Forest Fire Impact Prediction

Forest fires can be a friend and a foe. In the right place at the right time, forest fire can create many environmental benefits, such as reducing grass, brush, and trees that can fuel large and severe forest fires and improving wildlife habitat. In the wrong place at the wrong time, forest fires can wreak havoc, threatening lives, homes, communities, and natural and cultural resources.

In this project, our aim is to predict the burned area of forest fires, in the northeast region of Portugal. Based on the the spatial, temporal, and weather variables where the fire is spotted.

This prediction can be used for calculating the forces sent to the incident and deciding the urgency of the situation.

This program is designed to use geographical (Fire location), temporal (Month and Day), Fire Weather varibles () and weather variables (RH, Temp, Rain, Wind) to predict the area burned by forest fires. Data were obtained from the UCI Machine Learning database (http://archive.ics.uci.edu/ml/datasets/Forest+Fires (http://archive.ics.uci.edu/ml/datasets/Forest+Fires)) and contain details for 517 fires found in the Montesinho Natural Park in Portugal.

1. X - x-axis spatial coordinate within the Montesinho park map: 1 to 9
2. Y - y-axis spatial coordinate within the Montesinho park map: 2 to 9
3. month - month of the year: "jan" to "dec"
4. day - day of the week: "mon" to "sun"
5. FFMC - FFMC index from the FWI system: 18.7 to 96.20
6. DMC - DMC index from the FWI system: 1.1 to 291.3
7. DC - DC index from the FWI system: 7.9 to 860.6
8. ISI - ISI index from the FWI system: 0.0 to 56.10
9. temp - temperature in Celsius degrees: 2.2 to 33.30
10. RH - relative humidity in %: 15.0 to 100
11. wind - wind speed in km/h: 0.40 to 9.40
12. rain - outside rain in mm/m2 : 0.0 to 6.4
13. area - the burned area of the forest (in ha): 0.00 to 1090.84

Importing Libraries

In [55]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('ggplot')

from scipy.stats import zscore
import statsmodels.api as sm
from statsmodels.formula.api import ols
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
from statsmodels.stats.stattools import durbin_watson
from sklearn.linear_model import LinearRegression,RidgeCV,LassoCV,ElasticNetC
from sklearn.feature_selection import RFECV
!pip install mlxtend
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
```

```
Unable to create process using 'C:\Users\Aparna Akula\anaconda3\python.exe
"C:\Users\Aparna Akula\anaconda3\Scripts\pip-script.py" install mlxtend'
```

Load and describe data

In [56]:
```
forest_fires = pd.read_csv("forestfires.csv")
```

In [57]:
```
forest_fires.dtypes
```

Out[57]:
```
X          int64
Y          int64
month     object
day       object
FFMC     float64
DMC      float64
DC       float64
ISI      float64
temp     float64
RH         int64
wind     float64
rain     float64
area     float64
dtype: object
```

In [58]:
```
forest_fires.head()
```

Out[58]:

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	7	5	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.0
1	7	4	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.0
2	7	4	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.0
3	8	6	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.0
4	8	6	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.0

In [59]:
```
forest_fires.describe()
```

Out[59]:

	X	Y	FFMC	DMC	DC	ISI	temp	
count	517.000000	517.000000	517.000000	517.000000	517.000000	517.000000	517.000000	5
mean	4.669246	4.299807	90.644681	110.872340	547.940039	9.021663	18.889168	
std	2.313778	1.229900	5.520111	64.046482	248.066192	4.559477	5.806625	
min	1.000000	2.000000	18.700000	1.100000	7.900000	0.000000	2.200000	
25%	3.000000	4.000000	90.200000	68.600000	437.700000	6.500000	15.500000	
50%	4.000000	4.000000	91.600000	108.300000	664.200000	8.400000	19.300000	
75%	7.000000	5.000000	92.900000	142.400000	713.900000	10.800000	22.800000	
max	9.000000	9.000000	96.200000	291.300000	860.600000	56.100000	33.300000	1

Checking for Missing Values

In [60]:
```
forest_fires.isna().sum()
```

Out[60]:
```
X        0
Y        0
month    0
day      0
FFMC     0
DMC      0
DC       0
ISI      0
temp     0
RH       0
wind     0
rain     0
area     0
dtype: int64
```

In [61]:
```
plt.rcParams["figure.figsize"] = 9,5
```

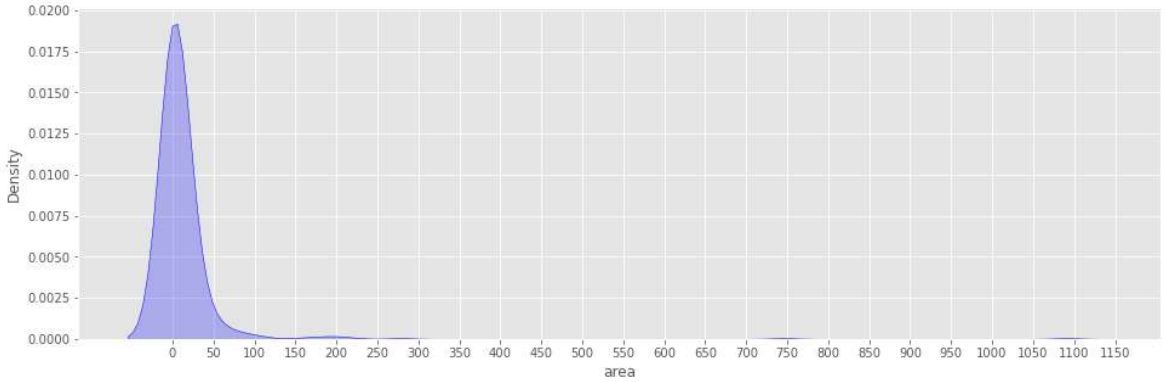
Exploratory Data Analysis

1.Univariate Analysis

Lets begin with the target variable as Area

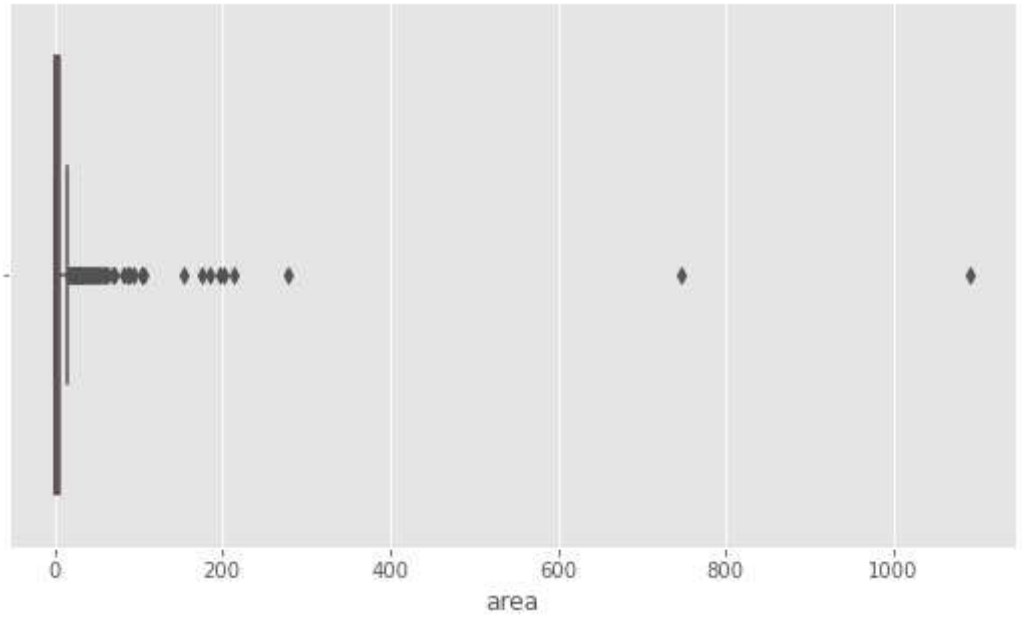
In [62]:
```
plt.figure(figsize=(16,5))
target = 'area'
print("Skew: {}".format(forest_fires[target].skew()))
print("Kurtosis: {}".format(forest_fires[target].kurtosis()))
ax = sns.kdeplot(forest_fires[target],shade=True,color='b')
plt.xticks([i for i in range(0,1200,50)])
plt.show()
```

```
Skew: 12.846933533934868
Kurtosis: 194.1407210942299
```



In [63]:
```
ax = sns.boxplot(forest_fires[target])
```

```
C:\Users\Aparna Akula\anaconda3\lib\site-packages\seaborn\_decorators.py:3
6: FutureWarning: Pass the following variable as a keyword arg: x. From ver
sion 0.12, the only valid positional argument will be `data`, and passing o
ther arguments without an explicit keyword will result in an error or misin
terpretation.
  warnings.warn(
```



Observations:-

- The data is highly skewed with a value of +12.84 and huge kurtosis value of 194.
- It also tells you that majority of the forest fires do not cover a large area, most of the damaged area is under 50 hectares of land.
- There are 4 outlier instances in our area columns

In [64]:
```
#Outlier points based on area
area_outliers = forest_fires[abs(zscore(forest_fires[target])) >= 3]
area_outliers
```

Out[64]:

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
237	1	2	sep	tue	91.0	129.5	692.6	7.0	18.8	40	2.2	0.0	212.88
238	6	5	sep	sat	92.5	121.1	674.4	8.6	25.1	27	4.0	0.0	1090.84
415	8	6	aug	thu	94.8	222.4	698.6	13.9	27.5	27	4.9	0.0	746.28
479	7	4	jul	mon	89.2	103.9	431.6	6.4	22.6	57	4.9	0.0	278.53

There are 4 rows as outliers condisering with area

Independent Columns

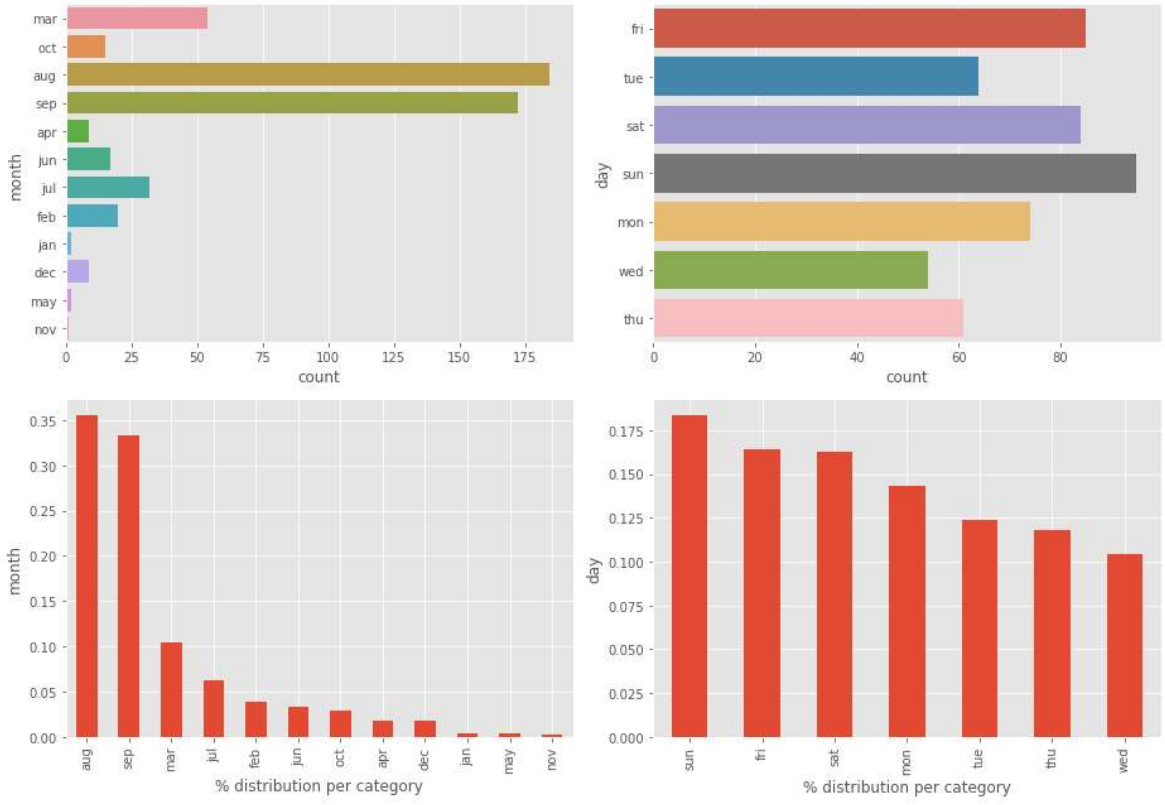
In [65]:
```
forest_fires_new = forest_fires.drop(columns=target)
cat_columns = forest_fires_new.select_dtypes(include='object').
                                                columns.tolist()
num_columns = forest_fires_new.select_dtypes(exclude='object').
                                                columns.tolist()

