CSC 177: Data Warehousing and Data Mining

Project 1: Data Preprocessing

Jagan Chidella

Group: An Lam, Jimmy Le, Tom Amir,
Dianna Melendez, Amrit Singh, Talal Jawaid, Min Li

Reader Manual:

Starting from Page 3 to Page 25 are screenshots of the code given in Tutorial 4 for Data Preprocessing. Each method of Preprocessing given to us was analyzed for our personal understanding. After realizing the capabilities of the various methods of Data Preprocessing, we began to seek out our data sets online.

After searching online, we chose two data sets that were interesting that we were going to thoroughly analyze for Part 3B of our project:

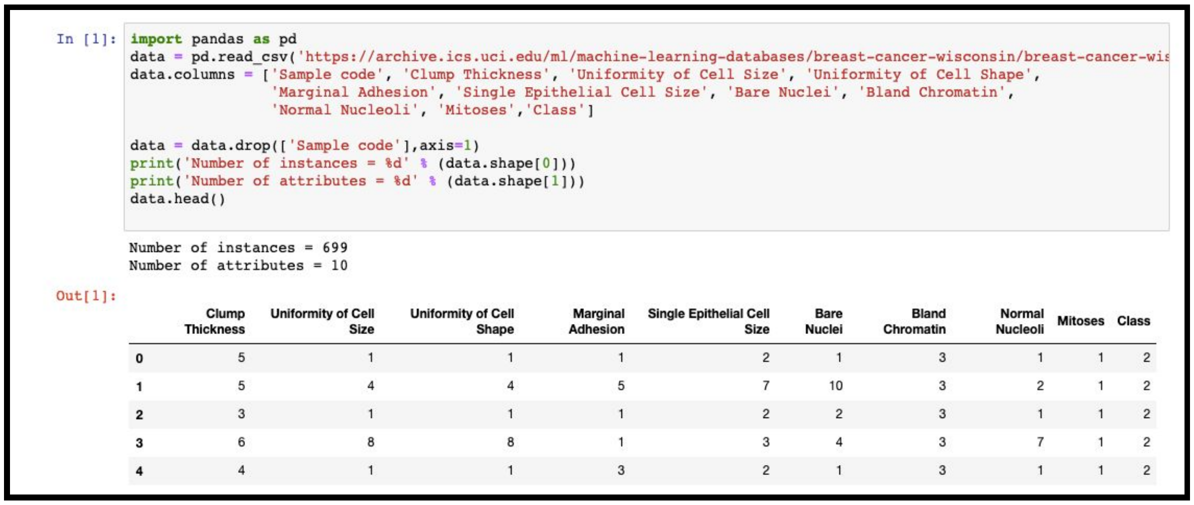
1) **Drugs**: In this zip file, we are using the “Product” Excel Sheet. We made sure to save it as a “.csv” file before proceeding applying the Pre-Processing techniques that we learned.
2) **London Air**: In this data set, we found that we were able to apply the more interesting Pre-Processing techniques, such as: missing values, and saving a dataframe.

Since the purpose of this project is Pre-Processing, we wanted to take data sets and clean them up in such a way where it was an easily readable format. In addition to being easily read, we wanted less attributes to deal with, while dropping unnecessary data. Starting from Page 17, until Page 23, the techniques that we applied were: Missing Values, Outliers, Duplicate Data, Shuffling Dataframes, Sorting Dataframes, Saving a Dataframes, Dropping Fields, Calculated Fields, Feature Normalization, Concatenation, Aggregation, and PCA.

After Pre-Processing our data, we ran a thorough analysis covered in Pages 24 onward. We split the data into training and testing by the ratio of 80:20. We figured out the mean and the standard deviation of each respective set and drew a conclusion from it.

Starting here, we ran through every method of Pre-Processing covered in the tutorial and posted the screenshots of the outputs from each, and then analyzed the results.

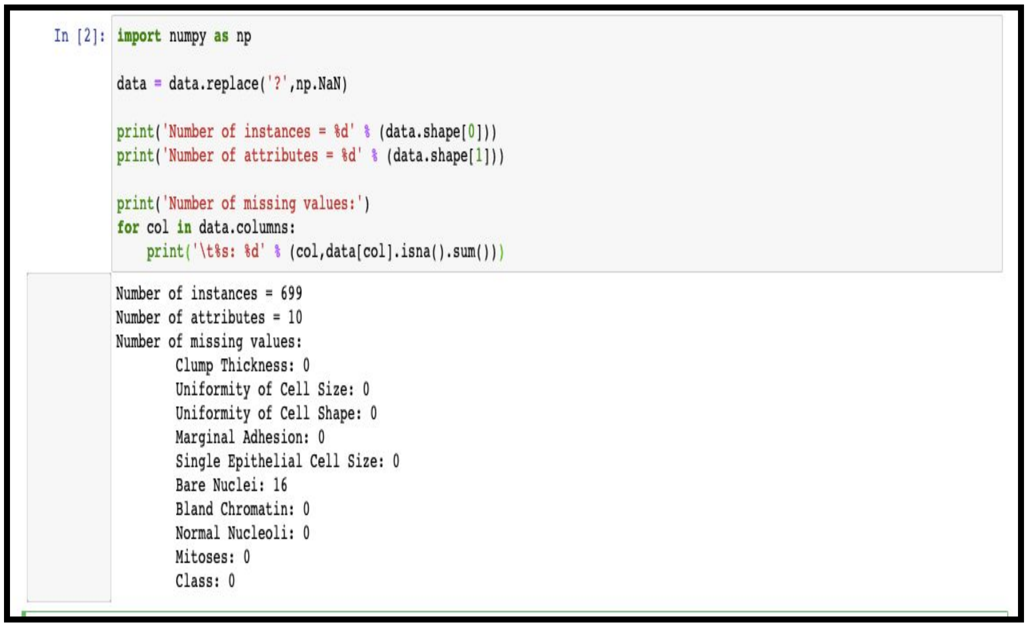
1. Data Quality Issues: The quality of the data is a major concern so we start by displaying our data. We start by printing out the number of rows, and the number of columns in the data set.



```
In [1]: import pandas as pd
        data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wis
        data.columns = ['Sample code', 'Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of Cell Shape',
                        'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromatin',
                        'Normal Nucleoli', 'Mitoses','Class']

        data = data.drop(['Sample code'],axis=1)
        print('Number of instances = %d' % (data.shape[0]))
        print('Number of attributes = %d' % (data.shape[1]))
        data.head()

        Number of instances = 699
        Number of attributes = 10
Out[1]:
```

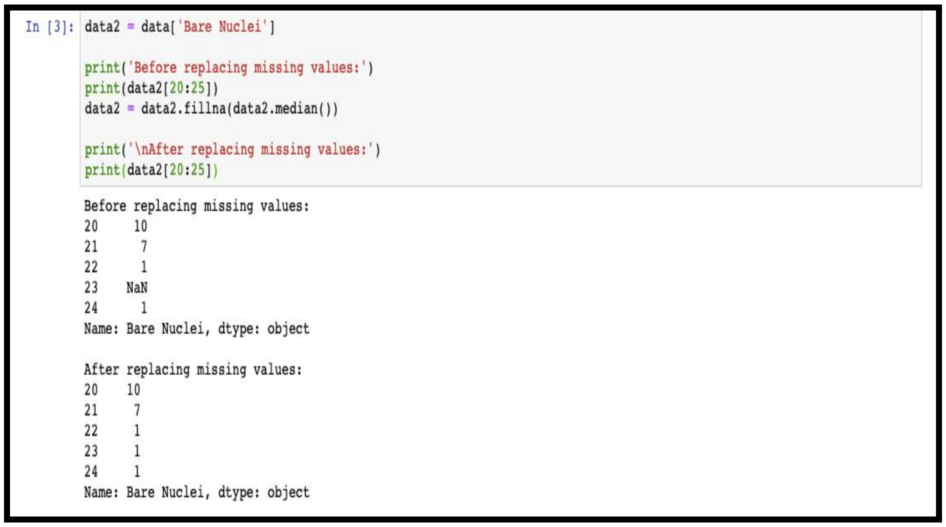
	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	5	1	1	1	2	1	3	1	1	2
1	5	4	4	5	7	10	3	2	1	2
2	3	1	1	1	2	2	3	1	1	2
3	6	8	8	1	3	4	3	7	1	2
4	4	1	1	3	2	1	3	1	1	2

2a. Missing Values: In the data set, it is important to know which columns have data that is missing in it. In this section of code, each column is parsed, counting each missing value.



```
In [2]: import numpy as np

        data = data.replace('?',np.NaN)

        print('Number of instances = %d' % (data.shape[0]))
        print('Number of attributes = %d' % (data.shape[1]))

        print('Number of missing values:')
        for col in data.columns:
            print('\t%s: %d' % (col,data[col].isna().sum()))

        Number of instances = 699
        Number of attributes = 10
        Number of missing values:
                Clump Thickness: 0
                Uniformity of Cell Size: 0
                Uniformity of Cell Shape: 0
                Marginal Adhesion: 0
                Single Epithelial Cell Size: 0
                Bare Nuclei: 16
                Bland Chromatin: 0
                Normal Nucleoli: 0
                Mitoses: 0
                Class: 0
```

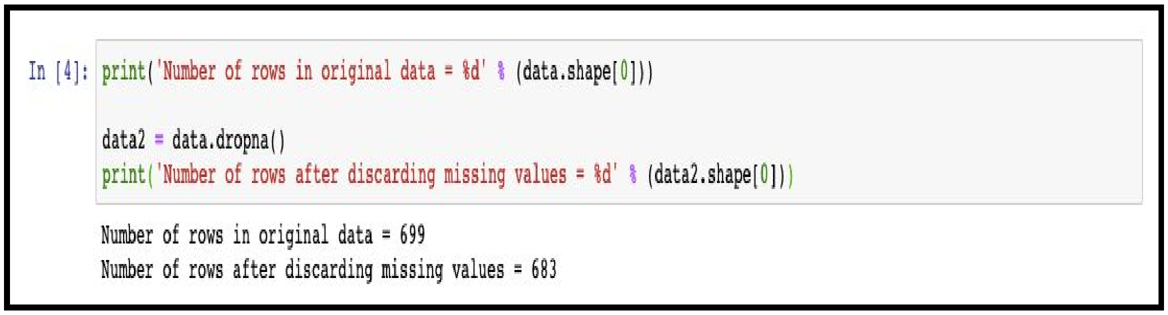
2b. Missing Values: After pinpointing which column had missing values, the rows that had missing values were replaced with the median of the set in order to prevent skewed data.



```
In [3]: data2 = data['Bare Nuclei']

        print('Before replacing missing values:')
        print(data2[20:25])
        data2 = data2.fillna(data2.median())

        print('\nAfter replacing missing values:')
        print(data2[20:25])
```

```
Before replacing missing values:
20     10
21      7
22      1
23    NaN
24      1
Name: Bare Nuclei, dtype: object

After replacing missing values:
20    10
21     7
22     1
23     1
24     1
Name: Bare Nuclei, dtype: object
```

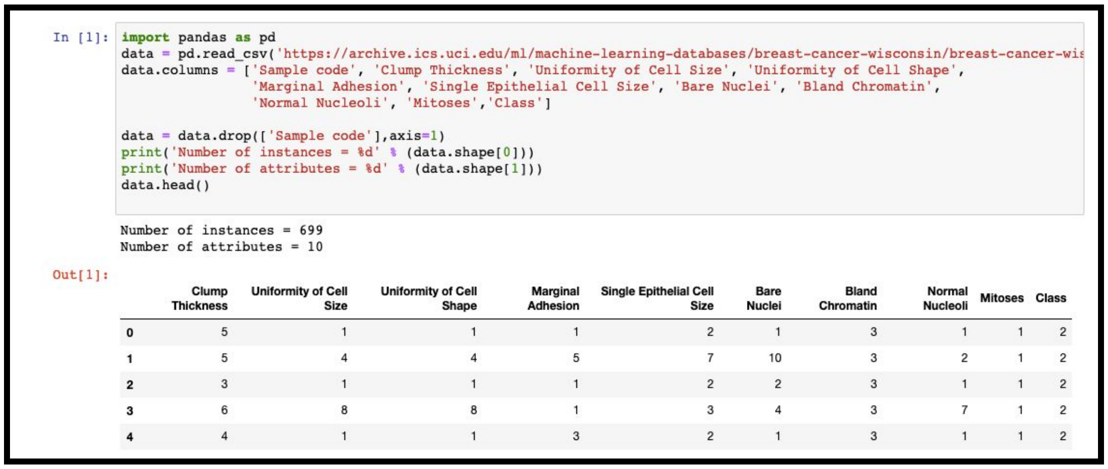
2c. Missing Values: Another approach to missing values is to simply drop the whole row if a column has missing data in it.



```
In [4]: print('Number of rows in original data = %d' % (data.shape[0]))

        data2 = data.dropna()
        print('Number of rows after discarding missing values = %d' % (data2.shape[0]))
```

```
Number of rows in original data = 699
Number of rows after discarding missing values = 683
```

3a. Outliers: Original data provided



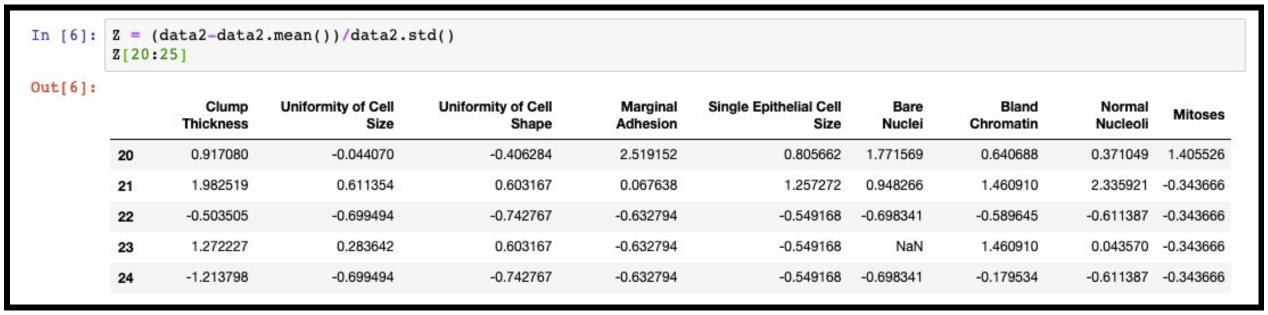
```
In [1]: import pandas as pd
        data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wis
        data.columns = ['Sample code', 'Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of Cell Shape',
                        'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromatin',
                        'Normal Nucleoli', 'Mitoses','Class']

        data = data.drop(['Sample code'],axis=1)
        print('Number of instances = %d' % (data.shape[0]))
        print('Number of attributes = %d' % (data.shape[1]))
        data.head()
```

```
Number of instances = 699
Number of attributes = 10
```

