Module 4: Data Preprocessing

The following tutorial contains Python examples for data preprocessing. You should refer to the "Data" chapter of the "Introduction to Data Mining" book (slides are available at https://www-users.cs.umn.edu/~kumar001/dmbook/index.php) to understand some of the concepts introduced in this tutorial. Data preprocessing consists of a broad set of techniques for cleaning, selecting, and transforming data to improve data mining analysis. Read the step-by-step instructions below carefully. To execute the code, click on the corresponding cell and press the SHIFT-ENTER keys simultaneously.

Data Quality Issues

Poor data quality can have an adverse effect on data mining. Among the common data quality issues include noise, outliers, missing values, and duplicate data. This section presents examples of Python code to alleviate some of these data quality problems. We begin with an example dataset from the UCI machine learning repository containing information about breast cancer patients. We will first download the dataset using Pandas read_csv() function and display its first 5 data points.

Code:

In [1]:
```
import pandas as pd
data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/breast-c
data.columns = ['Sample code', 'Clump Thickness', 'Uniformity of Cell Size', 'Uniformi
                'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Bl
                '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	Cla
0	5	1	1	1	2	1	3	1	1	
1	5	4	4	5	7	10	3	2	1	
2	3	1	1	1	2	2	3	1	1	
3	6	8	8	1	3	4	3	7	1	
4	4	1	1	3	2	1	3	1	1	

Missing Values

It is not unusual for an object to be missing one or more attribute values. In some cases, the information was not collected; while in other cases, some attributes are inapplicable to the data instances. This section presents examples on the different approaches for handling missing values.

According to the description of the data (https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original), the missing values are encoded as '?' in the original data. Our first task is to convert the missing values to NaNs. We can then count the number of missing values in each column of the data.

Code:

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
```

Observe that only the 'Bare Nuclei' column contains missing values. In the following example, the missing values in the 'Bare Nuclei' column are replaced by the median value of that column. The values before and after replacement are shown for a subset of the data points.

Code:

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    1.0
24      1
Name: Bare Nuclei, dtype: object
```

Instead of replacing the missing values, another common approach is to discard the data points that contain missing values. This can be easily accomplished by applying the dropna() function to the data frame.

Code:

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
```

Outliers

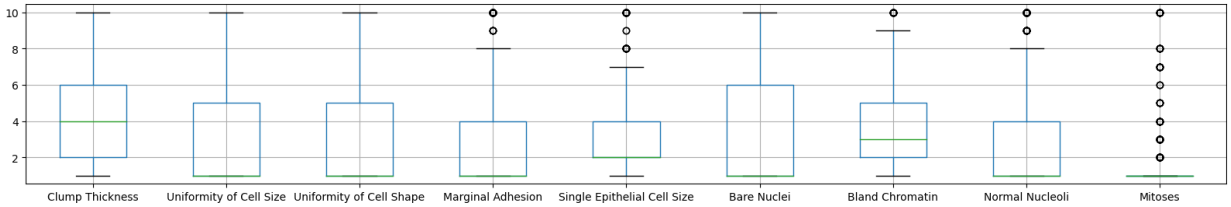
Outliers are data instances with characteristics that are considerably different from the rest of the dataset. In the example code below, we will draw a boxplot to identify the columns in the table that contain outliers. Note that the values in all columns (except for 'Bare Nuclei') are originally stored as 'int64' whereas the values in the 'Bare Nuclei' column are stored as string objects (since the column initially contains strings such as '?' for representing missing values). Thus, we must convert the column into numeric values first before creating the boxplot. Otherwise, the column will not be displayed when drawing the boxplot.

Code:

In [5]:
```
%matplotlib inline

data2 = data.drop(['Class'],axis=1)
data2['Bare Nuclei'] = pd.to_numeric(data2['Bare Nuclei'])
data2.boxplot(figsize=(20,3))
```

Out[5]:
```
<Axes: >
```



The boxplots suggest that only 5 of the columns (Marginal Adhesion, Single Epithetial Cell Size, Bland Cromatin, Normal Nucleoli, and Mitoses) contain abnormally high values. To discard the outliers, we can compute the Z-score for each attribute and remove those instances containing attributes with abnormally high or low Z-score (e.g., if Z > 3 or Z <= -3).

Code:

The following code shows the results of standardizing the columns of the data. Note that missing values (NaN) are not affected by the standardization process.

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	Mitos
20	0.917080	-0.044070	-0.406284	2.519152	0.805662	1.771569	0.640688	0.371049	1.4055
21	1.982519	0.611354	0.603167	0.067638	1.257272	0.948266	1.460910	2.335921	-0.3436
22	-0.503505	-0.699494	-0.742767	-0.632794	-0.549168	-0.698341	-0.589645	-0.611387	-0.3436
23	1.272227	0.283642	0.603167	-0.632794	-0.549168	NaN	1.460910	0.043570	-0.3436
24	-1.213798	-0.699494	-0.742767	-0.632794	-0.549168	-0.698341	-0.179534	-0.611387	-0.3436

Code:

The following code shows the results of discarding columns with Z > 3 or Z <= -3.