cat_columns,num_columns
```

Out[65]:
```
(['month', 'day'],
 ['X', 'Y', 'FFMC', 'DMC', 'DC', 'ISI', 'temp', 'RH', 'wind', 'rain'])
```

Categorical Columns

In [66]:
```
#analyzing categorical columns
#Number of ForesFires occured per month and day
plt.figure(figsize=(13,9))
for i,col in enumerate(cat_columns,1):
    plt.subplot(2,2,i)
    sns.countplot(data=forest_fires_new, y=col)
    plt.subplot(2,2,i+2)
    forest_fires[col].value_counts(normalize=True).plot.bar()
    plt.ylabel(col)
    plt.xlabel('% distribution per category')
plt.tight_layout()
plt.show()
```



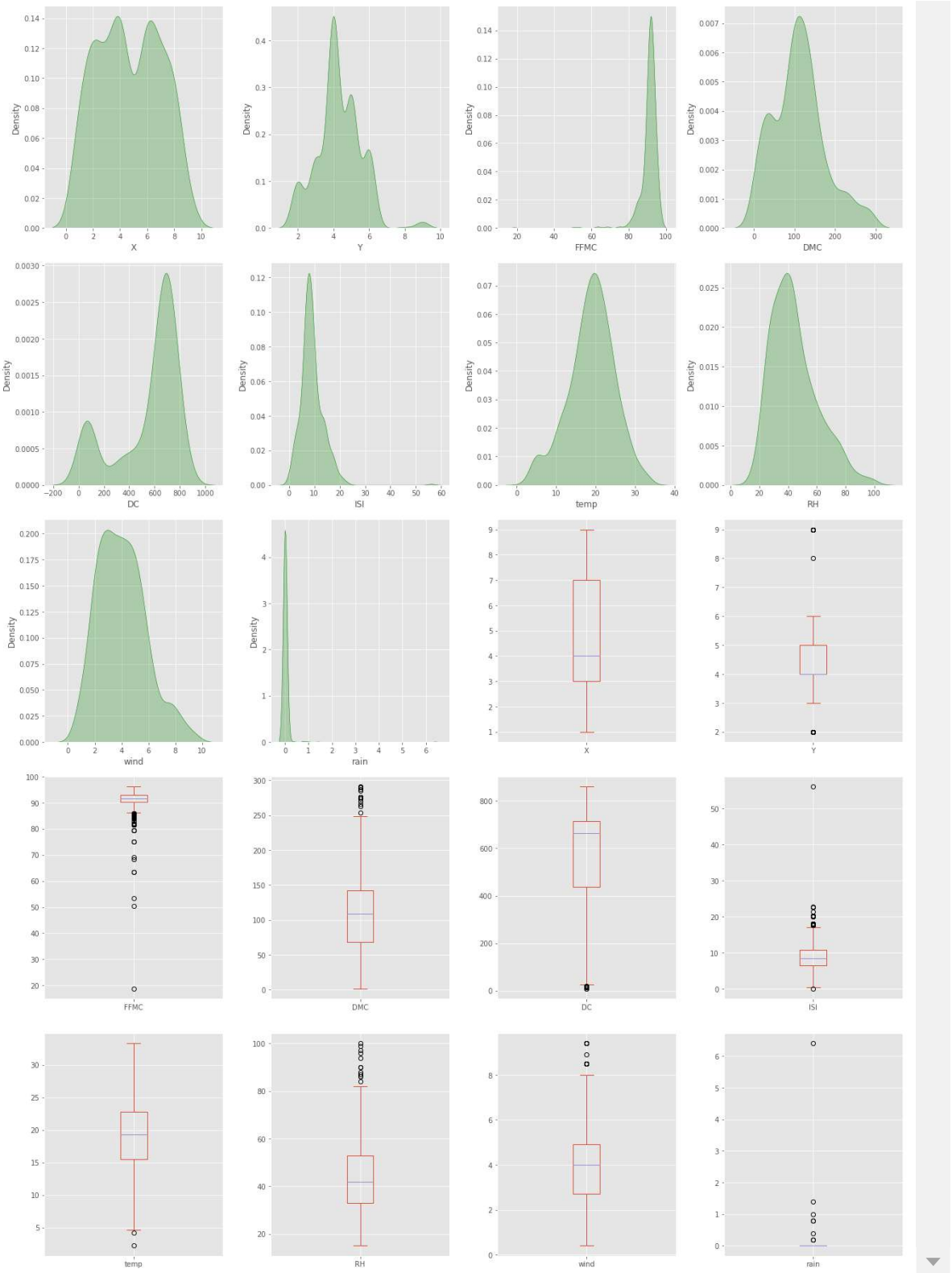
Observations:-

- The highest number of fires occured in the month of August and September.
- In case of days, sunday and fridays have more number of fires.

Numerical Columns

In [67]:

```
plt.figure(figsize=(18,40))
for i,col in enumerate(num_columns,1):
    plt.subplot(8,4,i)
    sns.kdeplot(forest_fires[col],color='g',shade=True)
    plt.subplot(8,4,i+10)
    forest_fires[col].plot.box()
plt.tight_layout()
plt.show()
num_data = forest_fires[num_columns]
pd.DataFrame(data=[num_data.skew(),num_data.kurtosis()],
                      index=['skewness','kurtosis'])
```



Out[67]:

	X	Y	FFMC	DMC	DC	ISI	temp	RH
skewness	0.036246	0.417296	-6.575606	0.547498	-1.100445	2.536325	-0.331172	0.862904
kurtosis	-1.172331	1.420553	67.066041	0.204822	-0.245244	21.458037	0.136166	0.438183

Observations:-

- Skewness and Kurtosis (high positive and negative) was observed in the following columns:

- FFMC
- ISI
- Rain

2.Bivariate Analysis on Target Variable

In [68]:
```
print(forest_fires['area'].describe(),'\n')
print(area_outliers)
```

```
count     517.000000
mean       12.847292
std        63.655818
min         0.000000
25%         0.000000
50%         0.520000
75%         6.570000
max      1090.840000
Name: area, dtype: float64

     X  Y month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain     ar
ea
237  1  2   sep  tue  91.0  129.5  692.6   7.0  18.8  40   2.2   0.0   212.
88
238  6  5   sep  sat  92.5  121.1  674.4   8.6  25.1  27   4.0   0.0  1090.
84
415  8  6   aug  thu  94.8  222.4  698.6  13.9  27.5  27   4.9   0.0   746.
28
479  7  4   jul  mon  89.2  103.9  431.6   6.4  22.6  57   4.9   0.0   278.
53
```

In [69]:
```
# a categorical variable based on forest fire area damage
# No damage, low, moderate, high, very high
def area_cat(area):
    if area == 0.0:
        return "No damage"
    elif area <= 1:
        return "low"
    elif area <= 25:
        return "moderate"
    elif area <= 100:
        return "high"
    else:
        return "very high"
forest_fires['damage_category'] = forest_fires['area'].
                                    apply(area_cat)
forest_fires.head()
```

Out[69]:

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area	damage_category
0	7	5	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.0	No damage
1	7	4	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.0	No damage
2	7	4	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.0	No damage
3	8	6	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.0	No damage
4	8	6	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.0	No damage

adding a categorical variable called "damage_category" this is based on the area burnt. So if the burnt area is equal to zero it is categorized as "No damage", the burnt area is less than or equal to 1 it is "low", if the brunt area is less than or equal to 25 it is "moderate" and if the brunt area is less than or equal to 100 it is "high"

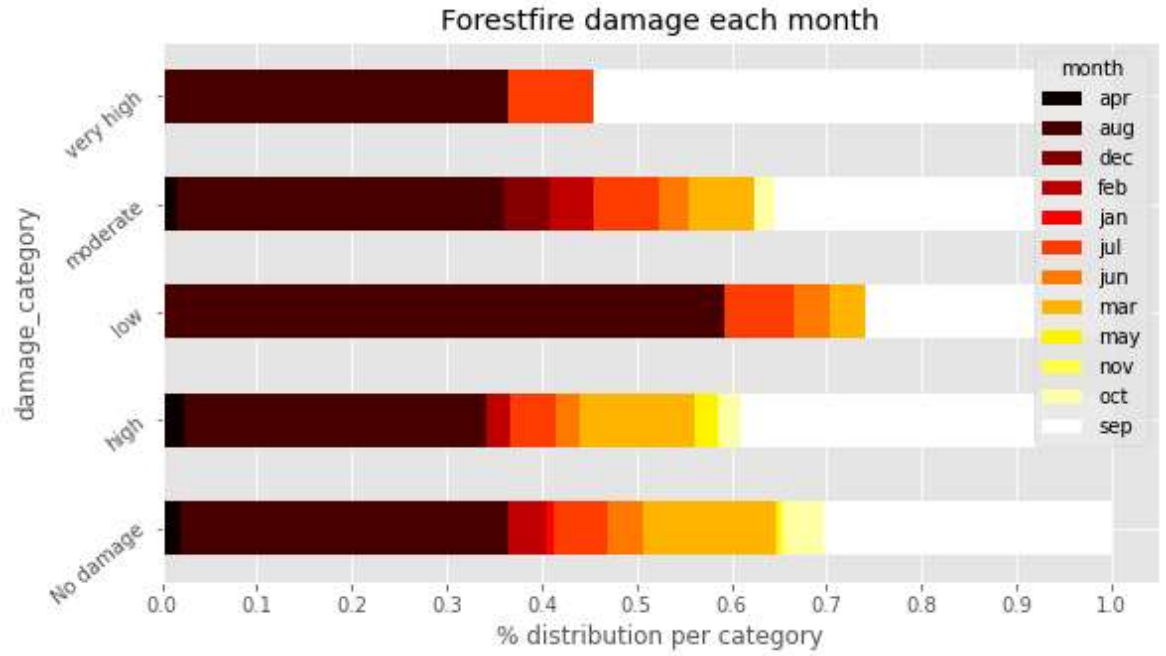
Categorical Columns

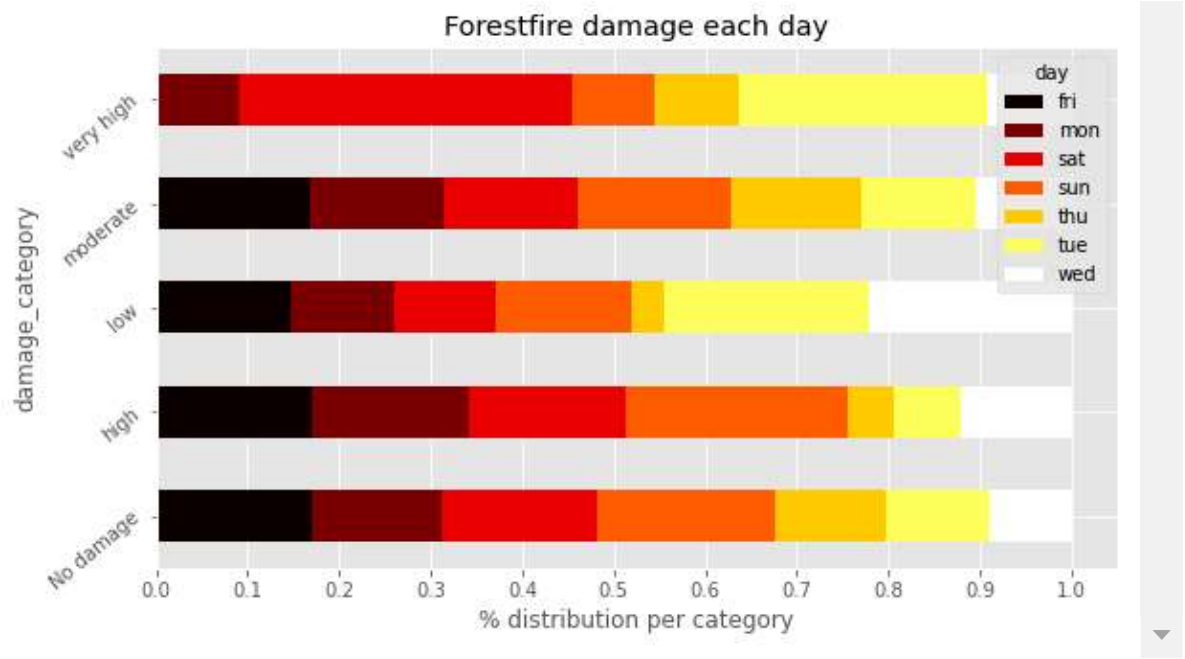
In [70]:
```
cat_columns
```

Out[70]:
```
['month', 'day']
```

In [71]:

```
#Forestfire damage each month and each day
for col in cat_columns:
    cross = pd.crosstab(index=forest_fires['damage_category'],
                columns=forest_fires[col],normalize='index')
    cross.plot.barh(stacked=True,rot=40,cmap='hot')
    plt.xlabel('% distribution per category')
    plt.xticks(np.arange(0,1.1,0.1))
    plt.title("Forestfire damage each {}".format(col))
plt.show()
```





Observations:-

- In the previous observations, we can see that highest number of fires are occured in the month of August and September. From the above plot of month, the below two are the things we can understand
  - Most of the area for the fires occured in August is low less than 1 hectare
  - Also, we can infer that highest damage to the area is done in July, August and September that is more that 100 hectares
- Regarding the fire damage per day, there were no very high damaging fires on Friday and on Saturdays it has been reported the most.

Numerical Columns

In [72]:
```
plt.figure(figsize=(20,40))
for i,col in enumerate(num_columns,1):
    plt.subplot(10,1,i)
    if col in ['X','Y']:
        sns.swarmplot(data=forest_fires,x=col,y=target,
                                hue='damage_category')
    else:
        sns.scatterplot(data=forest_fires,x=col,y=target,
                                  hue='damage_category')
plt.show()
```

```
1296: UserWarning: 46.9% of the points cannot be placed; you may want to
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
C:\Users\Aparna Akula\anaconda3\lib\site-packages\seaborn\categorical.py:
1296: UserWarning: 79.8% of the points cannot be placed; you may want to
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
C:\Users\Aparna Akula\anaconda3\lib\site-packages\seaborn\categorical.py:
1296: UserWarning: 72.8% of the points cannot be placed; you may want to
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
C:\Users\Aparna Akula\anaconda3\lib\site-packages\seaborn\categorical.py:
1296: UserWarning: 54.1% of the points cannot be placed; you may want to
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

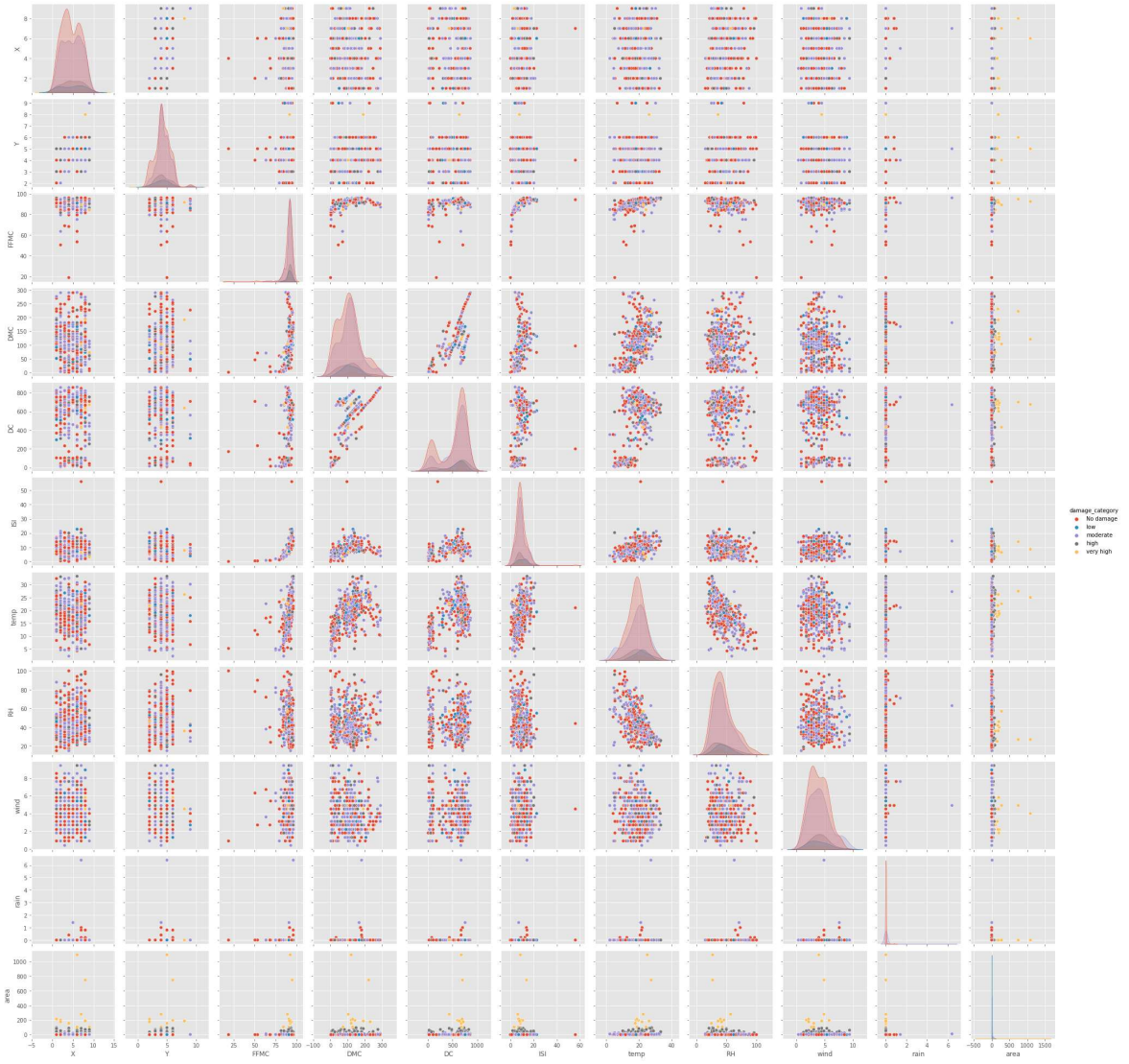
3.Multivariate Analysis

In [73]:
```
selected_features = forest_fires.drop(columns=
        ['damage_category','day','month']).columns
selected_features
```

Out[73]:
```
Index(['X', 'Y', 'FFMC', 'DMC', 'DC', 'ISI', 'temp', 'RH', 'wind', 'rain',
       'area'],
      dtype='object')
```

In [74]:

```
sns.pairplot(forest_fires,hue='damage_category',
             vars=selected_features)
plt.show()
```



Type *Markdown* and LaTeX: α^2

Outlier Treatment

We had observed outliers in the following columns: area FFMC ISI rain

In [75]:
```
out_columns = ['area','FFMC','ISI','rain']
```

However, the above outliers are not error values so we cannot remove it. In order to minimize the effect of outliers in our model we will transform the above features.

Preparing the data for Modelling

In [76]:
```
#encoding categorical columns
df1_ff = pd.get_dummies(forest_fires,columns=
                ['day','month'],drop_first=True)
```

In [77]:
```
#data transformations like log,root,inverse,exponential etc.,
print(df1_ff[out_columns].describe())
np.log1p(df1_ff[out_columns]).skew(),
        np.log1p(df1_ff[out_columns]).kurtosis()
```

```
              area        FFMC         ISI        rain
count   517.000000  517.000000  517.000000  517.000000
mean     12.847292   90.644681    9.021663    0.021663
std      63.655818    5.520111    4.559477    0.295959
min       0.000000   18.700000    0.000000    0.000000
25%       0.000000   90.200000    6.500000    0.000000
50%       0.520000   91.600000    8.400000    0.000000
75%       6.570000   92.900000   10.800000    0.000000
max    1090.840000   96.200000   56.100000    6.400000
```

Out[77]:
```
(area     1.217838
 FFMC   -11.675394
 ISI     -0.937218
 rain    14.173028
 dtype: float64,
 area      0.945668
 FFMC    185.482383
 ISI       2.584588
 rain    234.240025
 dtype: float64)
```

In [78]:
```
# FFMC and rain both have very high skew and kurtosis values, since we are us
# so for FFMC we can remove the outliers in them using z-score method
mask = df1_ff.loc[:,['FFMC']].apply(zscore).abs()<3

# Since most of the values in rain are 0.0, we can convert it as a categorica
df1_ff['rain'] = df1_ff['rain'].apply(lambda x: int(x > 0.0))

df1_ff = df1_ff[mask.values]
df1_ff.shape
```

Out[78]:
```
(510, 29)
```

In [79]:
```
out_columns.remove('rain')
df1_ff[out_columns] = np.log1p(df1_ff[out_columns])
df1_ff[out_columns].skew()
```

Out[79]:
```
area    1.208492
FFMC   -1.803993
ISI    -0.434372
dtype: float64
```

In [80]:
```
# we will use this dataframe for building our ML model
df_ml = df1_ff.drop(columns=['damage_category']).copy()
```

Linear Regression

we will be working on Linear regression using both Statistical and Machine learning approach.

