Utilizing Data Science Techniques to Predict Wildfires in United States

Emina Belekanich, Bryan Flores, and Nima Amin Taghavi

University of San Diego

ADS 502: Applied Data Mining

August 16, 2021

Abstract

Wildfires have several advantages and disadvantages. In one sense, wildfires act as nature's method of clearing the way for new life to begin by reducing debris, managing grasslands, and improving soil health. On the other hand, although beneficial to the biome, wildfires cause severe damage to urban-wildland interfaces. For example, in 2018, wildfire damages totaled $148.5 billion in California (Wang et al., 2021), which reallocates funding from beneficial services. Improved wildfire predictions better our understanding of wildfires themselves and inform fire management services for better decision-making (Taylor et al., 2013). We examine five models within the fire weather criteria to predict wildfires in the United States: CART, C5.0, Random Forests, Naive Bayes, and Naive Bayes Multinomial. The model with the greatest accuracy, sensitivity, and specificity is the Random Forest algorithm, which yielded 84.6% accuracy, 93.2% sensitivity, and 67.7% specificity, respectively.

Table of Contents

List of Figures

List of Tables

1. Introduction

Wildfires are among nature's most destructive forces -- mostly occurring naturally -- improved wildfire prediction saves lives, natural resources, and money (Taylor et al., 2013). Unfortunately, wildfires have become a common event in the United States during the year's summer months. Over the past sixty years, not only have we seen an increase in wildfires, but their severity as well (Weber et al., 2020). A critical factor in determining whether a fire will start and spread is fire weather obtained from meteorological services (Jain et al., 2020).

Improving wildfire detection could lead to better fire disaster management and thus reduce wasted resources. Early artificial intelligence applications for wildfire science date back to the 1990s, with neural networks being the first. After 2012, Random Forests became the popular choice for predicting fire occurrence, while Decision Tree methods with bagging displayed the best precision, and Random Forests having the superior recall (Jain et al., 2020). This paper compares five machine learning methods to predict fire occurrence from the Wildland Fires Perimeters historical data.

Section 2, Project Lifecycle, discusses the lifecycle of the project. Section 3, Pre-processing, provides an overview of the data set and pre-processing methods and describes our practices for data exploration and handling missing data. Section 4, Exploratory Analysis, describes the exploratory analysis processes. Section 5, Methodology, provides an overview of each applied model. Section 6, Results, discusses our findings for the project, and lastly, Section 7, Conclusion, summarizes our work.

2. Project Lifecycle

In-order to successfully execute project objective, project went through several stages. ***Figure 1*** shows the high level of the data science life cycle from data import, preprocessing, getting weather information, modeling evaluation, etc.

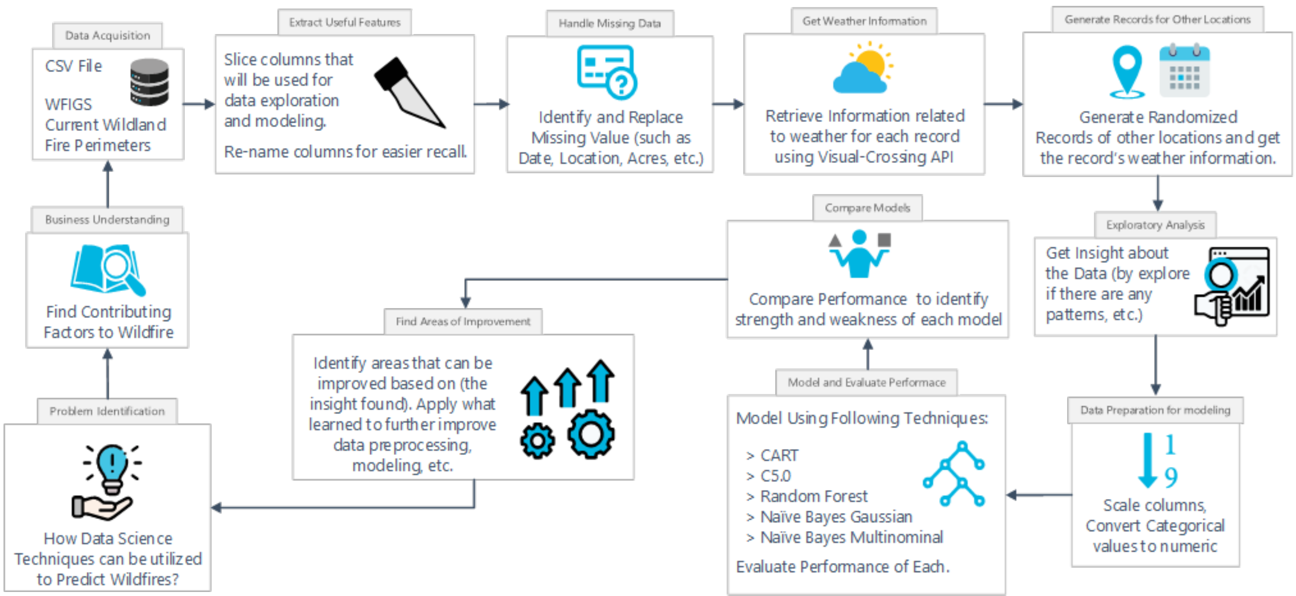


Figure 1 - *Project Life Cycle*

3. Pre-processing

This paper explores wildfires in the United States occurring between 2018 - 2021. Our goal is to use historical data to predict future wildfires while controlling several variables. We use the Wildland Fire Interagency Geospatial Services (WFIGS) Perimeters Full History dataset, consisting of 105 attributes and 7,125 records (as shown in **Table 1**) for all reported wildland fires in the United States, and open to the public from the WFIGS website in CSV format. Data acquisition for this dataset is ongoing and will increase in size over time as consolidation continues for interagency fire perimeter data. In addition, all incidents in this dataset are categorized in the IRWIN (Integrated Reporting of Wildland Fire Information) integration service and not "quarantined" in IRWIN due to potential conflicts with other records (WFIGS, 2021).

Table 1 – *Size of Original Dataset*

	Rows	Columns
Dataset Size	7125	105

Out of the 105 columns in the original dataset, ten columns were sliced out (shown in ***Table 2***). However, they help gain insights for section 5, Modeling Methodology. Refer to Appendix Pre-Process Section 1 and 2 for more detail.

Table 2 - *Usable Features that were sliced from Original Dataset*

	OBJECTID	Date_Start	Date_Finish	Acres	FireCause	Lat	Long	DispatchCenterID	PredominantFuelGroup	State
0	127	2020-10-18	2021-03-10	562.913504	Unknown	36.071140	-121.45050	CALPCC	NaN	US-CA
1	128	2020-05-01	NaN	0.151680	Unknown	39.556690	-119.55850	NVSFC	Grass	US-NV
2	129	2020-08-08	2020-08-20	0.300000	Human	33.293840	-110.45000	AZPHC	Grass-Shrub	US-AZ
3	130	2020-05-08	2020-05-26	44.300517	Human	35.875820	-115.20410	NVLIC	Grass-Shrub	US-NV
4	133	2020-08-21	2020-08-22	NaN	Human	NaN	NaN	SDGPC	Grass	US-SD
...	...	...	...	...	...	...	...	...	...	...
7120	12712	2021-07-07	NaN	191.735211	Natural	46.492160	-115.06930	IDGVC	NaN	US-ID
7121	12713	2021-05-03	NaN	38.700000	NaN	NaN	NaN	AZWDC	Timber	US-AZ
7122	12714	2021-06-06	2021-06-09	2.811551	Natural	42.122970	-113.96070	IDSCC	Brush	US-ID
7123	12716	2021-07-20	NaN	0.100000	Natural	41.108517	-119.46385	CASIFC	Grass-Shrub	US-NV
7124	12717	2021-07-12	NaN	NaN	Human	NaN	NaN	MNMNCC	Grass	US-MN

7125 rows × 10 columns

We observed missing data in numerous places. ***Table 3*** has the breakdown of how many missing values each column consisted of before cleaning the dataset.

Table 3 - *Columns with Missing Values*

	OBJECTID	Date_Start	Date_Finish	Acres	FireCause	Lat	Long	DispatchCenterID	PredominantFuelGroup	State
Count of Missing Values	0	0	1069	1698	60	1282	1282	3	1723	0

As shown in *Figure 2*, most fires lasted about ten days, while there are instances in which fire took more than two months to finish, **Figure *2*** is a histogram that shows distribution of how long fires last. We replace the missing values for Date_Finish columns by using the mean of the Fire_Duration column (in the unit of days). We find the mean of fire duration to be 14 days (for fire to finish). Refer to Appendix Pre-Process Section 3.2 for more detail.

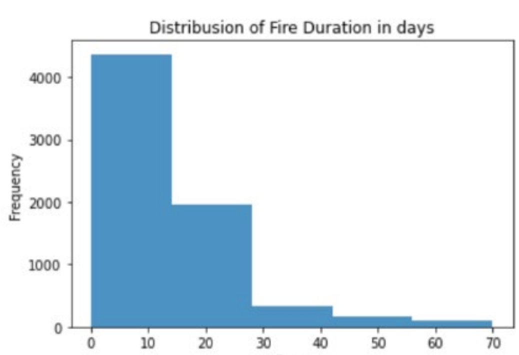


Figure 2 - *Distribution of Fire Duration*

Missing values for the location of fire were estimated using the mean of latitude and longitude based on each specific state. With the help of google public database, which contains the latitude and longitude of each state. We fill 84% of the records using this method; however, there were multiply states in which none of the records in that state had information about the latitude and longitude. Therefore, we use the Google public dataset to obtain those states' remaining records' location information. Refer to Appendix Pre-Process Section 3.6 for more detail.

We gather weather information such as temperature, humidity, pressure, etc., for each record through Visual Crossing API. In addition, it allows users to input variables (the date and address) and extract weather information as JSON code. Refer to Appendix Gathering Weather Information Section for more information.

4. Exploratory Analysis

After looking at frequency of fired occurred for each state, it was found that Montana, Arizona, Minnesota, and California (as shown in **Figure 3**) have the highest number of fires.

