Supervised Learning Practice

September 21, 2023

[32]:

```
%matplotlib inline
import numpy as np
import pandas as pd
from pylab import *
import matplotlib.pyplot as plt
```

[8]:

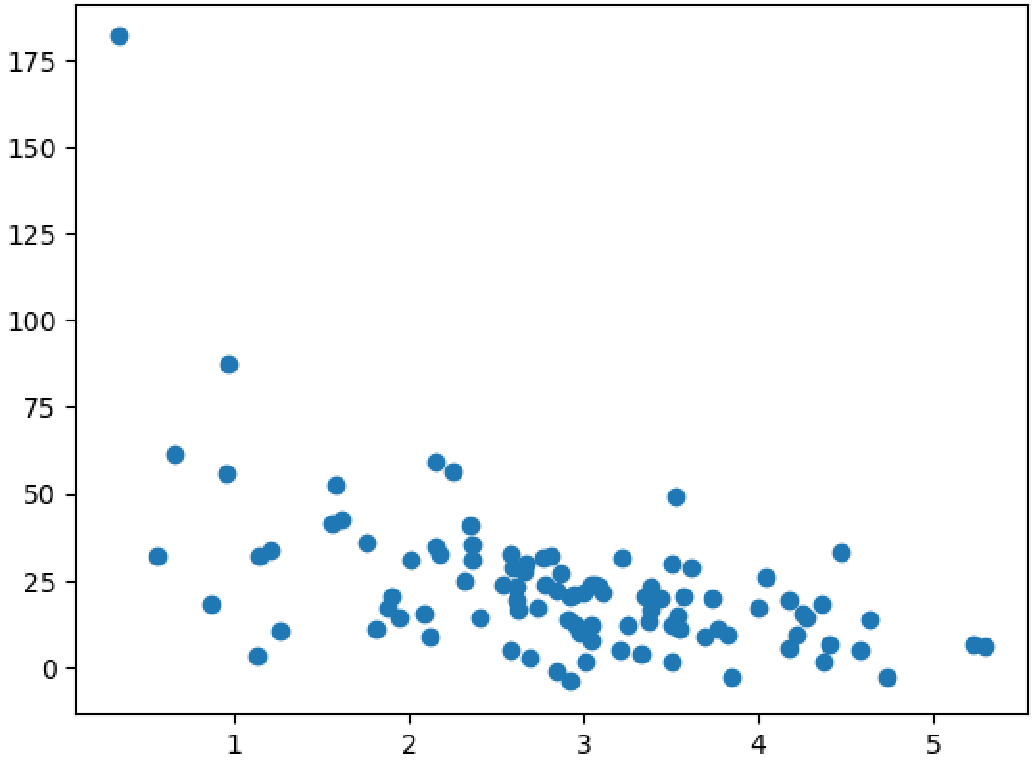
```
np.random.seed(2)
```

[9]:

```
pageSpeeds= np.random.normal(3.0, 1.0, 100)
purchaseAmount= np.random.normal(50.0, 30.0, 100)/pageSpeeds
scatter(pageSpeeds, purchaseAmount)
```

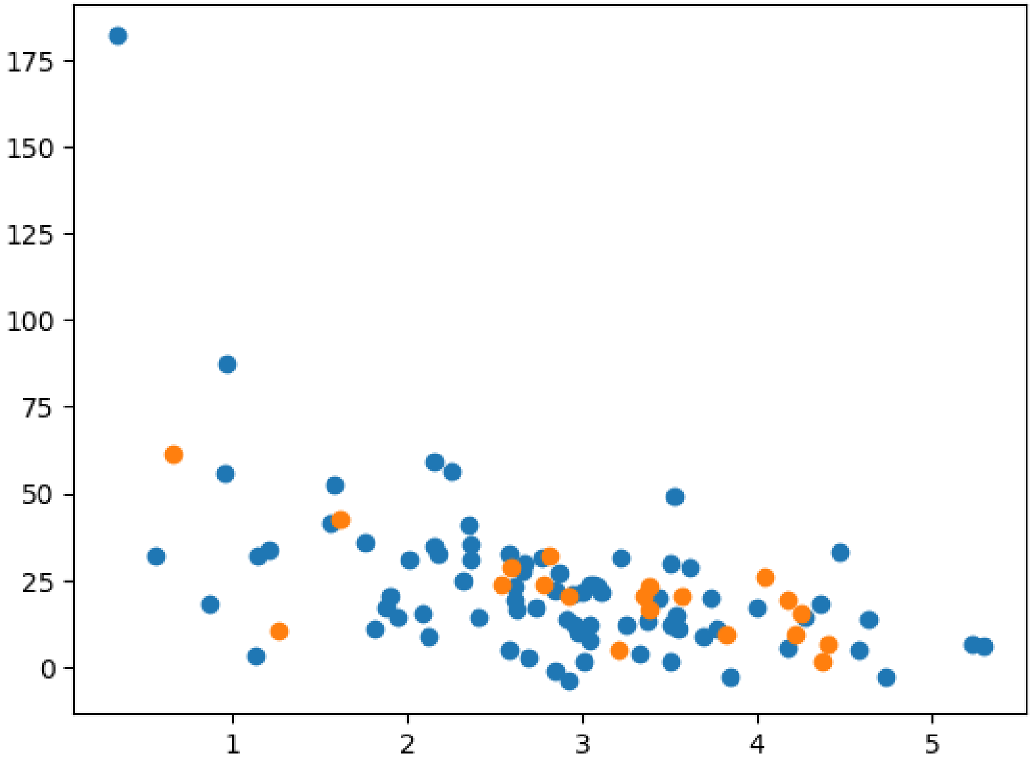
[9]:

```
<matplotlib.collections.PathCollection at 0x285eb647e80>
```