Out[1]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	5	1	1	1	2	1	3	1	1	2
1	5	4	4	5	7	10	3	2	1	2
2	3	1	1	1	2	2	3	1	1	2
3	6	8	8	1	3	4	3	7	1	2
4	4	1	1	3	2	1	3	1	1	2

3b. Outliers: Calculating the Z-Score to help locate outliers

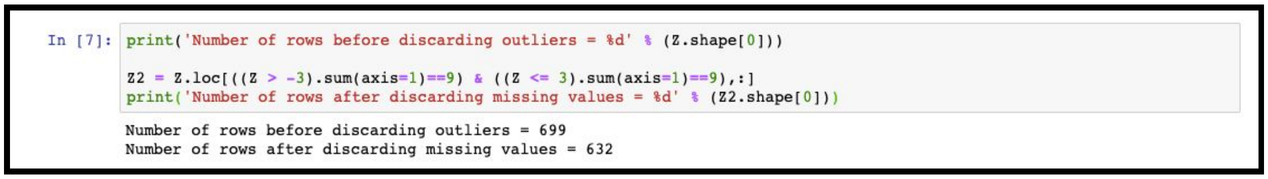


```
In [6]: Z = (data2-data2.mean())/data2.std()
        Z[20:25]
```

Out[6]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses
20	0.917080	-0.044070	-0.406284	2.519152	0.805662	1.771569	0.640688	0.371049	1.405526
21	1.982519	0.611354	0.603167	0.067638	1.257272	0.948266	1.460910	2.335921	-0.343666
22	-0.503505	-0.699494	-0.742767	-0.632794	-0.549168	-0.698341	-0.589645	-0.611387	-0.343666
23	1.272227	0.283642	0.603167	-0.632794	-0.549168	NaN	1.460910	0.043570	-0.343666
24	-1.213798	-0.699494	-0.742767	-0.632794	-0.549168	-0.698341	-0.179534	-0.611387	-0.343666

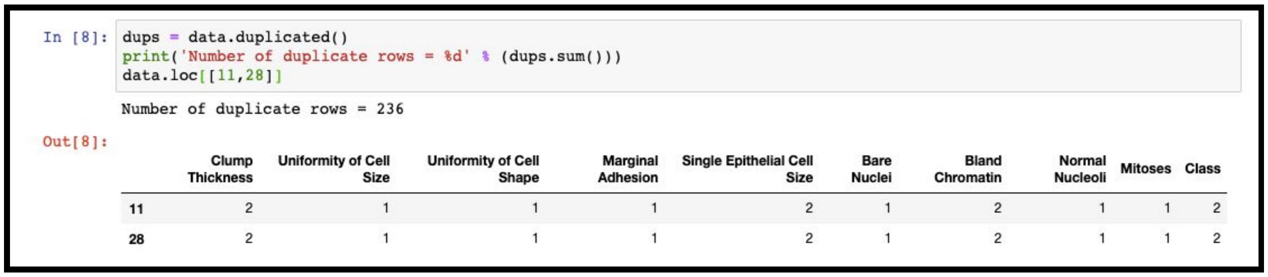
3c. Outliers: Discarding the outliers. Significantly reduces the number of rows



```
In [7]: print('Number of rows before discarding outliers = %d' % (Z.shape[0]))

        Z2 = Z.loc[((Z > -3).sum(axis=1)==9) & ((Z <= 3).sum(axis=1)==9),:]
        print('Number of rows after discarding missing values = %d' % (Z2.shape[0]))

        Number of rows before discarding outliers = 699
        Number of rows after discarding missing values = 632
```

4a. Duplicate Data

Some datasets, especially when obtained by merging multiple data sources can contain duplicate sets. The code below will check for duplicate data.



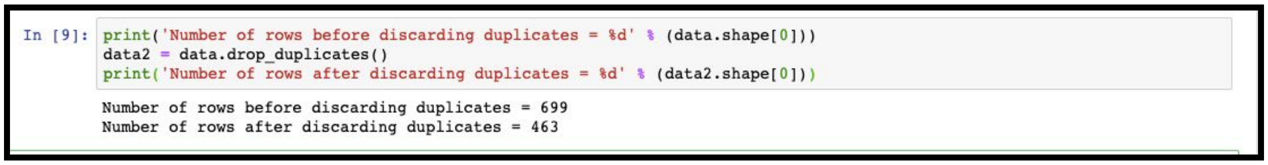
```
In [8]: dups = data.duplicated()
        print('Number of duplicate rows = %d' % (dups.sum()))
        data.loc[[11,28]]

        Number of duplicate rows = 236
```

Out[8]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
11	2	1	1	1	2	1	2	1	1	2
28	2	1	1	1	2	1	2	1	1	2

4b. Duplicate Data

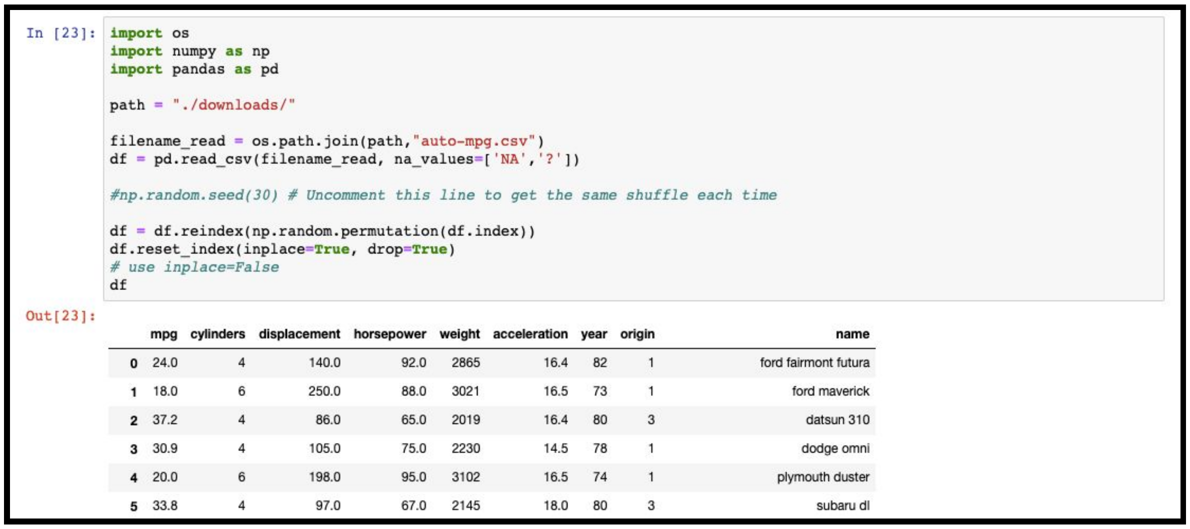
Some datasets, especially when obtained by merging multiple data sources can contain duplicate sets. The code below will discord the duplicate data after duplicate data is found.



```
In [9]: print('Number of rows before discarding duplicates = %d' % (data.shape[0]))
        data2 = data.drop_duplicates()
        print('Number of rows after discarding duplicates = %d' % (data2.shape[0]))

        Number of rows before discarding duplicates = 699
        Number of rows after discarding duplicates = 463
```

5. Shuffling Dataframes

The code below will shuffle the data frame.

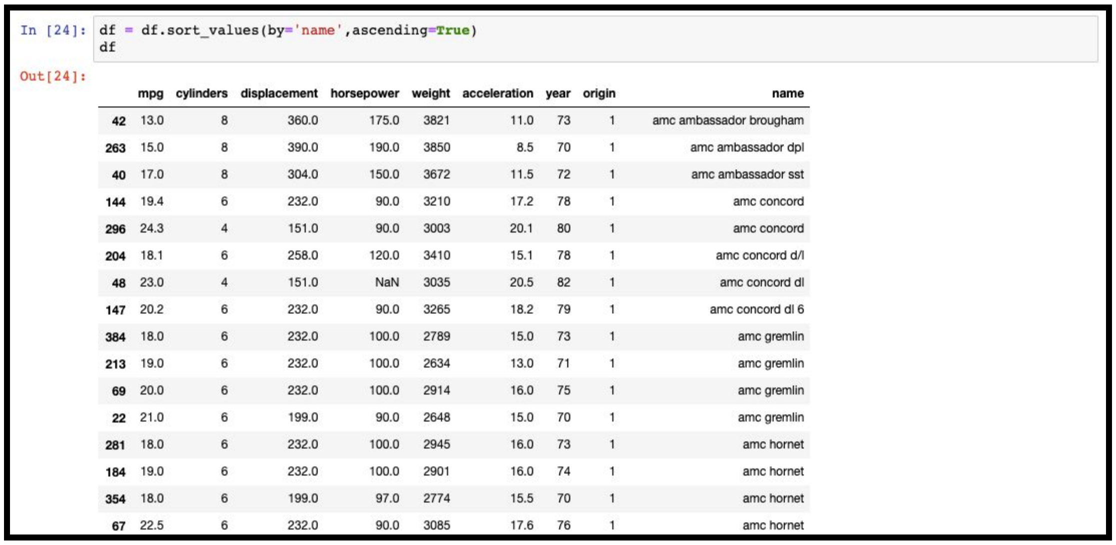


```
In [23]: import os
         import numpy as np
         import pandas as pd

         path = "./downloads/"

         filename_read = os.path.join(path,"auto-mpg.csv")
         df = pd.read_csv(filename_read, na_values=['NA','?'])

         #np.random.seed(30) # Uncomment this line to get the same shuffle each time

         df = df.reindex(np.random.permutation(df.index))
         df.reset_index(inplace=True, drop=True)
         # use inplace=False
         df
```

```
Out[23]:
    mpg  cylinders  displacement  horsepower  weight  acceleration  year  origin              name
0  24.0          4         140.0        92.0    2865          16.4    82       1  ford fairmont futura
1  18.0          6         250.0        88.0    3021          16.5    73       1         ford maverick
2  37.2          4          86.0        65.0    2019          16.4    80       3            datsun 310
3  30.9          4         105.0        75.0    2230          14.5    78       1           dodge omni
4  20.0          6         198.0        95.0    3102          16.5    74       1       plymouth duster
5  33.8          4          97.0        67.0    2145          18.0    80       3             subaru dl
```

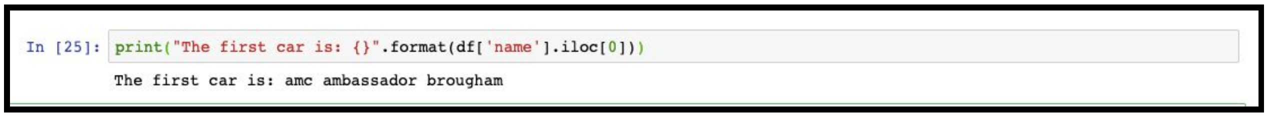
6. Sorting Dataframes

The code below sorts data frame by name and will show in ascending order.



```
In [24]: df = df.sort_values(by='name',ascending=True)
         df
```

```
Out[24]:
      mpg  cylinders  displacement  horsepower  weight  acceleration  year  origin                     name
42   13.0          8         360.0       175.0    3821          11.0    73       1  amc ambassador brougham
263  15.0          8         390.0       190.0    3850           8.5    70       1       amc ambassador dpl
40   17.0          8         304.0       150.0    3672          11.5    72       1       amc ambassador sst
144  19.4          6         232.0        90.0    3210          17.2    78       1              amc concord
296  24.3          4         151.0        90.0    3003          20.1    80       1              amc concord
204  18.1          6         258.0       120.0    3410          15.1    78       1          amc concord d/l
48   23.0          4         151.0         NaN    3035          20.5    82       1           amc concord dl
147  20.2          6         232.0        90.0    3265          18.2    79       1         amc concord dl 6
384  18.0          6         232.0       100.0    2789          15.0    73       1               amc gremlin
213  19.0          6         232.0       100.0    2634          13.0    71       1               amc gremlin
69   20.0          6         232.0       100.0    2914          16.0    75       1               amc gremlin
22   21.0          6         199.0        90.0    2648          15.0    70       1               amc gremlin
281  18.0          6         232.0       100.0    2945          16.0    73       1                amc hornet
184  19.0          6         232.0       100.0    2901          16.0    74       1                amc hornet
354  18.0          6         199.0        97.0    2774          15.5    70       1                amc hornet
67   22.5          6         232.0        90.0    3085          17.6    76       1                amc hornet
```

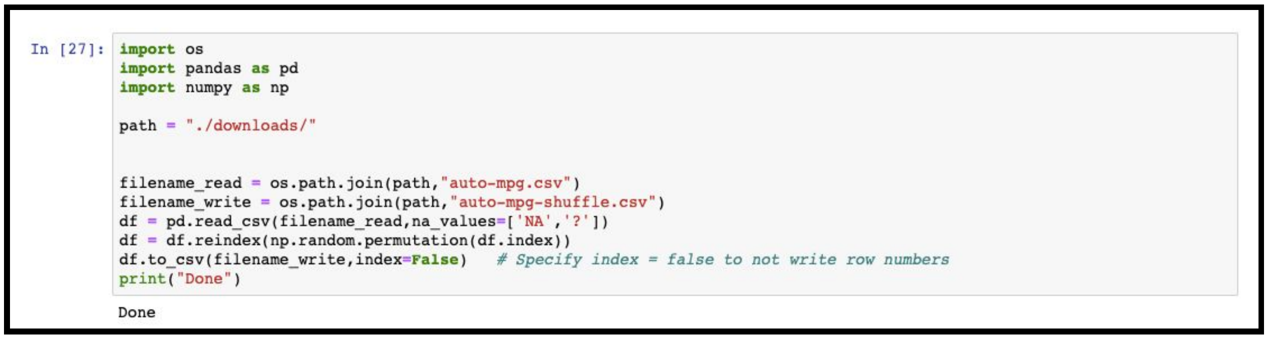


```
In [25]: print("The first car is: {}".format(df['name'].iloc[0]))

         The first car is: amc ambassador brougham
```

```
In [25]: print("The first car is: {}".format(df['name'].iloc[0]))

         The first car is: amc ambassador brougham
```