In [81]:
```
X = df_ml.drop(columns=['area'])
y = df_ml['area']
```

1.Statistical Approach

Checking assumptions for linear regression in statistics

Linearity of model

Normality of residuals

Homoscedasticity

No Autocorrelation

Multicollinearity

In [82]:
```
X_constant = sm.add_constant(X)

# Build OLS model
lin_reg = sm.OLS(y,X_constant).fit()
lin_reg.summary()
```

```
C:\Users\Aparna Akula\anaconda3\lib\site-packages\statsmodels\tsa\tsatools.
py:142: FutureWarning: In a future version of pandas all arguments of conca
t except for the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

Out[82]:

OLS Regression Results

Dep. Variable:	area	**R-squared:**	0.077
Model:	OLS	**Adj. R-squared:**	0.025
Method:	Least Squares	**F-statistic:**	1.489
Date:	Sat, 23 Apr 2022	**Prob (F-statistic):**	0.0558
Time:	18:20:10	**Log-Likelihood:**	-874.85
No. Observations:	510	**AIC:**	1806.
Df Residuals:	482	**BIC:**	1924.
Df Model:	27		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.3192	18.275	0.017	0.986	-35.590	36.228
X	0.0532	0.033	1.621	0.106	-0.011	0.118
Y	-0.0115	0.061	-0.187	0.852	-0.132	0.109
FFMC	-0.1061	4.160	-0.025	0.980	-8.280	8.068
DMC	0.0041	0.002	2.166	0.031	0.000	0.008
DC	-0.0019	0.001	-1.440	0.150	-0.004	0.001
ISI	-0.1039	0.290	-0.358	0.720	-0.674	0.466
temp	0.0443	0.023	1.964	0.050	-2.77e-05	0.089
RH	0.0041	0.007	0.624	0.533	-0.009	0.017
wind	0.0678	0.039	1.719	0.086	-0.010	0.145
rain	-0.9272	0.547	-1.695	0.091	-2.002	0.148
day_mon	0.1076	0.230	0.467	0.641	-0.345	0.560
day_sat	0.3312	0.222	1.493	0.136	-0.105	0.767
day_sun	0.1794	0.215	0.836	0.403	-0.242	0.601
day_thu	0.0714	0.243	0.294	0.769	-0.406	0.549
day_tue	0.3483	0.238	1.464	0.144	-0.119	0.816
day_wed	0.1960	0.248	0.791	0.429	-0.291	0.683
month_aug	0.2450	0.847	0.289	0.772	-1.419	1.909

month_dec	2.2223	0.804	2.764	0.006	0.643	3.802
month_feb	0.2217	0.568	0.391	0.696	-0.894	1.337
month_jan	-0.8605	1.483	-0.580	0.562	-3.775	2.054
month_jul	0.0319	0.731	0.044	0.965	-1.405	1.469
month_jun	-0.3316	0.679	-0.488	0.625	-1.666	1.002
month_mar	-0.2613	0.519	-0.503	0.615	-1.282	0.759
month_may	0.6256	1.106	0.565	0.572	-1.548	2.800
month_nov	-1.1779	1.490	-0.790	0.430	-4.106	1.750
month_oct	0.7718	1.005	0.768	0.443	-1.203	2.747
month_sep	0.8978	0.949	0.946	0.345	-0.967	2.762

Omnibus:	76.076	**Durbin-Watson:**	0.979
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	107.018
Skew:	1.074	**Prob(JB):**	5.77e-24
Kurtosis:	3.652	**Cond. No.**	1.89e+05

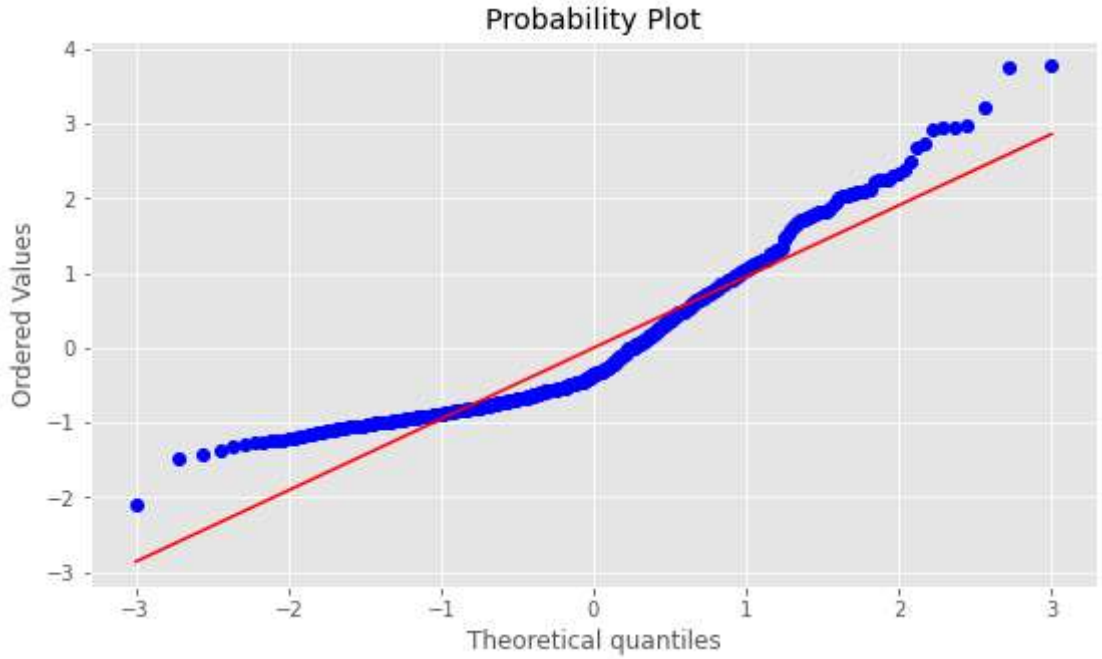
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.89e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

1. Linearity of Residuals

- Linearity can be measured by two methods:-
  - Rainbow test
  - Plot the observed values Vs predicted values and plot the Residual Vs predicted values and see the linearity of residuals.

In [83]:
```
#Rainbow Test
import scipy.stats as stats
import pylab

# get an instance of Influence with influence and outlier measures
st_resid = lin_reg.get_influence().resid_studentized_internal
stats.probplot(st_resid,dist="norm",plot=pylab)
plt.show()
```



- Null hypothesis (H0): The Null hypothesis is that the regression is correctly modeled as linear.
- Alternate hypothesis(H1): The model is non-linear

In [84]:
```
# return fstat and p-value
sm.stats.diagnostic.linear_rainbow(lin_reg)
```

Out[84]: (1.2832659161650852, 0.027048891500426695)

In [85]:
```
# The mean expected value around 0, it implies linearity is preserved
lin_reg.resid.mean()
```

Out[85]:
```
1.8829382497642654e-14
```

Linearity Test - Function for visually inspecting the assumption of linearity in a linear regression model. It plots observed vs. predicted values and residuals vs. predicted values.
Args

```
model - fitted OLS model from statsmodels
y - observed values
```

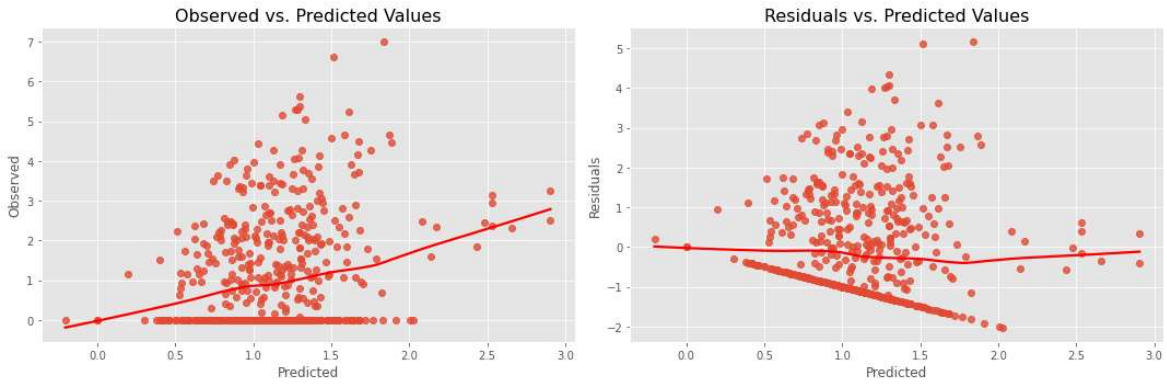
In [86]:
```
def linearity_test(model, y):
    fitted_vals = model.predict()
    resids = model.resid

    fig, ax = plt.subplots(1,2,figsize=(15,5))

    sns.regplot(x=fitted_vals, y=y, lowess=True,
                ax=ax[0], line_kws={'color': 'red'})
    ax[0].set_title('Observed vs. Predicted Values', fontsize=16)
    ax[0].set(xlabel='Predicted', ylabel='Observed')

    sns.regplot(x=fitted_vals, y=resids, lowess=True,
                ax=ax[1], line_kws={'color': 'red'})
    ax[1].set_title('Residuals vs. Predicted Values', fontsize=16)
    ax[1].set(xlabel='Predicted', ylabel='Residuals')

