DATA245_ForestFireSizePrediction_GRP7

November 13, 2023

1 Forest Fire Size Prediction

1.0.1 Importing all the required libraries.

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
import category_encoders as ce
```

1.0.2 Reading the CSV files

[3]:
```
df = pd.read_csv("data/FW_Veg_Rem_Combined.csv")
df.head(5)
```

[3]:
```
   Unnamed: 0.1  Unnamed: 0 fire_name  fire_size fire_size_class  \
0             0           0       NaN       10.0               C
1             1           1       NaN        3.0               B
2             2           2       NaN       60.0               C
3             3           3     WNA 1        1.0               B
4             4           4       NaN        2.0               B

     stat_cause_descr   latitude   longitude state disc_clean_date  …  \
0  Missing/Undefined  18.105072  -66.753044    PR       2/11/2007  …
1              Arson  35.038330  -87.610000    TN      12/11/2006  …
```

```
2              Arson  34.947800  -88.722500    MS       2/29/2004  …
3     Debris Burning  39.641400 -119.308300    NV        6/6/2005  …
4      Miscellaneous  30.700600  -90.591400    LA       9/22/1999  …

  Wind_cont Hum_pre_30 Hum_pre_15  Hum_pre_7   Hum_cont Prec_pre_30  \
0  3.250413  78.216590  76.793750  76.381579  78.724370         0.0
1  2.122320  70.840000  65.858911  55.505882  81.682678        59.8
2  3.369050  75.531629  75.868613  76.812834  65.063800       168.8
3  0.000000  44.778429  37.140811  35.353846   0.000000        10.4
4 -1.000000  -1.000000  -1.000000  -1.000000  -1.000000        -1.0

   Prec_pre_15 Prec_pre_7 Prec_cont  remoteness
0          0.0        0.0       0.0    0.017923
1          8.4        0.0      86.8    0.184355
2         42.2       18.1     124.5    0.194544
3          7.2        0.0       0.0    0.487447
4         -1.0       -1.0      -1.0    0.214633

[5 rows x 43 columns]
```

[4]:
```python
df.describe(include = "all")
```

[4]:
```
        Unnamed: 0.1    Unnamed: 0   fire_name      fire_size fire_size_class  \
count   55367.000000  55367.000000       25913   55367.000000           55367
unique           NaN           NaN       21793            NaN               6
top              NaN           NaN  GRASS FIRE            NaN               B
freq             NaN           NaN         128            NaN           36522
mean    27683.000000  27683.000000         NaN    2104.645161             NaN
std     15983.220514  15983.220514         NaN   14777.005364             NaN
min         0.000000      0.000000         NaN       0.510000             NaN
25%     13841.500000  13841.500000         NaN       1.200000             NaN
50%     27683.000000  27683.000000         NaN       4.000000             NaN
75%     41524.500000  41524.500000         NaN      20.000000             NaN
max     55366.000000  55366.000000         NaN  606945.000000             NaN

       stat_cause_descr      latitude     longitude  state disc_clean_date  \
count             55367  55367.000000  55367.000000  55367           55367
unique               13           NaN           NaN     51            8114
top      Debris Burning           NaN           NaN     TX       6/21/2008
freq              14278           NaN           NaN   6080              64
mean                NaN     36.172866    -94.757971    NaN             NaN
std                 NaN      6.724348     15.878194    NaN             NaN
min                 NaN     17.956533   -165.936000    NaN             NaN
25%                 NaN     32.265960   -102.541513    NaN             NaN
50%                 NaN     34.600000    -91.212359    NaN             NaN
75%                 NaN     38.975235    -82.847500    NaN             NaN
max                 NaN     69.849500    -65.285833    NaN             NaN
```

```
        …     Wind_cont    Hum_pre_30    Hum_pre_15     Hum_pre_7  \
count   …  55367.000000  55367.000000  55367.000000  55367.000000
unique  …           NaN           NaN           NaN           NaN
top     …           NaN           NaN           NaN           NaN
freq    …           NaN           NaN           NaN           NaN
mean    …      1.132284     40.781796     38.453935     37.001865
std     …      2.030611     31.086856     31.042541     30.827885
min     …     -1.000000     -1.000000     -1.000000     -1.000000
25%     …     -1.000000     -1.000000     -1.000000     -1.000000
50%     …      0.000000     55.657480     51.753846     48.230769
75%     …      2.848603     67.384352     65.911469     64.645296
max     …     24.200000     96.000000     94.000000     96.000000

           Hum_cont   Prec_pre_30   Prec_pre_15    Prec_pre_7     Prec_cont  \
count  55367.000000  55367.000000  55367.000000  55367.000000  55367.000000
unique          NaN           NaN           NaN           NaN           NaN
top             NaN           NaN           NaN           NaN           NaN
freq            NaN           NaN           NaN           NaN           NaN
mean      25.056738     26.277046     11.654253      4.689920     15.590440
std       31.187638    112.050198     56.920510     31.205327     59.757113
min       -1.000000     -1.000000     -1.000000     -1.000000     -1.000000
25%       -1.000000     -1.000000     -1.000000     -1.000000     -1.000000
50%        0.000000      0.000000      0.000000      0.000000      0.000000
75%       60.193606     18.900000      3.600000      0.000000      0.000000
max       94.000000  13560.800000   2527.000000   1638.000000   2126.000000

         remoteness
count  55367.000000
unique          NaN
top             NaN
freq            NaN
mean       0.236799
std        0.144865
min        0.000000
25%        0.137800
50%        0.202114
75%        0.284782
max        1.000000

[11 rows x 43 columns]
```

[5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55367 entries, 0 to 55366
Data columns (total 43 columns):
```

```
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Unnamed: 0.1      55367 non-null  int64
 1   Unnamed: 0        55367 non-null  int64
 2   fire_name         25913 non-null  object
 3   fire_size         55367 non-null  float64
 4   fire_size_class   55367 non-null  object
 5   stat_cause_descr  55367 non-null  object
 6   latitude          55367 non-null  float64
 7   longitude         55367 non-null  float64
 8   state             55367 non-null  object
 9   disc_clean_date   55367 non-null  object
 10  cont_clean_date   27477 non-null  object
 11  discovery_month   55367 non-null  object
 12  disc_date_final   28708 non-null  object
 13  cont_date_final   25632 non-null  object
 14  putout_time       27477 non-null  object
 15  disc_date_pre     55367 non-null  object
 16  disc_pre_year     55367 non-null  int64
 17  disc_pre_month    55367 non-null  object
 18  wstation_usaf     55367 non-null  object
 19  dstation_m        55367 non-null  float64
 20  wstation_wban     55367 non-null  int64
 21  wstation_byear    55367 non-null  int64
 22  wstation_eyear    55367 non-null  int64
 23  Vegetation        55367 non-null  int64
 24  fire_mag          55367 non-null  float64
 25  weather_file      55367 non-null  object
 26  Temp_pre_30       55367 non-null  float64
 27  Temp_pre_15       55367 non-null  float64
 28  Temp_pre_7        55367 non-null  float64
 29  Temp_cont         55367 non-null  float64
 30  Wind_pre_30       55367 non-null  float64
 31  Wind_pre_15       55367 non-null  float64
 32  Wind_pre_7        55367 non-null  float64
 33  Wind_cont         55367 non-null  float64
 34  Hum_pre_30        55367 non-null  float64
 35  Hum_pre_15        55367 non-null  float64
 36  Hum_pre_7         55367 non-null  float64
 37  Hum_cont          55367 non-null  float64
 38  Prec_pre_30       55367 non-null  float64
 39  Prec_pre_15       55367 non-null  float64
 40  Prec_pre_7        55367 non-null  float64
 41  Prec_cont         55367 non-null  float64
 42  remoteness        55367 non-null  float64
dtypes: float64(22), int64(7), object(14)
memory usage: 18.2+ MB
```

