Created By Pooja Patel

# WildFire Analysis : Predict the Cause of Wildfire in USA

**Introduction** : Over the past several decades, unexpected wildfires have caused serious damage to the environment and economy in the United states. The U.S. faces a wildfire crisis that costs the federal government $2.5 billion a year. Understanding the causes of wildfires is essential for effective wildfire prevention.

**Goal** : This project aims to analyze historical wildfire data from U.S. Department of Agriculture and develop predictive models to identify the primary causes of wildfires in the USA.

**Dataset URL** : https://doi.org/10.2737/RDS-2013-0009.6

The data is provided by a spatial database of historical wildfires in the United States from 1992 to 2020. The dataset is generated to support the Fire Program Analysis (FPA) system. There are over 2.3 million wildfire records in this dataset.

**Source:**

Short, Karen C. 2022. Spatial wildfire occurrence data for the United States, 1992-2020 [FPA_FOD_20221014]. 6th Edition. Fort Collins, CO: Forest Service Research Data Archive.

## Features and its descriptions

Meta-data:

FOD_ID Unique numeric record identifier.

FPA_ID Unique identifier that contains information necessary to track back to the original record in the source dataset.

SOURCE_SYSTEM_TYPE Type of source database or system that the record was drawn from (FED = federal, NONFED = nonfederal, or INTERAGCY = interagency).

SOURCE_SYSTEM Name of or other identifier for source database or system that the record was drawn from. See \Supplements\FPA_FOD_source_list.pdf for a list of sources and their identifier and Short (2014) for additional source information.

NWCG_REPORTING_AGENCY Active National Wildlife Coordinating Group (NWCG) Unit Identifier for the agency preparing the fire report (BIA = Bureau of Indian Affairs, BLM = Bureau of Land Management, BOR = Bureau of Reclamation, DOD = Department of Defense, DOE = Department of Energy, FS = Forest Service, FWS = Fish and Wildlife Service, IA = Interagency Organization, NPS = National Park Service, ST/C&L = State, County, or Local Organization, and TRIBE = Tribal Organization).

NWCG_REPORTING_UNIT_ID Active NWCG Unit Identifier for the unit preparing the fire report.

NWCG_REPORTING_UNIT_NAME Active NWCG Unit Name for the unit preparing the fire report.

SOURCE_REPORTING_UNIT Code for the agency unit preparing the fire report, based on code/name in the source dataset.

SOURCE_REPORTING_UNIT_NAME Name of reporting agency unit preparing the fire report, based on code/name in the source dataset.

LOCAL_FIRE_REPORT_ID Number or code that uniquely identifies an incident report for a particular reporting unit and a particular calendar year.

LOCAL_INCIDENT_ID Number or code that uniquely identifies an incident for a particular local fire management organization within a particular calendar year.

FIRE_CODE Code used within the interagency wildland fire community to track and compile cost information for emergency fire suppression (https://www.firecode.gov/).

FIRE_NAME Name of the incident, from the fire report (primary) or ICS-209 report (secondary).

ICS_209_PLUS_INCIDENT_JOIN_ID Primary identifier needed to join into operational situation reporting data for the incident in the ICS-209-PLUS dataset.

ICS_209_PLUS_COMPLEX_JOIN_ID If part of a complex, secondary identifier potentially needed to join to operational situation reporting data for the incident in the ICS-209-PLUS dataset.

MTBS_ID Incident identifier, from the MTBS perimeter dataset.

MTBS_FIRE_NAME Name of the incident, from the MTBS perimeter dataset.

COMPLEX_NAME Name of the complex under which the fire was ultimately managed, when discernible.

FIRE_YEAR Calendar year in which the fire was discovered or confirmed to exist.

DISCOVERY_DATE Date on which the fire was discovered or confirmed to exist.

DISCOVERY_DOY Day of year on which the fire was discovered or confirmed to exist.

DISCOVERY_TIME Time of day that the fire was discovered or confirmed to exist.

NWCG_CAUSE_CLASSIFICATION Broad classification of the reason the fire occurred (Human, Natural, Missing data/not specified/undetermined).

NWCG_GENERAL_CAUSE Event or circumstance that started a fire or set the stage for its occurrence (Arson/incendiarism, Debris and open burning, Equipment and vehicle use, Firearms and explosives use, Fireworks, Misuse of fire by a minor, Natural, Power generation/transmission/distribution, Railroad operations and maintenance, Recreation and ceremony, Smoking, Other causes, Missing data/not specified/undetermined).

NWCG_CAUSE_AGE_CATEGORY If cause attributed to children (ages 0-12) or adolescents (13-17), the value for this data element is set to Minor; otherwise null.

CONT_DATE Date on which the fire was declared contained or otherwise controlled (mm/dd/yyyy where mm=month, dd=day, and yyyy=year).

CONT_DOY Day of year on which the fire was declared contained or otherwise controlled.

CONT_TIME Time of day that the fire was declared contained or otherwise controlled (hhmm where hh=hour, mm=minutes).

FIRE_SIZE The estimate of acres within the final perimeter of the fire.

FIRE_SIZE_CLASS Code for fire size based on the number of acres within the final fire perimeter (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).

LATITUDE Latitude (NAD83) for point location of the fire (decimal degrees).

LONGITUDE Longitude (NAD83) for point location of the fire (decimal degrees).

OWNER_DESCR Name of primary owner or entity responsible for managing the land at the point of origin of the fire at the time of the incident.

STATE Two-letter alphabetic code for the state in which the fire burned (or originated), based on the nominal designation in the fire report (not from a spatial overlay).

COUNTY County, or equivalent, in which the fire burned (or originated), based on nominal designation in the fire report (not from a spatial overlay).

FIPS_CODE Five-digit code from the Federal Information Process Standards (FIPS) publication 6-4 for representation of counties and equivalent entities, based on the nominal designation in the fire report (not from a spatial overlay).

FIPS_NAME County name from the FIPS publication 6-4 for representation of counties and equivalent entities, based on the nominal designation in the fire report (not from a spatial overlay).

## PART 1

```
# imporing necessary Libraries
import matplotlib.pyplot as plt
import pandas as pd
```

```
import csv
df = pd.read_csv("/content/sample_data/data.csv", quoting=3, error_bad_lines=False)
# df.head()
```

```
# 39 features
df.columns
```

```
    Index(['OBJECTID', 'Shape', 'FOD_ID', 'FPA_ID', 'SOURCE_SYSTEM_TYPE',
           'SOURCE_SYSTEM', 'NWCG_REPORTING_AGENCY', 'NWCG_REPORTING_UNIT_ID',
           'NWCG_REPORTING_UNIT_NAME', 'SOURCE_REPORTING_UNIT',
           'SOURCE_REPORTING_UNIT_NAME', 'LOCAL_FIRE_REPORT_ID',
           'LOCAL_INCIDENT_ID', 'FIRE_CODE', 'FIRE_NAME',
           'ICS_209_PLUS_INCIDENT_JOIN_ID', 'ICS_209_PLUS_COMPLEX_JOIN_ID',
           'MTBS_ID', 'MTBS_FIRE_NAME', 'COMPLEX_NAME', 'FIRE_YEAR',
           'DISCOVERY_DATE', 'DISCOVERY_DOY', 'DISCOVERY_TIME',
           'NWCG_CAUSE_CLASSIFICATION', 'NWCG_GENERAL_CAUSE',
           'NWCG_CAUSE_AGE_CATEGORY', 'CONT_DATE', 'CONT_DOY', 'CONT_TIME',
           'FIRE_SIZE', 'FIRE_SIZE_CLASS', 'LATITUDE', 'LONGITUDE', 'OWNER_DESCR',
           'STATE', 'COUNTY', 'FIPS_CODE', 'FIPS_NAME'],
          dtype='object')
```

```
df['MTBS_FIRE_NAME'].unique()
```

```
    array([nan, 'POWER', 'FREDS', ..., 'MILL', 'CANYON 2', 'NARROWS'],
          dtype=object)
```

```
df['COMPLEX_NAME'].unique()
```

```
           'KINGSLEY COMPLEX', 'HAPPY CAMP COMPLEX', 'PAYETTE WFU COMPLEX',
           'BAR COMPLEX', 'TRIPOD COMPLEX', 'TWIN ELK COMPLEX',
           'TATOOSH COMPLEX', 'HERMAN COMPLEX', 'REVETT COMPLEX',
           'RIVAL NANNY COMPLEX', 'CUDDY COMPLEX', 'CARBON COPY COMPLEX',
           'CHEROKEE SOUTH COMPLEX\xa0', 'BRODER/BECK COMPLEX',
           'SHAKE TABLE COMPLEX', 'SAWTOOTH-MILLARD-HEART COMPLEX\xa0',
           'PEAVINE COMPLEX\xa0', 'SIZEROCK COMPLEX', 'ANTELOPE\xa0COMPLEX',
           'SHU LIGHTNING COMPLEX 2008', 'SNOW CREEK COMPLEX',
           'SUMMIT SPRINGS COMPLEX', 'TGU LIGHTNING COMPLEX',
           'LARCH MOUNTAIN COMPLEX', 'FREEWAY COMPLEX', 'GRAND COMPLEX',
           'FRIENDSHIP COMPLEX\xa0', 'FC EAST COMPLEX', 'CEDAR BENCH COMPLEX',
           'COUGAR RIDGE COMPLEX', 'UPPER COMPLEX 2009', 'TILLER COMPLEX',
           'BACKBONE COMPLEX', 'RUBY COMPLEX', 'SUSIE COMPLEX',
           'HAT CREEK COMPLEX', 'CACTUS COMPLEX\xa0',
           'SHU LIGHTNING COMPLEX 2009', 'ELK SUMMIT COMPLEX',
           'GOLD HILL COMPLEX', 'LION COMPLEX', 'TUMBLEBUG COMPLEX',
           'CHEVLON COMPLEX', 'WILDHORSE COMPLEX', 'BOZE COMPLEX',
           'JULY 4TH COMPLEX', 'CURTIS ROAD COMPLEX', 'BITTER-NEZ COMPLEX',
           'PRIEST LAKE COMPLEX', 'POWELL COMPLEX', 'KOOCANUSA COMPLEX',
           'YAAK-RED DRAGON COMPLEX', 'CABINET COMPLEX', 'LIBBY COMPLEX',
           'IDAHO CITY COMPLEX', 'TIN CUP COMPLEX', 'PIONEER COMPLEX',
           'CORRAL CREEK-BLACKWELL COMPLEX', 'PORPHYRY SOUTH COMPLEX',
           'LUCAS COMPLEX', 'LITTLE MALHEUR COMPLEX', 'METHOW COMPLEX',
           'THOMASON COMPLEX', 'SNAKE RIVER COMPLEX', 'HATCHERY COMPLEX',
           'BUNNY COMPLEX', 'SUGAR COMPLEX', 'CAMBELL COMPLEX',
           'AMBROSE COMPLEX', 'NEPHI "J" COMPLEX', 'SUN COMPLEX',
           'ROCKY HOLLOW COMPLEX', 'VIVIAN COMPLEX', 'SULA PEAK COMPLEX',
           'MAGOTSU COMPLEX', 'GLOBE MOUNTAIN COMPLEX', 'ABAJO COMPLEX',
           'SWET-WARRIOR COMPLEX', 'WILLOW CREEK COMPLEX', 'BULL COMPLEX',
           'BRIDGER COMPLEX', 'MOOLACK COMPLEX', 'WILDCAT COMPLEX',
           'SALT COMPLEX', 'ACKERSON COMPLEX', 'SORENSON COMPLEX',
           'BICHOTA COMPLEX', 'LEAMINGTON COMPLEX', 'ADELAIDE COMPLEX',
           'LINK COMPLEX', 'GREEN COMPLEX'], dtype=object)
```

```
df['STATE'].unique()
```

```
array(['CA', 'NM', 'OR', 'NC', 'WY', 'CO', 'WA', 'MT', 'UT', 'AZ', 'SD',
       'AR', 'NV', 'ID', 'MN', 'TX', 'FL', 'SC', 'LA', 'OK', 'KS', 'MO',
       'NE', 'MI', 'KY', 'OH', 'IN', 'VA', 'IL', 'TN', 'GA', 'AK', 'ND',
       'WV', 'WI', 'AL', 'NH', 'PA', 'MS', 'ME', 'VT', 'NY', nan],
      dtype=object)
```

```
df['FIRE_SIZE_CLASS'].unique()
```

```
array(['A', 'B', 'G', 'C', 'D', 'F', 'E', nan], dtype=object)
```

```
df['OWNER_DESCR'].unique()
```

```
array(['USFS', 'STATE OR PRIVATE', 'MISSING/NOT SPECIFIED',
       'OTHER FEDERAL', nan], dtype=object)
```