linearity_test(lin_reg, y)
plt.tight_layout()
```



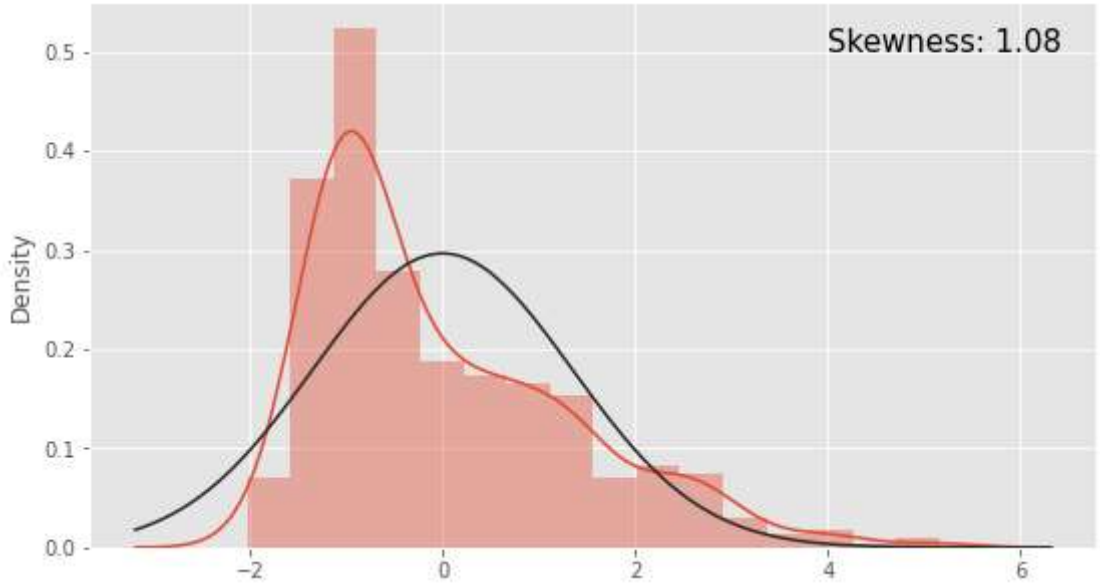
The desired outcome of plots is that points are symmetrically distributed around a diagonal line in the former plot or around horizontal line in the latter one.

- By observing the plots the linearity assumption is not there
- Adding new features might result in linearity of model
- Also, transforming the feature from non-linear to linear using various data transformation techniques can help.

2.Normality of the Residuals

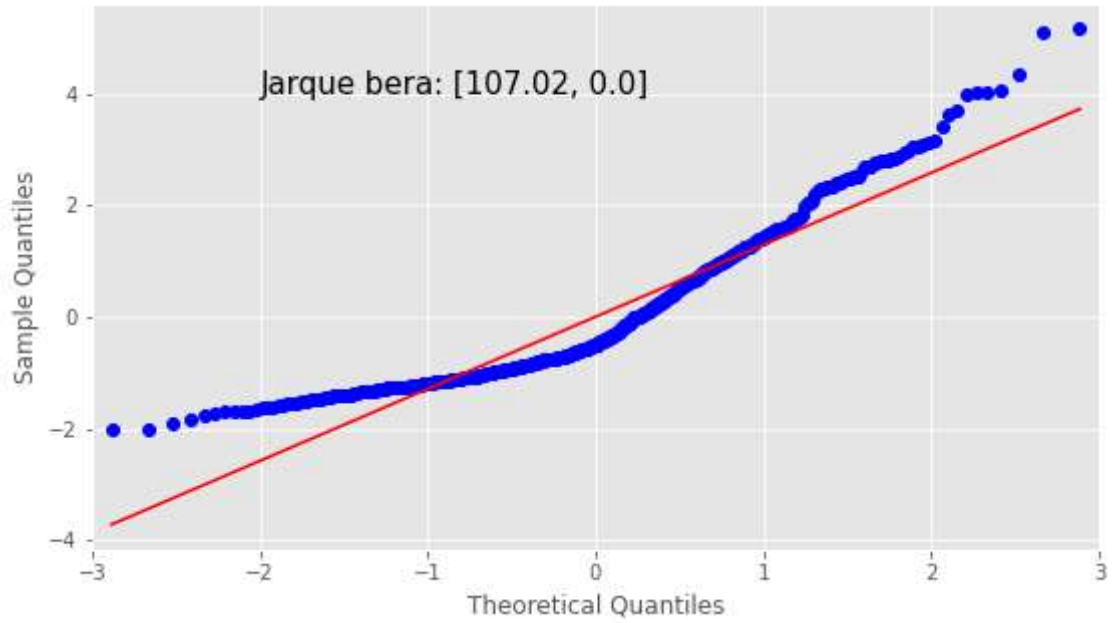
In [87]:
```
sns.distplot(lin_reg.resid,fit=stats.norm)
plt.text(4,0.5,f"Skewness: {round(lin_reg.resid.skew(),2)}",
                                                fontsize=15)
plt.show()
```

```
C:\Users\Aparna Akula\anaconda3\lib\site-packages\seaborn\distributions.py:
2619: FutureWarning: `distplot` is a deprecated function and will be remove
d in a future version. Please adapt your code to use either `displot` (a fi
gure-level function with similar flexibility) or `histplot` (an axes-level
function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [88]:
```
sm.qqplot(lin_reg.resid,line ='r')
jb = [round(n,2) for n in stats.jarque_bera(lin_reg.resid)]
plt.text(-2,4,f"Jarque bera: {jb}",fontsize=15)
plt.show()
```

```
C:\Users\Aparna Akula\anaconda3\lib\site-packages\statsmodels\graphics\gofp
lots.py:993: UserWarning: marker is redundantly defined by the 'marker' key
word argument and the fmt string "bo" (-> marker='o'). The keyword argument
will take precedence.
  ax.plot(x, y, fmt, **plot_style)
```



Test for normality: Jarque Bera

For a good model, the residuals should be normally distributed. The higher the value of Jarque Bera test, the lesser the residuals are normally distributed.

The Jarque–Bera test is a goodness-of-fit test of whether sample data have the skewness and kurtosis matching a normal distribution.

Jarque-Bera (JB): 104.39

The jarque bera test tests whether the sample data has the skewness and kurtosis matching a normal distribution.

Note that this test generally works good for large enough number of data samples(>2000) as the test statistics asymptotically has a chi squared distribution with degrees 2 of freedom.

Our dataframe length, 517

Null hypothesis (H0) - Residuals are normally distributed

- The p-value is 0 which simply means we can reject out NULL hypothesis. We can fix that by
  - Removing the outliers in the data
  - Fixing the Non-linearity in our dependent or target feature
  - Removing the bias, the bias might be contributing to the non-normality.

3.Homoscedasticity

Homoscedacity: If the residuals are symmetrically distributed across the trend , then it is called as homoscedacious.

Heteroscedacity: If the residuals are not symmetric across the trend, then it is called as heteroscedacious.

Goldfeld-Quandt test for Homoscedasticity

H0 = constant variance among residuals (Homoscedacity)

Ha = Heteroscedacity.

In [89]:
```
sms.het_goldfeldquandt(lin_reg.resid, lin_reg.model.exog)
```

Out[89]:
```
(0.9005330268628142, 0.7860123901512497, 'increasing')
```

In [90]:

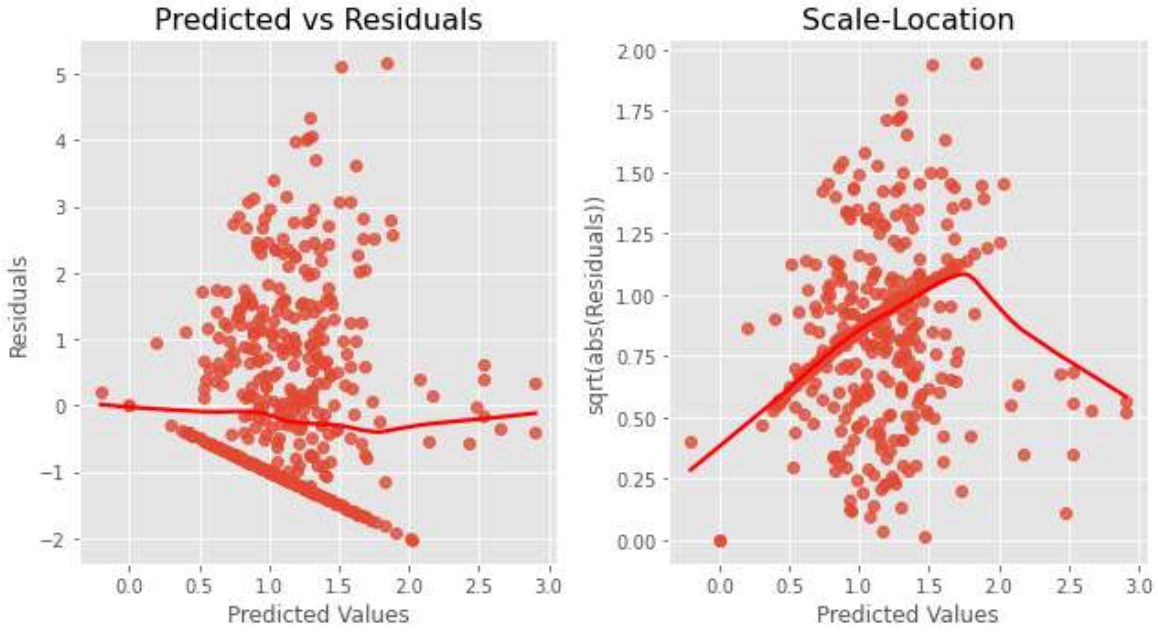
```
model = lin_reg
fitted_vals = model.predict()
resids = model.resid
resids_standardized = model.get_influence().resid_studentized_internal

fig, ax = plt.subplots(1,2)

sns.regplot(x=fitted_vals, y=resids, lowess=True,
            ax=ax[0], line_kws={'color': 'red'})
ax[0].set_title('Predicted vs Residuals', fontsize=16)
ax[0].set(xlabel='Predicted Values', ylabel='Residuals')

sns.regplot(x=fitted_vals, y=np.sqrt(np.abs(resids_standardized)),
            lowess=True, ax=ax[1], line_kws={'color': 'red'})
ax[1].set_title('Scale-Location', fontsize=16)
ax[1].set(xlabel='Predicted Values', ylabel='sqrt(abs(Residuals))')

name = ['F statistic', 'p-value']
test = sms.het_goldfeldquandt(model.resid, model.model.exog)
lzip(name, test)
plt.tight_layout()
```



- To identify homoscedasticity in the plots, the placement of the points should be equally distributed, random, no pattern (increase/decrease in values of residuals) should be visible and a flat red line.
- In the plots we can see there are no paticular patterns and P-Values is also greater than 0.05 ,so we can say that there is homoscedasticity.
- Outliers can make it Heteroscedacious, Transforming (log or Box cox, if > 0) the dependent or independent variables can help fix it.

4.No Autocorrelation Autocorrelation measures the relationship between a variable's current value and its past values.

Test for autocorrelation : Durbin- Watson Test

It's test statistic value ranges from 0-4. If the value is between

- 0-2, it's known as Positive Autocorrelation.
- 2-4, it is known as Negative autocorrelation.
- exactly 2, it means No Autocorrelation.

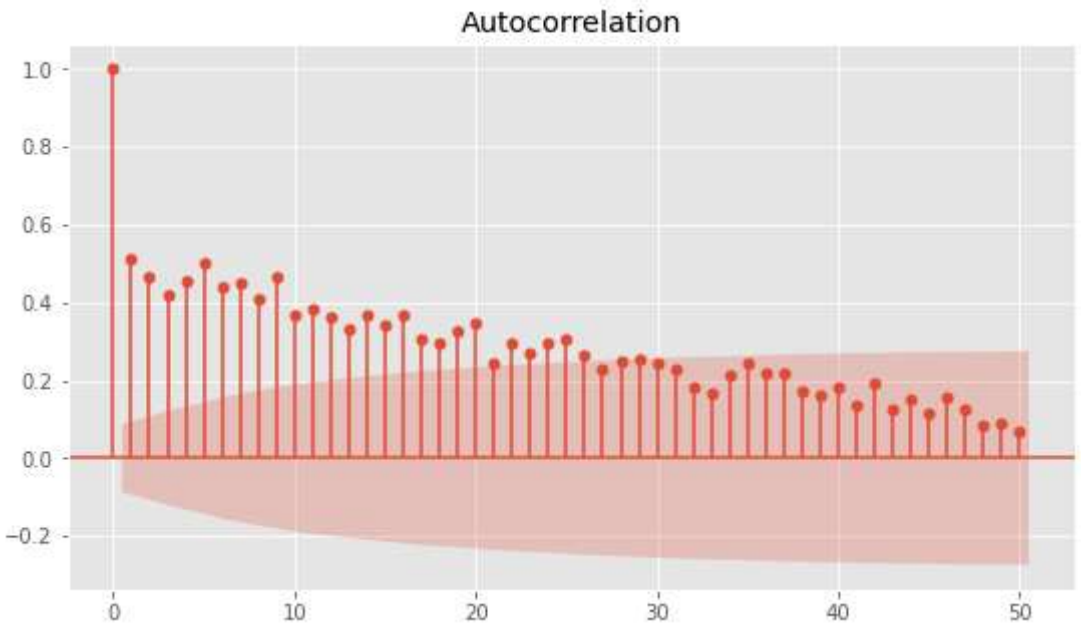
For a good linear model, it should have low or no autocorrelation.

