project1

February 1, 2021

1 Forest fires dataset (Part 1)

This dataset was gathered in 2007 by Paulo Cortez and Anibal Morais as a regression task to predict forest fires using meteorological data. It has 517 instances, each with 10 numerical and 2 categorical features. For the categorical attributes, month and day, I would suggest a cyclical encoding by assigning a sine and cosine value to each feature so that Jan comes after Dec, etc. There are no missing features. This data set is interesting because it is a difficult regression problem with very applicable real world value. If a model could be trained on this data and applied to a greater area, it would be extremely useful. I believe the month, day, wind, and rainfall attributes will be most useful as they are the best indicators of burn conditions.

[1]:
```
import pandas as pd
import numpy as np

data = pd.read_csv('forestfires.csv')
names = open('forestfires.names', 'r')
print(names.read())
```

```
Citation Request:
  This dataset is public available for research. The details are described in
[Cortez and Morais, 2007].
  Please include this citation if you plan to use this database:

  P. Cortez and A. Morais. A Data Mining Approach to Predict Forest Fires using
Meteorological Data.
  In J. Neves, M. F. Santos and J. Machado Eds., New Trends in Artificial
Intelligence,
  Proceedings of the 13th EPIA 2007 - Portuguese Conference on Artificial
Intelligence, December,
  Guimaraes, Portugal, pp. 512-523, 2007. APPIA, ISBN-13 978-989-95618-0-9.
  Available at: http://www.dsi.uminho.pt/~pcortez/fires.pdf

1. Title: Forest Fires

2. Sources
   Created by: Paulo Cortez and Aníbal Morais (Univ. Minho) @ 2007

3. Past Usage:
```

P. Cortez and A. Morais. A Data Mining Approach to Predict Forest Fires using Meteorological Data.
In Proceedings of the 13th EPIA 2007 - Portuguese Conference on Artificial Intelligence,
December, 2007. (http://www.dsi.uminho.pt/~pcortez/fires.pdf)

In the above reference, the output "area" was first transformed with a ln(x+1) function.
Then, several Data Mining methods were applied. After fitting the models, the outputs were
post-processed with the inverse of the ln(x+1) transform. Four different input setups were
used. The experiments were conducted using a 10-fold (cross-validation) x 30 runs. Two
regression metrics were measured: MAD and RMSE. A Gaussian support vector machine (SVM) fed
with only 4 direct weather conditions (temp, RH, wind and rain) obtained the best MAD value:
12.71 +- 0.01 (mean and confidence interval within 95% using a t-student distribution). The
best RMSE was attained by the naive mean predictor. An analysis to the regression error curve
(REC) shows that the SVM model predicts more examples within a lower admitted error. In effect,
the SVM model predicts better small fires, which are the majority.

4. Relevant Information:

This is a very difficult regression task. It can be used to test regression methods. Also,
it could be used to test outlier detection methods, since it is not clear how many outliers
are there. Yet, the number of examples of fires with a large burned area is very small.

5. Number of Instances: 517

6. Number of Attributes: 12 + output attribute

Note: several of the attributes may be correlated, thus it makes sense to apply some sort of
feature selection.

7. Attribute information:

For more information, read [Cortez and Morais, 2007].

```
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
(this output variable is very skewed towards 0.0, thus it may make
 sense to model with the logarithm transform).

8. Missing Attribute Values: None
```

1.1 Part 2

[2]:
```python
# Gives the multidimensional mean of a 2D numpy array
def mean(arr):
    means = np.empty(len(arr[0]))
    for i in range(len(arr)):
        for j in range(len(arr[0])):
            means[j] += arr[i][j]

    for i in range(len(means)):
        means[i] /= len(arr)
    return means
```

[3]:
```python
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(mean(arr))
```

```
[2.50000000e+00 3.50000000e+00 2.37906084e+30]
```

[4]:
```python
# Gives sample covariance between 2 attributes
def cov(attr1, attr2):
    mean1 = np.mean(attr1)
    mean2 = np.mean(attr2)
    cov = 0

    for i in range(len(attr1)):
        cov += (attr1[i] - mean1)*(attr2[i] - mean2)

    cov /= (len(attr1)-1)
    return cov
```

[5]:
```
cov(arr[:,0], arr[:,1])
```

[5]: 4.5

[6]:
```
# Gives standard deviation of a numerical array
def SD(arr):
    mean = np.mean(arr)
    stdev = 0

    for x in arr:
        stdev += (x - mean)**2

    stdev /= (len(arr)-1)
    return np.sqrt(stdev)
```

[7]:
```
# Gives correlation coefficient
def correlation(attr1, attr2):
    covariance = cov(attr1, attr2)
    stdev1 = SD(attr1)
    stdev2 = SD(attr2)
    corr = covariance / (stdev1 * stdev2)
    return corr
```