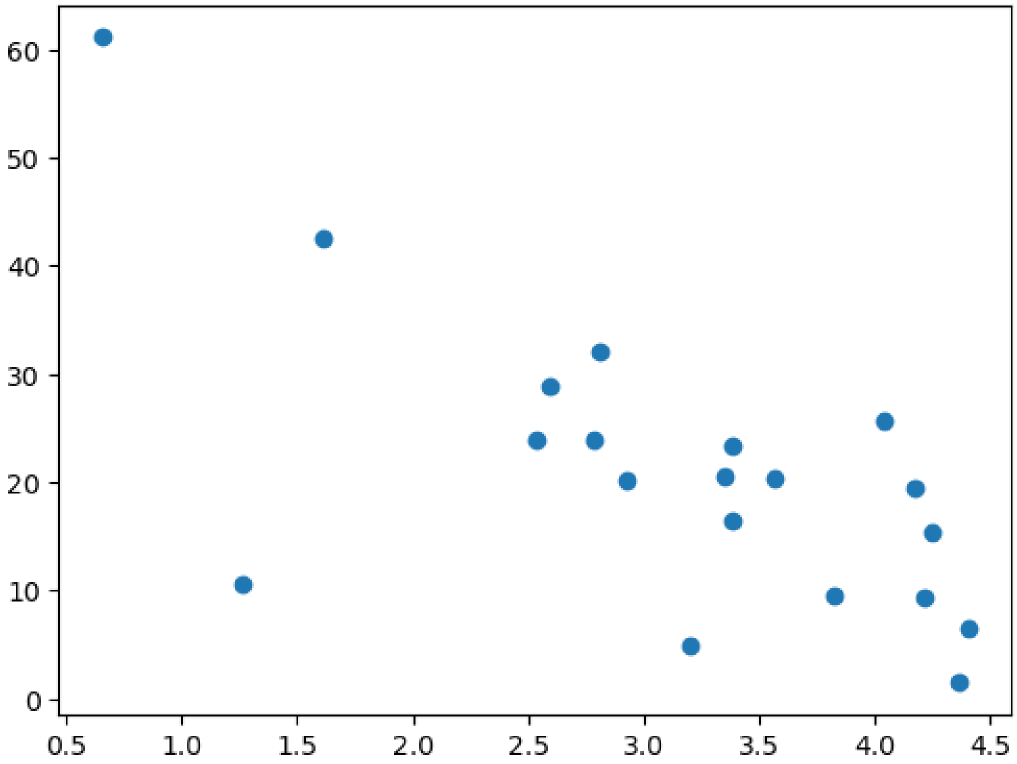
[12]:
```
trainX=pageSpeeds[:80]
testX= pageSpeeds[80:]

trainY= purchaseAmount[:80]
testY = purchaseAmount[80:]

scatter(trainX,trainY)
scatter(testX,testY)
```

[12]:
```
<matplotlib.collections.PathCollection at 0x285eb776850>
```



[13]:
```
scatter(testX,testY)
```

[13]:
```
<matplotlib.collections.PathCollection at 0x285eb8f3be0>
```

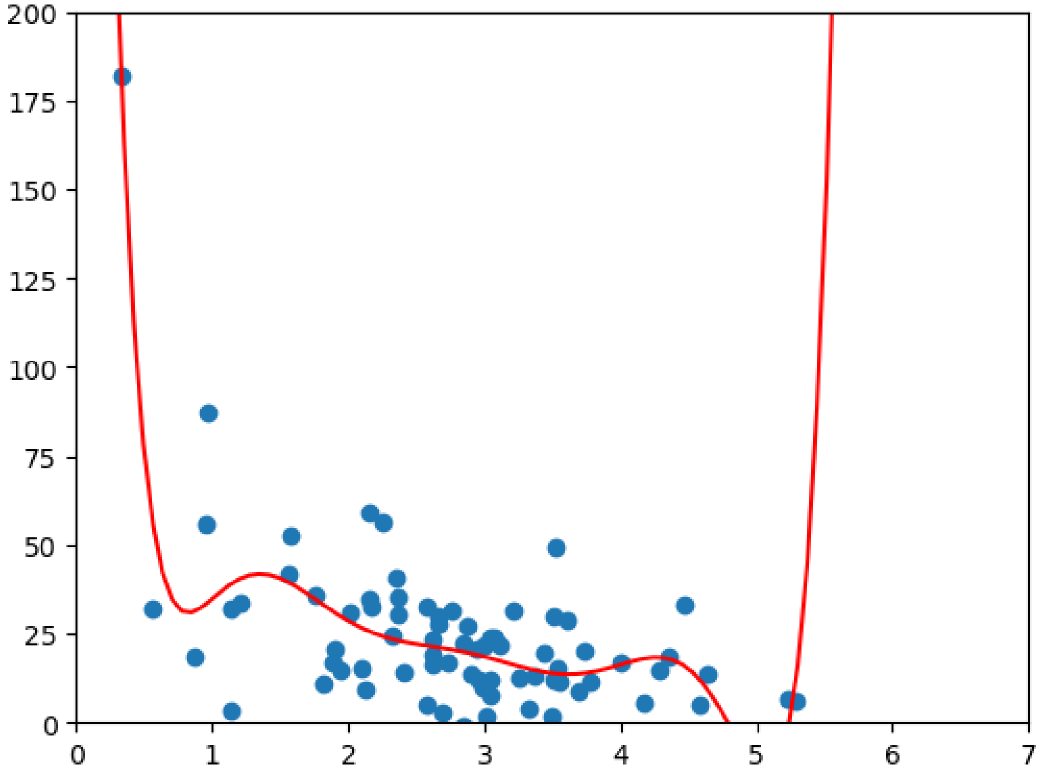


[15]:

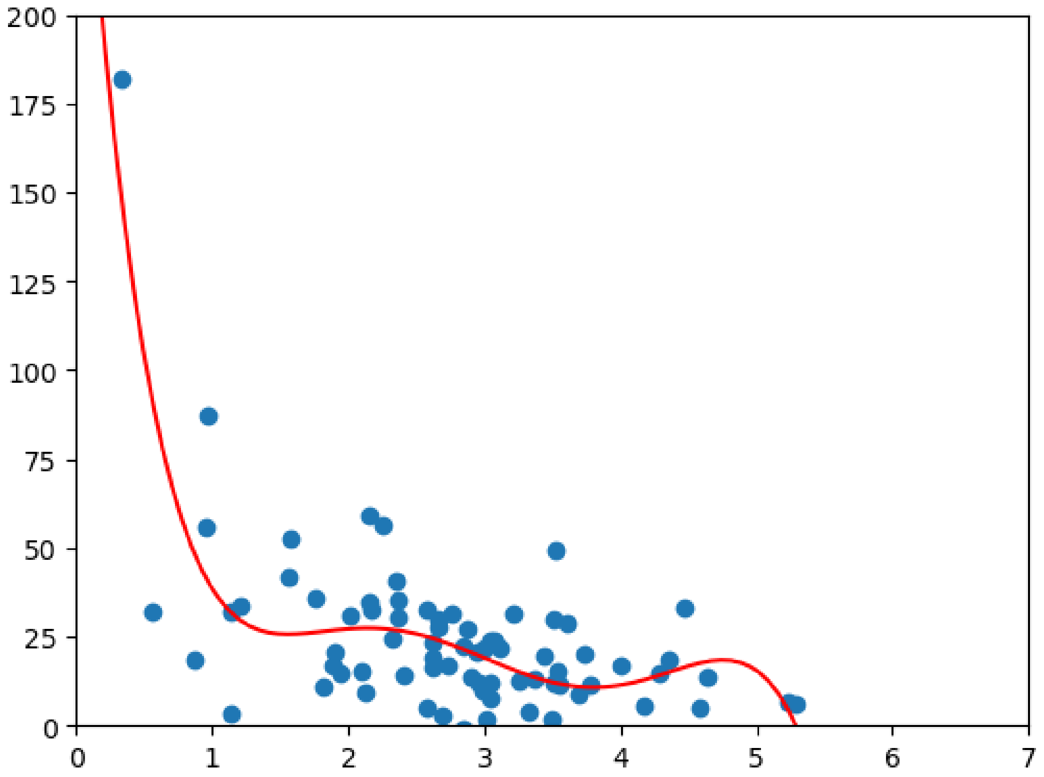
```
x = np.array(trainX)
y = np.array(trainY)

p4 = np.poly1d(np.polyfit(x, y, 8))
```

[17]:

```
xp = linspace(0, 7, 100)
axes = plt.axes()
axes.set_xlim([0,7])
axes.set_ylim([0, 200])
plt.scatter(x,y)
plt.plot(xp, p4(xp), c='r')
plt.show()
```

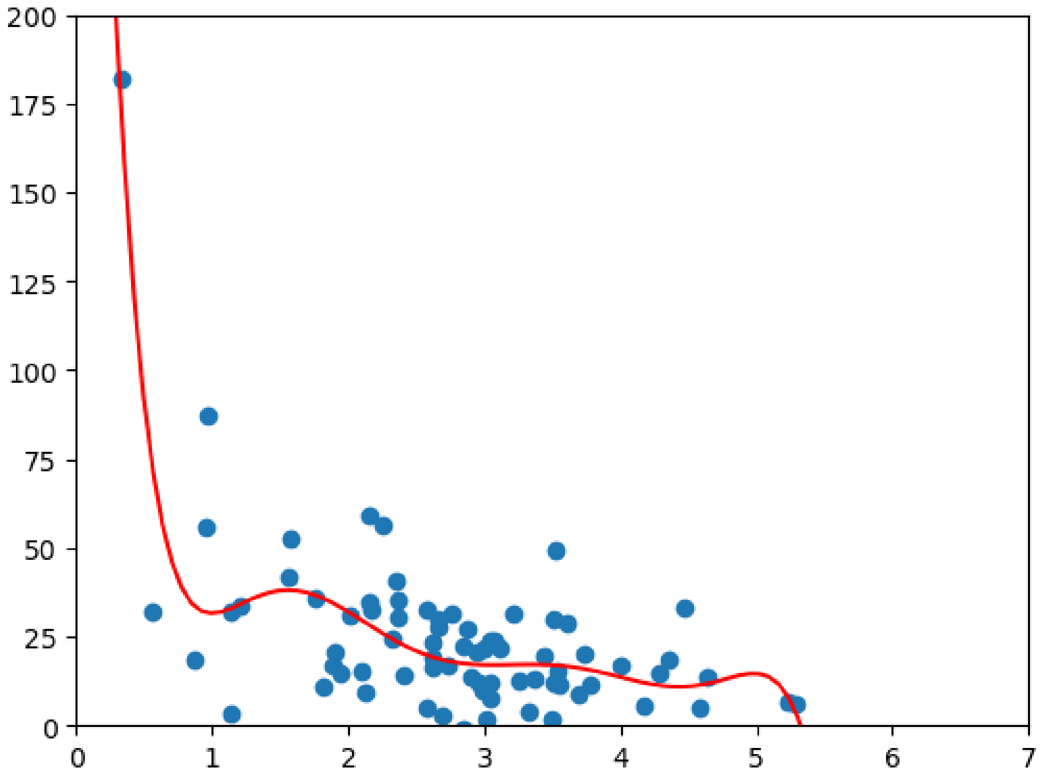


[31]:
```
p4 = np.poly1d(np.polyfit(x, y, 5))
xp = linspace(0, 7, 100)
axes = plt.axes()
axes.set_xlim([0,7])
axes.set_ylim([0, 200])
plt.scatter(x,y)
plt.plot(xp, p4(xp), c='r')
plt.show()
```



