This notebook contains codes executed for the Final Project on Data Mining R&D for the course CSIT558_01SP25 by:

- Chaudhari, Amishaben Natavarbhai (CWID:50129394)
- Javed, Syed Mehrose (CWID: 50128848)
- Subhani, Muhammad Kamal (CWID: 50136417)
- Thangavel Sathiyamoorthy, Punithan (CWID: 50135877)
- Venkata Appala Manoj Muvvala (CWID: 50122981)

IMPORTING THE REQUIRED LIBRARIES

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
from sklearn.metrics import classification_report
from sklearn.preprocessing import MinMaxScaler,
StandardScaler,LabelEncoder
from sklearn.model_selection import RandomizedSearchCV
from sklearn.utils import resample
from sklearn.preprocessing import OneHotEncoder
from scipy.stats import randint
import sqlite3
from prophet import Prophet
from prophet.plot import add_changepoints_to_plot
from datetime import datetime as dt
from sklearn import linear_model
import xgboost as xgb
from sklearn.ensemble import RandomForestClassifier,
RandomForestRegressor
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import seaborn as sns
import calendar
import warnings
import geopandas as gpd
import folium
import requests
import random

%matplotlib inline
warnings.filterwarnings('ignore')
```

LOADING THE DATA

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

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Connect to the sqlite db file and retrieve data as Pandas data
frame.
path = '/content/drive/MyDrive/FPA_FOD_20170508.sqlite'
cnx = sqlite3.connect(path)
sql = "select * from fires"
df = pd.read_sql_query(sql, cnx)
```

```
df.head()
```

```
{"type":"dataframe","variable_name":"df"}
```

```
df.shape
```

```
(1880465, 39)
```

```
# # Sample of the data
# df = df.sample(frac=0.01, random_state=42)
# df.shape
```

1. PREDICTING THE LIKELIHOOD OF A WILDFIRE

FEATURE ENGINEERING AND EDA

```
# Standardize date and location info
df['DISCOVERY_DATE'] = pd.to_datetime(df['DISCOVERY_DATE'],
origin='julian', unit='D')
df['YEAR_MONTH'] =
df['DISCOVERY_DATE'].dt.to_period('M').dt.to_timestamp()
df['FIPS_NAME'] = df['FIPS_NAME'].str.lower().str.strip()
df['STATE'] = df['STATE'].str.upper()
df['FIPS_CODE'] = df['FIPS_CODE'].fillna(0).astype(str)
```

FB PROPHET

```
# Create a function to County-wise wildfire forecast
def forecast_wildfires_by_county(df, state_name='CA', fips_name='Los
Angeles', fips_code='037', months_ahead=12):
    # Standardize input names
    fips_name = fips_name.lower().strip()
```

```
    state_name = state_name.upper()

    # Group by COUNTY, STATE, MONTH
    grouped = df.groupby(['STATE', 'FIPS_NAME', 'FIPS_CODE',
'YEAR_MONTH']).size().reset_index(name='FIRE_COUNT')

    # Filter for specific county
    location_df = grouped[
        (grouped['STATE'] == state_name) &
        (grouped['FIPS_NAME'] == fips_name) &
        (grouped['FIPS_CODE'] == fips_code)
    ][['YEAR_MONTH', 'FIRE_COUNT']]

    if location_df.empty:
        print(f"No fire records found for {fips_name.title()},
{state_name}")
        return None, None

    prophet_df = location_df.rename(columns={'YEAR_MONTH': 'ds',
'FIRE_COUNT': 'y'})

    # Fit Prophet model
    model = Prophet(interval_width=0.95, yearly_seasonality=True,
weekly_seasonality=False, daily_seasonality=False,
changepoint_prior_scale=0.01)
    model.fit(prophet_df)

    # Forecast into the future
    future = model.make_future_dataframe(periods=months_ahead,
freq='M')
    forecast = model.predict(future)

    # Normalize forecast to generate probability-like score
    max_historical = prophet_df['y'].max()
    forecast['fire_risk_score'] = (forecast['yhat'] /
max_historical).clip(0, 1)

    # Add FIPS_CODE column to forecast results
    forecast['FIPS_CODE'] = fips_code

    return forecast[['ds', 'yhat', 'fire_risk_score', 'yhat_lower',
'yhat_upper', 'FIPS_CODE']], model

# Define a function to display the forecast message
def generate_fire_forecast_message(forecast_df, fips_name, state_name,
fips_code, target_month, target_year):

    # Filter the forecast DataFrame for the target month and year
    forecast_for_target = forecast_df[
        (forecast_df['ds'].dt.month == target_month) &
```

```
        (forecast_df['ds'].dt.year == target_year)
    ]

    if forecast_for_target.empty:
        return f"No forecast available for {fips_name.title()} in
{target_month} {target_year}."

    # Get the forecasted values (mean, lower, upper)
    forecasted_chance =
forecast_for_target['fire_risk_score'].values[0] * 100  # Convert to
percentage
    forecasted_chance_lower =
forecast_for_target['yhat_lower'].values[0] * 100
    forecasted_chance_upper =
forecast_for_target['yhat_upper'].values[0] * 100

    # Convert the numerical month to month name
    month_name = calendar.month_name[target_month]

    # Create the human-readable message
    forecast_message = (
        f"The county of {fips_name.title()} in {state_name} is
predicted to have "
        f"{forecasted_chance:.1f}% chances of fire in {month_name}
{target_year}. "
    )

    return forecast_message

forecast_df, model = forecast_wildfires_by_county(
    df,
    state_name='CA',
    fips_name='Los Angeles',
    fips_code='037',
    months_ahead=12
)

# Check the forecast for the desired target date
print(forecast_df.tail())

# Generate a message
forecast_message = generate_fire_forecast_message(
    forecast_df,
    state_name='CA',
    fips_name='Los Angeles',
    fips_code='037',
    target_month=5,
    target_year=2016
)
```

```
print(forecast_message)
```

```
DEBUG:cmdstanpy:input tempfile: /tmp/tmpp6uxxccu/w55p5z9b.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpp6uxxccu/q8qnbaym.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-
packages/prophet/stan_model/prophet_model.bin', 'random',
'seed=96568', 'data', 'file=/tmp/tmpp6uxxccu/w55p5z9b.json',
'init=/tmp/tmpp6uxxccu/q8qnbaym.json', 'output',
'file=/tmp/tmpp6uxxccu/prophet_model6qvr_oeq/prophet_model-
20250501145829.csv', 'method=optimize', 'algorithm=lbfgs',
'iter=10000']
14:58:29 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
14:58:29 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```

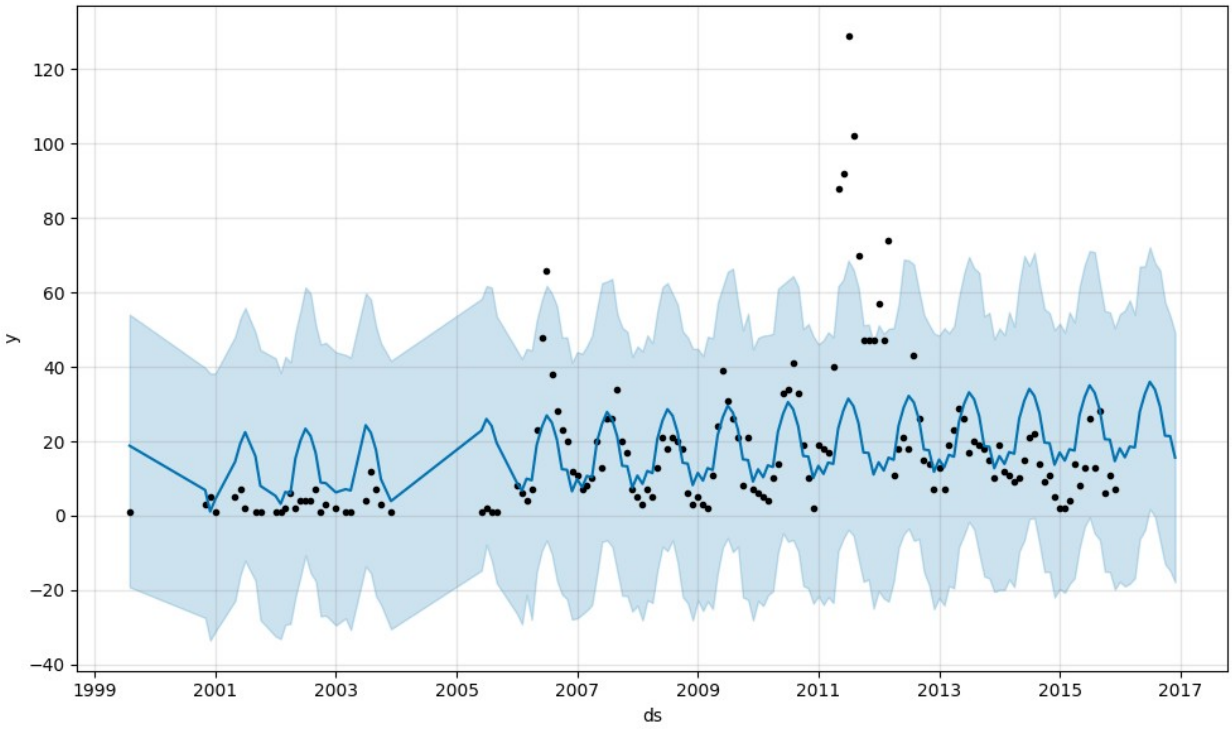
```
            ds       yhat  fire_risk_score  yhat_lower  yhat_upper
FIPS_CODE
159 2016-07-31  33.906305         0.262840   -0.230347   67.954126
037
160 2016-08-31  29.293695         0.227083   -6.789757   65.926002
037
161 2016-09-30  21.492948         0.166612  -12.990952   57.465027
037
162 2016-10-31  21.392665         0.165835  -14.818948   53.727591
037
163 2016-11-30  15.608804         0.120998  -17.925328   49.169222
037
The county of Los Angeles in CA is predicted to have 25.4% chances of
fire in May 2016.
```