[8]:
```
correlation(arr[0], arr[1])
```

[8]: 1.0

[9]:
```
np.corrcoef(arr[:,0], arr[:,1])
```

[9]:
```
array([[1., 1.],
       [1., 1.]])
```

[10]:
```
# Normalizes array to values between 0 and 1 (min/max normalization)
def rangeNormalize(arr):
    result = np.empty(arr.shape)
    c = 0
    for col in arr:
        xmax = col[np.argmax(col)]
        xmin = col[np.argmin(col)]
        for i in range(len(col)):
            result[c][i] = (col[i] - xmin) / (xmax - xmin)

        c += 1
    return result
```

[11]:
```
arr2 = rangeNormalize(arr)
arr2
```

[11]: array([[0. , 0.5, 1.],
       [0. , 0.5, 1.]])

[12]:
```python
from sklearn import preprocessing as pp
standard_scalar = pp.MinMaxScaler()
arr3 = standard_scalar.fit_transform(arr)
arr3
```

[12]: array([[0., 0., 0.],
       [1., 1., 1.]])

[13]:
```python
# Normalizes array using z-score normalization
def zscoreNormalize(arr):
    result = np.empty(arr.shape)
    means = mean(arr)
    sds = []
    for col in arr.transpose():
        sds.append(SD(col))

    for i in range(len(arr)):
        for j in range(len(arr[0])):
            result[i][j] = (arr[i][j] - means[j]) / sds[j]

    return result
```

[14]:
```python
arr4 = zscoreNormalize(arr)
arr4
```

[14]: array([[-0.94280904, -0.94280904, -0.94280904],
       [0.47140452, 0.47140452, 0.47140452]])

[15]:
```python
# Gives covariance matrix for a numerical data array
def covMatrix(arr):
    results = np.empty((len(arr), len(arr)))
    for i in range(len(results)):
        for j in range(len(results)):
            if i == j:
                results[i][j] = SD(arr[i])**2
            else:
                results[i][j] = cov(arr[i], arr[j])

    return results
```

[16]:
```python
arr5 = covMatrix(arr)
arr5
```

[16]: array([[1., 1.],
       [1., 1.]])

[17]:
```python
np.cov(arr)
```

[17]:
```
array([[1., 1.],
       [1., 1.]])
```

[18]:
```python
# Label encodes a categorical array
def labelEncode(arr):
    results = np.empty(arr.shape)

    if len(arr.shape) == 1:
        labels = list(np.unique(arr))
        for i in range(len(arr)):
            results[i] = labels.index(arr[i])
    else:
        c = 0
        for col in arr:
            labels = list(np.unique(col))
            for i in range(len(col)):
                results[c][i] = labels.index(col[i])
            c += 1

    return results
```

[19]:
```python
catArr = np.array([['A', 'B', 'C'], ['warm', 'cold', 'warm']])
arr6 = labelEncode(catArr)
arr6
```

[19]:
```
array([[0., 1., 2.],
       [1., 0., 1.]])
```

No null/missing values

[20]:
```python
data.isna().sum()
```

[20]:
```
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
```

1.2 Part 3

[21]:
```
data
```

[21]:
```
       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   0.00
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
..    .. ..   …   …    …      …      …     …    …   ..   …     …      …
512    4  3   aug  sun  81.6   56.7  665.6   1.9  27.8  32   2.7   0.0   6.44
513    2  4   aug  sun  81.6   56.7  665.6   1.9  21.9  71   5.8   0.0  54.29
514    7  4   aug  sun  81.6   56.7  665.6   1.9  21.2  70   6.7   0.0  11.16
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 x 13 columns]
```

[22]:
```
# label encoding
data['month'] = labelEncode(data['month'])
data['day'] = labelEncode(data['day'])

# cyclical encoding
data['month_sin'] = np.sin(data['month'] * 2 * np.pi / 12)
data['month_cos'] = np.cos(data['month'] * 2 * np.pi / 12)

data['day_sin'] = np.sin(data['day'] * 2 * np.pi / 7)
data['day_cos'] = np.cos(data['day'] * 2 * np.pi / 7)
```