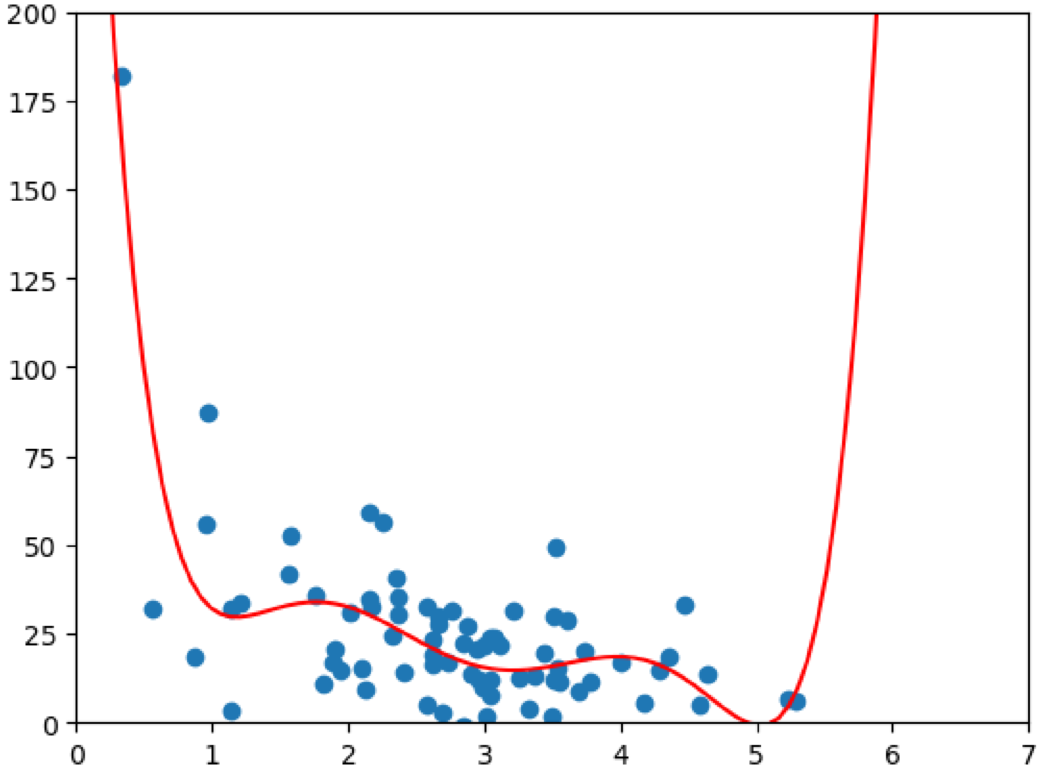
[33]:

```
p4 = np.poly1d(np.polyfit(x, y, 7))
xp = linspace(0, 7, 100)
axes = plt.axes()
axes.set_xlim([0,7])
axes.set_ylim([0, 200])
plt.scatter(x,y)
plt.plot(xp, p4(xp), c='r')
plt.show()
```

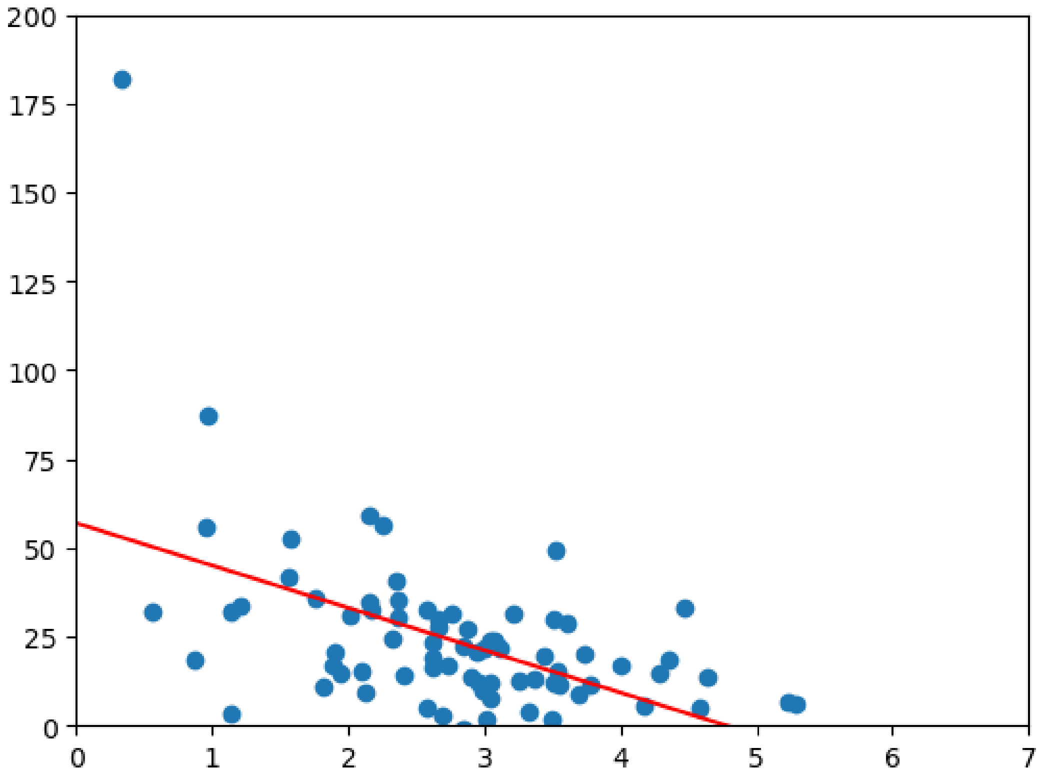


[20]:

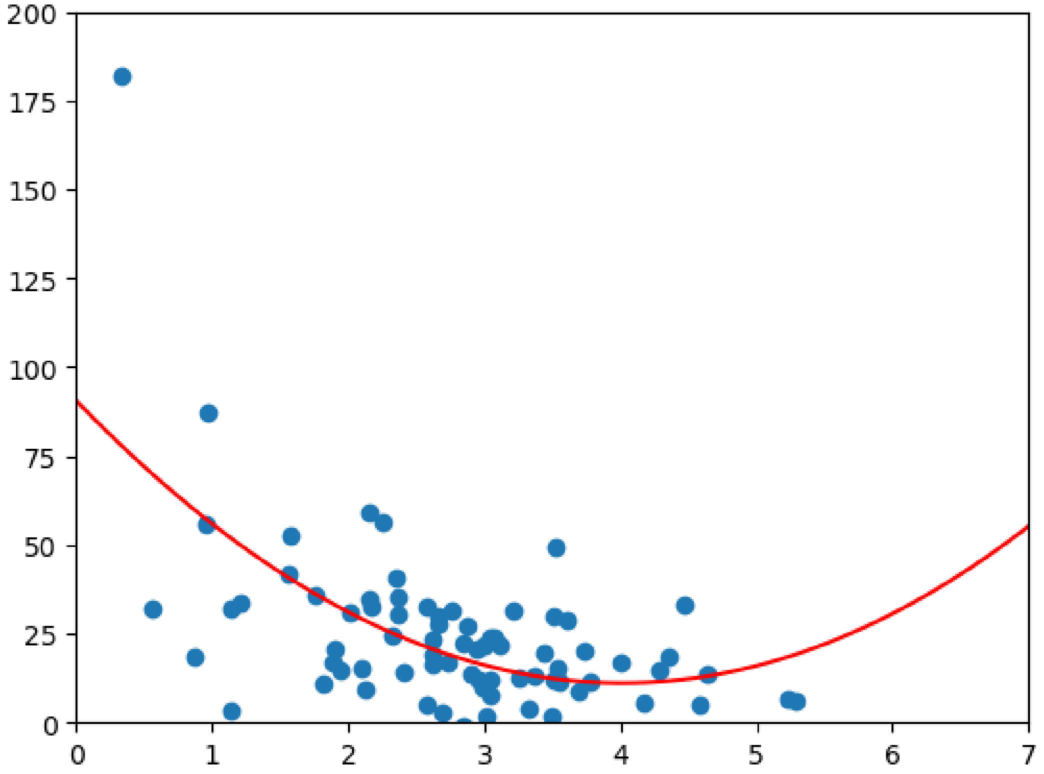
```
p4 = np.poly1d(np.polyfit(x, y, 6))
xp = linspace(0, 7, 100)
axes = plt.axes()
axes.set_xlim([0,7])
axes.set_ylim([0, 200])
plt.scatter(x,y)
plt.plot(xp, p4(xp), c='r')
plt.show()
```



