Name:	Bikis Muhammed
Email:	bmpb8@mst.edu
Course:	CS 5402
Assignment:	Exploratory Data Analysis
Course:	CS 5402

MISSOURI S&T

In [140]:
```
# For using csv file
import pandas as pd

# For ploting the frequency of common words
import matplotlib.pylab as plt

import seaborn as sb

import numpy as ny

from sklearn.impute import SimpleImputer

from sklearn.model_selection import train_test_split

import sklearn.feature_selection

from  sklearn.linear_model import LinearRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn import metrics
```

Concept Description

- The main objective of this project to make an exporative data analysis on collected data on forest fires and make a attribute prediction. First, we will try to get to know our data attributes, and then clean up the data to make it suitable for further processing. Finally, we will figure which variable more corrolate each other using different techniques.

In [*]:
```
# data = pd.read_csv("../src-data/forestfires.csv")
# data.index = ny.arange(1, len(data)+1)
# data
```

Data Collection

- The data has provided to our firm as a excel (.csv)file. - The given data set has 13 attributes. Those attributes are coord_X, coord_Y, month, day, FFMC, DMC, DC, ISI, temp, RH, wind, rain, and area. Each attribute will be explained in the example discription section. Only day and month attaributes are catagorical (ordinal), and the rest are numerical level of measurements.
- In addtion, the data set has 517 examples.

Example Description

Level of measurements

Level of measurements	Discription
Nominal	Just a label name and no sense of order E.g. True/False - No order -There is no distance between any two values -Having absense of True or False does not mean anything.
Ordinal	Name (Nominal) + Order A named label and then ordering E.g. : Minimum -> medium -> Maximum
Interval	Nominal + Ordinal + Fixed distance between each attribute values Or They can not be catagorical attribute Any numeric attribute that is and ordinal, but it can not have a true zero. E.g. Tempreture in degree celcious (65 -> 70 -> 75)
Ratio	Nominal + Ordinal + Interval + Meaningful zero value E.g Number of students.

Attribute	Discription	Level of measurements	Possible Range
coord_X	x-axis spacial coordinate within a typographical map	Ratio	0 to ∞
coord_Y	y-axis of spacial coordinate within a typographical map	Ratio	0 to ∞
month	the month in which forst fire happened	Ordinal	Jan - Dec (1 - 12)
day	the day when forest fire happened	Ordinal	Mon - Sun (1 - 7)
FFMC	Final Fuel Moisture Code from the Fire Weather Index (FWI) system. According to Wood, this measurement merge/affected by different measurements such as rainfall, wind speed, relative humidity, rain, and tempreture in order to find the the level of moisture in a forest floor, and estimate how its possiblity of expanding once egnited.	Interval	0 to 99
DMC	Duff Moisture Code from the FWI system "It relates rainfall, tempreture, and relative humidity." (Wood) It measures the moisture content of loosely-compacted organic layers of moderate depth. (William J. De Groot)	Ratio	0 to 350
DC	Draught Code from the FWI system "it Combines/affected by rainfall and temperature." (Wood) It measures/indicates the moisture, content in deep, and compact organic layers	Ratio	0 to 1200
ISI	Initial Spread Index from the FWI system. "It combines FFMC and wind velocity, and it can be used as an indicator of how quickly a fire is likely to spread." (Wood) ACcroding Willam, A wind speed increase by 13km/h will double ISI.	Ratio	0 to ∞
temp	tempreture in ^oC	Interval	0 to 100
RH	relative humidity in $percent(\%)$	Interval	0 to 100
wind	speed of the wind in km/h	Ratio	0 to ∞
rain	amout of rain in mm/m^2	Ratio	0 to ∞
area	burned are in hectars $(10000m^2)$	Ratio	0 to ∞

Data Import and Wrangling

In [27]:
```
data = pd.read_csv("../src-data/forestfires.csv")
data
```

Out[27]:

	coord_X	coord_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	0.00
1	7	4	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.00
2	7	4	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.00
3	8	6	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.00
4	8	6	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.00
515	1	4	aug	sat	94.4	146.0	614.7	11.3	25.6	42	4.0	0.0	0.00
516	6	3	nov	tue	79.5	3.0	106.7	1.1	11.8	31	4.5	0.0	0.00

517 rows × 13 columns

Exploratory Data Analysis

This part includes data preparation and cleaning of a data. It has three parts.

