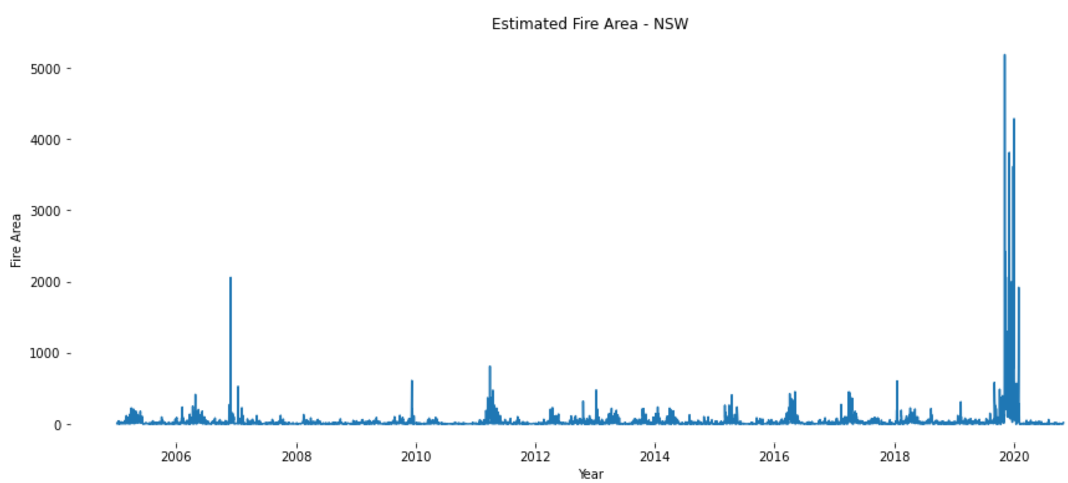
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.decomposition import FactorAnalysis
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import math
import scipy
from scipy.stats import norm,skew
import matplotlib.pyplot as plt
import numpy as np


import warnings
warnings.filterwarnings('ignore')

pd.set_option('display.max_columns', 500)
```

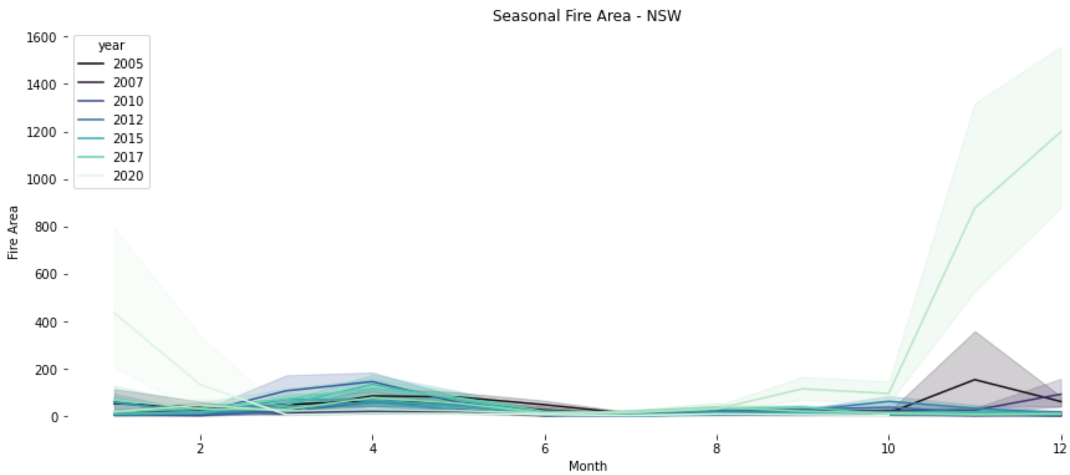
Historical Wildfire Dataset

```
# Load the dataset
df_wf = pd.read_csv("Historical_Wildfires.csv", parse_dates=[1])
VegetationIndex = pd.read_csv('VegetationIndex.csv')
LandClass = pd.read_csv('LandClass.csv')