Data Dictionary

- Fire_size_class - Class of Fire Size (A-G)
- Stat_cause_descr - Cause of Fire
- Latitude - Latitude of Fire
- Longitude - Longitude of Fire
- Discovery_month - Month in which Fire was discovered
- Vegetation - Dominant vegetation in the areas (can save some factors of vegetation)
- Temp_pre - temperature in deg C at the location of fire up to 30, 15 and 7 days prior
- Temp_cont - temperature in deg C at the location of fire up to day the fire was
- Wind_pre - wind in deg C at the location of fire up to 30, 15 and 7 days prior
- Wind_cont - wind in deg C at the location of fire up to day the fire was
- Prec_pre - Precipitation in deg C at the location of fire up to 30, 15 and 7 days prior
- Prec_cont - Precipitation in deg C at the location of fire up to day the fire was
- Hum_pre - Humidity in deg C at the location of fire up to 30, 15 and 7 days prior
- Hum_cont - Humidity in deg C at the location of fire up to day the fire was
- Remoteness - non-dimensional distance to closest city

1.0.3 Removing the redundant and unnecessary columns

[6]:
```
df = df.drop(['Unnamed: 0.1', 'Unnamed: 0', 'fire_name', 'state',␣
↪'cont_clean_date',
       'discovery_month', 'disc_date_final', 'cont_date_final',␣
↪'putout_time', 'disc_pre_year', 'disc_pre_month',
       'wstation_usaf', 'dstation_m', 'wstation_wban', 'fire_mag',␣
↪'weather_file'],axis=1)
```

Removed columns with null values, redundant columns like fire_mag, fire_size and date variables

[7]:
```
df.head(5)
```

[7]:
```
   fire_size fire_size_class   stat_cause_descr   latitude   longitude  \
0       10.0               C  Missing/Undefined  18.105072  -66.753044
1        3.0               B              Arson  35.038330  -87.610000
2       60.0               C              Arson  34.947800  -88.722500
3        1.0               B     Debris Burning  39.641400 -119.308300
4        2.0               B      Miscellaneous  30.700600  -90.591400

  disc_clean_date disc_date_pre  wstation_byear  wstation_eyear  Vegetation  \
0       2/11/2007     1/12/2007            1945            2018          12
1      12/11/2006    11/11/2006            1978            2020          15
2       2/29/2004     1/30/2004            1978            2020          16
3        6/6/2005      5/7/2005            1942            2020           0
4       9/22/1999     8/23/1999            1987            2016          12

   …  Wind_cont  Hum_pre_30  Hum_pre_15  Hum_pre_7   Hum_cont  Prec_pre_30  \
0  …   3.250413   78.216590   76.793750  76.381579  78.724370          0.0
```

```
1  …   2.122320   70.840000   65.858911  55.505882  81.682678          59.8
2  …   3.369050   75.531629   75.868613  76.812834  65.063800         168.8
3  …   0.000000   44.778429   37.140811  35.353846   0.000000          10.4
4  …  -1.000000   -1.000000   -1.000000  -1.000000  -1.000000          -1.0

   Prec_pre_15  Prec_pre_7  Prec_cont  remoteness
0          0.0         0.0        0.0    0.017923
1          8.4         0.0       86.8    0.184355
2         42.2        18.1      124.5    0.194544
3          7.2         0.0        0.0    0.487447
4         -1.0        -1.0       -1.0    0.214633

[5 rows x 27 columns]
```

1.0.4 Data Visualization

[8]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import folium
from folium.plugins import MarkerCluster, HeatMap
from folium import Choropleth
```

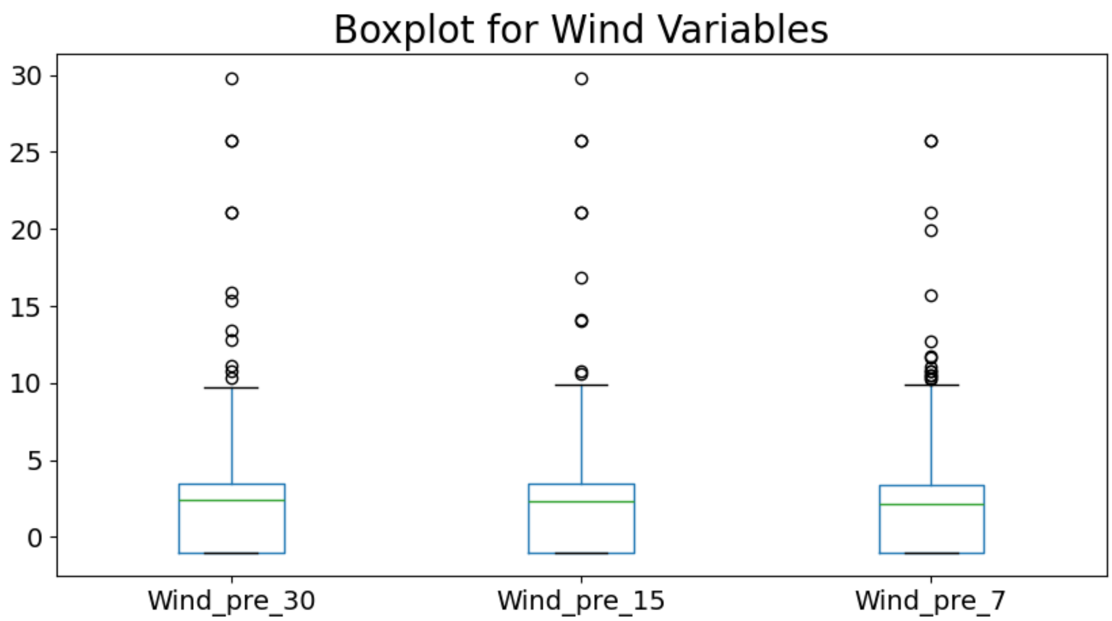
[9]:
```
# Create a base map centered at a specific location
map_center = [df['latitude'].mean(), df['longitude'].mean()]
mymap = folium.Map(location=map_center, zoom_start=4)