```
forecast_df.head()
```


```
{"summary":"{\n  \"name\": \"forecast_df\",\n  \"rows\": 164,\n
\"fields\": [\n    {\n      \"column\": \"ds\",\n      \"properties\":
{\n        \"dtype\": \"date\",\n        \"min\": \"1999-08-01
00:00:00\",\n        \"max\": \"2016-11-30 00:00:00\",\n
\"num_unique_values\": 164,\n        \"samples\": [\n          \"2014-
08-01 00:00:00\",\n          \"2012-12-01 00:00:00\",\n
\"2014-04-01 00:00:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"yhat\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 8.037538922503298,\n        \"min\":
1.1009171176192254,\n        \"max\": 35.9884941939465,\n
\"num_unique_values\": 164,\n        \"samples\": [\n
32.18209377566296,\n          11.791074708470331,\n
16.67003182213206\n        ],\n        \"semantic_type\": \"\",\n
```

```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"fire_risk_score\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.06230650327521935,\n        \"min\":
0.00853424122185446,\n        \"max\": 0.2789805751468721,\n
\"num_unique_values\": 164,\n        \"samples\": [\n
0.249473595160178,\n          0.09140367991062272,\n
0.12922505288474465\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"yhat_lower\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 8.275937852150115,\n        \"min\": -
33.55641174949194,\n        \"max\": 1.779498940554565,\n
\"num_unique_values\": 164,\n        \"samples\": [\n          -
0.6325069411468797,\n          -25.12303817265822,\n          -
19.20345579320436\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"yhat_upper\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 8.260008868264585,\n        \"min\":
38.24275538012232,\n        \"max\": 72.19491478558018,\n
\"num_unique_values\": 164,\n        \"samples\": [\n
70.71336945553871,\n          49.009521694549555,\n
50.83815825109035\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"FIPS_CODE\",\n      \"properties\": {\n        \"dtype\":
\"object\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"037\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"forecast_df"}
```


```
# Plotting the forecast graph to vizualize
fig = model.plot(forecast_df)
```



EVALUTING FB PROPHET MODEL ACCURACY

```
# Group by STATE, COUNTY, and YEAR_MONTH to get the fire count
historical_data = df.groupby(['STATE', 'FIPS_NAME', 'FIPS_CODE',
'YEAR_MONTH']).size().reset_index(name='FIRE_COUNT')

# Clean up COUNTY and STATE names (optional)
historical_data['FIPS_NAME'] =
historical_data['FIPS_NAME'].str.lower().str.strip()
historical_data['STATE'] = historical_data['STATE'].str.upper()

# Rename columns to match those in forecast_df
historical_data.rename(columns={'YEAR_MONTH': 'ds', 'FIRE_COUNT':
'y'}, inplace=True)

# Step 3: Merge forecasted data with historical data
merged_df = pd.merge(forecast_df[['ds', 'fire_risk_score']],
historical_data[['ds', 'y']], on='ds', how='inner')

# Step 4: Actual fire count values
actual_fire_counts = merged_df['y'].values

# Predicted fire risk scores (convert to actual count by scaling it)
predicted_fire_risk = merged_df['fire_risk_score'].values *
actual_fire_counts.max()  # Assuming it's a percentage

# Step 5: Calculate performance metrics
```

```
mae = mean_absolute_error(actual_fire_counts, predicted_fire_risk)
mse = mean_squared_error(actual_fire_counts, predicted_fire_risk)
r2 = r2_score(actual_fire_counts, predicted_fire_risk)

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R²): {r2}")
```

```
Mean Absolute Error (MAE): 39.86660698317263
Mean Squared Error (MSE): 1952.286244873239
R-squared (R²): -25.22099011809751
```

MSE is abnormally high and R-squared is in negative which tells us that the model is pretty bad. It's enough to figure out that FB Prophet is not able to handle this wildfire data for forecasting. Therefore, we look for other ML models for our use case.

FEATURE ENGINEERING FOR THE OTHER MODELS

```
def remove_nan_rows(X, y):
    mask = ~np.isnan(X).any(axis=1) & ~np.isnan(y)
    return X[mask], y[mask]


# Prepare the data (make a copy of historical_data)
df_hist = historical_data.copy()

# Feature engineering: Extract year, month, and lag features
df_hist['YEAR'] = df_hist['ds'].dt.year
df_hist['MONTH'] = df_hist['ds'].dt.month

# Add previous month's fire count as a feature (lag feature)
df_hist['PREV_MONTH_FIRE_COUNT'] = df_hist.groupby(['STATE',
'FIPS_NAME', 'FIPS_CODE'])['y'].shift(1)

# Drop rows where there are missing values due to lag
df_hist = df_hist.dropna(subset=['PREV_MONTH_FIRE_COUNT'])

# One-hot encode categorical geographic features (FIPS_CODE, STATE)
# Excluding FIPS_NAME because of (1) computing resource limitations of
one-hot encoding on huge dataset (2) redundency with FIPS_CODE
encoder_fips_code = OneHotEncoder(sparse_output=False)
encoded_fips_code =
encoder_fips_code.fit_transform(df_hist[['FIPS_CODE']])

encoder_state = OneHotEncoder(sparse_output=False)
encoded_state = encoder_state.fit_transform(df_hist[['STATE']])

# Convert the encoded features to DataFrames
# df_encoded_fips_name = pd.DataFrame(encoded_fips_name,
columns=encoder_fips_name.categories_[0])
df_encoded_fips_code = pd.DataFrame(encoded_fips_code,
```

```
columns=encoder_fips_code.categories_[0])
df_encoded_state = pd.DataFrame(encoded_state,
columns=encoder_state.categories_[0])

# Combine geographic features with the original features
df_hist = pd.concat([df_hist, df_encoded_fips_code, df_encoded_state],
axis=1)

df_hist = df_hist.drop(columns=['ds','FIPS_CODE', 'FIPS_NAME',
'STATE'])

# Split data into features (X) and target variable (y)
X = df_hist.drop(columns=['y']) #+ list(df_encoded_fips_name.columns)
+ list(df_encoded_fips_code.columns) + list(df_encoded_state.columns)]
# Features
y = df_hist['y']  # Target variable (actual fire count)

# Split into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Clean X_train and y_train
X_train, y_train = remove_nan_rows(X_train, y_train)
X_test, y_test = remove_nan_rows(X_test, y_test)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Check the final shape of X_train_scaled and y_train
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
print(X_train_scaled.shape)
print(y_train.shape)
```

```
(168862, 331)
(168862,)
(42210, 331)
(42210,)
(168862, 331)
(168862,)
```

XGBOOST

```
# Check for NaN values in the features (X_train_scaled and y_train)

print("Null values in X_train_scaled?",
```

```
np.any(np.isnan(X_train_scaled)))
print("Null values in y_train?", np.any(np.isnan(y_train)))
```

```
Null values in X_train_scaled? False
Null values in y_train? False
```

```
# Train XGBoost model
xgb_model = xgb.XGBRegressor(objective='reg:squarederror',
n_estimators=100, learning_rate=0.05, max_depth=7)
xgb_model.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred_xgb = xgb_model.predict(X_test_scaled)

# Remove NaNs from y_test and y_pred_xgb
mask = ~np.isnan(y_test) & ~np.isnan(y_pred_xgb)
y_test = y_test[mask]
y_pred_xgb = y_pred_xgb[mask]

# Evaluate the XGBoost model
mae_xgb = mean_absolute_error(y_test, y_pred_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)

print(f"XGBoost Model - Mean Absolute Error (MAE): {mae_xgb}")
print(f"XGBoost Model - R-squared (R²): {r2_xgb}")
```

```
XGBoost Model - Mean Absolute Error (MAE): 3.6722097099265114
XGBoost Model - R-squared (R²): 0.3961337079941908
```

RANDOM FOREST

```
# Train Random Forest model
rf_model = RandomForestRegressor(n_estimators=100, max_depth=7,
random_state=42)
rf_model.fit(X_train_scaled, y_train)

# If y_test is a pandas Series or DataFrame
if hasattr(y_test, 'reset_index'):
    y_test_rf = y_test.reset_index(drop=True)

# Convert y_test to NumPy array if needed
y_test_rf = np.array(y_test)

# Make predictions on the test set
y_pred_rf = rf_model.predict(X_test_scaled)

# Evaluate the Random Forest model
mae_rf = mean_absolute_error(y_test_rf, y_pred_rf)
r2_rf = r2_score(y_test_rf, y_pred_rf)
```

```
print(f"Random Forest Model - Mean Absolute Error (MAE): {mae_rf}")
print(f"Random Forest Model - R-squared (R²): {r2_rf}")
```

```
Random Forest Model - Mean Absolute Error (MAE): 3.794444639890967
Random Forest Model - R-squared (R²): 0.34853452521298456
```

COMPARISON BETWEEN THE MODELS

```
# Compare both models' performance
print("Comparison of Models:")
print(f"XGBoost - MAE: {mae_xgb}, R²: {r2_xgb}")
print(f"Random Forest - MAE: {mae_rf}, R²: {r2_rf}")
```

```
Comparison of Models:
XGBoost - MAE: 3.6722097099265114, R²: 0.3961337079941908
Random Forest - MAE: 3.794444639890967, R²: 0.34853452521298456
```

The R-squared for both the models is similar while MAE for XGBoost is slightly better than that of Randbom Forest. Therefore, we choose the more accurate model - XGBOOST - to train our final model for the forecast of chances of wildfire.

PREDICTION USING XGBOOST

```
def bootstrap_predict(xgb_model, X_train, y_train, X_test,
n_iterations=50, percentile_lower=2.5, percentile_upper=97.5):
    # Store predictions from each iteration
    predictions = []

    # Bootstrapping loop
    for _ in range(n_iterations):
        # Create a bootstrap sample
        X_resampled, y_resampled = resample(X_train, y_train)

        # Train the model on the bootstrap sample
        xgb_model.fit(X_resampled, y_resampled)

        # Predict on the test set
        y_pred = xgb_model.predict(X_test)
        # Store the prediction for the target
        predictions.append(y_pred[0])

    # Calculate the confidence interval
    lower_bound = np.percentile(predictions, percentile_lower)
    upper_bound = np.percentile(predictions, percentile_upper)

    return lower_bound, upper_bound


def generate_fire_forecast_message(xgb_model, X_train, y_train,
target_month, target_year, scaler, fips_name='Los Angeles',
state_name='CA', fips_code='037'):
```

```
    # Prepare input features for the target month/year
    target_features = pd.DataFrame({
        'YEAR': [target_year],
        'MONTH': [target_month],
        'PREV_MONTH_FIRE_COUNT': [0]
    })

    # Ensure target features have the same columns as X_train (before
scaling)
    target_features = target_features.reindex(columns=X_train.columns,
fill_value=0)

    # Scale the input features
    target_features_scaled = scaler.transform(target_features)

    # Predict the fire risk score using the XGBoost model
    fire_risk_prediction = xgb_model.predict(target_features_scaled)
[0]
    fire_risk_percentage = max(fire_risk_prediction, 0)

    # Confidence interval using bootstrap (this assumes you have a
function for bootstrapping)
    lower_bound, upper_bound = bootstrap_predict(
        xgb_model, X_train, y_train, target_features_scaled
    )
    lower_bound_percentage = max(lower_bound, 0)
    upper_bound_percentage = max(upper_bound, 0)

    # Step 6: Month name based on the provided month number
    month_name = calendar.month_name[target_month]

    # Step 7: Generate the final output message
    forecast_message = (
        f"The county of {fips_name.title()} in {state_name.upper()} is
predicted to have "
        f"{fire_risk_percentage:.1f}% chances of fire in {month_name}
{target_year}. "
        f"Confidence interval: {lower_bound_percentage:.1f}% to
{upper_bound_percentage:.1f}%."
    )

    return forecast_message

# Generate a forecast with confidence interval for March 2016 in Los
Angeles
forecast_message_xgb = generate_fire_forecast_message(
    xgb_model,
    X_train,
    y_train,
    target_month=3,
```

```
    target_year=2015,
    scaler=scaler,

)

print(forecast_message_xgb)
```

PREDICTION TOOL

```
# Create an interactive interface for user input
def get_fire_forecast(xgb_model, X_train_scaled, y_train, scaler):
    print("Please input the following parameters to generate the
wildfire forecast:")

    # Getting user input
    target_month = int(input("Enter the target month (1-12): "))
    target_year = int(input("Enter the target year (YYYY): "))
    fips_name = input("Enter the FIPS name (e.g., 'Los Angeles'): ")
    fips_code = input("Enter the FIPS code (e.g., '037'): ")
    state_name = input("Enter the state name (e.g., 'CA'): ")

    xgb_model.fit(X_train_scaled, y_train)

    # Generate the forecast message
    forecast_message = generate_fire_forecast_message(
        xgb_model=xgb_model,
        X_train=X_train,
        y_train=y_train,
        target_month=target_month,
        target_year=target_year,
        scaler=scaler,
        fips_name=fips_name,
        fips_code=fips_code,
        state_name=state_name
    )

    # Output the forecast message
    print(forecast_message)

# Getting the prediction
get_fire_forecast(xgb_model, X_train, y_train, scaler)
```

IMPROVING THE MODEL - HYPERPARAMETER TUNING

```
# XGBoost Hyperparameter Tuning
xgb_param_grid = {
    'learning_rate': [0.01, 0.03, 0.05, 0.1],
    'max_depth': [3, 5, 7, 9],
    'n_estimators': [100, 500, 1000],
    'subsample': [0.6, 0.7, 0.8, 0.9],
```

```
    'colsample_bytree': [0.6, 0.7, 0.8, 1]
}

xgb_model = xgb.XGBRegressor(objective='reg:squarederror',
random_state=42)

# Initialize RandomizedSearchCV
xgb_random_search = RandomizedSearchCV(
    xgb_model,
    param_distributions=xgb_param_grid,
    n_iter=10,  # Number of random combinations to try
    scoring='neg_mean_absolute_error',
    cv=3,  # 3-fold cross-validation
    verbose=2,
    random_state=42,
    n_jobs=-1
)

# Fit the XGBoost model with RandomizedSearchCV (no log transformation
used)
xgb_random_search.fit(X_train_scaled, y_train)

# Print the best hyperparameters found
print("Best Parameters for XGBoost:", xgb_random_search.best_params_)

# Get the best XGBoost model after hyperparameter tuning
best_xgb_model = xgb_random_search.best_estimator_

# Evaluate the tuned model on the test set (no log transformation)
y_pred_xgb = best_xgb_model.predict(X_test_scaled)

# Print evaluation metrics
print("XGBoost (Tuned):")
print(f"MAE: {mean_absolute_error(y_test, y_pred_xgb):.2f}, R²:
{r2_score(y_test, y_pred_xgb):.2f}")
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Best Parameters for XGBoost: {'subsample': 0.6, 'n_estimators': 500,
'max_depth': 9, 'learning_rate': 0.1, 'colsample_bytree': 0.8}
XGBoost (Tuned):
MAE: 3.37, R²: 0.48
```

The values of the hyperparameter obtained as the output give us the best result for accuracy that can be obtained with the available data and the trained model. Therefore, we shall go ahead with using these tuned hyperparameters to enhance our model.

FINAL MODEL USING XGBOOST

```
# Retrain XGBoost model
xgb_model = xgb.XGBRegressor(
```

```
    objective='reg:squarederror',
    n_estimators=500,
    max_depth=9,
    learning_rate=0.1,
    subsample=0.6,
    colsample_bytree=0.8
)
xgb_model.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred_xgb = xgb_model.predict(X_test_scaled)

# Evaluate the XGBoost model
mae_xgb = mean_absolute_error(y_test, y_pred_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)

print(f"XGBoost Model - Mean Absolute Error (MAE): {mae_xgb}")
print(f"XGBoost Model - R-squared (R²): {r2_xgb}")
```

```
XGBoost Model - Mean Absolute Error (MAE): 3.36939201175745
XGBoost Model - R-squared (R²): 0.4746585478406169
```

```
# Getting the prediction
get_fire_forecast(xgb_model, X_train_scaled, y_train, scaler)
```

```
Please input the following parameters to generate the wildfire
forecast:
Enter the target month (1-12): 3
Enter the target year (YYYY): 2016
Enter the FIPS name (e.g., 'Los Angeles'): Los Angeles
Enter the FIPS code (e.g., '037'): 037
Enter the state name (e.g., 'CA'): CA
The county of Los Angeles in CA is predicted to have 1.0% chances of
fire in March 2016. Confidence interval: 1.0% to 1.1%.
```

Our fine-tuned final model explains varinace better than the original model as evidenced by the increase in R-squared.

The prediction accuracy of the occurance of wildfire is low given the random nature of the event itself. Wildfires, unlike some other events or phenonmenon, do not contain seasonlity or any fixed pattern. The lack of these and other contributing factors such as natural conditons, weather conditions, etc. makes it quite challenging for the model to be very precise in prediction.

The model has tendency to be imporved with introduction of other geological factors into the mix of data such as climate, weather conditons, wind speed, moisture, etc.

2. PREDICTING THE CAUSE OF THE WILDFIRE

The Objective is to predict the the cause of the wildfire based on the size, location and date of the ocurance.

FEATURE ENGINEERING AND EDA

```
# Arrange the dates for the Model Data Frame.
df_model =
df[["FIRE_YEAR","DISCOVERY_DOY","DISCOVERY_TIME","DISCOVERY_DATE","CON
T_DOY","CONT_TIME","STAT_CAUSE_CODE","STAT_CAUSE_DESCR","FIRE_SIZE","L
ATITUDE","LONGITUDE","STATE","FIPS_NAME"]]
df_model = df_model.drop_duplicates()
df_model["combined_date_dis"] = df_model["FIRE_YEAR"]*1000 +
df_model["DISCOVERY_DOY"]
df_model["combined_date_dis"] =
pd.to_datetime(df_model["combined_date_dis"], format = "%Y%j")
df_model["combined_date_dis"] = df_model.combined_date_dis.astype(str)
+ " " + df_model.DISCOVERY_TIME.str[:2] + ":" +
df_model.DISCOVERY_TIME.str[2:]
df_model["combined_date_dis"] =
pd.to_datetime(df_model["combined_date_dis"])
df_model["combined_date_con"] = df_model["FIRE_YEAR"]*1000 +
df_model["CONT_DOY"]
df_model["combined_date_con"] =
pd.to_datetime(df_model["combined_date_con"], format = "%Y
%j",errors="ignore")
df_model["combined_date_con"] = df_model.combined_date_con.astype(str)
+ " " + df_model.CONT_TIME.str[:2] + ":" + df_model.CONT_TIME.str[2:]
df_model["combined_date_con"] =
pd.to_datetime(df_model["combined_date_con"],errors="coerce")
# Calculate the durations of the fire from substraction of discovery
and control time.
df_model["duration"] = None
df_model.loc[df_model['combined_date_con'].notna(), 'duration'] =
(df_model["combined_date_con"] -
df_model["combined_date_dis"]).dt.seconds/60
# Extract week day and the month for each fire.
df_model["week_day"] = df_model.combined_date_dis.dt.weekday
df_model["month"] = df_model.combined_date_dis.dt.month
df_model = df_model.set_index("combined_date_dis")
df_model["duration"] = df_model.duration.astype(float)
df_model.head()
```

{"type":"dataframe","variable_name":"df_model"}

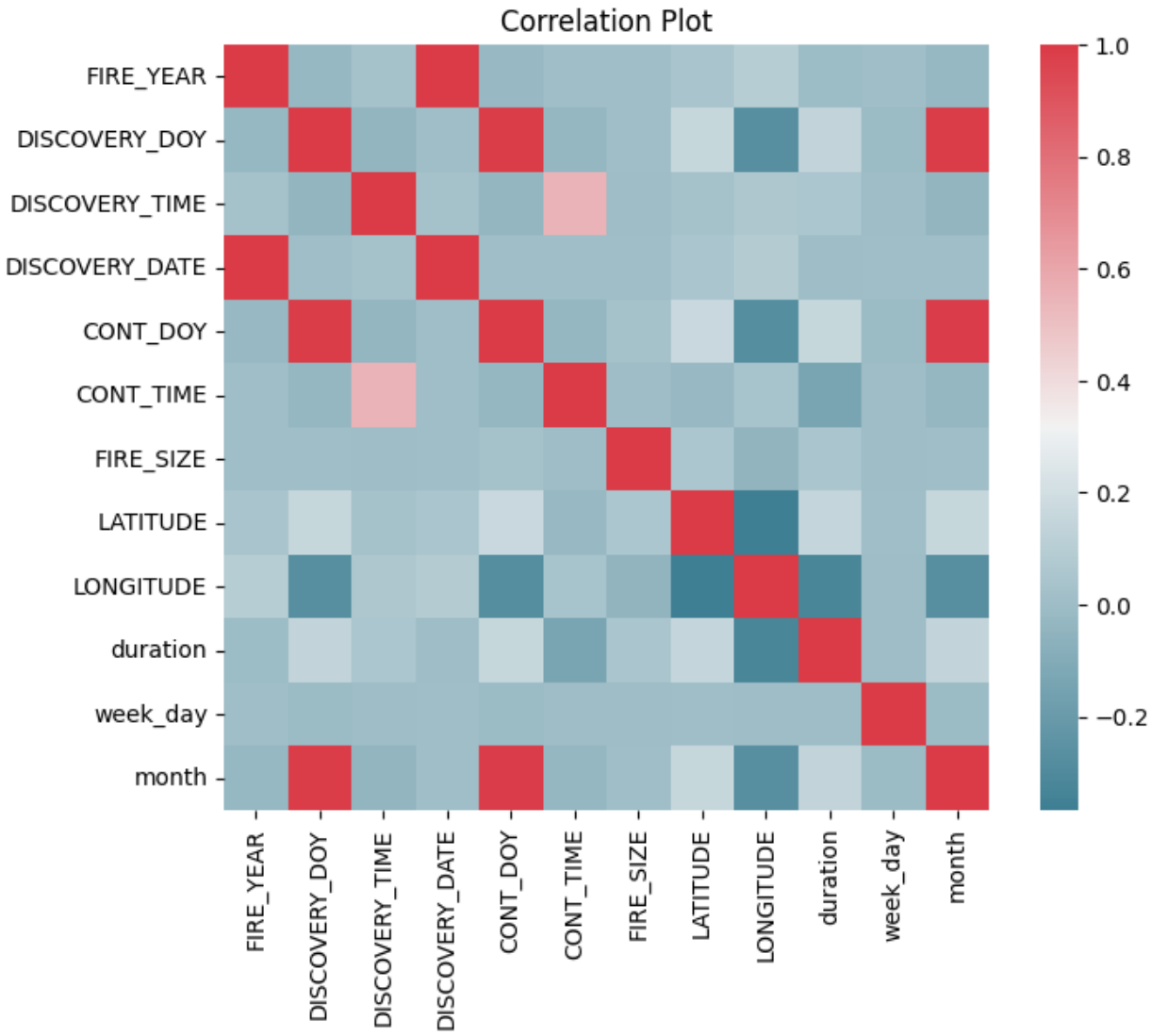
```
# Calculate the correlations among numerical features.
cat=['STATE','FIPS_NAME',"week_day","month"]
num=['FIRE_YEAR',"DISCOVERY_DOY","DISCOVERY_TIME","DISCOVERY_DATE","CO
```