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
```

Duplicate Data

Some datasets, especially those obtained by merging multiple data sources, may contain duplicates or near duplicate instances. The term deduplication is often used to refer to the process of dealing with duplicate data issues.

Code:

In the following example, we first check for duplicate instances in the breast cancer dataset.

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	Cl
11	2	1	1	1	2	1	2	1	1	
28	2	1	1	1	2	1	2	1	1	

The duplicated() function will return a Boolean array that indicates whether each row is a duplicate of a previous row in the table. The results suggest there are 236 duplicate rows in the breast cancer dataset. For example, the instance with row index 11 has identical attribute values as the instance with row index 28. Although such duplicate rows may correspond to samples for different individuals, in this hypothetical example, we assume that the duplicates are samples taken from the same individual and illustrate below how to remove the duplicated rows.

Code:

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
```

Shuffling Dataframes

It is possible to shuffle.

In [10]:
```
import os
import numpy as np
import pandas as pd

path = "./data/"

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[10]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	15.5	8	351.0	142.0	4054	14.3	79	1	ford country squire (sw)
1	29.0	4	98.0	83.0	2219	16.5	74	2	audi fox
2	30.7	6	145.0	76.0	3160	19.6	81	2	volvo diesel
3	26.0	4	97.0	46.0	1950	21.0	73	2	volkswagen super beetle
4	11.0	8	429.0	208.0	4633	11.0	72	1	mercury marquis
...	...	...	...	...	...	...	...	...	...
393	17.5	8	318.0	140.0	4080	13.7	78	1	dodge magnum xe
394	14.5	8	351.0	152.0	4215	12.8	76	1	ford gran torino
395	14.0	8	302.0	137.0	4042	14.5	73	1	ford gran torino
396	21.0	6	155.0	107.0	2472	14.0	73	1	mercury capri v6
397	26.6	8	350.0	105.0	3725	19.0	81	1	oldsmobile cutlass ls

398 rows × 9 columns

Sorting Dataframes

It is possible to sort.

In [11]:
```
df = df.sort_values(by='name',ascending=True)
df
```

Out[11]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
228	13.0	8	360.0	175.0	3821	11.0	73	1	amc ambassador brougham
280	15.0	8	390.0	190.0	3850	8.5	70	1	amc ambassador dpl
159	17.0	8	304.0	150.0	3672	11.5	72	1	amc ambassador sst
389	24.3	4	151.0	90.0	3003	20.1	80	1	amc concord
270	19.4	6	232.0	90.0	3210	17.2	78	1	amc concord
...	...	...	...	...	...	...	...	...	...
35	44.0	4	97.0	52.0	2130	24.6	82	2	vw pickup
319	41.5	4	98.0	76.0	2144	14.7	80	2	vw rabbit
282	29.0	4	90.0	70.0	1937	14.2	76	2	vw rabbit
51	44.3	4	90.0	48.0	2085	21.7	80	2	vw rabbit c (diesel)
178	31.9	4	89.0	71.0	1925	14.0	79	2	vw rabbit custom

398 rows × 9 columns

In [12]:
```
print("The first car is: {}".format(df['name'].iloc[0]))
```

```
The first car is: amc ambassador brougham
```

In [13]:
```
print("The first car is: {}".format(df['name'].loc[0]))

#loc gets rows (or columns) with particular labels from the index.
#iloc gets rows (or columns) at particular positions in the index (so it only takes ir
```

```
The first car is: ford country squire (sw)
```

Saving a Dataframe

The following code performs a shuffle and then saves a new copy.

In [14]:
```
import os
import pandas as pd
import numpy as np

path = "./data/"

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 numbe
print("Done")
```

```
Done
```

Dropping Fields

Some fields are of no value to the neural network and can be dropped. The following code removes the name column from the MPG dataset.

In [15]:
```
import os
import pandas as pd
import numpy as np

path = "./data/"

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[15]:

	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

Calculated Fields

It is possible to add new fields to the dataframe that are calculated from the other fields. We can create a new column that gives the weight in kilograms. The equation to calculate a metric weight, given a weight in pounds is:

$ m_{(kg)} = m_{(lb)} \times 0.45359237 $

This can be used with the following Python code:

In [16]:
```
import os
import pandas as pd
import numpy as np
```

```
path = "./data/"

filename_read = os.path.join(path,"auto-mpg.csv")
df = pd.read_csv(filename_read,na_values=['NA','?'])
df.insert(1,'weight_kg',(df['weight']*0.45359237).astype(int))
df
```

