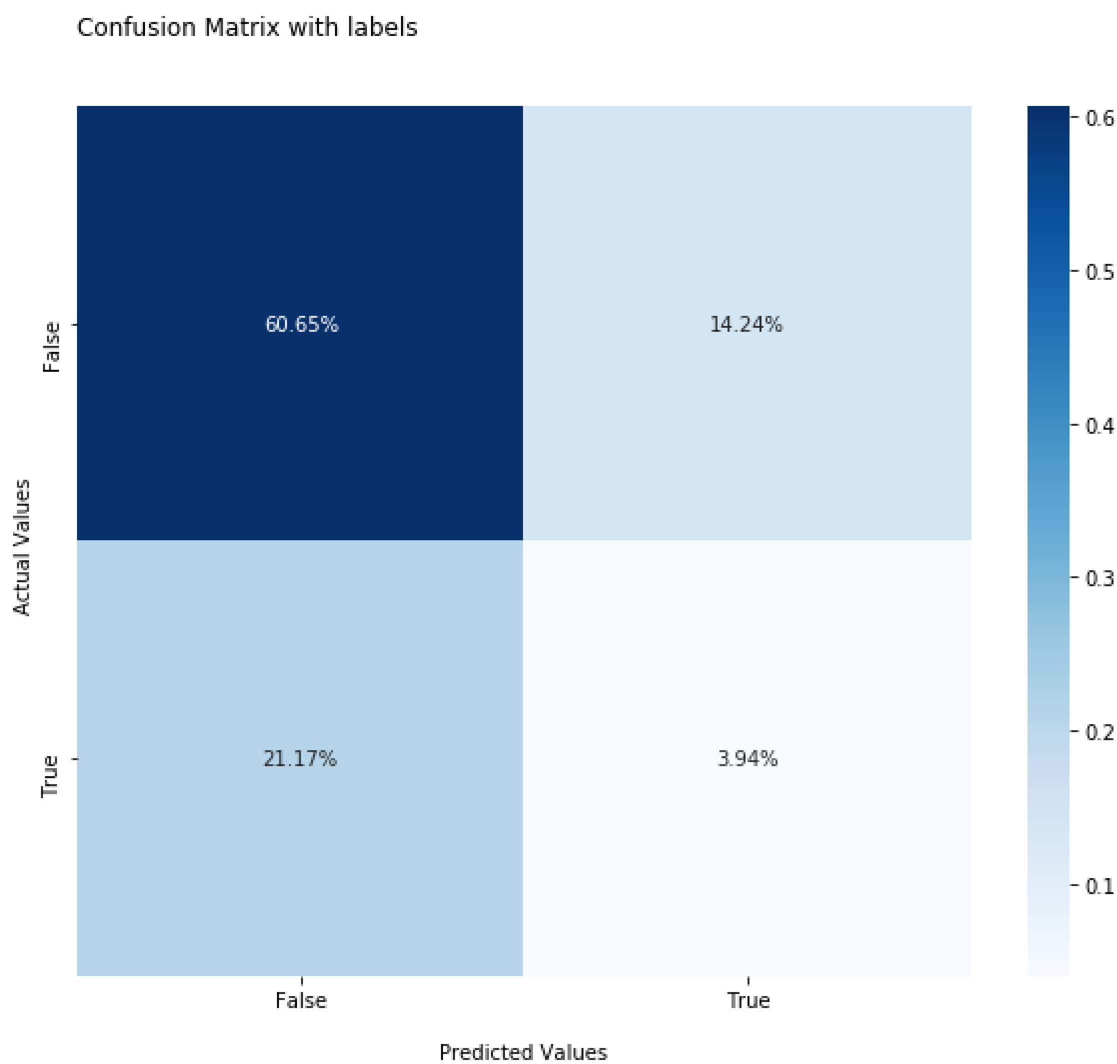
```
1 lstm3 = lstm_model(x_train, y_train, x_val, y_val, x_test, y_test,7,27,
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_16 (LSTM)               (None, 7, 4)              512
_________________________________________________________________
batch_normalization_16 (Batc (None, 7, 4)              16
_________________________________________________________________
lstm_17 (LSTM)               (None, 7, 4)              144
_________________________________________________________________
batch_normalization_17 (Batc (None, 7, 4)              16
_________________________________________________________________
lstm_18 (LSTM)               (None, 4)                 144
_________________________________________________________________
batch_normalization_18 (Batc (None, 4)                 16
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 5
=================================================================
Total params: 853
Trainable params: 829
Non-trainable params: 24
_________________________________________________________________
None
-----------------------------------------------------------------
Training was completed in 724.57 secs
-----------------------------------------------------------------


Score for Model Testing
8872/8872 [==============================] - 1s 73us/step
[1.2699199670655326, 0.6459648609161377]
```






```
Accuracy score: 64.6%
Precision score: 21.7%
Recall score: 15.71%
```




```
In [ ]: 1
```