[21]:
```
p4 = np.poly1d(np.polyfit(x, y, 1))
xp = linspace(0, 7, 100)
axes = plt.axes()
axes.set_xlim([0,7])
axes.set_ylim([0, 200])
plt.scatter(x,y)
plt.plot(xp, p4(xp), c='r')
plt.show()
```

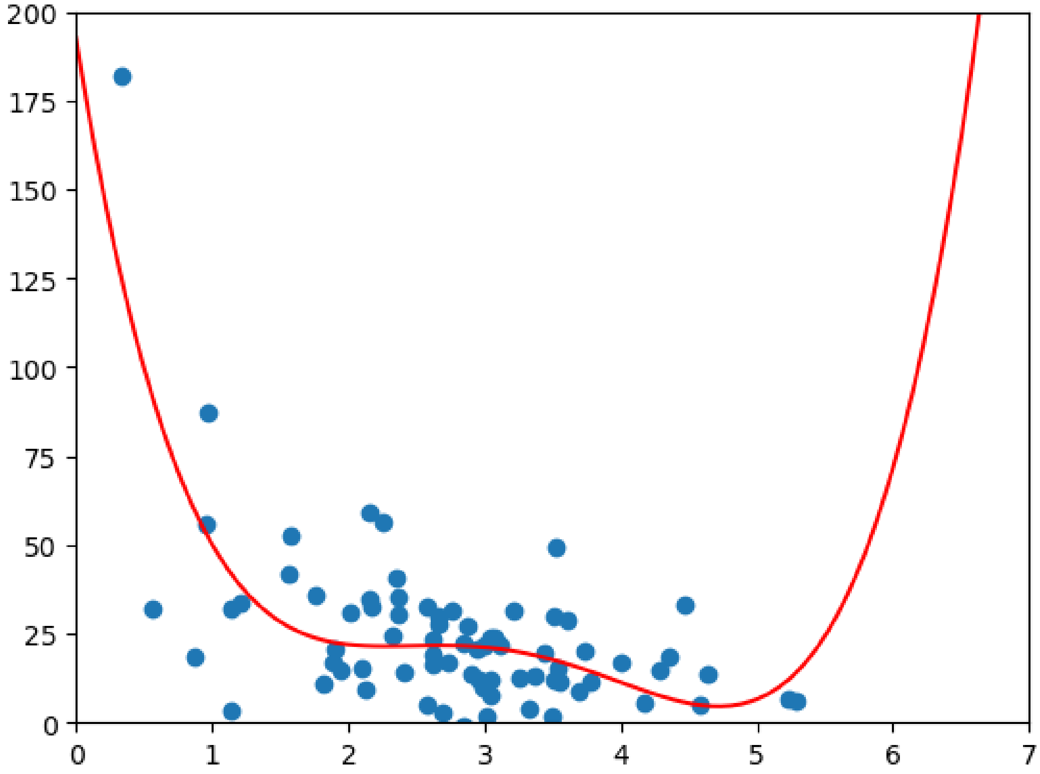


[29]:
```
p4 = np.poly1d(np.polyfit(x, y, 2))
xp = linspace(0, 7, 100)
axes = plt.axes()
axes.set_xlim([0,7])
axes.set_ylim([0, 200])
plt.scatter(x,y)
plt.plot(xp, p4(xp), c='r')
plt.show()
```



[23]:

```
p4 = np.poly1d(np.polyfit(x, y, 4))
xp = linspace(0, 7, 100)
axes = plt.axes()
axes.set_xlim([0,7])
axes.set_ylim([0, 200])
plt.scatter(x,y)
plt.plot(xp, p4(xp), c='r')
plt.show()
```



```
[8]: from sklearn.metrics import r2_score
     r2 = r2_score(testY, p4(testX))
     print(r2)

     ### best fit with 7th grade polynomial
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_5976\1044721873.py in <module>
      1 from sklearn.metrics import r2_score
----> 2 r2 = r2_score(testY, p4(testX))
      3 print(r2)
      4
      5 ### best fit with 7th grade polynomial

NameError: name 'testY' is not defined
```

```
[35]: r2 = r2_score(np.array(trainY), p4(np.array(trainX)))
      print(r2)
```