## Data Pre-processing & Transformation

```
df[['FIRE_SIZE', 'FIRE_SIZE_CLASS', 'LATITUDE', 'LONGITUDE', 'OWNER_DESCR',
    'STATE', 'COUNTY', 'FIPS_CODE', 'FIPS_NAME']][:10]
```

| | FIRE_SIZE | FIRE_SIZE_CLASS | LATITUDE | LONGITUDE | OWNER_DESCR | STATE | COUNTY | FIPS_CODE | FIPS_NAME |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.10 | A | 40.036944 | -121.005833 | USFS | CA | 63.0 | 6063.0 | Plumas County |
| 1 | 0.25 | A | 38.933056 | -120.404444 | USFS | CA | 61.0 | 6061.0 | Placer County |
| 2 | 0.10 | A | 38.984167 | -120.735556 | STATE OR PRIVATE | CA | 17.0 | 6017.0 | El Dorado County |
| 3 | 0.10 | A | 38.559167 | -119.913333 | USFS | CA | 3.0 | 6003.0 | Alpine County |
| 4 | 0.10 | A | 38.559167 | -119.933056 | USFS | CA | 3.0 | 6003.0 | Alpine County |
| 5 | 0.10 | A | 38.635278 | -120.103611 | USFS | CA | 5.0 | 6005.0 | Amador County |
| 6 | 0.10 | A | 38.688333 | -120.153333 | USFS | CA | 17.0 | 6017.0 | El Dorado County |
| 7 | 0.80 | B | 40.968056 | -122.433889 | STATE OR PRIVATE | CA | NaN | NaN | NaN |
| 8 | 1.00 | B | 41.233611 | -122.283333 | STATE OR PRIVATE | CA | NaN | NaN | NaN |
| 9 | 0.10 | A | 38.548333 | -120.149167 | USFS | CA | 5.0 | 6005.0 | Amador County |

```
# removing unnecessary columns

cols_to_drop = ['OBJECTID',
                'FOD_ID',
                'LOCAL_FIRE_REPORT_ID',
                'LOCAL_INCIDENT_ID',
                'ICS_209_PLUS_INCIDENT_JOIN_ID',
                'ICS_209_PLUS_COMPLEX_JOIN_ID',
                'MTBS_ID',
                'MTBS_FIRE_NAME',
                'COUNTY',
                'FIRE_CODE',
                'COMPLEX_NAME',
                'CONT_DATE',
                'CONT_TIME',
                'Shape',
                'FPA_ID',
                'SOURCE_REPORTING_UNIT',
                'SOURCE_SYSTEM',
                'FIRE_NAME',
                'CONT_DOY',
                'DISCOVERY_DOY',
                'DISCOVERY_TIME',
                'SOURCE_SYSTEM_TYPE',
                'NWCG_REPORTING_AGENCY',
                'NWCG_REPORTING_UNIT_ID',
                'NWCG_REPORTING_UNIT_NAME',
                'SOURCE_REPORTING_UNIT_NAME',
                'OWNER_DESCR'
               ]

df = df.drop(columns = cols_to_drop, axis=1)
```

```
# saving necessary columns dataset into csv file
df.to_csv('fire1data.csv')
```

## PART 2

```
import pandas as pd
import numpy as np

# Graphing Libraries
import seaborn as sns
import matplotlib.pyplot as plt
import plotly
import plotly.graph_objects as go
import plotly.express as px
```

```
from google.colab import drive
drive.mount('\content\drive')
```

```
# /content/contentdrive/MyDrive/Datasets/fire1data.csv
```

```
# loading the selected columns dataset
wildfire_df = pd.read_csv('/content/contentdrive/MyDrive/Datasets/fire1data.csv')
```

```
wildfire_df
```

| | Unnamed: 0 | FIRE_YEAR | DISCOVERY_DATE | NWCG_CAUSE_CLASSIFICATION | NWCG_GENERAL_CAUSE | NWCG_CAUSE_AG |
|---|---|---|---|---|---|---|
| **0** | 0 | 2005 | 2/2/2005 | Human | Power generation/transmission/distribution | |
| **1** | 1 | 2004 | 5/12/2004 | Natural | Natural | |
| **2** | 2 | 2004 | 5/31/2004 | Human | Debris and open burning | |
| **3** | 3 | 2004 | 6/28/2004 | Natural | Natural | |
| **4** | 4 | 2004 | 6/28/2004 | Natural | Natural | |
| **...** | ... | ... | ... | ... | ... | |
| **2303561** | 2303561 | 2020 | 6/5/2020 | Natural | Natural | |
| **2303562** | 2303562 | 2020 | 7/11/2020 | Missing data/not specified/undetermined | Missing data/not specified/undetermined | |
| **2303563** | 2303563 | 2020 | 8/27/2020 | Natural | Natural | |
| **2303564** | 2303564 | 2020 | 8/17/2020 | Natural | Natural | |
| **2303565** | 2303565 | 2020 | 11/20/2020 | Human | Missing data/not specified/undetermined | |

2303566 rows × 13 columns

```
# droping the unwanted column
wildfire_df.drop(['Unnamed: 0','NWCG_CAUSE_AGE_CATEGORY','FIPS_CODE', 'FIPS_NAME'],axis=1, inplace=True)
```

```
# shape of data
wildfire_df.shape
```

```
(2303566, 9)
```

```
# checking datatype of features
wildfire_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2303566 entries, 0 to 2303565
Data columns (total 9 columns):
 #   Column                     Dtype
---  ------                     -----
 0   FIRE_YEAR                  int64
 1   DISCOVERY_DATE             object
 2   NWCG_CAUSE_CLASSIFICATION  object
 3   NWCG_GENERAL_CAUSE         object
 4   FIRE_SIZE                  float64
 5   FIRE_SIZE_CLASS            object
 6   LATITUDE                   float64
 7   LONGITUDE                  float64
 8   STATE                      object
dtypes: float64(3), int64(1), object(5)
memory usage: 158.2+ MB
```

```
# checking missing values
wildfire_df.isnull().sum()
```

```
FIRE_YEAR                    0
DISCOVERY_DATE               0
NWCG_CAUSE_CLASSIFICATION    0
NWCG_GENERAL_CAUSE           0
FIRE_SIZE                    0
FIRE_SIZE_CLASS              0
LATITUDE                     0
LONGITUDE                    0
STATE                        0
dtype: int64
```

```
wildfire_df['FIRE_YEAR'].unique()
```

```
array([2005, 2004, 2006, 2008, 2002, 2007, 2009, 2001, 2003, 1992, 1993,
       1994, 1995, 1996, 1997, 1998, 1999, 2000, 2010, 2011, 2012, 2013,
       2014, 2015, 2016, 2017, 2018, 2019, 2020])
```

## Data Wrangling

### Data type converions to reduce memory usage

```
# Converting data type to reduce memory usage:
convert_feature_datatype_dict = {'FIRE_YEAR':'int16',
                'FIRE_SIZE':'float',
                'FIRE_SIZE_CLASS':'category',
                'STATE':'category',
                'NWCG_GENERAL_CAUSE':'category',
                'NWCG_CAUSE_CLASSIFICATION':'category'
               }

# Convert data types
wildfire_df = wildfire_df.astype(convert_feature_datatype_dict)
```

### Trim the blank spaces

```
wildfire_df['NWCG_GENERAL_CAUSE'].unique()
```

```
['Power generation/transmission/distribution', 'Natural', 'Debris and open burning', 'Missing data/not
specified/undetermined', 'Recreation and ceremony', ..., 'Other causes', 'Railroad operations and maintenance', 'Smoking',
'Misuse of fire by a minor', 'Firearms and explosives use']
Length: 13
Categories (13, object): ['Arson/incendiarism', 'Debris and open burning', 'Equipment and vehicle use',
                          'Firearms and explosives use', ..., 'Power generation/transmission/distribution',
                          'Railroad operations and maintenance', 'Recreation and ceremony', 'Smoking']
```

```
# trim the blank spaces
wildfire_df['NWCG_GENERAL_CAUSE'] = wildfire_df['NWCG_GENERAL_CAUSE'].str.strip()
```