7. Saving a Dataframes

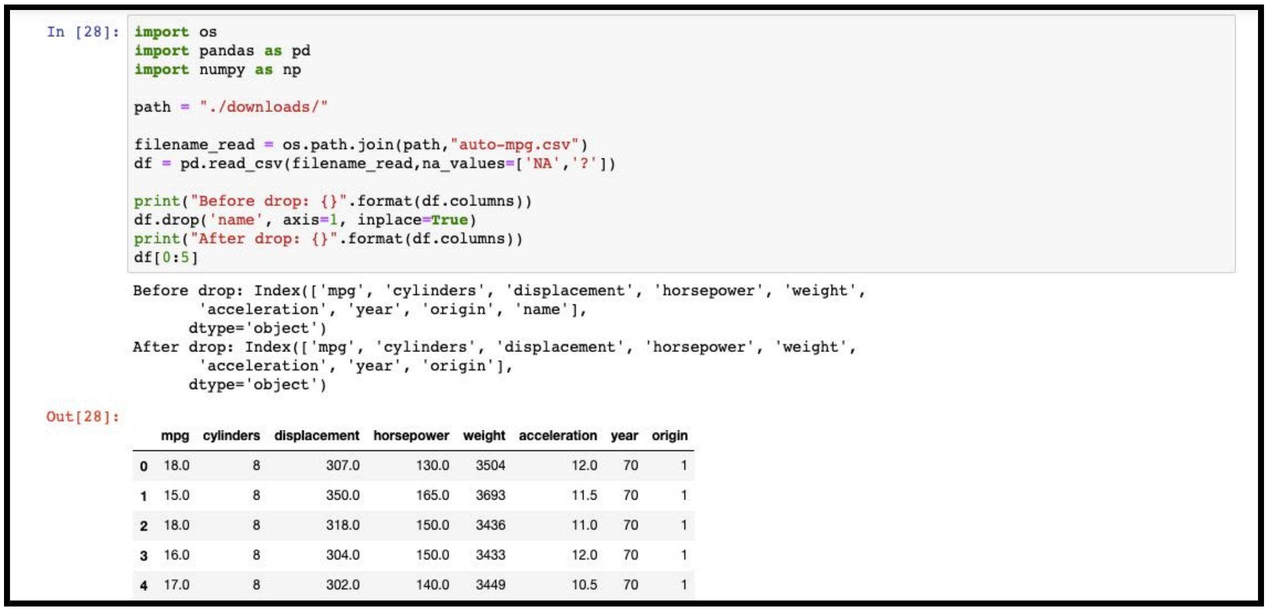
The code below will shuffle and then save a new copy to the data.



```
In [27]: import os
         import pandas as pd
         import numpy as np

         path = "./downloads/"


         filename_read = os.path.join(path,"auto-mpg.csv")
         filename_write = os.path.join(path,"auto-mpg-shuffle.csv")
         df = pd.read_csv(filename_read,na_values=['NA','?'])
         df = df.reindex(np.random.permutation(df.index))
         df.to_csv(filename_write,index=False)   # Specify index = false to not write row numbers
         print("Done")

         Done
```

8. Dropping Fields

When fields are of no value to the data set we can drop them. The code below removes the column from the MPG dataset.

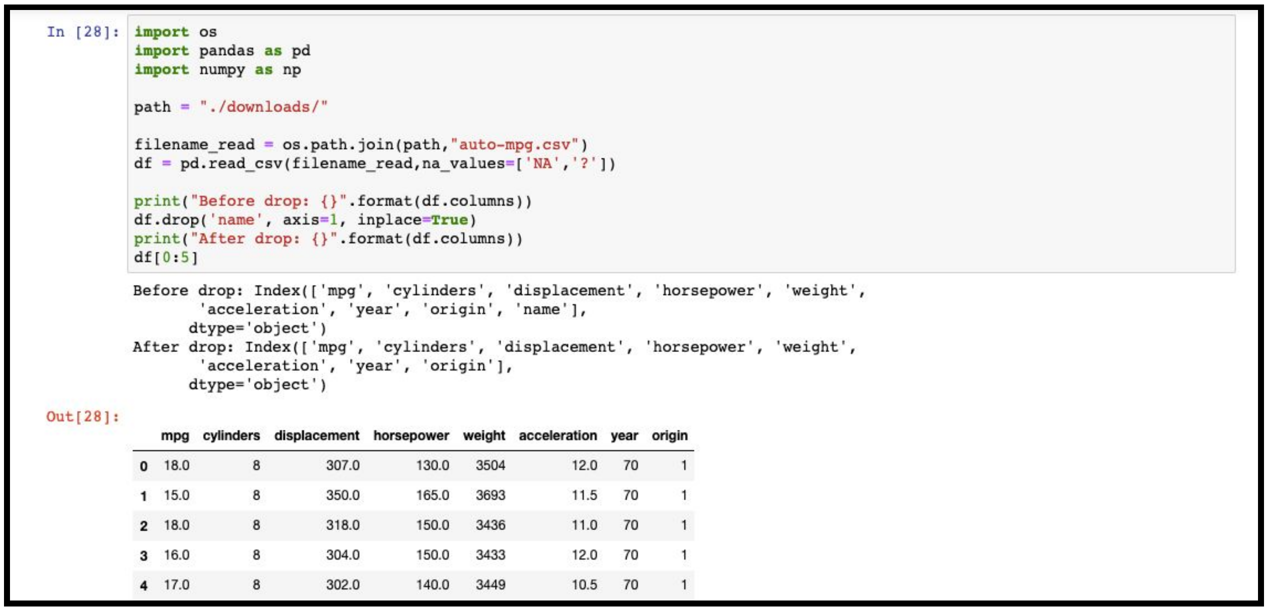


```
In [28]: import os
         import pandas as pd
         import numpy as np

         path = "./downloads/"

         filename_read = os.path.join(path,"auto-mpg.csv")
         df = pd.read_csv(filename_read,na_values=['NA','?'])

         print("Before drop: {}".format(df.columns))
         df.drop('name', axis=1, inplace=True)
         print("After drop: {}".format(df.columns))
         df[0:5]

         Before drop: Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
                'acceleration', 'year', 'origin', 'name'],
               dtype='object')
         After drop: Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
                'acceleration', 'year', 'origin'],
               dtype='object')

Out[28]:
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin
0	18.0	8	307.0	130.0	3504	12.0	70	1
1	15.0	8	350.0	165.0	3693	11.5	70	1
2	18.0	8	318.0	150.0	3436	11.0	70	1
3	16.0	8	304.0	150.0	3433	12.0	70	1
4	17.0	8	302.0	140.0	3449	10.5	70	1

9. Calculated Fields

Calculated fields can be added as new fields to the data frame that are calculated from our data. We can create a new column and give the weight in kilograms.



```
In [28]: import os
         import pandas as pd
         import numpy as np

         path = "./downloads/"

         filename_read = os.path.join(path,"auto-mpg.csv")
         df = pd.read_csv(filename_read,na_values=['NA','?'])

         print("Before drop: {}".format(df.columns))
         df.drop('name', axis=1, inplace=True)
         print("After drop: {}".format(df.columns))
         df[0:5]
```

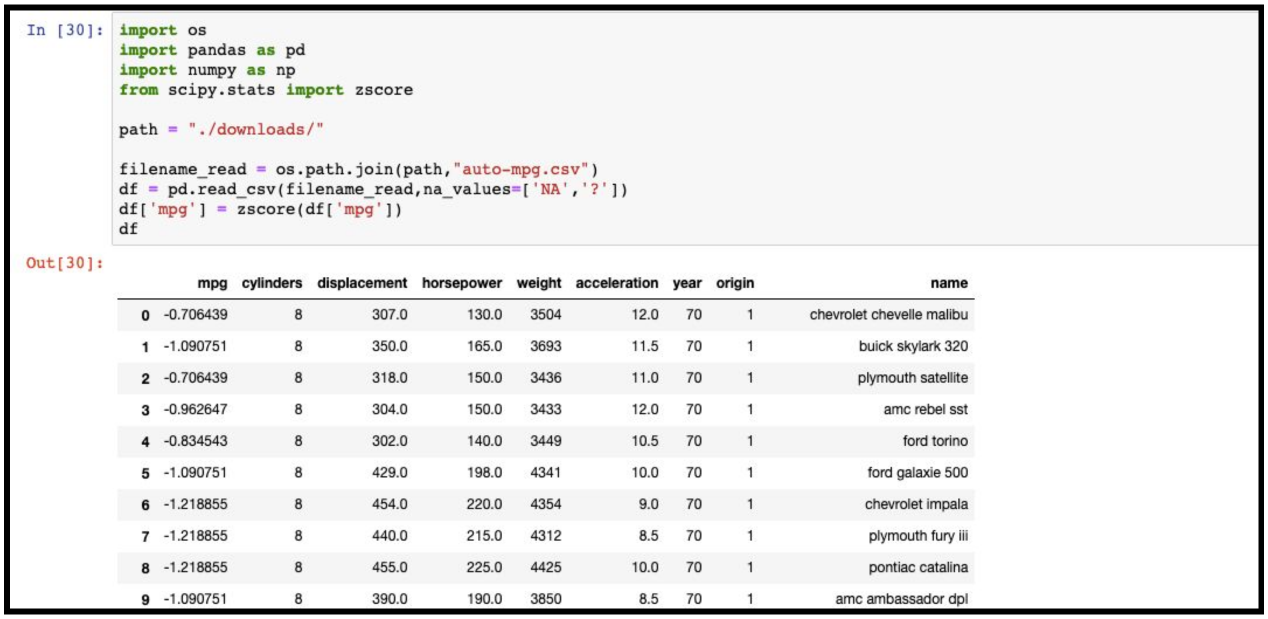
```
Before drop: Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'year', 'origin', 'name'],
      dtype='object')
After drop: Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'year', 'origin'],
      dtype='object')
```

Out[28]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin
0	18.0	8	307.0	130.0	3504	12.0	70	1
1	15.0	8	350.0	165.0	3693	11.5	70	1
2	18.0	8	318.0	150.0	3436	11.0	70	1
3	16.0	8	304.0	150.0	3433	12.0	70	1
4	17.0	8	302.0	140.0	3449	10.5	70	1

10. Feature Normalization

Feature normalization data allows us to put numbers in standard form so that two values can be compared. This code replaces the mpg with a z-score. Cars with average MPG wil be near zero is above average. And below zero is below average. Outliers at >3 or <-3



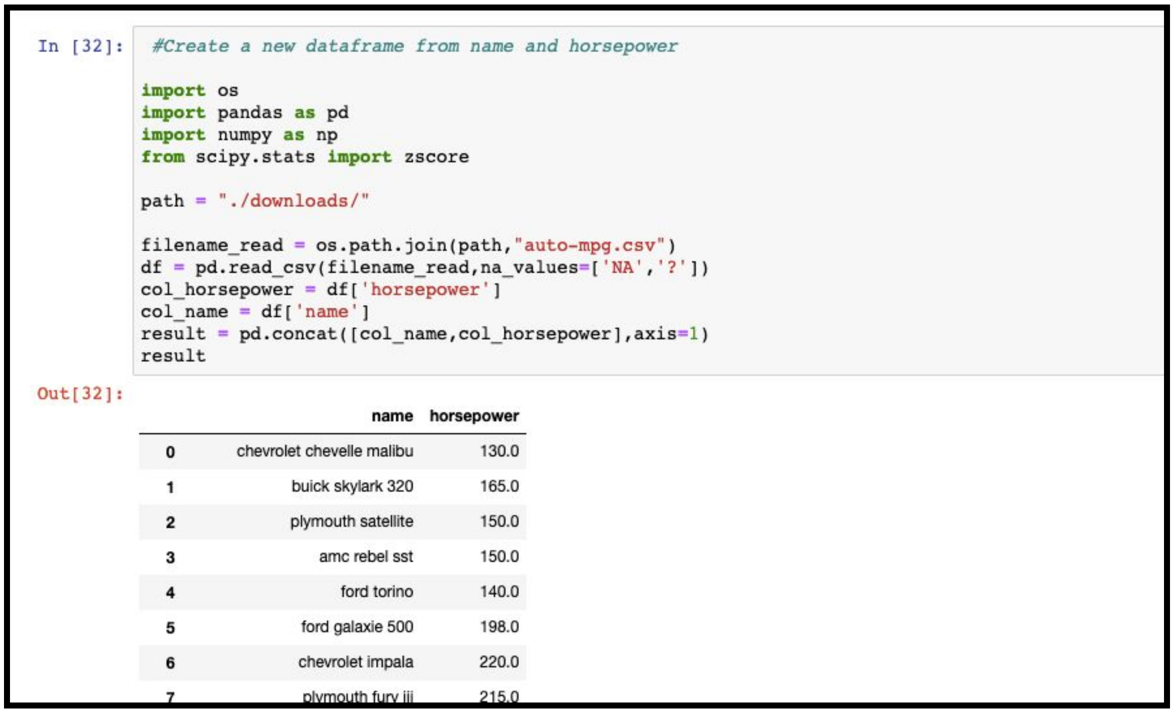
```
In [30]: import os
         import pandas as pd
         import numpy as np
         from scipy.stats import zscore

         path = "./downloads/"