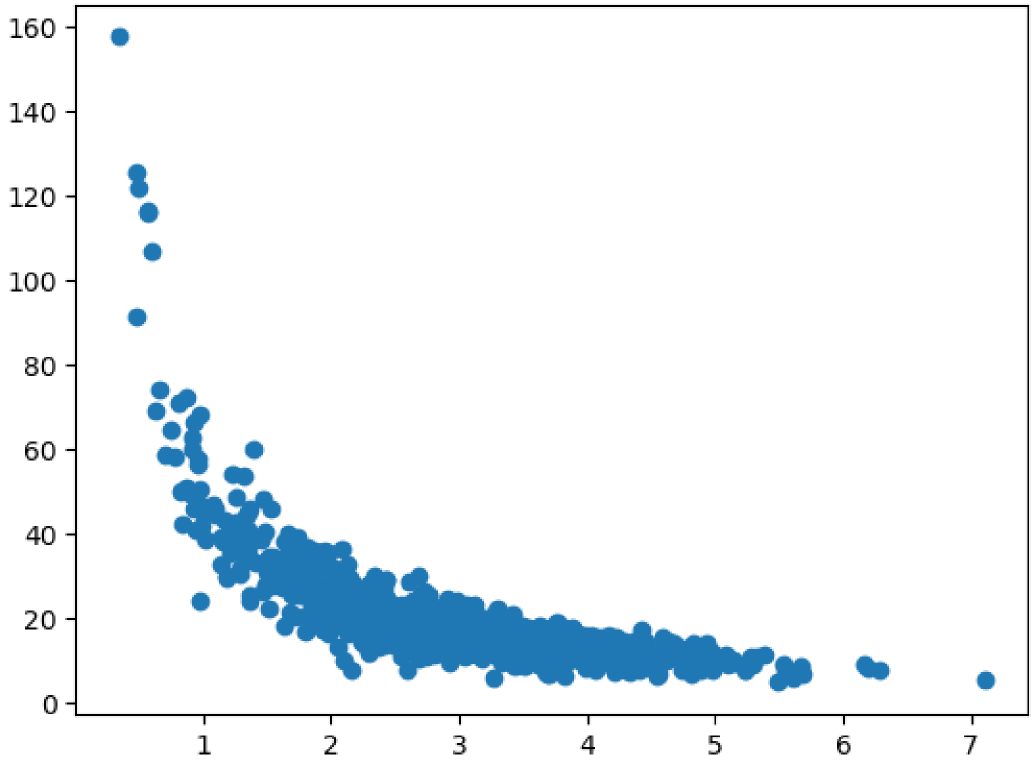
```
0.6170116571732289
```

[3]:

```
%matplotlib inline
from pylab import *
np.random.seed(2)
pageSpeeds = np.random.normal(3,1,1000)
purchaseAmount = np.random.normal(50,10,1000)/pageSpeeds
scatter(pageSpeeds, purchaseAmount)
```

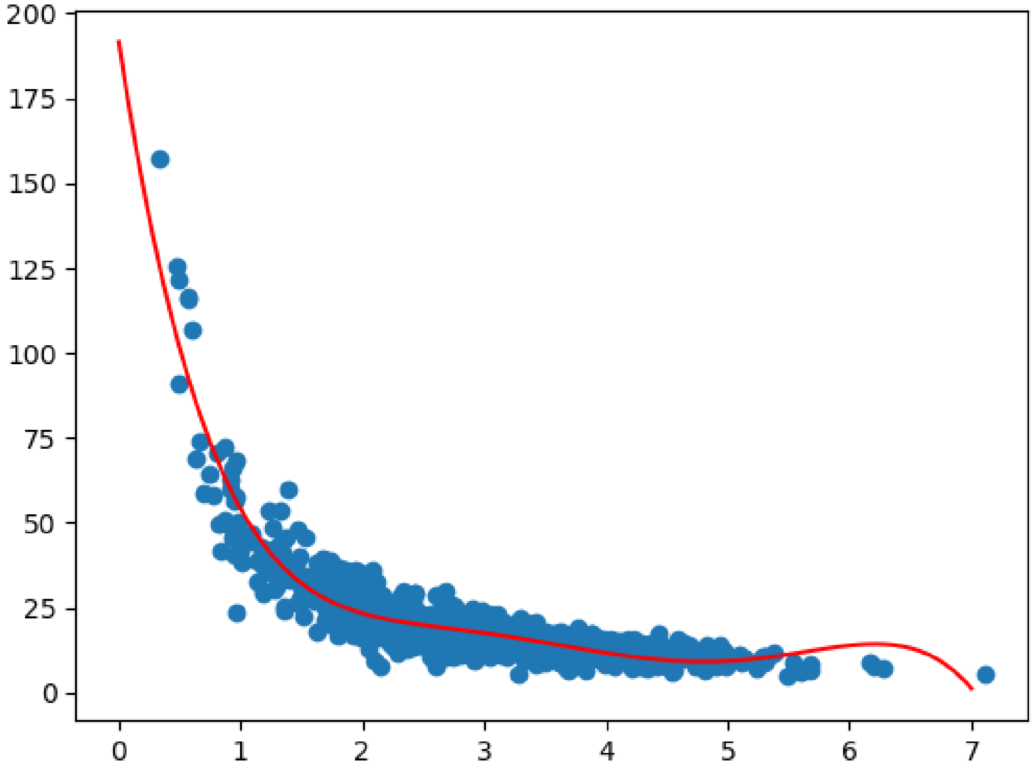
[3]:

```
<matplotlib.collections.PathCollection at 0x2787ae41f70>
```



[29]:

```
x = np.array(pageSpeeds)
y = np.array(purchaseAmount)
p4 = np.poly1d(np.polyfit(x,y,5))
xp = np.linspace(0, 7, 100)
plt.scatter(x,y)
plt.plot(xp,p4(xp), c='r')
plt.show()
```



[30]:

```
r2 = r2_score(y, p4(x))
print(r2)
```

```
0.8553884386186104
```

[34]:

```
df = pd.read_excel('http://cdn.sundog-soft.com/Udemy/DataScience/cars.xls')
df.head()
```

[34]:

```
          Price  Mileage   Make    Model      Trim   Type  Cylinder  Liter  \
0  17314.103129     8221  Buick  Century  Sedan 4D  Sedan         6    3.1
1  17542.036083     9135  Buick  Century  Sedan 4D  Sedan         6    3.1
2  16218.847862    13196  Buick  Century  Sedan 4D  Sedan         6    3.1
3  16336.913140    16342  Buick  Century  Sedan 4D  Sedan         6    3.1
4  16339.170324    19832  Buick  Century  Sedan 4D  Sedan         6    3.1

   Doors  Cruise  Sound  Leather
0      4       1      1        1
1      4       1      1        0
2      4       1      1        0
3      4       1      0        0
4      4       1      0        1
```

[54]:

```
df
```

[54]:

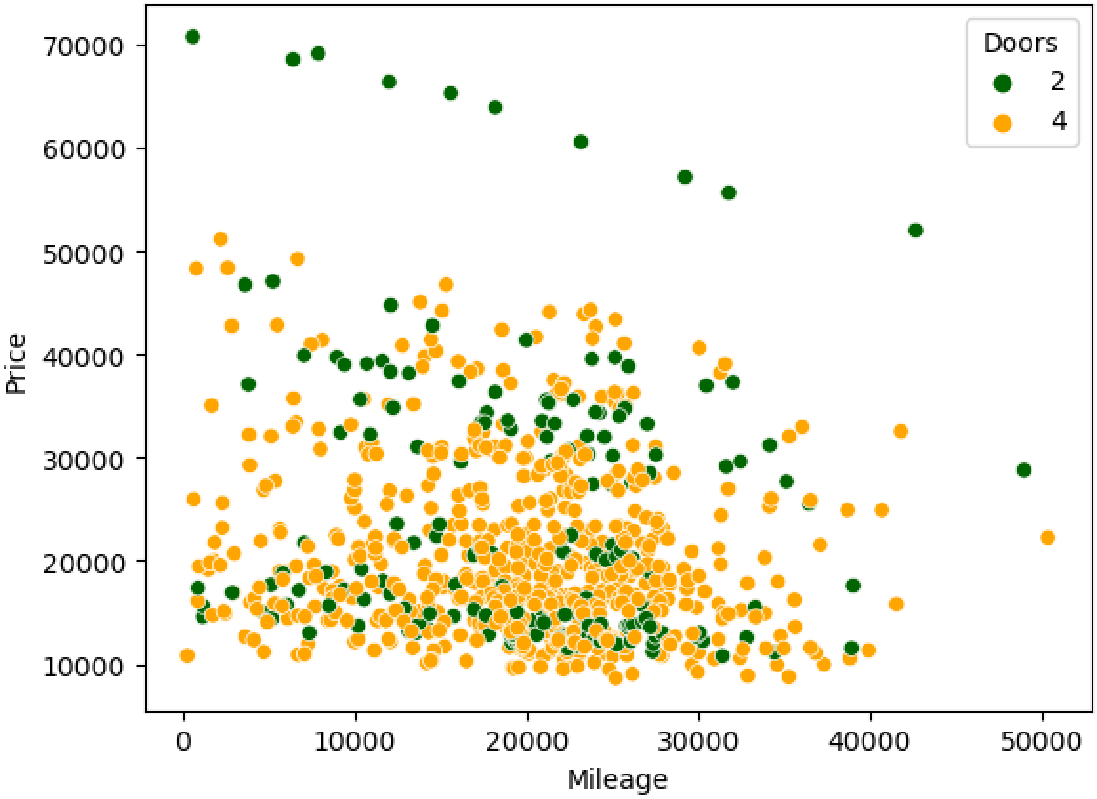
```
             Price  Mileage    Make    Model          Trim   Type  Cylinder  \
0     17314.103129     8221   Buick  Century      Sedan 4D  Sedan         6
1     17542.036083     9135   Buick  Century      Sedan 4D  Sedan         6
2     16218.847862    13196   Buick  Century      Sedan 4D  Sedan         6
3     16336.913140    16342   Buick  Century      Sedan 4D  Sedan         6
4     16339.170324    19832   Buick  Century      Sedan 4D  Sedan         6
..             …        …      …        …          …     …       …
799   16507.070267    16229  Saturn  L Series  L300 Sedan 4D  Sedan         6
800   16175.957604    19095  Saturn  L Series  L300 Sedan 4D  Sedan         6
801   15731.132897    20484  Saturn  L Series  L300 Sedan 4D  Sedan         6
802   15118.893228    25979  Saturn  L Series  L300 Sedan 4D  Sedan         6
803   13585.636802    35662  Saturn  L Series  L300 Sedan 4D  Sedan         6

     Liter  Doors  Cruise  Sound  Leather  Model_ord
0      3.1      4       1      1        1         10
1      3.1      4       1      1        0         10
2      3.1      4       1      1        0         10
3      3.1      4       1      0        0         10
4      3.1      4       1      0        1         10
..     …    …     …     …      …        …
799    3.0      4       1      0        0         21
800    3.0      4       1      1        0         21
801    3.0      4       1      1        0         21
802    3.0      4       1      1        0         21
803    3.0      4       1      0        0         21

[804 rows x 13 columns]
```

[149]:

```python
import seaborn as sns
import matplotlib.pyplot as plt
sns.scatterplot(x = df.Mileage, y = df.Price, hue = df.Doors, palette =␣
 ↪['darkgreen', 'orange'])
```

[149]:

```
<AxesSubplot:xlabel='Mileage', ylabel='Price'>
```



[53]:
```
df.isnull().sum()
```

[53]:
```
Price        0
Mileage      0
Make         0
Model        0
Trim         0
Type         0
Cylinder     0
Liter        0
Doors        0
Cruise       0
Sound        0
Leather      0
Model_ord    0
dtype: int64
```

[74]:
```
cylinders_df = df.groupby('Cylinder').count()
cylinders_df
```

[74]:

```
          Price  Mileage  Make  Model  Trim  Type  Liter  Doors  Cruise  \
Cylinder
6           804      804   804    804   804   804    804    804     804

          Sound  Leather  Model_ord
Cylinder
6           804      804        804
```

[80]:

```
make_df = df.groupby('Make').mean()
make_df
```

[80]:

```
                  Price       Mileage  Cylinder     Liter     Doors    Cruise  \
Make
Buick      20815.113883  20428.100000       6.0  3.662500  4.000000  1.000000
Cadillac   40936.335448  18908.562500       6.0  4.387500  3.750000  1.000000
Chevrolet  16427.599348  19655.587500       6.0  2.868750  3.375000  0.596875
Pontiac    18412.100422  19320.660000       6.0  3.300000  3.600000  0.753333
SAAB       29494.704687  20964.122807       6.0  2.149123  3.473684  1.000000
Saturn     13978.807560  20335.750000       6.0  2.333333  3.333333  0.450000

              Sound   Leather  Model_ord
Make
Buick      0.662500  0.437500  21.750000
Cadillac   0.550000  1.000000  17.875000
Chevrolet  0.828125  0.809375  14.281250
Pontiac    0.606667  0.640000  18.066667
SAAB       0.578947  0.728070   2.368421
Saturn     0.450000  0.483333  20.166667
```