### Converting NWCG_GENERAL_CAUSE categories to new categories for better reading

```
# retrive unique categories of NWCG_GENERAL_CAUSE:
all_cats = wildfire_df['NWCG_GENERAL_CAUSE'].unique()
all_cats
```

```
array(['Power generation/transmission/distribution', 'Natural',
       'Debris and open burning',
       'Missing data/not specified/undetermined',
       'Recreation and ceremony', 'Equipment and vehicle use',
       'Arson/incendiarism', 'Fireworks', 'Other causes',
       'Railroad operations and maintenance', 'Smoking',
       'Misuse of fire by a minor', 'Firearms and explosives use'],
      dtype=object)
```

```
# converting exisiting long categories into short and simple category:
cat_map = {'Missing data/not specified/undetermined':'Missing/Unknown',
           'Debris and open burning':'Debris/Open Burning',
           'Natural':'Natural',
           'Arson/incendiarism':'Arson',
           'Equipment and vehicle use':'Equipment/Vehicle Use',
           'Recreation and ceremony':'Recreation/Ceremony',
           'Misuse of fire by a minor':'Fire by Minor',
           'Smoking':'Smoking',
           'Railroad operations and maintenance':'Railroad Operations',
           'Power generation/transmission/distribution':'Electrical Utility',
           'Fireworks':'Fireworks',
           'Other causes':'Other Causes',
           'Firearms and explosives use':'Firearms & Explosives'
          }
```

### Sanity Check before applying mapping

```
# Check if any categories are missing from the cat_map dictionary and raise an error if any are missing:
missing_cats = set(all_cats) - set(cat_map.keys())
if missing_cats:
    raise ValueError("Total no of missing categories{}".format(missing_cats))

# creating new mapping dict of all cats
new_cat_map = {cat: cat_map[cat] for cat in all_cats}

# total df records and unique category counts before mapping:
starting_rows = wildfire_df.shape[0]
starting_cats = wildfire_df['NWCG_GENERAL_CAUSE'].nunique()
print("Total df records before mapping: {}".format(starting_rows))
print("Total unique categories before mapping: {}".format(starting_cats))
```

```
Total df records before mapping: 2303566
Total unique categories before mapping: 13
```

### Sanity Check after applying mapping

```
# maping category of NWCG_GENERAL_CAUSE column:
wildfire_df['NWCG_GENERAL_CAUSE'] = wildfire_df['NWCG_GENERAL_CAUSE'].map(new_cat_map)

# total df records and unique category counts before mapping:
ending_rows = wildfire_df.shape[0]
ending_cats = wildfire_df['NWCG_GENERAL_CAUSE'].nunique()
print("Total df records after mapping: {}".format(ending_rows))
print("Total unique categories after mapping: {}".format(ending_cats))
```

```
Total df records after mapping: 2303566
Total unique categories after mapping: 13
```

```
wildfire_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2303566 entries, 0 to 2303565
Data columns (total 9 columns):
 #   Column                     Dtype
---  ------                     -----
 0   FIRE_YEAR                  int16
 1   DISCOVERY_DATE             object
 2   NWCG_CAUSE_CLASSIFICATION  category
 3   NWCG_GENERAL_CAUSE         object
 4   FIRE_SIZE                  float64
 5   FIRE_SIZE_CLASS            category
 6   LATITUDE                   float64
 7   LONGITUDE                  float64
 8   STATE                      category
dtypes: category(3), float64(3), int16(1), object(2)
memory usage: 98.9+ MB
```

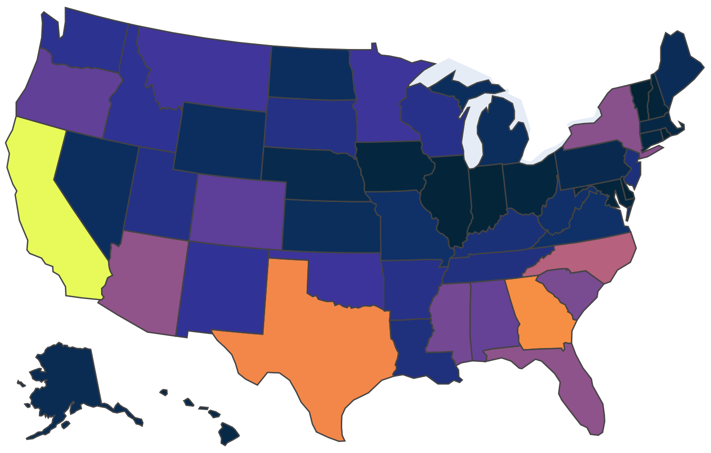
## EDA

### Wildfire Locations

```
state_fires = wildfire_df['STATE'].value_counts().rename_axis('STATES').reset_index(name='counts')
print(state_fires.head(5))
fig = go.Figure(data = go.Choropleth(locations = state_fires['STATES'], # coordinates
                    z = state_fires['counts'],
                    locationmode = 'USA-states', # location which matches USA states
                    colorscale = 'thermal',
                    colorbar_title = "Fire Intensity"
                                    ))
fig.update_layout(
    title_text = '1992-2020 United States Fires',
    geo = dict(
        scope = 'usa',
        showlakes=False
    )
)
fig.show()
```

```
  STATES  counts
0     CA  251881
1     GA  185040
2     TX  180087
3     NC  130165
4     AZ  104956
```

1992-2020 United States Fires

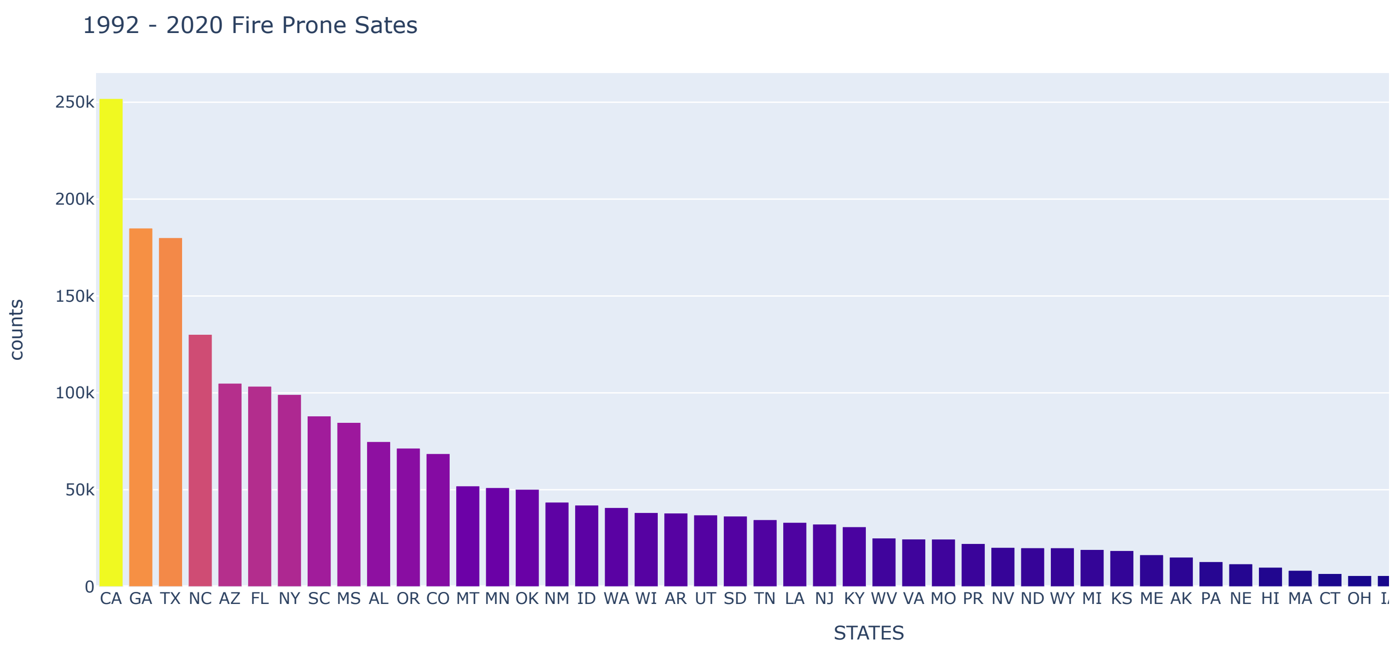


Results : CA, GA and TX states have highest fire intensity in USA.

California (CA) has 251.88k Fire Intensity which is highest among all the states of USA. After California, Georgia (GA) and Texas (TX) also have elevated fire intensities.

## Fire Prone States :

```
fig = px.bar(state_fires, x = 'STATES', y = 'counts', color = 'counts')
fig.update_layout(
                  title_text = '1992 - 2020 Fire Prone Sates',
                  coloraxis_colorbar_title_text = 'Fire Intensity')
fig.show()
```



Results : Looking at the top 10 most fire prone states of the USA, 6/10 are from southeastern states. And most Fire Prone states are the District of Columbia (DC), Delaware (DE) and Vermont (VT).