[23]:
```
del data['month']
del data['day']
```

[24]:
```
data
```

[24]:
```
       X  Y  FFMC    DMC     DC   ISI  temp  RH  wind  rain   area  month_sin  \
0      7  5  86.2   26.2   94.3   5.1   8.2  51   6.7   0.0   0.00  -0.500000
1      7  4  90.6   35.4  669.1   6.7  18.0  33   0.9   0.0   0.00  -0.866025
2      7  4  90.6   43.7  686.9   6.7  14.6  33   1.3   0.0   0.00  -0.866025
3      8  6  91.7   33.3   77.5   9.0   8.3  97   4.0   0.2   0.00  -0.500000
4      8  6  89.3   51.3  102.2   9.6  11.4  99   1.8   0.0   0.00  -0.500000
..    .. ..   …      …      …     …    …   ..   …     …      …        …
512    4  3  81.6   56.7  665.6   1.9  27.8  32   2.7   0.0   6.44   0.500000
513    2  4  81.6   56.7  665.6   1.9  21.9  71   5.8   0.0  54.29   0.500000
514    7  4  81.6   56.7  665.6   1.9  21.2  70   6.7   0.0  11.16   0.500000
515    1  4  94.4  146.0  614.7  11.3  25.6  42   4.0   0.0   0.00   0.500000
516    6  3  79.5    3.0  106.7   1.1  11.8  31   4.5   0.0   0.00  -1.000000
```

```
        month_cos   day_sin   day_cos
0   -8.660254e-01  0.000000  1.000000
1    5.000000e-01 -0.974928 -0.222521
2    5.000000e-01  0.974928 -0.222521
3   -8.660254e-01  0.000000  1.000000
4   -8.660254e-01  0.433884 -0.900969
..            …         …         …
512  8.660254e-01  0.433884 -0.900969
513  8.660254e-01  0.433884 -0.900969
514  8.660254e-01  0.433884 -0.900969
515  8.660254e-01  0.974928 -0.222521
516 -1.836970e-16 -0.974928 -0.222521

[517 rows x 15 columns]
```

[25]:
```python
np.mean(data)
```

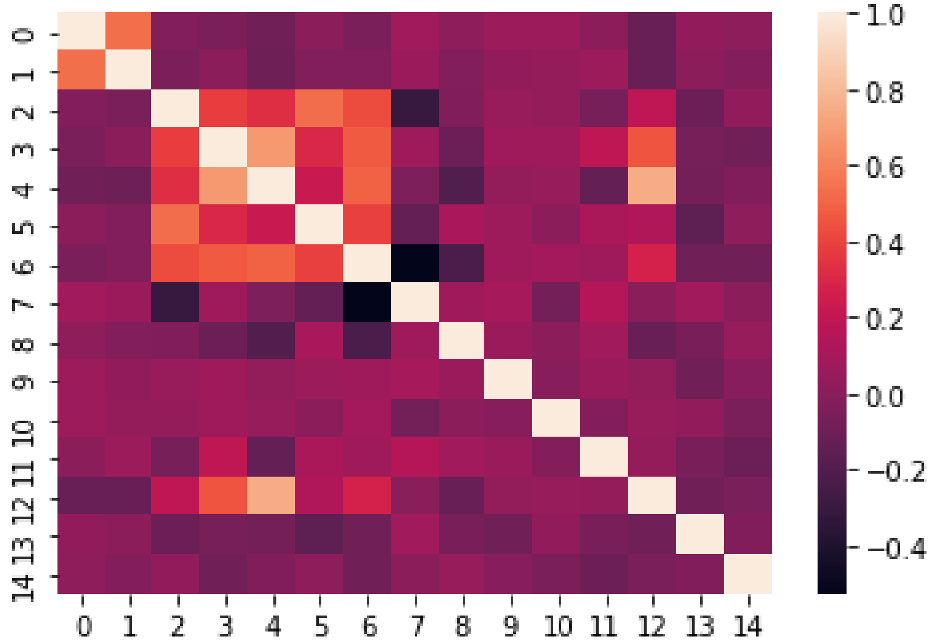
[25]:
```
X              4.669246
Y              4.299807
FFMC          90.644681
DMC          110.872340
DC           547.940039
ISI            9.021663
temp          18.889168
RH            44.288201
wind           4.017602
rain           0.021663
area          12.847292
month_sin      0.017029
month_cos      0.456145
day_sin        0.096494
day_cos       -0.016784
dtype: float64
```

[26]:
```python
data_np = np.array(list(data.values))
mean(data_np)
```

[26]:
```
array([ 4.67117988e+00,  4.30174081e+00,  9.06466151e+01,  1.10874275e+02,
        5.47941973e+02,  9.02359768e+00,  1.88911025e+01,  4.42901354e+01,
        4.01953578e+00,  2.35976789e-02,  1.28492263e+01,  1.89629547e-02,
        4.58079031e-01,  9.84279207e-02, -1.48501876e-02])
```

[27]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
sns.heatmap(np.corrcoef(data_np.T))
```

[27]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2087f67b708>
```



The heatmap shows two lighter areas, one around features 0 and 1, and the other around features 2-6.
I will use these features in the plots below