In [91]:
```
from statsmodels.stats.stattools import durbin_watson
durbin_watson(lin_reg.resid)
```

Out[91]:
```
0.979385011086449
```

In [92]:
```
import statsmodels.tsa.api as smt
# Confidence intervals are drawn as a cone.
# By default, this is set to a 95% confidence interval,
# suggesting that correlation values outside of this code are very likely a c
# and not a statistical fluke
acf = smt.graphics.plot_acf(lin_reg.resid, lags=50 , alpha=0.05)
acf.show()
```

```
C:\Users\APARNA~1\AppData\Local\Temp/ipykernel_17064/2880793929.py:7: UserW
arning: Matplotlib is currently using module://matplotlib_inline.backend_in
line, which is a non-GUI backend, so cannot show the figure.
  acf.show()
```



- By observing the above data we can say that there is positive autocorrelation is present , we can reduce it by using fine tuning our parameters
- We can even use Generalize Least Squares (GLS) model

5.Multicollinearity

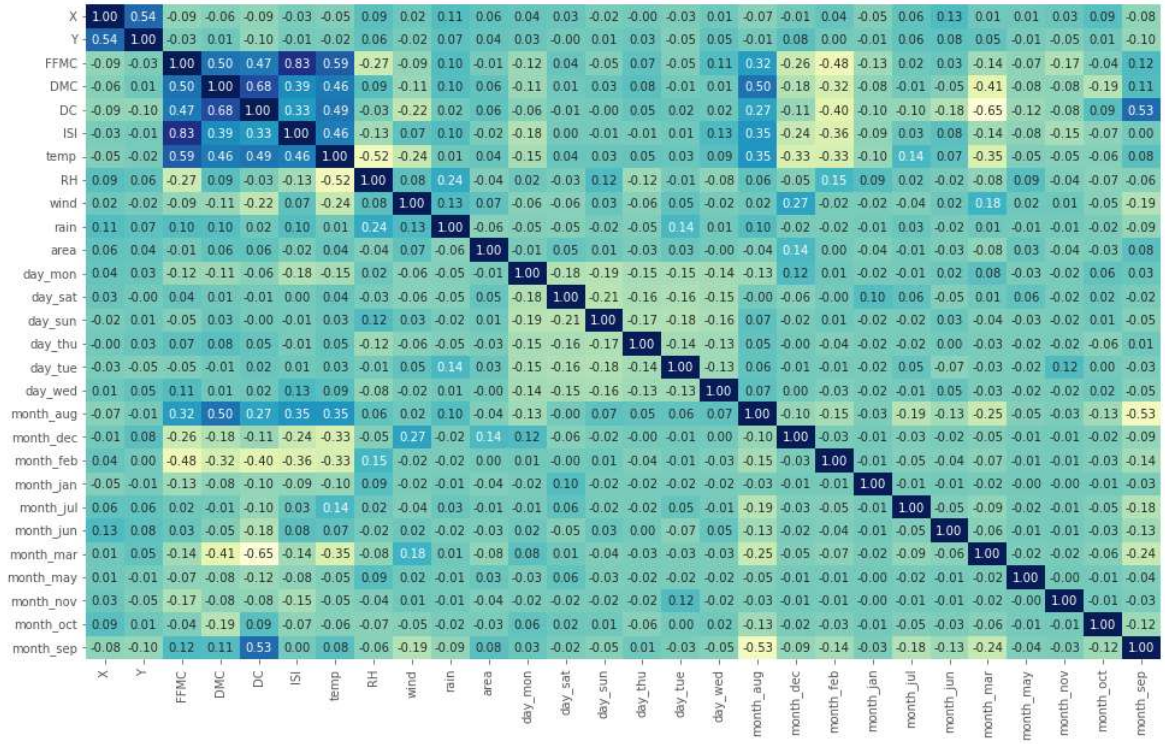
Multicollineariy arises when one independent variable can be linearly predicted by others with a substantial level of accuracy.

substantial level of accuracy.

In [93]:

```
plt.figure(figsize =(16,10))

sns.heatmap(df1_ff.corr(),annot=True,cmap='YlGnBu',fmt=".2f",cbar=False)
plt.show()
```



In [94]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = [variance_inflation_factor(X_constant.values, i)
                  for i in range(X_constant.shape[1])]
pd.DataFrame({'vif': vif[1:]}, index=X.columns).
            sort_values(by="vif",ascending=False)
```

Out[94]:

	vif
month_sep	53.307716
month_aug	43.939403
DC	26.792896
month_jul	8.378570
month_oct	7.681340
month_mar	6.694845
FFMC	5.629386
temp	4.535721
ISI	4.107793
DMC	3.978913
month_jun	3.731938
month_feb	3.077978
month_dec	2.984761
RH	2.870672
day_sun	1.829250
day_sat	1.750570
day_mon	1.733149
day_thu	1.654243
day_tue	1.653962
day_wed	1.547815
X	1.539358
Y	1.521475
wind	1.323007
month_may	1.273889
rain	1.231386
month_nov	1.157750
month_jan	1.146912

There is multicollinearity present between some features where vif >5.

- We can even use PCA to reduce features to a smaller set of uncorrelated components.

- To deal with multicollinearity we should iteratively remove features with high values of VIF.

2.Machine Learning Approach

In [95]:
```
lr = LinearRegression()
lr.fit(X, y)

print(f'Intercept: {lr.intercept_}')
print(f'R^2 score: {lr.score(X, y)}')
pd.DataFrame({"Coefficients": lr.coef_}, index=X.columns)
```

```
Intercept: 0.319206835515748
R^2 score: 0.07698615338292181
```

Out[95]:

	Coefficients
X	0.053161
Y	-0.011457
FFMC	-0.106058
DMC	0.004149
DC	-0.001854
ISI	-0.103922
temp	0.044297
RH	0.004063
wind	0.067775
rain	-0.927225
day_mon	0.107572
day_sat	0.331167
day_sun	0.179442
day_thu	0.071364
day_tue	0.348268
day_wed	0.196002
month_aug	0.244968
month_dec	2.222281
month_feb	0.221676
month_jan	-0.860523
month_jul	0.031948
month_jun	-0.331633
month_mar	-0.261304
month_may	0.625607
month_nov	-1.177853
month_oct	0.771841
month_sep	0.897800

Improving Stats Model

Dropping columns to improve accuracy:

By checking high Variance inflation factor and p-value we will decide whether to keep the column or drop it.

R^2 = 1 - SSE(Sum of Square of Residuals)/SST (Sum of square Total)

Just by dropping constant we got a huge bump in adjusted R2 from 2.5% to 40.6%.

In [96]:
```
X = df1_ff.drop(columns=['area','damage_category'])
y = df1_ff['area']
```

In [97]:

```
def check_stats(X,y):
    vif = [variance_inflation_factor(X.values, i) for
                                i in range(X.shape[1])]
    print(pd.DataFrame({'vif': vif}, index=X.columns).
              sort_values(by="vif",ascending=False)[:10])
    lin_reg = sm.OLS(y,X).fit()
    print(lin_reg.summary())
check_stats(X,y)
```

```
                 vif
FFMC      202.334784
DC        159.585267
month_sep  77.976513
month_aug  67.081874
temp       53.012135
ISI        52.449380
RH         23.764931
Y          19.961996
DMC        15.417494
month_jul   8.764939
                                OLS Regression Results
==============================================================================
==============
Dep. Variable:                    area   R-squared (uncentered):
0.438
Model:                             OLS   Adj. R-squared (uncentered):
0.406
Method:                  Least Squares   F-statistic:
13.92
Date:                 Sat, 23 Apr 2022   Prob (F-statistic):
7.32e-45
Time:                         18:20:17   Log-Likelihood:
-874.85
No. Observations:                 510   AIC:
1804.
Df Residuals:                     483   BIC:
1918.
Df Model:                          27
Covariance Type:            nonrobust
==============================================================================
=====
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-----
X              0.0532      0.033      1.632      0.103      -0.011
0.117
Y             -0.0115      0.061     -0.187      0.852      -0.132
0.109
FFMC          -0.0335      0.192     -0.174      0.862      -0.412
0.345
DMC            0.0041      0.002      2.219      0.027       0.000
0.008
DC            -0.0019      0.001     -1.445      0.149      -0.004
0.001
ISI           -0.1077      0.195     -0.551      0.582      -0.491
```

```
0.276
temp           0.0443      0.022      1.969      0.050    8.25e-05
0.088
RH             0.0041      0.006      0.641      0.522      -0.008
0.017
wind           0.0679      0.039      1.750      0.081      -0.008
0.144
rain          -0.9289      0.538     -1.728      0.085      -1.985
0.127
day_mon        0.1074      0.230      0.467      0.641      -0.344
0.559
day_sat        0.3310      0.221      1.496      0.135      -0.104
0.766
day_sun        0.1795      0.214      0.838      0.403      -0.242
0.601
day_thu        0.0710      0.242      0.294      0.769      -0.404
0.546
day_tue        0.3486      0.237      1.473      0.141      -0.116
0.814
day_wed        0.1959      0.247      0.792      0.429      -0.290
0.682
month_aug      0.2428      0.837      0.290      0.772      -1.401
1.887
month_dec      2.2214      0.801      2.772      0.006       0.647
3.796
month_feb      0.2224      0.566      0.393      0.694      -0.889
1.334
month_jan     -0.8589      1.479     -0.581      0.562      -3.765
2.047
month_jul      0.0302      0.723      0.042      0.967      -1.391
1.452
month_jun     -0.3332      0.673     -0.495      0.621      -1.655
0.989
month_mar     -0.2629      0.511     -0.514      0.607      -1.267
0.741
month_may      0.6241      1.102      0.566      0.571      -1.541
2.789
month_nov     -1.1762      1.486     -0.792      0.429      -4.095
1.743
month_oct      0.7690      0.990      0.776      0.438      -1.177
2.715
month_sep      0.8952      0.936      0.956      0.339      -0.944
2.735
==============================================================================
Omnibus:                       76.079   Durbin-Watson:                   1.
979
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              107.
024
Skew:                           1.074   Prob(JB):                     5.76e-
24
Kurtosis:                       3.653   Cond. No.                     2.07e+
04
==============================================================================

Notes:
```

```
[1] R² is computed without centering (uncentered) since the model does no
t contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[3] The condition number is large, 2.07e+04. This might indicate that the
re are
strong multicollinearity or other numerical problems.
```

In [98]:
```
# dropping FFMC, Y, month_jul, day_thu, day_mon, month_aug as P-value is high
X.drop(columns=['FFMC'],inplace=True)
X.drop(columns=['Y'],inplace=True)
X.drop(columns=['month_jul'],inplace=True)
X.drop(columns=['day_thu'],inplace=True)
X.drop(columns=['day_mon'],inplace=True)
X.drop(columns=['month_aug'],inplace=True)
check_stats(X,y)
```