Out[16]:

	mpg	weight_kg	cylinders	displacement	horsepower	weight	acceleration	year	origin	na
0	18.0	1589	8	307.0	130.0	3504	12.0	70	1	chevr chev ma
1	15.0	1675	8	350.0	165.0	3693	11.5	70	1	bu sky
2	18.0	1558	8	318.0	150.0	3436	11.0	70	1	plymc sate
3	16.0	1557	8	304.0	150.0	3433	12.0	70	1	a rebe
4	17.0	1564	8	302.0	140.0	3449	10.5	70	1	f to
...	...	...	...	...	...	...	...	...	...	
393	27.0	1265	4	140.0	86.0	2790	15.6	82	1	f must
394	44.0	966	4	97.0	52.0	2130	24.6	82	2	pic
395	32.0	1040	4	135.0	84.0	2295	11.6	82	1	do ramp
396	28.0	1190	4	120.0	79.0	2625	18.6	82	1	f rar
397	31.0	1233	4	119.0	82.0	2720	19.4	82	1	chev

398 rows × 10 columns

Feature Normalization

A normalization allows numbers to be put in a standard form so that two values can easily be compared. One very common machine learning normalization is the Z-Score:

$z = {x- \mu \over \sigma} $

To calculate the Z-Score you need to also calculate the mean(μ) and the standard deviation (σ). The mean is calculated as follows:

$\mu = \bar{x} = \frac{x_1+x_2+\cdots +x_n}{n}$

The standard deviation is calculated as follows:

$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}, {\rm \ \ where\ \ } \mu = \frac{1}{N} \sum_{i=1}^N x_i$

The following Python code ***replaces the mpg with a z-score***. Cars with average MPG will be near zero, above zero is above average, and below zero is below average. Z-Scores above/below -3/3 are very rare, these are outliers.

In [17]:
```
import os
import pandas as pd
import numpy as np
from scipy.stats import zscore

path = "./data/"

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[17]:

	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
...	...	...	...	...	...	...	...	...	...
393	0.446497	4	140.0	86.0	2790	15.6	82	1	ford mustang gl
394	2.624265	4	97.0	52.0	2130	24.6	82	2	vw pickup
395	1.087017	4	135.0	84.0	2295	11.6	82	1	dodge rampage
396	0.574601	4	120.0	79.0	2625	18.6	82	1	ford ranger
397	0.958913	4	119.0	82.0	2720	19.4	82	1	chevy s-10

398 rows × 9 columns

Missing Values

You can also simply drop any rows with any NA values. Another common practice is to replace missing values with the median value for that column. The following code replaces any NA values in horsepower with the median:

In [18]:
```
import os
import pandas as pd
import numpy as np
from scipy.stats import zscore

path = "./data/"

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

# df = df.dropna() # you can also simply drop NA values
```

Concatenating Rows and Columns

Rows and columns can be concatenated together to form new data frames.

In [19]:
```
# Create a new dataframe from name and horsepower

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

path = "./data/"

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[19]:

	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
...	...	...
393	ford mustang gl	86.0
394	vw pickup	52.0
395	dodge rampage	84.0
396	ford ranger	79.0
397	chevy s-10	82.0

398 rows × 2 columns

In [20]:
```
# Create a new dataframe from name and horsepower, but this time by row

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

path = "./data/"

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])
result
```

Out[20]:
```
0      chevrolet chevelle malibu
1              buick skylark 320
2             plymouth satellite
3                  amc rebel sst
4                    ford torino
                  ...
393                          86.0
394                          52.0
395                          84.0
396                          79.0
397                          82.0
Length: 796, dtype: object
```

Helpful Functions for Tensorflow (Little Gems)

The following functions will be used with TensorFlow to help preprocess the data.

They allow you to build the feature vector in the format that TensorFlow expects from raw data.

(1) Encoding data:

- **encode_text_dummy** - Encode text fields as numeric, such as the iris species as a single field for each class. Three classes would become "0,0,1" "0,1,0" and "1,0,0". Encode non-target features this way. used when the data is part of input ***(one hot encoding)***
- **encode_text_index** - Encode text fields to numeric, such as the iris species as a single numeric field as "0" "1" and "2". Encode the target field for a classification this way. used when data is part of output ***(label encoding)***

(2) Normalizing data:

- **encode_numeric_zscore** - Encode numeric values as a z-score. Neural networks deal well with "normalized" fields only.

- **encode_numeric_range** - Encode a column to a range between the given normalized_low and normalized_high.

(3) Dealing with missing data:

- **missing_median** - Fill all missing values with the median value.

(4) Removing outliers:

- **remove_outliers** - Remove outliers in a certain column with a value beyond X times SD

(5) Creating the feature vector and target vector that ***Tensorflow needs***:

- **to_xy** - ***Once all fields are encoded to numeric, this function can provide the x and y matrixes that TensorFlow needs to fit the neural network with data.***

(6) Other utility functions:

- **hms_string** - Print out an elapsed time string.
- **chart_regression** - Display a chart to show how well a regression performs.