# estimated fire area of specific region over time

fig, ax = plt.subplots(figsize = (15, 6))
df_NSW = df_wf[df_wf["Region"] == "NSW"]

sns.lineplot(df_NSW["Date"], df_NSW["Estimated_fire_area"])

ax.set_title("Estimated Fire Area - NSW")
ax.set_xlabel("Year")
ax.set_ylabel("Fire Area")

sns.despine(left=True,bottom=True)
```

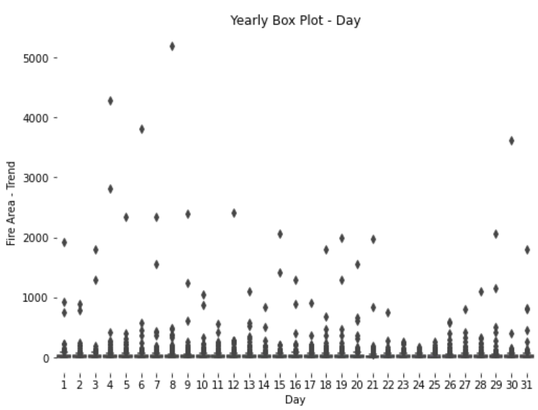


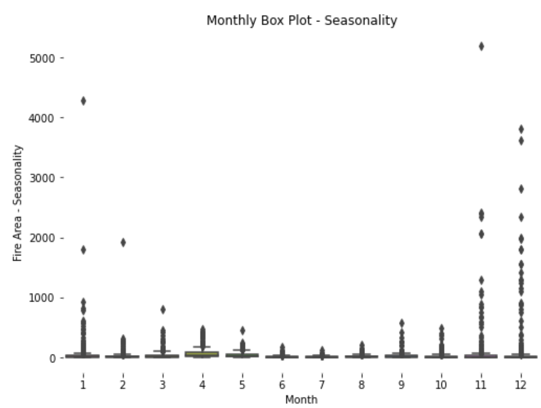

```
# segregation of day, month, year
df_NSW["day"] = df_NSW["Date"].dt.day
df_NSW["month"] = df_NSW["Date"].dt.month
df_NSW["year"] = df_NSW["Date"].dt.year

# seasonality in Estimated fire area of a region over time
fig, ax = plt.subplots(figsize = (15, 6))

sns.lineplot(df_NSW["month"], df_NSW["Estimated_fire_area"], hue = df_NSW["year"], palette = "mako")

ax.set_title("Seasonal Fire Area - NSW")
ax.set_xlabel("Month")
```

```
ax.set_ylabel("Fire Area")

sns.despine(left=True,bottom=True)
```




```
# check the seasonality wrt year and month

fig, ax = plt.subplots(nrows = 1, ncols = 2, figsize = (18, 6))

sns.boxplot(df_NSW["day"], df_NSW["Estimated_fire_area"], ax = ax[0])
ax[0].set_title("Yearly Box Plot - Day")
ax[0].set_xlabel("Day")
ax[0].set_ylabel("Fire Area - Trend")

sns.despine(left=True,bottom=True)

sns.boxplot(df_NSW["month"], df_NSW["Estimated_fire_area"], ax = ax[1])
ax[1].set_title("Monthly Box Plot - Seasonality")
ax[1].set_xlabel("Month")
ax[1].set_ylabel("Fire Area - Seasonality")