```
                vif
DC        57.707223
ISI       38.292380
temp      27.680452
DMC       13.223716
RH        12.130774
wind       7.359276
X          5.267220
month_sep  3.915822
month_mar  2.741742
month_feb  1.631273
                                 OLS Regression Results
==============================================================================
============
Dep. Variable:                    area   R-squared (uncentered):
0.437
Model:                             OLS   Adj. R-squared (uncentered):
0.413
Method:                  Least Squares   F-statistic:
18.07
Date:                 Sat, 23 Apr 2022   Prob (F-statistic):
2.20e-48
Time:                         18:20:17   Log-Likelihood:
-875.21
No. Observations:                 510   AIC:
1792.
Df Residuals:                     489   BIC:
1881.
Df Model:                          21
Covariance Type:            nonrobust
==============================================================================
===
                 coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
X              0.0488      0.027      1.823      0.069      -0.004       0.
101
DMC            0.0039      0.002      2.275      0.023       0.001       0.
007
DC            -0.0014      0.001     -1.848      0.065      -0.003    8.99e
-05
ISI           -0.1152      0.166     -0.694      0.488      -0.441       0.
211
temp           0.0406      0.016      2.512      0.012       0.009       0.
072
RH             0.0032      0.005      0.698      0.486      -0.006       0.
```

```
012
wind           0.0660      0.038      1.757      0.079      -0.008       0.
140
rain          -0.9250      0.514     -1.801      0.072      -1.934       0.
084
day_sat        0.2728      0.182      1.498      0.135      -0.085       0.
631
day_sun        0.1318      0.175      0.752      0.452      -0.213       0.
476
day_tue        0.2961      0.201      1.472      0.142      -0.099       0.
691
day_wed        0.1418      0.213      0.666      0.506      -0.277       0.
560
month_dec      2.0388      0.525      3.885      0.000       1.008       3.
070
month_feb      0.1796      0.403      0.446      0.656      -0.612       0.
971
month_jan     -0.8810      1.421     -0.620      0.536      -3.673       1.
911
month_jun     -0.4236      0.400     -1.059      0.290      -1.210       0.
363
month_mar     -0.3034      0.313     -0.970      0.332      -0.918       0.
311
month_may      0.5821      1.023      0.569      0.570      -1.428       2.
592
month_nov     -1.2378      1.403     -0.882      0.378      -3.994       1.
519
month_oct      0.4906      0.448      1.096      0.274      -0.389       1.
370
month_sep      0.6113      0.209      2.930      0.004       0.201       1.
021
==============================================================================
===
Omnibus:                       78.346   Durbin-Watson:                   0.
978
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              111.
420
Skew:                           1.091   Prob(JB):                     6.39e
-25
Kurtosis:                       3.693   Cond. No.                     1.45e
+04
==============================================================================
===

Notes:
[1] R² is computed without centering (uncentered) since the model does not
contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is corr
ectly specified.
[3] The condition number is large, 1.45e+04. This might indicate that there
are
strong multicollinearity or other numerical problems.
```

Similarly, you can continue to optimize the model.

Our Prob (F-statistic) has improved from 7.32e-45 to 2.20e-48. As the value is less than 0.05, the model becomes more significant.

Improving ML model

Feature Selection techniques

The following can be used for selecting relevant features for model building

- Using Pearson Correlation
- Wrapper method
  - Forward Selection: Forward selection is an iterative method in which we start with having no feature in the model. In each iteration, we keep adding the feature which best improves our model till an addition of a new variable does not improve the performance of the model.
  - Backward Elimination: In backward elimination, we start with all the features and removes the least significant feature at each iteration which improves the performance of the model. We repeat this until no improvement is observed on removal of features.
  - Recursive Feature elimination: It is a greedy optimization algorithm which aims to find the best performing feature subset. It repeatedly creates models and keeps aside the best or the worst performing feature at each iteration. It constructs the next model with the left features until all the features are exhausted. It then ranks the features based on the order of their elimination.
- Embedded method, lasso is one such method which penalizes features based on feature importance, making lesss important feature to 0.

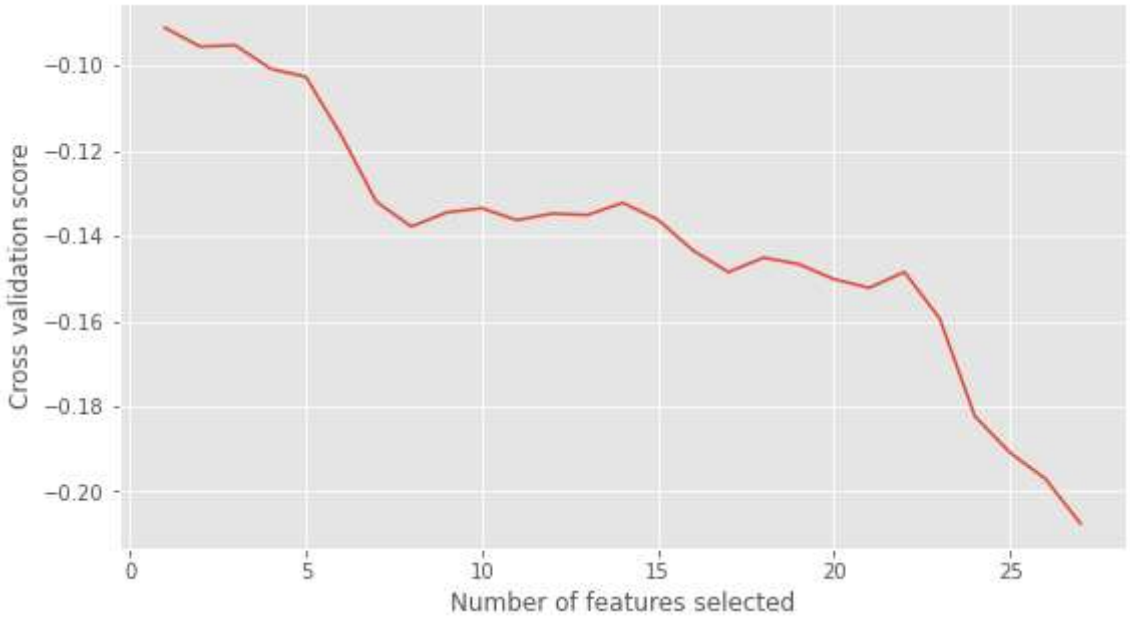
In [99]:
```
X_m, y_m = df_ml.drop(columns=[target]), df_ml[target]
```

Recursive Feature Elimination

In [100]:

```
# RFECV is a variant with inbuilt Cross validation
model = LinearRegression()
selector = RFECV(model,cv=5)
selector = selector.fit(X_m, y_m)
print(f"Out of {len(X_m.columns)} features,
      best number of features {selector.n_features_}")
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score")
plt.plot(range(1, len(X_m.columns) + 1), selector.grid_scores_)
print(X_m.columns[selector.support_].values)
plt.show()
```

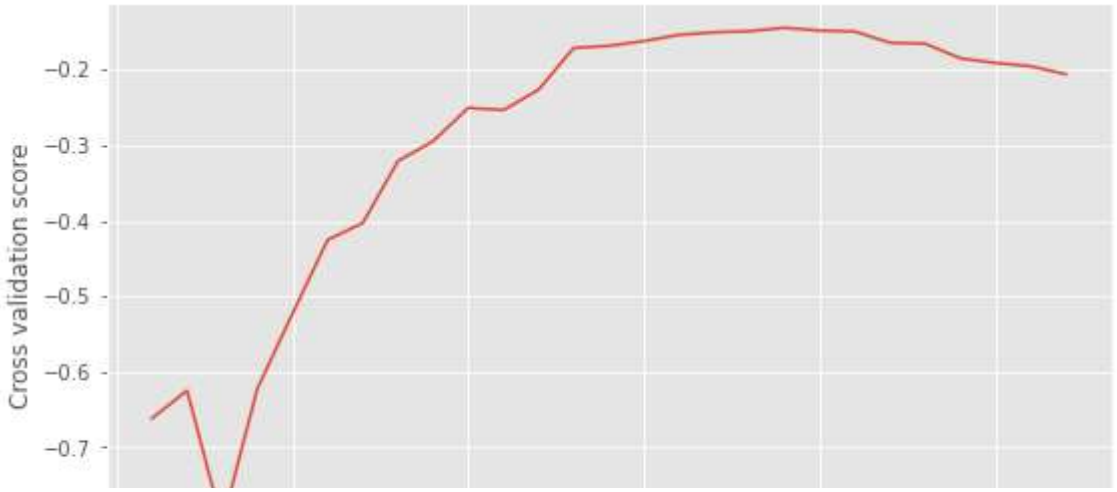
```
Out of 27 features, best number of features 1
['month_dec']
```



In [101]:
```
# In our stats method we found that the intercept was not relevant
# Let's try that feature out in our ML model
model = LinearRegression(fit_intercept=False)
selector = RFECV(model,cv=5)
selector = selector.fit(X_m, y_m)
print(f"Out of {len(X_m.columns)} features,
      best number of features {selector.n_features_}")

plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score")
plt.plot(range(1, len(X_m.columns) + 1), selector.grid_scores_)
print(X_m.columns[selector.support_].values)
plt.show()
```

```
Out of 27 features, best number of features 19
['X' 'wind' 'rain' 'day_mon' 'day_sat' 'day_sun' 'day_thu' 'day_tue'
 'day_wed' 'month_aug' 'month_dec' 'month_feb' 'month_jan' 'month_jul'
 'month_jun' 'month_mar' 'month_may' 'month_nov' 'month_oct']
```



Building model with the best features and checking the R2 score for the same

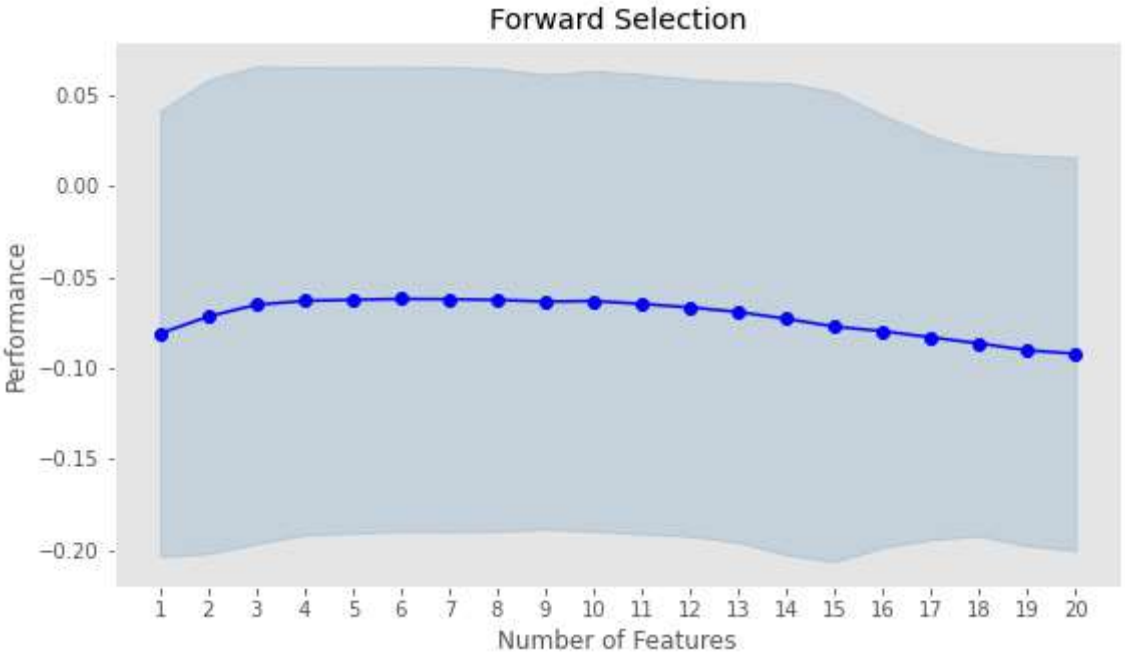
In [102]:
```
mask = selector.support_
print(f"Best features according to RFE {X_m.columns[mask].values}")