## Cause of Fire : 1992 to 2020

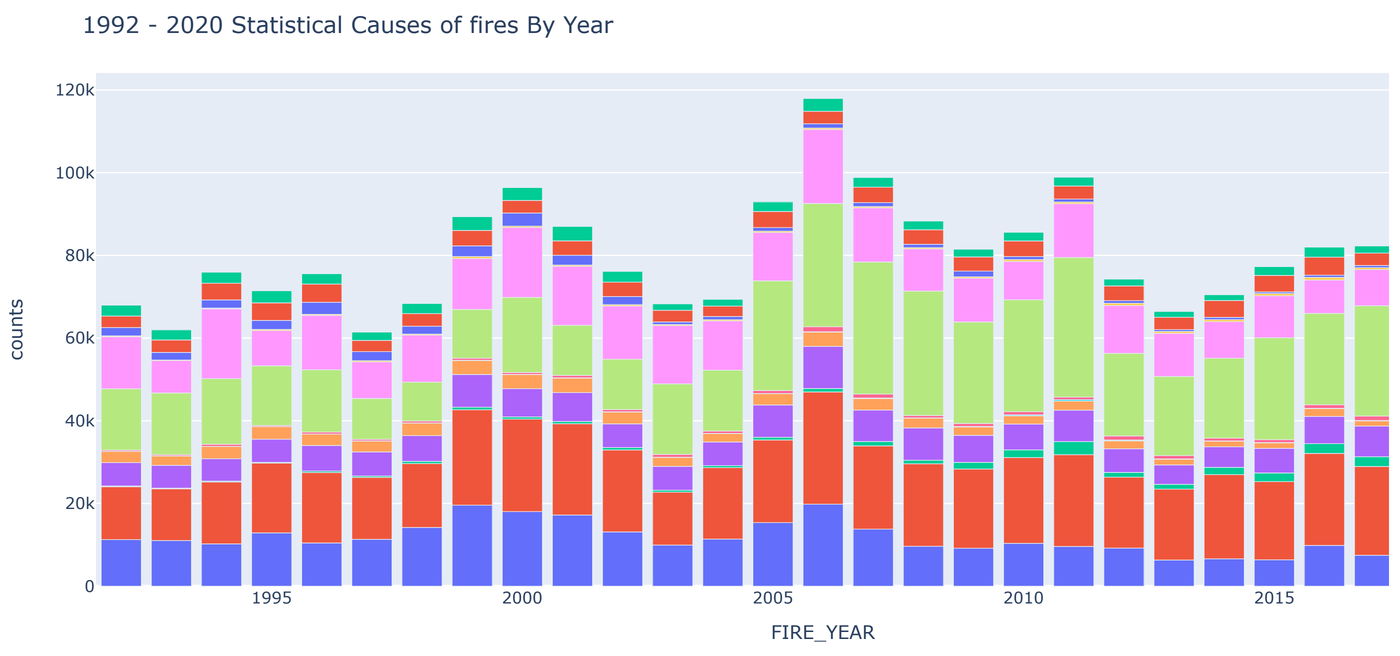
```
wildfire_causes = wildfire_df[['NWCG_GENERAL_CAUSE', 'FIRE_YEAR', 'STATE']]
wildfire_causes.head()

# grouping cause and year by counts
uni_values = wildfire_causes[['NWCG_GENERAL_CAUSE', 'FIRE_YEAR']] \
            .groupby(['NWCG_GENERAL_CAUSE', 'FIRE_YEAR']).size().reset_index(name='counts') \
            .sort_values(by=['NWCG_GENERAL_CAUSE'])

fig = px.bar(uni_values, x = 'FIRE_YEAR', y = 'counts', color = 'NWCG_GENERAL_CAUSE')
fig.update_layout(
                  title_text = '1992 - 2020 Statistical Causes of fires By Year',

)

fig.show()
```



Results : In 2006, the incidence of wildfires was at its highest compared to other years. The causes of these fires are distributed across various factors, making it challenging to identify a specific primary cause from the graph.

## Further Analysis of Fire Cause with most wildfire impacted states

Since California, Texas, and Georgia states stand out for their high wildfire intensities, I've chosen to focus my analysis on them to understand the main reasons behind the wildfires.

```
w_df_CA = wildfire_df[wildfire_df['STATE']=='CA']
w_df_GA = wildfire_df[wildfire_df['STATE']=='GA']
w_df_TX = wildfire_df[wildfire_df['STATE']=='TX']
```

## Cause Classification of Fire

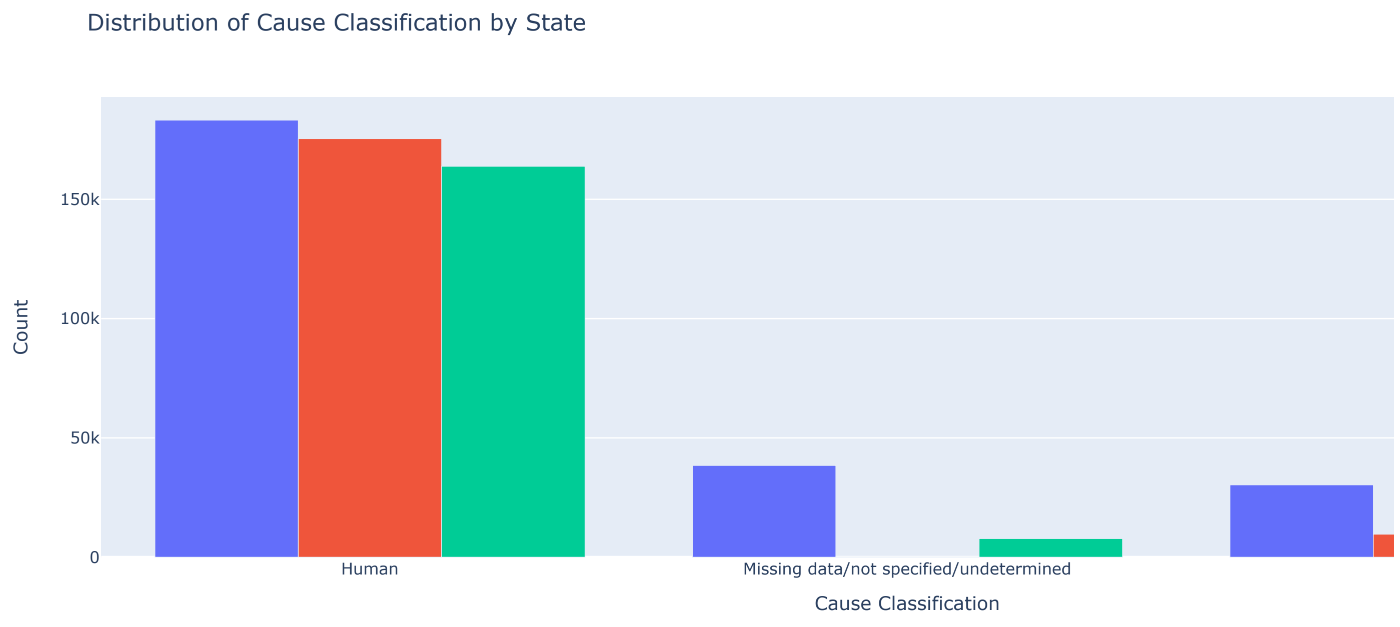
```
# Create grouped bar charts for each DataFrame
fig = go.Figure()

CA_value_counts = w_df_CA['NWCG_CAUSE_CLASSIFICATION'].value_counts()
GA_value_counts = w_df_GA['NWCG_CAUSE_CLASSIFICATION'].value_counts()
TX_value_counts = w_df_TX['NWCG_CAUSE_CLASSIFICATION'].value_counts()

fig.add_trace(go.Bar(x=CA_value_counts.index, y=CA_value_counts.values, name='CA'))
fig.add_trace(go.Bar(x=GA_value_counts.index, y=GA_value_counts.values, name='GA'))
fig.add_trace(go.Bar(x=TX_value_counts.index, y=TX_value_counts.values, name='TX'))

# Customize the layout
fig.update_layout(
    title='Distribution of Cause Classification by State',
    xaxis_title='Cause Classification',
    yaxis_title='Count')

# Show the grouped bar chart
fig.show()
```



## Results : Across all 3 states, Human activity is the main contributing factor to the fires.

```
# filtering 3 states
state_mask = wildfire_df['STATE'].isin(['CA','TX','GA'])
max_fire_state_df = wildfire_df[state_mask]

# select all rows except the ones that contain 'Missing/Unknown' and 'other'
mask = max_fire_state_df['NWCG_GENERAL_CAUSE'].isin(['Missing/Unknown', 'Other Causes'])
max_fire_state_df = max_fire_state_df[~mask]
```

```
max_fire_state_df
```

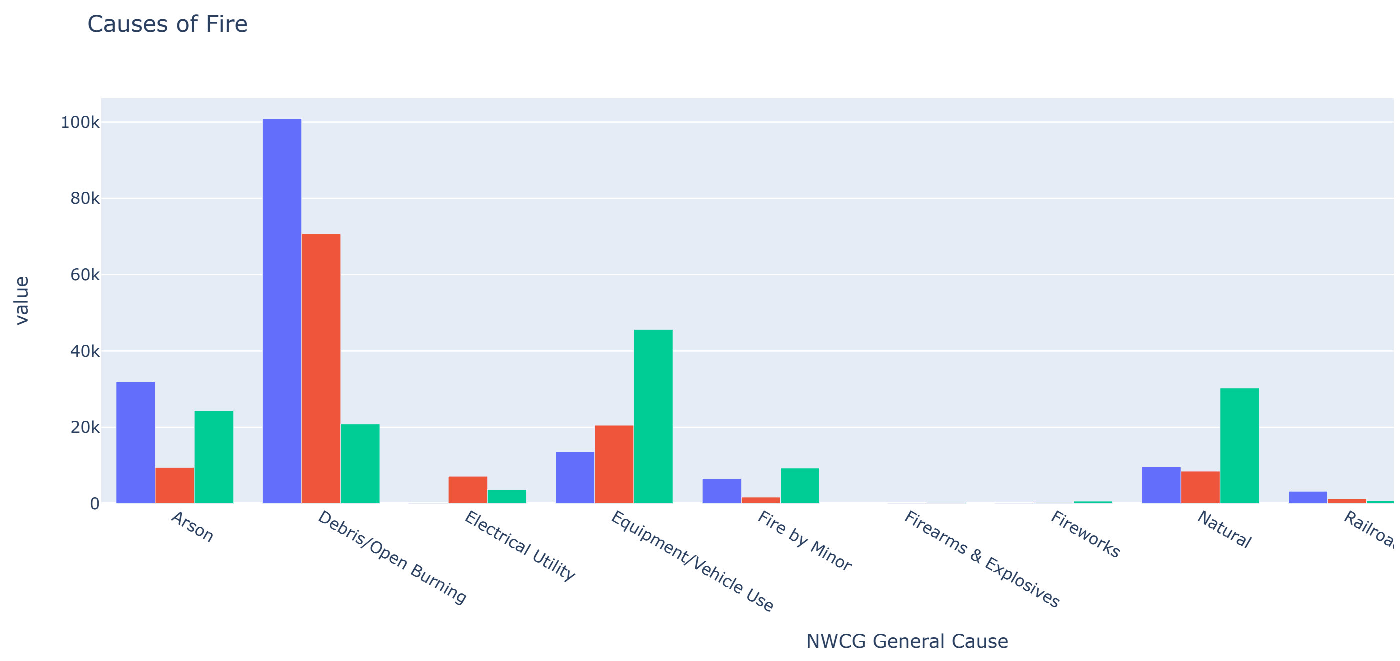
| | FIRE_YEAR | DISCOVERY_DATE | NWCG_CAUSE_CLASSIFICATION | NWCG_GENERAL_CAUSE | FIRE_SIZE | FIRE_SIZE_CLASS | LATITUDE | LONGIT |
|---|---|---|---|---|---|---|---|---|
| **0** | 2005 | 2/2/2005 | Human | Electrical Utility | 0.10 | A | 40.036944 | -121.005 |
| **1** | 2004 | 5/12/2004 | Natural | Natural | 0.25 | A | 38.933056 | -120.404 |
| **2** | 2004 | 5/31/2004 | Human | Debris/Open Burning | 0.10 | A | 38.984167 | -120.735 |
| **3** | 2004 | 6/28/2004 | Natural | Natural | 0.10 | A | 38.559167 | -119.913 |
| **4** | 2004 | 6/28/2004 | Natural | Natural | 0.10 | A | 38.559167 | -119.933 |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **2300333** | 2020 | 4/2/2020 | Human | Recreation/Ceremony | 0.10 | A | 34.962287 | -85.382 |
| **2300342** | 2020 | 3/10/2020 | Human | Electrical Utility | 0.60 | B | 33.025839 | -83.703 |
| **2300344** | 2020 | 2/29/2020 | Human | Arson | 0.10 | A | 33.262506 | -83.727 |
| **2300345** | 2020 | 4/15/2020 | Human | Debris/Open Burning | 0.10 | A | 33.303284 | -83.439 |
| **2302309** | 2020 | 7/1/2020 | Natural | Natural | 0.20 | A | 30.878289 | -81.439 |