```
NT_DOY","CONT_TIME","FIRE_SIZE","LATITUDE","LONGITUDE","duration","wee
k_day","month"]
corr_df=df_model[num]
corr_df = corr_df.dropna()
cor= corr_df.corr(method='pearson')
cor
```


```
{"summary":"{\n  \"name\": \"cor\",\n  \"rows\": 12,\n  \"fields\": [\
n    {\n      \"column\": \"FIRE_YEAR\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.3868536462802191,\n
\"min\": -0.027384075536906798,\n        \"max\": 1.0,\n
\"num_unique_values\": 12,\n        \"samples\": [\n
0.004076944038354567,\n          -0.003943235823643214,\n
1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DISCOVERY_DOY\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.4667521392842791,\n        \"min\": -
0.2729942814229992,\n        \"max\": 1.0,\n
\"num_unique_values\": 12,\n        \"samples\": [\n          -
0.013973249801319443,\n          0.1388080568617648,\n          -
0.027384075536906798\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DISCOVERY_TIME\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.3161570837837698,\n        \"min\": -
0.04230651757357552,\n        \"max\": 1.0,\n
\"num_unique_values\": 12,\n        \"samples\": [\n          -
0.0005429507019353256,\n          0.0515258435754646,\n
0.024319647632995132\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DISCOVERY_DATE\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.3826086187731824,\n        \"min\":
0.0004331979354716844,\n        \"max\": 1.0,\n
\"num_unique_values\": 12,\n        \"samples\": [\n
0.0036380434521680735,\n          0.0004331979354716844,\n
0.9995025065223067\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"CONT_DOY\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.464913722124301,\n        \"min\": -
0.27932855957800273,\n        \"max\": 1.0,\n
\"num_unique_values\": 12,\n        \"samples\": [\n          -
0.014162055000380175,\n          0.1535076285982061,\n          -
0.021693756017871366\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"CONT_TIME\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.3274666878240334,\n        \"min\": -
0.14046166263074766,\n        \"max\": 1.0,\n
\"num_unique_values\": 12,\n        \"samples\": [\n          -
0.0024077548579122207,\n          -0.14046166263074766,\n
0.0034446299271288765\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

\"FIRE_SIZE\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0.28727321175351417,\n   \"min\": -
0.04943012714345302,\n   \"max\": 1.0,\n
\"num_unique_values\": 12,\n   \"samples\": [\n   -
0.0006719053933477057,\n   0.04793622347250535,\n
0.006089277000835132\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"LATITUDE\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0.31249372723929747,\n   \"min\": -
0.3668604962362831,\n   \"max\": 1.0,\n
\"num_unique_values\": 12,\n   \"samples\": [\n
0.0069447014425334345,\n   0.1510510790102699,\n
0.04417987706427377\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"LONGITUDE\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0.36750684069155487,\n   \"min\": -
0.3668604962362831,\n   \"max\": 1.0,\n
\"num_unique_values\": 12,\n   \"samples\": [\n   -
0.00034288618992998963,\n   -0.3273525312922132,\n
0.09383382591169594\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"duration\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0.3155389594983574,\n   \"min\": -
0.3273525312922132,\n   \"max\": 1.0,\n
\"num_unique_values\": 12,\n   \"samples\": [\n   -
0.002571991924960465,\n   1.0,\n   -
0.003943235823643214\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"week_day\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0.28965435962288894,\n   \"min\": -
0.014162055000380175,\n   \"max\": 1.0,\n
\"num_unique_values\": 12,\n   \"samples\": [\n   1.0,\n
-0.002571991924960465,\n   0.004076944038354567\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"month\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\":
0.46582018675977976,\n   \"min\": -0.2732386526308551,\n
\"max\": 1.0,\n   \"num_unique_values\": 12,\n
\"samples\": [\n   -0.013861498915287O9,\n
0.1386193901388841,\n   -0.02632920277259484\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"cor"}

```
# Visualize the pair-wise correlations.
fig, ax =plt.subplots(figsize=(8, 6))
plt.title("Correlation Plot")
sns.heatmap(cor, mask=np.zeros_like(cor, dtype=np.bool),
cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True, ax=ax)
plt.show()
```