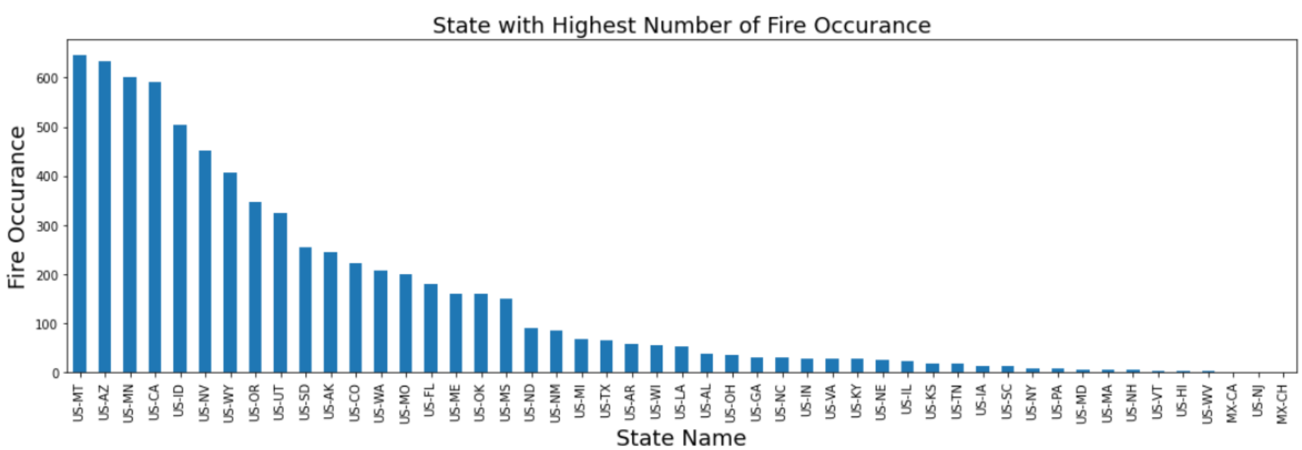


Figure 3- *Frequency of Fire Within Each State*

Furthermore, after exploring the data using pairwise relationships plot (shown in **Figure 4**), it can be observed that Quarter 2 and Quarter 3 of calendar year have higher temperature relative to the rest of calendar year (Refer to top left relationship chart to see the temperature distribution per quarter). It can also be observed that windspeed is higher these two quarters as well. These two important elements are some of the contributing factors as to why most of fire occurs in these two quarters.

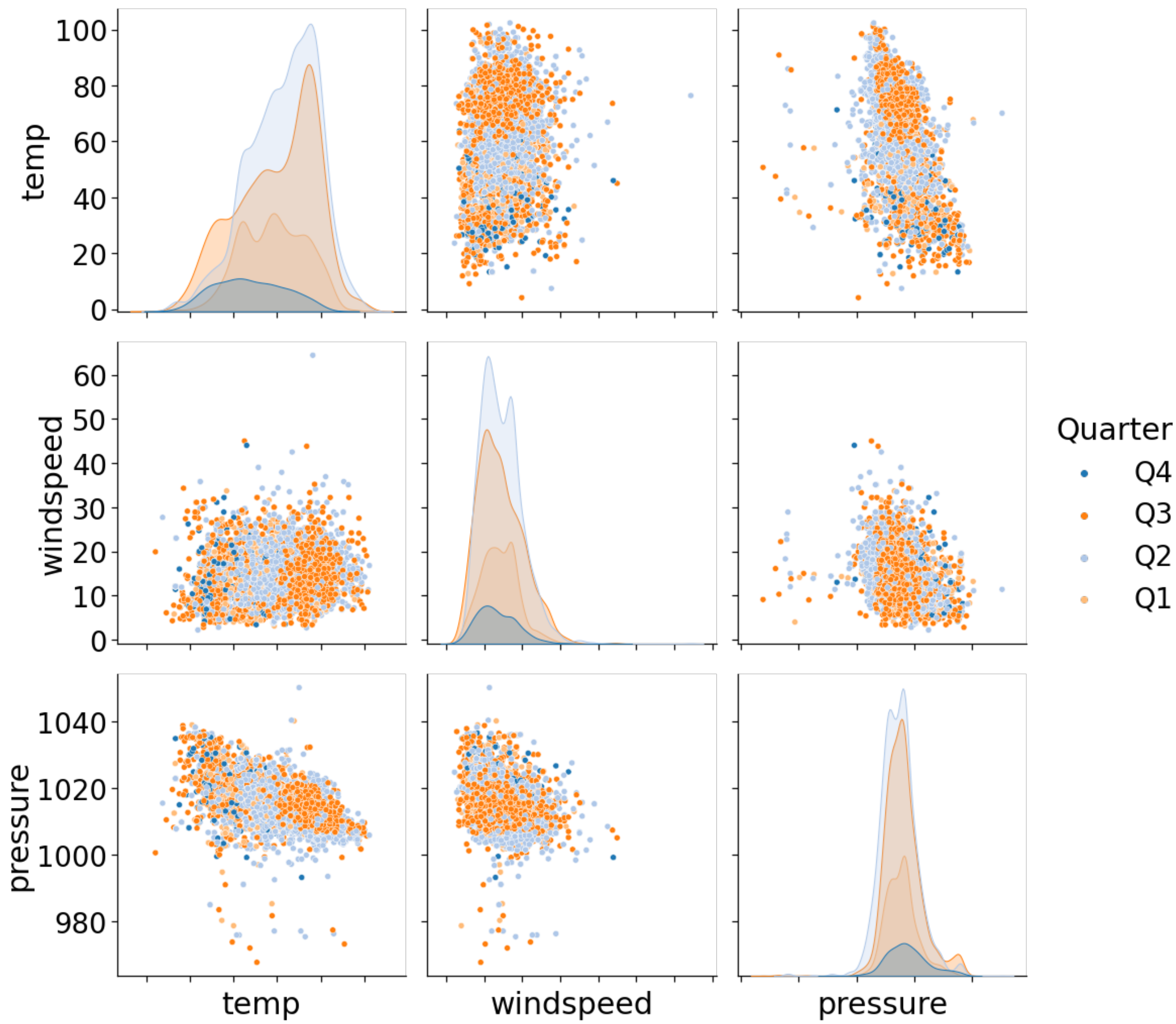


Figure 4 - Pairwise Relationship Plot

5. Modeling Methodology

Critical fire weather consists of but is not limited to unstable air, dry airmass, strong surface winds, and low relative humidity (NWCG, 2021). Therefore, we use the following three input variables to determine whether a fire occurred (target variable): temperature, humidity, and weather conditions. Since the target variable is binary, we predict a fire will happen given the three inputs. We compare the following models: CART, C5.0, Random Forests, Naive Bayes, and Naive Bayes Multinomial. To maintain consistency, each model's training and test set makes up 70% and 30%, respectively, of the original dataset.

5.1. Decision Tree – CART

Classification and Regression Trees, or CART, is a supervised learning algorithm producing binary trees, meaning there are two branches for every decision node. CART recursively partitions records into subsets with similar values for the target attribute while searching through all variables and splitting values to find the optimal split (Larose & Larose, 2019). The objective of decision trees, including C5.0, is to accurately capture the relationship between inputs and outputs using the smallest possible tree while avoiding overfitting (Jain et al., 2020). The maximum allowable number of leaf nodes is 10 due to the size of the data set and the number of input variables. The CART model yielded 75.6% accuracy, 87.1% sensitivity, and 52.9% specificity.

5.2. Decision Tree - C5.0

The CART model uses the Gini Index to calculate the probability of an incorrectly classified feature when selected randomly (Tyagi, 2020). The C5.0 algorithm is not restricted to binary splits and uses entropy reduction to choose the optimal split, selecting the split with the most significant information gain. Information gain increases information by partitioning the training data according to the candidate split (Larose & Larose, 2019). Like the CART model, we set the maximum allowable number of leaf nodes to 10. As a result, the C5.0 model yielded 75.4% accuracy, 94.1% sensitivity, and 38.8% specificity.

5.3. Decision Tree - Random Forest

The Random Forest is the third algorithm selected in the decision tree family. This approach builds on multiple decision trees and then merges them into one to predict a more accurate method. The algorithm splits a node by using the bootstrap method to develop its decision tree while searching for the most important feature amongst many random subsets of features. For our data set, we have created a random forest using 100 trees. This model yielded 84.6% model accuracy, 93.2% sensitivity, and 66.7% specificity.

5.4. Naïve Bayes – Gaussian

Another classification modeling technique used to predict whether fire occurred is Naïve Bayes. This Theorem was developed by Thomas Bayes which can reform its knowledge about the data by utilizing previous knowledge and can update its parameter knowledge (Larose et al., *Chapter 8* 2019).

Even though this technique was able to predict the event which fire occurred correctly (with Sensitivity of 85.5%), this method is not suitable since it is falsely predicting negative events which fire did not occur (with Specificity of 40.7%)

5.5. Naïve Bayes – Multinominal

Knowing the Gaussian algorithm was not a suitable approach, Multinomial algorithm was put in to test to check whether it can perform better especially with false positives scenarios. Multinomial failed to predict any events which fire did not occur. Hence the sensitivity is 100% and specificity is 0.0%.

6. Results

We lastly reviewed evaluation results for all five models using accuracy, sensitivity, specificity, and F1 Measure (***Table 4***). Accuracy was chosen to represent the proportion of correct classifications. We chose sensitivity to measure the model's ability to classify positivity of the record, while sensitivity was chosen to measure its negativity. Lastly, F1 score measured model's precision and recall. Based on these results, random forest had the highest accuracy, sensitivity, specificity and F1 Score, yielding 84.6%, 93.2%, 67.7%, and 78.4% respectively. Naïve Bayes Gaussian had the lowest accuracy, sensitivity, specificity and F1 Score coming in at 70.4%, 85.5%, 40.7%, and 55.1% respectively. Random Forest had the highest specificity score of 67.7% compared to other models, which implies that it is the best option for avoiding false positives.

Table 4 - *Evaluation Results*

Modeling Technique	Accuracy	Sensitivity	Specificity	F1Measure
Decision Tree - CART	0.756	0.871	0.529	0.658
Decision Tree - C5.0	0.754	0.941	0.388	0.549
Decision Tree - Random Forest	0.846	0.932	0.677	0.784
Naïve Bayes - Gaussian	0.704	0.855	0.407	0.551
Naïve Bayes - Multinomial	0.663	1.000	0.000	0.000

7. Conclusion

Our work shows there are many different methods for accurately predicting whether a fire will occur. The Random Forest (RF) produced the highest level of accuracy, and the highest level of sensitivity. RF displayed high performance in discerning “yes” a fire will occur, and satisfactory performance in determining a fire will not occur. Interestingly, the Naive Bayes showed good accuracy, and high performance in determining a fire will occur. However, the Naive Bayes model performs poorly when determining the fire will not occur. Like the RF algorithm, the two decision tree methods, CART and C5.0, performed well in all three phases. Comparing our work with that of Jain et al. (2020), who found that in most studies comparing machine learning methods, ensemble methods tended to outperform single classifier methods. Therefore, we recommend the Random Forest as the most appropriate model for future prediction efforts on this dataset.

References

Jain, P., Coogan, S. C., Subramanian, S. G., Crowley, M., Taylor, S., & Flannigan, M. D. (2020). A review of machine learning applications in wildfire science and management. *Environmental Reviews*, 28(4), 478-505.

Larose, C. D., Larose, D. T. (2019). Decision Trees. *Data Science Using Python and R* (pp. 81-96). Wiley.

National Interagency Fire Center. (2021, August 14). *WFIGS – Wildland Fire Perimeters Full History*. https://data-nifc.opendata.arcgis.com/datasets/nifc::wfigs-wildland-fire-perimeters-full-history/about

National Wildfire Coordinating Group. (2021, July 8). *Critical Fire Weather*. https://www.nwcg.gov/publications/pms437/weather/critical-fire-weather

Taylor, S. W., Woolford, D. G., Dean, C. B., & Martell, D. L. (2013). Wildfire prediction to inform fire management: statistical science challenges. Statistical Science, 28(4), 586-615.