# Create a MarkerCluster to group markers at the same location
marker_cluster = MarkerCluster().add_to(mymap)

# Add markers for each data point
for index, row in df.iterrows():
    folium.Marker(
        location=[row['latitude'], row['longitude']],
        popup=f"Fire Size Class: {row['fire_size_class']}",
        icon=None,  # You can customize the icon if needed
    ).add_to(marker_cluster)

# Display the map in the notebook
display(mymap)
```

```
<folium.folium.Map at 0x7fbeb9183220>
```

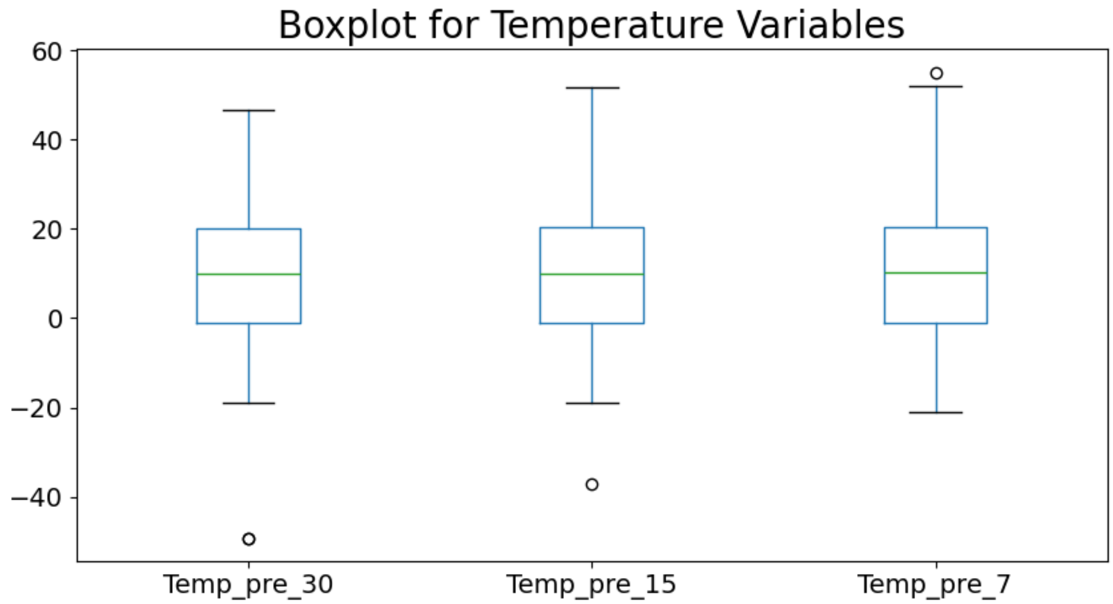
The dataset has fire occurances data from all over US, concentrated in california and Southeastern part.

[10]:
```
plt.figure(figsize=(10, 5))
plt.title("Boxplot for Wind Variables", fontsize=20)
plt.grid(color='grey', linestyle='-', linewidth=0.25, alpha=0.5)
```

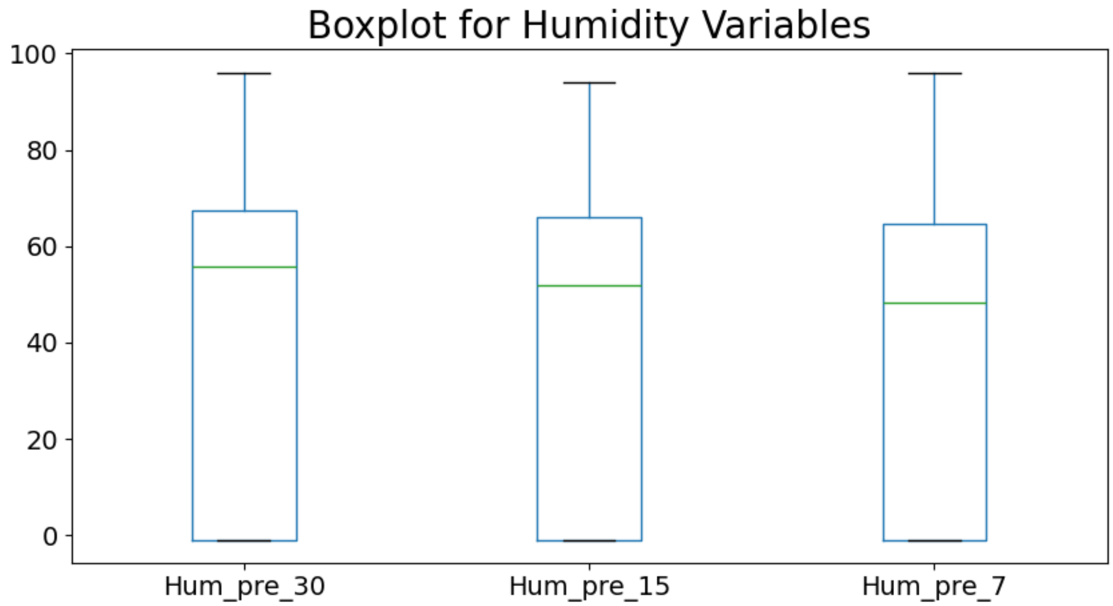
```
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

# Create boxplot
df.boxplot(column=["Wind_pre_30", "Wind_pre_15", "Wind_pre_7"], grid=False)

plt.show()
```



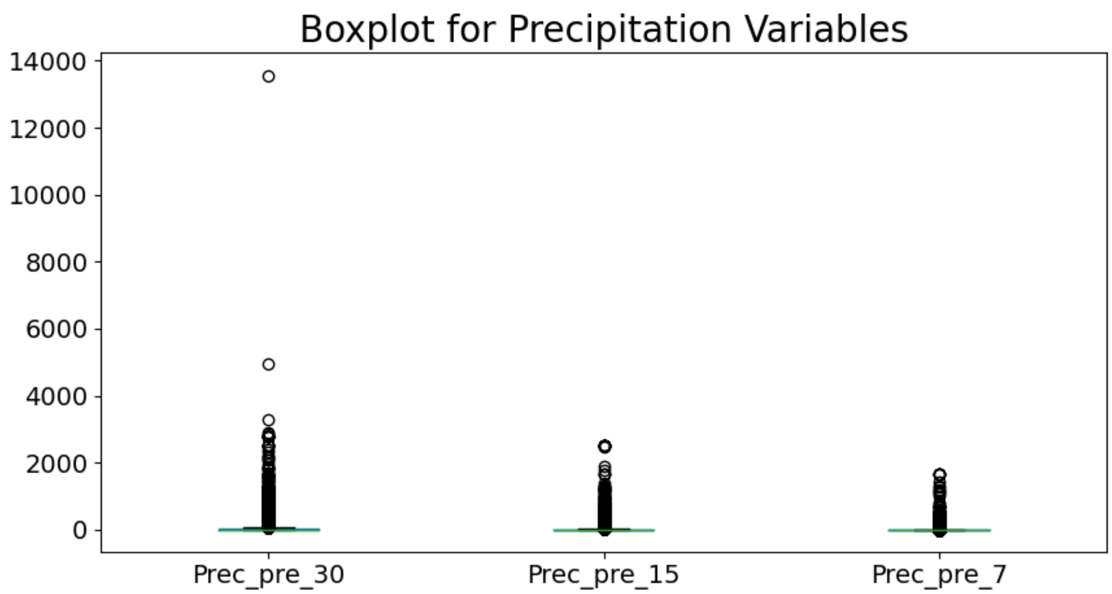
[11]:
```
plt.figure(figsize=(10, 5))
plt.title("Boxplot for Temperature Variables", fontsize=20)
plt.grid(color='grey', linestyle='-', linewidth=0.25, alpha=0.5)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

