Data Science Lab 3: Pandas, Scikit-Learn

Woong-Kee Loh

2022

Install

- Launch *Command Prompt* as *Administrator*
- Run:
  - python -m pip install -U pip
  - python -m pip install -U pandas scikit-learn
- Refer to:
  - https://pandas.pydata.org/
  - https://scikit-learn.org/

pandas

$$y_it = \beta' x_{it} + \mu_i + \epsilon_{it}$$

Pandas Overview

- A Python package that provides
  - fast, flexible, and expressive data structures making working with relational or labeled data easy and intuitive
  - fundamental high-level building block for doing practical, real world data analysis in Python
- Data structures

May have heterogeneous objects

Dimension	Name	Description
1	*Series*	1D labeled homogeneously-typed array (time-series)
2	*DataFrame*	General 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed column (table, matrix)

Object Creation

- Creating a Series

```
>>> import numpy as np
>>> import pandas as pd
>>> s = pd.Series([1, 3, 5, np.nan, 6, 8])
>>> s
0     1.0
1     3.0
2     5.0
3     NaN
4     6.0
5     8.0
dtype: float64
```

index

Object Creation

- Creating a DataFrame

```
>>> df = pd.DataFrame(np.random.randn(6, 4))
>>> df
          0         1         2         3
0 -0.066602  0.871518 -0.450363  0.393921
1  1.090587  0.687492 -0.159559 -1.314033
2  1.354810  2.578029  0.649564  1.430082
3  0.028198 -0.936094 -0.163997 -0.667506
4  0.715358  0.282984  0.077506  0.663482
5  1.463316  0.208368  0.343688  0.501415
```

column names

index

Object Creation

- Creating a DataFrame (*cont'd*)
  - With a *datetime* index and labeled columns
  - Datetime units – *ns*: nanosecond, *us*: microsecond, etc.

```
>>> dates = pd.date_range('20130101', periods=6)
>>> dates
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
>>> df = pd.DataFrame(np.random.randn(6, 4), index=dates,
                      columns=['A','B','C','D'])
>>> df
                   A         B         C         D
2013-01-01 -0.497082 -0.973194  2.448383 -1.293616
2013-01-02  0.167663  1.224422  0.493177 -0.040777
2013-01-03 -1.086327 -0.943617  0.248452 -0.945163
2013-01-04 -0.051333 -2.193703 -1.927687 -0.861045
2013-01-05  1.095492 -0.373013 -0.629053 -0.580930
2013-01-06 -1.761057  0.686159 -1.652103  1.452296
```

M(onth), D(ay), H(our)

Object Creation

- Creating a DataFrame (*cont'd*)
  - Passing a dict of objects that can be converted to series-like

```
>>> df2 = pd.DataFrame({'A': 1.,
                        'B': pd.Timestamp('20130102'),
                        'C': pd.Series([1,2,3,4], dtype='float32'),
                        'D': np.array([3]*4, dtype='int32'),
                        'E': pd.Categorical(["train", "test", "train", "test"]),
                        'F': 'foo'})
>>> df2
     A          B    C  D      E    F
0  1.0 2013-01-02  1.0  3  train  foo
1  1.0 2013-01-02  2.0  3   test  foo
2  1.0 2013-01-02  3.0  3  train  foo
3  1.0 2013-01-02  4.0  3   test  foo
```

Viewing Data

- View top and bottom rows

```
>>> df.head() # default count = 5
                   A         B         C         D
2013-01-01  1.217339 -0.232313 -1.067197  0.094020
2013-01-02 -0.753951  0.548411  1.974326 -0.333166
2013-01-03  0.795369 -0.190015  1.158397  1.356929
2013-01-04 -0.915161  0.004413 -0.127103 -0.396280
2013-01-05 -0.799695  0.307816  0.273115  1.120795
>>> df.tail(2)
                   A         B         C         D
2013-01-05 -0.799695  0.307816  0.273115  1.120795
2013-01-06 -1.103973  0.392896  0.722698  0.768046
```

- View index, columns

```
>>> df.index
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
>>> df.columns
Index(['A', 'B', 'C', 'D'], dtype='object')
```

Viewing Data

- Get a NumPy representation of underlying data

```
>>> df.to_numpy()
array([[ 1.21733896, -0.23231318, -1.0671974 ,  0.09401981],
       [-0.75395061,  0.54841076,  1.9743259 , -0.33316563],
       [ 0.79536942, -0.19001537,  1.15839726,  1.35692876],
       [-0.9151613 ,  0.00441301, -0.12710348, -0.39627975],
       [-0.79969538,  0.30781642,  0.27311519,  1.12079533],
       [-1.10397347,  0.39289625,  0.72269781,  0.76804575]])
```

- Sort the data by value
  - in ascending order by default

```
>>> df.sort_values(by='B', ascending=False)
                   A         B         C         D
2013-01-02 -0.753951  0.548411  1.974326 -0.333166
2013-01-06 -1.103973  0.392896  0.722698  0.768046
2013-01-05 -0.799695  0.307816  0.273115  1.120795
2013-01-04 -0.915161  0.004413 -0.127103 -0.396280
2013-01-03  0.795369 -0.190015  1.158397  1.356929
2013-01-01  1.217339 -0.232313 -1.067197  0.094020
```

Selecting Data

- Column selection
  - returns a Series; equivalent to df.A

```
>>> df['A']
2013-01-01    1.217339
2013-01-02   -0.753951
2013-01-03    0.795369
2013-01-04   -0.915161
2013-01-05   -0.799695
2013-01-06   -1.103973
Freq: D, Name: A, dtype: float64
```

- Row selection
  - equivalent to df['20130102':'20130104']

```
>>> df[1:4]
                   A         B         C         D
2013-01-02 -0.753951  0.548411  1.974326 -0.333166
2013-01-03  0.795369 -0.190015  1.158397  1.356929
2013-01-04 -0.915161  0.004413 -0.127103 -0.396280
```

Selecting Data

- Select a subset of a Data Frame based on column values

```
students = [ ('jack', 'Apples' , 34) ,
        ('Riti', 'Mangos'  , 31) ,
        ('Aadi', 'Grapes' , 30) ,
        ('Sonia', 'Apples', 32) ,
        ('Lucy', 'Mangos'  , 33) ,
        ('Mike', 'Apples' , 35)
         ]
#Create a DataFrame object
dfx = pd.DataFrame(students, columns = ['Name' , 'Product', 'Sale'])
```

```
0  jack  Apples   34
1  Riti  Mangos   31
2  Aadi  Grapes   30
3  Sonia  Apples   32
4  Lucy  Mangos   33
5  Mike  Apples   35
```

Select Data

- (cont'd) Select rows in the above DataFrame for which the 'Product' column contains the value 'Apples'.

```
subDF = dfx[dfx['Product'] == 'Apples']
```

	Name	Product	Sale
0	jack	Apples	34
3	Sonia	Apples	32
5	Mike	Apples	35

Select Data

- (cont'd) How does this work internally?

  dfx['Product'] == 'Apples'] returns a Series of True or False

```
0    True
1    False
2    False
3    True
4    False
5    True
Name: Product, dtype: bool
```

When we pass this series object to the () operator of Data Frame "dfx()", it will return a new DataFrame with only those rows that have True in the passed Series object.

Selecting Data

- Selection by label
  - selecting a cross section using a label

```
>>> df.loc[dates[0]] # equivalent to df.loc['20130101']
A    1.217339
B   -0.232313
C   -1.067197
D    0.094020
Name: 2013-01-01 00:00:00, dtype: float64
```

  - selecting on a multi-axis by label

```
>>> df.loc[:, ['A','B']] # equivalent to df.loc['20130101':'20130106', ['A','B']]
                   A         B
2013-01-01  1.217339 -0.232313
2013-01-02 -0.753951  0.548411
2013-01-03  0.795369 -0.190015
2013-01-04 -0.915161  0.004413
2013-01-05 -0.799695  0.307816
2013-01-06 -1.103973  0.392896
```

Selecting Data

- Selection by position

```
>>> df.iloc[0]
A    1.217339
B   -0.232313
C   -1.067197
D    0.094020
Name: 2013-01-01 00:00:00, dtype: float64
>>> df.iloc[3:5, 0:2] # exclude end position
                   A         B
2013-01-04 -0.915161  0.004413
2013-01-05 -0.799695  0.307816
>>> df.iloc[[1, 2, 4], [0, 2]]
                   A         C
2013-01-02 -0.753951  1.974326
2013-01-03  0.795369  1.158397
2013-01-05 -0.799695  0.273115
```

Setting Data

- Adding a new column

```
>>> df3 = df.copy() # deep copy of df
>>> df3['F'] = pd.Series([2,3,4,5,6,7], index=pd.date_range('20130101', periods=6))
>>> df3
                   A         B         C         D  F
2013-01-01  1.217339 -0.232313 -1.067197  0.094020  2
2013-01-02 -0.753951  0.548411  1.974326 -0.333166  3
2013-01-03  0.795369 -0.190015  1.158397  1.356929  4
2013-01-04 -0.915161  0.004413 -0.127103 -0.396280  5
2013-01-05 -0.799695  0.307816  0.273115  1.120795  6
2013-01-06 -1.103973  0.392896  0.722698  0.768046  7
```

Setting Data

- Setting values

```
>>> df3.at[dates[0], 'A'] = 0 # by label
>>> df3.iat[0, 1] = 0 # by position
>>> df3.loc[:, 'D'] = np.array([5.] * len(df3)) # by assigning a numpy array
>>> df3
                   A         B         C    D  F
2013-01-01  0.000000  0.000000 -1.067197  5.0  2
2013-01-02 -0.753951  0.548411  1.974326  5.0  3
2013-01-03  0.795369 -0.190015  1.158397  5.0  4
2013-01-04 -0.915161  0.004413 -0.127103  5.0  5
2013-01-05 -0.799695  0.307816  0.273115  5.0  6
2013-01-06 -1.103973  0.392896  0.722698  5.0  7
```

Categorical Data

- A DataFrame including categorical data

```
>>> grade = pd.Series(['A','B','B','A','A','C'])
>>> df = pd.DataFrame({'id': [2,3,5,6,7,9],
                       'grade': grade.astype('category')})
>>> df
   id grade
0   2     A
1   3     B
2   5     B
3   6     A
4   7     A
5   9     C
>>> df['grade']
0    A
1    B
2    B
3    A
4    A
5    C
Name: grade, dtype: category
Categories (3, object): [A, B, C]
```

casts a Panda object to a specified dtype

Categorical Data

- Renaming categories

```
>>> df['grade'].cat.categories
Index(['A', 'B', 'C'], dtype='object')
>>> df['grade'].cat.categories = ['수','우','미']
>>> df
   id grade
0   2     수
1   3     우
2   5     우
3   6     수
4   7     수
5   9     미
```

- Reorder categories & add missing categories

```
>>> df['grade'].cat.categories
Index(['A', 'B', 'C'], dtype='object')
>>> df['grade'] = df['grade'].cat.set_categories(['수','우','미','양','가'])
>>> df['grade'].cat.categories
Index(['수', '우', '미', '양', '가'], dtype='object')
```

Missing Data

- Representation
  - Pandas primarily uses the value np.nan to represent missing data. It is by default not included in co[illegible]

```
>>> df3.loc[0:3, 'G'] = 1
>>> df3
                   A         B         C         D  F    G
2013-01-01  1.217339 -0.232313 -1.067197  0.094020  2  1.0
2013-01-02 -0.753951  0.548411  1.974326 -0.333166  3  1.0
2013-01-03  0.795369 -0.190015  1.158397  1.356929  4  1.0
2013-01-04 -0.915161  0.004413 -0.127103 -0.396280  5  NaN
2013-01-05 -0.799695  0.307816  0.273115  1.120795  6  NaN
2013-01-06 -1.103973  0.392896  0.722698  0.768046  7  NaN
```

missing data

Missing Data

- Operations

```
>>> df3.dropna(how='any') # drop any rows that have missing data; df3 not affected
                   A         B         C         D  F    G
2013-01-01  1.217339 -0.232313 -1.067197  0.094020  2  1.0
2013-01-02 -0.753951  0.548411  1.974326 -0.333166  3  1.0
2013-01-03  0.795369 -0.190015  1.158397  1.356929  4  1.0
>>> df3.fillna(value=0) # fill missing data
                   A         B         C         D  F    G
2013-01-01  1.217339 -0.232313 -1.067197  0.094020  2  1.0
2013-01-02 -0.753951  0.548411  1.974326 -0.333166  3  1.0
2013-01-03  0.795369 -0.190015  1.158397  1.356929  4  1.0
2013-01-04 -0.915161  0.004413 -0.127103 -0.396280  5  0.0
2013-01-05 -0.799695  0.307816  0.273115  1.120795  6  0.0
2013-01-06 -1.103973  0.392896  0.722698  0.768046  7  0.0
>>> df3.isna() # boolean mask of missing data
                A      B      C      D      F      G
2013-01-01  False  False  False  False  False  False
2013-01-02  False  False  False  False  False  False
2013-01-03  False  False  False  False  False  False
2013-01-04  False  False  False  False  False   True
2013-01-05  False  False  False  False  False   True
2013-01-06  False  False  False  False  False   True
```

Getting Data In/Out

- CSV file

```
>>> df.to_csv('c:/work/foo.csv') # writing to a csv file
>>> df = pd.read_csv('c:/work/foo.csv') # read from a csv file
>>> df
   Unnamed: 0 Unnamed: 0.1         A         B         C         D
0           0   2013-01-01  1.217339 -0.232313 -1.067197  0.094020
1           1   2013-01-02 -0.753951  0.548411  1.974326 -0.333166
2           2   2013-01-03  0.795369 -0.190015  1.158397  1.356929
3           3   2013-01-04 -0.915161  0.004413 -0.127103 -0.396280
4           4   2013-01-05 -0.799695  0.307816  0.273115  1.120795
5           5   2013-01-06 -1.103973  0.392896  0.722698  0.768046
```

Getting Data In/Out

- Excel file
  - Install libraries
    - python -m pip install -U xlrd openpyxl

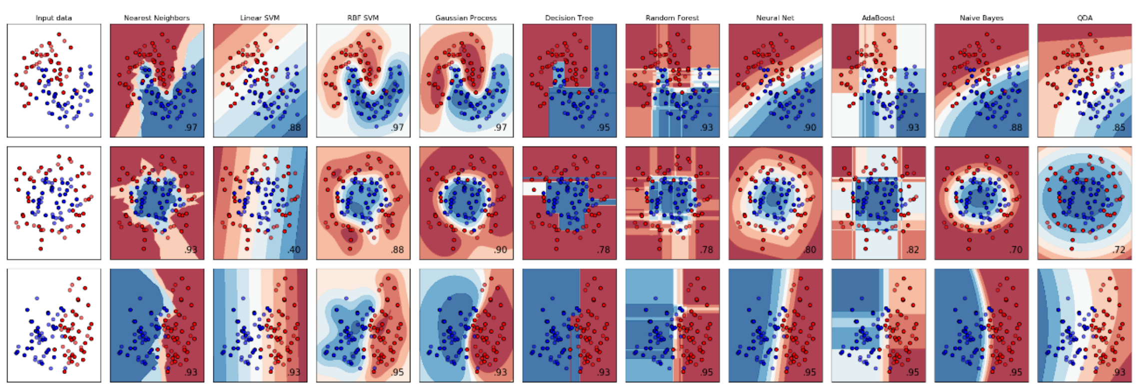
```
>>> df3.to_excel('c:/work/foo.xlsx', sheet_name='Sheet1')
>>> df4 = pd.read_excel('c:/work/foo.xlsx', 'Sheet1', index_col=None)
>>> df4
  Unnamed: 0         A         B         C         D  F    G
0 2013-01-01  1.217339 -0.232313 -1.067197  0.094020  2  1.0
1 2013-01-02 -0.753951  0.548411  1.974326 -0.333166  3  1.0
2 2013-01-03  0.795369 -0.190015  1.158397  1.356929  4  1.0
3 2013-01-04 -0.915161  0.004413 -0.127103 -0.396280  5  NaN
4 2013-01-05 -0.799695  0.307816  0.273115  1.120795  6  NaN
5 2013-01-06 -1.103973  0.392896  0.722698  0.768046  7  NaN
```

More Pandas

- See:
  - http://pandas.pydata.org/pandas-docs/version/0.24/getting_started/10min.html
  - http://pandas.pydata.org/pandas-docs/version/0.24/user_guide/cookbook.html
- For topics:
  - Operations – statistics, function application, string methods
  - Merging – concatenate, join, append
  - Grouping, reshaping, plotting

Introduction

- Scikit-learn
  - Free software machine learning library for Python
    - designed to interoperate with Python numerical and scientific libraries NumPy and SciPy
  - Features various classification, regression, clustering, transformation algorithms
    - e.g., support vector machines (SVM), random forests, gradient boosting, k-means, and DBSCAN



Regression

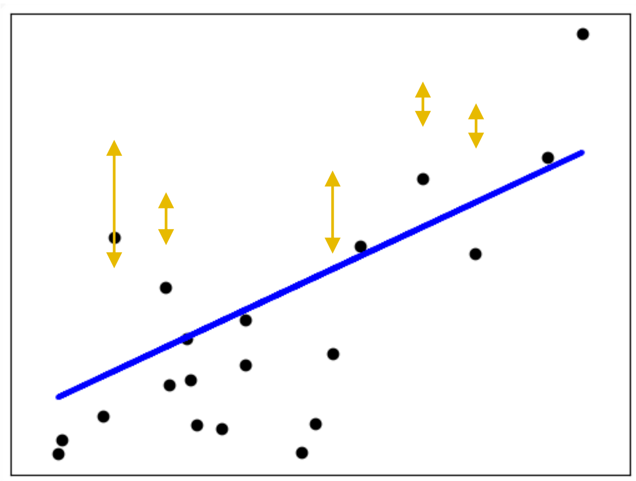
Simple Linear Regression	$y = b_0 + b_1 x_1$
Multiple Linear Regression	$y = b_0 + b_1 x_1 + b_2 x_2 + \dots + b_n x_n$
Polynomial Linear Regression	$y = b_0 + b_1 x_1 + b_2 x_1^2 + \dots + b_n x_1^n$

Linear and Multiple Regression

- Multiple regression is a generalized linear model
  - Target value is expected to be a linear combination of the input variables
    - $\hat{y}(w, x) = w_0 + w_1 x_1 + \ldots + w_p x_p$
    - $\hat{y}$: predicted value
  - Designations
    - w_0 as intercept_ and $w = (w_1, \ldots, w_p)$ as coefficients_
  - Note
    - Coefficients w_i are also called weights.
    - They are designated as w_i, b_i, or β_i

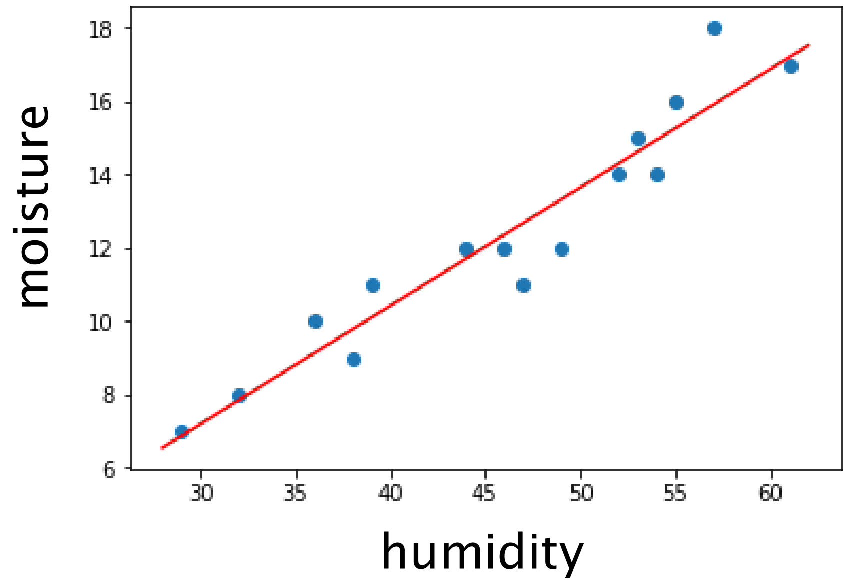
Ordinary Least Squares

- *LinearRegression* class
  - Minimizes the residual (differences) sum of squares between the observed target values y in the dataset and the target values $\hat{y}$ predicted by the linear approximation
  - i.e., solves $\min_w \|Xw - y\|_2{}^2$



Example: Ordinary Least Squares

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
humidity = np.array([46, 53, 29, 61, 36, 39, 47, 49, 52, 38, 55, 32, 57, 54, 44])
moisture = np.array([12, 15,  7, 17, 10, 11, 11, 12, 14,  9, 16,  8, 18, 14, 12])
reg = linear_model.LinearRegression() # an object for linear regression
reg.fit(humidity[:, np.newaxis], moisture)
    # fit linear model; param #1 is a list of X vectors (x_1, ..., x_p)
px = np.array([humidity.min()-1, humidity.max()+1])
py = reg.predict(px[:, np.newaxis]) # predict using the linear model
plt.scatter(humidity, moisture)
plt.plot(px,py,color='r')
plt.show()
```



Pause… *newaxis* (1/3)

- https://stackoverflow.com/questions/29241056/how-does-numpy-newaxis-work-and-when-to-use-it
- newaxis is used to increase the dimension of an existing array by one more dimension
  - 1-d array will become a 2-d array, etc.
- Use case scenario 1: convert a 1d array into a vector

```
arr = np.arange(4)
row_vec = arr[np.newaxis, :]   * make arr a row vector
row_vec.shape
(1, 4)
col_vec = arr[ :, np.newaxis]   * make arr a column vector
col_vec.shape
(4, 1)
```

Pause… *newaxis* (2/3)

- Use case scenario 2: make use of NumPy broadcasting of some operation (e.g., addition of arrays).

```
x1 = np.array([1, 2, 3, 4, 5])
x2 = np.array([5, 4, 3])
x1_new = x1[ :, np.newaxis]   * shape of new_x1 is (5, 1)
x1_new + x2
```

```
array([[ 6, 5, 4],
       [7, 6, 5],
       [8, 7, 6],
       [9, 8, 7],
       [10, 9, 8]])
```

Pause… *newaxis* (3/3)

- Use case scenario 3: use np.newaxis more than once to promote an array to higher dimensions.

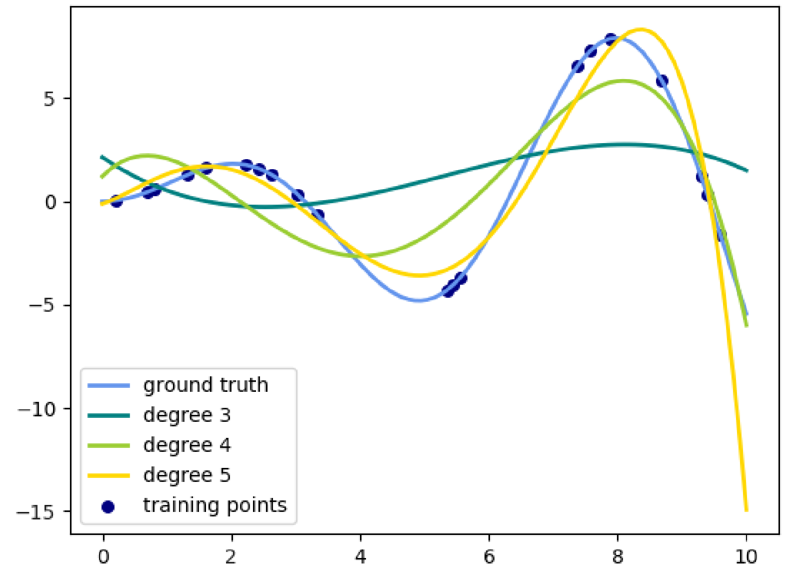
```
  arr = np.arange(5*5).reshape(5,5)   * shape is (5, 5)
arr_5D = arr[np.newaxis, …, np.newaxis, np.newaxis]
arr_5D.shape
(1, 5, 5, 1, 1)
```

- newaxis vs. reshape
  - newaxis uses slicing to recreate the array, while reshape reshapes the array to the desired layout

```
A = np.ones ((3, 4, 5, 6))
B = np.ones ((4, 6))
(A + B[ :, np.newaxis,]).shape     *  inserts temporary axis
(3, 4, 5, 6)       * between the first and second axes of B to make
                   *  the broadcasting operation work
```

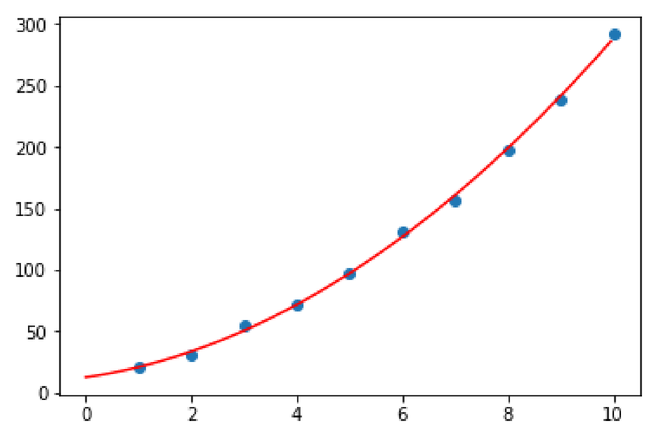
Polynomial Regression

- Use trained linear models on nonlinear functions
  - See https://scikit-learn.org/stable/modules/linear_model.html#polynomial-regression-extending-linear-models-with-basis-function
- *PolynomialFeatures* preprocessor
  - Transforms an input data matrix (a list of X) into a new data matrix of a given degree



Example: Polynomial Regression

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
x = np.arange(1,11)
Y = np.array([20.6, 30.8, 55.0, 71.4, 97.3, 131.8, 156.3, 197.3, 238.7, 291.7])
model = Pipeline([('poly', PolynomialFeatures(degree=2)),
                  ('linear', LinearRegression(fit_intercept=False))])
    # an object representing a simple order-2 polynomial regression
    # preprocessing can be streamlined with the Pipeline tools
model = model.fit(x[:, np.newaxis], Y) # fit to a polynomial data
px = np.arange(0, 10, 0.05)
pY = model.predict(px[:, np.newaxis]) # predict using the polynomial model
plt.scatter(x,Y)
plt.plot(px,pY,color='r')
plt.show()
```



Preprocessing

- *sklearn.preprocessing* package
  - Provides common utility functions and transformer classes
  - Changes raw feature vectors into a representation that is more suitable for the downstream estimators (models)

Preprocessing

- Standardization
  - Transforms to standard normally distributed data, i.e., Gaussian with zero mean and unit variance

```
>>> import numpy as np
>>> from sklearn import preprocessing
>>> score = np.array([20, 15, 26, 32, 18, 28, 35, 14, 26, 22, 17], dtype=float)
>>> score_scale = preprocessing.scale(score)
>>> score_scale
array([-0.45226702, -1.20604538,  0.45226702,  1.35680105, -0.75377836,
        0.75377836,  1.80906807, -1.35680105,  0.45226702, -0.15075567,
       -0.90453403])
```

Preprocessing

- Scaling features to a range
  - *MinMaxScaler* – scaling features to lie between a given minimum and maximum value, often between zero and one
  - *MaxAbsScaler* – scale to (-1, 1) by dividing each value in a column by the absolute max value

```
>>> score = np.array([20, 15, 26, 32, 18, 28, 35, 14, 26, 22, 17], dtype=float)
>>> score = score-np.median(score)
>>> score
array([-2., -7.,  4., 10., -4.,  6., 13., -8.,  4.,  0., -5.])
>>> minMaxScaler = preprocessing.MinMaxScaler()
>>> score_minmax = minMaxScaler.fit_transform(score[:, np.newaxis]).reshape(-1)
>>> score_minmax
array([0.28571429, 0.04761905, 0.57142857, 0.85714286, 0.19047619,
       0.66666667, 1.        , 0.        , 0.57142857, 0.38095238,
       0.14285714])
>>> maxAbsScaler = preprocessing.MaxAbsScaler()
>>> score_maxabs = maxAbsScaler.fit_transform(score[:, np.newaxis]).reshape(-1)
>>> score_maxabs
array([-0.15384615, -0.53846154,  0.30769231,  0.76923077, -0.30769231,
        0.46153846,  1.        , -0.61538462,  0.30769231,  0.        ,
       -0.38461538])
```

Fit to data, then transform it

Pause… *reshape(-1)*

- "-1" means Numpy should figure out the dimension compatible with the original shape.
- z = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
  z.shape
  (3, 4)
- z.reshape(-1)
  array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
- z.reshape(2, -1)
  array([[1, 2, 3, 4, 5, 6], [7, 8, 9, 10, 11, 12]])
- z.reshape(-1, 2)
  array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]])

Encoding Categorical Features

- *OrdinalEncoder*
  - Transforms each categorical feature to one new feature of integers (0 ~ #categories - 1)

```
>>> enc = preprocessing.OrdinalEncoder()
>>> X = [['male','from US','uses Safari'],
      ['female','from Europe','uses Firefox'],
      ['male','from Asia','uses Chrome']]
>>> enc.fit(X)
OrdinalEncoder(categories='auto', dtype=<class 'numpy.float64'>)
>>> enc.transform(
    [['female', 'from US', 'uses Firefox'],
     ['male', 'from Europe', 'uses Chrome']])
array([[0., 2., 1.],
       [1., 1., 0.]])
```

determines categories automatically from the training data (default)

Encoding Categorical Features

- *OrdinalEncoder* (*cont'd*)

male	from US	uses Safari
female	from Europe	uses Firefox
male	from Asia	uses Chrome

training records

fit

1	2	2
0	1	1
1	0	0

(apply)

female	from US	uses Firefox
male	from Europe	uses Chrome

records to transform

transform

0	2	1
1	1	0

Encoding Categorical Features

- *OneHotEncoder*
  - Transforms each categorical feature with #categories possible values into #categories binary features
    - i.e., one of them 1, and all others 0

```
>>> enc = preprocessing.OneHotEncoder()
>>> X = [['male','from US','uses Safari'],
     ['female','from Europe','uses Firefox'],
     ['male','from Asia','uses Chrome']]
>>> enc.fit(X)
OneHotEncoder(categorical_features=None, categories=None,
       dtype=<class 'numpy.float64'>, handle_unknown='error',
       n_values=None, sparse=True)
>>> enc.transform(
    [['female', 'from US', 'uses Firefox'],
     ['male', 'from Europe', 'uses Chrome']]).toarray()
array([[1., 0., 0., 0., 1., 0., 1., 0.],
       [0., 1., 0., 1., 0., 1., 0., 0.]])
```

Discretization

- *KBinsDiscretizer*
  - Discretizes features into *k* equal-width bins
  - For each feature, the bin edges are computed during fit, together with the number of bins

```
>>> import numpy as np
>>> from sklearn import preprocessing
>>> score = np.array([20, 15, 26, 32, 18, 28, 35, 14, 26, 22, 17], dtype=float)
>>> discretizer = preprocessing.KBinsDiscretizer(n_bins=[5], encode='ordinal')
>>> est = discretizer.fit(score[:,np.newaxis])
>>> est.transform(score[:,np.newaxis]).reshape(-1)
array([2., 0., 3., 4., 1., 4., 4., 0., 3., 2., 1.])
```