Tyagi, N. (2020, March 23). *Understanding the Gini Index and Information Gain in Decision Trees*. Medium. https://medium.com/analytics-steps/understanding-the-gini-index-and-information-gain-in-decision-trees-ab4720518ba8

Wang, D., Guan, D., Zhu, S., Mac Kinnon, M., Geng, G., Zhang, Q., Zheng, H., Lei, T., Shao, S., Gong, P., & Davis, S. J. (2021). Economic footprint of California wildfires in 2018. *Nature Sustainability*, 4(3), 252-260.

Weber K.T., & Yadav R. (2020). Spatiotemporal Trends in Wildfires across the Western United States (1950–2019). *Remote Sensing*, 12(18), 7-9. https://doi.org/10.3390/rs12182959

Appendix

```
# Location of where files are saved.
original_df                  = 'WildlandFirePerimeters.csv'
PreProcessed_df              = 'csv_file_preprocessing\df_A.csv'
Weather_WithAddress          = 'csv_file_preprocessing\df_B_Stage1.csv'
Weather_Fire_WeatherInfo     = 'csv_file_preprocessing\df_B_Stage2.csv'
Weather_NoFire_WeatherInfo   = 'csv_file_preprocessing\df_B_Stage3.csv'
Weather_Final_Cleaned_df     = 'df_Cleaned.csv'
```

Pre-Process

Pre-Process (Section 1)

```
#Import required libraries
import numpy as np, pandas as pd, matplotlib as mpl
import matplotlib.pyplot as plt, seaborn as sns, requests
from pandas import to_datetime as dt
from datetime import timedelta
```

```
# Load database as DataFrame
df_ORG = pd.read_csv(original_df)
```

```
# Checking for overall statistics of original dataset (mean, number of missing records, etc.)

stat = df_ORG.describe()

display(pd.DataFrame([df_ORG.shape[0],df_ORG.shape[1]],
                     columns=['Dataset Size'],index = ['Rows','Columns']).T)

for col in df_ORG.columns:
    stat.loc['Missing',col] = len(df_ORG[df_ORG[col].isnull() == True])
    stat.loc['Zero',col] = len(df_ORG[df_ORG[col] == 0])

stat.style.format("{:.0f}")
```

	Rows	Columns
Dataset Size	7125	105

	OBJECTID	poly_GISAcres	poly_Acres_AutoCalc	irwin_CalculatedAcres	irwin_DailyAcres	irwin_DiscoveryAcres	irwin_EstimatedCostToDate	irwin_
count	7125	5427	7119	2025	7060	6444	1241	
mean	7926	1980	1677	5307	1794	74	2868080	
std	3756	17315	15528	29629	19141	1640	12483849	
min	127	0	0	0	0	0	0	
25%	7147	1	1	4	1	0	20000	
50%	8950	4	8	39	8	1	200000	
75%	10817	65	80	674	80	3	1075085	
max	12717	589368	589833	589835	1032648	115997	193000000	
Missing	0	1698	6	5100	65	681	5884	
Zero	0	11	0	0	0	0	35	

```
# Checking for columns headers (variable names)
print(f'The dataset has {len(df_ORG.columns)} columns.')
pd.DataFrame(df_ORG.columns,columns =['Column Name'])
```

```
The dataset has 105 columns
```

	Column Name
0	OBJECTID
1	poly_IncidentName
2	poly_FeatureCategory
3	poly_MapMethod
4	poly_GISAcres
...	...
100	irwin_ModifiedOnDateTime_dt
101	irwin_CreatedOnDateTime_dt
102	GlobalID
103	SHAPE_Length
104	SHAPE_Area

105 rows × 1 columns

Pre-Process (Section 2)

```
# Slicing the columns that will be used for modeling

Column_remap = {
'OBJECTID'                      : 'OBJECTID',
'irwin_FireDiscoveryDateTime'   : 'Date_Start',
'irwin_FireOutDateTime'         : 'Date_Finish',
'poly_GISAcres'                 : 'Acres',
'irwin_FireCause'               : 'FireCause',
'irwin_InitialLatitude'         : 'Lat',
'irwin_InitialLongitude'        : 'Long',
'irwin_POODispatchCenterID'     : 'DispatchCenterID',
'irwin_PredominantFuelGroup'    : 'PredominantFuelGroup',
'irwin_POOState'                : 'State'}

df = df_ORG.rename(columns=Column_remap)
df = df[Column_remap.values()]

# Standardize the date column
df.Date_Start = pd.to_datetime(df.Date_Start).dt.strftime('%Y-%m-%d')
df.Date_Finish = pd.to_datetime(df.Date_Finish).dt.strftime('%Y-%m-%d')

df
```

	OBJECTID	Date_Start	Date_Finish	Acres	FireCause	Lat	Long	DispatchCenterID	PredominantFuelGroup	State
0	127	2020-10-18	2021-03-10	562.913504	Unknown	36.071140	-121.45050	CALPCC	NaN	US-CA
1	128	2020-05-01	NaN	0.151680	Unknown	39.556690	-119.55850	NVSFC	Grass	US-NV
2	129	2020-08-08	2020-08-20	0.300000	Human	33.293840	-110.45000	AZPHC	Grass-Shrub	US-AZ
3	130	2020-05-08	2020-05-26	44.300517	Human	35.875820	-115.20410	NVLIC	Grass-Shrub	US-NV
4	133	2020-08-21	2020-08-22	NaN	Human	NaN	NaN	SDGPC	Grass	US-SD
...	...	...	...	...	...	...	...	...	...	...
7120	12712	2021-07-07	NaN	191.735211	Natural	46.492160	-115.06930	IDGVC	NaN	US-ID
7121	12713	2021-05-03	NaN	38.700000	NaN	NaN	NaN	AZWDC	Timber	US-AZ
7122	12714	2021-06-06	2021-06-09	2.811551	Natural	42.122970	-113.96070	IDSCC	Brush	US-ID
7123	12716	2021-07-20	NaN	0.100000	Natural	41.108517	-119.46385	CASIFC	Grass-Shrub	US-NV
7124	12717	2021-07-12	NaN	NaN	Human	NaN	NaN	MNMNCC	Grass	US-MN

7125 rows × 10 columns

Pre-Process (Section 3.1)

```
# Check how many values are missing in each column.

pd.DataFrame(df.isnull().sum(), columns = ['Count of Missing Values']).T
```

	OBJECTID	Date_Start	Date_Finish	Acres	FireCause	Lat	Long	DispatchCenterID	PredominantFuelGroup	State
Count of Missing Values	0	0	1069	1698	60	1282	1282	3	1723	0

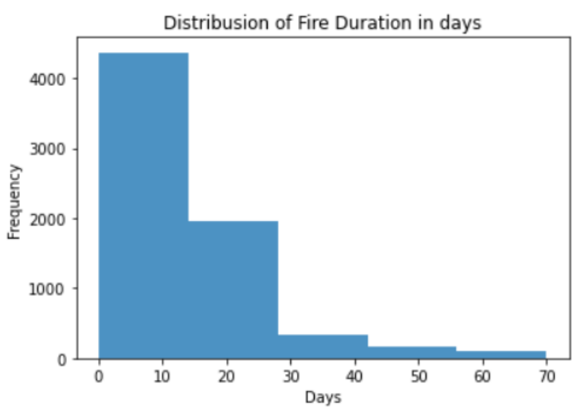
Pre-Process (Section 3.2)

```
# Filling up the Missing Date_Finish

Empty_Date_Columns = df[df.Date_Finish.isnull() == True]
MissingValuesCount = len(Empty_Date_Columns)

print(f"{MissingValuesCount/len(df):.0%} in Date_Finish column",
      f"({MissingValuesCount:,} records) have missing values.\n")

# Find Duration of Fire in unit of days and store values in integer
df['Fire_Duration'] = dt(df.Date_Finish) - dt(df.Date_Start)
df['Fire_Duration'] = df['Fire_Duration'].dt.days

# Get mean of fire duration
Fire_Duration_mean  = df['Fire_Duration'].mean()

print(f"It took ~ {Fire_Duration_mean:.0f} days for fires to be finshed (based column's mean.\n")

# Calculate Date_Finish for empty records based on Containment_Duration_mean.
for index in Empty_Date_Columns.index:
    df.loc[index, 'Date_Finish'] = \
        dt(df.loc[index, 'Date_Start']) + timedelta(days=Fire_Duration_mean)

# Show distribution of Fire Duration
ax = df.Fire_Duration.plot.hist(bins=5, alpha=0.8,range=[0,70])
```

```
# Add title and axis names
plt.title('Distribusion of Fire Duration in days')
plt.xlabel('Days')
plt.ylabel('Frequency')

# To Verify
Empty_Date_Columns = df[df.Date_Finish.isnull() == True]
print(f'There are now {len(Empty_Date_Columns)} records with missing records in Date_Finish columns
 after clean up.')

# Convert to standard format and display first 5 rows
df.Date_Finish = pd.to_datetime(df.Date_Finish).dt.strftime('%Y-%m-%d'); df.head()
```

```
15% in Date_Finish column (1,069 records) have missing values.

It took ~14 days for fires to be finshed (based column's mean.

There are now 0 records with missing records in Date_Finish columns after clean up.
```

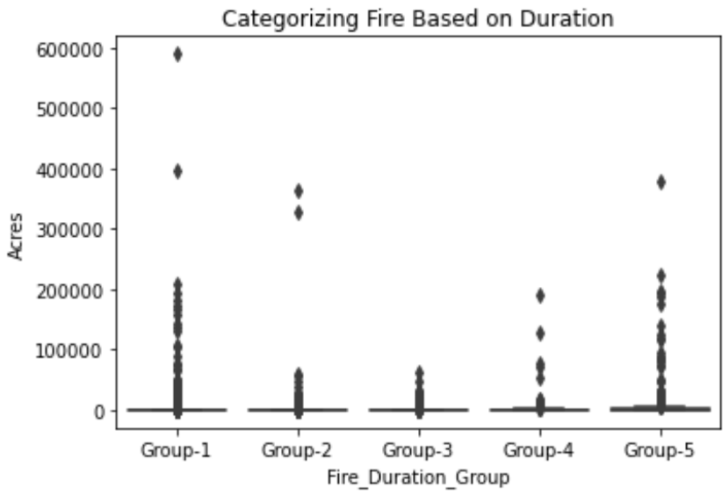
	OBJECTID	Date_Start	Date_Finish	Acres	FireCause	Lat	Long	DispatchCenterID	PredominantFuelGroup	State	Fire_Duration
0	127	2020-10-18	2021-03-10	562.913504	Unknown	36.07114	-121.4505	CALPCC	NaN	US-CA	143
1	128	2020-05-01	2020-05-15	0.151680	Unknown	39.55669	-119.5585	NVSFC	Grass	US-NV	14
2	129	2020-08-08	2020-08-20	0.300000	Human	33.29384	-110.4500	AZPHC	Grass-Shrub	US-AZ	12
3	130	2020-05-08	2020-05-26	44.300517	Human	35.87582	-115.2041	NVLIC	Grass-Shrub	US-NV	18
4	133	2020-08-21	2020-08-22	NaN	Human	NaN	NaN	SDGPC	Grass	US-SD	1