In [21]:
```
import collections
from sklearn import preprocessing
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import shutil
import os


# Encode text values to dummy variables(i.e. [1,0,0],[0,1,0],[0,0,1] for red,green,blu
def encode_text_dummy(df, name):
    dummies = pd.get_dummies(df[name])
    for x in dummies.columns:
        dummy_name = "{}-{}".format(name, x)
        df[dummy_name] = dummies[x]
    df.drop(name, axis=1, inplace=True)


# Encode text values to indexes(i.e. [1],[2],[3] for red,green,blue).
def encode_text_index(df, name):
    le = preprocessing.LabelEncoder()
    df[name] = le.fit_transform(df[name])
    return le.classes_


# Encode a numeric column as zscores
def encode_numeric_zscore(df, name, mean=None, sd=None):
    if mean is None:
        mean = df[name].mean()

    if sd is None:
        sd = df[name].std()

    df[name] = (df[name] - mean) / sd
```

```
# Convert all missing values in the specified column to the median
def missing_median(df, name):
    med = df[name].median()
    df[name] = df[name].fillna(med)


# Convert all missing values in the specified column to the default
def missing_default(df, name, default_value):
    df[name] = df[name].fillna(default_value)


# Convert a Pandas dataframe to the x,y inputs that TensorFlow needs
def to_xy(df, target):
    result = []
    for x in df.columns:
        if x != target:
            result.append(x)
    # find out the type of the target column.
    target_type = df[target].dtypes
    target_type = target_type[0] if isinstance(target_type, collections.abc.Sequence)
    # Encode to int for classification, float otherwise. TensorFlow likes 32 bits.
    if target_type in (np.int64, np.int32):
        # Classification
        dummies = pd.get_dummies(df[target])
        return df[result].values.astype(np.float32), dummies.values.astype(np.float32)
    else:
        # Regression
        return df[result].values.astype(np.float32), df[target].values.astype(np.float

# Nicely formatted time string
def hms_string(sec_elapsed):
    h = int(sec_elapsed / (60 * 60))
    m = int((sec_elapsed % (60 * 60)) / 60)
    s = sec_elapsed % 60
    return "{}:{:>02}:{:>05.2f}".format(h, m, s)


# Regression chart.
def chart_regression(pred,y,sort=True):
    t = pd.DataFrame({'pred' : pred, 'y' : y.flatten()})
    if sort:
        t.sort_values(by=['y'],inplace=True)
    a = plt.plot(t['y'].tolist(),label='expected')
    b = plt.plot(t['pred'].tolist(),label='prediction')
    plt.ylabel('output')
    plt.legend()
    plt.show()

# Remove all rows where the specified column is +/- sd standard deviations
def remove_outliers(df, name, sd):
    drop_rows = df.index[(np.abs(df[name] - df[name].mean()) >= (sd * df[name].std()))]
    df.drop(drop_rows, axis=0, inplace=True)


# Encode a column to a range between normalized_low and normalized_high.
def encode_numeric_range(df, name, normalized_low=-1, normalized_high=1,
                         data_low=None, data_high=None):
    if data_low is None:
        data_low = min(df[name])
```

```
        data_high = max(df[name])

    df[name] = ((df[name] - data_low) / (data_high - data_low)) * (normalized_high - n
```

Examples of label encoding, one hot encoding, and creating X/Y for TensorFlow

In [22]:
```
df=pd.read_csv("data/iris.csv",na_values=['NA','?'])
df
```

Out[22]:

	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
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [23]:
```
encode_text_index(df,"species")   # label encoding
df
```

Out[23]:

	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
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

In [24]:
```
df=pd.read_csv("data/iris.csv",na_values=['NA','?'])

encode_text_dummy(df,"species")   # One hot encoding
df
```

Out[24]:

	sepal_l	sepal_w	petal_l	petal_w	species-Iris-setosa	species-Iris-versicolor	species-Iris-virginica
0	5.1	3.5	1.4	0.2	1	0	0
1	4.9	3.0	1.4	0.2	1	0	0
2	4.7	3.2	1.3	0.2	1	0	0
3	4.6	3.1	1.5	0.2	1	0	0
4	5.0	3.6	1.4	0.2	1	0	0
...	...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	0	0	1
146	6.3	2.5	5.0	1.9	0	0	1
147	6.5	3.0	5.2	2.0	0	0	1
148	6.2	3.4	5.4	2.3	0	0	1
149	5.9	3.0	5.1	1.8	0	0	1

150 rows × 7 columns

Make sure you encode the lables first before you call to_xy()

In [25]:
```
df=pd.read_csv("data/iris.csv",na_values=['NA','?'])
```

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

df
```

Out[25]:

	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
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

In [26]:
```
x,y = to_xy(df,"species")
```

In [27]:
```
x
```

Out[27]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
       [4.9, 2.4, 3.3, 1. ],
       [6.6, 2.9, 4.6, 1.3],
       [5.2, 2.7, 3.9, 1.4],
```

```
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
[6.8, 3. , 5.5, 2.1],
[5.7, 2.5, 5. , 2. ],
[5.8, 2.8, 5.1, 2.4],
[6.4, 3.2, 5.3, 2.3],
[6.5, 3. , 5.5, 1.8],
[7.7, 3.8, 6.7, 2.2],
[7.7, 2.6, 6.9, 2.3],
[6. , 2.2, 5. , 1.5],
```

```
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]], dtype=float32)
```

In [28]:
```
y
```

Out[28]:
```
array([[1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
```

```
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
```

```
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.]], dtype=float32)
```

Example of Deal with Missing Values and Outliers

In [29]:

```
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
```

Training and Validation

The machine learning model will learn from the training data, but ultimately be evaluated based on the validation data.