Notes on KBinsDiscretizer

- Parameters
  - n_bins: number of bins, default=5
  - encode: onehot, onehot-dense, ordinal

    default=onehot

    ordinal returns the bin ID as an integer value
  - strategy: quantile, uniform, kmeans

    default: quantile (all bins have the same number of points)

    uniform: all bins have the same width

fit and *transform* methods

- The fit() method calculates a model (e.g., formula, mean and std) on a training dataset.
- The transform() method applies the model to a testing dataset or new dataset.
- The fit_transform() method is basically a fit followed by a transform

Motivating Example

- To impute missing values, we will use the Scikit-learn imputer function.
- First, we will train the imputer using the fit method to calculate the means of a two-column (training) dataset

```
[[1,      2],
 [np.nan, 3],
 [7,      6]]
```

- The imputer learns to use the mean (1+7)/2 = 4.0 for the first column, and mean (2+3+6)/3 = 3.6666667 for the second column.
- Next, we will use the transform method to apply the model (i.e., the means) to a new two-column dataset X

  X = [[np.nan, 2], [6, np.nan], [7, 6]]
- The result is a transformed two-column dataset [[4., 2.], [6., 3.66666667], [7, 6]].
- The fit_transform() method can be used if both the training dataset and the new dataset are the same.

Imputation of Missing Values

- *SimpleImputer* class
  - Missing values are imputed with a provided constant value or using the statistics (mean, median, or most frequent)

```
>>> import numpy as np
>>> from sklearn.impute import SimpleImputer
>>> imp = SimpleImputer(missing_values=np.nan, strategy='mean')
>>> imp.fit([[1, 2],
         [np.nan, 3],
         [7, 6]])
SimpleImputer(copy=True, fill_value=None, missing_values=nan, strategy='mean',
       verbose=0)
>>> X = [[np.nan, 2],
     [6, np.nan],
     [7, 6]]
>>> imp.transform(X)
array([[4.        , 2.        ],
       [6.        , 3.66666667],
       [7.        , 6.        ]])
```

A Lot More Scikit-Learn

- Supervised learning:
  - generalized linear models, support vector machines, stochastic gradient descent, naive Bayes, decision trees, ensemble methods, feature selection, etc.
- Unsupervised learning:
  - Gaussian mixture models, clustering, covariance estimation, novelty & outlier detection, neural network models, etc.
- Model selection and evaluation:
  - cross validation, model evaluation, validation curves, etc.
- Dataset transformations:
  - pipelines, feature extraction, dimensionality reduction, etc.

Lab 3

- Dataset: bmi_data_lab3.csv
  - Attributes: gender, age, height (inches), weight (pounds), body mass index (BMI)
  - BMI: extremely weak (0), weak (1), normal (2), overweight (3), obesity (4)

Lab 3 (cont'd)

- Read the CSV dataset file
- Peek into the dataset (data exploration)
  - Print dataset statistical data, feature names & data types
  - Plot height & weight histograms (bins=10) for each BMI value
    - Use seaborn.FacetGrid or matplotlib.pyplot.subplots
  - Plot scaling results for height and weight
    - Use StandardScaler, MinMaxScaler, and RobustScaler
    - See preprocessing-1 PPT

Lab 3 (cont'd)

- Missing value manipulation (simple)
  - Identify all dirty records with likely-wrong or missing height or weight values (by eye inspection)
  - Remove all likely-wrong values; i.e., make them NAN
  - Print # of rows with NAN, and # of NAN for each column
  - Extract all rows without NAN
  - Fill NAN with mean, median, or using ffill / bfill methods

Lab 3 (cont'd)

- Hints
  - Sample Dataset

	A	B	C	D
1	Gender	Height	Weight	BMI
2	Male	174	96	4
3	Female	195	104	3
4	Male	405	90	
5	Female	159	80	
6	Female	192	-33	3
7	Male	155	51	2
8	Female	153	149	5
9	Female		97	4
10	Male	185		5
11	Female	172		2

header line: should be skipped

likely-wrong

missing

Lab 3 (cont'd)

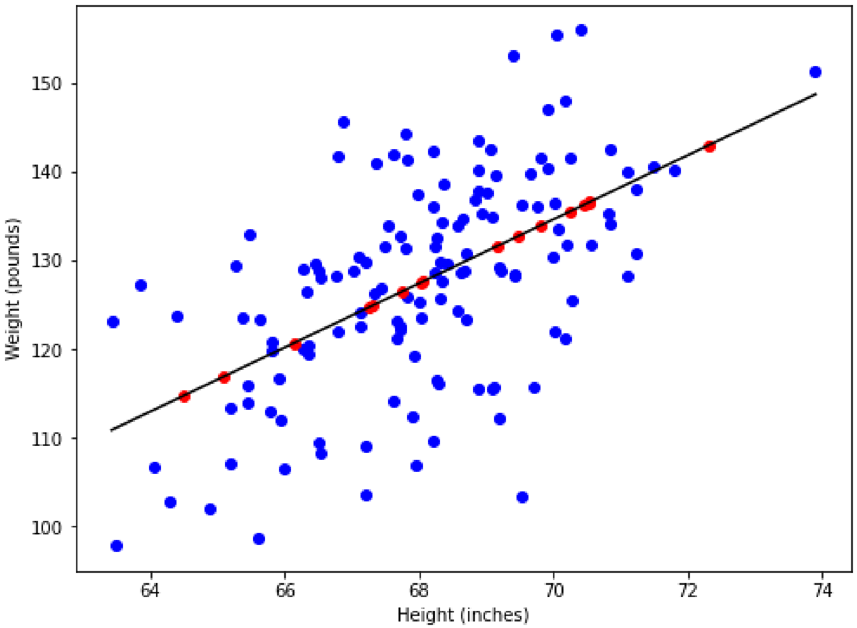
- Missing value manipulation (more elaborate)
  - Identify all dirty records with likely-wrong or missing height or weight values (by eye inspection)
  - Clean the dirty values using linear regression (see the next page)
  - Draw a scatter plot of (height, weight) in the clean dataset emphasizing previously dirty records with a different color