# Create boxplot
df.boxplot(column=["Temp_pre_30", "Temp_pre_15", "Temp_pre_7"], grid=False)

plt.show()
```



[12]:
```
plt.figure(figsize=(10, 5))
plt.title("Boxplot for Humidity Variables", fontsize=20)
plt.grid(color='grey', linestyle='-', linewidth=0.25, alpha=0.5)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

# Create boxplot
df.boxplot(column=["Hum_pre_30", "Hum_pre_15", "Hum_pre_7"], grid=False)

plt.show()
```



[13]:
```
plt.figure(figsize=(10, 5))
plt.title("Boxplot for Precipitation Variables", fontsize=20)
plt.grid(color='grey', linestyle='-', linewidth=0.25, alpha=0.5)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

# Create boxplot
df.boxplot(column=["Prec_pre_30", "Prec_pre_15", "Prec_pre_7"], grid=False)

plt.show()
```



Will apply minmaxscaler to take care of these outliers in the weather data

[14]:
```
(df.fire_size_class.value_counts()/df.shape[0])*100
```

[14]:
```
fire_size_class
B    65.963480
C    19.526071
G     7.173948
F     3.554464
D     2.517745
E     1.264291
Name: count, dtype: float64
```

As the target class is unbalanced we decided to club the smaller group of classes as 1.

Clubbing (C,D,E,F,G) as (1) class and A,B as (0) class.

0 idicates small fire <25 Acres and 1 represents a widespread fire >25Acres.

[15]:
```
class_mapping = {'A': 0, 'B': 0, 'C':1, 'D':1, 'E':1, 'F':1, 'G':1}
df = df.replace(class_mapping)
```

[16]:
```
df.fire_size_class.value_counts()
```

[16]:
```
fire_size_class
0    36522
1    18845
Name: count, dtype: int64
```

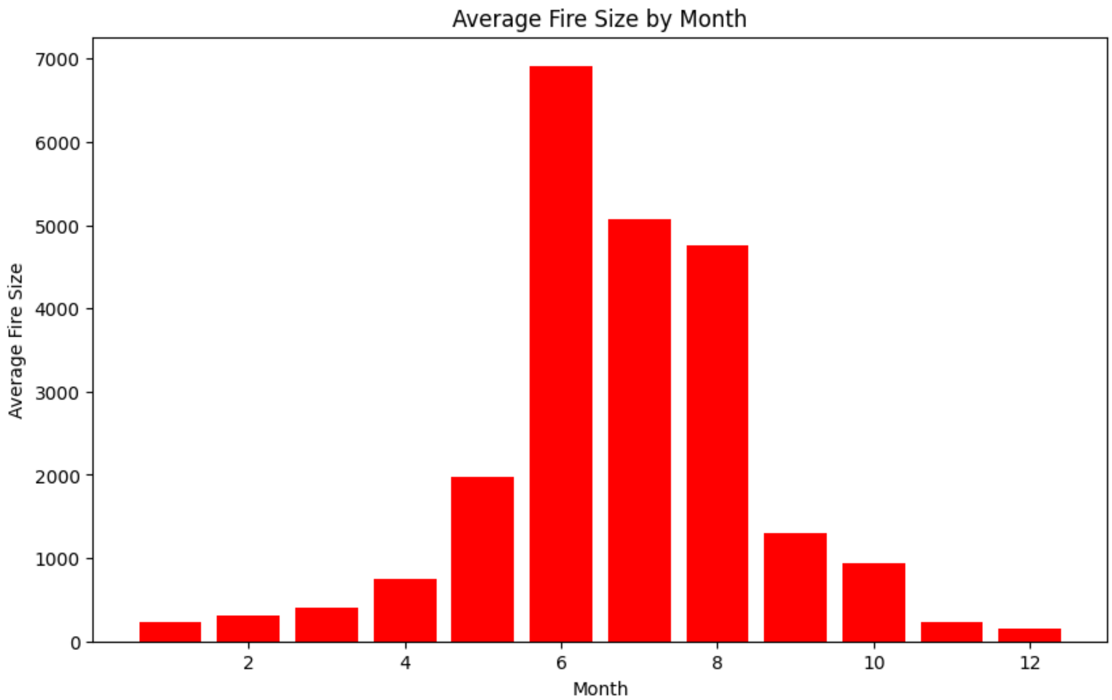
[17]:
```
(df.fire_size_class.value_counts()/df.shape[0])*100
```

[17]:
```
fire_size_class
0    65.96348
1    34.03652
Name: count, dtype: float64
```

1.0.5 Extracting the date and month and removing the redundant columns.

[18]:
```
# Extract day, month, year from discovery clean date
df['disc_clean_date'] = pd.to_datetime(df['disc_clean_date'])

df['disc_month'] = df['disc_clean_date'].dt.month

# Drop the columns which are not required
df = df.drop(['disc_clean_date', 'disc_date_pre', \
              'wstation_byear', 'wstation_eyear'],axis=1)
```

[19]:
```
# Group by 'disc_month' and calculate the average fire size for each month
average_fire_size_per_month = df.groupby('disc_month')['fire_size'].mean().
 ↪reset_index()

# Create a bar chart
plt.figure(figsize=(10, 6))
plt.bar(average_fire_size_per_month['disc_month'],␣
 ↪average_fire_size_per_month['fire_size'], color='red')
plt.xlabel('Month')
plt.ylabel('Average Fire Size')
plt.title('Average Fire Size by Month')
plt.show()
```



1.0.6 Bigger fires occur during the summer season, so months columns will be an important variable

[20]:
```
df= df.drop(["fire_size"], axis = 1)
```

[21]:
```
df['Vegetation'] = df['Vegetation'].astype(object)
```

1.0.7 Applying MinMaxScaler to the weather variables

[22]:
```
from sklearn.preprocessing import MinMaxScaler

trans = MinMaxScaler()
df.iloc[:, 5:21] = trans.fit_transform(df.iloc[:, 5:21])
```

[23]:
```
X = df.drop('fire_size_class',axis=1)
y = df['fire_size_class']
```

[24]:
```
X.columns
```

[24]:
```
Index(['stat_cause_descr', 'latitude', 'longitude', 'Vegetation',
       'Temp_pre_30', 'Temp_pre_15', 'Temp_pre_7', 'Temp_cont', 'Wind_pre_30',
       'Wind_pre_15', 'Wind_pre_7', 'Wind_cont', 'Hum_pre_30', 'Hum_pre_15',
       'Hum_pre_7', 'Hum_cont', 'Prec_pre_30', 'Prec_pre_15', 'Prec_pre_7',
```