The covariance matrix is huge, but here is a portion of it

```
[44]: covM = covMatrix(data_np)
      covM
```

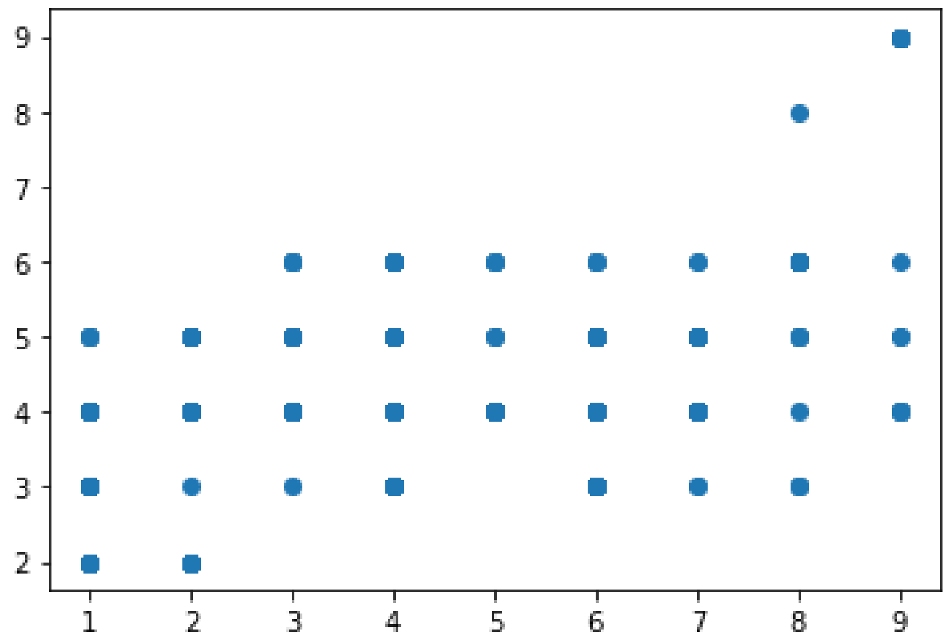
```
[44]: array([[ 1017.3976245 ,  4080.28043129,  4179.39579154, …,
               4091.06208469,  3868.60661753,  1003.56024123],
             [ 4080.28043129, 29215.36089484, 29979.62013265, …,
              28871.44795366, 26612.9002158 ,  4710.07414997],
             [ 4179.39579154, 29979.62013265, 30769.55215041, …,
              29635.88063139, 27362.29894944,  4815.48765853],
             …,
             [ 4091.06208469, 28871.44795366, 29635.88063139, …,
              28659.10619343, 26449.14593104,  4648.73910894],
             [ 3868.60661753, 26612.9002158 , 27362.29894944, …,
              26449.14593104, 25079.68404177,  4270.222045  ],
             [ 1003.56024123,  4710.07414997,  4815.48765853, …,
               4648.73910894,  4270.222045  ,  1064.33329995]])
```

X and Y
These features have a correlation greater than 0.5. This makes sense since they are related as x and y coordinates of fires. The scatter plot shows this nicely.

[29]:
```
print(correlation(data['X'], data['Y']))
plt.scatter(data['X'], data['Y'])
```

```
0.5395481711380373
```