- **Training Data** - **In Sample Data** - The data that the machine learning model was fit to/created from.
- **Validation Data** - **Out of Sample Data** - The data that the machine learning model is evaluated upon after it is fit to the training data.

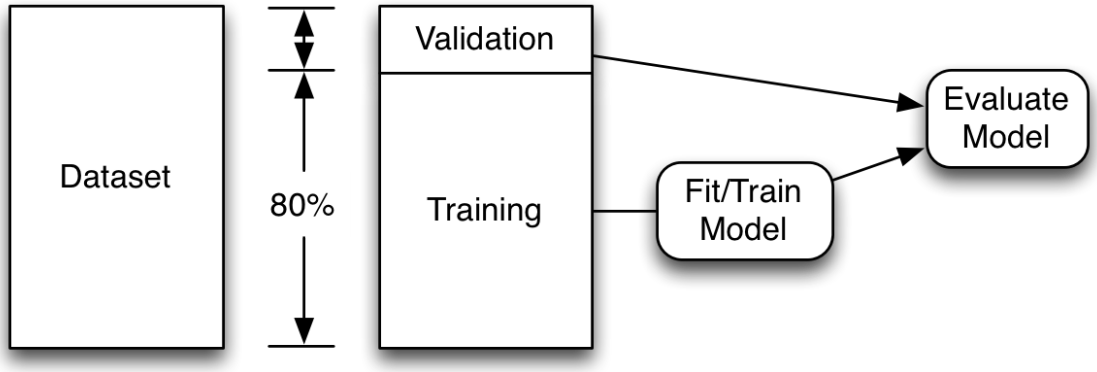
There are two predominant means of dealing with training and validation data:

- **Training/Test Split** - The data are split according to some ratio between a training and validation (hold-out) set. Common ratios are 80% training and 20% validation.
- **K-Fold Cross Validation** - The data are split into a number of folds and models. Because a number of models equal to the folds is created out-of-sample predictions can be generated for the entire dataset.

Training/Test Split

The code below performs a split of the MPG data into a training and validation set. The training set uses 80% of the data and the test(validation) set uses 20%.

The following image shows how a model is trained on 80% of the data and then validated against the remaining 20%.



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[['sepal_l', 'sepal_w', 'petal_l
```

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,)
```

In [ ]:
```

```

Aggregation

Data aggregation is a preprocessing task where the values of two or more objects are combined into a single object. The motivation for aggregation includes (1) reducing the size of data to be processed, (2) changing the granularity of analysis (from fine-scale to coarser-scale), and (3) improving the stability of the data.

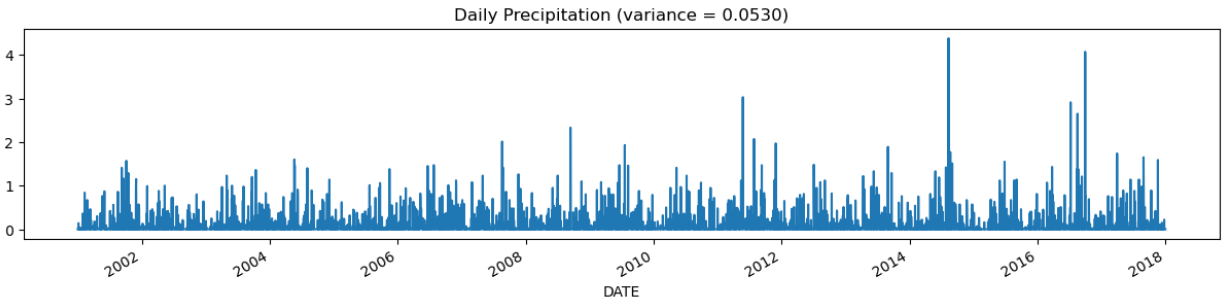
In the example below, we will use the daily precipitation time series data for a weather station located at Detroit Metro Airport. The raw data was obtained from the Climate Data Online website (https://www.ncdc.noaa.gov/cdo-web/). The daily precipitation time series will be compared against its monthly values.

Code:

The code below will load the precipitation time series data and draw a line plot of its daily time series.