         filename_read = os.path.join(path,"auto-mpg.csv")
         df = pd.read_csv(filename_read,na_values=['NA','?'])
         df['mpg'] = zscore(df['mpg'])
         df
```

Out[30]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	-0.706439	8	307.0	130.0	3504	12.0	70	1	chevrolet chevelle malibu
1	-1.090751	8	350.0	165.0	3693	11.5	70	1	buick skylark 320
2	-0.706439	8	318.0	150.0	3436	11.0	70	1	plymouth satellite
3	-0.962647	8	304.0	150.0	3433	12.0	70	1	amc rebel sst
4	-0.834543	8	302.0	140.0	3449	10.5	70	1	ford torino
5	-1.090751	8	429.0	198.0	4341	10.0	70	1	ford galaxie 500
6	-1.218855	8	454.0	220.0	4354	9.0	70	1	chevrolet impala
7	-1.218855	8	440.0	215.0	4312	8.5	70	1	plymouth fury iii
8	-1.218855	8	455.0	225.0	4425	10.0	70	1	pontiac catalina
9	-1.090751	8	390.0	190.0	3850	8.5	70	1	amc ambassador dpl

12. Concatenating Rows and Columns

This data shows how rows and columns can be concatenated together to form new data frames.



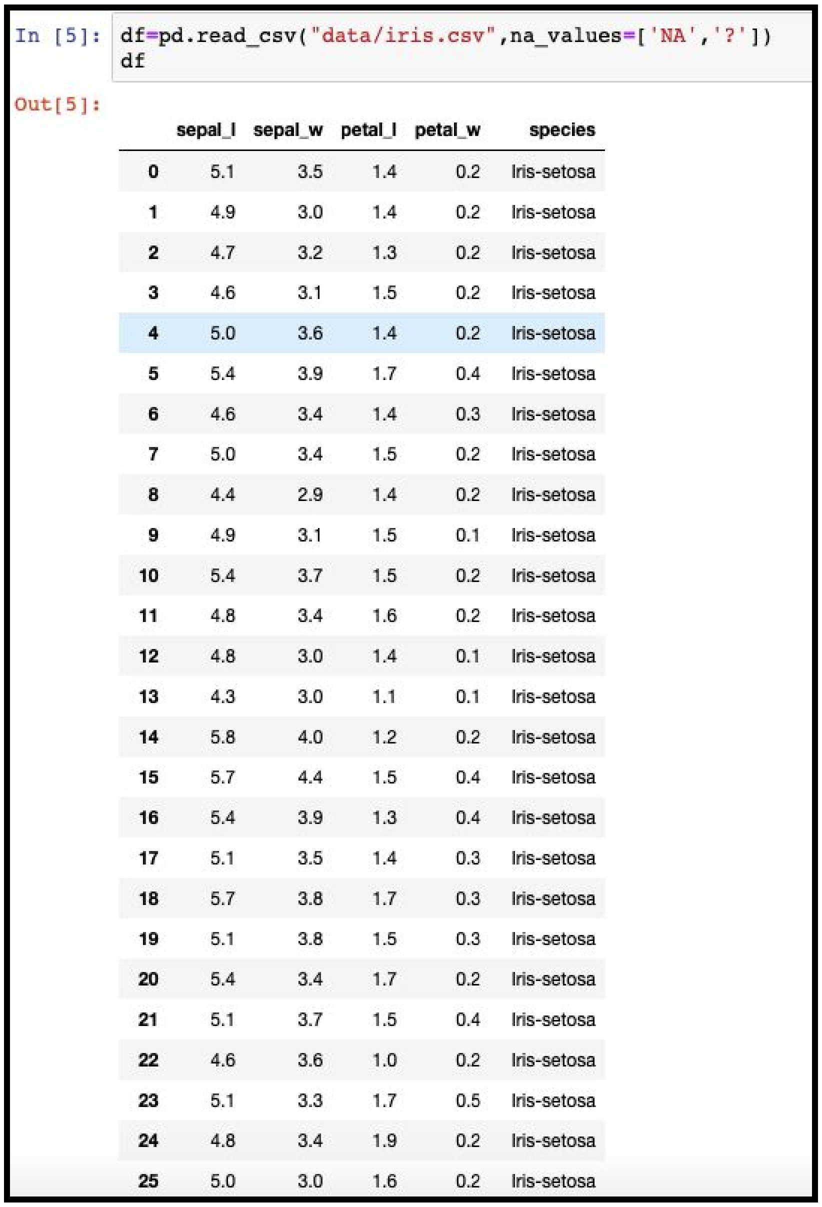
```
In [32]: #Create a new dataframe from name and horsepower

         import os
         import pandas as pd
         import numpy as np
         from scipy.stats import zscore

         path = "./downloads/"

         filename_read = os.path.join(path,"auto-mpg.csv")
         df = pd.read_csv(filename_read,na_values=['NA','?'])
         col_horsepower = df['horsepower']
         col_name = df['name']
         result = pd.concat([col_name,col_horsepower],axis=1)
         result
Out[32]:
```

	name	horsepower
0	chevrolet chevelle malibu	130.0
1	buick skylark 320	165.0
2	plymouth satellite	150.0
3	amc rebel sst	150.0
4	ford torino	140.0
5	ford galaxie 500	198.0
6	chevrolet impala	220.0
7	plymouth fury iii	215.0

13. Tensorflow

- Tensorflow allows you to build the feature vector in the format expected from raw data.
  1. Encoding Data
     - encode_text_dummy
     - Encode_text_index
  2. Normalizing Data
     - Encode_numberic_zscore
     - Encode_numeric_range
  3. Dealing with missing data:
     - missing_median
  4. Removing outliers:
     - Remove_outliers
  5. Create feature vector and target vector
     - To_xy
  6. Other utility functions:
     - Hms_string
     - chart_regression

14. Label encoding, one hot encoding, creating X/Y for TensorFlow

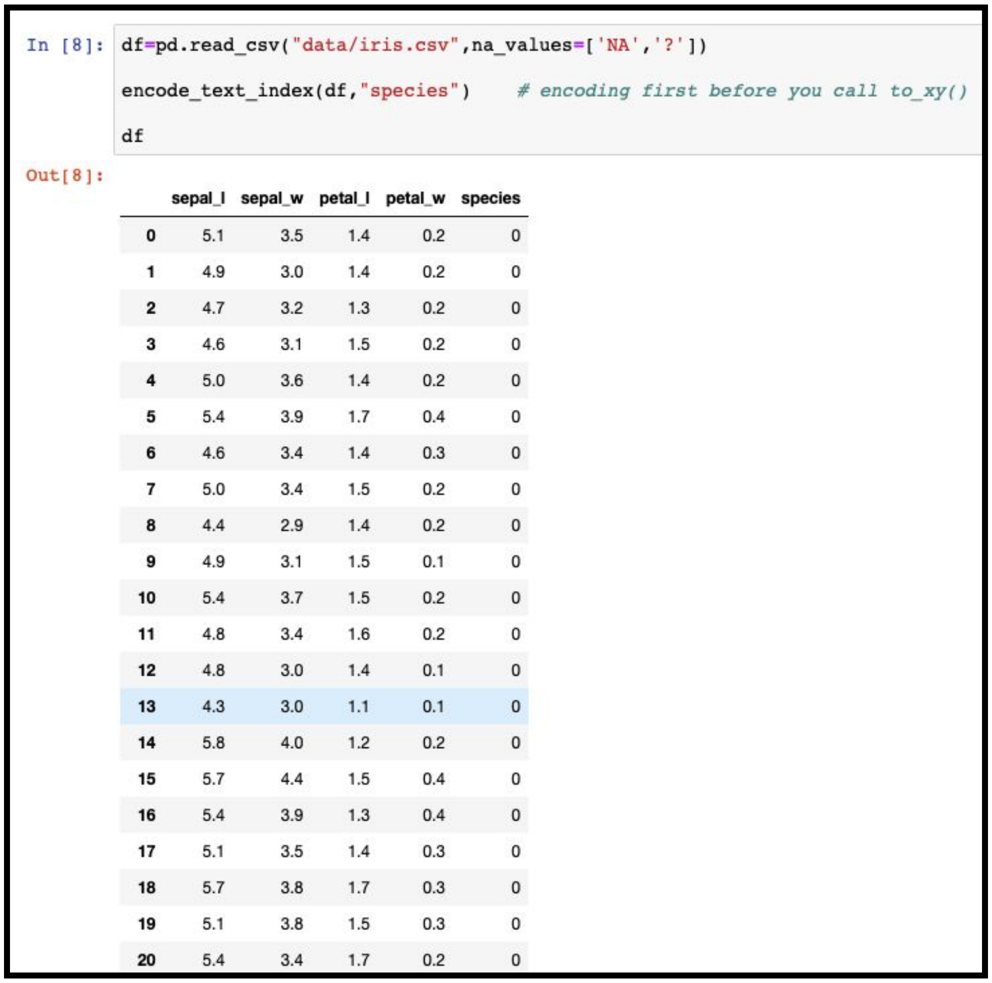


```
In [5]: df=pd.read_csv("data/iris.csv",na_values=['NA','?'])
        df
```

Out[5]:

	sepal_l	sepal_w	petal_l	petal_w	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa
10	5.4	3.7	1.5	0.2	Iris-setosa
11	4.8	3.4	1.6	0.2	Iris-setosa
12	4.8	3.0	1.4	0.1	Iris-setosa
13	4.3	3.0	1.1	0.1	Iris-setosa
14	5.8	4.0	1.2	0.2	Iris-setosa
15	5.7	4.4	1.5	0.4	Iris-setosa
16	5.4	3.9	1.3	0.4	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
18	5.7	3.8	1.7	0.3	Iris-setosa
19	5.1	3.8	1.5	0.3	Iris-setosa
20	5.4	3.4	1.7	0.2	Iris-setosa
21	5.1	3.7	1.5	0.4	Iris-setosa
22	4.6	3.6	1.0	0.2	Iris-setosa
23	5.1	3.3	1.7	0.5	Iris-setosa
24	4.8	3.4	1.9	0.2	Iris-setosa
25	5.0	3.0	1.6	0.2	Iris-setosa

15.Encoding labels before calling to_xy()

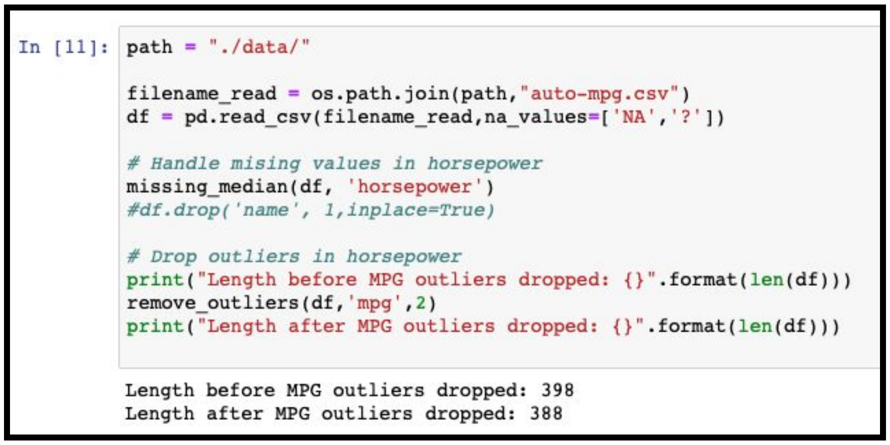


```
In [8]: df=pd.read_csv("data/iris.csv",na_values=['NA','?'])

        encode_text_index(df,"species")    # encoding first before you call to_xy()

        df
```

Out[8]:

	sepal_l	sepal_w	petal_l	petal_w	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
5	5.4	3.9	1.7	0.4	0
6	4.6	3.4	1.4	0.3	0
7	5.0	3.4	1.5	0.2	0
8	4.4	2.9	1.4	0.2	0
9	4.9	3.1	1.5	0.1	0
10	5.4	3.7	1.5	0.2	0
11	4.8	3.4	1.6	0.2	0
12	4.8	3.0	1.4	0.1	0
13	4.3	3.0	1.1	0.1	0
14	5.8	4.0	1.2	0.2	0
15	5.7	4.4	1.5	0.4	0
16	5.4	3.9	1.3	0.4	0
17	5.1	3.5	1.4	0.3	0
18	5.7	3.8	1.7	0.3	0
19	5.1	3.8	1.5	0.3	0
20	5.4	3.4	1.7	0.2	0

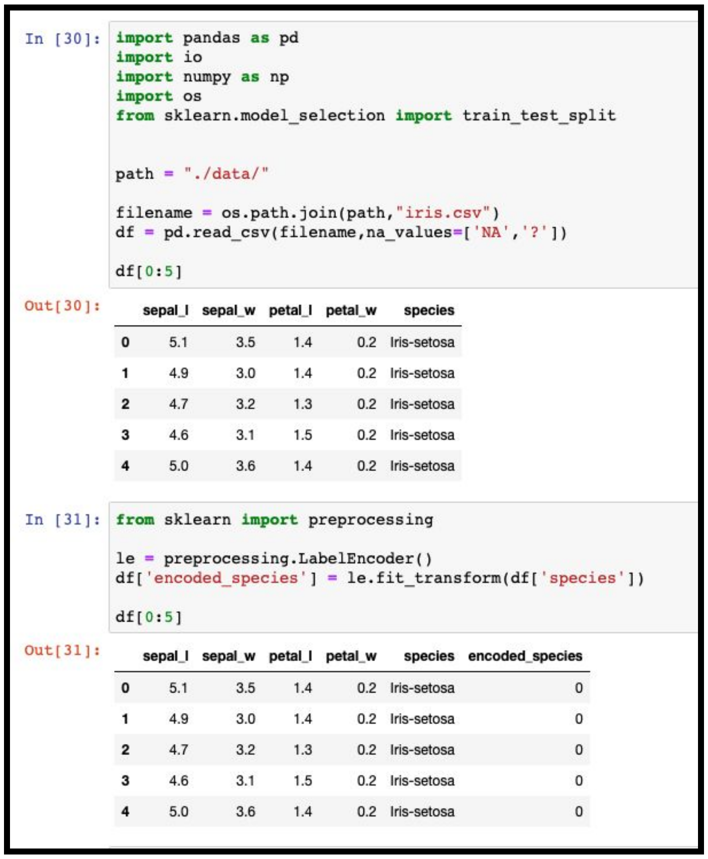
16. Example of Deal with Missing values and outliers



```
In [11]: path = "./data/"

         filename_read = os.path.join(path,"auto-mpg.csv")
         df = pd.read_csv(filename_read,na_values=['NA','?'])

         # Handle mising values in horsepower
         missing_median(df, 'horsepower')
         #df.drop('name', 1,inplace=True)

         # Drop outliers in horsepower
         print("Length before MPG outliers dropped: {}".format(len(df)))
         remove_outliers(df,'mpg',2)
         print("Length after MPG outliers dropped: {}".format(len(df)))

         Length before MPG outliers dropped: 398
         Length after MPG outliers dropped: 388
```

17. Training/Test split

Training data / Test Split means that the data are split according to a ratio with a training and validation set. Common ratios are 80% training and 20% validation.



```
In [30]: import pandas as pd
         import io
         import numpy as np
         import os
         from sklearn.model_selection import train_test_split


         path = "./data/"

         filename = os.path.join(path,"iris.csv")
         df = pd.read_csv(filename,na_values=['NA','?'])

         df[0:5]
```

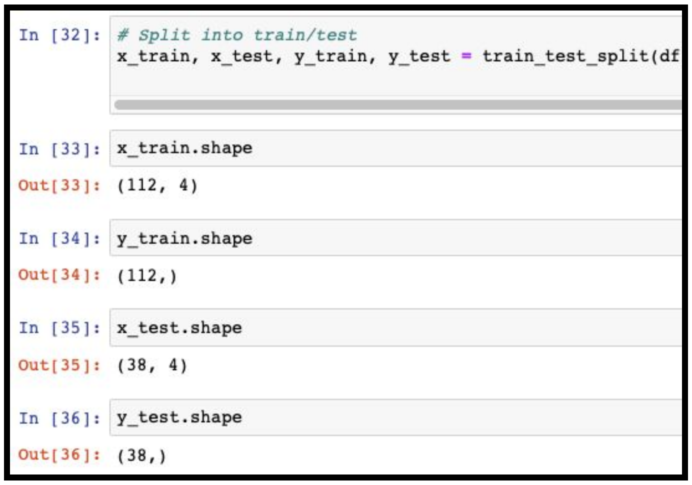
Out[30]:

	sepal_l	sepal_w	petal_l	petal_w	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
In [31]: from sklearn import preprocessing

         le = preprocessing.LabelEncoder()
         df['encoded_species'] = le.fit_transform(df['species'])

         df[0:5]
```

Out[31]:

	sepal_l	sepal_w	petal_l	petal_w	species	encoded_species
0	5.1	3.5	1.4	0.2	Iris-setosa	0
1	4.9	3.0	1.4	0.2	Iris-setosa	0
2	4.7	3.2	1.3	0.2	Iris-setosa	0
3	4.6	3.1	1.5	0.2	Iris-setosa	0
4	5.0	3.6	1.4	0.2	Iris-setosa	0

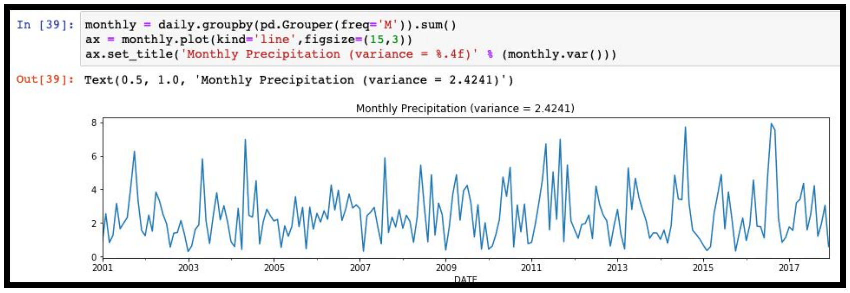


```
In [32]: # Split into train/test
         x_train, x_test, y_train, y_test = train_test_split(df

In [33]: x_train.shape
Out[33]: (112, 4)

In [34]: y_train.shape
Out[34]: (112,)

In [35]: x_test.shape
Out[35]: (38, 4)

In [36]: y_test.shape
Out[36]: (38,)
```

This training data shows how the training set uses 80% of the data test validation set uses 20%.