sns.despine(left=True,bottom=True)
```

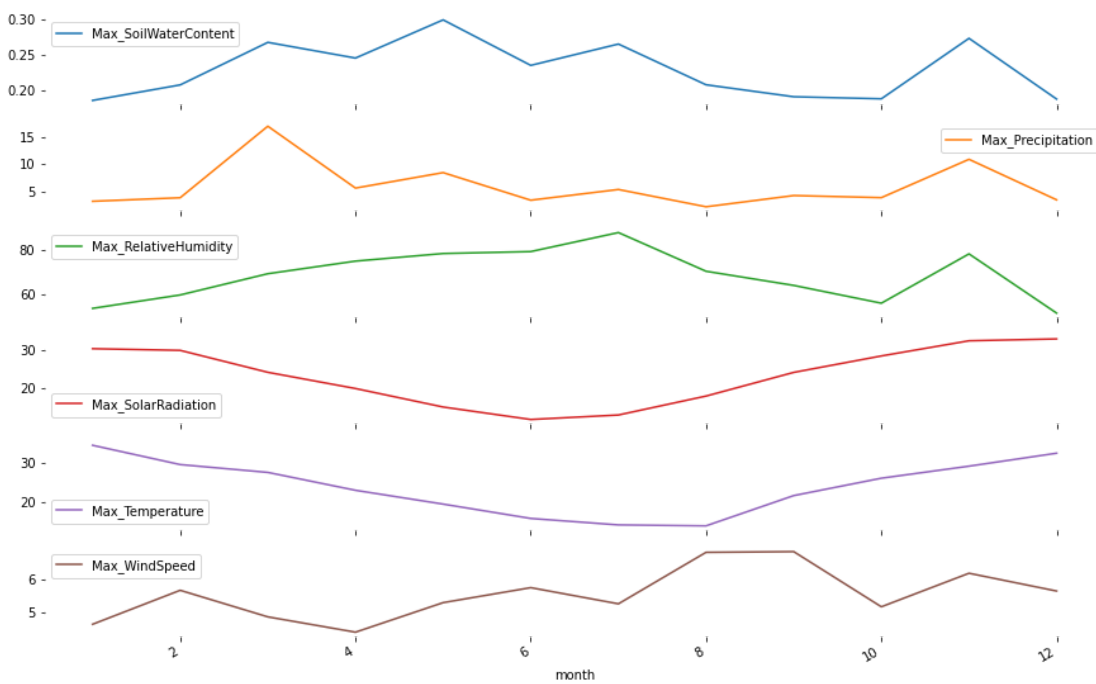




Historical Weather Dataset

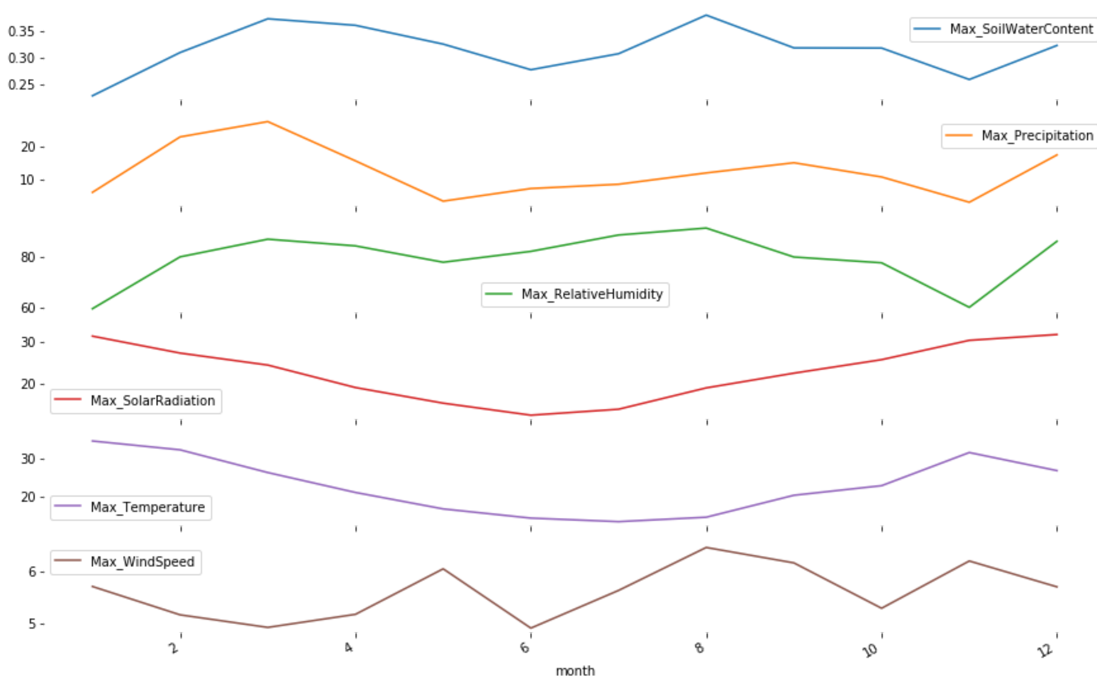
```
# multivariate analysis for 2019 (Groupby Max())

df_wt = pd.read_csv("HistoricalWeather.csv", parse_dates=[1])
df_wt["Date"] = pd.to_datetime(df_wt["Date"])

df_NT = df_wt[df_wt["Region"] == "NSW"]
df_NT["day"] = df_NT["Date"].dt.day
df_NT["month"] = df_NT["Date"].dt.month
df_NT["year"] = df_NT["Date"].dt.year

df_NT_2019 = df_NT[df_NT["year"] == 2019]