[29]:
```
<matplotlib.collections.PathCollection at 0x2087f82b988>
```

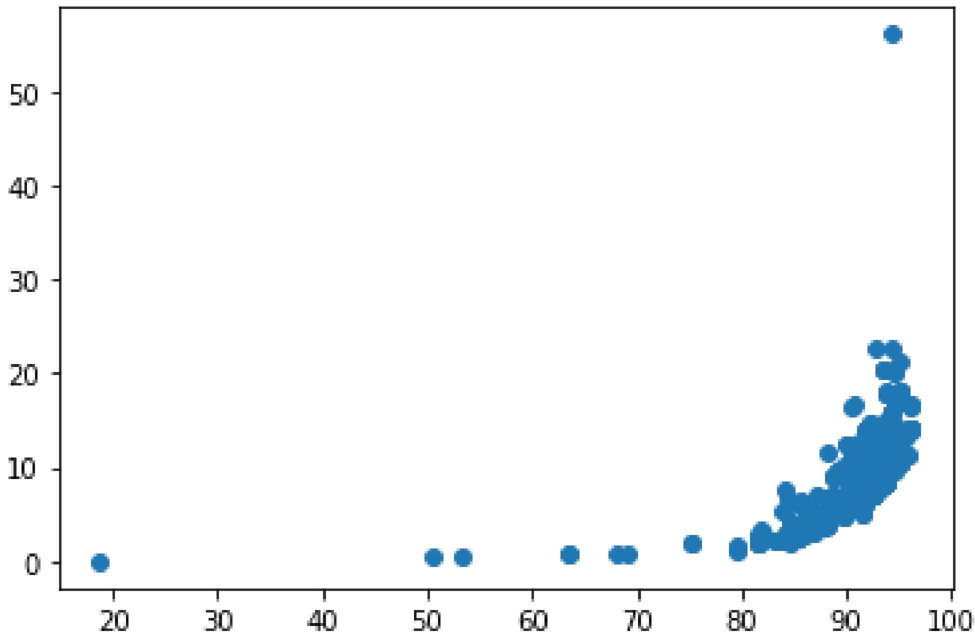


FFMC and ISI
These features also have a correlation greater than 0.5. These are both metrics used in the Fire Weather Index (FWI)
system and it makes sense that they would be related.

[30]:
```
print(correlation(data['FFMC'], data['ISI']))
plt.scatter(data['FFMC'], data['ISI'])
```

```
0.5318049310435652
```

[30]:
```
<matplotlib.collections.PathCollection at 0x2087fac3a48>
```

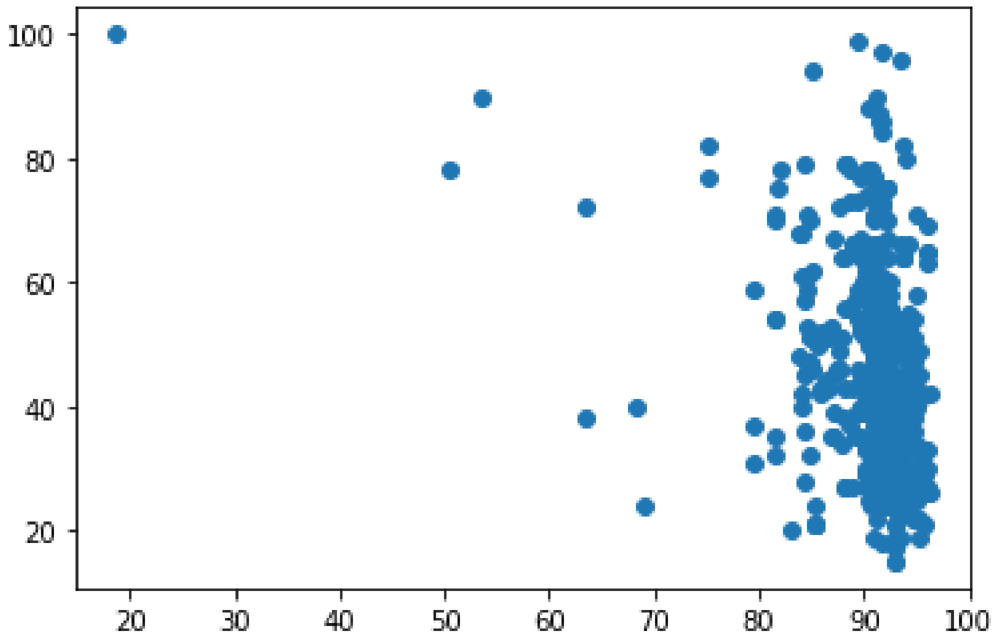


FFMC and RH
These attributes have a correlation less than 0.5 and are not strongly related, shown by the scatter plot.

[31]:
```
print(correlation(data['FFMC'], data['RH']))
plt.scatter(data['FFMC'], data['RH'])
```

```
-0.3009954160617394
```