Pre-Process (Section 3.3)

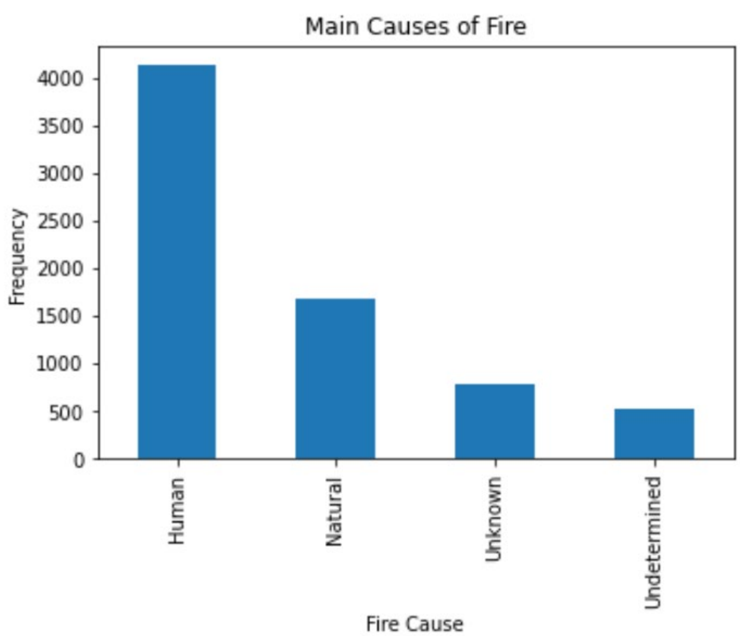
```
# Replace missing records in Acres column with column's median

Empty_Acres_Columns = df[df.Acres.isnull() == True]
df.loc[Empty_Acres_Columns.index,'Acres'] = df.Acres.median()

Fire_Duration_Group = dict(
    [(n, 'Group-1') for n in range(0, 15)]   +
    [(n, 'Group-2') for n in range(15, 30)]  +
    [(n, 'Group-3') for n in range(30, 45)]  +
    [(n, 'Group-4') for n in range(45, 60)]  +
    [(n, 'Group-5') for n in range(60, 1000)])

for i in df.index:
    Duration = df.loc[i,'Fire_Duration']
    try:
        df.loc[i,'Fire_Duration_Group'] = Fire_Duration_Group[Duration]
    except:pass
```

```
df_plot = pd.DataFrame(data = df[['Acres','Fire_Duration_Group']], columns = ['Acres','Fire_Durati
on_Group'])
df_plot.sort_values(by='Fire_Duration_Group', ascending=True, inplace = True)
sns.boxplot(x="Fire_Duration_Group", y="Acres", data=df_plot).set_title('Categorizing Fire Based
on Duration')
plt.show()
```



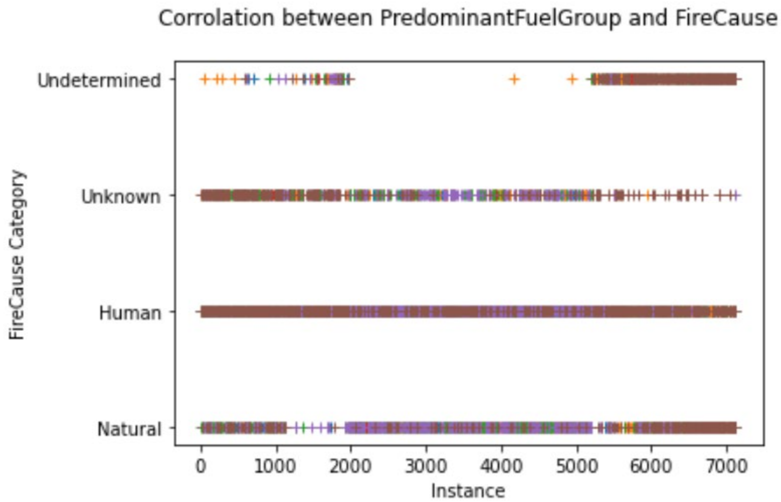
Pre-Process (Section 3.4)

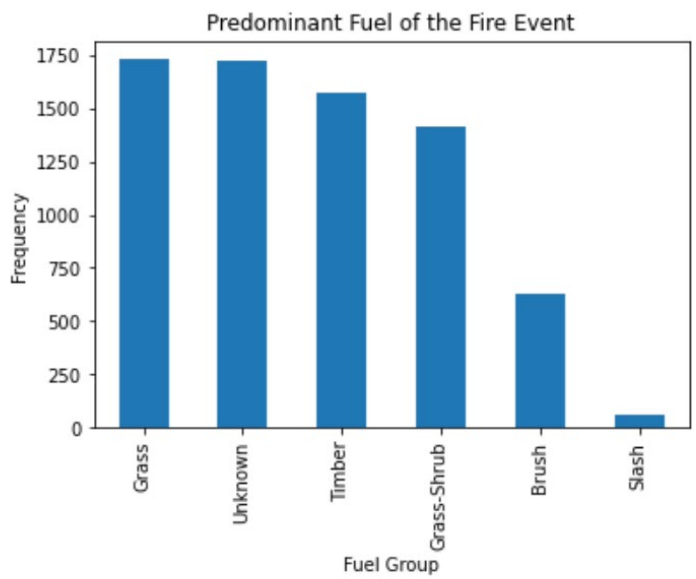
```
# Replace missing records in FireCause column with 'Unknown'

Empty_FireCause_Columns = df[df.FireCause.isnull() == True]
df.loc[Empty_FireCause_Columns.index,'FireCause'] = 'Unknown'

df['FireCause'].value_counts().plot(kind='bar',
                                    xlabel='Fire Cause',
                                    ylabel='Frequency',
                                    title='Main Causes of Fire')
```



Pre-Process (Section 3.5)

```
# Replace missing records in PredominantFuelGroup column with 'Unknown'

Empty_PredominantFuelGroup_Columns = df[df.PredominantFuelGroup.isnull() == True]
df.loc[Empty_PredominantFuelGroup_Columns.index,'PredominantFuelGroup'] = 'Unknown'

# As shown in graph, most of missing PredominantFuelGroup values have undetermined FireCause.

groups = df.groupby("PredominantFuelGroup")

for name, group in groups:
    plt.plot(group["FireCause"], marker="+", linestyle="", label=name)

plt.xlabel('Instance')
plt.ylabel('FireCause Category')
plt.title('Corrolation between PredominantFuelGroup and FireCause\n')
plt.show()

df['PredominantFuelGroup'].value_counts().plot(kind='bar',
                                               xlabel='Fuel Group',
                                               ylabel='Frequency',
                                               title ='Predominant Fuel of the Fire Event')
```





Pre-Process (Section 3.6.1)

```
# Replace missing records for Latitude and Longtitude usin mean of other occured fire in that State.
Empty_Location_Columns = df[df.Lat.isnull() == True]

for i in Empty_Location_Columns.index:
    Row_State = df.loc[i,'State']
    df.loc[i,'Lat'] = df.loc[df.State == Row_State,'Lat'].mean()
    df.loc[i,'Long'] = df.loc[df.State == Row_State,'Long'].mean()

print(f'There were {len(Empty_Location_Columns)} rows had empty values.')

Empty_Location_Columns_After = df[df.Lat.isnull() == True]
Still_Left = len(Empty_Location_Columns_After) / len(Empty_Location_Columns)
print(f'{1-Still_Left:0.0%} of the records were filled using this method.')
```

```
There were 1282 rows had empty values.
84% of the records were filled using this method.

An exception has occurred, use %tb to see the full traceback.
```

Pre-Process (Section 3.6.2)

```
Empty_Location_Columns_Stage1 = df[df.Lat.isnull() == True]

print(f'{len(Empty_Location_Columns_Stage1)} records still have missing data since no location
was recorded in that state.')

# Get name of state with no location record.
print('\n', 'States with no record: ', Empty_Location_Columns_Stage1.State.unique(),'\n')

# Fill empty records using Latitude and Longtitude gathered by google
# Refer to https://developers.google.com/public-data/docs/canonical/states_csv for state

url = 'https://developers.google.com/public-data/docs/canonical/states_csv'
html = requests.get(url).content
Ref_Table = pd.read_html(html)[-1]

Column_remap = {
'latitude'  : 'Lat',
'longitude' : 'Long',
'state'     : 'State'}
Ref_Table.index = Ref_Table.name

Ref_Table.rename(columns=Column_remap, inplace = True)
Ref_Table.drop(['name'], axis = 1, inplace = True)
Ref_Table.State = "US-" + Ref_Table.State

Ref_Table.T
```

```
199 records still have missing data since no location was recorded in that state.

 States with no record:  ['US-MA' 'US-NY' 'US-ME' 'US-IA' 'US-HI' 'US-MD' 'US-NJ']
```

name	Alaska	Alabama	Arkansas	Arizona	California	Colorado	Connecticut	District of Columbia	Delaware	Florida	...	South Dakota	Tennessee
State	US-AK	US-AL	US-AR	US-AZ	US-CA	US-CO	US-CT	US-DC	US-DE	US-FL	...	US-SD	US-TN
Lat	63.588753	32.318231	35.20105	34.048928	36.778261	39.550051	41.603221	38.905985	38.910832	27.664827	...	43.969515	35.517491
Long	-154.493062	-86.902298	-91.831833	-111.093731	-119.417932	-105.782067	-73.087749	-77.033418	-75.52767	-81.515754	...	-99.901813	-86.580447

3 rows × 52 columns

Pre-Process (Section 3.6.3)

```
# Get index of recods that still have not been filled
Empty_Location_Columns_Stage2 = \
    Empty_Location_Columns_Stage1[Empty_Location_Columns_Stage1.Lat.isnull() == True]

# Map Latitude and Longtitude using Ref_Table
df.loc[Empty_Location_Columns_Stage2.index,'Lat'] = \
    Empty_Location_Columns_Stage2.State.replace(Ref_Table.set_index('State')['Lat'])

df.loc[Empty_Location_Columns_Stage2.index,'Long'] = \
    Empty_Location_Columns_Stage2.State.replace(Ref_Table.set_index('State')['Long'])

# Verify that all records now have latitude and longtitude values.
Empty_Location_Columns_Final = df[df.Lat.isnull() == True]
print(f'There are now {len(Empty_Location_Columns_Final)} record(s) with empty location values.')

# Print the ones that were replaced.
df.loc[Empty_Location_Columns_Stage2.index,['State','Lat','Long']].head()
```

```
There are now 0 record(s) with empty location values.
```