# multivariate analysis for 2019 (Groupby Max())
```

```
Precipitation = []
RelativeHumidity = []
SoilWaterContent = []
SolarRadiation = []
Temperature = []
WindSpeed = []

for i in df_NT_2019['month'].unique():

    Precipitation.append(float(df_NT_2019[df_NT_2019['month'] == i].groupby('Parameter').max()[['mean()']].values[0]))
    RelativeHumidity.append(float(df_NT_2019[df_NT_2019['month'] == i].groupby('Parameter').max()[['mean()']].values[1]))
    SoilWaterContent.append(float(df_NT_2019[df_NT_2019['month'] == i].groupby('Parameter').max()[['mean()']].values[2]))
    SolarRadiation.append(float(df_NT_2019[df_NT_2019['month'] == i].groupby('Parameter').max()[['mean()']].values[3]))
    Temperature.append(float(df_NT_2019[df_NT_2019['month'] == i].groupby('Parameter').max()[['mean()']].values[4]))
    WindSpeed.append(float(df_NT_2019[df_NT_2019['month'] == i].groupby('Parameter').max()[['mean()']].values[5]))


df_NT_2019 = pd.DataFrame({'Max_Precipitation':Precipitation,'Max_RelativeHumidity':RelativeHumidity,'Max_SoilWaterContent':SoilWaterContent,
df_NT_2019 = df_NT_2019[["Max_SoilWaterContent", "Max_Precipitation", "Max_RelativeHumidity",
                         "Max_SolarRadiation", "Max_Temperature", "Max_WindSpeed"]]

df_NT_2019['month'] = df_NT['month'].unique()
df_NT_2019.set_index('month',inplace = True)


df_NT_2019.plot(subplots = True, figsize = (15, 10))
sns.despine(left=True,bottom=True);
```




```
# multivariate analysis for 2020 (Groupby Max())

df_wt = pd.read_csv("HistoricalWeather.csv", parse_dates=[1])
df_wt["Date"] = pd.to_datetime(df_wt["Date"])

df_NT = df_wt[df_wt["Region"] == "NSW"]
df_NT["day"] = df_NT["Date"].dt.day
df_NT["month"] = df_NT["Date"].dt.month
df_NT["year"] = df_NT["Date"].dt.year

df_NT_2019 = df_NT[df_NT["year"] == 2020]


Precipitation = []
RelativeHumidity = []
SoilWaterContent = []
SolarRadiation = []
Temperature = []
WindSpeed = []
```

```
for i in df_NT_2019['month'].unique():

    Precipitation.append(float(df_NT_2019[df_NT_2019['month'] == i].groupby('Parameter').max()[['mean()']].values[0]))
    RelativeHumidity.append(float(df_NT_2019[df_NT_2019['month'] == i].groupby('Parameter').max()[['mean()']].values[1]))
    SoilWaterContent.append(float(df_NT_2019[df_NT_2019['month'] == i].groupby('Parameter').max()[['mean()']].values[2]))
    SolarRadiation.append(float(df_NT_2019[df_NT_2019['month'] == i].groupby('Parameter').max()[['mean()']].values[3]))
    Temperature.append(float(df_NT_2019[df_NT_2019['month'] == i].groupby('Parameter').max()[['mean()']].values[4]))
    WindSpeed.append(float(df_NT_2019[df_NT_2019['month'] == i].groupby('Parameter').max()[['mean()']].values[5]))


df_NT_2019 = pd.DataFrame({'Max_Precipitation':Precipitation,'Max_RelativeHumidity':RelativeHumidity,'Max_SoilWaterContent':SoilWaterContent,
df_NT_2019 = df_NT_2019[["Max_SoilWaterContent", "Max_Precipitation", "Max_RelativeHumidity",
                             "Max_SolarRadiation", "Max_Temperature", "Max_WindSpeed"]]

df_NT_2019['month'] = df_NT['month'].unique()
df_NT_2019.set_index('month',inplace = True)


df_NT_2019.plot(subplots = True, figsize = (15, 10))
sns.despine(left=True,bottom=True);
```



Data Cleaning