In [37]:
```
daily = pd.read_csv('DTW_prec.csv', header='infer')
daily.index = pd.to_datetime(daily['DATE'])
daily = daily['PRCP']
ax = daily.plot(kind='line',figsize=(15,3))
ax.set_title('Daily Precipitation (variance = %.4f)' % (daily.var()))
```

Out[37]:
```
Text(0.5, 1.0, 'Daily Precipitation (variance = 0.0530)')
```



In [38]:
```
daily
```

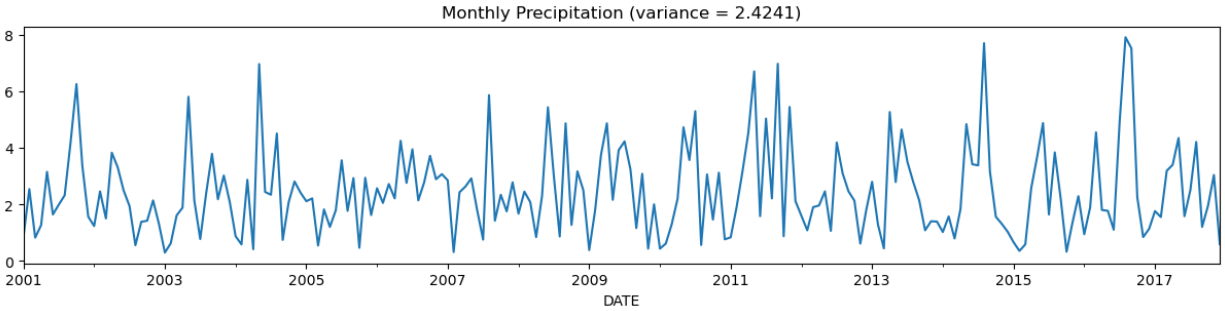
Out[38]:
```
DATE
2001-01-01    0.00
2001-01-02    0.00
2001-01-03    0.00
2001-01-04    0.04
2001-01-05    0.14
              ...
2017-12-27    0.00
2017-12-28    0.00
2017-12-29    0.00
2017-12-30    0.00
2017-12-31    0.00
Name: PRCP, Length: 6191, dtype: float64
```

Observe that the daily time series appear to be quite chaotic and varies significantly from one time step to another. The time series can be grouped and aggregated by month to obtain the total monthly precipitation values. The resulting time series appears to vary more smoothly compared to the daily time series.

Code:

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)')
```

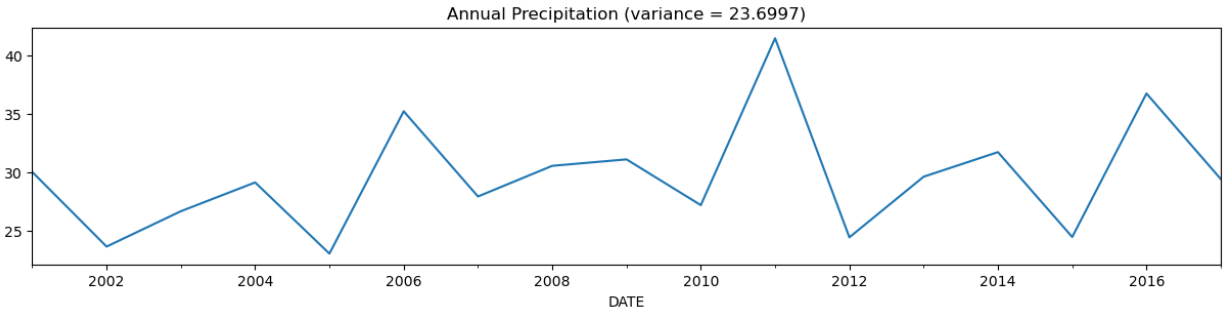


In the example below, the daily precipitation time series are grouped and aggregated by year to obtain the annual precipitation values.

Code:

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)')
```



Sampling

Sampling is an approach commonly used to facilitate (1) data reduction for exploratory data analysis and scaling up algorithms to big data applications and (2) quantifying uncertainties due to varying data distributions. There are various methods available for data sampling, such as sampling without replacement, where each selected instance is removed from the dataset, and sampling with replacement, where each selected instance is not removed, thus allowing it to be selected more than once in the sample.

In the example below, we will apply sampling with replacement and without replacement to the breast cancer dataset obtained from the UCI machine learning repository.

Code:

We initially display the first five records of the table.

In [41]:
```
data.head()
```

Out[41]:

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

In the following code, a sample of size 3 is randomly selected (without replacement) from the original data.

Code:

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	C
236	10	8	8	2	8	10	4	8	10	
129	1	1	1	1	10	1	1	1	1	
598	3	1	1	1	2	1	2	1	1	

In the next example, we randomly select 1% of the data (without replacement) and display the selected samples. The random_state argument of the function specifies the seed value of the random number generator.

Code:

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
584	5	1	1	6	3	1	1	1	1
417	1	1	1	1	2	1	2	1	1
606	4	1	1	2	2	1	1	1	1
349	4	2	3	5	3	8	7	6	1
134	3	1	1	1	3	1	2	1	1
502	4	1	1	2	2	1	2	1	1
117	4	5	5	10	4	10	7	5	8

Finally, we perform a sampling with replacement to create a sample whose size is equal to 1% of the entire data. You should be able to observe duplicate instances in the sample by increasing the sample size.