1. Data visualization and understanding
2. Cleaning of the data
3. Finding a relationship or corolation between two or more attributes.

1. Data visualization and understanding

Day and month will not be included since they categorical attribites.

In [29]:
```
(data.describe()).T
```

Out[29]:

	count	mean	std	min	25%	50%	75%	max
coord_X	517.0	4.669246	2.313778	1.0	3.00	4.00	7.00	9.00
coord_Y	517.0	4.299807	1.229900	2.0	4.00	4.00	5.00	9.00
FFMC	517.0	92.091296	37.111003	9.9	90.20	91.60	92.90	921.00
DMC	517.0	110.872340	64.046482	1.1	68.60	108.30	142.40	291.30
DC	517.0	547.940039	248.066192	7.9	437.70	664.20	713.90	860.60
ISI	517.0	9.021663	4.559477	0.0	6.50	8.40	10.80	56.10
temp	515.0	18.895922	5.815985	2.2	15.55	19.30	22.80	33.30
RH	517.0	44.288201	16.317469	15.0	33.00	42.00	53.00	100.00
wind	517.0	4.017602	1.791653	0.4	2.70	4.00	4.90	9.40
rain	517.0	0.021663	0.295959	0.0	0.00	0.00	0.00	6.40
area	517.0	12.847292	63.655818	0.0	0.00	0.52	6.57	1090.84

In [37]:
```
data['month'].describe()
```
Out[37]:
```
count      517
unique      12
top        aug
freq       184
Name: month, dtype: object
```

In [34]:
```
data['day'].describe()
```
Out[34]:
```
count      517
unique       7
top        sun
freq        95
Name: day, dtype: object
```

Minimum and maximum value of each attribute in our data

In [38]:
```
data.describe().loc[['min','max']].T
```
Out[38]:

	min	max
coord_X	1.0	9.00
coord_Y	2.0	9.00
FFMC	9.9	921.00
DMC	1.1	291.30
DC	7.9	860.60
ISI	0.0	56.10
temp	2.2	33.30
RH	15.0	100.00
wind	0.4	9.40
rain	0.0	6.40
area	0.0	1090.84

```
Note: The FFMC value range should be between 0 to 99. In the above table, the maximum value is 921 that is not in the interval.
If it is only one or two values, the imputation technique will be used to change those values.  If it is many, the whole column
will be dropped.
```

2. Data Cleaning /Tidying

The first step will be changing non-numeric attributes to numeric.

In [39]:
```
for i in ['day', 'month']:
    dummies = pd.get_dummies(data[i], prefix= i, dummy_na=False)
    data = data.drop(i, 1)
    data = pd.concat([data, dummies], axis=1)
```

In [40]:
```
data.head(10).T
```
Out[40]:

	0	1	2	3	4	5	6	7	8	9
coord_X	7.0	7.0	7.0	8.0	8.0	8.0	8.0	8.0	8.0	7.0
coord_Y	5.0	4.0	4.0	6.0	6.0	6.0	6.0	6.0	6.0	5.0
FFMC	86.2	90.6	90.6	91.7	89.3	92.3	92.3	91.5	91.0	92.5
DMC	26.2	35.4	43.7	33.3	51.3	85.3	88.9	145.4	129.5	88.0
DC	94.3	669.1	686.9	77.5	102.2	488.0	495.6	608.2	692.6	698.6
ISI	5.1	6.7	6.7	9.0	9.6	14.7	8.5	10.7	7.0	7.1
temp	8.2	18.0	14.6	8.3	11.4	22.2	24.1	8.0	13.1	22.8
RH	51.0	33.0	33.0	97.0	99.0	29.0	27.0	86.0	63.0	40.0
wind	6.7	0.9	1.3	4.0	1.8	5.4	3.1	2.2	5.4	4.0
rain	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0
area	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
day_fri	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
day_mon	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0
day_sat	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
day_sun	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0
day_thu	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
day_tue	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
day_wed	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
month_apr	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
month_aug	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0
month_dec	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
month_feb	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
month_jan	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
month_jul	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
month_jun	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

month_mar	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
month_may	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
month_nov	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
month_oct	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
month_sep	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0