```
# Main dataset including Estimated_fire_area

df_wf.head()
```

	Region	Date	Estimated_fire_area	Mean_estimated_fire_brightness	Mean_estimated_fire_radiative_power	M
0	NSW	2005-01-04	8.68000	312.266667	42.400000	
1	NSW	2005-01-05	16.61125	322.475000	62.362500	
2	NSW	2005-01-06	5.52000	325.266667	38.400000	
3	NSW	2005-01-07	6.26400	313.870000	33.800000	
4	NSW	2005-01-08	5.40000	337.383333	122.533333	

```
# how weather dataset looks like

df_wt.head(-5)
```

	Date	Region	Parameter	count()[unit: km^2]	min()	max()	mean()	variance()
0	2005-01-01	NSW	Precipitation	8.002343e+05	0.000000	1.836935	0.044274	0.028362
1	2005-01-01	NSW	RelativeHumidity	8.002343e+05	13.877194	80.522964	36.355567	253.559937
2	2005-01-01	NSW	SoilWaterContent	8.002343e+05	0.002245	0.414305	0.170931	0.007758
3	2005-01-01	NSW	SolarRadiation	8.002343e+05	14.515009	32.169781	26.749389	6.078587
4	2005-01-01	NSW	Temperature	8.002343e+05	14.485785	35.878704	27.341182	18.562212
...	...	...	...	...	...	...	...	...
242771	2020-10-31	VI	SoilWaterContent	2.294532e+05	0.000000	0.455111	0.324260	0.005050
242772	2020-10-31	VI	SolarRadiation	2.294532e+05	11.170260	28.041906	19.553751	9.917196
242773	2020-10-31	VI	Temperature	2.294532e+05	9.186510	17.307510	13.167147	4.088503
242774	2020-10-31	VI	WindSpeed	2.294532e+05	1.783996	6.605598	3.838360	1.019079
242775	2020-10-31	WA	Precipitation	2.528546e+06	0.000000	15.154541	0.328437	2.097161

242776 rows × 8 columns

```
# done it region by region and takes max() value for each parameter

NSW_dic = {}

NEW = df_wt[df_wt['Region'] == 'NSW']
for date in NEW['Date'].unique():
    NSW_dic[date] = NEW[NEW["Date"] == date].pivot_table(columns = ['Parameter']).loc['max()',:]

NSW_features = pd.DataFrame(NSW_dic).T
NSW_features.reset_index(inplace = True)
NSW_features.rename(columns={'index' : 'Date'},inplace = True)

NEW = df_wf[df_wf['Region'] == 'NSW']
NEW = pd.merge(NEW,NSW_features,on = 'Date')


NT_dic = {}

NT = df_wt[df_wt['Region'] == 'NT']
for date in NT['Date'].unique():
    NT_dic[date] = NT[NT["Date"] == date].pivot_table(columns = ['Parameter']).loc['max()',:]

NT_features = pd.DataFrame(NT_dic).T
NT_features.reset_index(inplace = True)
NT_features.rename(columns={'index' : 'Date'},inplace = True)

NT = df_wf[df_wf['Region'] == 'NT']
NT = pd.merge(NT,NT_features,on = 'Date')


QL_dic = {}

QL = df_wt[df_wt['Region'] == 'QL']
for date in QL['Date'].unique():
    QL_dic[date] = QL[QL["Date"] == date].pivot_table(columns = ['Parameter']).loc['max()',:]

QL_features = pd.DataFrame(QL_dic).T
QL_features.reset_index(inplace = True)
QL_features.rename(columns={'index' : 'Date'},inplace = True)

QL = df_wf[df_wf['Region'] == 'QL']
QL = pd.merge(QL,QL_features,on = 'Date')


SA_dic = {}

SA = df_wt[df_wt['Region'] == 'SA']
for date in SA['Date'].unique():
    SA_dic[date] = SA[SA["Date"] == date].pivot_table(columns = ['Parameter']).loc['max()',:]
```

```
SA_features = pd.DataFrame(SA_dic).T
SA_features.reset_index(inplace = True)
SA_features.rename(columns={'index' : 'Date'},inplace = True)

SA = df_wf[df_wf['Region'] == 'SA']
SA = pd.merge(SA,SA_features,on = 'Date')


TA_dic = {}

TA = df_wt[df_wt['Region'] == 'TA']
for date in TA['Date'].unique():
    TA_dic[date] = TA[TA["Date"] == date].pivot_table(columns = ['Parameter']).loc['max()',:]