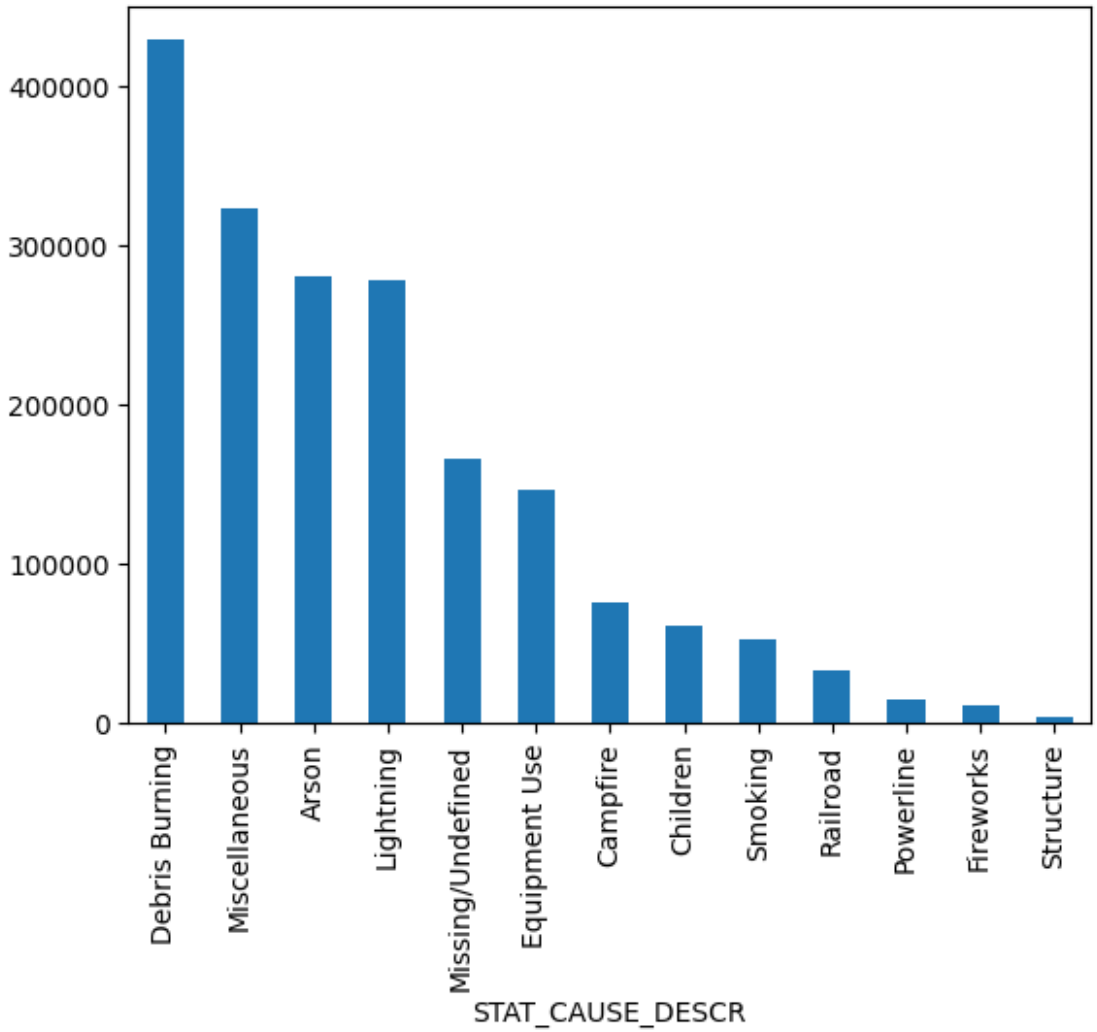


Very strong correlations are exist between DISCOVERY_DOY,CONT_DOY, DISCOVERY_DATE,FIRE_YEAR. In order to prevent data redundancy, DISCOVERY_DATE and CONT_DOY can be removed from the training data.

```
# Drop DISCOVERY_DATE, DISCOVERY_DOY and CONT_DOY
df_model =
df_model.drop(["DISCOVERY_DATE","CONT_DOY","DISCOVERY_DOY"],axis=1)

# Check if the data is balanced in terms of the STAT_CAUSE_DESCR
labels.
df_model["STAT_CAUSE_DESCR"].value_counts().plot.bar()
```

```
<Axes: xlabel='STAT_CAUSE_DESCR'>
```




```
# Calculate the rate of the category that has highest counts.
(df_model.query('STAT_CAUSE_DESCR == "Debris
Burning"').count().STAT_CAUSE_DESCR /
df_model.count().STAT_CAUSE_DESCR ) * 100
```

```
np.float64(22.85339206729258)
```

Some over or under sampling techniques are required to balance the data since it is unbalanced.

Using label encoding (instead of One-Hot_Endcoing due to memory consumption) on STATE and COUNTY.

```
df_model.head()
```

```
{"type":"dataframe","variable_name":"df_model"}
```

```
# Select the numeric features for the classification model.
df_model =
```

```
df_model[["FIRE_YEAR","STAT_CAUSE_DESCR","FIRE_SIZE","LATITUDE","LONGI
TUDE","FIPS_NAME","STATE","duration","week_day","month"]]

# Implement Label Encoding for the State and Fips Name.
fips_le = LabelEncoder()
fips_labels = fips_le.fit_transform(df_model.FIPS_NAME)
df_model["FIPS_NAME"] = fips_labels

state_le = LabelEncoder()
state_labels = state_le.fit_transform(df_model.STATE)
df_model["STATE"] = state_labels

# Instead of using STAT_CAUSE_CODE as label, implement LE for
STAT_CAUSE_DESCR because STAT_CAUSE_CODE starts from 1.
le = LabelEncoder()
df_model["STAT_CAUSE_DESCR"] =
le.fit_transform(df_model['STAT_CAUSE_DESCR'])
df_model["STAT_CAUSE_DESCR"] = df_model.STAT_CAUSE_DESCR.astype(int)
```