[31]:
```
<matplotlib.collections.PathCollection at 0x2087fac3ac8>
```

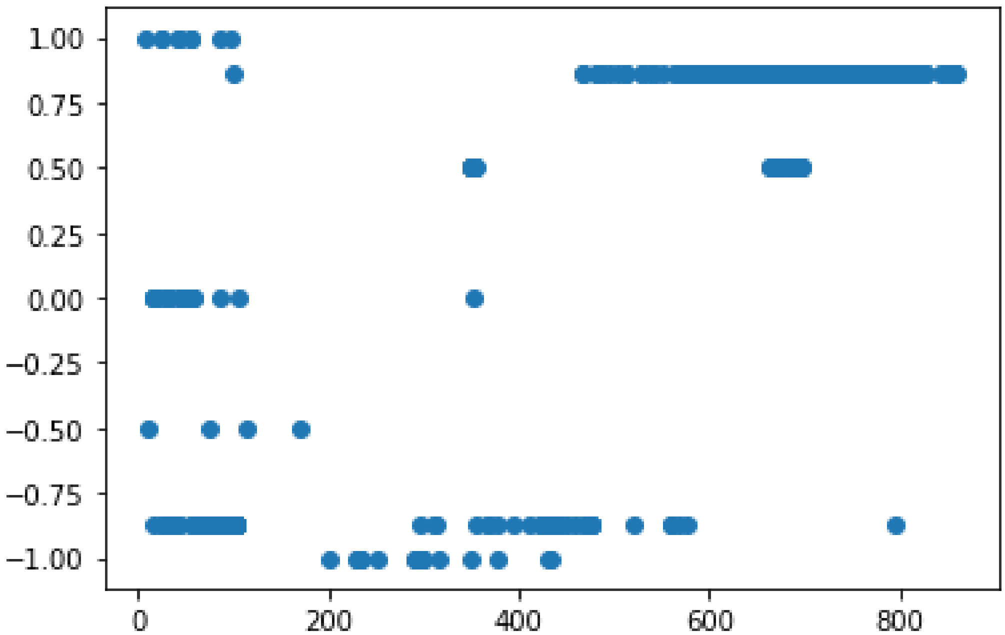


DC and month_cos
These attributes have the strongest correlation of the bunch (0.76). DC is a drought indicator metric that appears
to tie to the month feature rather well. This makes sense as there are hotter, drier months and colder, wetter months.

[32]:
```
print(correlation(data['DC'], data['month_cos']))
plt.scatter(data['DC'], data['month_cos'])
```

```
0.75515212116119
```

[32]:
```
<matplotlib.collections.PathCollection at 0x2087fba0d48>
```

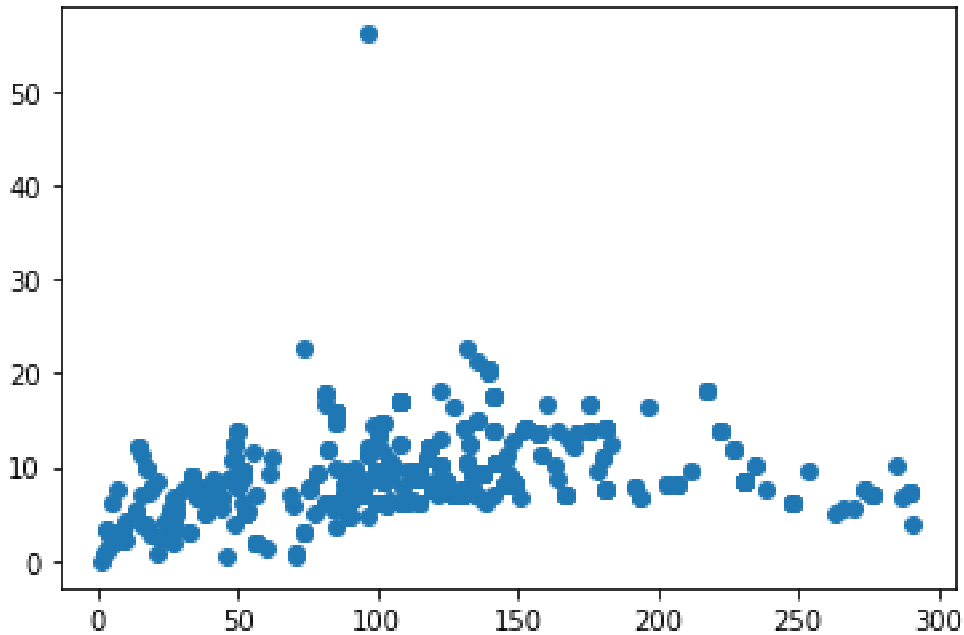


DMC and ISI
These features have a correlation less than 0.5 and are not strongly related. This is seen in the scatter plot.

[33]:
```
print(correlation(data['DMC'], data['ISI']))
plt.scatter(data['DMC'], data['ISI'])
```

```
0.3051278348697831
```

[33]: `<matplotlib.collections.PathCollection at 0x2087fc11048>`



Range-normalized attributes 2 and 3 (FFMC and DMC) have the greatest sample covariance of 0.0098.
There is only one pair of attributes with negative sample covariance (DC and ISI).