TA_features = pd.DataFrame(TA_dic).T
TA_features.reset_index(inplace = True)
TA_features.rename(columns={'index' : 'Date'},inplace = True)

TA = df_wf[df_wf['Region'] == 'TA']
TA = pd.merge(TA,TA_features,on = 'Date')


VI_dic = {}

VI = df_wt[df_wt['Region'] == 'VI']
for date in VI['Date'].unique():
    TA_dic[date] = VI[VI["Date"] == date].pivot_table(columns = ['Parameter']).loc['max()',:]

VI_features = pd.DataFrame(VI_dic).T
VI_features.reset_index(inplace = True)
VI_features.rename(columns={'index' : 'Date'},inplace = True)

VI = df_wf[df_wf['Region'] == 'VI']
VI = pd.merge(VI,VI_features,on = 'Date')


WA_dic = {}

WA = df_wt[df_wt['Region'] == 'WA']
for date in WA['Date'].unique():
    WA_dic[date] = WA[WA["Date"] == date].pivot_table(columns = ['Parameter']).loc['max()',:]

WA_features = pd.DataFrame(WA_dic).T
WA_features.reset_index(inplace = True)
WA_features.rename(columns={'index' : 'Date'},inplace = True)

WA = df_wf[df_wf['Region'] == 'WA']
WA = pd.merge(WA,WA_features,on = 'Date')

# concat all region together

df = pd.concat([NEW, NT, QL, SA, TA, VI, WA])


# Merge with vegetationIndex based on 'year','month','Region'

df['Date'] = pd.to_datetime(df['Date'])
VegetationIndex['Date'] = pd.to_datetime(VegetationIndex['Date'])

df['year'] = df['Date'].dt.year
df['month'] = df['Date'].dt.month

VegetationIndex['year'] = VegetationIndex['Date'].dt.year
VegetationIndex['month'] = VegetationIndex['Date'].dt.month

df = pd.merge(df,VegetationIndex,on = ['year','month','Region'])

# Merge region

df = pd.merge(df,LandClass,on = 'Region')
```

Feature Engineering

```
# check the missing all columns
df.isnull().sum().sort_values().tail(7).index
```

```
# fillna by their median since they are not normal distribution

for col in df.isnull().sum().sort_values().tail(7).index:
    df[col] = df[col].fillna(df[col].median())

# consider these varibles as object rather than Int
df['month'] = df['month'].astype(str)
df['year'] = df['year'].astype(str)

# collect continuous variables
conti_features = df.drop(columns=['Date_x','Date_y','year','month','Replaced','Estimated_fire_area','Region'

# feature scaling using StandardScaler for continuous variables
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df[conti_features] = scaler.fit_transform(df[conti_features])

# feature encoding using onehot for three categorical varibales
from feature_engine.categorical_encoders import OneHotCategoricalEncoder
ohe_enc = OneHotCategoricalEncoder(
    top_categories=None,
    variables=['Region','Replaced','month'], # we can select which variables to encode
    drop_last=True) # to return k-1, false to return k

df = ohe_enc.fit_transform(df)
```

```
Index(['Precipitation', 'SolarRadiation', 'Temperature', 'WindSpeed',
       'RelativeHumidity', 'Var_confidence', 'Std_confidence'],
      dtype='object')
```

Split The Data

```
from sklearn.model_selection import train_test_split

X = df.drop(columns=['Date_x','Date_y','year','Estimated_fire_area'],axis=1)
y = df['Estimated_fire_area']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=0)

X_train.shape,X_test.shape
```

```
((16234, 48), (7996, 48))
```

Models building

```
from catboost import CatBoostRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from xgboost import XGBRegressor

model1 = LinearRegression()
model1.fit(X_train,y_train)


model2 = RandomForestRegressor(max_depth=4,n_jobs=-1)
model2.fit(X_train,y_train)