18. Aggregation

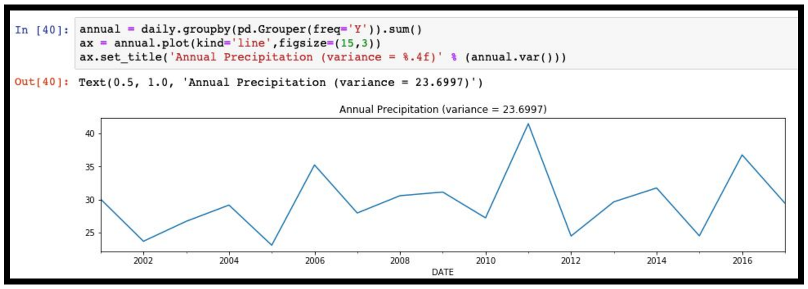
- Data preprocessing where values of two or more objects are combined.
- Data aggregation can help reduce the size of data, change granularity of analysis, and improve stability of the data.
- Grouped by month



```
In [39]: monthly = daily.groupby(pd.Grouper(freq='M')).sum()
         ax = monthly.plot(kind='line',figsize=(15,3))
         ax.set_title('Monthly Precipitation (variance = %.4f)' % (monthly.var()))

Out[39]: Text(0.5, 1.0, 'Monthly Precipitation (variance = 2.4241)')
```

-

- Grouped by year

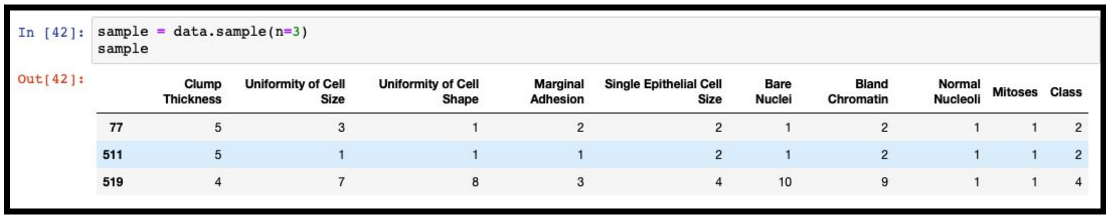


```
In [40]: annual = daily.groupby(pd.Grouper(freq='Y')).sum()
         ax = annual.plot(kind='line',figsize=(15,3))
         ax.set_title('Annual Precipitation (variance = %.4f)' % (annual.var()))

Out[40]: Text(0.5, 1.0, 'Annual Precipitation (variance = 23.6997)')
```

-

19. Sampling: 1, data reduction for exploratory data analysis and scaling up algorithms to big data applications and 2, quantify uncertainties due to varying data distributions.

- Sample size 3



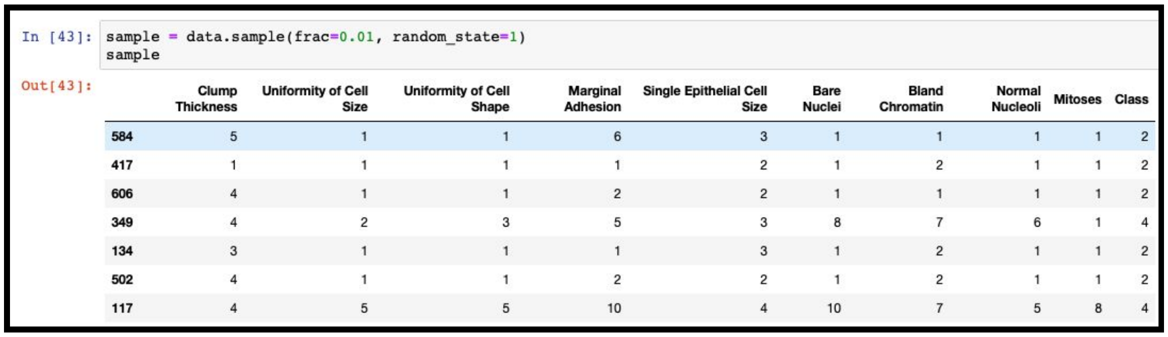
```
In [42]: sample = data.sample(n=3)
         sample
```

Out[42]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
77	5	3	1	2	2	1	2	1	1	2
511	5	1	1	1	2	1	2	1	1	2
519	4	7	8	3	4	10	9	1	1	4

-

- Randomly select 1% of data



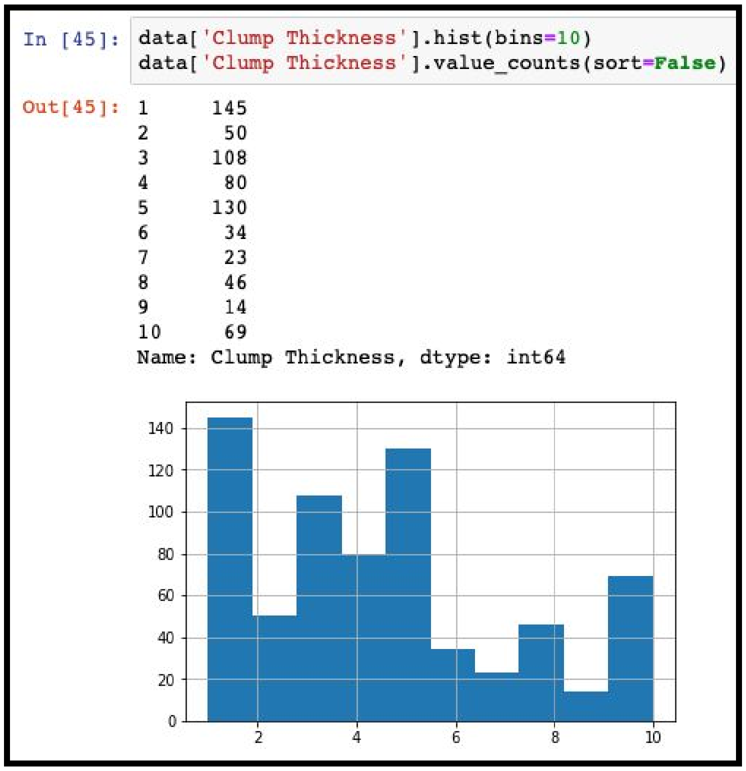
```
In [43]: sample = data.sample(frac=0.01, random_state=1)
         sample
```

Out[43]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
584	5	1	1	6	3	1	1	1	1	2
417	1	1	1	1	2	1	2	1	1	2
606	4	1	1	2	2	1	1	1	1	2
349	4	2	3	5	3	8	7	6	1	4
134	3	1	1	1	3	1	2	1	1	2
502	4	1	1	2	2	1	2	1	1	2
117	4	5	5	10	4	10	7	5	8	4

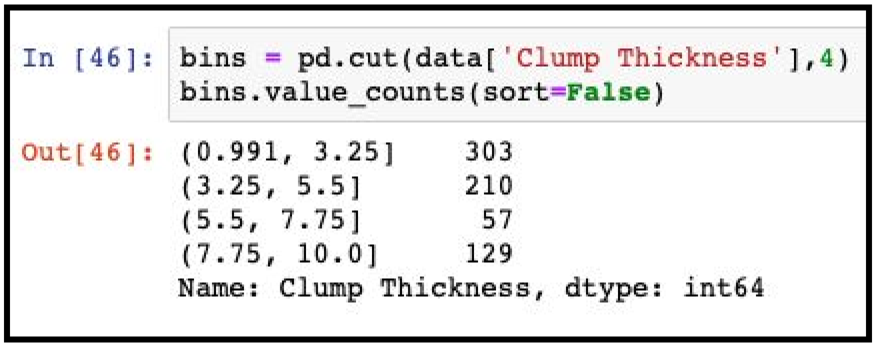
-

20. Discretization: data preprocessing step used to transform a continuous-valued attribute to a categorical attribute.



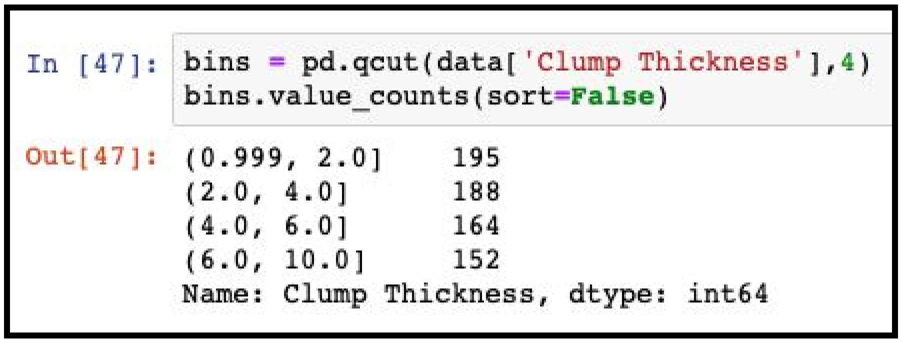
```
In [45]: data['Clump Thickness'].hist(bins=10)
         data['Clump Thickness'].value_counts(sort=False)

Out[45]: 1     145
         2      50
         3     108
         4      80
         5     130
         6      34
         7      23
         8      46
         9      14
         10     69
         Name: Clump Thickness, dtype: int64
```

-



```
In [46]: bins = pd.cut(data['Clump Thickness'],4)
         bins.value_counts(sort=False)

Out[46]: (0.991, 3.25]    303
         (3.25, 5.5]      210
         (5.5, 7.75]       57
         (7.75, 10.0]     129
         Name: Clump Thickness, dtype: int64
```

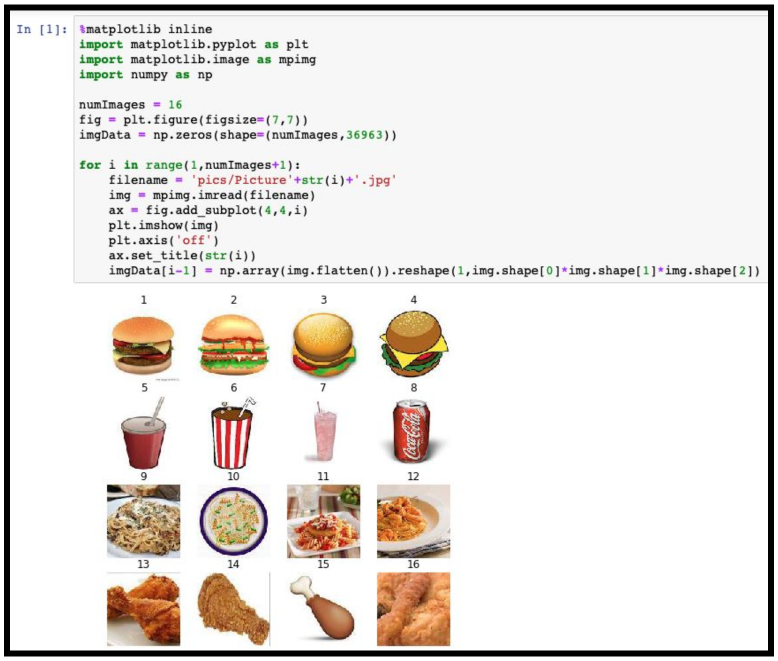
-



```
In [47]: bins = pd.qcut(data['Clump Thickness'],4)
         bins.value_counts(sort=False)

Out[47]: (0.999, 2.0]    195
         (2.0, 4.0]      188
         (4.0, 6.0]      164
         (6.0, 10.0]     152
         Name: Clump Thickness, dtype: int64
```

-

21. Principal Component Analysis: PCA is a method for reducing the number of attributes in the data by projecting the data from its original high-dimensional space into a lower dimensional space. The new components have: 1. Linear combinations of the original attributes, 2. Perpendicular to each other, 3. They capture the maximum amount of variation in the data.