[34]:
```
rangeNormData = rangeNormalize(data_np)
```

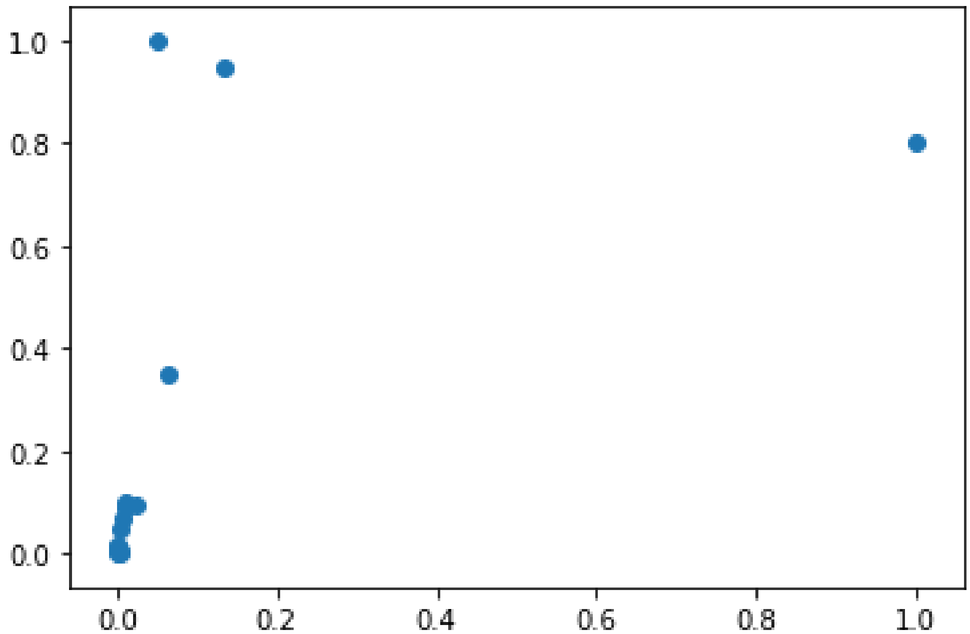
[35]:
```
covArr = []
tempArr = rangeNormData.transpose()
for i in range(len(rangeNormData[0])-1):
    covArr.append(cov(tempArr[i], tempArr[i+1]))

covArr.append(cov(tempArr[-1], tempArr[0]))

covArr
```

[35]:
```
[0.0004033203245356389,
 0.005682839004863395,
 0.009810190246049438,
 0.0027403935837673065,
 -0.0020021985304969534,
 0.0009746148342135041,
 0.005756181293488708,
 0.002976076513023419,
 5.4080218832914596e-05,
 5.265600944415544e-05,
 5.5451748129586725e-05,
```

```
3.735766642694078e-06,
3.9976660355628005e-07,
8.744804415107163e-06,
5.7667640104943634e-05]
```

[36]:
```
plt.scatter(rangeNormData[2], rangeNormData[3])
```

[36]:
```
<matplotlib.collections.PathCollection at 0x2087fc73c88>
```



Z-score normalized attributes 3 and 4 (DMC and DC) have the greatest correlation of 0.68
Attributes 9 and 10 (rain and area) have the least correlation of -0.007. Only 3 pairs of attributes have
a correlation greater than 0.5.

[37]:
```
zNormData = zscoreNormalize(data_np)
zNormData
```

[37]:
```
array([[ 0.99295856,  0.542259  , -0.81120648, …, -1.9208721 ,
        -0.18890964,  1.36324682],
       [ 0.99295856, -0.27081497, -0.01412104, …,  0.01500253,
        -1.60821969, -0.33128434],
       [ 0.99295856, -0.27081497, -0.01412104, …,  0.01500253,
         1.23040041, -0.33128434],
       …,
       [ 0.99295856, -0.27081497, -1.64452308, …,  0.53371857,
         0.44274275, -1.27167819],
```

```
[-1.60020312, -0.27081497,  0.67427093, …,  0.53371857,
  1.23040041, -0.33128434],
[ 0.56076494, -1.08388893, -2.02495023, …, -0.69357676,
 -1.60821969, -0.33128434]])
```

[38]:

```python
corrArr = []
tempArr = zNormData.transpose()
for i in range(len(zNormData[0])-1):
    corrArr.append(correlation(tempArr[i], tempArr[i+1]))

corrArr.append(correlation(tempArr[-1], tempArr[0]))

corrArr
```

[38]:

```
[0.5395481711380352,
 -0.046307545554489124,
 0.38261880004942983,
 0.6821916119833157,
 0.22915416908818784,
 0.39428710420800545,
 -0.5273903386376695,
 0.0694100671560725,
 0.06111888020217677,
 -0.0073657292792923,
 -0.01597107737607042,
 0.04098398560938756,
 -0.08046320461609571,
 -0.027544095812031395,
 0.019883474848008512]
```