model3 = XGBRegressor(n_estimators=1000, learning_rate=0.02,gamma=0.1,
    min_child_weight=10,max_depth=3)
model3.fit(X_train, y_train,
             early_stopping_rounds=10,
             eval_set = [(X_train, y_train), (X_test, y_test)],
             verbose=False,
              eval_metric = 'rmse')
```

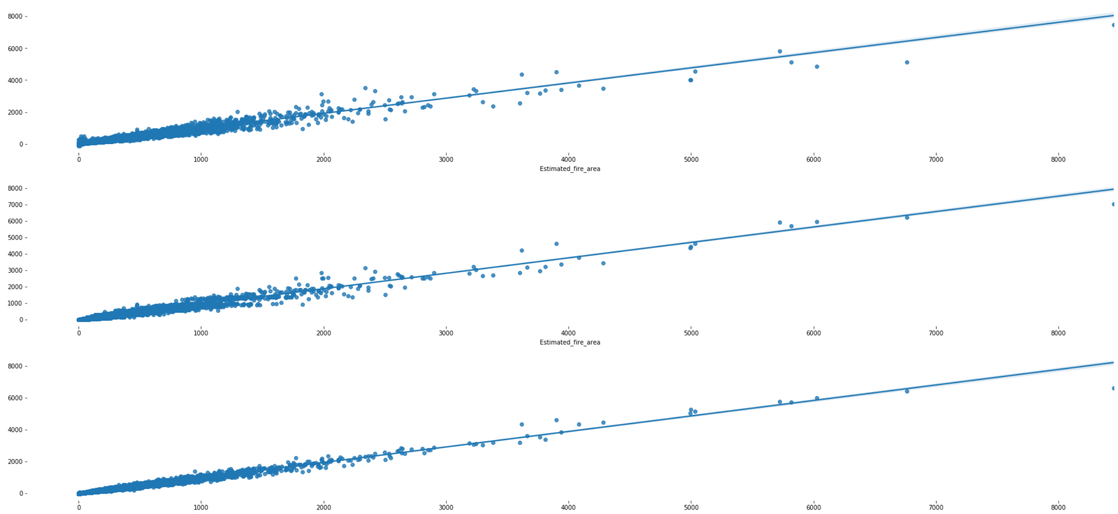
```
[00:59:54] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0.1,
             importance_type='gain', learning_rate=0.02, max_delta_step=0,
```

```
                  max_depth=3, min_child_weight=10, missing=None, n_estimators=1000,
                  n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
                  reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                  silent=None, subsample=1, verbosity=1)
```

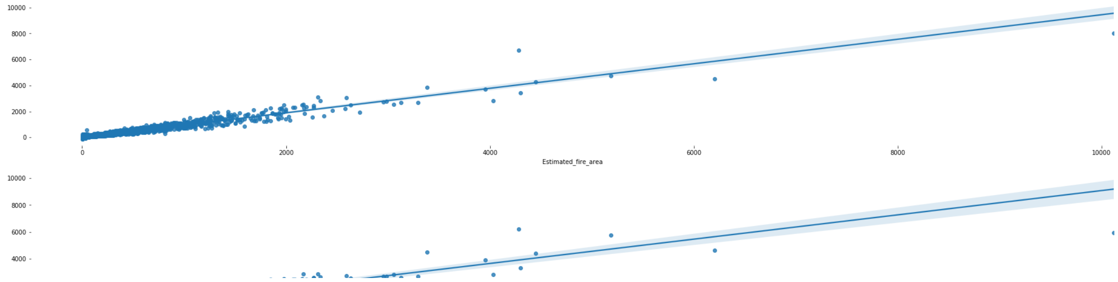
```
# Make prediction

prediction_1 = model1.predict(X_test)

prediction_2 = model2.predict(X_test)

prediction_3 = model3.predict(X_test)
```

Evaluation Performance

```
# Fitting on train dataset - graphs sequence are : LinearRegression,RandomForestRegressor,XGBRegressor


models = [model1,model2,model3]

i = 1
for mod in models:

    plt.figure(figsize=(32,15))
    plt.subplot(3,1,i)
    sns.regplot(y_train,mod.predict(X_train))
    sns.despine(left=True,bottom=True);

    i += 1
```




```
# Fitting on test dataset - graphs sequence are : LinearRegression,RandomForestRegressor,XGBRegressor

i = 1
for mod in models:

    plt.figure(figsize=(32,15))
    plt.subplot(3,1,i)
    sns.regplot(y_test,mod.predict(X_test))
    sns.despine(left=True,bottom=True)

    i += 1
```



The most common interpretation of r-squared is how well the regression model fits the observed data. For example, an r-squared of 60% reveals that 60% of the data fit the regression model. Generally, a higher r-squared indicates a better fit for the model.