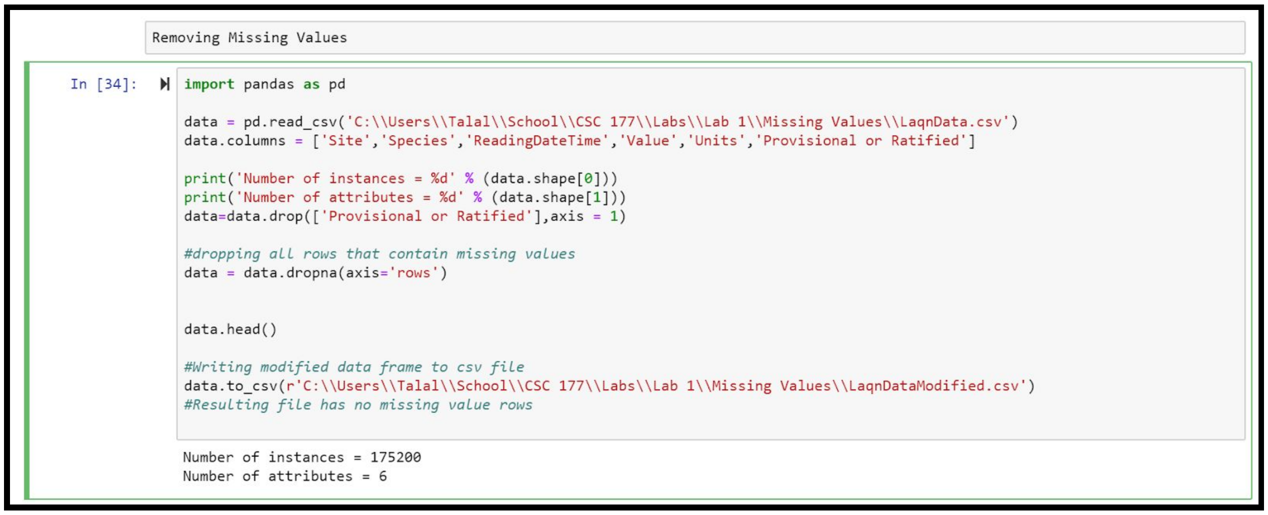
```
In [1]: %matplotlib inline
        import matplotlib.pyplot as plt
        import matplotlib.image as mpimg
        import numpy as np

        numImages = 16
        fig = plt.figure(figsize=(7,7))
        imgData = np.zeros(shape=(numImages,36963))

        for i in range(1,numImages+1):
            filename = 'pics/Picture'+str(i)+'.jpg'
            img = mpimg.imread(filename)
            ax = fig.add_subplot(4,4,i)
            plt.imshow(img)
            plt.axis('off')
            ax.set_title(str(i))
            imgData[i-1] = np.array(img.flatten()).reshape(1,img.shape[0]*img.shape[1]*img.shape[2])
```

-

Pre-Processing of our Data:

From page 17 onward, we start the Pre-Processing of the datasets that we found online. Those datasets can be found on Page 2 of this document as hyperlinks.

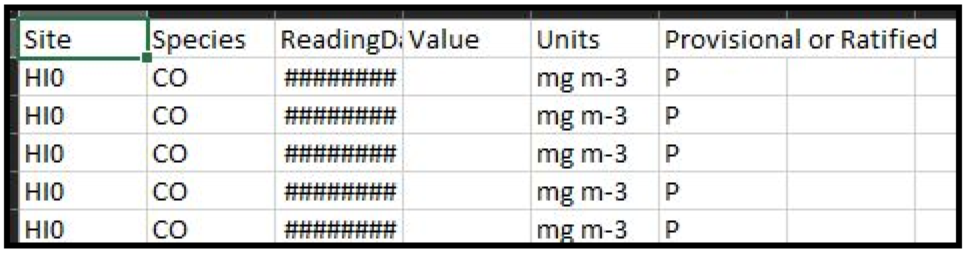
Missing Values - Talal (London Air)



```
Removing Missing Values

In [34]:  import pandas as pd

          data = pd.read_csv('C:\\Users\\Talal\\School\\CSC 177\\Labs\\Lab 1\\Missing Values\\LaqnData.csv')
          data.columns = ['Site','Species','ReadingDateTime','Value','Units','Provisional or Ratified']

          print('Number of instances = %d' % (data.shape[0]))
          print('Number of attributes = %d' % (data.shape[1]))
          data=data.drop(['Provisional or Ratified'],axis = 1)

          #dropping all rows that contain missing values
          data = data.dropna(axis='rows')


          data.head()

          #Writing modified data frame to csv file
          data.to_csv(r'C:\\Users\\Talal\\School\\CSC 177\\Labs\\Lab 1\\Missing Values\\LaqnDataModified.csv')
          #Resulting file has no missing value rows

          Number of instances = 175200
          Number of attributes = 6
```

Original Dataset (with missing values):

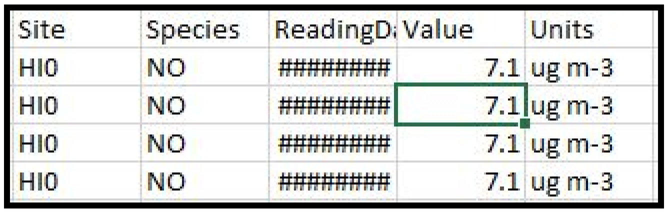
In this dataset, multiple columns are left blank. The code above is meant to drop all of the columns that have missing values.



Site	Species	ReadingD:	Value	Units	Provisional or Ratified
HI0	CO	########		mg m-3	P
HI0	CO	########		mg m-3	P
HI0	CO	########		mg m-3	P
HI0	CO	########		mg m-3	P
HI0	CO	########		mg m-3	P

Modified Dataset (with missing values and last column removed):

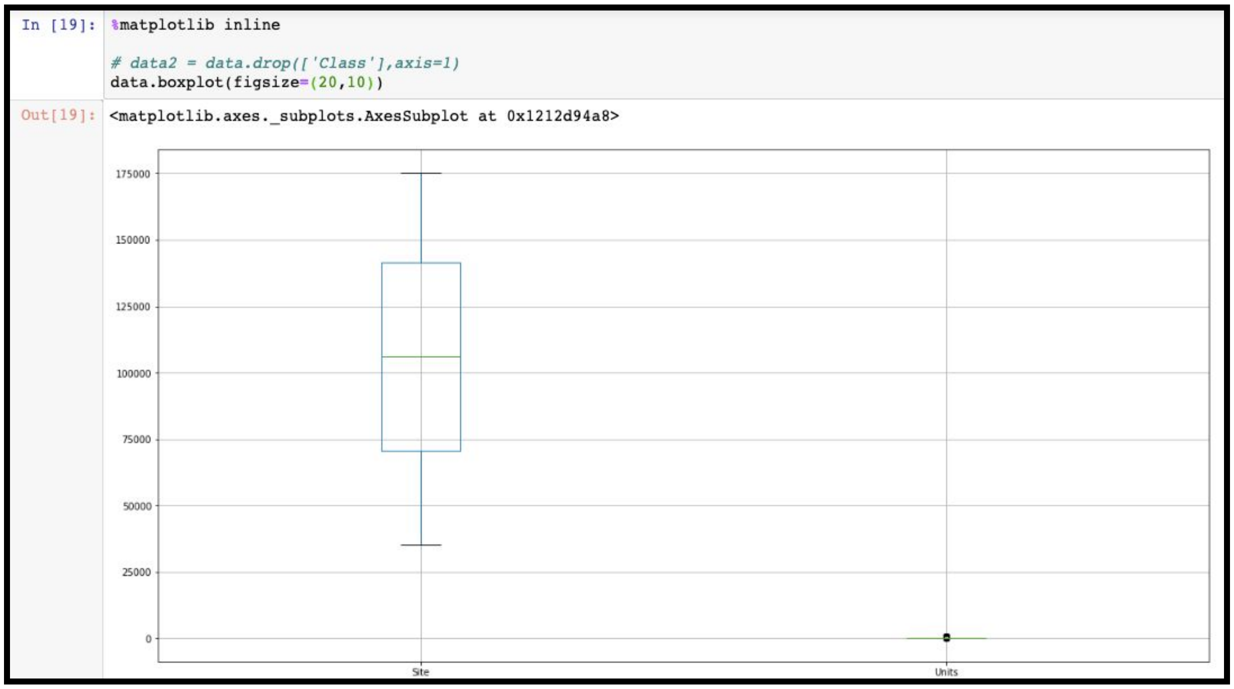
After running the code, the columns that were left blank were completely removed, leaving us with fewer columns to use for our analysis.



Site	Species	ReadingD:	Value	Units
HI0	NO	########	7.1	ug m-3
HI0	NO	########	7.1	ug m-3
HI0	NO	########	7.1	ug m-3
HI0	NO	########	7.1	ug m-3

Outliers - Dianna

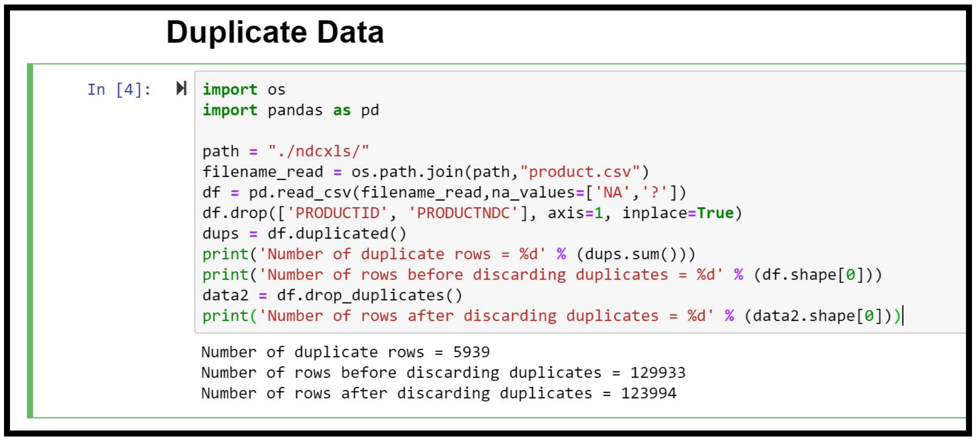
Outliers are data instances where the characteristics are notably different from the rest of the data. After running the code we can see that there were some outliers far above and below our data.



```
In [19]: %matplotlib inline

         # data2 = data.drop(['Class'],axis=1)
         data.boxplot(figsize=(20,10))

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1212d94a8>
```

Duplicate Data - Tom (Drugs)

- The .drop_duplicates() method will delete all duplicate rows, leaving us with a lot less rows to analyze in our data set.



Duplicate Data

In [4]:

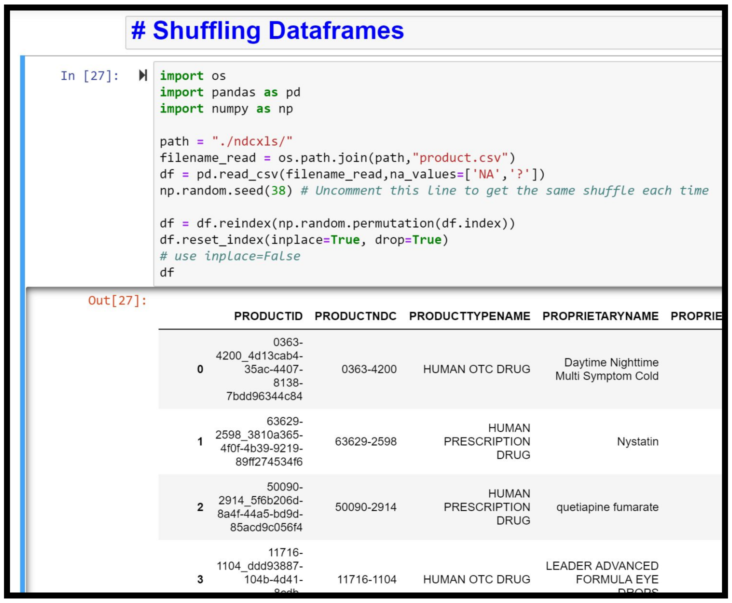
```
import os
import pandas as pd

path = "./ndcxls/"
filename_read = os.path.join(path,"product.csv")
df = pd.read_csv(filename_read,na_values=['NA','?'])
df.drop(['PRODUCTID', 'PRODUCTNDC'], axis=1, inplace=True)
dups = df.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))
print('Number of rows before discarding duplicates = %d' % (df.shape[0]))
data2 = df.drop_duplicates()
print('Number of rows after discarding duplicates = %d' % (data2.shape[0]))
```

```
Number of duplicate rows = 5939
Number of rows before discarding duplicates = 129933
Number of rows after discarding duplicates = 123994
```

- After running the code, it clearly states that the number of rows before and after running the code went from “129933” to “123994”, which is about 6000 rows that we deleted. This is significant, because it is about 5% less data for us to look at after Pre-Processing.

Shuffling Data Frames- Tom (Drugs)

- You can shuffle rows in your dataframe around in order to get a cleaner-looking dataset



Shuffling Dataframes

In [27]:

```
import os
import pandas as pd
import numpy as np

path = "./ndcxls/"
filename_read = os.path.join(path,"product.csv")
df = pd.read_csv(filename_read,na_values=['NA','?'])
np.random.seed(38) # Uncomment this line to get the same shuffle each time

df = df.reindex(np.random.permutation(df.index))
df.reset_index(inplace=True, drop=True)
# use inplace=False
df
```

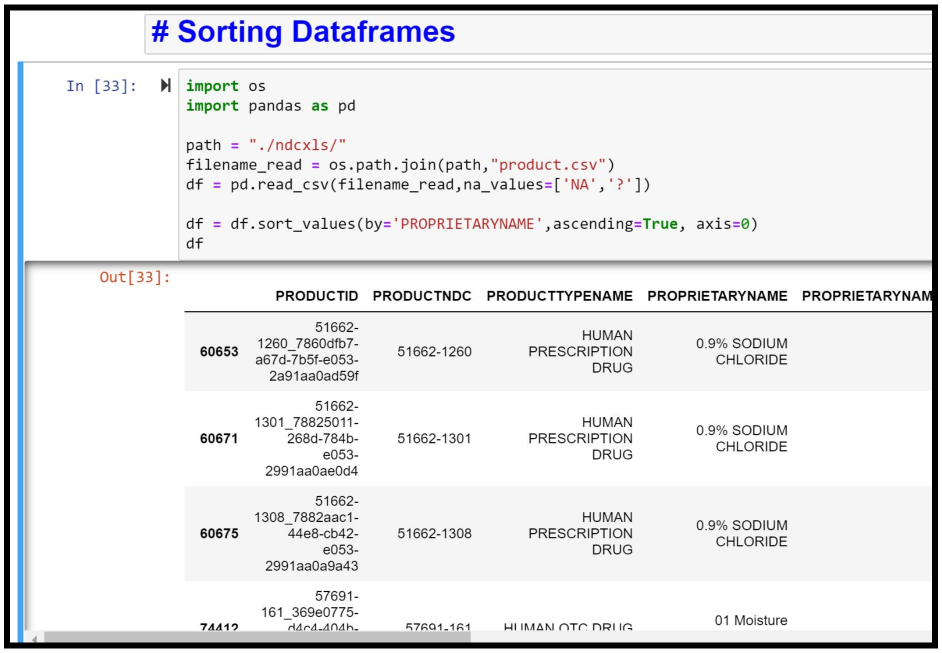
Out[27]:

	PRODUCTID	PRODUCTNDC	PRODUCTTYPENAME	PROPRIETARYNAME	PROPRIE
0	0363-4200_4d13cab4-35ac-4407-8138-7bdd96344c84	0363-4200	HUMAN OTC DRUG	Daytime Nighttime Multi Symptom Cold	
1	63629-2598_3810a365-4f0f-4b39-9219-89ff274534f6	63629-2598	HUMAN PRESCRIPTION DRUG	Nystatin	
2	50090-2914_5f6b206d-8a4f-44a5-bd9d-85acd9c056f4	50090-2914	HUMAN PRESCRIPTION DRUG	quetiapine fumarate	
3	11716-1104_ddd93887-104b-4d41-	11716-1104	HUMAN OTC DRUG	LEADER ADVANCED FORMULA EYE	

- After shuffling, the output of the data is easier for the user to read and analyze

Sorting Dataframes - Jimmy (Drugs)

- Columns can be sorted through the use of quicksort, mergesort, or heapsort. This is useful, because some data may be significant than other data for the purpose of analysis.



Sorting Dataframes

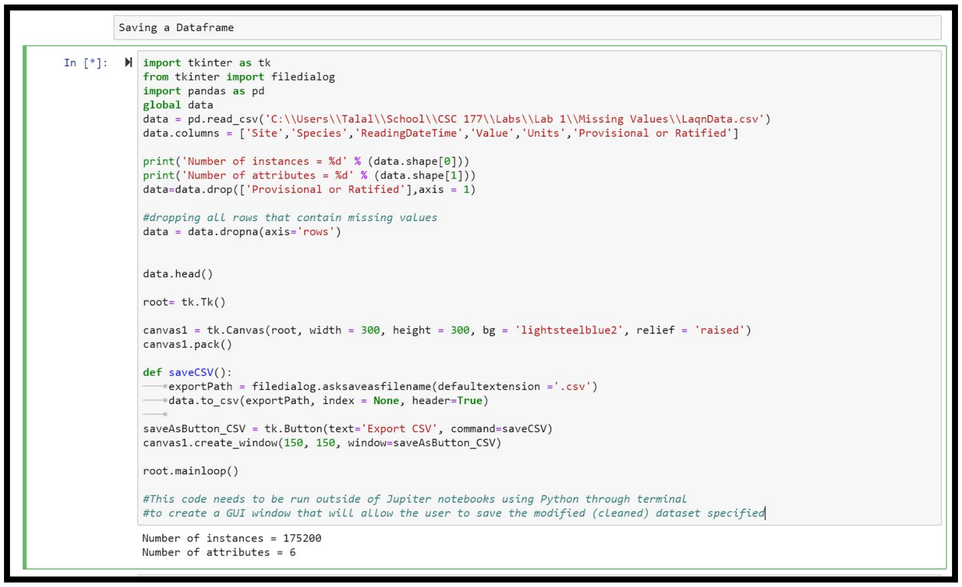
```
In [33]:
import os
import pandas as pd

path = "./ndcxls/"
filename_read = os.path.join(path,"product.csv")
df = pd.read_csv(filename_read,na_values=['NA','?'])

df = df.sort_values(by='PROPRIETARYNAME',ascending=True, axis=0)
df
```

Out[33]:

	PRODUCTID	PRODUCTNDC	PRODUCTTYPENAME	PROPRIETARYNAME	PROPRIETARYNAM
60653	51662-1260_7860dfb7-a67d-7b5f-e053-2a91aa0ad59f	51662-1260	HUMAN PRESCRIPTION DRUG	0.9% SODIUM CHLORIDE	
60671	51662-1301_78825011-268d-784b-e053-2991aa0ae0d4	51662-1301	HUMAN PRESCRIPTION DRUG	0.9% SODIUM CHLORIDE	
60675	51662-1308_7882aac1-44e8-cb42-e053-2991aa0a9a43	51662-1308	HUMAN PRESCRIPTION DRUG	0.9% SODIUM CHLORIDE	
74412	57691-161_369e0775-d4c4-404b-	57691-161	HUMAN OTC DRUG	01 Moisture	

- In this case, Proprietary Name was the column that was sorted, and this is because we were interested in the name of the chemical, rather than the actual ID, or Product Type

Saving a Dataframe - Talal (London Air)



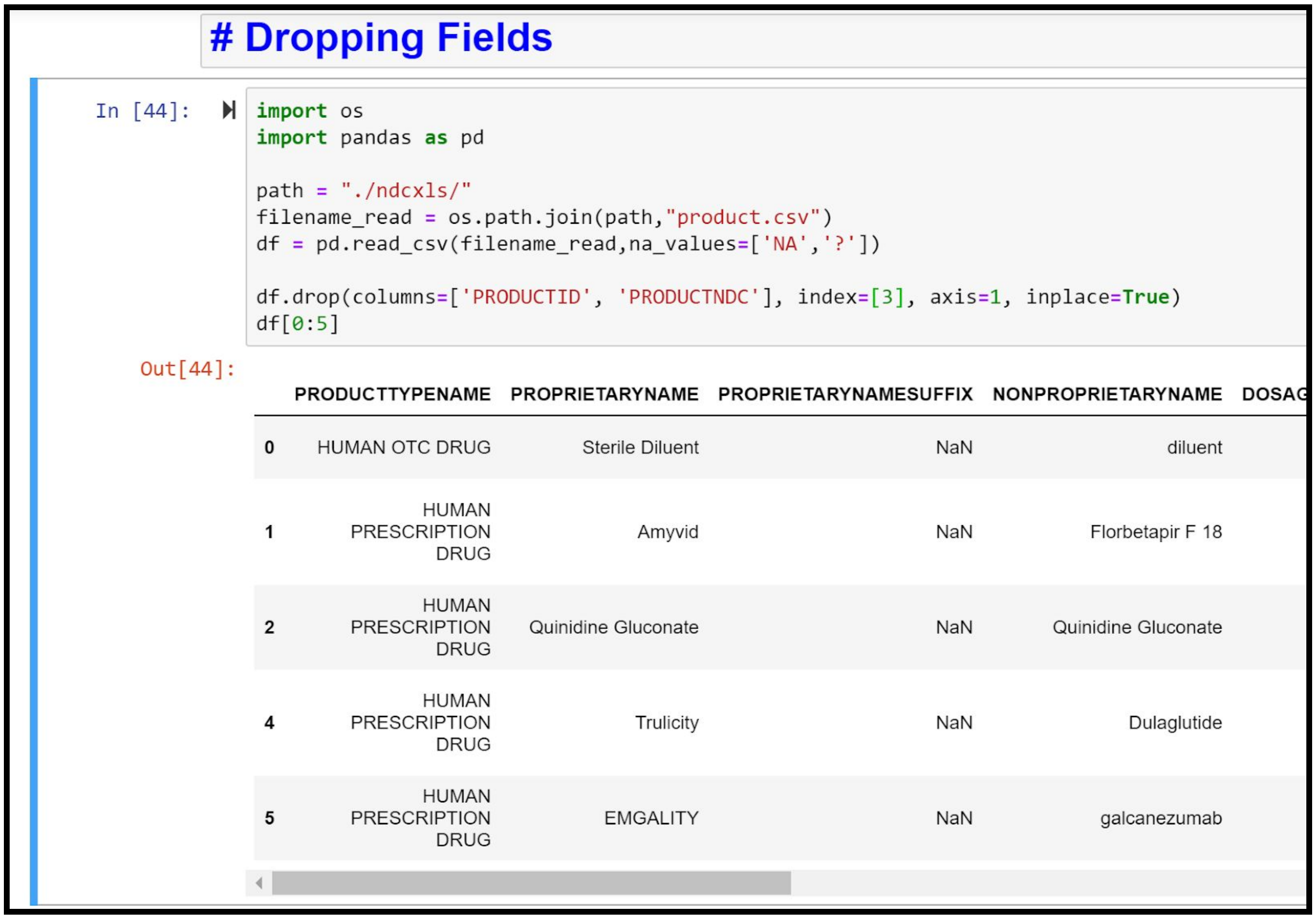
Saving a Dataframe

```
In [*]:
import tkinter as tk
from tkinter import filedialog
import pandas as pd
global data
data = pd.read_csv('C:\\Users\\Talal\\School\\CSC 177\\Labs\\Lab 1\\Missing Values\\LaqnData.csv')
data.columns = ['Site','Species','ReadingDateTime','Value','Units','Provisional or Ratified']

print('Number of instances = %d' % (data.shape[0]))
print('Number of attributes = %d' % (data.shape[1]))
data=data.drop(['Provisional or Ratified'],axis = 1)

#dropping all rows that contain missing values
data = data.dropna(axis='rows')


data.head()

root= tk.Tk()

canvas1 = tk.Canvas(root, width = 300, height = 300, bg = 'lightsteelblue2', relief = 'raised')
canvas1.pack()

def saveCSV():
    exportPath = filedialog.asksaveasfilename(defaultextension ='.csv')
    data.to_csv(exportPath, index = None, header=True)
    
saveAsButton_CSV = tk.Button(text='Export CSV', command=saveCSV)
canvas1.create_window(150, 150, window=saveAsButton_CSV)

root.mainloop()

#This code needs to be run outside of Jupiter notebooks using Python through terminal
#to create a GUI window that will allow the user to save the modified (cleaned) dataset specified
```

```
Number of instances = 175200
Number of attributes = 6
```

Dropping Fields - Jimmy (Drugs)

- You can drop multiple columns and rows with one line of code



Dropping Fields

In [44]:
```
import os
import pandas as pd

path = "./ndcxls/"
filename_read = os.path.join(path,"product.csv")
df = pd.read_csv(filename_read,na_values=['NA','?'])

df.drop(columns=['PRODUCTID', 'PRODUCTNDC'], index=[3], axis=1, inplace=True)
df[0:5]
```

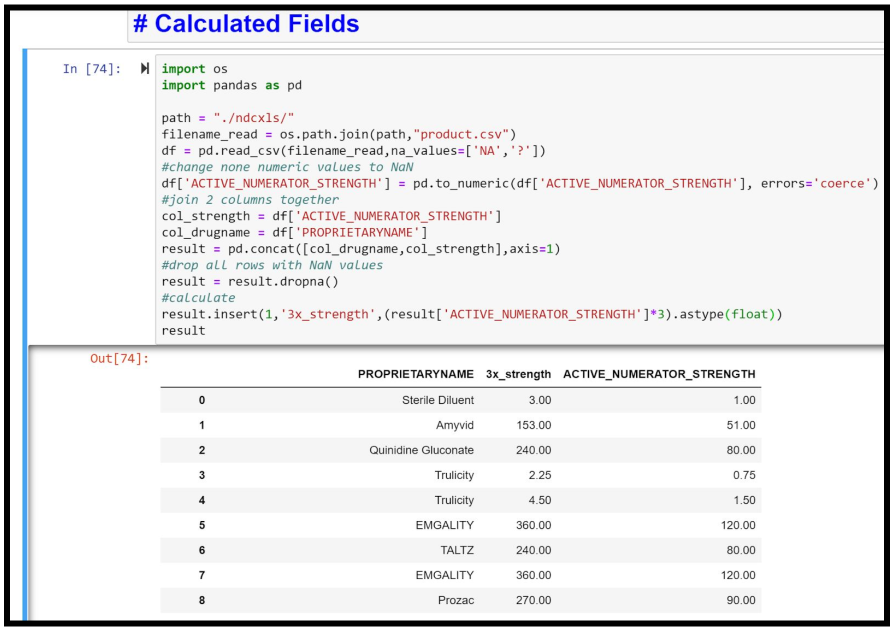
Out[44]:

	PRODUCTTYPENAME	PROPRIETARYNAME	PROPRIETARYNAMESUFFIX	NONPROPRIETARYNAME	DOSAG
0	HUMAN OTC DRUG	Sterile Diluent	NaN	diluent	
1	HUMAN PRESCRIPTION DRUG	Amyvid	NaN	Florbetapir F 18	
2	HUMAN PRESCRIPTION DRUG	Quinidine Gluconate	NaN	Quinidine Gluconate	
4	HUMAN PRESCRIPTION DRUG	Trulicity	NaN	Dulaglutide	
5	HUMAN PRESCRIPTION DRUG	EMGALITY	NaN	galcanezumab	

- We chose to drop the “PRODUCTID”, and the “PRODUCTNDC” fields, because we deemed that they were unnecessary fields, and did not provide useful information.

Calculated Fields - Amrit

- When doing calculations, all values must be numeric or you will get an error.



Calculated Fields

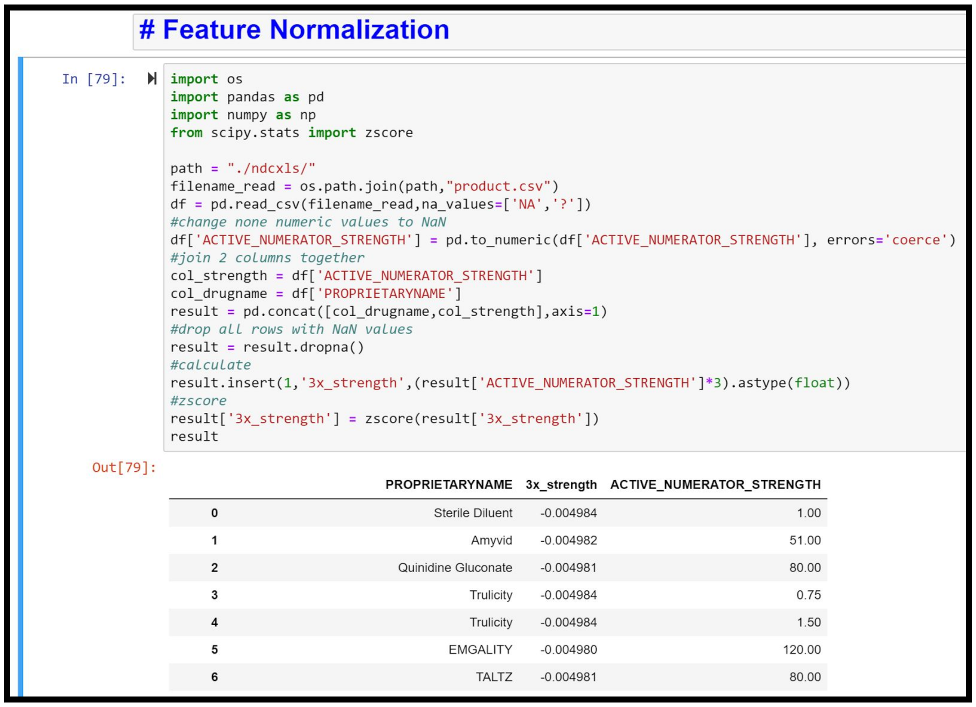
In [74]:
```
import os
import pandas as pd

path = "./ndcxls/"
filename_read = os.path.join(path,"product.csv")
df = pd.read_csv(filename_read,na_values=['NA','?'])
#change none numeric values to NaN
df['ACTIVE_NUMERATOR_STRENGTH'] = pd.to_numeric(df['ACTIVE_NUMERATOR_STRENGTH'], errors='coerce')
#join 2 columns together
col_strength = df['ACTIVE_NUMERATOR_STRENGTH']
col_drugname = df['PROPRIETARYNAME']
result = pd.concat([col_drugname,col_strength],axis=1)
#drop all rows with NaN values
result = result.dropna()
#calculate
result.insert(1,'3x_strength',(result['ACTIVE_NUMERATOR_STRENGTH']*3).astype(float))
result
```

Out[74]:

	PROPRIETARYNAME	3x_strength	ACTIVE_NUMERATOR_STRENGTH
0	Sterile Diluent	3.00	1.00
1	Amyvid	153.00	51.00
2	Quinidine Gluconate	240.00	80.00
3	Trulicity	2.25	0.75
4	Trulicity	4.50	1.50
5	EMGALITY	360.00	120.00
6	TALTZ	240.00	80.00
7	EMGALITY	360.00	120.00
8	Prozac	270.00	90.00

- To ensure that all of the fields were numeric, we dropped the rows where there were "NaN" values.

Feature Normalization



Feature Normalization

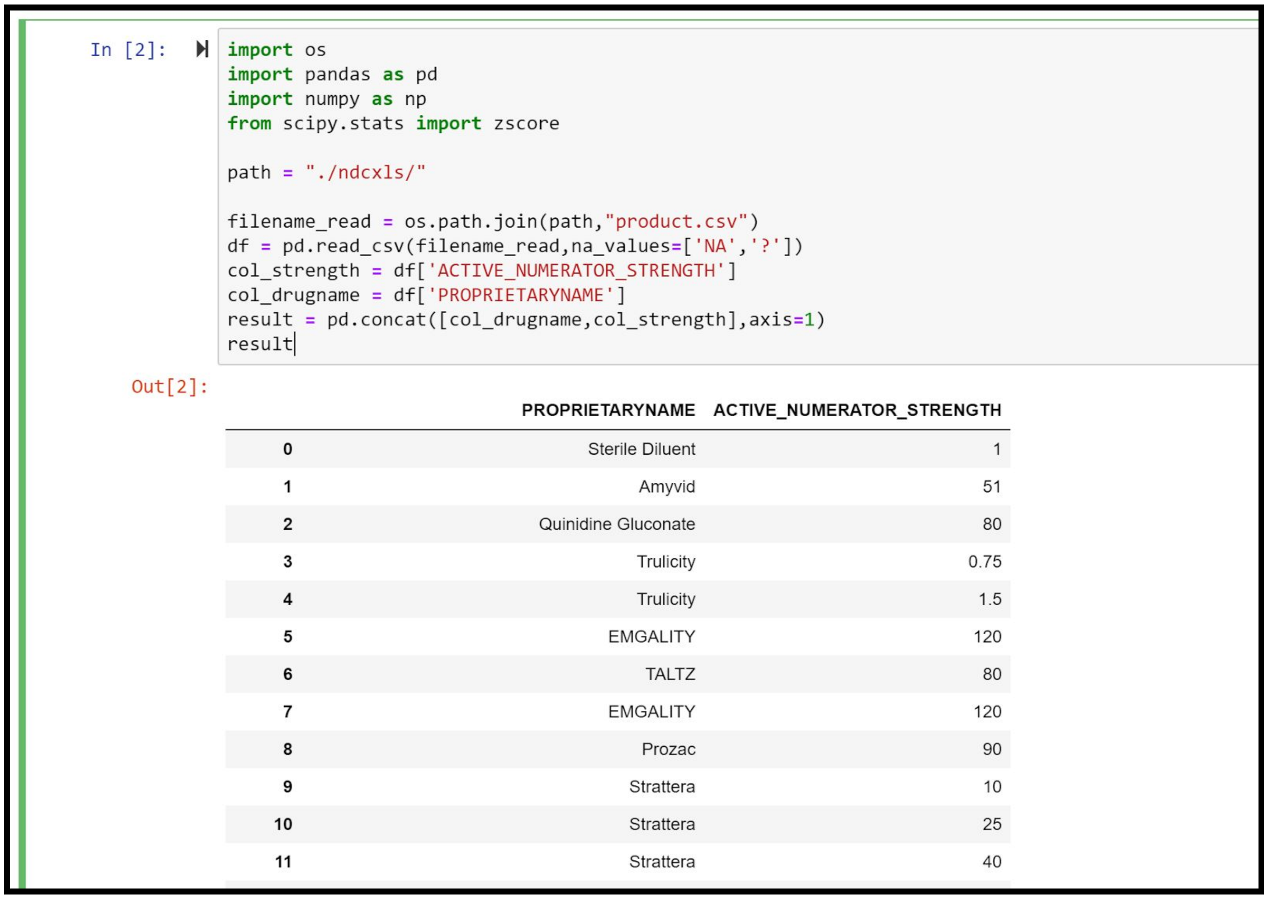
In [79]:
```
import os
import pandas as pd
import numpy as np
from scipy.stats import zscore

path = "./ndcxls/"
filename_read = os.path.join(path,"product.csv")
df = pd.read_csv(filename_read,na_values=['NA','?'])
#change none numeric values to NaN
df['ACTIVE_NUMERATOR_STRENGTH'] = pd.to_numeric(df['ACTIVE_NUMERATOR_STRENGTH'], errors='coerce')
#join 2 columns together
col_strength = df['ACTIVE_NUMERATOR_STRENGTH']
col_drugname = df['PROPRIETARYNAME']
result = pd.concat([col_drugname,col_strength],axis=1)
#drop all rows with NaN values
result = result.dropna()
#calculate
result.insert(1,'3x_strength',(result['ACTIVE_NUMERATOR_STRENGTH']*3).astype(float))
#zscore
result['3x_strength'] = zscore(result['3x_strength'])
result
```

Out[79]:

	PROPRIETARYNAME	3x_strength	ACTIVE_NUMERATOR_STRENGTH
0	Sterile Diluent	-0.004984	1.00
1	Amyvid	-0.004982	51.00
2	Quinidine Gluconate	-0.004981	80.00
3	Trulicity	-0.004984	0.75
4	Trulicity	-0.004984	1.50
5	EMGALITY	-0.004980	120.00
6	TALTZ	-0.004981	80.00

Concatenation - Min (Drugs)

- The method .concat() adds multiple columns of data together. Axis 1 adds columns horizontally and axis 0 adds everything in one column vertically.



In [2]:
```
import os
import pandas as pd
import numpy as np
from scipy.stats import zscore

path = "./ndcxls/"

filename_read = os.path.join(path,"product.csv")
df = pd.read_csv(filename_read,na_values=['NA','?'])
col_strength = df['ACTIVE_NUMERATOR_STRENGTH']
col_drugname = df['PROPRIETARYNAME']
result = pd.concat([col_drugname,col_strength],axis=1)
result
```

Out[2]:

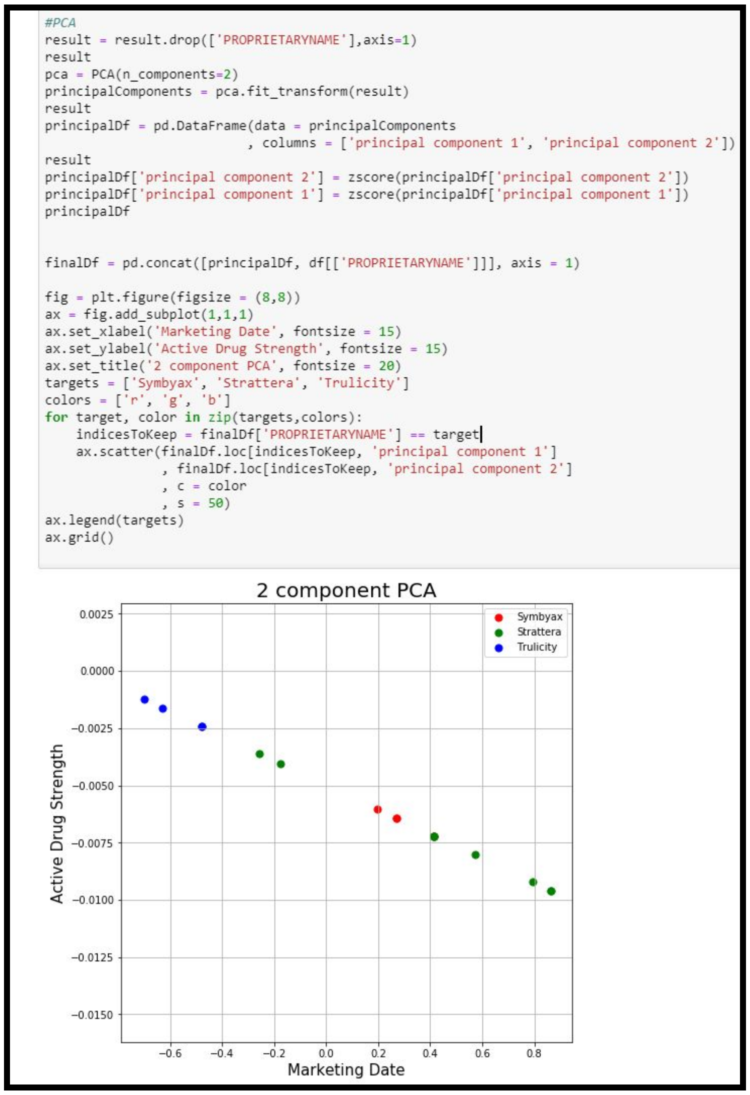
	PROPRIETARYNAME	ACTIVE_NUMERATOR_STRENGTH
0	Sterile Diluent	1
1	Amyvid	51
2	Quinidine Gluconate	80
3	Trulicity	0.75
4	Trulicity	1.5
5	EMGALITY	120
6	TALTZ	80
7	EMGALITY	120
8	Prozac	90
9	Strattera	10
10	Strattera	25
11	Strattera	40

Aggregation

-

PCA:

One thing that was interesting to analyze is the PCA performed on drugs Symbyax, Strattera, and Trulicity. PCA performed on active drug strength relative starting marketing date. From the PCA, the data suggests that as the Standardized (Z-Score) Marketing Date increases, the Strattera was manufactured at a much lower strength. When compared to Symbyax, Trulicity continued to stay unchanged. It was interesting to see a negative correlation in a large set of data.



```
#PCA
result = result.drop(['PROPRIETARYNAME'],axis=1)
result
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(result)
result
principalDf = pd.DataFrame(data = principalComponents
                          , columns = ['principal component 1', 'principal component 2'])
result
principalDf['principal component 2'] = zscore(principalDf['principal component 2'])
principalDf['principal component 1'] = zscore(principalDf['principal component 1'])
principalDf


finalDf = pd.concat([principalDf, df[['PROPRIETARYNAME']]], axis = 1)

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Marketing Date', fontsize = 15)
ax.set_ylabel('Active Drug Strength', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = ['Symbyax', 'Strattera', 'Trulicity']
colors = ['r', 'g', 'b']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['PROPRIETARYNAME'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```

Data split

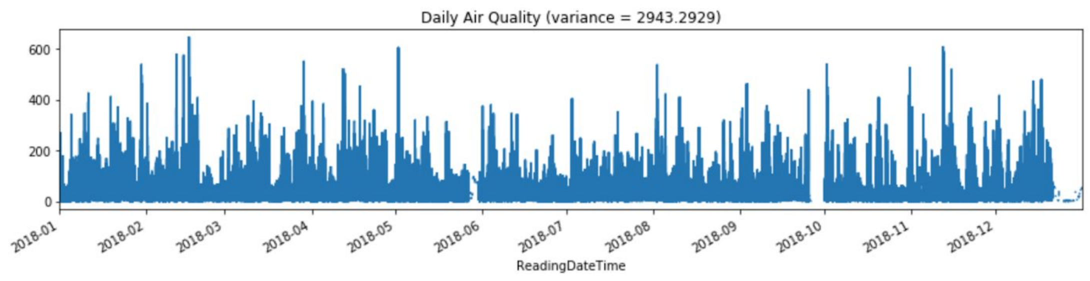
```
#---Training/Test split---
filename_read = os.path.join(path, 'product2.csv')
df = pd.read_csv(filename_read, na_values=['NA', '?'])

le = preprocessing.LabelEncoder()
df['encoded_species'] = le.fit_transform(df['Species'])

x_train, x_test, y_train, y_test = train_test_split(df[['Species', 'Value']],
                                                    df['encoded_species'],
                                                    test_size=0.25, random_state=60)
print('Training set mean = %d' % (x_train.Value.mean()))
print('Training set standard deviation = %d' % (x_train.Value.std()))
print('Test set mean = %d' % (x_test.Value.mean()))
print('Test set standard deviation = %d' % (x_test.Value.std()))
```

```
Training set mean = 47
Training set standard deviation = 54
Test set mean = 48
Test set standard deviation = 54
```

Aggregation

```
#---AGGREGATION---
daily = df_air
daily.index = pd.to_datetime(daily['ReadingDateTime'])
daily = daily['Value']
fig = plt.figure(figsize = (8,8))
ax1 = fig.add_subplot(1,1,1)
ax1 = daily.plot(kind = 'line',figsize=(15,3))
ax1.set_title('Daily Air Quality (variance = %.4f)' % (daily.var()))
```

```
Out[40]: Text(0.5, 1.0, 'Daily Air Quality (variance = 2943.2929)')
```



1) Interpreting human insights on the data and its possible effect on predictions

Through the Pre-Processing of our data, the number of rows and columns were significantly reduced. Since we “scrubbed” the data so that it became cleaner, our confidence in the legitimacy of the data increased. We got rid of NaN data, blank data, unnecessary fields, duplicate data, outliers, and sorted the data in such a way where the format was easily readable. After condensing the data into what we deemed necessary, the amount of information to look at is significantly less for us to read, and less for our program to parse through. This will result in quicker, and more accurate predictions, because of the removal of junk data and dimensions.

2) Split data into two sets: training data (first 80% of the product.csv file), and test data (last 20% of the product.csv file), and calculate the mean and standard deviation

Training Data:

Mean: 47

Standard Deviation: 54

Test Data:

Mean: 48

Standard Deviation: 54

3) Document ratios for splitting and the results

We decided to primarily focus on the “Drugs” link, specifically “Product.csv”. The reason for this is because we have done the most thorough Pre-Processing on this data set. We decided to split the data as follows:

Training Data: First 80% of the data after Pre-Processing

Test Data: Last 20% of the data after Pre-Processing

The reason we decided on 80/20 is because since we did “Sorting Dataframe”, the rows are all randomized. That’s why it sufficient to pick the first 80%, and the last 20% of the data.

Results:

4) Compare the two sets: the training data and the test data and analyze it, developing an intuition and meaning of your results.

What we can draw from this data is that splitting data into two consequent sets is a vital step in creating a machine learning model. The training data receives the majority share of the partitions since this is the data that the model is built on, whereas the remaining test set will be used to validate the training data against. In our case, it is apparent that the two sets of randomized data provide consistent results in terms of the mean and standard deviation of the "value" attribute.