457549 rows × 9 columns

```
# Cause and STATE by counts
counts = max_fire_state_df.groupby(['NWCG_GENERAL_CAUSE', 'STATE']).size().reset_index(name='COUNT').sort_values('COUNT', ascend
df_filtered = counts[counts['COUNT'] > 0]
df_pivot = df_filtered.pivot(index='NWCG_GENERAL_CAUSE', columns='STATE', values='COUNT')
```

## Cause of Fire : CA, TX, GA

```
# Plotting with Plotly Express
fig = px.bar(df_pivot, barmode='group',
             labels={'COUNT': 'Count', 'NWCG_GENERAL_CAUSE': 'NWCG General Cause'},
             title='Causes of Fire')
fig.show()
```



Among all 3 states, Equipment/Vehicle Use, Open buring and Arson causes the wildfire. State and Cause of Fire gives useful information. So STATE is a useful feature to include in the machine learning model.

- Debris/Open Burning : Vegetation, dead plants, and other organic materials are on the forest floor. These ground materials on the ground acts like fuel for the fire, making it spread. This fire can start due to lightning or human activity.
- Natural : During the storm, lightning strikes a tree, creating a spark. This spark ignites a fire.

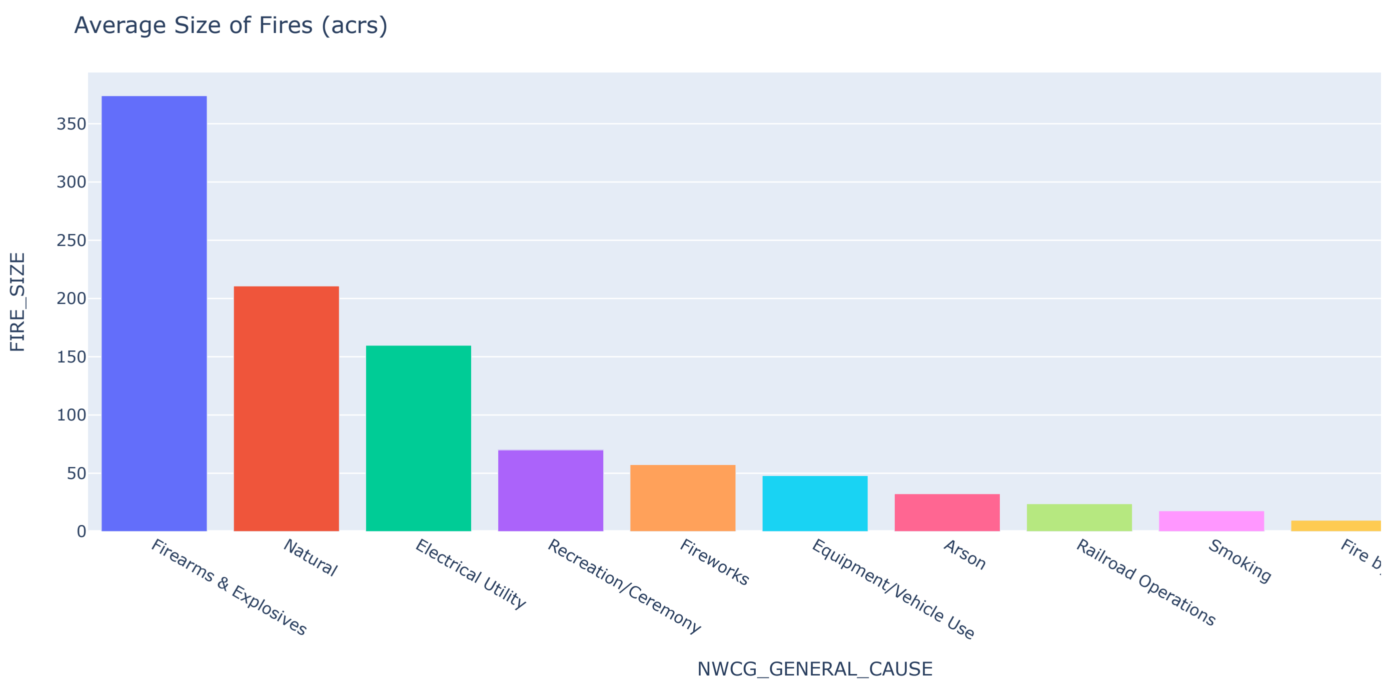
## Average Size of Fires (acrs)

```
max_fire_state_df.head()
```

| | FIRE_YEAR | DISCOVERY_DATE | NWCG_CAUSE_CLASSIFICATION | NWCG_GENERAL_CAUSE | FIRE_SIZE | FIRE_SIZE_CLASS | LATITUDE | LONGITUDE | S |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 2005 | 2/2/2005 | Human | Electrical Utility | 0.10 | A | 40.036944 | -121.005833 | |
| **1** | 2004 | 5/12/2004 | Natural | Natural | 0.25 | A | 38.933056 | -120.404444 | |
| **2** | 2004 | 5/31/2004 | Human | Debris/Open Burning | 0.10 | A | 38.984167 | -120.735556 | |
| **3** | 2004 | 6/28/2004 | Natural | Natural | 0.10 | A | 38.559167 | -119.913333 | |
| **4** | 2004 | 6/28/2004 | Natural | Natural | 0.10 | A | 38.559167 | -119.933056 | |

```
# Average size of fire by cause
avg_fire_size_cause = max_fire_state_df.groupby('NWCG_GENERAL_CAUSE')['FIRE_SIZE'].mean().reset_index().sort_values(by='FIRE_SIZ

fig = px.bar(avg_fire_size_cause, x = 'NWCG_GENERAL_CAUSE', y = 'FIRE_SIZE', color = 'NWCG_GENERAL_CAUSE')
fig.update_layout(
                  title_text = 'Average Size of Fires (acrs)')
fig.show()
```



The average size of fires in arcs is influenced by various causes. Factors such as the use of firearms and explosives, natural events, and electrical utilities, including power lines, contribute significantly to larger fires.

## Feature Engineering

```
# copy of df
df = max_fire_state_df.copy()


# creating DATE feature
df['DATE'] = pd.to_datetime(df['DISCOVERY_DATE'])


# creating MONTH feature
df['MONTH'] = pd.DatetimeIndex(df['DATE']).month
df_actual = df.copy() # for later use
print(df.head())
```

```
   FIRE_YEAR DISCOVERY_DATE NWCG_CAUSE_CLASSIFICATION   NWCG_GENERAL_CAUSE  \
0       2005       2/2/2005                     Human   Electrical Utility
1       2004      5/12/2004                   Natural              Natural
2       2004      5/31/2004                     Human  Debris/Open Burning
3       2004      6/28/2004                   Natural              Natural
4       2004      6/28/2004                   Natural              Natural

   FIRE_SIZE FIRE_SIZE_CLASS   LATITUDE   LONGITUDE STATE       DATE  MONTH
0       0.10               A  40.036944 -121.005833    CA 2005-02-02      2
1       0.25               A  38.933056 -120.404444    CA 2004-05-12      5
2       0.10               A  38.984167 -120.735556    CA 2004-05-31      5
3       0.10               A  38.559167 -119.913333    CA 2004-06-28      6
4       0.10               A  38.559167 -119.933056    CA 2004-06-28      6
```

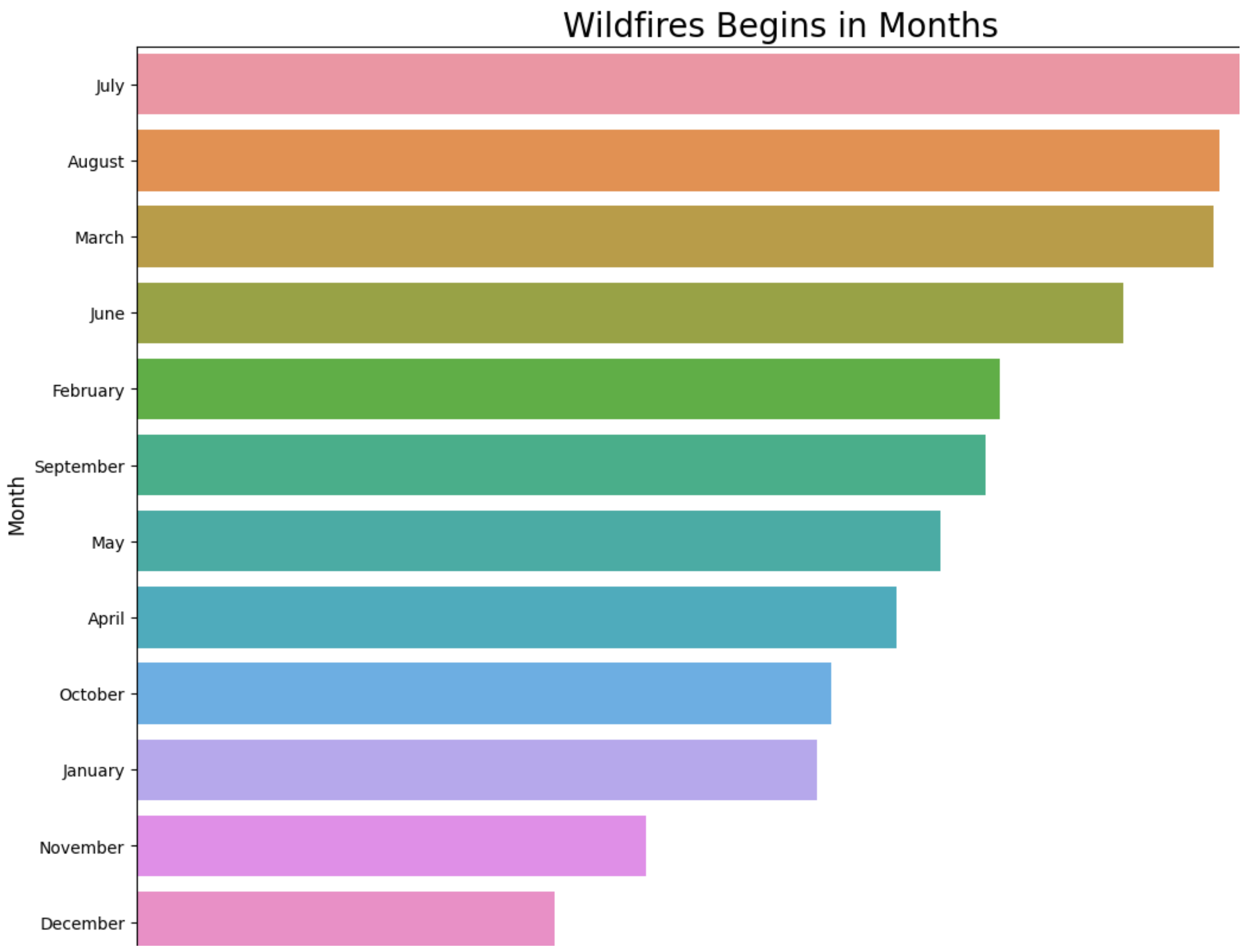
```
df.dtypes
```

```
FIRE_YEAR                             int16
DISCOVERY_DATE                       object
NWCG_CAUSE_CLASSIFICATION          category
NWCG_GENERAL_CAUSE                   object
FIRE_SIZE                           float64
FIRE_SIZE_CLASS                    category
LATITUDE                            float64
LONGITUDE                           float64
STATE                              category
DATE                         datetime64[ns]
MONTH                                 int64
dtype: object
```

## Month analysis of Fire

```
# creating New Feature
# Converting DATE into a month
df['MONTH_string'] = df['DATE'].dt.strftime("%B")

month_cnt = df['MONTH_string'].value_counts()
plt.figure(figsize=(14,10))
sns.barplot(y=month_cnt.index, x=month_cnt)
plt.title('Wildfires Begins in Months', fontsize=20)
plt.ylabel('Month', fontsize=12)
plt.xlabel('Counts', fontsize=12);
```



Results : Late spring through summer are the months in which most of fires occur throughout the year. Firstly, the temperature is high and conditions are more arid leading to spontaneous fires that result from 'debris burning'. Second, people spend more time outside which leads to higher chances of accidental fires.

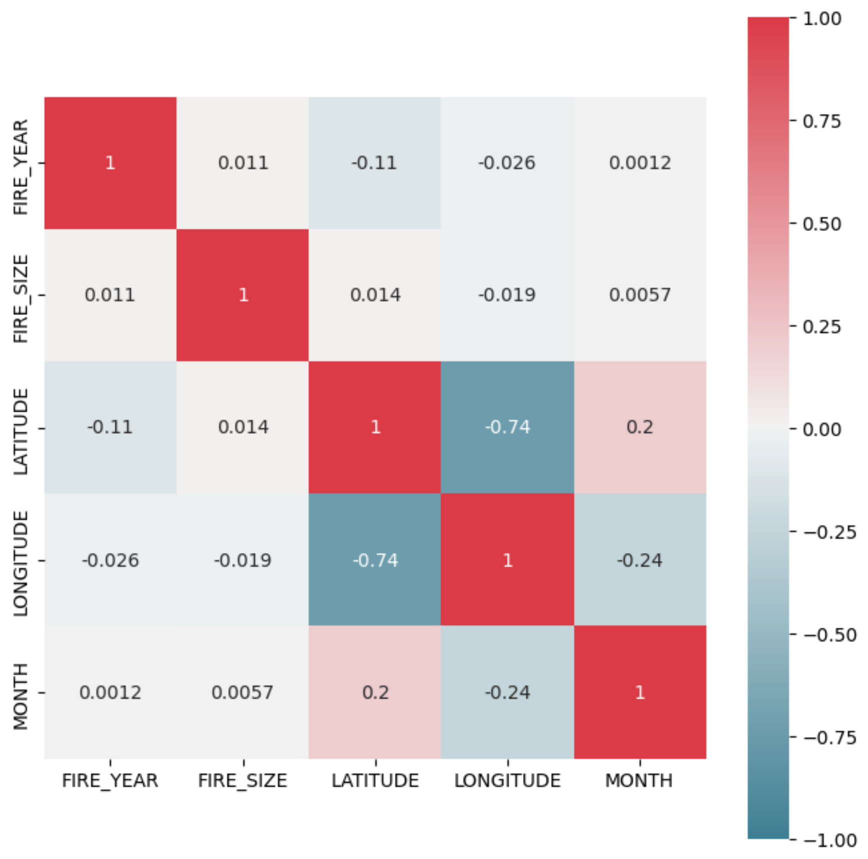
## Co-relation Matrix

```
f, ax = plt.subplots(figsize=(8, 8))
corr = df.corr()
sns.heatmap(corr, annot=True, cmap=sns.diverging_palette(220, 10, as_cmap=True), vmin=-1.0, vmax=1.0, square=True, ax=ax)
```

```
<ipython-input-34-38db8b14aba9>:2: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only

<Axes: >
```



## Model Building

```
from sklearn import tree, preprocessing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report,roc_curve,precision_score,recall_score,auc,f1_
```

```
# using label encoder for NWCG_GENERAL_CAUSE which basically converts category into number
le = preprocessing.LabelEncoder()
df['NWCG_GENERAL_CAUSE'] = le.fit_transform(df['NWCG_GENERAL_CAUSE'])
df['STATE'] = le.fit_transform(df['STATE'])
print(df.head())
```

```
   FIRE_YEAR DISCOVERY_DATE NWCG_CAUSE_CLASSIFICATION  NWCG_GENERAL_CAUSE  \
0       2005       2/2/2005                     Human                   2
1       2004      5/12/2004                   Natural                   7
2       2004      5/31/2004                     Human                   1
3       2004      6/28/2004                   Natural                   7
4       2004      6/28/2004                   Natural                   7

   FIRE_SIZE FIRE_SIZE_CLASS   LATITUDE   LONGITUDE  STATE       DATE  MONTH  \
0       0.10               A  40.036944 -121.005833      0 2005-02-02      2
1       0.25               A  38.933056 -120.404444      0 2004-05-12      5
2       0.10               A  38.984167 -120.735556      0 2004-05-31      5
3       0.10               A  38.559167 -119.913333      0 2004-06-28      6
4       0.10               A  38.559167 -119.933056      0 2004-06-28      6

  MONTH_string
0     February
```

```
    1         May
    2         May
    3        June
    4        June
```

When analyzing this correlation matrix, there are no features that have strong correlations, or at least correlations greater than 0.3. It's interesting to note that the fire year has a slight correlation with STATE. It's clear that FIRE YEAR should relate to STATE which is 0.25, but it's nice to see that represented visually. Also, GENERAL CAUSE has correlation with STATE and LONGITUDE & LATITUDE. Based on the above analysis of different features, let's move to Machine Learning Models.

## Machine Learning Models

```
df = df.drop(['DATE','DISCOVERY_DATE','NWCG_CAUSE_CLASSIFICATION','FIRE_SIZE_CLASS','MONTH_string'],axis=1)
df = df.dropna()
```

```
X = df.drop(['NWCG_GENERAL_CAUSE'], axis=1).values
y = df['NWCG_GENERAL_CAUSE'].values
```

```
df
```

| | FIRE_YEAR | NWCG_GENERAL_CAUSE | FIRE_SIZE | LATITUDE | LONGITUDE | STATE | MONTH |
|---|---|---|---|---|---|---|---|
| **0** | 2005 | 2 | 0.10 | 40.036944 | -121.005833 | 0 | 2 |
| **1** | 2004 | 7 | 0.25 | 38.933056 | -120.404444 | 0 | 5 |
| **2** | 2004 | 1 | 0.10 | 38.984167 | -120.735556 | 0 | 5 |
| **3** | 2004 | 7 | 0.10 | 38.559167 | -119.913333 | 0 | 6 |
| **4** | 2004 | 7 | 0.10 | 38.559167 | -119.933056 | 0 | 6 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **2300333** | 2020 | 9 | 0.10 | 34.962287 | -85.382505 | 1 | 4 |
| **2300342** | 2020 | 2 | 0.60 | 33.025839 | -83.703334 | 1 | 3 |
| **2300344** | 2020 | 0 | 0.10 | 33.262506 | -83.727226 | 1 | 2 |
| **2300345** | 2020 | 1 | 0.10 | 33.303284 | -83.439060 | 1 | 4 |
| **2302309** | 2020 | 7 | 0.20 | 30.878289 | -81.439409 | 1 | 7 |

457549 rows × 7 columns

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state=0) #30% for testing, 70% for training
```

```
def get_confusion_matrix(cm,title):
    ax = sns.heatmap(cm,annot=True,fmt='g',cmap='Blues',annot_kws={"fontsize":14})
    ax.set_title(title+'\n\n');
    ax.set_xlabel('\nPredicted Values')
    ax.set_ylabel('Actual Values');
    return plt.show()

def classification_algorithm(algo_instance, X_train, y_train, X_test, title):
  print("{} Algorithm :".format(title))
  algo_pred = algo_instance.fit(X_train, y_train)
  algo_pred = algo_instance.predict(X_test)
  algo_cm = confusion_matrix(y_test, algo_pred)
  print("\n------ Acurracy of {} Algorithm ------".format(title))
  algo_score = algo_instance.score(X_test,y_test)
  print("{} Score : {}".format(title,round(algo_score,2)))
```

## Method 1 : Default ML Models

Initially, I have decided to use a Logistic Regression algorithm and Support Vector Machine. Since it's a multi classification problem, both Logistic Regression and Support Vector Machine work on binary classification problems. Thus, I have decided to use classification algorithms like Decision Trees, Random Forest and XGBoost which can perform well on multi classification problems.

## Decision Trees

```
classification_algorithm(DecisionTreeClassifier(), X_train, y_train, X_test, title='Decision Tree Classifier')
```

```
Decision Tree Classifier Algorithm :

------ Acurracy of Decision Tree Classifier Algorithm ------
Decision Tree Classifier Score : 0.49
```

## Random Forest