```
# Evaluation metrix for train

from sklearn.metrics import r2_score,mean_squared_error,explained_variance_score

print('R2 for LinearRegression: ',r2_score(y_train,model1.predict(X_train))*100)
print('R2 for Ramdomforest: ',r2_score(y_train,model2.predict(X_train))*100)
print('R2 for XgboostRegressor: ',r2_score(y_train,model3.predict(X_train))*100)
```

```
R2 for LinearRegression:  95.24019875934758
R2 for Ramdomforest:  96.03643093015685
R2 for XgboostRegressor:  98.3080879175802
```

```
# evaluation metrix for test
# Simple weight averging ensemble

real_prediction = prediction_1 * 0.7 + prediction_3 * 0.3
# ----------------------

from sklearn.metrics import r2_score,mean_squared_error,explained_variance_score

print('R2 for LinearRegression: ',r2_score(y_test,prediction_1)*100)
print('R2 for Ramdomforest: ',r2_score(y_test,prediction_2)*100)
print('R2 for XgboostRegressor: ',r2_score(y_test,prediction_3)*100)
print('----------------------------------------------------------------')
print('R2 for Ensemble Method(70% linear mode & 30% Xgboost): ',r2_score(y_test,real_prediction)*100)
```

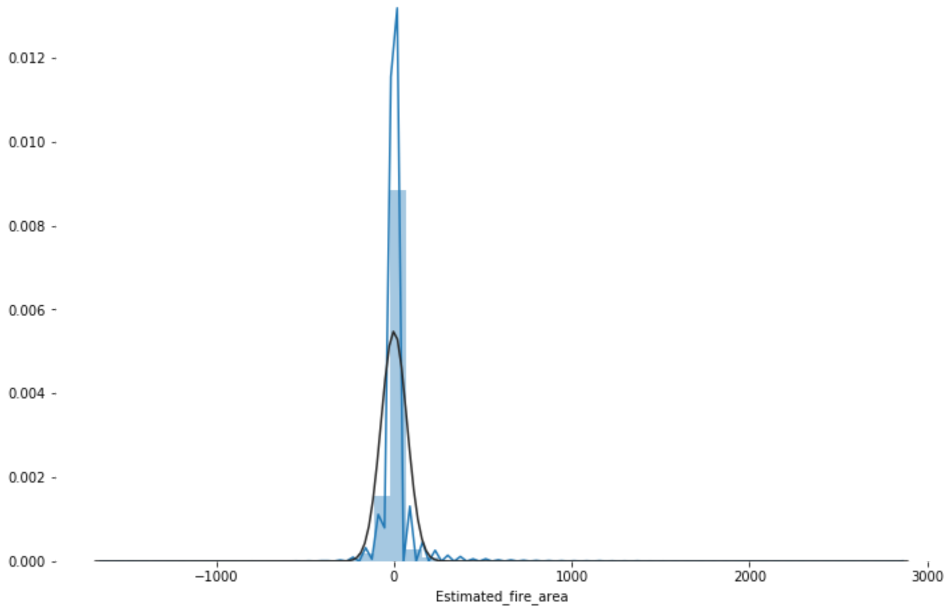
```
R2 for LinearRegression:  94.40883080881392
R2 for Ramdomforest:  93.19067008885854
R2 for XgboostRegressor:  94.90884668180595
----------------------------------------------------------------
R2 for Ensemble Method(70% linear mode & 30% Xgboost):  95.30598306220303
```

```
# Comprasion of real value and predictions

pd.DataFrame({'Real Value':y_test,'Predicition' : real_prediction}).tail(10)
```

	Real Value	Predicition
17479	24.360000	24.877834
547	16.250000	2.737882
16003	21.836923	-9.040801
6058	207.568421	137.656870
2421	13.407000	20.445118
17351	5.400000	30.130921
11655	74.710286	72.740106
19155	130.701613	103.164671
209	7.466667	20.045062
6164	1124.498506	1200.595070

```
# Residual graph
```

```
plt.figure(figsize=(12,8))
sns.distplot(y_test - real_prediction,fit=norm)
sns.despine(left=True,bottom=True)
```




```
# Feature importance of Xgboostregressor

pd.Series(model3.feature_importances_,index = X_train.columns).sort_values(ascending=True).tail(10).plot.barh(figsize=(12,10))
sns.despine(left=True,bottom=True);
```