```
 'Prec_cont', 'remoteness', 'disc_month'],
dtype='object')
```

1.0.8 Defining the function for target encoding

[25]:
```
def target_encode_multiclass(X,y): #X,y are pandas df and series
    y=y.astype(str)   #convert to string to onehot encode
    enc=ce.OneHotEncoder().fit(y)
    y_onehot=enc.transform(y)
    class_names=y_onehot.columns  #names of onehot encoded columns
    X_obj=X.select_dtypes('object') #separate categorical columns
    X=X.select_dtypes(exclude='object')
    for class_ in class_names:
        enc=ce.TargetEncoder()
        enc.fit(X_obj,y_onehot[class_]) #convert all categorical
        temp=enc.transform(X_obj)       #columns for class_
        temp.columns=[str(x)+'_'+str(class_) for x in temp.columns]
        X=pd.concat([X,temp],axis=1)    #add to original dataset

    return X
```

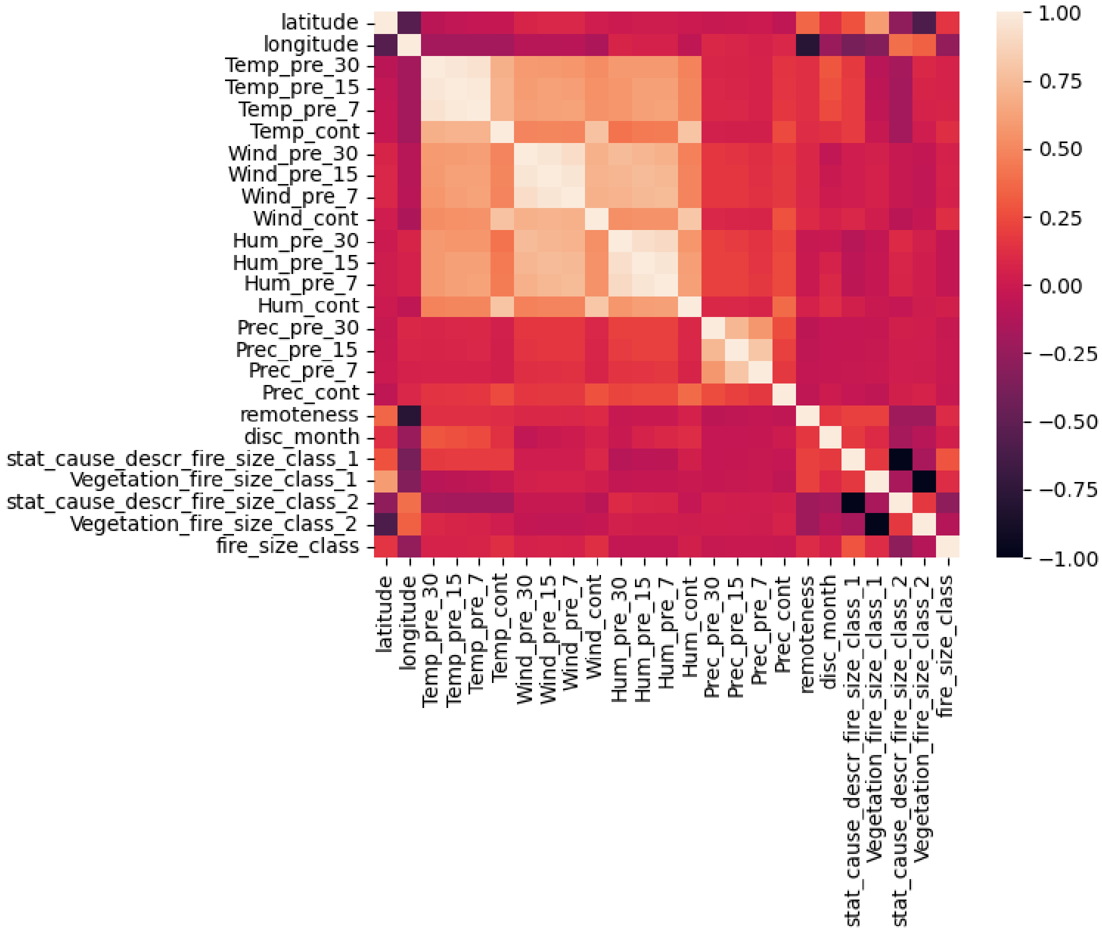
[26]:
```
X = target_encode_multiclass(X,y)
```

[27]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55367 entries, 0 to 55366
Data columns (total 24 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   latitude                          55367 non-null  float64
 1   longitude                         55367 non-null  float64
 2   Temp_pre_30                       55367 non-null  float64
 3   Temp_pre_15                       55367 non-null  float64
 4   Temp_pre_7                        55367 non-null  float64
 5   Temp_cont                         55367 non-null  float64
 6   Wind_pre_30                       55367 non-null  float64
 7   Wind_pre_15                       55367 non-null  float64
 8   Wind_pre_7                        55367 non-null  float64
 9   Wind_cont                         55367 non-null  float64
 10  Hum_pre_30                        55367 non-null  float64
 11  Hum_pre_15                        55367 non-null  float64
 12  Hum_pre_7                         55367 non-null  float64
 13  Hum_cont                          55367 non-null  float64
 14  Prec_pre_30                       55367 non-null  float64
 15  Prec_pre_15                       55367 non-null  float64
 16  Prec_pre_7                        55367 non-null  float64
```

```
 17  Prec_cont                          55367 non-null  float64
 18  remoteness                         55367 non-null  float64
 19  disc_month                         55367 non-null  int32
 20  stat_cause_descr_fire_size_class_1 55367 non-null  float64
 21  Vegetation_fire_size_class_1       55367 non-null  float64
 22  stat_cause_descr_fire_size_class_2 55367 non-null  float64
 23  Vegetation_fire_size_class_2       55367 non-null  float64
dtypes: float64(23), int32(1)
memory usage: 9.9 MB
```