Check for Missing Value

In [41]:
```
data.isnull().sum().sort_values(ascending=False)
```

Out[41]:
```
temp          2
coord_X       0
day_tue       0
month_oct     0
month_nov     0
month_may     0
month_mar     0
month_jun     0
month_jul     0
month_jan     0
month_feb     0
month_dec     0
month_aug     0
month_apr     0
day_wed       0
day_thu       0
coord_Y       0
day_sun       0
day_sat       0
day_mon       0
day_fri       0
area          0
rain          0
wind          0
RH            0
ISI           0
DC            0
DMC           0
FFMC          0
month_sep     0
dtype: int64
```

Impute Missing Value

In [42]:
```
imput = SimpleImputer(missing_values=ny.nan, strategy='mean')
imput.fit(data)
data = pd.DataFrame(data=imput.transform(data) , columns=data.columns)
```

Check for missing value again

In [43]:
```
data.isnull().sum().sort_values(ascending=False)
```

Out[43]:
```
coord_X       0
coord_Y       0
month_oct     0
month_nov     0
month_may     0
month_mar     0
month_jun     0
month_jul     0
month_jan     0
month_feb     0
month_dec     0
month_aug     0
month_apr     0
day_wed       0
day_tue       0
day_thu       0
day_sun       0
day_sat       0
day_mon       0
day_fri       0
area          0
rain          0
wind          0
RH            0
temp          0
ISI           0
DC            0
DMC           0
FFMC          0
month_sep     0
dtype: int64
```

Data attribute unit conversion

In [44]:
```
# Manual unit conversion using lambda function
# area: 1 hectar  = 10000m2
data['area'].apply(lambda x: x *10000)
# rain: mm/m2 -> I do not think this need conversion
#temp: oC
data['temp'].apply(lambda x: x *1)
#RH: percent(%)
data['RH'].apply(lambda x: x * 1)
# wind: km/h   -> 0.277778 m/s
data['wind'].apply(lambda x: x * 0.277778)
```

Out[44]:
```
0      1.861113
1      0.250000
2      0.361111
3      1.111112
4      0.500000
         ...
512    0.750001
513    1.611112
514    1.861113
515    1.111112
516    1.250001
Name: wind, Length: 517, dtype: float64
```

Make sure all value are within their range

In [45]:
```
data = data[data.FFMC < 99]
```

Corrolation Matrix

In [46]:
```
corr = data.corr()
corr
```

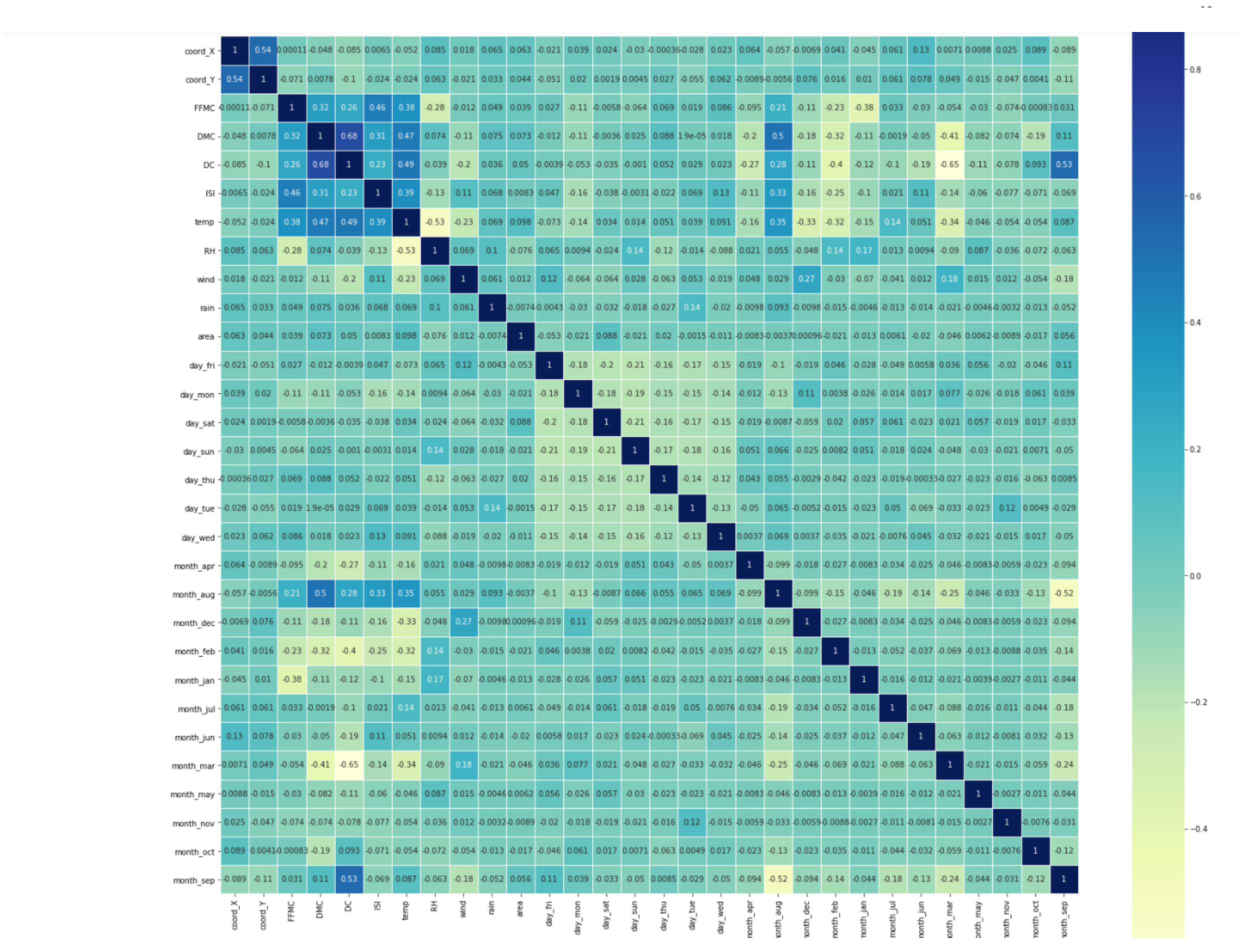
Out[46]:

	coord_X	coord_Y	FFMC	DMC	DC	ISI	temp	RH	wind	rain	...	month_dec	month_feb	month_
coord_X	1.000000	0.537882	0.000110	-0.048435	-0.085083	0.006503	-0.052403	0.085432	0.018300	0.065308	...	-0.006853	0.041328	-0.045
coord_Y	0.537882	1.000000	-0.071361	0.007827	-0.099982	-0.024110	-0.023917	0.062592	-0.021261	0.033081	...	0.075626	0.015684	0.009
FFMC	0.000110	-0.071361	1.000000	0.315966	0.264463	0.457830	0.376249	-0.285000	-0.011597	0.049475	...	-0.112019	-0.231907	-0.380
DMC	-0.048435	0.007827	0.315966	1.000000	0.682309	0.305131	0.468255	0.073795	-0.105345	0.074791	...	-0.176303	-0.317909	-0.105
DC	-0.085083	-0.099982	0.264463	0.682309	1.000000	0.229093	0.494358	-0.039235	-0.203319	0.035928	...	-0.105552	-0.399197	-0.115
ISI	0.006503	-0.024110	0.457830	0.305131	0.229093	1.000000	0.393135	-0.132530	0.106891	0.067688	...	-0.162295	-0.249741	-0.103
temp	-0.052403	-0.023917	0.376249	0.468255	0.494358	0.393135	1.000000	-0.528772	-0.225832	0.069417	...	-0.329942	-0.320403	-0.146
RH	0.085432	0.062592	-0.285000	0.073795	-0.039235	-0.132530	-0.528772	1.000000	0.069434	0.099758	...	-0.047704	0.140452	0.170
wind	0.018300	-0.021261	-0.011597	-0.105345	-0.203319	0.106891	-0.225832	0.069434	1.000000	0.061089	...	0.269661	-0.029524	-0.070
rain	0.065308	0.033081	0.049475	0.074791	0.035928	0.067688	0.069417	0.099758	0.061089	1.000000	...	-0.009771	-0.014727	-0.004
area	0.063074	0.044386	0.038657	0.072998	0.049540	0.008301	0.097723	-0.075506	0.012239	-0.007391	...	0.000965	-0.020802	-0.012
day_fri	-0.021085	-0.050563	0.027053	-0.012007	-0.003853	0.046814	-0.072577	0.064556	0.117918	-0.004324	...	-0.019259	0.046161	-0.027
day_mon	0.038508	0.020230	-0.108241	-0.107934	-0.052672	-0.158528	-0.137141	0.009412	-0.064080	-0.030008	...	0.114434	0.003775	-0.025
day_sat	0.023510	0.001892	-0.005783	-0.003649	-0.034836	-0.038485	0.034331	-0.023837	-0.064013	-0.032340	...	-0.058751	0.020239	0.056
day_sun	-0.030446	0.004467	-0.063740	0.025365	-0.001037	-0.003127	0.013780	0.136290	0.027774	-0.017943	...	-0.025094	0.008234	0.050
day_thu	-0.000362	0.026640	0.069252	0.087687	0.052178	-0.022319	0.050970	-0.123046	-0.062730	-0.026853	...	-0.002932	-0.042427	-0.022
day_tue	-0.028402	-0.055033	0.018604	0.000019	0.028689	0.068713	0.038684	-0.014181	0.053236	0.139278	...	-0.005222	-0.014640	-0.023
day_wed	0.022807	0.061841	0.086468	0.018061	0.022564	0.125746	0.091461	-0.088494	-0.018799	-0.020499	...	0.003686	-0.034867	-0.021
month_apr	0.063617	-0.008918	-0.095328	-0.197545	-0.268153	-0.106449	-0.157284	0.021246	0.048210	-0.009771	...	-0.017751	-0.026754	-0.008
month_aug	-0.056831	-0.005553	0.209335	0.498790	0.278783	0.334902	0.351497	0.054743	0.029238	0.093457	...	-0.098769	-0.148860	-0.046
month_dec	-0.006853	0.075626	-0.112019	-0.176303	-0.105552	-0.162295	-0.329942	-0.047704	0.269661	-0.009771	...	1.000000	-0.026754	-0.008
month_feb	0.041328	0.015684	-0.231907	-0.317909	-0.399197	-0.249741	-0.320403	0.140452	-0.029524	-0.014727	...	-0.026754	1.000000	-0.012
month_jan	-0.045200	0.009961	-0.380971	-0.105646	-0.115034	-0.103574	-0.146653	0.170929	-0.070277	-0.004575	...	-0.008311	-0.012526	1.000
month_jul	0.060570	0.060728	0.033036	-0.001944	-0.100698	0.021047	0.142304	0.013208	-0.040766	-0.013428	...	-0.034259	-0.051633	-0.016
month_jun	0.129375	0.078194	-0.029682	-0.050403	-0.186069	0.111566	0.050789	0.009398	0.012042	-0.013537	...	-0.024592	-0.037064	-0.011
month_mar	0.007073	0.049430	-0.054196	-0.407447	-0.650333	-0.143454	-0.338701	-0.089817	0.181308	-0.020795	...	-0.045551	-0.068652	-0.021
month_may	0.008789	-0.015484	-0.029794	-0.081979	-0.114178	-0.060478	-0.045638	0.086828	0.015027	-0.004575	...	-0.008311	-0.012526	-0.003
month_nov	0.025278	-0.046889	-0.073751	-0.074218	-0.078357	-0.076550	-0.053875	-0.035881	0.011845	-0.003232	...	-0.005871	-0.008849	-0.002
month_oct	0.089276	0.004104	-0.000832	-0.187635	0.093442	-0.071115	-0.053762	-0.072322	-0.053932	-0.012690	...	-0.023054	-0.034746	-0.010
month_sep	-0.088954	-0.108447	0.030943	0.110968	0.532796	-0.068737	0.087207	-0.062566	-0.181893	-0.051859	...	-0.094211	-0.141990	-0.044