Lab 3 (cont'd)

- Cleaning the Input Dataset:
  - Compute the linear regression equation E for (height, weight) values in the input dataset
  - For dirty height and weight values, compute replacement values using E
    - Computed with known weight and height values, respectively
  - Do the same for the groups divided by gender and BMI, respectively
    - e.g., the dirty value of a female record is cleaned using the equation E_f computed for the female group
  - For a dirty record, compare the replacement values computed using different regression equations
    - e.g., the height replacement values for a dirty record (NAN, w) computed using E and E_f might be different

Lab 3 (cont'd)

- Hints
  - sample plot



Programming Homework 3

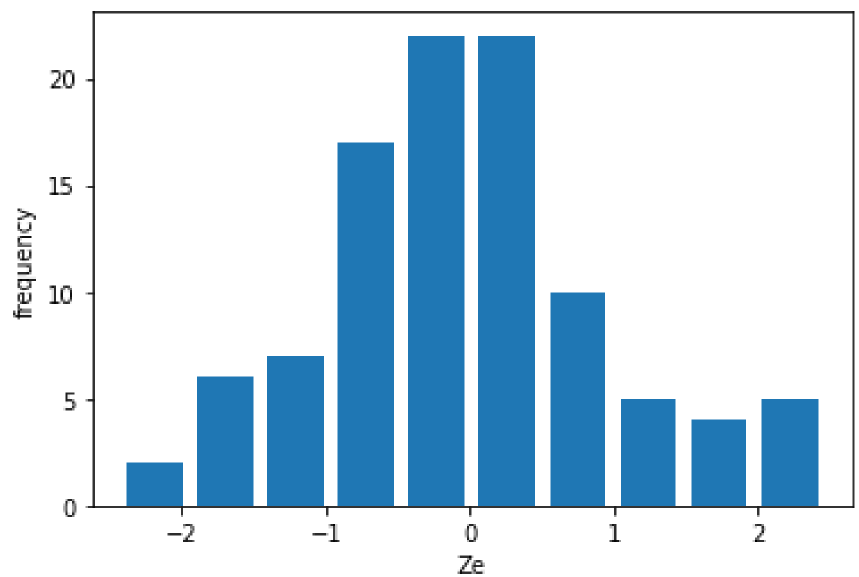
- Dataset: bmi_data_phw3.xlsx
  - Composed of the same attributes as in Lab3
  - Assume no missing or wrong values
- Data exploration
  - Print dataset statistical data, feature names & data types
  - Plot height & weight histograms (bins=10) for each BMI value
    - Use seaborn.FacetGrid or matplotlib.pyplot.subplots
  - Plot scaling results for height and weight
    - Use StandardScaler, MinMaxScaler, RobustScaler
    - See preprocessing-1 PPT

Programming Homework 3 (cont'd)

- Program: find outlier people
  - Although the given dataset has BMI values, we are going to estimate the values as if we don't know them
    - We will compare the estimated BMI values (0 and 4) with the actual values to see how much our estimation is correct
  - Read the Excel dataset file, and compute the linear regression equation E for the input dataset D
  - For (height h, weight w) of each record, compute $e=w-w'$, where w' is obtained for h using E
  - Normalize the e values, i.e., compute $z_e=[e-\mu(e)]/\sigma(e)$, and plot a histogram showing the distribution of z_e (~10 bins)
  - Decide a value α (≥ 0); for records with $z_e<-\alpha$, set BMI = 0; for those with $z_e>\alpha$, set BMI = 4

Programming Homework 3 (cont'd)

- Hint: Sample Plot



Programming Homework 3 (cont'd)

- More programming
  - Divide the input dataset D into two groups D_f and D_m according to gender
  - Do the same as done previously for each of D_f and D_m
  - Compare your BMI estimates with the actual BMI values in the given dataset

Notes on Lab & PHW

- Internal documentation
  - Comments (in English) are required for every important coding blocks, functions, parameters, and data structures (matrices)

Active Learning Homework: Jupyter Notebook or Colaboratory

- “Active learning” means students’ learning a topic on their own.
- Jupyter Notebook and Google Colaboratory are both Web-based IDE(integrated development environment). (IDE examples: Eclipse and Visual Studio).
- Study either Jupyter Notebook or Colaboratory.
- Do PHW3 twice: once without using it, and once using it.
- Then report on the practical advantages (if any) of using it. (The advantage may or may not be significant, depending on your needs.)

End of Lab 3

- Acknowledgments
  - Original sources of this presentation are
  - http://pandas.pydata.org/pandas-docs/version/0.24/getting_started/10min.html
  - https://scikit-learn.org/stable/modules/preprocessing.html
  - https://scikit-learn.org/stable/supervised_learning.html
- See also
  - https://pandas.pydata.org/pandas-docs/stable/reference/index.html
  - https://scikit-learn.org/stable/user_guide.html
  - https://scikit-learn.org/stable/modules/classes.html