[28]:
```
sns.heatmap(pd.concat([X, y], axis = 1).corr())
plt.show()
```



[29]:
```
X.describe(include="all")
```

[29]:
```
          latitude     longitude   Temp_pre_30   Temp_pre_15    Temp_pre_7  \
count  55367.000000  55367.000000  55367.000000  55367.000000  55367.000000
mean      36.172866    -94.757971      0.625182      0.539708      0.418654
```

```
std         6.724348     15.878194      0.109253      0.120064      0.142499
min        17.956533   -165.936000      0.000000      0.000000      0.000000
25%        32.265960   -102.541513      0.503186      0.406468      0.261663
50%        34.600000    -91.212359      0.617428      0.532045      0.409829
75%        38.975235    -82.847500      0.722531      0.646503      0.545169
max        69.849500    -65.285833      1.000000      1.000000      1.000000

           Temp_cont    Wind_pre_30   Wind_pre_15    Wind_pre_7     Wind_cont  \
count   55367.000000   55367.000000  55367.000000  55367.000000  55367.000000
mean        0.393417      0.095004      0.092342      0.104544      0.084614
std         0.135891      0.068382      0.068919      0.080139      0.080580
min         0.000000      0.000000      0.000000      0.000000      0.000000
25%         0.279743      0.000000      0.000000      0.000000      0.000000
50%         0.292605      0.111132      0.107011      0.118930      0.039683
75%         0.511406      0.145122      0.144009      0.164504      0.152722
max         1.000000      1.000000      1.000000      1.000000      1.000000

       …   Prec_pre_30   Prec_pre_15    Prec_pre_7     Prec_cont  \
count  …  55367.000000  55367.000000  55367.000000  55367.000000
mean   …      0.002011      0.005006      0.003472      0.007800
std    …      0.008262      0.022516      0.019039      0.028095
min    …      0.000000      0.000000      0.000000      0.000000
25%    …      0.000000      0.000000      0.000000      0.000000
50%    …      0.000074      0.000396      0.000610      0.000470
75%    …      0.001467      0.001820      0.000610      0.000470
max    …      1.000000      1.000000      1.000000      1.000000

          remoteness    disc_month  stat_cause_descr_fire_size_class_1  \
count   55367.000000  55367.000000                        55367.000000
mean        0.236799      5.694331                            0.340366
std         0.144865      3.024138                            0.136714
min         0.000000      1.000000                            0.133594
25%         0.137800      3.000000                            0.224751
50%         0.202114      5.000000                            0.329925
75%         0.284782      8.000000                            0.349959
max         1.000000     12.000000                            0.639328

       Vegetation_fire_size_class_1  stat_cause_descr_fire_size_class_2  \
count                  55367.000000                        55367.000000
mean                       0.340365                            0.659634
std                        0.052303                            0.136714
min                        0.298379                            0.360672
25%                        0.298379                            0.650041
50%                        0.299193                            0.670075
75%                        0.399263                            0.775249
max                        0.440397                            0.866406
```

```
       Vegetation_fire_size_class_2
count                  55367.000000
mean                       0.659635
std                        0.052303
min                        0.559603
25%                        0.600737
50%                        0.700807
75%                        0.701621
max                        0.701621

[8 rows x 24 columns]
```

[30]:
```
from sklearn.metrics import roc_curve, accuracy_score
from sklearn.metrics import auc, classification_report
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
```

1.0.9 Calculating Accuracy for 5 folds

[31]:
```
X = X.values
y = y.values

# Define the number of folds for cross-validation
num_folds = 5

# Initialize StratifiedKFold for stratified sampling based on the distribution␣
 ↪of classes
stratified_kfold = StratifiedKFold(n_splits=num_folds,
                                   shuffle=True,
                                   random_state=42)
```

[32]:
```
classifiers=[['Logistic Regression :',LogisticRegression()],
             ['Decision Tree Classification :',DecisionTreeClassifier()],
             ['Random Forest Classification :',RandomForestClassifier()],
             ['K-Neighbors Classification :',KNeighborsClassifier()],
             ['Gausian Naive Bayes :',GaussianNB()],
             ['Support Vector Classification :',SVC()]]

cla_pred=[]

for name,model in classifiers:
    model=model
    # Perform K-fold cross-validation
    for fold, (train_index, test_index) in enumerate(stratified_kfold.split(X,␣
 ↪y)):
        X_train, X_test = X[train_index], X[test_index]
```

```
        y_train, y_test = y[train_index], y[test_index]

        model.fit(X_train, y_train)

        # Make predictions on the test set
        predictions = model.predict(X_test)

        # Evaluate the model
        cla_pred.append(accuracy_score(y_test,predictions))
        print(name, fold+1, accuracy_score(y_test,predictions))
```

```
Logistic Regression : 1 0.7214195412678346
Logistic Regression : 2 0.7254831135994221
Logistic Regression : 3 0.725729251332069
Logistic Regression : 4 0.7242842951323039
Logistic Regression : 5 0.7306962882687619
Decision Tree Classification : 1 0.6708506411414124
Decision Tree Classification : 2 0.679971103485642
Decision Tree Classification : 3 0.6780456967398176
Decision Tree Classification : 4 0.6751557843402872
Decision Tree Classification : 5 0.6796712724645534
Random Forest Classification : 1 0.7515802781289507
Random Forest Classification : 2 0.7519414845584251
Random Forest Classification : 3 0.7520093922152985
Random Forest Classification : 4 0.7546283753273729
Random Forest Classification : 5 0.7567958096270206
K-Neighbors Classification : 1 0.7165432544699296
K-Neighbors Classification : 2 0.7174462705436156
K-Neighbors Classification : 3 0.7101959721845932
K-Neighbors Classification : 4 0.715704867696198
K-Neighbors Classification : 5 0.7117312381468437
Gausian Naive Bayes : 1 0.6743724038287882
Gausian Naive Bayes : 2 0.6794292938414304
Gausian Naive Bayes : 3 0.6756073331527138
Gausian Naive Bayes : 4 0.6798518919895241
Gausian Naive Bayes : 5 0.677052289352479
Support Vector Classification : 1 0.6731984829329962
Support Vector Classification : 2 0.6744627054361567
Support Vector Classification : 3 0.673981757427978
Support Vector Classification : 4 0.673710828140522
Support Vector Classification : 5 0.6749751648153165
```

1.0.10 Printing Classifiction Report for 5 folds

Logistic Regression

[33]:
```
X1 = df.drop('fire_size_class',axis=1)
y1 = df['fire_size_class']
```

```
X1 = target_encode_multiclass(X1, y1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X1, y1, test_size=0.2,␣
↪random_state=42)

# Set up k-fold cross-validation
stratified_kf = StratifiedKFold(n_splits=num_folds, shuffle=True,␣
↪random_state=42)
```

[34]:
```
LR = LogisticRegression()

# Lists to store metrics for each fold
LR_precision_list = []
LR_recall_list = []
LR_f1_list = []

for train_index, val_index in stratified_kf.split(X_train, y_train):
    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.
↪iloc[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.
↪iloc[val_index]

    LR.fit(X_train_fold, y_train_fold)
    LR_pred = LR.predict(X_val_fold)

    # Calculate metrics for each fold
    classification_report_fold = classification_report(y_val_fold, LR_pred,␣
↪output_dict=True)

    LR_precision_list.append(classification_report_fold['weighted␣
↪avg']['precision'])
    LR_recall_list.append(classification_report_fold['weighted avg']['recall'])
    LR_f1_list.append(classification_report_fold['weighted avg']['f1-score'])

# Calculate mean metrics across all folds
mean_LR_precision = np.mean(LR_precision_list)
mean_LR_recall = np.mean(LR_recall_list)
mean_LR_f1 = np.mean(LR_f1_list)

# Print or use the mean metrics as needed
print(f'Mean Precision for LR: {mean_LR_precision}')
print(f'Mean Recall for LR: {mean_LR_recall}')
print(f'Mean F1-Score for LR: {mean_LR_f1}')
```

```
Mean Precision for LR: 0.7175383891667414
```

```
Mean Recall for LR: 0.7224844405762785
Mean F1-Score for LR: 0.6879843015053047
```