Corrolation Chart

In [47]:

```
f, ax = plt.subplots(figsize=(25,25))
sb.heatmap(corr, square = True, annot=True, cmap='YlGnBu', linewidth=0.5)
plt.savefig("../generated-output/corrolation.png")
```



Corolation pairplot Graph

In [*]:
```
f, ax = plt.subplots(figsize=(40,40))
sb.pairplot(data)
plt.savefig("../generated-output/pairplotData.png")
```

Temp, RH, and month_june are selected for analysis.

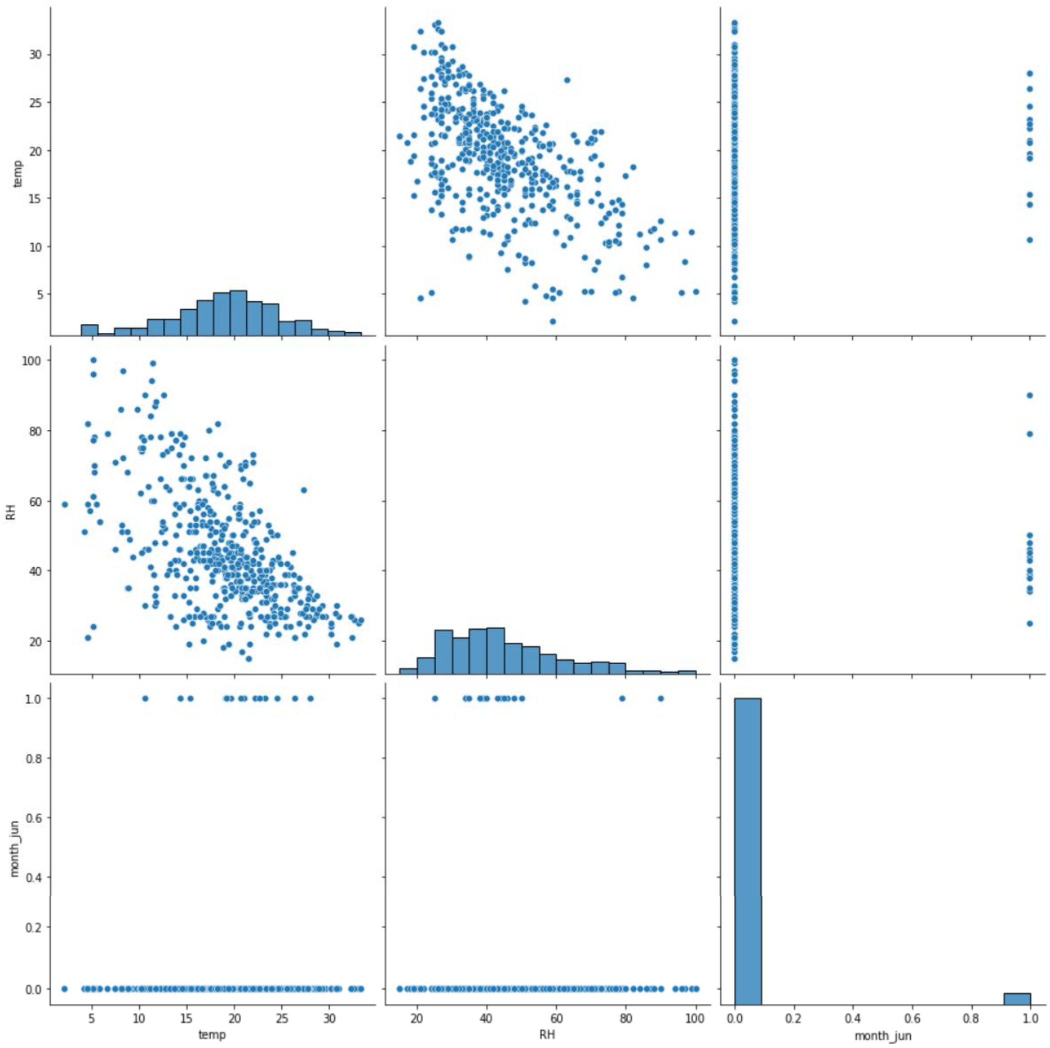
In [16]:
```
cor = corr.abs().unstack()
cor = cor.sort_values(ascending = False)
cor['temp']
```

Out[16]:
```
temp          1.000000
RH            0.528772
DC            0.494358
DMC           0.468255
ISI           0.393135
FFMC          0.376249
month_aug     0.351497
month_mar     0.338701
month_dec     0.329942
month_feb     0.320403
wind          0.225832
month_apr     0.157284
month_jan     0.146653
month_jul     0.142304
day_mon       0.137141
area          0.097723
day_wed       0.091461
month_sep     0.087207
day_fri       0.072577
rain          0.069417
month_nov     0.053875
month_oct     0.053762
coord_X       0.052403
day_thu       0.050970
month_jun     0.050789
month_may     0.045638
day_tue       0.038684
day_sat       0.034331
coord_Y       0.023917
day_sun       0.013780
dtype: float64
```

Note: temp attribute has a data missing problem, and RH has a highest corrolation with tempreture. month_june is an imputed ordinal attrbute.