	State	Lat	Long
10	US-MA	42.407211	-71.382437
73	US-NY	43.299428	-74.217933
131	US-ME	45.253783	-69.445469
178	US-ME	45.253783	-69.445469
302	US-MA	42.407211	-71.382437

Pre-Process (Section 3.7)

```
# Save Dataframe to be used for next section
df.to_csv(PreProcessed_df)
```

Gathering Weather Information

Gathering Weather Information (Section 1)

```
# Import Required libraries
import pandas as pd, json, requests, ast
from datetime import datetime
from geopy.geocoders import Nominatim

geolocator = Nominatim(user_agent="geoapiExercises")
```

Gathering Weather Information (Section 2)

```
# Load Cleanned data to df dataframe
df = pd.read_csv(PreProcessed_df)
df.head()
```

	OBJECTID	Date_Start	Date_Finish	Acres	FireCause	Lat	Long	DispatchCenterID	PredominantFuelGroup	State	Fire_Duration	Fire_
0	127	10/18/2020	3/10/2021	562.913504	Unknown	36.071140	-121.450500	CALPCC	Unknown	US-CA	143	
1	128	5/1/2020	5/15/2020	0.151680	Unknown	39.556690	-119.558500	NVSFC	Grass	US-NV	14	
2	129	8/8/2020	8/20/2020	0.300000	Human	33.293840	-110.450000	AZPHC	Grass-Shrub	US-AZ	12	
3	130	5/8/2020	5/26/2020	44.300517	Human	35.875820	-115.204100	NVLIC	Grass-Shrub	US-NV	18	
4	133	8/21/2020	8/22/2020	4.000000	Human	44.035131	-103.036037	SDGPC	Grass	US-SD	1	

Gathering Weather Information (Section 3.1)

```
# Get weather information for the records and store them in the dataframe
for i in df.index:
    Lat = df.loc[i,'Lat'].astype(str)
    Long = df.loc[i,'Long'].astype(str)
    df.loc[i,'Address'] = str(geolocator.reverse(f'{Lat},{Long}'))
    print(pd.DataFrame(df.loc[i,['State','Lat','Long','Address']]).T,'\n')
```

```
  State       Lat      Long                                  Address
0 US-CA  36.07114 -121.4505  Monterey County, California, United States

  State       Lat      Long                                     Address
1 US-NV  39.55669 -119.5585  NV-655, Washoe County, Nevada, United States

  State       Lat    Long                                          Address
2 US-AZ  33.29384 -110.45  2, West la Bamba, Graham County, Arizona, 8554...

  State       Lat      Long                             Address
3 US-NV  35.87582 -115.2041  Clark County, Nevada, United States

  State        Lat         Long  \
4 US-SD  44.035131 -103.036037
```

```
# Save it to dataframe to be used for next stage
df.to_csv(Weather_WithAddress)
df[['State','Lat','Long','Address']].head(10)
```

	State	Lat	Long	Address
0	US-CA	36.071140	-121.450500	Monterey County, California, United States
1	US-NV	39.556690	-119.558500	28, I 80, Patrick, Washoe County, Nevada, 8943...
2	US-AZ	33.293840	-110.450000	2, West la Bamba, Graham County, Arizona, 8554...
3	US-NV	35.875820	-115.204100	Clark County, Nevada, United States
4	US-SD	44.035131	-103.036037	Pennington County, South Dakota, United States
5	US-AZ	34.455900	-114.371900	981, Beachcomber Boulevard, Lake Havasu City, ...
6	US-AZ	33.336000	-110.468600	Powerline Road, Gila County, Arizona, 85542, U...
7	US-OK	34.892490	-95.261150	Blue Mountain Road, Latimer County, Oklahoma, ...
8	US-OK	34.602440	-98.686040	991, Post Oak Road, Comanche County, Oklahoma,...
9	US-MT	46.145110	-114.045900	Ravalli County, Montana, United States

Gathering Weather Information (Section 3.2)

```
# Get Number of records that geolocator did not find its address.
Empty_Address_Columns = df[df.Address == 'None']

print(f'{len(Empty_Address_Columns)/len(df):0.2%} ({len(Empty_Address_Columns)} records)',
      'were not able to get address based on their locations.')

# Replace Address by mean of State's Long and Lat location
for i in Empty_Address_Columns.index:
    Lat = df.loc[df.State == Empty_Address_Columns.loc[i,'State'],'Lat'].mean()
    Long = df.loc[df.State == Empty_Address_Columns.loc[i,'State'],'Long'].mean()
    df.loc[i,'Address'] = str(geolocator.reverse(f'{Lat},{Long}'))

# Verify that all records have address
Empty_Address_Columns_Final = df[df.Lat.isnull() == True]
print(f'There are now {len(Empty_Address_Columns_Final)} record(s) with empty Adress values.')

# Overwrite the fully filled dataframe
df.to_csv(Weather_WithAddress)
```

```
0.21% (15 records) were not able to get address based on their locations.
There are now 0 record(s) with empty Adress values.
```

Gathering Weather Information (Section 3.3.1)

```
# Create a Function to get Weather Information

URL_base = 'https://weather.visualcrossing.com/VisualCrossingWebServices/rest/services/timeline/'
API_Key = 'XXXXXXXXXXXXXXXXXXXXXXXXX'

suffix = '?unitGroup=us&key=' + API_Key

def PlaceWeatherInfo(row):
    row_address = df.loc[row,'URLAddress']

    row_date = str(pd.to_datetime(df.loc[row,'Date_Start'])).split()[0]
    URL_full = f'{URL_base}{row_address}/{row_date}{suffix}'
    try:
        response = requests.get(URL_full)
        JSONCode = json.loads(response.text)
        WeatherInfo = str(JSONCode.get('days'))[1:-1]
        BreakDown = ast.literal_eval(WeatherInfo)

        df.loc[row,'tempmax'] = BreakDown.get('tempmax')
        df.loc[row,'tempmin'] = BreakDown.get('tempmin')
        df.loc[row,'temp'] = BreakDown.get('temp')
        df.loc[row,'humidity'] = BreakDown.get('humidity')
        df.loc[row,'precip'] = BreakDown.get('precip')
        df.loc[row,'windspeed'] = BreakDown.get('windspeed')
        df.loc[row,'pressure'] = BreakDown.get('pressure')
        df.loc[row,'visibility'] = BreakDown.get('visibility')
        df.loc[row,'solarradiation'] = BreakDown.get('solarradiation')
        df.loc[row,'conditions'] = BreakDown.get('conditions')
    except:
        pass
```

Gathering Weather Information (Section 3.3.2)

```
replace_with = {',':'%2C', ' ':'%20', '/':'%2F'}
df['URLAddress'] = df['Address'].replace(replace_with, regex=True)

for row in range(1, 10):
    PlaceWeatherInfo(row)

df.to_csv(Weather_Fire_WeatherInfo)
```

Gathering Weather Information (Section 3.3.3)

```
# Setup States Dictionary which will be used to generate random zip codes
States_Dictionary = \
   [{'min': 35000, 'max':36999, 'code': 'US-AL'}, {'min': 99500, 'max':99999, 'code': 'US-AK'},
    {'min': 85000, 'max':86999, 'code': 'US-AZ'}, {'min': 71600, 'max':72999, 'code': 'US-AR'},
    {'min': 90000, 'max':96699, 'code': 'US-CA'}, {'min': 80000, 'max':81999, 'code': 'US-CO'},
    {'min': 6000,  'max':6999,  'code': 'US-CT'}, {'min': 19700, 'max':19999, 'code': 'US-DE'},
    {'min': 32000, 'max':34999, 'code': 'US-FL'}, {'min': 30000, 'max':31999, 'code': 'US-GA'},
    {'min': 96700, 'max':96999, 'code': 'US-HI'}, {'min': 83200, 'max':83999, 'code': 'US-ID'},
    {'min': 60000, 'max':62999, 'code': 'US-IL'}, {'min': 46000, 'max':47999, 'code': 'US-IN'},
    {'min': 50000, 'max':52999, 'code': 'US-IA'}, {'min': 66000, 'max':67999, 'code': 'US-KS'},
    {'min': 40000, 'max':42999, 'code': 'US-KY'}, {'min': 70000, 'max':71599, 'code': 'US-LA'},
    {'min': 3900,  'max':4999,  'code': 'US-ME'}, {'min': 20600, 'max':21999, 'code': 'US-MD'},
    {'min': 1000,  'max':2799,  'code': 'US-MA'}, {'min': 48000, 'max':49999, 'code': 'US-MI'},
    {'min': 55000, 'max':56999, 'code': 'US-MN'}, {'min': 38600, 'max':39999, 'code': 'US-MS'},
    {'min': 63000, 'max':65999, 'code': 'US-MO'}, {'min': 59000, 'max':59999, 'code': 'US-MT'},
    {'min': 27000, 'max':28999, 'code': 'US-NC'}, {'min': 58000, 'max':58999, 'code': 'US-ND'},
    {'min': 68000, 'max':69999, 'code': 'US-NE'}, {'min': 88900, 'max':89999, 'code': 'US-NV'},
    {'min': 3000,  'max':3899,  'code': 'US-NH'}, {'min': 7000,  'max':8999,  'code': 'US-NJ'},
    {'min': 87000, 'max':88499, 'code': 'US-NM'}, {'min': 10000, 'max':14999, 'code': 'US-NY'},
    {'min': 43000, 'max':45999, 'code': 'US-OH'}, {'min': 73000, 'max':74999, 'code': 'US-OK'},
    {'min': 97000, 'max':97999, 'code': 'US-OR'}, {'min': 15000, 'max':19699, 'code': 'US-PA'},
    {'min': 300  , 'max':999,   'code': 'US-PR'}, {'min': 2800 , 'max':2999,  'code': 'US-RI'},
    {'min': 29000, 'max':29999, 'code': 'US-SC'}, {'min': 57000, 'max':57999, 'code': 'US-SD'},
    {'min': 37000, 'max':38599, 'code': 'US-TN'}, {'min': 75000, 'max':79999, 'code': 'US-TX'},
    {'min': 88500, 'max':88599, 'code': 'US-TX'}, {'min': 84000, 'max':84999, 'code': 'US-UT'},
    {'min': 5000,  'max':5999,  'code': 'US-VT'}, {'min': 22000, 'max':24699, 'code': 'US-VA'},
    {'min': 20000, 'max':20599, 'code': 'US-DC'}, {'min': 98000, 'max':99499, 'code': 'US-WA'},
    {'min': 24700, 'max':26999, 'code': 'US-WV'}, {'min': 53000, 'max':54999, 'code': 'US-WI'},
    {'min': 82000, 'max':83199, 'code': 'US-WY'}]

def GetState(zipcode):
    WithinStateRange = lambda : 'Found' if int(zipcode) in range(state['min'], state['max'] + 1)
else None

    for state in States_Dictionary:
        if WithinStateRange() == 'Found': return state['code']
```

Gathering Weather Information (Section 3.3.4)

```
import datetime, random

# Set the algorithm that will generate random dates
start_date = datetime.date(2019, 12, 1)
end_date = datetime.date(2021, 7, 1)

time_between_dates = end_date - start_date
days_between_dates = time_between_dates.days
```

Gathering Weather Information (Section 3.3.5)

```
# Create random dates and zipcode using Perform function

def Perform(i):
    # Placing Random Date
    random_number_of_days  = random.randrange(days_between_dates)
    df.loc[i,'Date_Start'] = start_date + datetime.timedelta(days=random_number_of_days)

    # Placing Random Zipcode
    random_zipcde = random.randint(300,99999)
    Row_State = GetState(random_zipcde)
    df.loc[i,'Address']    = random_zipcde
    df.loc[i,'URLAddress'] = random_zipcde

    # Placing State and location based on the zipcode
    df.loc[i,'State']      = Row_State
    df.loc[i,'Lat']        = Ref_Table.loc[Ref_Table.State == Row_State,'Lat'][0]
    df.loc[i,'Long']       = Ref_Table.loc[Ref_Table.State == Row_State,'Long'][0]

    # Placing Weather information
    PlaceWeatherInfo(i)

    try:
        outcome = df.loc[i,'conditions']
    except: outcome = None
    return outcome

Counter = 14279

for i in range(len(df),Counter+len(df)):
    outcome = None
    while pd.isnull(outcome) == True:
        (outcome:=Perform(i))

df.to_csv(Weather_Final_Cleaned_df)
```

Exploratory Analysis

Exploratory Analysis (Section 1)

```
import numpy as np, pandas as pd, matplotlib as mpl, matplotlib.pyplot as plt, seaborn as sns
from datetime import datetime as dt

df = pd.read_csv(Weather_Final_Cleaned_df)
df['Month'] = pd.DatetimeIndex(df['Date_Start']).month

for i in df.index:
    df.loc[i,'Quarter'] = f"Q{(df.loc[i,'Month']-1)//3+1}"

df_Fire = df.query('FireOccured == 1')
df_Fire.head()
```

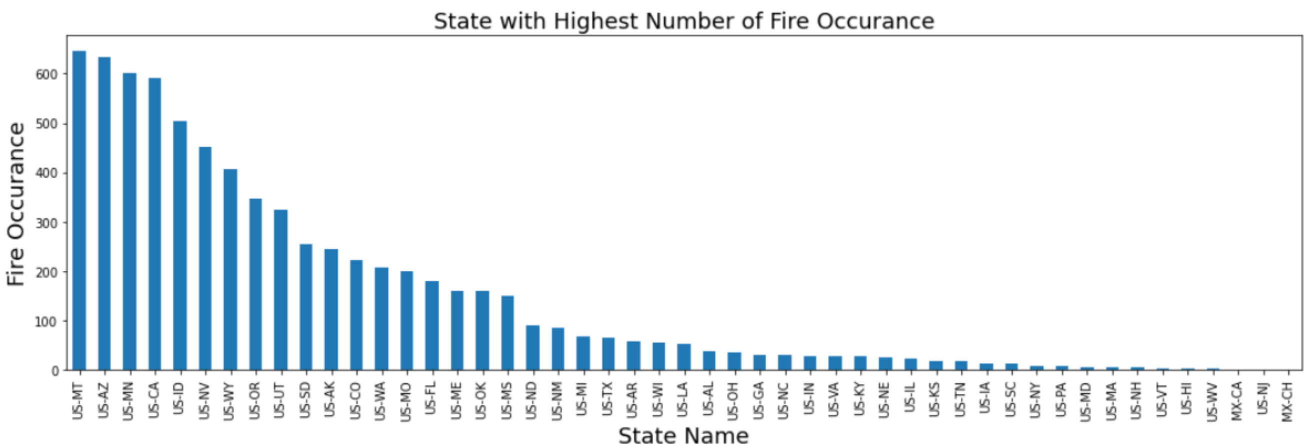
	OBJECTID	Date_Start	Date_Finish	Acres	FireCause	Lat	Long	DispatchCenterID	PredominantFuelGroup	State	...	humidity	precip
0	127.0	10/18/2020	3/10/2021	562.913504	Unknown	36.071140	-121.450500	CALPCC	Unknown	US-CA	...	67.29	0.00
1	128.0	5/1/2020	5/15/2020	0.151680	Unknown	39.556690	-119.558500	NVSFC	Grass	US-NV	...	25.39	0.00
2	129.0	8/8/2020	8/20/2020	0.300000	Human	33.293840	-110.450000	AZPHC	Grass-Shrub	US-AZ	...	NaN	NaN
3	130.0	5/8/2020	5/26/2020	44.300517	Human	35.875820	-115.204100	NVLIC	Grass-Shrub	US-NV	...	11.12	0.00
4	133.0	8/21/2020	8/22/2020	4.000000	Human	44.035131	-103.036037	SDGPC	Grass	US-SD	...	59.35	0.07

5 rows × 27 columns

Exploratory Analysis (Section 2)

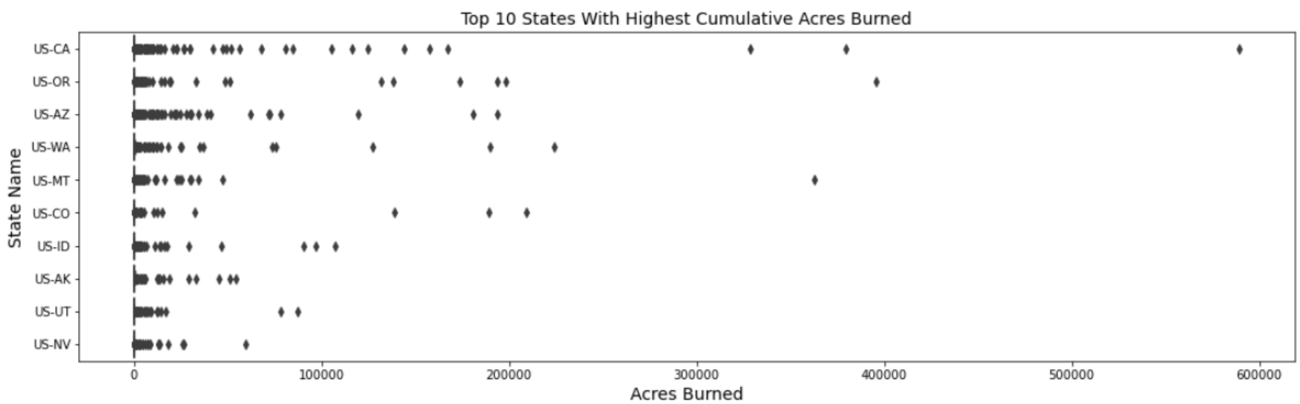
```
plt.figure(figsize=(18, 5))

State_Rank_Fire_Freq =  pd.DataFrame(df_Fire['State'].value_counts(ascending = False))
State_Rank_Fire_Freq.index.name = 'State'
State_Rank_Fire_Freq.columns = ['Fire Occurance']
display(State_Rank_Fire_Freq.head())

df_Fire['State'].value_counts().plot(kind='bar')

plt.xlabel('State Name', fontsize=18)
plt.ylabel('Fire Occurance', fontsize=18)
plt.title('State with Highest Number of Fire Occurance', fontsize=18)
```

	Fire Occurance
State	
US-MT	645
US-AZ	633
US-MN	600
US-CA	592
US-ID	503



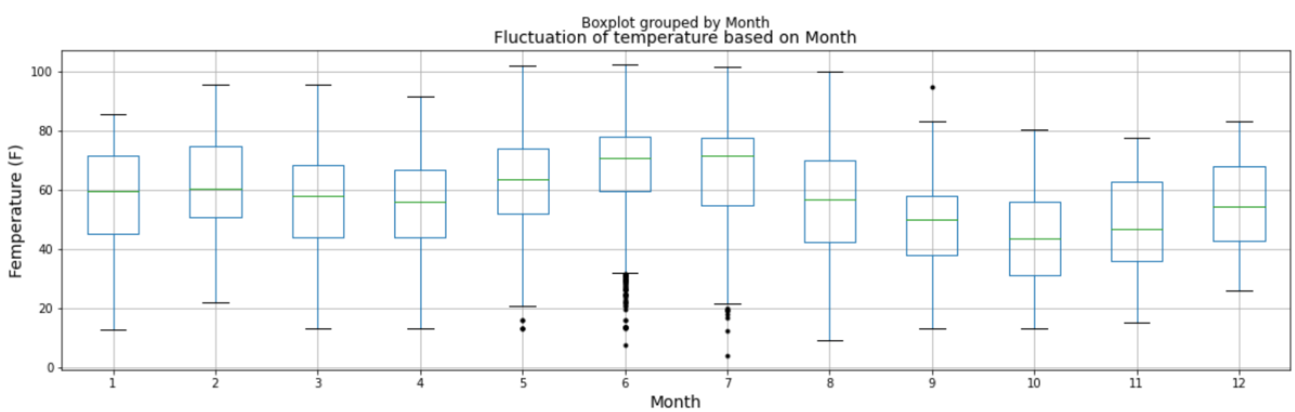
Exploratory Analysis (Section 3)

```
State_AcresBurned = df_Fire.groupby('State')['Acres'].sum()
State_AcresBurned = State_AcresBurned.sort_values(ascending = False)
Top10_State_AcresBurned = State_AcresBurned[:10]

plt.figure(figsize=(18, 5))
sns.boxplot(x='Acres', y='State', data=df_Fire, order=Top10_State_AcresBurned.index);
plt.xlabel('Acres Burned',fontsize = 14)
plt.ylabel('State Name',fontsize = 14)
plt.title("Top 10 States With Highest Cumulative Acres Burned",fontsize = 14)
plt.show()
```