DecisionTree

[35]:
```
DT = DecisionTreeClassifier()

# Lists to store metrics for each fold
DT_precision_list = []
DT_recall_list = []
DT_f1_list = []

for train_index, val_index in stratified_kf.split(X_train, y_train):
    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.
 ↪iloc[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.
 ↪iloc[val_index]

    DT.fit(X_train_fold, y_train_fold)
    DT_pred = DT.predict(X_val_fold)

    # Calculate metrics for each fold
    classification_report_fold = classification_report(y_val_fold, DT_pred,␣
 ↪output_dict=True)

    DT_precision_list.append(classification_report_fold['weighted␣
 ↪avg']['precision'])
    DT_recall_list.append(classification_report_fold['weighted avg']['recall'])
    DT_f1_list.append(classification_report_fold['weighted avg']['f1-score'])

# Calculate mean metrics across all folds
mean_DT_precision = np.mean(DT_precision_list)
mean_DT_recall = np.mean(DT_recall_list)
mean_DT_f1 = np.mean(DT_f1_list)

# Print or use the mean metrics as needed
print(f'Mean Precision for DT: {mean_DT_precision}')
print(f'Mean Recall for DT: {mean_DT_recall}')
print(f'Mean F1-Score for DT: {mean_DT_f1}')
```

```
Mean Precision for DT: 0.6809505451590552
Mean Recall for DT: 0.6792944892577222
Mean F1-Score for DT: 0.6800503067789476
```

Random Forest

[36]:
```
RF = RandomForestClassifier()

# Lists to store metrics for each fold
```

```
RF_precision_list = []
RF_recall_list = []
RF_f1_list = []

for train_index, val_index in stratified_kf.split(X_train, y_train):
    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.
 ↪iloc[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.
 ↪iloc[val_index]

    RF.fit(X_train_fold, y_train_fold)
    RF_pred = RF.predict(X_val_fold)

    # Calculate metrics for each fold
    classification_report_fold = classification_report(y_val_fold, RF_pred,␣
 ↪output_dict=True)

    RF_precision_list.append(classification_report_fold['weighted␣
 ↪avg']['precision'])
    RF_recall_list.append(classification_report_fold['weighted avg']['recall'])
    RF_f1_list.append(classification_report_fold['weighted avg']['f1-score'])

# Calculate mean metrics across all folds
mean_RF_precision = np.mean(RF_precision_list)
mean_RF_recall = np.mean(RF_recall_list)
mean_RF_f1 = np.mean(RF_f1_list)

# Print or use the mean metrics as needed
print(f'Mean Precision for RF: {mean_RF_precision}')
print(f'Mean Recall for RF: {mean_RF_recall}')
print(f'Mean F1-Score for RF: {mean_RF_f1}')
```

```
Mean Precision for RF: 0.743376848791913
Mean Recall for RF: 0.7493508584389676
Mean F1-Score for RF: 0.7307146385847803
```

KNN

[37]:
```
RF = KNeighborsClassifier()

# Lists to store metrics for each fold
RF_precision_list = []
RF_recall_list = []
RF_f1_list = []

for train_index, val_index in stratified_kf.split(X_train, y_train):
    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.
 ↪iloc[val_index]
```

```
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.
 ↪iloc[val_index]

    RF.fit(X_train_fold, y_train_fold)
    RF_pred = RF.predict(X_val_fold)

    # Calculate metrics for each fold
    classification_report_fold = classification_report(y_val_fold, RF_pred,␣
 ↪output_dict=True)

    RF_precision_list.append(classification_report_fold['weighted␣
 ↪avg']['precision'])
    RF_recall_list.append(classification_report_fold['weighted avg']['recall'])
    RF_f1_list.append(classification_report_fold['weighted avg']['f1-score'])

# Calculate mean metrics across all folds
mean_RF_precision = np.mean(RF_precision_list)
mean_RF_recall = np.mean(RF_recall_list)
mean_RF_f1 = np.mean(RF_f1_list)

# Print or use the mean metrics as needed
print(f'Mean Precision for RF: {mean_RF_precision}')
print(f'Mean Recall for RF: {mean_RF_recall}')
print(f'Mean F1-Score for RF: {mean_RF_f1}')
```

```
Mean Precision for RF: 0.6973526563246721
Mean Recall for RF: 0.7094800146730681
Mean F1-Score for RF: 0.6988606461552431
```

Naive Bayes

[38]:
```
RF = GaussianNB()

# Lists to store metrics for each fold
RF_precision_list = []
RF_recall_list = []
RF_f1_list = []

for train_index, val_index in stratified_kf.split(X_train, y_train):
    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.
 ↪iloc[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.
 ↪iloc[val_index]

    RF.fit(X_train_fold, y_train_fold)
    RF_pred = RF.predict(X_val_fold)
```

```
    # Calculate metrics for each fold
    classification_report_fold = classification_report(y_val_fold, RF_pred,␣
↪output_dict=True)

    RF_precision_list.append(classification_report_fold['weighted␣
↪avg']['precision'])
    RF_recall_list.append(classification_report_fold['weighted avg']['recall'])
    RF_f1_list.append(classification_report_fold['weighted avg']['f1-score'])

# Calculate mean metrics across all folds
mean_RF_precision = np.mean(RF_precision_list)
mean_RF_recall = np.mean(RF_recall_list)
mean_RF_f1 = np.mean(RF_f1_list)

# Print or use the mean metrics as needed
print(f'Mean Precision for RF: {mean_RF_precision}')
print(f'Mean Recall for RF: {mean_RF_recall}')
print(f'Mean F1-Score for RF: {mean_RF_f1}')
```

```
Mean Precision for RF: 0.6615354006784583
Mean Recall for RF: 0.6781883307616214
Mean F1-Score for RF: 0.6643549622105983
```