X_m1 = X_m.iloc[:,mask]
# We could have used train test split or cross validation strategies
# for scoring the model but in order to compare with the stats model
# we will use the whole data
model1 = LinearRegression().fit(X_m1,y_m)
print(f"R2 Score: {model1.score(X_m1,y_m)}")
```

```
Best features according to RFE ['X' 'wind' 'rain' 'day_mon' 'day_sat' 'day_
sun' 'day_thu' 'day_tue'
 'day_wed' 'month_aug' 'month_dec' 'month_feb' 'month_jan' 'month_jul'
 'month_jun' 'month_mar' 'month_may' 'month_nov' 'month_oct']
R2 Score: 0.0554282443680163
```

Forward Selection

In [103]:
```
model = LinearRegression(fit_intercept=False)
sfs1 = sfs(model,k_features=20,forward=True,scoring='r2',cv=5)
sfs1.fit(X_m,y_m)
fig = plot_sfs(sfs1.get_metric_dict())
plt.title('Forward Selection')
plt.grid()
plt.show()
```



In [104]:
```
print(sfs1.k_features, sfs1.k_feature_names_,sep="\n")
```

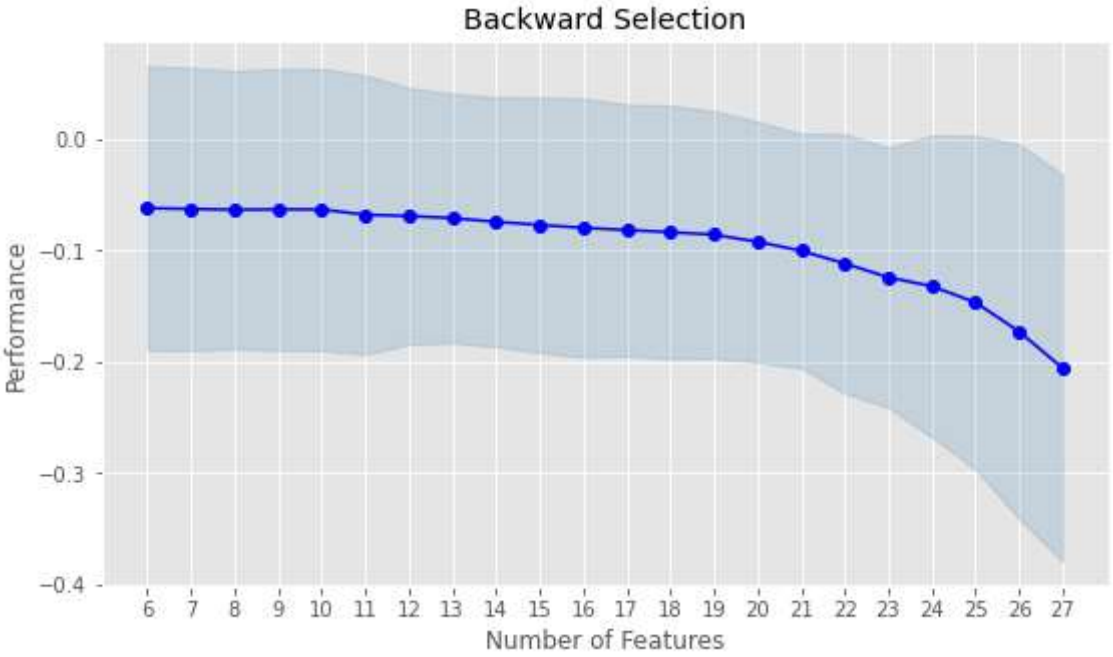
```
20
('X', 'FFMC', 'ISI', 'RH', 'wind', 'rain', 'day_mon', 'day_sat', 'day_sun',
'day_thu', 'day_tue', 'day_wed', 'month_aug', 'month_dec', 'month_feb', 'mo
nth_jan', 'month_jul', 'month_mar', 'month_nov', 'month_oct')
```

In [105]:
```
index = list(sfs1.k_feature_idx_)
X_m1 = X_m.iloc[:,index]
model1 = LinearRegression().fit(X_m1,y_m)
print(f"R2 Score: {model1.score(X_m1,y_m)}")
```

```
R2 Score: 0.05472198851071042
```

Backward Selection

In [106]:
```
model = LinearRegression(fit_intercept=False)
sfs1 = sfs(model,k_features=6,forward=False,scoring='r2',cv=5)
sfs1.fit(X_m,y_m)
fig = plot_sfs(sfs1.get_metric_dict())
plt.title('Backward Selection')
plt.grid(True)
plt.show()
```



In [107]:
```
index = list(sfs1.k_feature_idx_)
print(f"Best features according to RFE: {X_m.columns[index]}")

X_m1 = X_m.iloc[:,index]
model1 = LinearRegression().fit(X_m1,y_m)
print(f"R2 Score: {model1.score(X_m1,y_m)}")
```

```
Best features according to RFE: Index(['X', 'FFMC', 'rain', 'month_jan', 'm
onth_mar', 'month_nov'], dtype='object')
R2 Score: 0.01821264603161088
```

Overall Summary:-

- After doing Univariate Analysis as the target variable as Area,

```
- we can determine that the data is highly skewed and it has huge ku
rtosis value,
- also most of the damaged area is under 50 hectares of land for maj
ority of the forest fires occured.
- Skewness and Kurtosis was observed in FFMC, ISI and Rain
```

- The highest number of fires occured in the month of August and September.
- In case of days, sunday and fridays have more number of fires.
- Later to understand the data more I did a Bivariate Analysis, by adding a categorical variable called "damange_category"

```
- the area brunt in August beacuse of forest fires is less than 1 he
ctare
- the area brunt in July, August and September is more that 100 hect
are, this is where the highest damage is occured to           area
- there were no very high damaging fires on Friday and on Saturdays,
so the area brunt is less
```

- FFMC and rain both have very high skew and kurtosis values, since we are using linear regression model we cannot operate with high such values. So for FFMC we can remove the outliers in them using z-score method and rain is taken care by using lambda functions.
- I did Linear Regression using both statistical and machine learning approach, here I didnt train any databecause we are trying to predict the burnt area with the existing columns itself.

```
- By default, statsmodels fits a line passing through the origin, i.
e. it doesn't fit an intercept.
- Hence, used 'add_constant' so that it also fits an intercept.
- R squared of the statistical model is 0.077 and F-statistic value
 is 1.489.
```

- Checking assumptions for linear regression in statistics.

1. Linearity of model:- used rainbow test and plotted observed values Vs predicted values, residual Vs predicted values. From the rainbow test the mean value is not around zero and through plots we can observe that the plots are not linear.
2. Normality of the Residuals:- Normality can be verified by conducting a Jarque Bera Test.

```
              The higher the value of Jarque Bera test, the lesser t
he residuals are normally distributed.
              The value of Jarque-Bera test is 104.39, meaning the r
esiduals are not normally distributed.
```

3. Homoscedasticity:- used Goldfeld-Quandt test for HomoscedasticityIn the plots we can see there are no paticular patterns and P-Values is also greater than 0.05, so we can say that there is homoscedasticity.
4. No Autocorrelation:- Test for autocorrelation is by Durbin- Watson Test. With a value of 0.97 the model is considered to be in a positive autocorelation.
5. Multicollinearity:- Multicollinearity is calculated using VIF, the values with VIF greater than 5 are condidered to be multicollinear.

- R squared value using Machine Learning approach is 0.07

- Improved the stats model by dropping high Variance inflation factor and p-value columns. then the R squared value is 0.438 and F-statistic value is 13 which is considered to be very good model.
- Improved machine learning model by implementing wrappers methods,
  - using Recursive Feature Elimination - derived the 19 best features and R squared value is 0.055
  - using Forwrad Selection - 20 best features were derived with a R squared value of 0.054
  - using Backwrad Selection - 6 best features were derived with a R squared value of 0.018
- Hence the improved stats model is the best model.

Reference :-

https://www.datarobot.com/blog/ordinary-least-squares-in-python/ (https://www.datarobot.com/blog/ordinary-least-squares-in-python/) https://www.jmp.com/en_us/statistics-knowledge-portal/what-is-regression/simple-linear-regression-assumptions.html (https://www.jmp.com/en_us/statistics-knowledge-portal/what-is-regression/simple-linear-regression-assumptions.html) https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/assumptions-of-linear-regression/ (https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/assumptions-of-linear-regression/) https://www.analyticsvidhya.com/blog/2020/07/univariate-analysis-visualization-with-illustrations-in-python/ (https://www.analyticsvidhya.com/blog/2020/07/univariate-analysis-visualization-with-illustrations-in-python/) https://www.analyticsvidhya.com/blog/2022/02/a-quick-guide-to-bivariate-analysis-in-python/ (https://www.analyticsvidhya.com/blog/2022/02/a-quick-guide-to-bivariate-analysis-in-python/) https://www.statsmodels.org/dev/generated/statsmodels.stats.diagnostic.het_goldfeldquandt.html (https://www.statsmodels.org/dev/generated/statsmodels.stats.diagnostic.het_goldfeldquandt.html) https://www.statology.org/durbin-watson-test-python/ (https://www.statology.org/durbin-watson-test-python/) https://stackoverflow.com/questions/58410187/how-to-plot-predicted-values-vs-the-true-value (https://stackoverflow.com/questions/58410187/how-to-plot-predicted-values-vs-the-true-value) https://www.statology.org/jarque-bera-test-python/ (https://www.statology.org/jarque-bera-test-python/) https://machinelearningmastery.com/rfe-feature-selection-in-python/ (https://machinelearningmastery.com/rfe-feature-selection-in-python/) https://towardsdatascience.com/feature-selection-with-pandas-e3690ad8504b (https://towardsdatascience.com/feature-selection-with-pandas-e3690ad8504b) https://machinelearningmastery.com/rfe-feature-selection-in-python/ (https://machinelearningmastery.com/rfe-feature-selection-in-python/) https://fivestepguide.com/technology/machine-learning/backward-elimination-code-in-python-0321/ (https://fivestepguide.com/technology/machine-learning/backward-elimination-code-in-python-0321/)