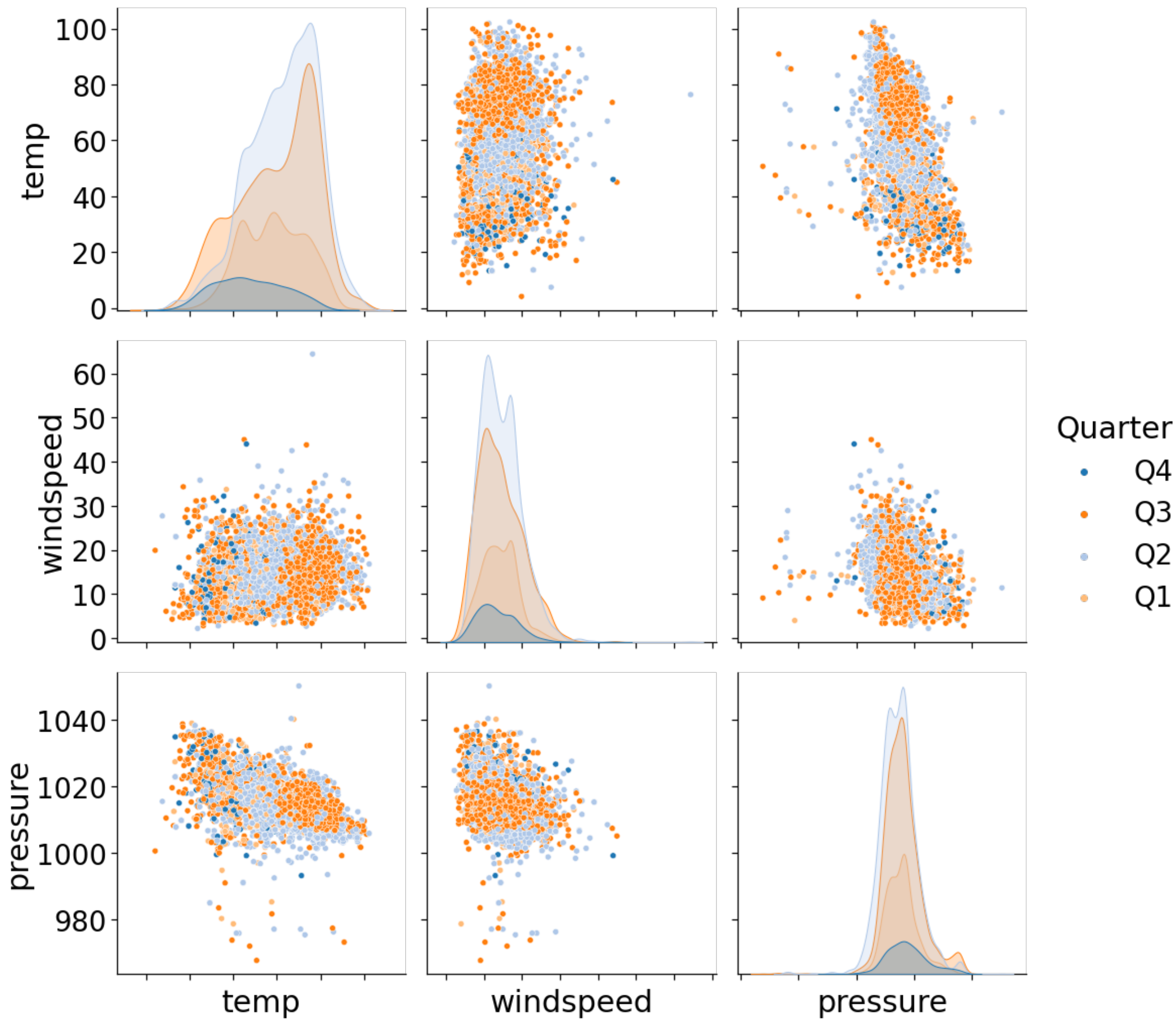
Exploratory Analysis (Section 4)

```
df_Fire.boxplot(column='temp', by = 'Month', sym = 'k.', figsize = (18,5))
plt.xlabel('Month',fontsize = 14)
plt.ylabel('Temperature (F)',fontsize = 14)
plt.title('Fluctuation of temperature based on Month',fontsize = 14)
```



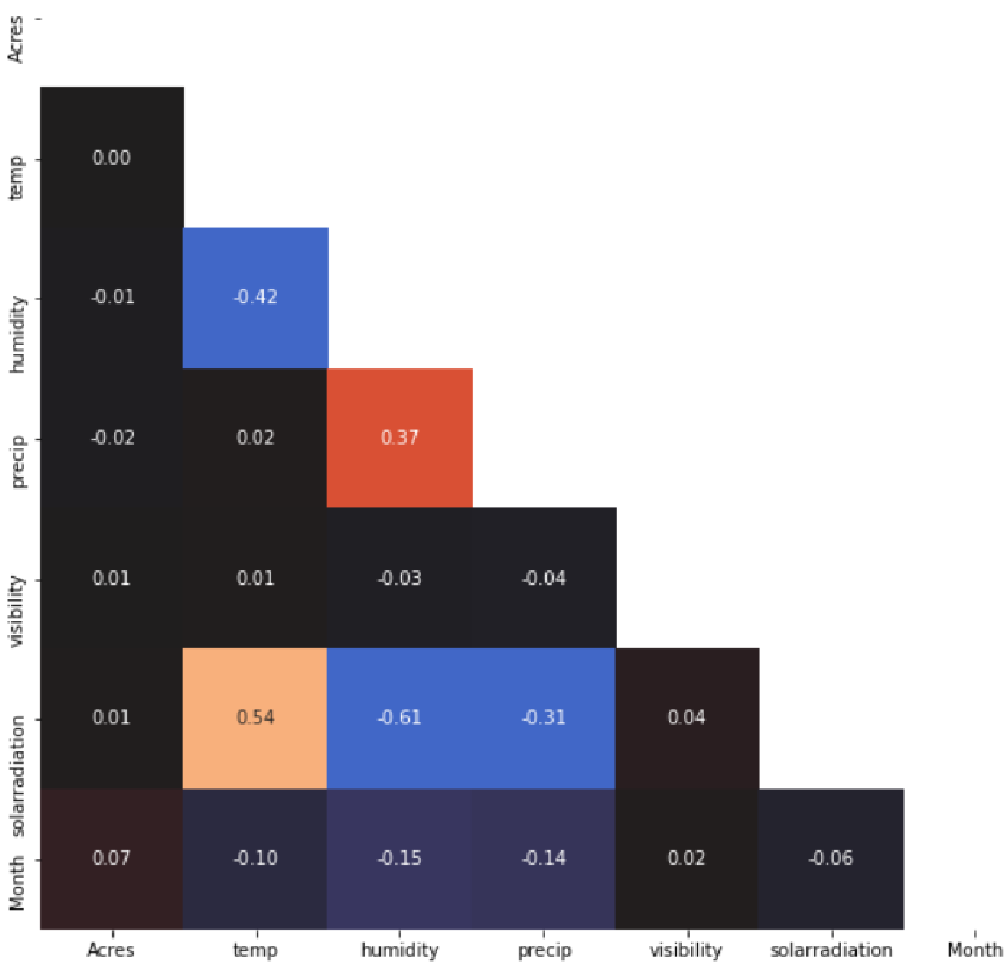
Exploratory Analysis (Section 5)

```
import datetime as dt

df_sub = df_Fire[['Quarter','temp', 'windspeed','pressure']]

with sns.plotting_context('notebook', font_scale = 2.5):
    g = sns.pairplot(df_sub, hue = 'Quarter', palette = 'tab20', height = 5)

g.set(xticklabels = [])
```



Exploratory Analysis (Section 6)

```
ColsToConsider = ['Acres', 'temp','humidity', 'precip', 'visibility', 'solarradiation', 'Month']
df_c = df_Fire.loc[:,ColsToConsider]

fig, ax = plt.subplots(figsize=(10,10))
sns_plot   = sns.heatmap(
    data   = df_c.corr(),
    vmin   = -0.3,
    vmax   = 0.6,
    center = 0,
    annot  = True,
    fmt    = '.2f',
    mask   = ~np.tri(df_c.corr().shape[1], k =-1, dtype = bool),
    cbar   = False,
    ax     = ax)
```



Modeling

Setup Training and Testing dataset (Section 1.A)

```
#Import required libraries
import numpy as np, pandas as pd, matplotlib as mpl, matplotlib.pyplot as plt, seaborn as sns, req
uests
from IPython.display import display
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import statsmodels.tools.tools as stattools

#Load data to a dataframe for training purposes
df_All = pd.read_csv(Weather_Final_Cleaned_df)

df_FireOccured = df_All.query('FireOccured == 1')
df_Not_FireOccured = df_All.query('FireOccured == 0')

print(f'{len(df_FireOccured)/len(df_All):.0%} ({len(df_FireOccured):,} records) are events which f
ire did occure.')
print(f'{len(df_Not_FireOccured)/len(df_All):.0%} ({len(df_Not_FireOccured):,} records) are events
 which fire did NOT occure.')


display(df_FireOccured.head())
display(df_Not_FireOccured.head())
```

```
33% (7,125 records) are events which fire did occure.
67% (14,279 records) are events which fire did NOT occure.
```

	OBJECTID	Date_Start	Date_Finish	Acres	FireCause	Lat	Long	DispatchCenterID	PredominantFuelGroup	State	...	tempmin	temp
0	127.0	10/18/2020	3/10/2021	562.913504	Unknown	36.071140	-121.450500	CALPCC	Unknown	US-CA	...	49.9	58.8
1	128.0	5/1/2020	5/15/2020	0.151680	Unknown	39.556690	-119.558500	NVSFC	Grass	US-NV	...	41.2	60.6
2	129.0	8/8/2020	8/20/2020	0.300000	Human	33.293840	-110.450000	AZPHC	Grass-Shrub	US-AZ	...	NaN	NaN
3	130.0	5/8/2020	5/26/2020	44.300517	Human	35.875820	-115.204100	NVLIC	Grass-Shrub	US-NV	...	58.6	76.9
4	133.0	8/21/2020	8/22/2020	4.000000	Human	44.035131	-103.036037	SDGPC	Grass	US-SD	...	33.3	47.4

5 rows × 25 columns

	OBJECTID	Date_Start	Date_Finish	Acres	FireCause	Lat	Long	DispatchCenterID	PredominantFuelGroup	State	...	tempmin	temp	hı
7125	NaN	7/8/2020	NaN	NaN	NaN	40.633125	-89.398528	NaN	NaN	US-IL	...	72.8	81.7	
7126	NaN	6/14/2021	NaN	NaN	NaN	45.253783	-69.445469	NaN	NaN	US-ME	...	49.1	60.3	
7127	NaN	12/29/2020	NaN	NaN	NaN	42.407211	-71.382437	NaN	NaN	US-MA	...	33.7	38.2	
7128	NaN	2/29/2020	NaN	NaN	NaN	39.045755	-76.641271	NaN	NaN	US-MD	...	66.8	70.9	
7129	NaN	10/25/2020	NaN	NaN	NaN	43.969515	-99.901813	NaN	NaN	US-SD	...	48.3	51.5	

5 rows × 25 columns

Setup Training and Testing dataset (Section 1.B)

```
from sklearn.model_selection import train_test_split as split
from IPython.display import display_html
from itertools import chain,cycle
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder


def DisplayMultiply(*args,titles=cycle(['']),html_str = ''):
    for df,title in zip(args, chain(titles,cycle(['</br>'])) ):
        html_str += '<th style="text-align:center"><td style="vertical-align:top">'
        html_str += f"<h6 style='text-align:center'>{title}</h5>"
        html_str += df.to_html().replace('table','table style="display:inline"')
    display_html(html_str,raw=True)

# Slice columns that will be used for modeling
df = df_All
df.fillna(df.mean(),inplace=True)

ColumnsToConsider = ['humidity','windspeed','pressure','temp','conditions']
```

```
# Convert Catigorical condition column to scaler depending on level.
conditions_dict = {
    'Clear'                   : 1,
    'Rain, Partially cloudy' : 2,
    'Partially cloudy'        : 3,
    'Overcast'                : 4,
    'Rain'                    : 5,
    'Rain, Overcast'          : 6,
    'Rain, Fog'               : 7,
    'Snow, Partially cloudy' : 8,
    'Snow, Overcast'          : 9,
    'Snow'                    : 10
    }

df.conditions.replace(conditions_dict, inplace=True)

# Split data to Test and Train set
X_train, X_test, y_train, y_test = \
split(df[ColumnsToConsider], df.FireOccured, test_size=0.30, random_state=1)

# Display sample records for each set
Top10 = lambda Data:pd.DataFrame(Data).head(10)

DisplayMultiply(Top10(X_train), Top10(X_test),
     titles=['Predictors (Train Set)', 'Predictors (Test Set)'])
```