SVM

[39]:
```
RF = SVC()

# Lists to store metrics for each fold
RF_precision_list = []
RF_recall_list = []
RF_f1_list = []

for train_index, val_index in stratified_kf.split(X_train, y_train):
    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.
↪iloc[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.
↪iloc[val_index]

    RF.fit(X_train_fold, y_train_fold)
    RF_pred = RF.predict(X_val_fold)

    # Calculate metrics for each fold
    classification_report_fold = classification_report(y_val_fold, RF_pred,␣
↪output_dict=True)

    RF_precision_list.append(classification_report_fold['weighted␣
↪avg']['precision'])
```

```
    RF_recall_list.append(classification_report_fold['weighted avg']['recall'])
    RF_f1_list.append(classification_report_fold['weighted avg']['f1-score'])

# Calculate mean metrics across all folds
mean_RF_precision = np.mean(RF_precision_list)
mean_RF_recall = np.mean(RF_recall_list)
mean_RF_f1 = np.mean(RF_f1_list)

# Print or use the mean metrics as needed
print(f'Mean Precision for RF: {mean_RF_precision}')
print(f'Mean Recall for RF: {mean_RF_recall}')
print(f'Mean F1-Score for RF: {mean_RF_f1}')
```

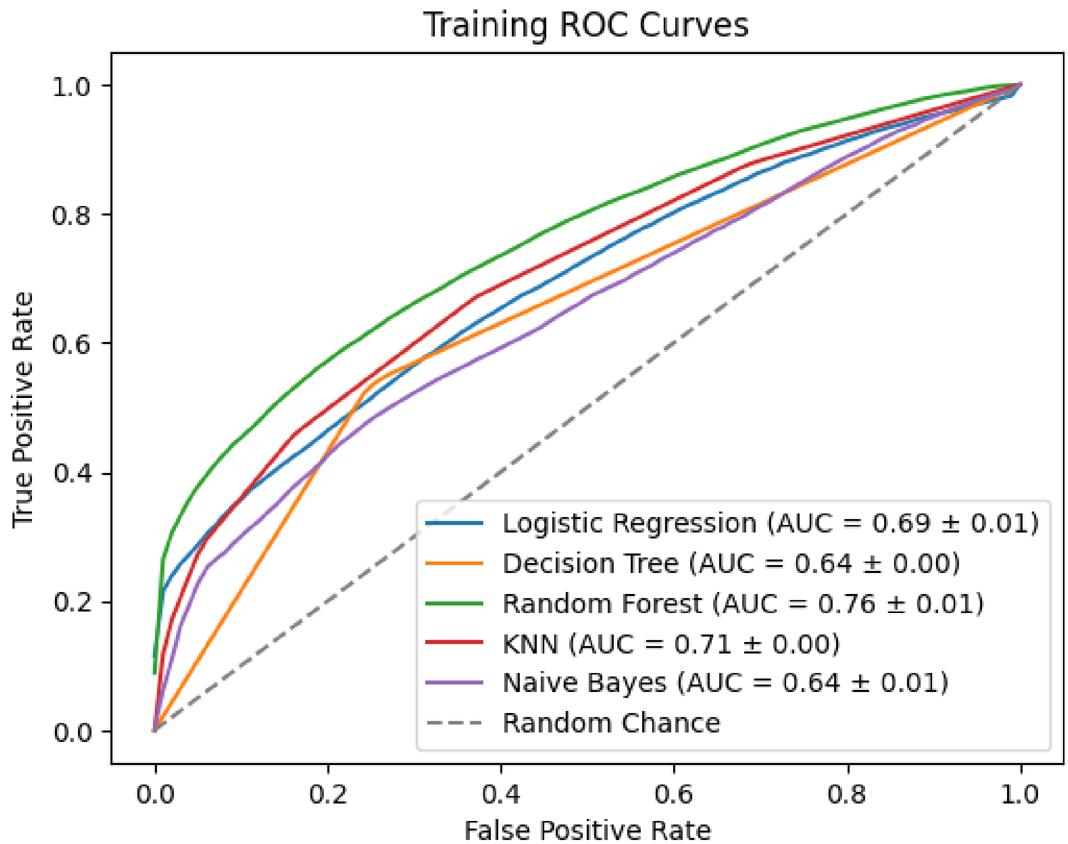
```
Mean Precision for RF: 0.7479669321680513
Mean Recall for RF: 0.674892186514749
Mean F1-Score for RF: 0.5607237621001302
```

1.0.11 Plotting the ROC curves for all 6 models

[40]:
```
models = {
    'Logistic Regression': LogisticRegression(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'KNN': KNeighborsClassifier(),
    'Naive Bayes': GaussianNB(),
#     'SVM': SVC(probability=True)
}

# Set up k-fold cross-validation
stratified_kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Iterate through each model
for model_name, model in models.items():
    print(f"Evaluating {model_name}")

    # Perform k-fold cross-validation
    mean_fpr = np.linspace(0, 1, 100)
    tpr_list = []

    for train_index, val_index in stratified_kf.split(X_train, y_train):
        X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.
↪iloc[val_index]
        y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.
↪iloc[val_index]

        model.fit(X_train_fold, y_train_fold)
        y_pred = model.predict_proba(X_val_fold)[:, 1]
```

```
        fpr, tpr, _ = roc_curve(y_val_fold, y_pred)
        tpr_list.append(np.interp(mean_fpr, fpr, tpr))

    # Compute mean and standard deviation of the ROC curves
    mean_tpr = np.mean(tpr_list, axis=0)
    mean_auc = auc(mean_fpr, mean_tpr)
    std_auc = np.std(tpr_list, axis=0)

    # Plot the training ROC curve with mean and standard deviation
    plt.plot(mean_fpr, mean_tpr, label=f'{model_name} (AUC = {mean_auc:.2f} ±␣
 ↪{np.mean(std_auc):.2f})')


# Plot the random chance line
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random Chance')

# Set plot labels and title
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Training ROC Curves')
plt.legend(loc='lower right')
plt.show()
```

```
Evaluating Logistic Regression
Evaluating Decision Tree
Evaluating Random Forest
Evaluating KNN
Evaluating Naive Bayes
```



1.0.12 select models for ensemble based on roc curves, 3 or all?

Ensemble Method

[41]:
```
from sklearn.metrics import log_loss
from sklearn.ensemble import VotingClassifier, StackingClassifier

LR = LogisticRegression()
DT = DecisionTreeClassifier()
RF = RandomForestClassifier()
KNN = KNeighborsClassifier()
SVM = SVC()
NB = GaussianNB()

# # Voting Classifier
voting_classifier = VotingClassifier(estimators=[('rf', LR),
                                                 ('knn', KNN),
                                                 ('lr', LR)],
                                     voting='hard')
# 'hard' for majority voting,
#'soft' for weighted voting based on probabilities
```

```python
# Stacking Classifier
stacking_classifier = StackingClassifier(estimators=[('rf', LR),
                                                     ('knn', KNN),
                                                     ('lr', LR)],
                                         final_estimator=LogisticRegression())

# Train and evaluate each classifier
classifiers = [voting_classifier,
               stacking_classifier]

for classifier in classifiers:
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{classifier.__class__.__name__} Accuracy: {accuracy}")
```

```
VotingClassifier Accuracy: 0.7246703991331046
StackingClassifier Accuracy: 0.7338811630847029
```

1.0.13 Random forest works better, followed by KNN, Logistic Regression, Decision Tree and SVM in that order