```
classification_algorithm(RandomForestClassifier(), X_train, y_train, X_test, title='Random Forest Classifier')
```

```
Random Forest Classifier Algorithm :

------ Acurracy of Random Forest Classifier Algorithm ------
Random Forest Classifier Score : 0.6
```

## XGBoost

```
classification_algorithm(XGBClassifier(), X_train, y_train, X_test, title='XGB Classifier')
```

```
XGB Classifier Algorithm :

------ Acurracy of XGB Classifier Algorithm ------
XGB Classifier Score : 0.6
```

Note : Logistic Regression and SVM performs well on binary classifer. However, its a multi-classification problem, we can not able to use this two models.

## Method 2 : Improved Machine Learning Models

The Machine Learning models encounter difficulties distinguishing between numerous classes, with getting very low accuracy results. To refine model performance, I plan to consolidate these classes and reevaluate the results.

I have combined some cause Label from 12 cause to 4 cause labels (Target variable) :

The improved ML models will use these four labels as target variable and each label information is below:

**Label 0 = Natural**

- Natural

**Label 1 = Accidentally** (Accidentally refers to actions which spread of a wildfire due to carelessness, recklessness, or failure to take reasonable precautions.)

- Recreation/Ceremony :- Recreation : Campfire, Ceremony : Certain cultural or religious ceremonies involve the use of fire.
- Raildroad Operations
- Fireworks
- Smoking
- Fire by a Minor
- Equipment/Vehicle Use
- Debris/Open Burning

**Label 2 = Arson**

- Arson

**Label 3 = Other**

- Firearms & Explosives
- Missing/Undefined,
- Miscellaneous

```
# created function which combined causes into four different groups
def set_cause_label(category):
    cause = 0
    natural = ['Natural']
    accidentally = ['Fireworks','Railroad Operations','Smoking','Fire by a Minor','Recreation/Ceremony',
                    'Equipment/Vehicle Use','Debris/Open Burning']
    arson = ['Arson']
    other = ['Missing/Undefined','Miscellaneous','Firearms & Explosives']
    if category in natural:
        cause = 0
    elif category in accidentally:
        cause = 1
    elif category in arson:
        cause = 2
    else:
        cause = 3
    return cause
df['LABEL'] = df_actual['NWCG_GENERAL_CAUSE'].apply(lambda x: set_cause_label(x)) # used previous created df
df = df.drop('NWCG_GENERAL_CAUSE',axis=1)
print(df.head())
```

```
   FIRE_YEAR  FIRE_SIZE   LATITUDE   LONGITUDE  STATE  MONTH  LABEL
0       2005       0.10  40.036944 -121.005833      0      2      3
1       2004       0.25  38.933056 -120.404444      0      5      0
2       2004       0.10  38.984167 -120.735556      0      5      1
3       2004       0.10  38.559167 -119.913333      0      6      0
4       2004       0.10  38.559167 -119.933056      0      6      0
```

```
# check no of unique labels
df['LABEL'].unique()
```

```
array([3, 0, 1, 2])
```

```
# selecting depenedent and independent features
X = df.drop(['LABEL'], axis=1).values
y = df['LABEL'].values
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state=0)
```

## Decision Trees Improved

```
classification_algorithm(DecisionTreeClassifier(), X_train, y_train, X_test, title='Decision Tree Classifier')
```

```
Decision Tree Classifier Algorithm :

------ Acurracy of Decision Tree Classifier Algorithm ------
Decision Tree Classifier Score : 0.71
```

## Random Forest Improved

```
classification_algorithm(RandomForestClassifier(n_estimators=150), X_train, y_train, X_test, title='Random Forest Classifier')
```

```
Random Forest Classifier Algorithm :

------ Acurracy of Random Forest Classifier Algorithm ------
Random Forest Classifier Score : 0.8
```

## XGBoost Improved

```
classification_algorithm(XGBClassifier(), X_train, y_train, X_test, title='XGB Classifier')
```

```
XGB Classifier Algorithm :

------ Acurracy of XGB Classifier Algorithm ------
XGB Classifier Score : 0.79
```

## Hyperparamter Tuning on improved ML models :

As per the above improved machine learning models method, Random Forest and XGBoost algorithm performs well on default parameters. Therefore, the focus shifts to discovering the optimal combination of hyperparameter values for these algorithms, aiming for the highest possible results and a robust model.

```
## Hyperparameter Tuning with Random Forest

rf = RandomForestClassifier()
forest_params = [{'n_estimators':[50,150,200], 'criterion':['gini','entropy']}]

clf = GridSearchCV(rf, forest_params, cv = 5,verbose=2)

clf.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[CV] END ....................criterion=gini, n_estimators=50; total time=  37.0s
[CV] END ....................criterion=gini, n_estimators=50; total time=  27.4s
[CV] END ....................criterion=gini, n_estimators=50; total time=  25.6s
[CV] END ....................criterion=gini, n_estimators=50; total time=  25.6s
[CV] END ....................criterion=gini, n_estimators=50; total time=  26.0s
[CV] END ...................criterion=gini, n_estimators=150; total time= 1.2min
[CV] END ...................criterion=gini, n_estimators=150; total time= 1.2min
[CV] END ...................criterion=gini, n_estimators=150; total time= 1.2min
[CV] END ...................criterion=gini, n_estimators=150; total time= 1.2min
[CV] END ...................criterion=gini, n_estimators=150; total time= 1.2min
[CV] END ...................criterion=gini, n_estimators=200; total time= 1.7min
[CV] END ...................criterion=gini, n_estimators=200; total time= 1.7min
[CV] END ...................criterion=gini, n_estimators=200; total time= 1.6min
[CV] END ...................criterion=gini, n_estimators=200; total time= 1.6min
[CV] END ...................criterion=gini, n_estimators=200; total time= 1.7min
[CV] END .................criterion=entropy, n_estimators=50; total time=  30.6s
[CV] END .................criterion=entropy, n_estimators=50; total time=  30.1s
[CV] END .................criterion=entropy, n_estimators=50; total time=  32.1s
[CV] END .................criterion=entropy, n_estimators=50; total time=  30.0s
[CV] END .................criterion=entropy, n_estimators=50; total time=  29.9s
[CV] END ................criterion=entropy, n_estimators=150; total time= 1.5min
[CV] END ................criterion=entropy, n_estimators=150; total time= 1.5min
[CV] END ................criterion=entropy, n_estimators=150; total time= 1.5min
[CV] END ................criterion=entropy, n_estimators=150; total time= 1.5min
[CV] END ................criterion=entropy, n_estimators=150; total time= 1.5min
[CV] END ................criterion=entropy, n_estimators=200; total time= 2.0min
[CV] END ................criterion=entropy, n_estimators=200; total time= 2.0min
[CV] END ................criterion=entropy, n_estimators=200; total time= 2.0min
[CV] END ................criterion=entropy, n_estimators=200; total time= 2.0min
[CV] END ................criterion=entropy, n_estimators=200; total time= 2.0min
```

```
▸            GridSearchCV
▸ estimator: RandomForestClassifier
        ▸ RandomForestClassifier
```

```
# Random Forest best parameter
clf.best_params_
```

```
{'criterion': 'gini', 'n_estimators': 200}
```

```
# Hyper parameter tuning with XGBoost
xg = XGBClassifier()
xgb_params = [{'learning_rate': [0.01, 0.1, 0.3], 'n_estimators': [150, 200, 500], 'objective': ["multi:softmax"]}]

xg_clf = GridSearchCV(xg, xgb_params, cv = 5,verbose=2)

xg_clf.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV] END learning_rate=0.01, n_estimators=150, objective=multi:softmax; total time=  25.4s
[CV] END learning_rate=0.01, n_estimators=150, objective=multi:softmax; total time=  14.9s
[CV] END learning_rate=0.01, n_estimators=150, objective=multi:softmax; total time=  11.9s
[CV] END learning_rate=0.01, n_estimators=150, objective=multi:softmax; total time=  11.8s
[CV] END learning_rate=0.01, n_estimators=150, objective=multi:softmax; total time=  11.8s
[CV] END learning_rate=0.01, n_estimators=200, objective=multi:softmax; total time=  14.3s
[CV] END learning_rate=0.01, n_estimators=200, objective=multi:softmax; total time=  14.1s
[CV] END learning_rate=0.01, n_estimators=200, objective=multi:softmax; total time=  14.1s
[CV] END learning_rate=0.01, n_estimators=200, objective=multi:softmax; total time=  14.3s
[CV] END learning_rate=0.01, n_estimators=200, objective=multi:softmax; total time=  14.4s
[CV] END learning_rate=0.01, n_estimators=500, objective=multi:softmax; total time=  36.3s
[CV] END learning_rate=0.01, n_estimators=500, objective=multi:softmax; total time=  38.6s
[CV] END learning_rate=0.01, n_estimators=500, objective=multi:softmax; total time=  35.4s
[CV] END learning_rate=0.01, n_estimators=500, objective=multi:softmax; total time=  36.2s
[CV] END learning_rate=0.01, n_estimators=500, objective=multi:softmax; total time=  36.3s
[CV] END learning_rate=0.1, n_estimators=150, objective=multi:softmax; total time=  10.6s
[CV] END learning_rate=0.1, n_estimators=150, objective=multi:softmax; total time=  10.6s
[CV] END learning_rate=0.1, n_estimators=150, objective=multi:softmax; total time=  12.5s
[CV] END learning_rate=0.1, n_estimators=150, objective=multi:softmax; total time=  10.6s
[CV] END learning_rate=0.1, n_estimators=150, objective=multi:softmax; total time=  10.8s
[CV] END learning_rate=0.1, n_estimators=200, objective=multi:softmax; total time=  13.1s
[CV] END learning_rate=0.1, n_estimators=200, objective=multi:softmax; total time=  13.2s
[CV] END learning_rate=0.1, n_estimators=200, objective=multi:softmax; total time=  13.1s
[CV] END learning_rate=0.1, n_estimators=200, objective=multi:softmax; total time=  13.1s
[CV] END learning_rate=0.1, n_estimators=200, objective=multi:softmax; total time=  13.2s
[CV] END learning_rate=0.1, n_estimators=500, objective=multi:softmax; total time=  31.3s
[CV] END learning_rate=0.1, n_estimators=500, objective=multi:softmax; total time=  30.5s
[CV] END learning_rate=0.1, n_estimators=500, objective=multi:softmax; total time=  33.0s
[CV] END learning_rate=0.1, n_estimators=500, objective=multi:softmax; total time=  30.7s
[CV] END learning_rate=0.1, n_estimators=500, objective=multi:softmax; total time=  31.6s
[CV] END learning_rate=0.3, n_estimators=150, objective=multi:softmax; total time=   8.2s
[CV] END learning_rate=0.3, n_estimators=150, objective=multi:softmax; total time=  10.1s
[CV] END learning_rate=0.3, n_estimators=150, objective=multi:softmax; total time=  10.3s
[CV] END learning_rate=0.3, n_estimators=150, objective=multi:softmax; total time=  10.0s
[CV] END learning_rate=0.3, n_estimators=150, objective=multi:softmax; total time=   8.1s
[CV] END learning_rate=0.3, n_estimators=200, objective=multi:softmax; total time=  12.7s
[CV] END learning_rate=0.3, n_estimators=200, objective=multi:softmax; total time=  12.7s
[CV] END learning_rate=0.3, n_estimators=200, objective=multi:softmax; total time=  16.9s
[CV] END learning_rate=0.3, n_estimators=200, objective=multi:softmax; total time=  12.7s
[CV] END learning_rate=0.3, n_estimators=200, objective=multi:softmax; total time=  12.4s
[CV] END learning_rate=0.3, n_estimators=500, objective=multi:softmax; total time=  31.7s
[CV] END learning_rate=0.3, n_estimators=500, objective=multi:softmax; total time=  30.8s
[CV] END learning_rate=0.3, n_estimators=500, objective=multi:softmax; total time=  31.2s
[CV] END learning_rate=0.3, n_estimators=500, objective=multi:softmax; total time=  30.8s
[CV] END learning_rate=0.3, n_estimators=500, objective=multi:softmax; total time=  33.2s
```