Predictors (Train Set)

	humidity	windspeed	pressure	temp	conditions
15656	74.00	7.8	1012.4	69.9	2
11485	62.19	6.7	1027.5	48.5	1
21108	72.17	16.0	1015.7	71.5	2
11909	73.61	8.8	1021.4	50.0	5
5913	67.54	11.9	1010.9	68.1	1
3097	39.21	23.0	1009.1	56.8	1
17017	73.59	10.3	1026.4	37.4	2
18542	60.55	9.3	1025.3	47.4	1
8870	53.07	12.5	1011.2	59.0	2
5744	61.59	6.9	1011.1	68.6	1

Predictors (Test Set)

	humidity	windspeed	pressure	temp	conditions
15709	66.25	22.1	1013.4	62.1	2
3520	62.72	5.6	1030.4	26.2	1
18276	56.79	15.3	1026.5	43.0	2
12390	94.14	12.1	1016.4	26.8	1
6530	44.22	15.9	1017.3	73.5	5
989	62.61	8.1	1021.0	53.3	2
8316	61.25	6.7	1016.8	64.3	3
12134	91.05	19.9	1012.6	45.5	2
4211	72.21	6.3	1020.3	26.9	1
5396	19.89	25.3	1006.4	67.0	1

Modeling Decision Tree - CART (Section 2)

```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.model_selection import train_test_split as tts
Model_CART = DecisionTreeClassifier(criterion="gini", max_leaf_nodes=10).fit(X_train, y_train)
export_graphviz(Model_CART, out_file="fire_cart.dot", feature_names=ColumnsToConsider,
   class_names='FireOccured')

y_pred_CART = Model_CART.predict(X_test)

cmtx_CART = pd.DataFrame(
    confusion_matrix(y_test, y_pred_CART, labels=[0,1]),
    index=['true: {:}'.format(x) for x in [0,1]],
    columns=['pred: {:}'.format(x) for x in [0,1]])

print("CART Model\n\n",classification_report(y_test, y_pred_CART))
cmtx_CART
```

```
CART Model

              precision    recall  f1-score   support

           0       0.78      0.87      0.83      4255
           1       0.68      0.53      0.59      2167

    accuracy                           0.76      6422
   macro avg       0.73      0.70      0.71      6422
weighted avg       0.75      0.76      0.75      6422
```

	pred: 0	pred: 1
true: 0	3705	550
true: 1	1020	1147

Modeling Decision Tree - C5.0 (Section 3)

```
Model_C50 = DecisionTreeClassifier(criterion="entropy", max_leaf_nodes=10).fit(X_train, y_train)
export_graphviz(Model_C50, out_file="fire_C50.dot", feature_names=ColumnsToConsider, class_names='
FireOccured')

y_pred_C50 = Model_C50.predict(X_test)

cmtx_C50 = pd.DataFrame(
    confusion_matrix(y_test, y_pred_C50, labels=[0,1]),
    index=['true: {:}'.format(x) for x in [0,1]],
    columns=['pred: {:}'.format(x) for x in [0,1]])

print("C5.0 Model\n\n",classification_report(y_test, y_pred_C50))
cmtx_C50
```

```
C5.0 Model

              precision    recall  f1-score   support

           0       0.75      0.94      0.84      4255
           1       0.77      0.39      0.52      2167

    accuracy                           0.75      6422
   macro avg       0.76      0.66      0.68      6422
weighted avg       0.76      0.75      0.73      6422
```

	pred: 0	pred: 1
true: 0	4002	253
true: 1	1327	840

Modeling Decision Tree - Random Forest (Section 4)

```
from sklearn.ensemble import RandomForestClassifier

Model_RandomForest = RandomForestClassifier(n_estimators = 100).fit(X_train, y_train)
y_pred_RandomForest = Model_RandomForest.predict(X_test)

cmtx_RF = pd.DataFrame(
    confusion_matrix(y_test, y_pred_RandomForest, labels=[0,1]),
    index=['true: {:}'.format(x) for x in [0,1]],
    columns=['pred: {:}'.format(x) for x in [0,1]])

print("Random Forest Model\n\n",classification_report(y_test, y_pred_RandomForest))
cmtx_RF
```

```
Random Forest Model

              precision    recall  f1-score   support

           0       0.85      0.93      0.89      4255
           1       0.84      0.68      0.75      2167

    accuracy                           0.85      6422
   macro avg       0.84      0.80      0.82      6422
weighted avg       0.84      0.85      0.84      6422
```

	pred: 0	pred: 1
true: 0	3966	289
true: 1	701	1466

Modeling Naïve Bayes - Gaussian (Section 5)

```
from sklearn.naive_bayes import GaussianNB

Model_GaussianNB = GaussianNB().fit(X_train, y_train)
y_pred_GaussianNB = Model_GaussianNB.predict(X_test)

cmtx_NB1 = pd.DataFrame(
    confusion_matrix(y_test, y_pred_GaussianNB, labels=[0,1]),
    index=['true: {:}'.format(x) for x in [0,1]],
    columns=['pred: {:}'.format(x) for x in [0,1]])

print("GaussianNB Model\n\n",classification_report(y_test, y_pred_GaussianNB))
cmtx_NB1
```

```
GaussianNB Model

              precision    recall  f1-score   support

           0       0.74      0.85      0.79      4255
           1       0.59      0.41      0.48      2167

    accuracy                           0.70      6422
   macro avg       0.66      0.63      0.64      6422
weighted avg       0.69      0.70      0.69      6422
```

	pred: 0	pred: 1
true: 0	3637	618
true: 1	1285	882

Modeling Naïve Bayes - Multinomial (Section 6)

```
from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import statsmodels.tools.tools as stattools
from sklearn.preprocessing import MinMaxScaler

X_train_scaler = MinMaxScaler().fit_transform(X_train)
X_test_scaler = MinMaxScaler().fit_transform(X_test)

Model_MultinomialNB = MultinomialNB().fit(X_train_scaler, y_train)
y_pred_MultinomialNB = Model_MultinomialNB.predict(X_test_scaler)

cmtx_NB2 = pd.DataFrame(
    confusion_matrix(y_test, y_pred_MultinomialNB, labels=[0,1]),
    index=['true: {:}'.format(x) for x in [0,1]],
    columns=['pred: {:}'.format(x) for x in [0,1]])

print("MultinomialNB Model\n\n",classification_report(y_test, y_pred_MultinomialNB))
cmtx_NB2
```

```
MultinomialNB Model

              precision    recall  f1-score   support

           0       0.66      1.00      0.80      4255
           1       0.00      0.00      0.00      2167

    accuracy                           0.66      6422
   macro avg       0.33      0.50      0.40      6422
weighted avg       0.44      0.66      0.53      6422
```

	pred: 0	pred: 1
true: 0	4255	0
true: 1	2167	0

Evaluation

```
Models_List = ['Decision Tree – CART','Decision Tree – C5.0','Decision Tree – Random Forest ','Gau
ssian Naïve Bayes','MultinomialNB Naïve Bayes']

EvalTable = pd.DataFrame(columns = ['Accuracy','Sensitivity','Specificity','F1Measure'], index=[Mo
dels_List])

cmtx_dict = {
   'Decision Tree – CART': cmtx_CART,
   'Decision Tree – C5.0': cmtx_C50,
   'Decision Tree – Random Forest ': cmtx_RF,
   'Gaussian Naïve Bayes': cmtx_NB1,
   'MultinomialNB Naïve Bayes': cmtx_NB2
            }
```

```
# Calculating Accuracy
for TableName, TableVal in cmtx_dict.items():
    EvalTable.Accuracy[TableName] = \
    (TableVal.iloc[0,0]+TableVal.iloc[1,1]) / sum(sum(TableVal.values))

# Calculating Sensitivity
for TableName, TableVal in cmtx_dict.items():
    EvalTable.Sensitivity[TableName] = TableVal.iloc[0,0]/(TableVal.iloc[0,0]+TableVal.iloc[0,1])

# Calculating Specificity
for TableName, TableVal in cmtx_dict.items():
    EvalTable.Specificity[TableName] = TableVal.iloc[1,1]/(TableVal.iloc[1,0]+TableVal.iloc[1,1])

# Calculating F1 Measure
for TableName, TableVal in cmtx_dict.items():
    EvalTable.F1Measure[TableName] = \
         2 * ((EvalTable.Specificity[TableName] * EvalTable.Sensitivity[TableName]) /
              (EvalTable.Specificity[TableName] + EvalTable.Sensitivity[TableName]))


pd.options.display.float_format = '{:.3f}'.format
Bold = ['\033[1m','\033[0m']
print('──────────────────────────────────────────────────────────────────────────────')
print(f'{Bold[0]}                   Decision Tree Models — Confusion Matrix {Bold[1]}')
print('──────────────────────────────────────────────────────────────────────────────')
DisplayMultiply(cmtx_CART,cmtx_C50,cmtx_RF, titles = Models_List[:3])

print('\n\n')
print('──────────────────────────────────────────────────')
print(f'{Bold[0]}           Naïve Bayes — Confusion Matrix {Bold[1]}')
print('──────────────────────────────────────────────────')
DisplayMultiply(cmtx_NB1,cmtx_NB2, titles = Models_List[3:])

print('\n\n')
print('───────────────────────────────────────────────────────────────')
print(f'{Bold[0]}                           Evaluation Table {Bold[1]}')
print('───────────────────────────────────────────────────────────────')

EvalTable
```

Decision Tree Models — Confusion Matrix

Decision Tree — CART

	pred: 0	pred: 1
true: 0	3705	550
true: 1	1020	1147

Decision Tree — C5.0

	pred: 0	pred: 1
true: 0	4002	253
true: 1	1327	840

Decision Tree — Random Forest

	pred: 0	pred: 1
true: 0	3966	289
true: 1	701	1466

Naïve Bayes — Confusion Matrix

Gaussian Naïve Bayes

	pred: 0	pred: 1
true: 0	3637	618
true: 1	1285	882

MultinomialNB Naïve Bayes

	pred: 0	pred: 1
true: 0	4255	0
true: 1	2167	0

Evaluation Table

	Accuracy	Sensitivity	Specificity	F1Measure
Decision Tree — CART	0.756	0.871	0.529	0.658
Decision Tree — C5.0	0.754	0.941	0.388	0.549
Decision Tree — Random Forest	0.846	0.932	0.677	0.784
Gaussian Naïve Bayes	0.704	0.855	0.407	0.551
MultinomialNB Naïve Bayes	0.663	1.000	0.000	0.000