Pairplot of their corrolation

In [209]:
```
sb.pairplot(data, vars =['temp', 'RH', 'month_jun'], height=4.5)
plt.show()
```



It does not look like a good representation of our data, and lets see farther.

In [211]:
```
Y = data['RH']
X = data[['temp', 'month_jun']]
xtrain, xtest, ytrain, ytest = train_test_split(X, Y, train_size =0.8)
```

In [213]:
```
model = LinearRegression()
model.fit(xtrain, ytrain)
```

Out[213]: LinearRegression()

In [214]:
```
predict = model.predict(xtrain)
```

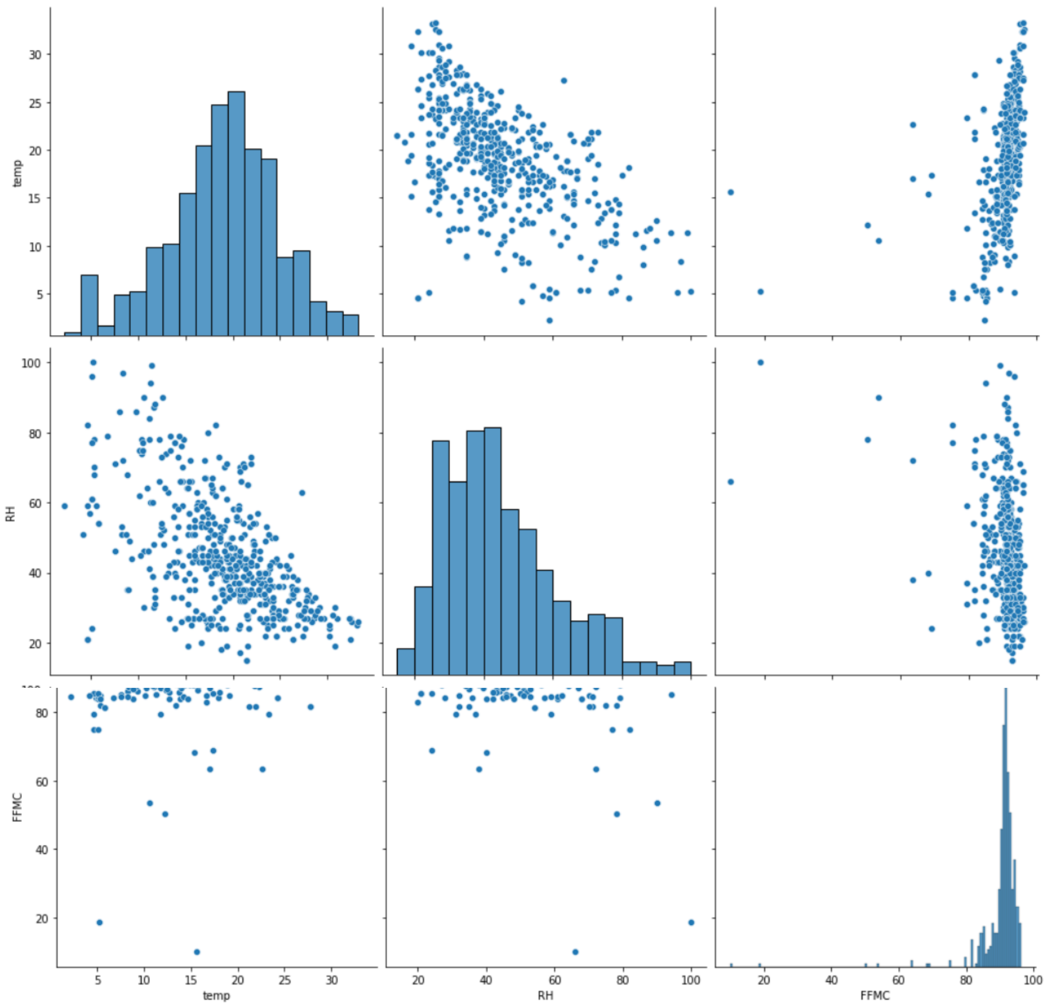
In [215]:
```
accuracy  = ny.sqrt(metrics.mean_squared_error(ytrain, predict))
accuracy
```

Out[215]: 13.465035238553105

Temp, RH, and FFMC are selected for analysis.