```
▸       GridSearchCV
▸ estimator: XGBClassifier
     ▸ XGBClassifier
```

```
# XGBoost best parameter
xg_clf.best_params_
```

```
{'learning_rate': 0.3, 'n_estimators': 500, 'objective': 'multi:softmax'}
```

## Camparing Models

```
# Application of all Machine Learning methods

MLA_dict = {'Decision Tree' : DecisionTreeClassifier(),
            'Random Forest': RandomForestClassifier(n_estimators=150),
            'Random Forest with Best Param': RandomForestClassifier(n_estimators=200, criterion= 'gini'),
            'XGBoost':XGBClassifier(),
            'XGBoost with Best param':XGBClassifier(objective="multi:softmax",n_estimators=500,learning_rate=0.3)
             }


def campare_models(MLA_dict,X_train,y_train):
    MLA_columns = []
    MLA_compare = pd.DataFrame(columns = MLA_columns)

    row_index = 0
    for algname, alg in MLA_dict.items():
        predicted = alg.fit(X_train, y_train).predict(X_test)
        MLA_name = algname
        MLA_compare.loc[row_index,'MLA used'] = MLA_name
        MLA_compare.loc[row_index, 'Accuracy'] = round(alg.score(X_test, y_test), 4)
        MLA_compare.loc[row_index, 'Precission'] = round(precision_score(y_test, predicted, average='micro'),3)
        MLA_compare.loc[row_index, 'Recall'] = round(recall_score(y_test, predicted, average='micro'),2)
        MLA_compare.loc[row_index, 'F1-Score'] = round(f1_score(y_test, predicted, average='micro'),2)

        row_index+=1

    return MLA_compare
```

```
MLA_compare = campare_models(MLA_dict=MLA_dict,X_train=X_train,y_train=y_train)
MLA_compare
```

| | MLA used | Accuracy | Precission | Recall | F1-Score |
|---|---|---|---|---|---|
| **0** | Decision Tree | 0.7123 | 0.712 | 0.71 | 0.71 |
| **1** | Random Forest | 0.7984 | 0.798 | 0.80 | 0.80 |
| **2** | Random Forest with Best Param | 0.7992 | 0.799 | 0.80 | 0.80 |
| **3** | XGBoost | 0.7879 | 0.788 | 0.79 | 0.79 |
| **4** | XGBoost with Best param | 0.7957 | 0.796 | 0.80 | 0.80 |

Results : When camparing all the default parameters and hyperparameter tuned model, Random Forest with Best Parameters performing with highest accuarcy which is 0.7992 and highest Recall 0.80 Lets check the confusion matrix for the best algorthm.

## Confusion Matrix

Confusion Matrix evaluates the performance of a classification algorithm on a set of data for which the true values are known. Below is the confusion matrix for predicting 4 different cause labels such as Natural, Accidentally, Arson and Other.

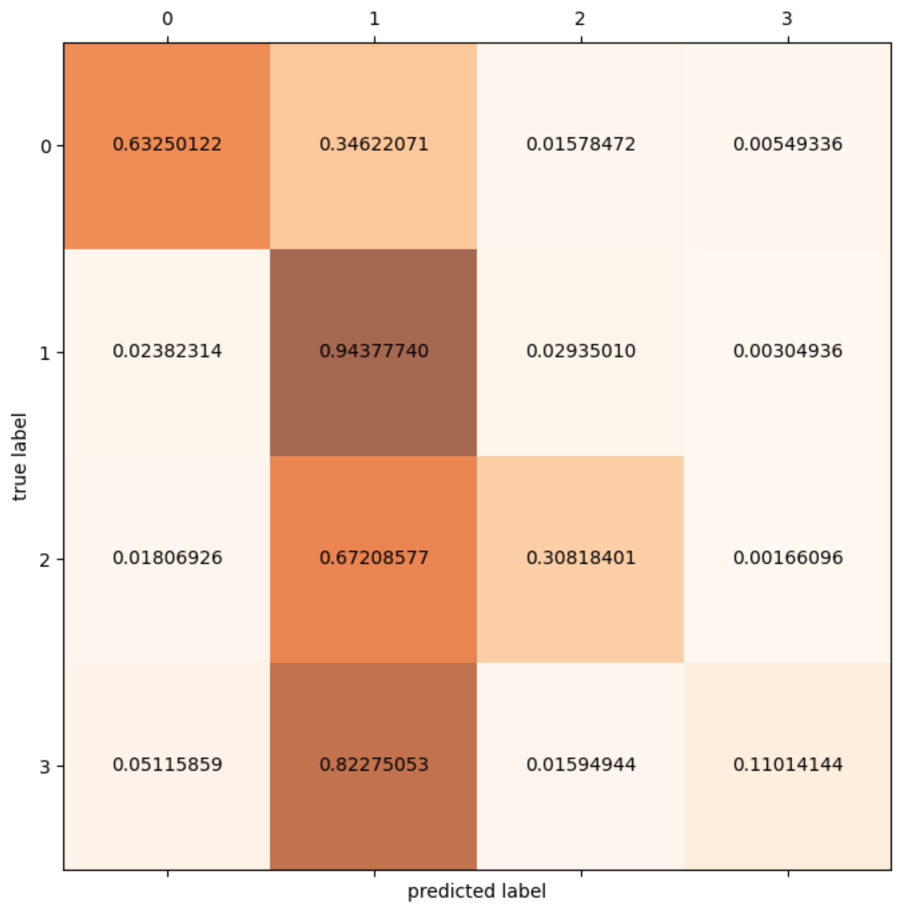
```
def plot_confusion_matrix(cm):
    # cmn = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.figure(figsize=(8, 8))
    sns.heatmap(cm, annot=True, cmap="Oranges", fmt=".2f", linewidths=.5, square=True)
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()


def get_cm_precentage(cm):
    cmn = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    fig,ax = plt.subplots(figsize=(8,18))
    ax.matshow(cmn,cmap=plt.cm.Oranges,alpha=0.7)
    for i in range(cmn.shape[0]):
        for j in range(cmn.shape[1]):
            value = "{:.8f}".format(cmn[i, j])
            ax.text(x=j,y=i,s=value,va='center',ha='center')
    plt.xlabel('predicted label')
    plt.ylabel('true label')
    return plt.show()
```

```
# Confusion matrix of Random forest
y_pred = RandomForestClassifier(n_estimators=200, criterion= 'gini').fit(X_train, y_train).predict(X_test)
cm = confusion_matrix(y_true=y_test,y_pred=y_pred)
print(cm)
```

```
[[ 9096  4979   227    79]
 [ 2375 94088  2926   304]
 [  359 13353  6123    33]
 [  170  2734    53   366]]
```

```
# actual vs Predicted label of confusion matrix(precentage)
get_cm_precentage(cm)
```



Results : Camparing above machine learning models using default classifer, improved classifer and hyperparameter tuned classifier models, The best model is Hyper-paramter tuned Random Forest which accuracy is around 0.7992 and Recall is around 0.80

Let's review the cause labels:

- Label 0 = Natural Cause
- Label 1 = Accidentally
- Label 2 = Arson
- Label 3 = Other

• The model achieved a 63% accuracy when predicting natural causes.

• The model achieved a 94% accuracy when predicting Accidentally cause fire.

• The model achieved a 31% accuracy when predicting Arson.

• The model achieved a 11% accuracy when predicting Other.

## Best Model :

Random Forest with Accuracy : 0.80

## Conclusion

The analysis shows the distribution of data and demonstrates the impact of wildfires across the country. Later on analyze the most wildfire impacted states California, Texas and georgia. Wildfires were analyzed on locations like most wildfire impacted states, least wildfire impacted states, month of the year, average fire size per cause, cause classification of wildfire and many more characteristics.

And predicted causes of wildfire with the overall accuracy is around 60%. Thereafter, reducing the number of labels from twelve into four categories improved the prediction score to 80% for the random forest algorithm. As per the confusion matrix, unfortunately, the algorithm did not perform well when trying to distinguish between 'Arson' and 'Other" causes. The algorithm did perform relatively well, however, in distinguishing natural causes and Accidental causes.

### Future Work

In future consider expanding the dataset to include more comprehensive information on factors influencing wildfires. Also, refine and improve predictive models by exploring advanced machine learning techniques. Plus, I am planning to do from offline analysis to real-time deployment using Flask or Streamlit, allowing the model to provide timely predictions and insights.

## Recommendation

To emphasize the practical value of this project, fire departments and respective agencies could use the resulting models to predict future wildfire causes in California, Texas and Georgia.

The following recommendations would be used by fire reporting departments and agencies to allocate resources appropriately to fight future fires.

- **Check and replace defective Electrical Utility** infrastructure like power lines which has contributed to the third-largest average size of fires, as illustrated in below. Although these Electrical Utility related incidents are less frequent when analyzing total causes of fires, they have a significant impact, leading to some of the largest fires. Therefore, enhancing and upgrading electrical utility systems is crucial for reducing the severity of wildfires.