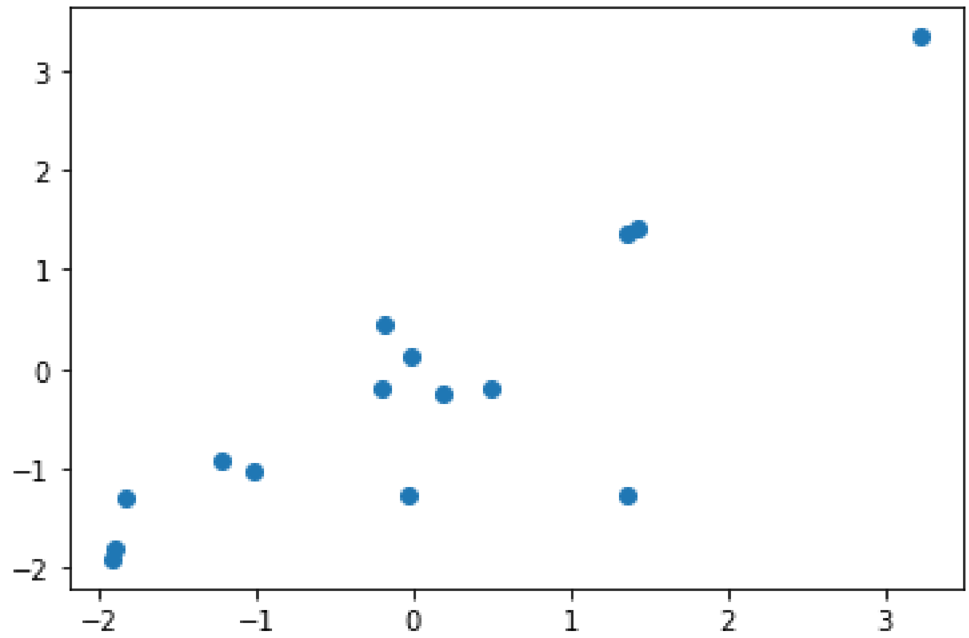
Greatest correlation

[39]:

```python
plt.scatter(zNormData[3], zNormData[4])
```

[39]:

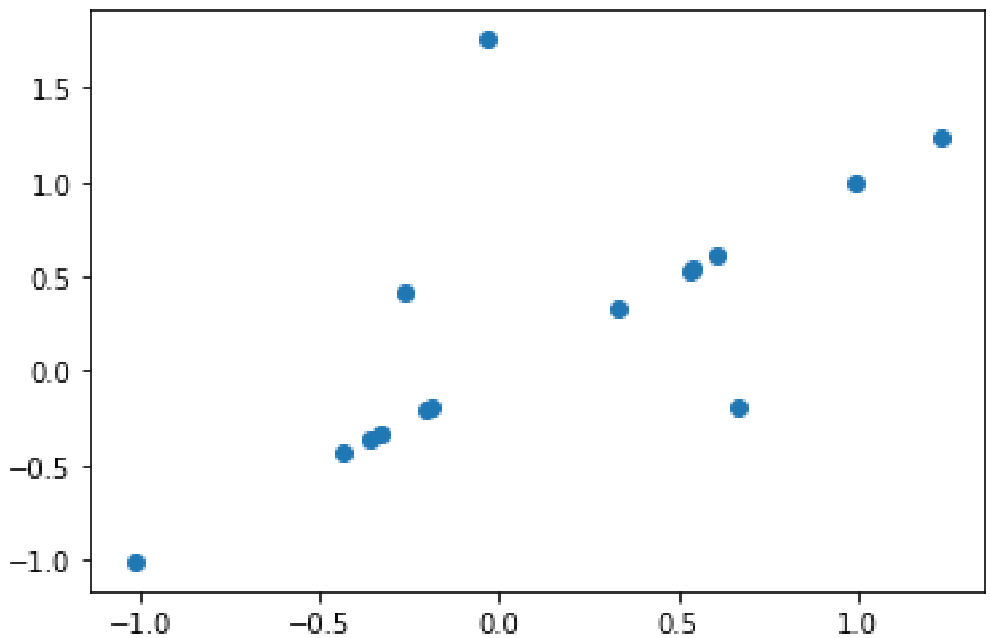
```
<matplotlib.collections.PathCollection at 0x2087fcd3888>
```



Least correlated (magnitude)

```
[40]: plt.scatter(zNormData[9], zNormData[10])
```

```
[40]: <matplotlib.collections.PathCollection at 0x2087fd38908>
```



Varience questions

[41]:
```
varArr = np.var(data_np, axis=0, ddof=1)
varArr
```

[41]:
```
array([5.35356784e+00, 1.51265500e+00, 3.04716238e+01, 4.10195189e+03,
       6.15368355e+04, 2.07888321e+01, 3.37168980e+01, 2.66259802e+02,
       3.21001904e+00, 8.75918012e-02, 4.05206322e+03, 2.95252096e-01,
       4.97924042e-01, 4.71835326e-01, 5.20491320e-01])
```

[42]:
```
totalVariance = sum(varArr)
totalVariance
```

[42]:
```
70054.03707346127
```

[43]:
```
biggestVarience = [varArr[3], varArr[4], varArr[7], varArr[10], varArr[6]]
sum(biggestVarience)
```

[43]:
```
69990.82728109627
```