- Both temp and RH will be independent variables
  - FFMC will be dependant variablle

In [216]:
```
sb.pairplot(data, vars =['temp', 'RH', 'FFMC'], height=4.5)
# plt.show()
plt.savefig("../generated-output/airplot.png")
```



Mining or Analytics

Spliting training and test set

In [206]:
```
Y = data['FFMC']
X = data[['temp', 'RH']]
```

In [207]:
```
xtrain, xtest, ytrain, ytest = train_test_split(X, Y, train_size =0.8)
```

Creating Multiple Regression Model

In [208]:
```
model = LinearRegression()
model.fit(xtrain, ytrain)
```

Out[208]: LinearRegression()

Make a prediction using the training set

In [204]:
```
predict = model.predict(xtrain)
```

Evaluation

In [205]:
```
Result = pd.DataFrame()
Result['Training Value(ytrain)'] = list(ytrain)
Result['Predicted value'] = list(predict)

# for i, j in zip(list(ytrain),list(predict)):
#     print(i, '- ', j)
Result.to_csv("../generated-output/result.csv", index=False)
Result.head(11)
```

Out[205]:

	Training Value(ytrain)	Predicted value
0	91.6	90.740870
1	87.6	87.543704
2	88.1	89.045987
3	91.0	93.283194
4	92.1	90.933470
5	92.1	89.700828
6	91.5	89.431187
7	95.9	93.899516
8	91.5	87.274063
9	91.4	91.049030
10	90.7	87.736304

Accuracy of the prediction in root mean square

In [152]:
```
accuracy  = ny.sqrt(metrics.mean_squared_error(ytrain, predict))
accuracy
```

Out[152]: 6.051094122179813

Testing using testing set

In [191]:
```
testpredict = model.predict(xtest)
```

In [195]:
```
TestResult = pd.DataFrame()
TestResult['Testing Value(ytest)'] = list(ytest)
TestResult['Predicted value'] = list(testpredict)
TestResult.to_csv("../generated-output/testresult.csv", index=False)
TestResult.head(11)
```

Out[195]:

	Testing Value(ytest)	Predicted value
0	87.6	85.823573
1	91.0	90.661278
2	93.7	93.824413
3	91.6	86.475536
4	90.3	89.523423
5	93.7	93.433251
6	92.4	89.460382
7	96.1	90.822056
8	85.4	86.147255
9	93.5	89.955677
10	89.4	87.908032

In [196]:
```
accuracy  = ny.sqrt(metrics.mean_squared_error(ytest, testpredict))
accuracy
```

Out[196]: 3.829412225984367

In [156]:
```
# plt.scatter(X, Y, color = 'red')
# plt.plot(x, LR.predict(x), color = 'blue')
# plt.savefig("../generated-output/scattergraph.png")
```

Results

The root-mean-square error of the testing data set is 3.83, which is smaller than 6.05. One of the main reasons for this difference is that the training data set is 80% of the whole data. Specifically, 412 examples were leveraged for training, and 104 examples were used for testing. The second reason may be the correlation between each selected element is small. For instance, the correlation between Relative humidity and FFMC (Final Fuel Moisture Code) is small in the correlation table. Generally, my model minimally predicts the correct FFMC value.

Reference

- David A. Wood (2021). Prediction and datamining of burned areas of forest fires: Optimized data matching and mining algorithm provides valuable insight. Retrived (2021, jun 19) from https://doi.org/10.1016/j.aiia.2021.01.004
- Rising Odegua (2018) Exploratory Data Analysis, Feature Engineering, and Modelling using Supermarket Sales Data. Retrived (2021, jun 19) from https://towardsdatascience.com/exploratory-data-analysis-feature-engineering-and-modelling-using-supermarket-sales-data-part-1-228140f89298
- Stackoverflow (2021). Python Libraries. Retrived (2021, Jun 19) from https://stackoverflow.com/
- William J. De Groot (?) INTERPRETING THE CANADIAN FOREST FIRE WEATHER INDEX (FWI) SYSTEM Retrived (2021, Jun 23) from https://www.dnr.state.mi.us/WWW/FMD/WEATHER/Reference/FWI_Background.pdf