Code:

In [44]:
```
sample = data.sample(frac=0.01, replace=True, random_state=1)
sample
```

Out[44]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses
37	6	2	1	1	1	1	7	1	1
235	3	1	4	1	2	NaN	3	1	1
72	1	3	3	2	2	1	7	2	1
645	3	1	1	1	2	1	2	1	1
144	2	1	1	1	2	1	2	1	1
129	1	1	1	1	10	1	1	1	1
583	3	1	1	1	2	1	1	1	1

Discretization

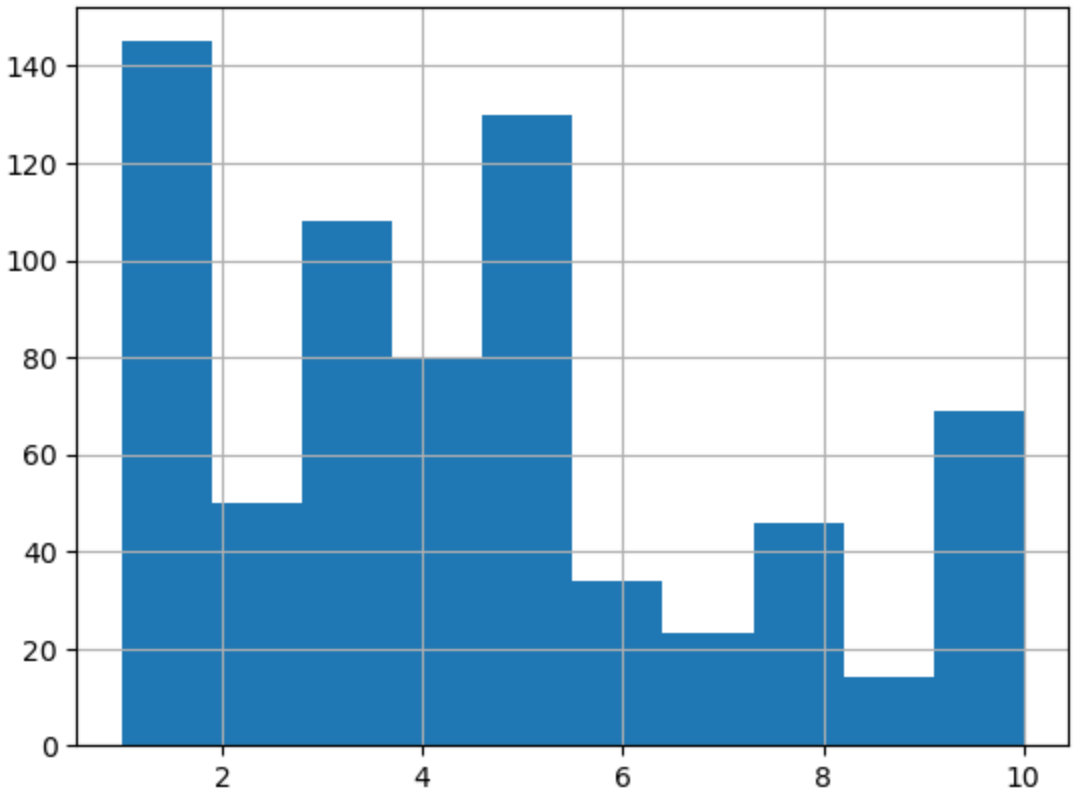
Discretization is a data preprocessing step that is often used to transform a continuous-valued attribute to a categorical attribute. The example below illustrates two simple but widely-used unsupervised discretization methods (equal width and equal depth) applied to the 'Clump Thickness' attribute of the breast cancer dataset.

First, we plot a histogram that shows the distribution of the attribute values. The value_counts() function can also be applied to count the frequency of each attribute value.

Code:

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

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



For the equal width method, we can apply the cut() function to discretize the attribute into 4 bins of similar interval widths. The value_counts() function can be used to determine the number of instances in each bin.

Code:

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
```

For the equal frequency method, the qcut() function can be used to partition the values into 4 bins such that each bin has nearly the same number of instances.

Code:

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
```

Principal Component Analysis

Principal component analysis (PCA) is a classical 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 attributes (also known as components) created by PCA have the following properties: (1) they are linear combinations of the original attributes, (2) they are orthogonal (perpendicular) to each other, and (3) they capture the maximum amount of variation in the data.

The example below illustrates the application of PCA to an image dataset. There are 16 RGB files, each of which has a size of 111 x 111 pixels. The example code below will read each image file and convert the RGB image into a 111 x 111 x 3 = 36963 feature values. This will create a data matrix of size 16 x 36963.

Code:

In [48]:
```
%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.sha
```

Using PCA, the data matrix is projected to its first two principal components. The projected values of the original image data are stored in a pandas DataFrame object named projected.