[141]:

```
df['Make'].value_counts()
```

[141]:

```
Chevrolet    320
Pontiac      150
SAAB         114
Buick         80
Cadillac      80
Saturn        60
Name: Make, dtype: int64
```

[142]:

```
df['Model'].value_counts()
```

[142]:

```
Malibu         60
AVEO           60
Cavalier       60
Ion            50
Cobalt         50
9_3 HO         40
Vibe           30
```

```
Bonneville     30
Monte Carlo    30
Lacrosse       30
Impala         30
Grand Prix     30
9_5            30
Deville        30
Lesabre        20
Corvette       20
9_3            20
9_5 HO         20
G6             20
Grand Am       20
Park Avenue    20
Sunfire        10
Century        10
GTO            10
Classic        10
XLR-V8         10
STS-V8         10
STS-V6         10
CTS            10
CST-V          10
L Series       10
9-2X AWD        4
Name: Model, dtype: int64
```

df.describe().round(2)

[39]:
```
import statsmodels.api as sm
```

[42]:
```
df['Model_ord'] = pd.Categorical(df.Model).codes
df.head()
```

[42]:
```
          Price  Mileage   Make    Model      Trim   Type  Cylinder  Liter  \
0  17314.103129     8221  Buick  Century  Sedan 4D  Sedan         6    3.1
1  17542.036083     9135  Buick  Century  Sedan 4D  Sedan         6    3.1
2  16218.847862    13196  Buick  Century  Sedan 4D  Sedan         6    3.1
3  16336.913140    16342  Buick  Century  Sedan 4D  Sedan         6    3.1
4  16339.170324    19832  Buick  Century  Sedan 4D  Sedan         6    3.1

   Doors  Cruise  Sound  Leather  Model_ord
0      4       1      1        1         10
1      4       1      1        0         10
2      4       1      1        0         10
3      4       1      0        0         10
4      4       1      0        1         10
```

[44]:

```
X = df[['Mileage', 'Model_ord', 'Doors']]
y = df['Price']
X1=  sm.add_constant(X)
est = sm.OLS(y,X1).fit()
est.summary()
```

[44]:

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.042
Model:                            OLS   Adj. R-squared:                  0.038
Method:                 Least Squares   F-statistic:                     11.57
Date:                Fri, 21 Jul 2023   Prob (F-statistic):           1.98e-07
Time:                        16:32:46   Log-Likelihood:                -8519.1
No. Observations:                 804   AIC:                         1.705e+04
Df Residuals:                     800   BIC:                         1.706e+04
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       3.125e+04   1809.549     17.272      0.000    2.77e+04    3.48e+04
Mileage       -0.1765      0.042     -4.227      0.000      -0.259      -0.095
Model_ord    -39.0387     39.326     -0.993      0.321    -116.234      38.157
Doors      -1652.9303    402.649     -4.105      0.000   -2443.303    -862.558
==============================================================================
Omnibus:                      206.410   Durbin-Watson:                   0.080
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              470.872
Skew:                           1.379   Prob(JB):                    5.64e-103
Kurtosis:                       5.541   Cond. No.                     1.15e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.15e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

[48]:

```
y.groupby(df.Doors).mean().round(2)
```

[48]:

```
Doors
2    23807.14
4    20580.67
Name: Price, dtype: float64
```

[86]:

```
import os
import io
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from pandas import DataFrame
```

[122]:

```
import warnings
warnings.filterwarnings("ignore")
```

[123]:

```
def readFiles(path):
    for root, dirnames, filenames in os.walk(path):
        for filename in filenames:
            path = os.path.join(root, filename)

            inBody = False
            lines = []
            f = io.open(path, 'r', encoding = 'latin1')
            for line in f:
                if inBody:
                    lines.append(line)
                elif line == '\n':
                    inBody = True
            f.close()
            message = '\n'.join(lines)
            yield path, message

def dataFrameFromDirectory(path, classification):
    rows = []
    index = []
    for filename, message in readFiles(path):
        rows.append({"message":message, "class":classification})
        index.append(filename)
    return DataFrame(rows, index= index)
```

[167]:

```
data = DataFrame({'message':[], 'class': []})
data = data.append(dataFrameFromDirectory('../Rstudio and Python/DataScience/
 ↪DataScience-Python3/emails/spam', 'spam'))
data = data.append(dataFrameFromDirectory('../Rstudio and Python/DataScience/
 ↪DataScience-Python3/emails/ham', 'ham'))
```

[168]:

```
data.head()
```

[168]:

```
              message  \
../Rstudio and Python/DataScience/DataScience-P…  <!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.0 Tr…
../Rstudio and Python/DataScience/DataScience-P…  1) Fight The Risk of
Cancer!\n\nhttp://www.adc…
```

```
../Rstudio and Python/DataScience/DataScience-P…  1) Fight The Risk of
Cancer!\n\nhttp://www.adc…
../Rstudio and Python/DataScience/DataScience-P…
################################################…
../Rstudio and Python/DataScience/DataScience-P…  I thought you might like
these:\n\n1) Slim Dow…

                                                   class
../Rstudio and Python/DataScience/DataScience-P…  spam
../Rstudio and Python/DataScience/DataScience-P…  spam
../Rstudio and Python/DataScience/DataScience-P…  spam
../Rstudio and Python/DataScience/DataScience-P…  spam
../Rstudio and Python/DataScience/DataScience-P…  spam
```

[169]:

```
vectorizer = CountVectorizer()
counts = vectorizer.fit_transform(data['message'].values)
classifier = MultinomialNB()
targets = data['class'].values
classifier.fit(counts, targets)
classifier
```

[169]:

```
MultinomialNB()
```

[170]:

```
example = ['Free Money now!!', 'Hello']
example_counts = vectorizer.transform(example)
predictions = classifier.predict(example_counts)
predictions
```

[170]:

```
array(['spam', 'ham'], dtype='<U4')
```

[ ]:

```

```

[ ]:

```

```