ML Modeling

```
# Data will be split into test and train for being used on the model.
X = df_model.drop(['STAT_CAUSE_DESCR'], axis=1).values
y = df_model['STAT_CAUSE_DESCR'].values
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=0.3,
random_state=0)

# XG Boost with na values in the dataset.(1313835 rows)
clf_xgboost = xgb.XGBClassifier(objective="multi:softprob",
random_state=42)
clf_xgboost.fit(X_train, Y_train)
predicted = clf_xgboost.predict(X_test)
print("Accuracy rate: ",np.mean(predicted == Y_test))
```

```
Accuracy rate:  0.5641522904301682
```

```
# XG Boost without na values in the dataset.(890796 rows)

# Drop the rows that has na.
df_model = df_model.dropna()

# Recreate test and train sets.
X = df_model.drop(['STAT_CAUSE_DESCR'], axis=1).values
y = df_model['STAT_CAUSE_DESCR'].values
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=0.3,
random_state=0)

clf_xgboost = xgb.XGBClassifier(objective="multi:softprob",
random_state=42)
clf_xgboost.fit(X_train, Y_train)
```

```
predicted = clf_xgboost.predict(X_test)
print("Accuracy rate: ",np.mean(predicted == Y_test))
```

```
Accuracy rate:  0.599461923007843
```

```
# Random Forest
clf_rf = RandomForestClassifier(n_estimators=100)
clf_rf.fit(X_train,Y_train)
predicted = clf_rf.predict(X_test)
print("Accuracy rate: ",np.mean(predicted == Y_test))
```

The CONT_TIME feature has approximately 800k nan values. Since, Xgboost is capable of handling nan values (unlike Random Forest), the data is implemented on Xgboost using both - nan present and absent. Resultant accuracies are as follows:

- Random forest 0.62
- Xgboost 0.60 without na (890796 rows)
- Xgboost 0.56 with na (1313835 rows)

Low accuracy on Xgboost even with higher amount of data shows that there is an overfitting on the model. Hence Random Forest is chosen for final model.

The following classification models are implemented and the resultant accuracies are recorded:

- Random Forest : 0.62
- Xgboost Classifier : 0.60
- DecisionTreeClassifier : 0.50
- KNeighbors Classifier : 0.47
- Logistic Regression : 0.35
- Linear SVC : 0.25
- SGD Classifier : 0.17

This results show that the decision tree classifiers and simplistic models (like Logistic Regression, Linear SVC) are not as good as the ensemble methods such as Random Forest and Xgboost classifiers.

Comparison of the F1 Scores of each label shows that Random Forest performs worse on the following labels.

Children, Miscellaneous, Powerline, Railroad, Smoking

```
print("\nClassification Report\n")
print(classification_report(Y_test, predicted))

# Least performers: 2,7,9,10,11
le.inverse_transform([2,7,9,10,11])
```

Since the number of labels in the dependent variable is high, it is unlikely to obtain higher accuracy rates. In order to decrease the number of the labels, we can ramove Miscellaneous and Missing/Undefined. These causes can be set in case of fires that have variety of values in their

features, which may consider as noise for our model. If the dataset is noisy, then the complex models can fit with noise. And this situation results with low accuracy rates.

```
le.inverse_transform([7,8])

# Random Forest

# Drop the rows of Causes Miscellaneous and
print(f"Dropped labels: {le.inverse_transform([7,8])}")
df_model = df_model[~df_model.STAT_CAUSE_DESCR.isin([7,8])]

# Recreate test and train sets.
X = df_model.drop(['STAT_CAUSE_DESCR'], axis=1).values
y = df_model['STAT_CAUSE_DESCR'].values
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=0.3,
random_state=0)

print(dt.now())
clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train,Y_train)
predicted = clf.predict(X_test)
print(dt.datetime.now())

print("Accuracy rate: ",np.mean(predicted == Y_test))
```

Removing Miscellaneous and Missing/Undefined labels increased our accuracy to 0.66. Grouping the labels, or even reduce the number of labels to one(transforming the problem into a binary classification problem.) may increase the accuracy to higher rates.

As discussed in the EDA section, the classes are highly inbalanced. To balance the data with SMOTE, can be a good option to increase the accuracy. After the following implemetation, it is shown in the graph that all labels become in the same rates.

```
X = df_model.drop(['STAT_CAUSE_DESCR'], axis=1).values
y = df_model['STAT_CAUSE_DESCR'].values

oversample = SMOTE()
X, y = oversample.fit_resample(X, y)

dfx = pd.DataFrame(y)
dfx.value_counts().plot.bar()

# Random Forest
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=0.3,
random_state=0)

clf_rf = RandomForestClassifier(n_estimators=100)
clf_rf.fit(X_train,Y_train)
predicted = clf_rf.predict(X_test)
print("Accuracy rate: ",np.mean(predicted == Y_test))
```

After the implementation of SMOTE, with a balanced data the accuracy is increased to 0.83 which is a significant improve.

FINAL MODEL USING RANDOM FOREST

```
def get_wildfire_cause():
    # Get user input for necessary parameters
    print("Please input the following parameters to predict the
wildfire cause:")

    fire_size = float(input("Fire Size (in acres): "))
    latitude = float(input("Latitude: "))
    longitude = float(input("Longitude: "))
    state = input("State: ")  # User input as string, needs label
encoding
    fips_name = input("FIPS Name: ")  # User input as string, needs
label encoding
    duration = float(input("Fire Duration (in minutes): "))

    # Get the current date info (for week_day and month)
    date_str = input("Enter the discovery date (YYYY-MM-DD): ")
    discovery_date = dt.strptime(date_str, "%Y-%m-%d")
    week_day = discovery_date.weekday()
    month = discovery_date.month

    # Prepare the input data
    user_input = pd.DataFrame({
        "FIRE_SIZE": [fire_size],
        "LATITUDE": [latitude],
        "LONGITUDE": [longitude],
        "STATE": [state],
        "FIPS_NAME": [fips_name],
        "duration": [duration],
        "week_day": [week_day],
        "month": [month]
    })

    # Apply label encoding to categorical data
    user_input["STATE"] = state_le.transform(user_input["STATE"])
    user_input["FIPS_NAME"] =
fips_le.transform(user_input["FIPS_NAME"])

    # Convert the user input into the format that can be passed to the
model
    X_input = user_input.values

    # Predict the cause using the model
    predicted_cause = clf_rf.predict(X_input)

    # Decode the prediction
    predicted_cause_label =
```

```
stat_cause_le.inverse_transform(predicted_cause)

    print(f"The predicted cause of the wildfire is:
{predicted_cause_label[0]}")
```

PREDICTION TOOL

```
get_wildfire_cause()

# Sample Input

Fire Size (in acres): 250
Latitude: 36.7783
Longitude: -119.4179
State: California
FIPS Name: 06077
Fire Duration (in minutes): 180
Enter the discovery date (YYYY-MM-DD): 2024-06-15
```

With a high accuracy of 83 percentage, the model is performing quite well. There are still ways that can be taken to improve the model further to give a more precise and robust model such as increasing the size of data further more.

```

```