Code:

In [49]:
```
import pandas as pd
from sklearn.decomposition import PCA

numComponents = 2
pca = PCA(n_components=numComponents)
pca.fit(imgData)

projected = pca.transform(imgData)
projected = pd.DataFrame(projected,columns=['pc1','pc2'],index=range(1,numImages+1))
projected['food'] = ['burger', 'burger','burger','burger','drink','drink','drink','dri
                     'pasta', 'pasta', 'pasta', 'pasta', 'chicken', 'chicken', 'chick
projected
```

Out[49]:

	pc1	pc2	food
1	-1576.693529	6639.310197	burger
2	-493.820100	6398.288020	burger
3	990.082034	7236.575491	burger
4	2189.873761	9051.415487	burger
5	-7843.047855	-1061.938362	drink
6	-8498.425849	-5438.132978	drink
7	-11181.844337	-5320.075055	drink
8	-6851.923587	1124.952225	drink
9	7635.132567	-5043.912333	pasta
10	-708.051739	-529.182830	pasta
11	7236.240829	-5300.920582	pasta
12	4417.326344	-4658.630348	pasta
13	11864.496244	1472.509923	chicken
14	76.466276	1365.909278	chicken
15	-7505.629276	-1163.599009	chicken
16	10249.818217	-4772.569124	chicken

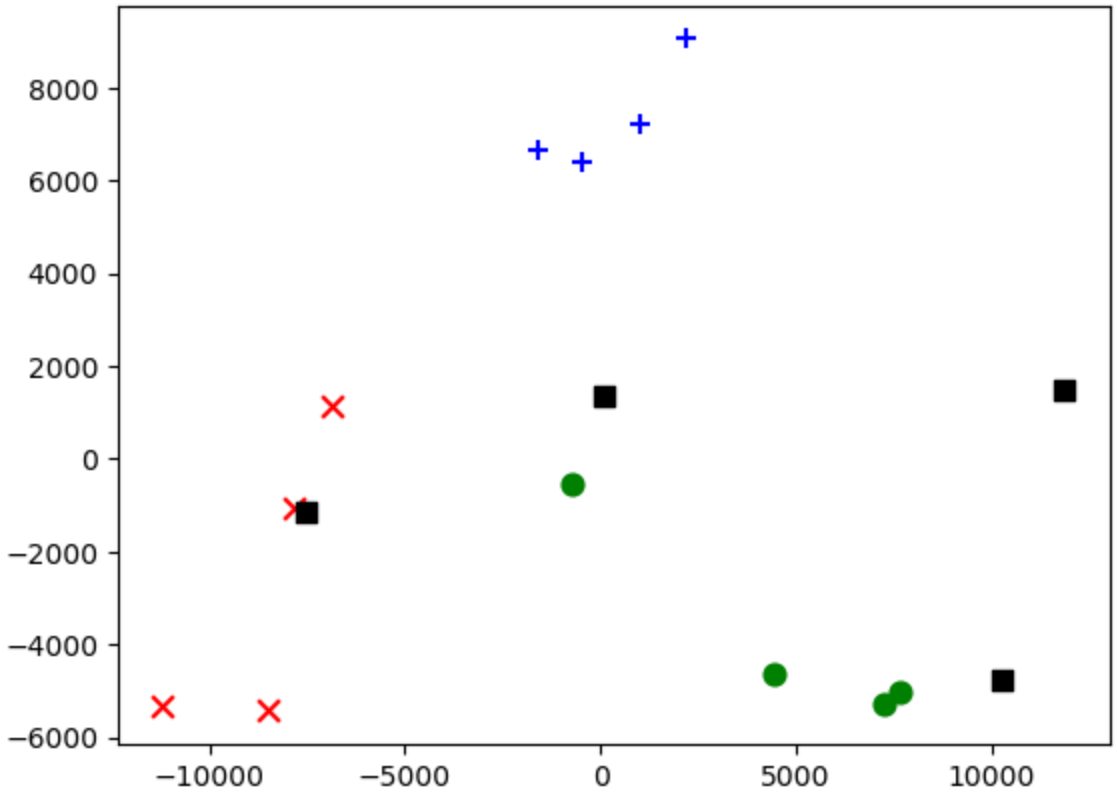
Finally, we draw a scatter plot to display the projected values. Observe that the images of burgers, drinks, and pastas are all projected to the same region. However, the images for fried chicken (shown as black squares in the diagram) are harder to discriminate.

Code:

In [50]:
```
import matplotlib.pyplot as plt

colors = {'burger':'b', 'drink':'r', 'pasta':'g', 'chicken':'k'}
markerTypes = {'burger':'+', 'drink':'x', 'pasta':'o', 'chicken':'s'}

for foodType in markerTypes:
    d = projected[projected['food']==foodType]
    plt.scatter(d['pc1'],d['pc2'],c=colors[foodType],s=60,marker=markerTypes[foodType]
```



In [ ]: