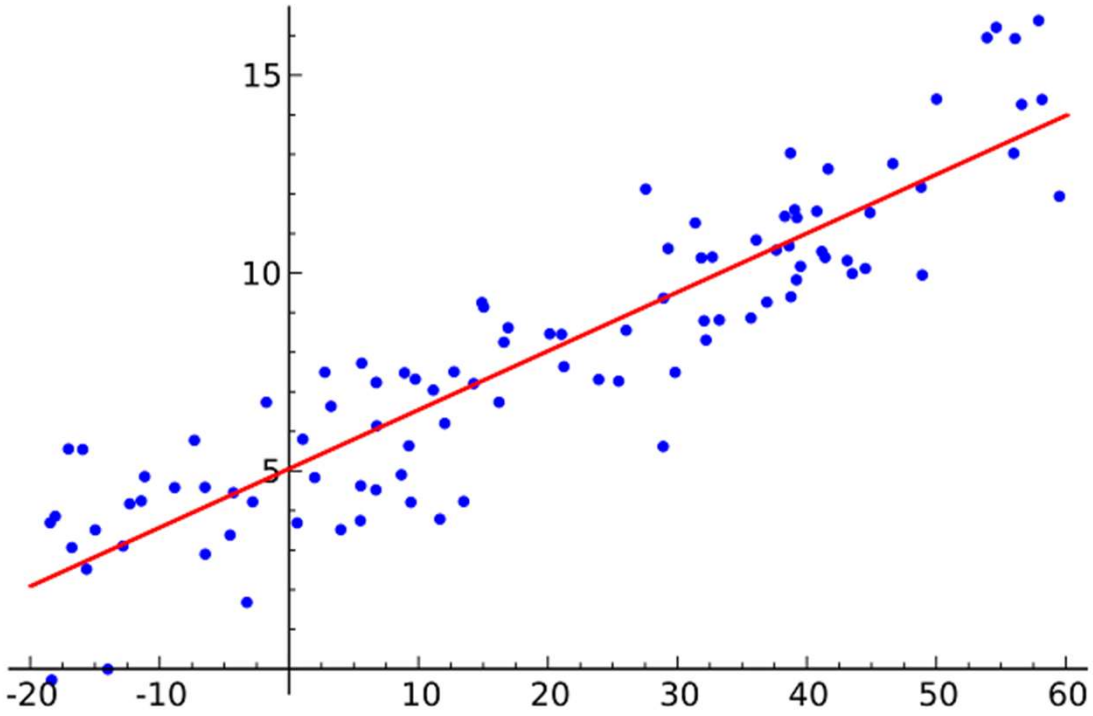


Chapter 10 Regression analysis

CSS 341 Introduction to Data Science

Chukiat Worasucheep

Learning objectives

1. เข้าใจแนวคิดของการวิเคราะห์การถดถอย (Regression analysis)
2. สร้างตัวแบบและตีความผลการวิเคราะห์การถดถอยเชิงเส้น (Linear Regression) ด้วย Python ได้
3. สร้างตัวแบบและตีความผลการใช้ Multiple Linear Regression ด้วย Python ได้
4. สร้างตัวแบบและตีความผลการใช้ Polynomial Regression ด้วย Python ได้
5. เข้าใจสาเหตุ ความสำคัญ หลักการ และแนวทางจัดการ Overfitting and Underfitting ได้

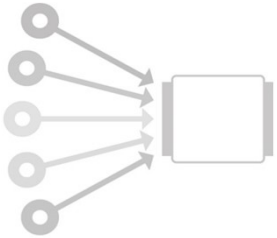
Important Notice

การเรียนการสอนหัวข้อนี้ ผ่านทางสื่อออนไลน์ (Online meeting)

และมีการบันทึกภาพและเสียงเพื่อประโยชน์ทางการศึกษาต่อไปในอนาคต.

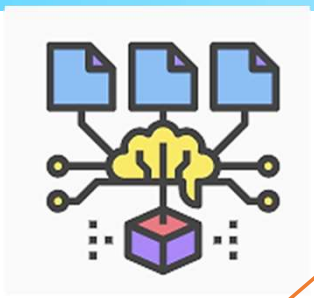
หากท่านไม่ยินยอมให้มีการเผยแพร่การบันทึกดังกล่าว ขอให้แจ้งให้ผู้สอนทราบภายใน 36 ชั่วโมง.

Data science process



Data acquisition 02

Data Preparation 04



06 Data modeling

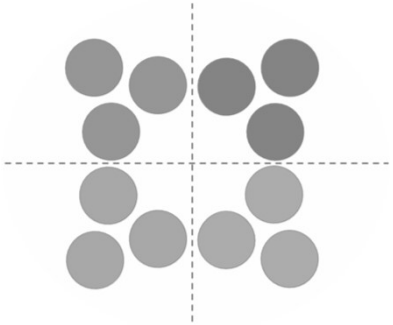
07 Deployment and maintenance

05 Data visualization

03 Data Exploration

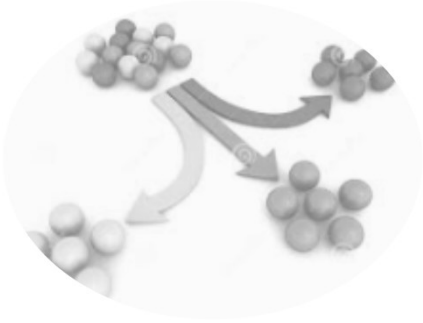
01 Understand business and problems

Major modeling techniques for data science



Clustering

การจัดกลุ่ม

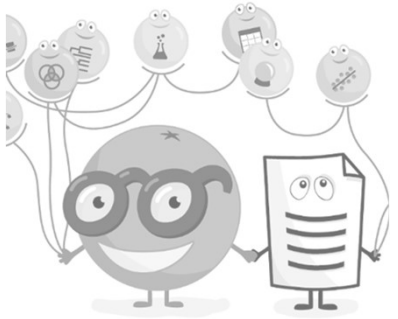


Classification

การจำแนกประเภท

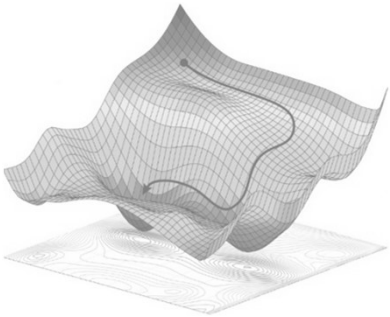
Regression

การวิเคราะห์ถดถอย



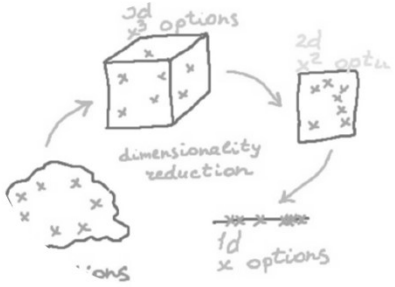
Association rule mining

การค้นหากฎความสัมพันธ์



Optimization

การหาค่าเหมาะที่สุด



Dimensionality reduction

การลดมิติข้อมูล

Contents

- Concept of regression analysis
- Linear Regression
- Multiple Linear Regression
- Polynomial Regression
- Overfitting and Underfitting
- More advanced regressors
- Case studies

Regression Analysis (การวิเคราะห์การถดถอย)

- *Regression analysis* is used to model the relationship between *a dependent variable (Target)* and *one or more independent variables (Predictors)*.

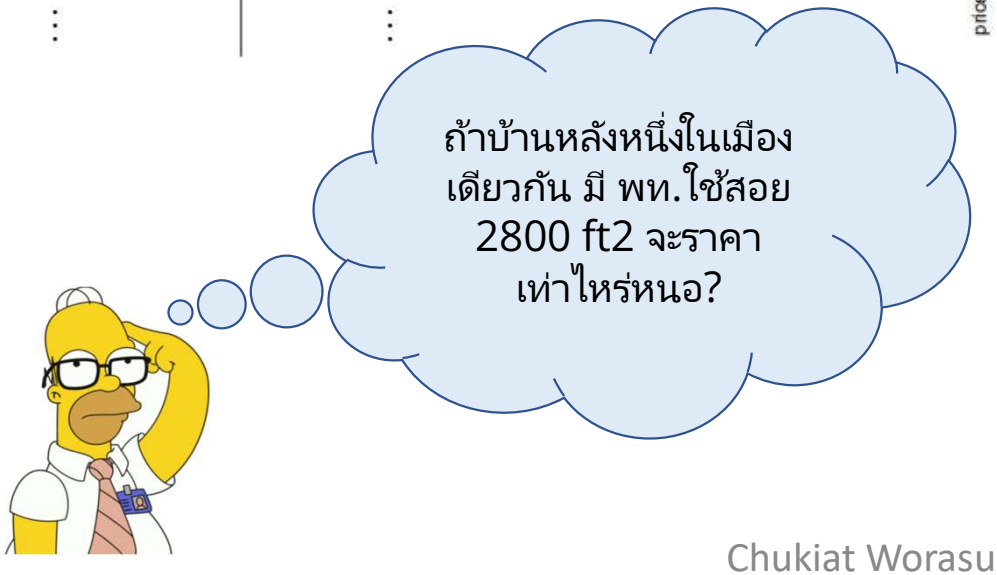
- Examples of applications:
  1. ยอดขายสินค้าขึ้นกับปัจจัยใดบ้าง และต้องการทราบประมาณการยอดขายเดือนหน้า
  2. การพยากรณ์ราคารถยนต์มือสองจากปัจจัยต่าง ๆ เช่น ความจุเครื่องยนต์ (cc) อัตราการใช้เชื้อเพลิง อายุรถ ความยาวรถ จำนวนครั้งอุบัติเหตุใหญ่ ฯลฯ
  3. ผลของปริมาณฝนต่อปริมาณผลผลิตผลไม้ชนิดหนึ่ง

Source: https://www.listendata.com/2018/03/regression-analysis.html#What-is-Regression-Analysis-

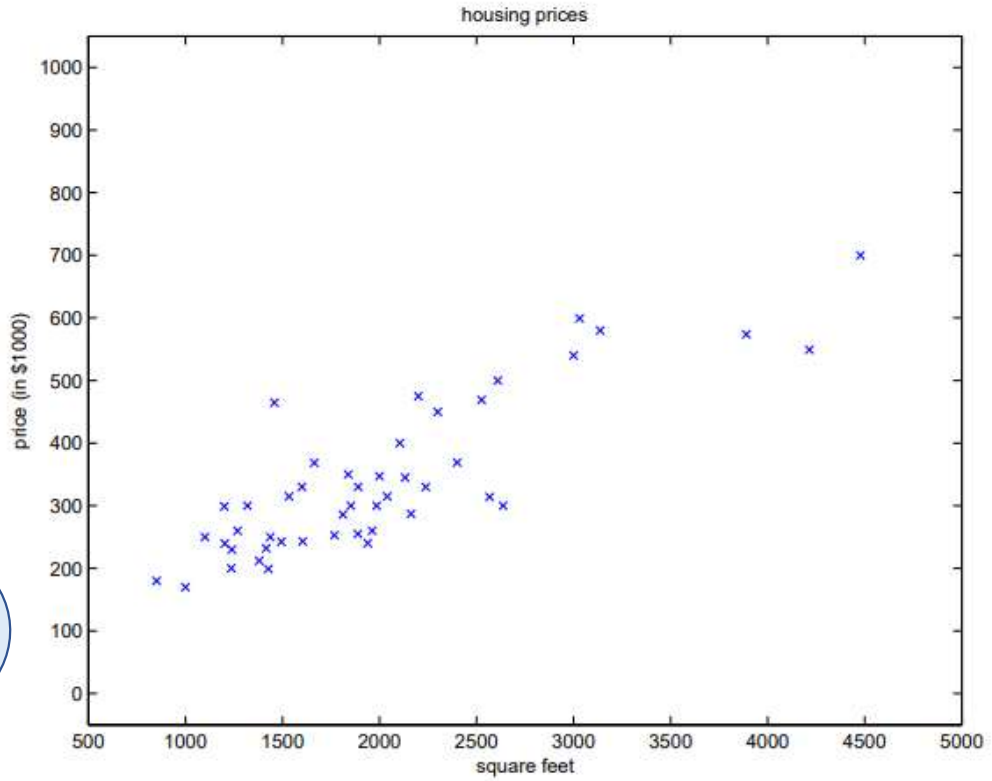
Example Application of Regression

- สมมติเรามีข้อมูลราคาบ้านชุดหนึ่งของเมือง Portland รัฐ Oregon

Living area ($feet^2$)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮



- นำมาพล็อตกราฟ



Regression analysis

- ด้วยข้อมูลเช่นในตารางข้างต้นนี้ เราสามารถคาดการณ์ (พยากรณ์) ราคาบ้านจาก พท.ใช้สอยได้หรือไม่ อย่างไร

- ปัญหาเช่นนี้ เรียกว่า *regression analysis* (การวิเคราะห์การถดถอย)

- เรียกข้อมูล Living area ว่าเป็นตัวแปรอิสระ *independent variable* (ในทางสถิติ) หรือ *feature* (ในทาง machine learning)
- เรียกข้อมูล Price ว่าเป็นตัวแปรตาม (*dependent variable*) (ในทางสถิติ)
- เรียกข้อมูลความสัมพันธ์ที่ให้มาแล้วนี้ว่า *training set*
- เรียกแต่ละชุดความสัมพันธ์ในนี้ว่า training example
- เรียกข้อมูลชุดที่ต้องการหาค่าตัวแปรตาม (เช่น 2800, etc.) ว่า testing example
- หากมี testing example หลายชุด จะเรียกรวมกันว่า *testing set*

More independent variables

Living area ($feet^2$)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

โมเดลรูปแบบนี้เรียกว่า

Multiple *Linear*

Regression model

ตัวแปรตาม

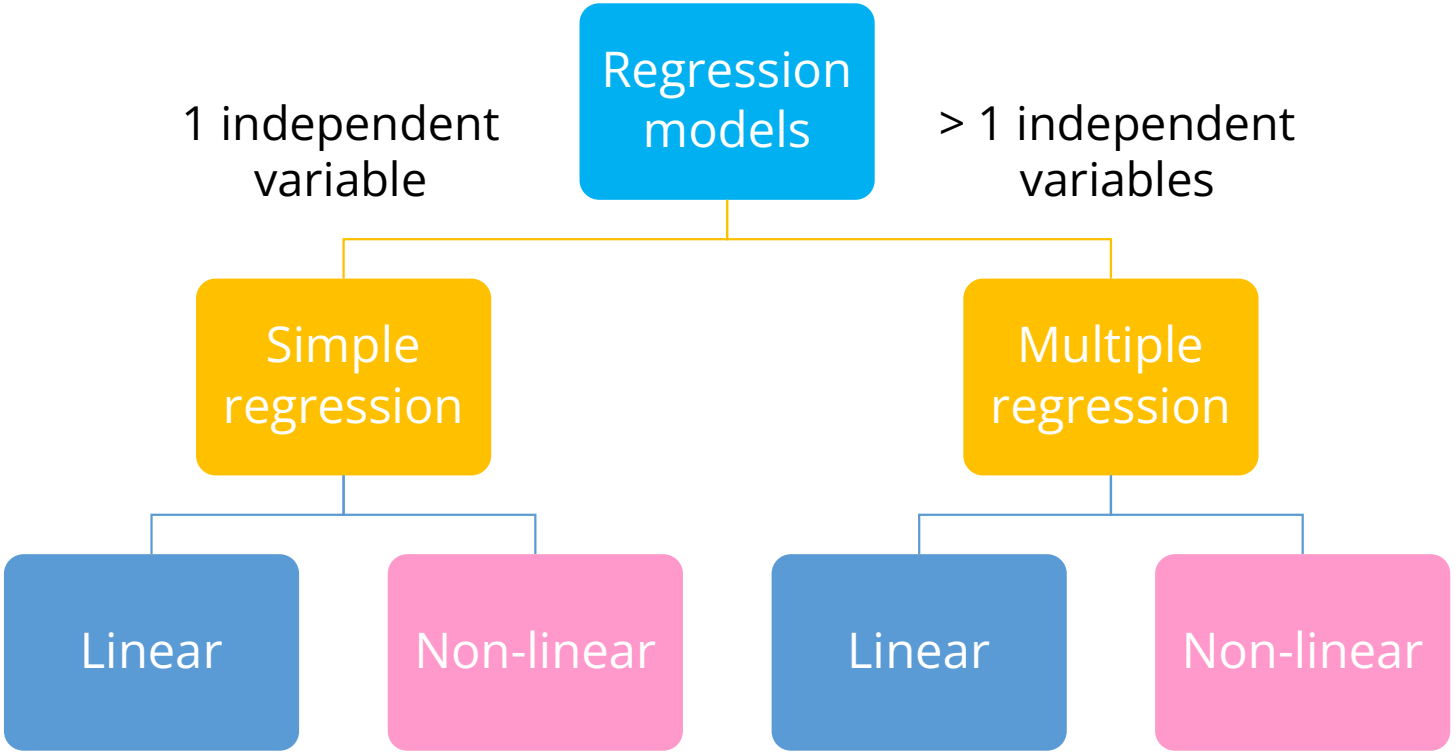
ตัวแปรต้น

$$price = \beta_1 \times area + \beta_2 \times no.of.bedrooms + \varepsilon$$

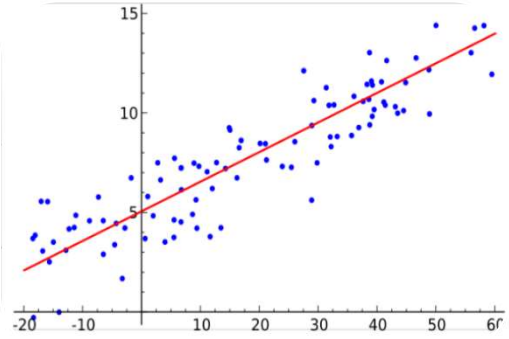
สัมประสิทธิ์ของตัวแปรต้น

ค่าความคลาดเคลื่อน

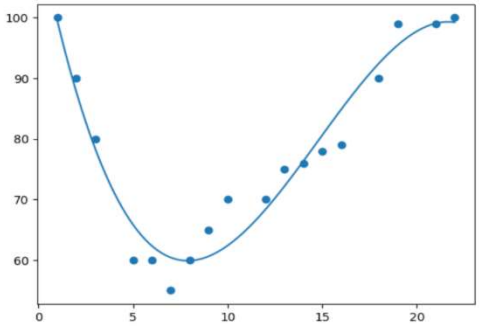
Types of Regression Models



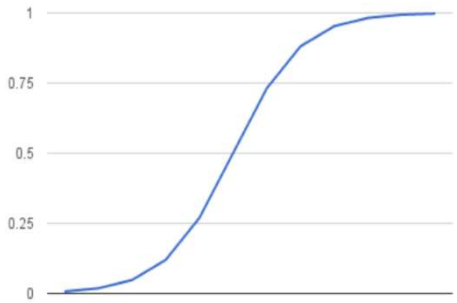
A broader types of regression analysis



Linear regression



Polynomial regression



Logistic regression *(Classification)*

etc

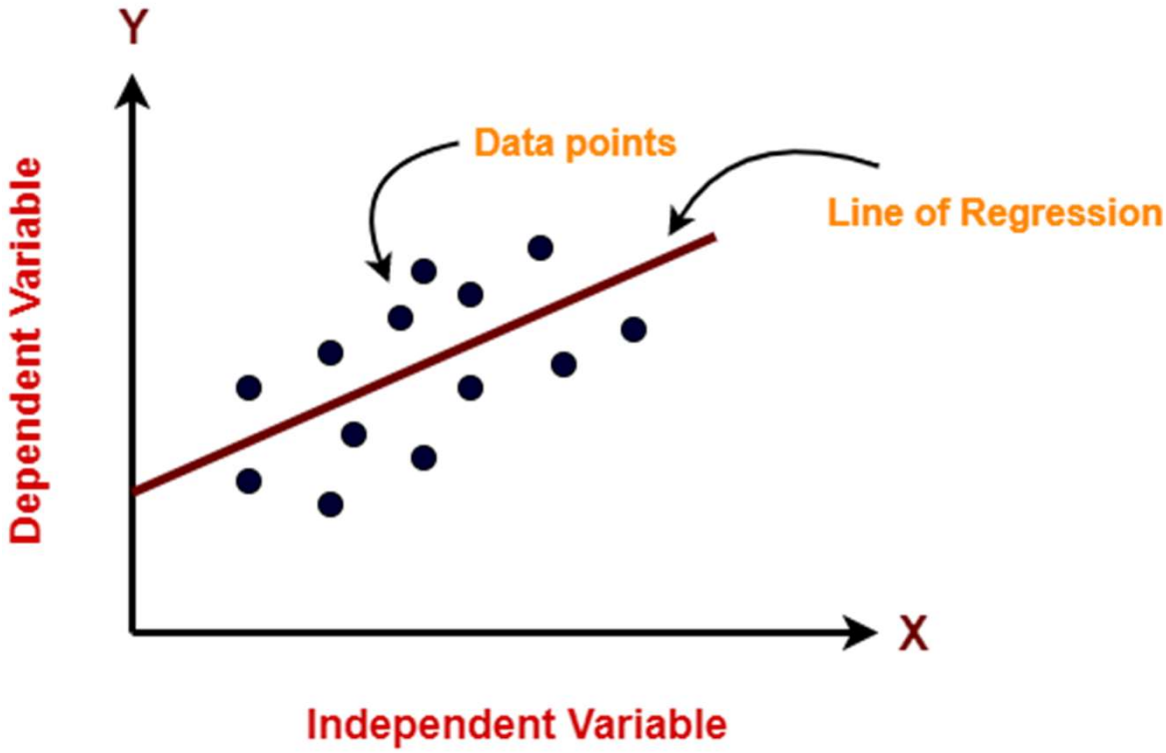
.

Contents

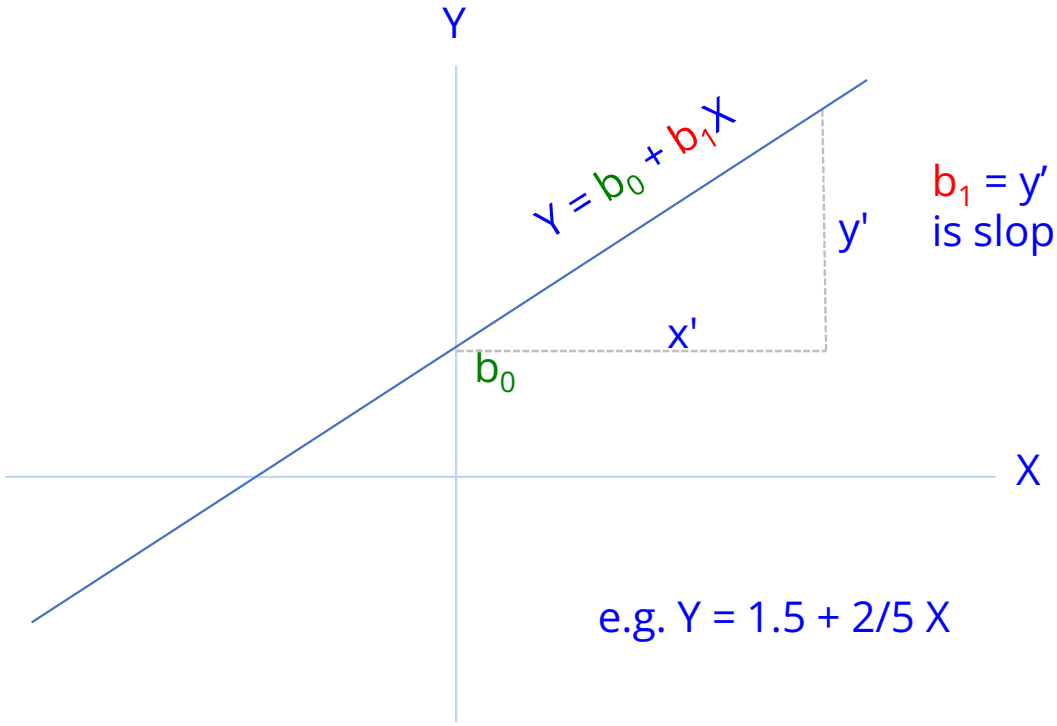
- Concept of regression analysis
- Linear Regression
- Multiple Linear Regression
- Polynomial Regression
- Overfitting and Underfitting
- More advanced regressors
- Case studies

Linear regression

- Linear regression is a simple approach to supervised learning. It *assumes* that the dependence of Y on X1, X2, . . . Xp is *linear*.



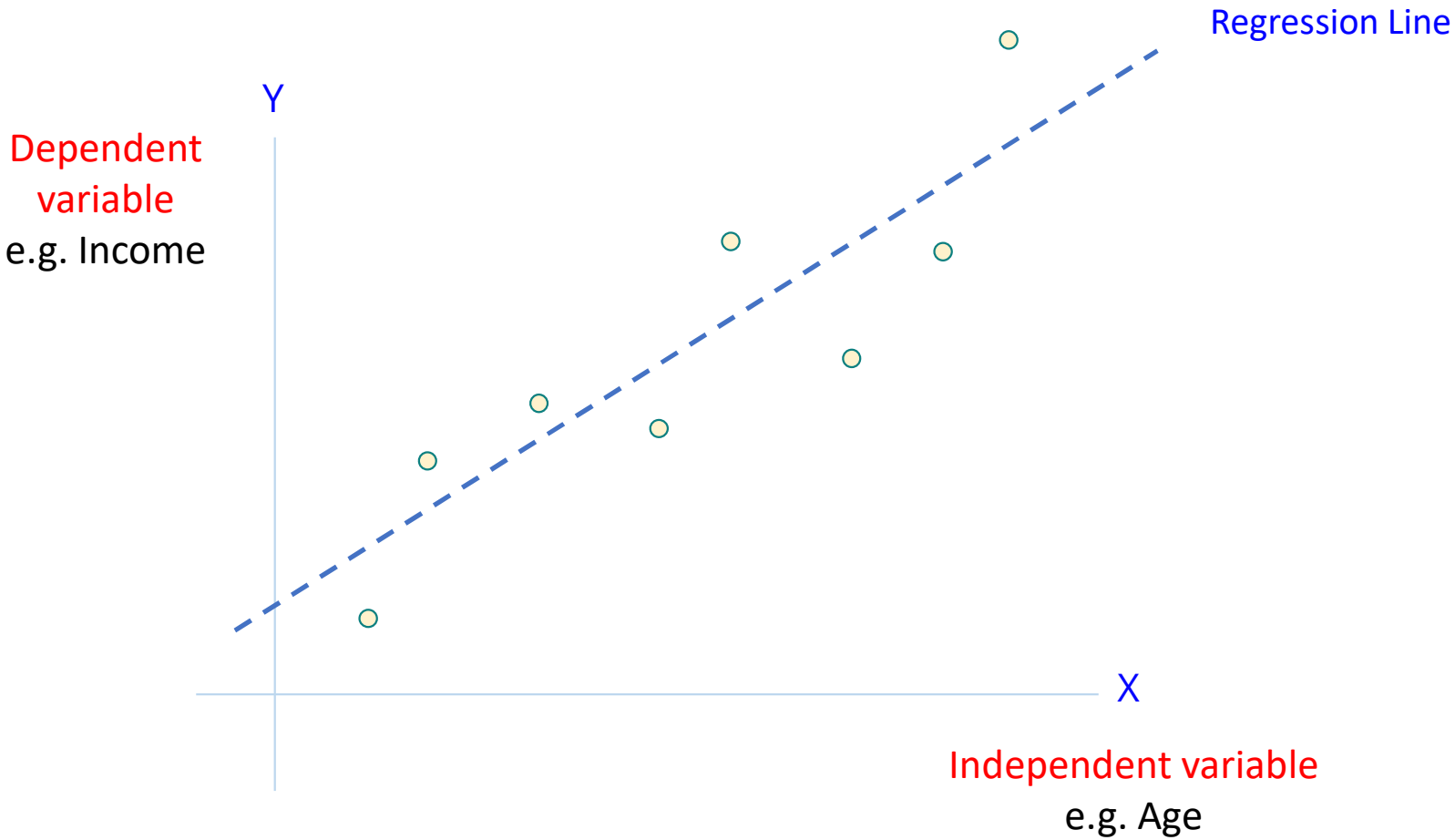
Review of Linear Equations



$b_1 = y' / x'$
is slope of the line

e.g. Y = 1.5 + 2/5 X

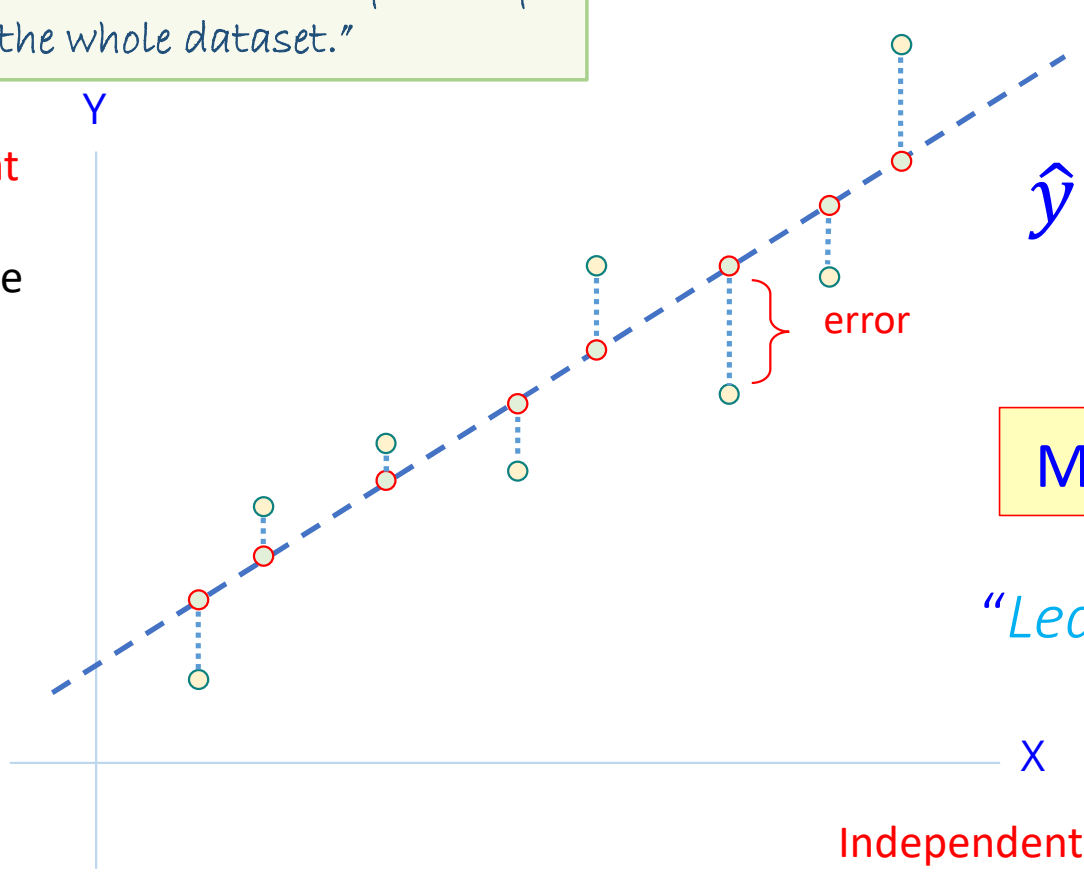
Estimation of regression line



Estimation of regression line

Goal →

"Regression analysis will find *a* and *b* to minimize the summation of error² for the whole dataset."

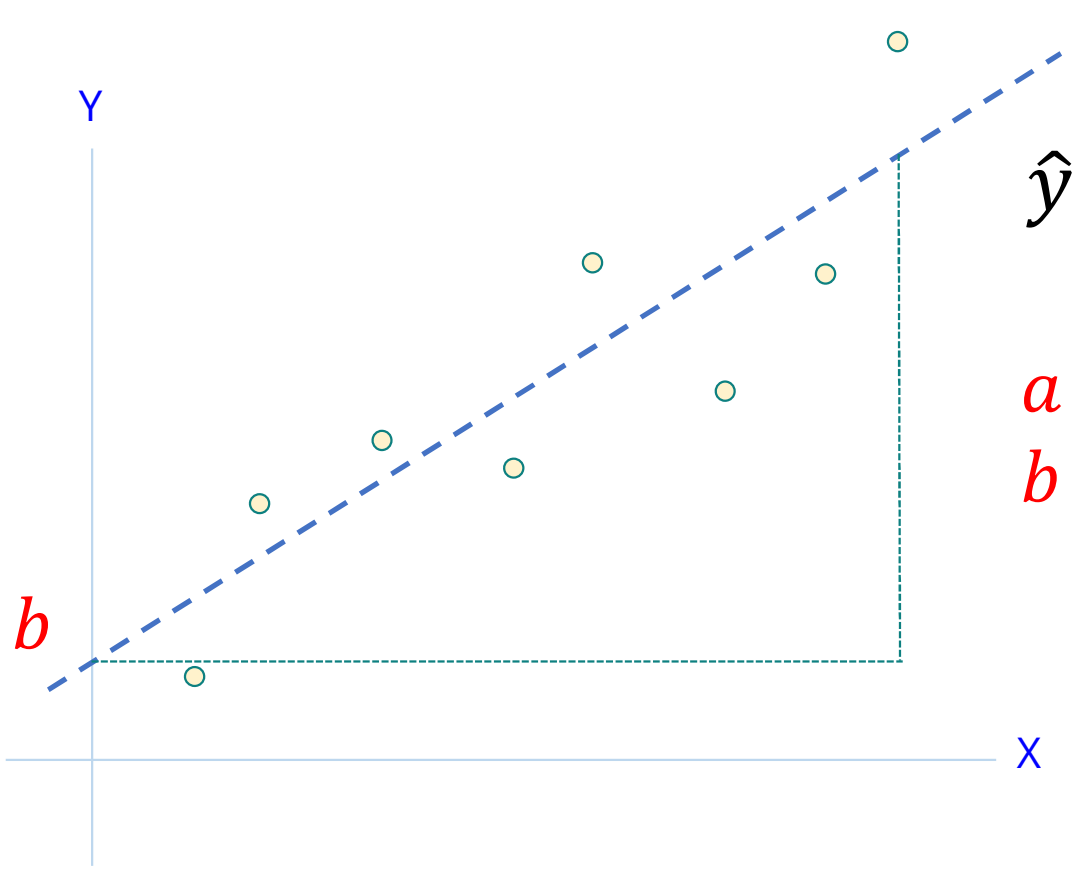


$$\hat{y} = b + ax + \varepsilon$$

Minimize $\sum error^2$

"Least square method"

Estimation of regression line



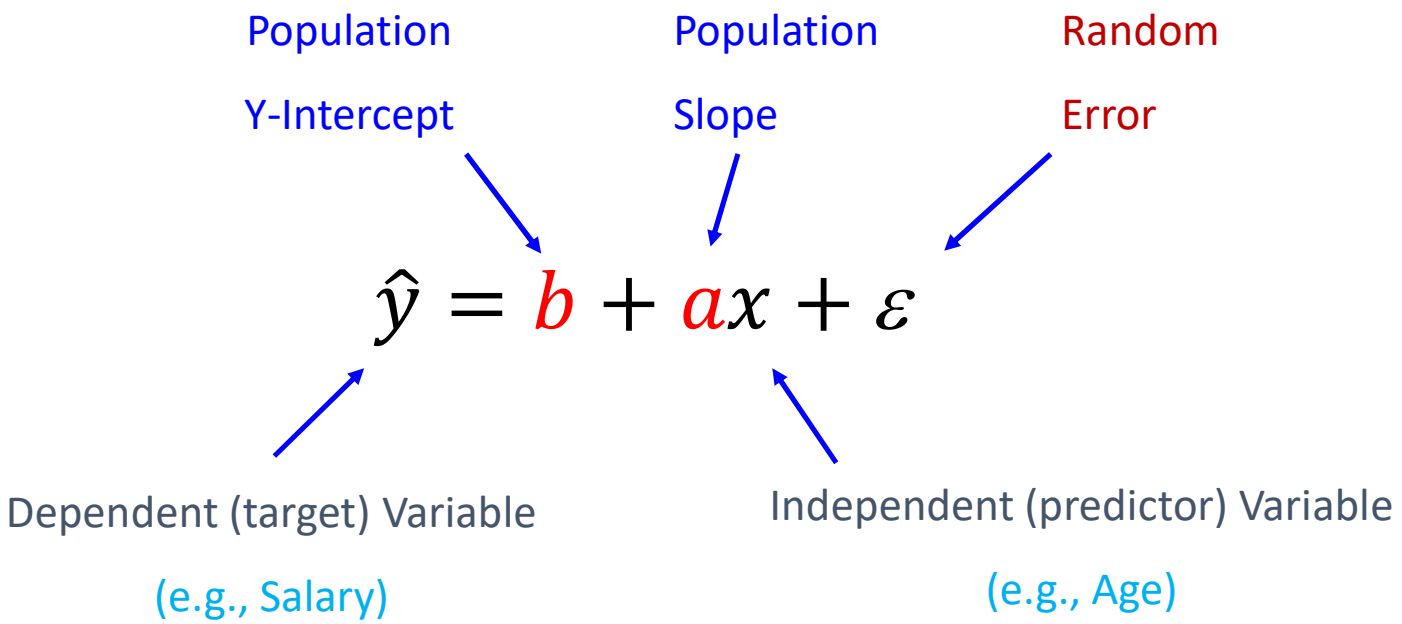
$$\hat{y} = b + ax + \varepsilon$$

a is slope of the line.

b is Y-intercept.

Linear regression model

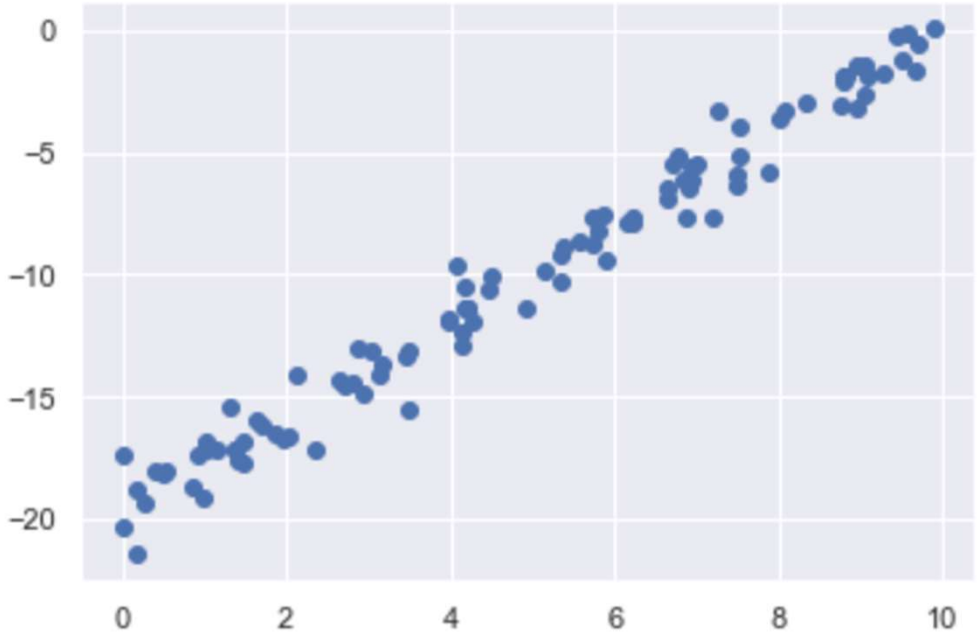
- Relationship between variables is a linear function



ตย. ความสัมพันธ์ระหว่างอายุงาน (age) กับเงินเดือน (salary)

Linear regression analysis with sklearn

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


# Generate random values as a linear function with some noise
#
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()


rng = np.random.RandomState(1)
x = 10 * rng.rand(100)
y = 2 * x - 20 + rng.randn(100)
plt.scatter(x, y);
```



Linear regression analysis with sklearn

```
# Do linear regression analysis using sklearn
#
from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=True)

# change to 2D array, required by the model.fit() (n_samples, n_features)
X = x[:, np.newaxis]
print(X.shape)

# Fit (or train) the model
model.fit(X, y)
```

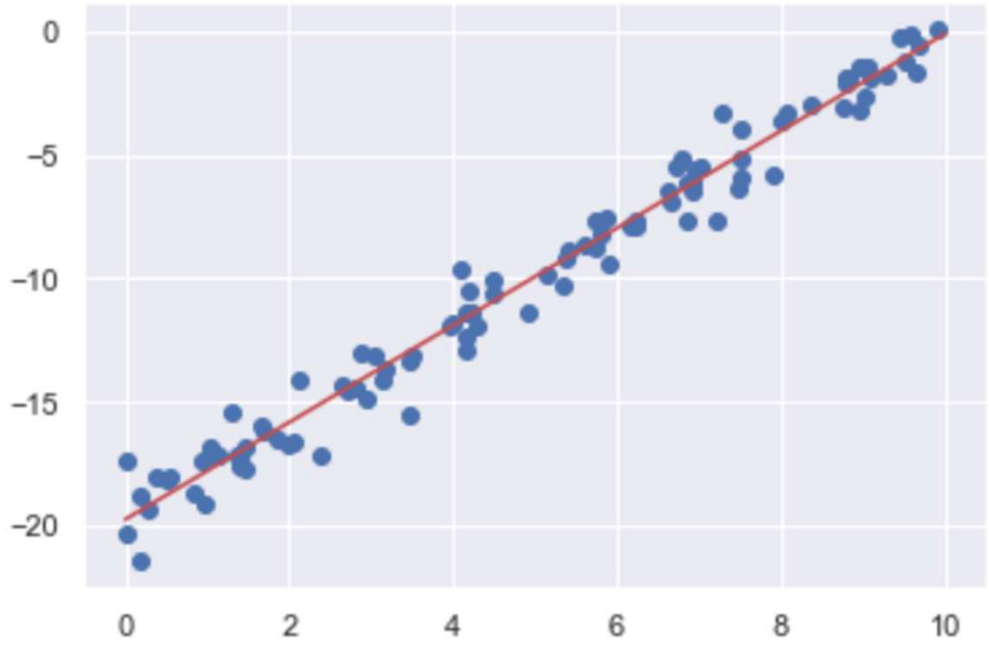
```
(100, 1)

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Important Reference

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Linear regression analysis with sklearn

```
print('Slope:', model.coef_)
print('Intercept:', model.intercept_)


# Draw the linear regression line on the chart
#
xfit = np.linspace(0, 10, 100)
yfit = model.predict(xfit[:, np.newaxis])
plt.scatter(x, y)
plt.plot(xfit, yfit, 'r')
plt.show()
```




```
Slope: [1.96849251]
Intercept: -19.763042745851095
```

Linear regression analysis with sklearn

```
# model evaluation
#
from sklearn.metrics import mean_squared_error, r2_score
rmse = mean_squared_error(y, yfit)
r2 = r2_score(y, yfit)

print('RMSE:     ', rmse)  # Root mean squared error
print('R2 s
```

```
RMSE:       56.40657644271777
R2 score:   -0.6404439133160331
```

R-square is the "percentage variation in dependent variable that is explainable by independent variables"

$$R^2 = 1 - \frac{RSS}{TSS}$$

R^2 = coefficient of determination

RSS = sum of squares of residuals

TSS = total sum of squares

sklearn.linear_model.LinearRegression

sklearn.linear_model.LinearRegression

class `sklearn.linear_model.` **`LinearRegression`**(*, *fit_intercept=True, normalize=False, copy_X=True, n_jobs=None*) [source]

Ordinary least squares Linear Regression.

LinearRegression fits a linear model with coefficients w = (w1, ..., wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

Parameters:

fit_intercept : *bool, default=True*
Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).

normalize : *bool, default=False*
This parameter is ignored when `fit_intercept` is set to False. If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm. If you wish to standardize, please use `sklearn.preprocessing.StandardScaler` before calling `fit` on an estimator with `normalize=False`.

copy_X : *bool, default=True*
If True, X will be copied; else, it may be overwritten.

n_jobs : *int, default=None*
The number of jobs to use for the computation. This will only provide speedup for n_targets > 1 and sufficient large problems. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See Glossary for more details.

Contents

- Concept of regression analysis
- Linear Regression
- Multiple Linear Regression
- Polynomial Regression
- Overfitting and Underfitting
- More advanced regressors
- Case studies

Multiple Linear Regression

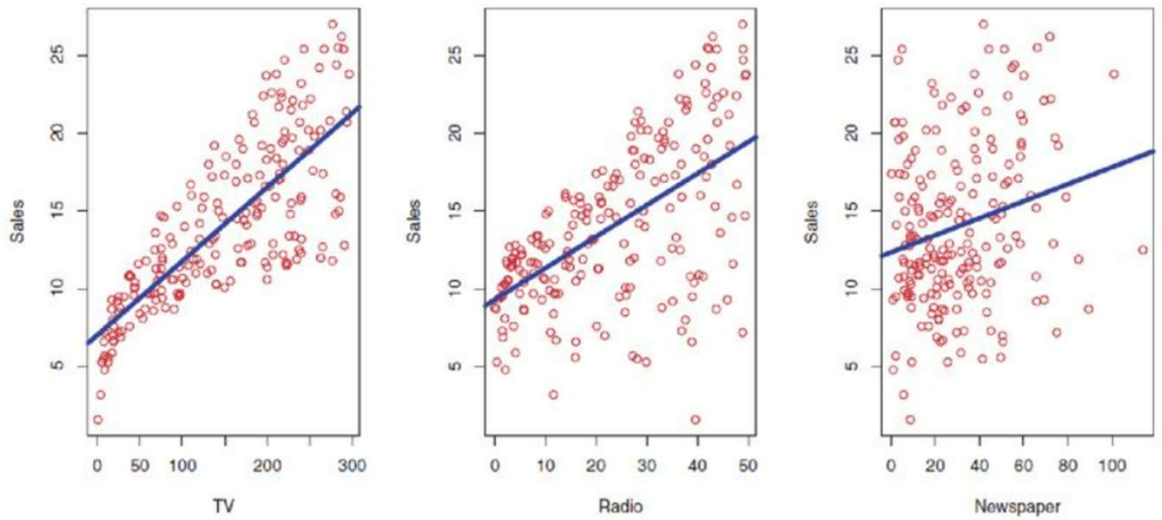
- A typical model looks like this:

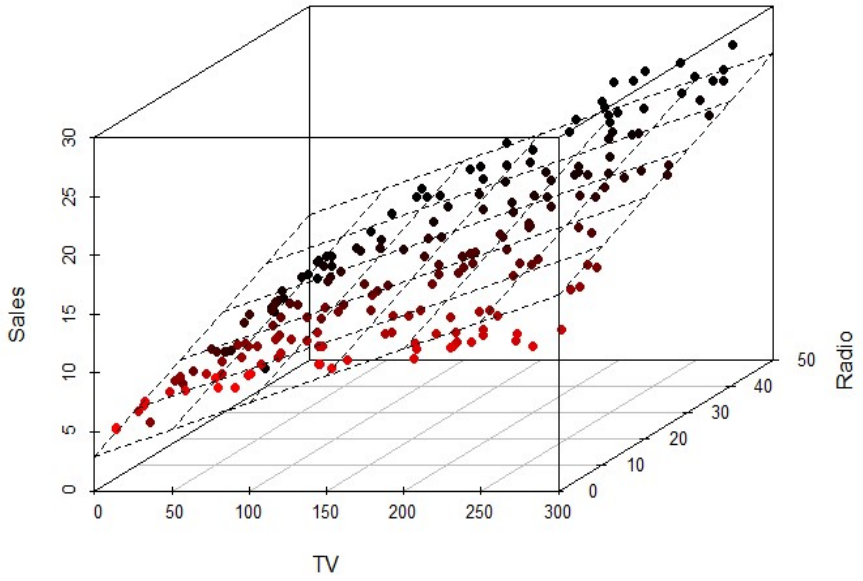
$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon,$$

- β_j means the average effect on Y of a one unit increase in X_j , *holding all other predictors fixed*.

- E.g.

$$\texttt{sales} = \beta_0 + \beta_1 \times \texttt{TV} + \beta_2 \times \texttt{radio} + \beta_3 \times \texttt{newspaper} + \epsilon.$$





Boston Housing dataset (see the .ipynb file)

```
**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.
```

Regression Analysis of Boston Housing Dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import warnings
warnings.filterwarnings('ignore')

from sklearn import datasets
boston = datasets.load_boston()

print(boston.feature_names)
target = pd.DataFrame(boston.target,
                      columns=['MEDV'])
print(type(target), target)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX'
'PTRATIO'
 'B' 'LSTAT']
<class 'pandas.core.frame.DataFrame'>      MEDV
0    24.0
1    21.6
2    34.7
3    33.4
4    36.2
..    ...
501  22.4
502  20.6
503  23.9
504  22.0
505  11.9


[506 rows x 1 columns]
```

MLR for Boston Housing Data – prepare predictors and target

```
# define the data/predictors as the pre-set feature names
# and add the target 'MEDV' into the dataframe as the 1st column
df = pd.DataFrame(boston.data, columns=boston.feature_names)
df = target.join(df)
df
```

	MEDV	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	24.0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	21.6	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	34.7	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	33.4	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	36.2	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	22.4	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67
502	20.6	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08
503	23.9	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64
504	22.0	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48
505	11.9	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88

506 rows × 14 columns

See correlation of all data

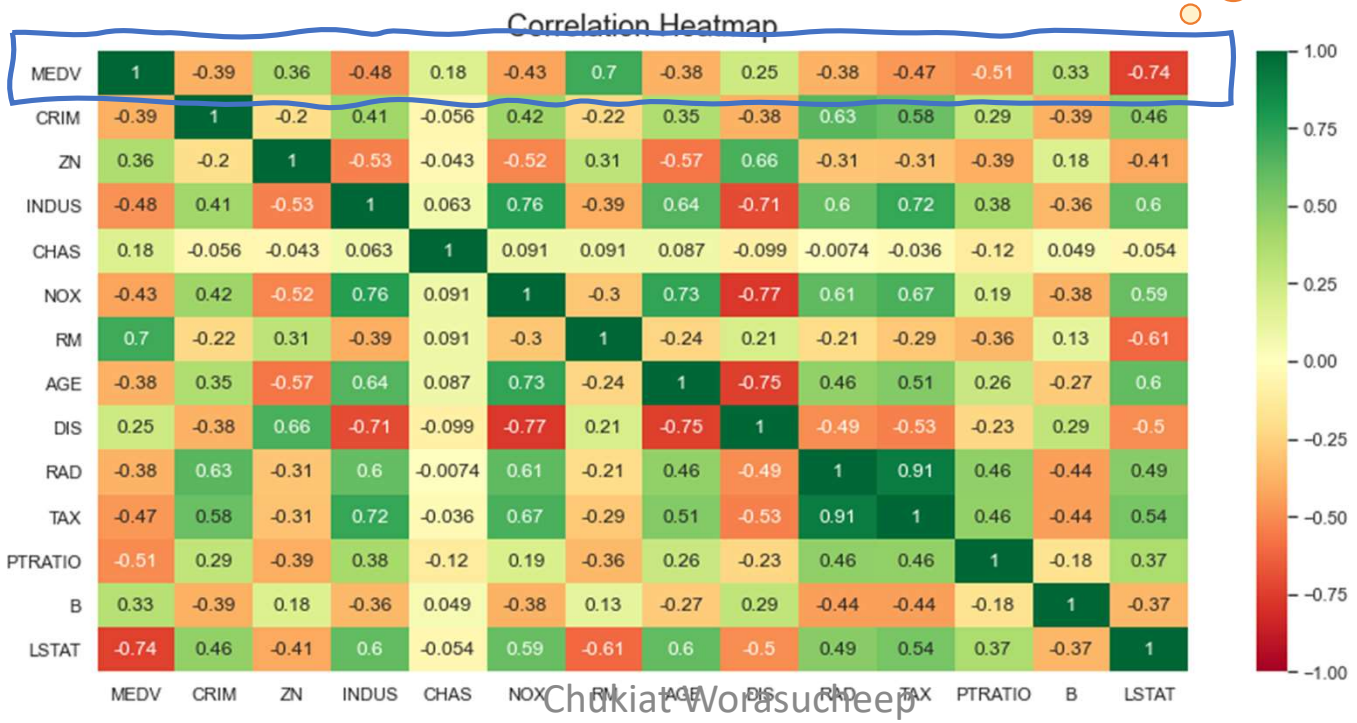
df.corr().round(4)

	MEDV	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
MEDV	1.0000	-0.3883	0.3604	-0.4837	0.1753	-0.4273	0.6954	-0.3770	0.2499	-0.3816	-0.4685	-0.5078	0.3335	-0.7377
CRIM	-0.3883	1.0000	-0.2005	0.4066	-0.0559	0.4210	-0.2192	0.3527	-0.3797	0.6255	0.5828	0.2899	-0.3851	0.4556
ZN	0.3604	-0.2005	1.0000	-0.5338	-0.0427	-0.5166	0.3120	-0.5695	0.6644	-0.3119	-0.3146	-0.3917	0.1755	-0.4130
INDUS	-0.4837	0.4066	-0.5338	1.0000	0.0629	0.7637	-0.3917	0.6448	-0.7080	0.5951	0.7208	0.3832	-0.3570	0.6038
CHAS	0.1753	-0.0559	-0.0427	0.0629	1.0000	0.0912	0.0913	0.0865	-0.0992	-0.0074	-0.0356	-0.1215	0.0488	-0.0539
NOX	-0.4273	0.4210	-0.5166	0.7637	0.0912	1.0000	-0.3022	0.7315	-0.7692	0.6114	0.6680	0.1889	-0.3801	0.5909
RM	0.6954	-0.2192	0.3120	-0.3917	0.0913	-0.3022	1.0000	-0.2403	0.2052	-0.2098	-0.2920	-0.3555	0.1281	-0.6138
AGE	-0.3770	0.3527	-0.5695	0.6448	0.0865	0.7315	-0.2403	1.0000	-0.7479	0.4560	0.5065	0.2615	-0.2735	0.6023
DIS	0.2499	-0.3797	0.6644	-0.7080	-0.0992	-0.7692	0.2052	-0.7479	1.0000	-0.4946	-0.5344	-0.2325	0.2915	-0.4970
RAD	-0.3816	0.6255	-0.3119	0.5951	-0.0074	0.6114	-0.2098	0.4560	-0.4946	1.0000	0.9102	0.4647	-0.4444	0.4887
TAX	-0.4685	0.5828	-0.3146	0.7208	-0.0356	0.6680	-0.2920	0.5065	-0.5344	0.9102	1.0000	0.4609	-0.4418	0.5440
PTRATIO	-0.5078	0.2899	-0.3917	0.3832	-0.1215	0.1889	-0.3555	0.2615	-0.2325	0.4647	0.4609	1.0000	-0.1774	0.3740
B	0.3335	-0.3851	0.1755	-0.3570	0.0488	-0.3801	0.1281	-0.2735	0.2915	-0.4444	-0.4418	-0.1774	1.0000	-0.3661
LSTAT	-0.7377	0.4556	-0.4130	0.6038	-0.0539	0.5909	-0.6138	0.6023	-0.4970	0.4887	0.5440	0.3740	-0.3661	1.0000

See correlation heatmap for better understanding

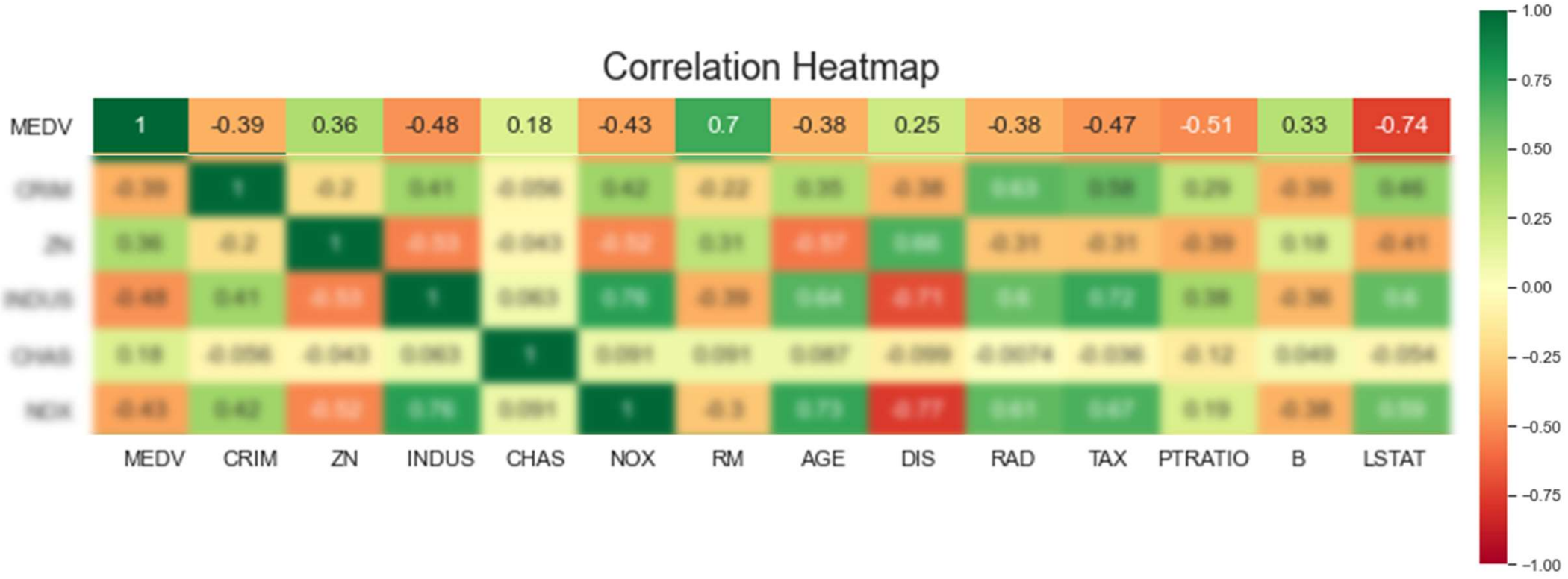
```
plt.figure(figsize=(15, 7))
# set the range of values to be displayed on the colormap from -1 to 1.
# and set the annotation to True to display the correlation values on the heatmap.
hm = sns.heatmap(df.corr(), vmin=-1, vmax=1, annot=True, cmap='RdYlGn')
hm.set_title('Correlation Heatmap', fontdict={'fontsize':18}, pad=9)
```

Now, let's choose predictors from this correlation coefficient table.



Choose two predictors with the highest correlation coefficients

- By using correlation coefficient, we choose two variables:
  - RM — the average number of rooms
  - LSTAT — percentage of lower status of the population, to predict values of MEDV.



Two common ways to do regression analysis with Python

1. statmodels's *OLS* model
   - "Ordinary Least Squares“
   - 'Least Squares’ … minimize the square of distance from the regression line.
   - *statsmodels does not add a constant by default.*

2. Sklearn’s linear_model.*LinearRegression* model
   - .fit()
   - .predict()

Run OLS and interpret the results

R-square is the "percentage variation in dependent that is explained by independent variables"

```
import statsmodels.api as sm


X = df[['RM', 'LSTAT']] ## Without a constant
y = target['MEDV']


# create stat's Ordinary Least Squares model
model = sm.OLS(y, X).fit()


# make the predictions with the model
predictions = model.predict(X)


# display the statistics
model.summary()
```

Adj. R-squared is the modified version of R-squared which is adjusted for the number of variables in the regression.

OLS Regression Results

Dep. Variable:	MEDV	**R-squared (uncentered):**	0.948
Model:	OLS	**Adj. R-squared (uncentered):**	0.948
Method:	Least Squares	**F-statistic:**	4637.
Date:	Fri, 24 Jun 2022	**Prob (F-statistic):**	0.00
Time:	10:57:56	**Log-Likelihood:**	-1582.9
No. Observations:	506	**AIC:**	3170.
Df Residuals:	504	**BIC:**	3178.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
RM	4.9069	0.070	69.906	0.000	4.769	5.045
LSTAT	-0.6557	0.031	-21.458	0.000	-0.716	-0.596

Omnibus:	145.153	**Durbin-Watson:**	0.834
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	442.157
Skew:	1.351	**Prob(JB):**	9.70e-97
Kurtosis:	6.698	**Cond. No.**	4.72

Run OLS and interpret the results

OLS Regression Results

Dep. Variable:	MEDV	**R-squared (uncentered):**	0.948
Model:	OLS	**Adj. R-squared (uncentered):**	0.948
Method:	Least Squares	**F-statistic:**	4637.
Date:	Fri, 24 Jun 2022	**Prob (F-statistic):**	0.00
Time:	10:57:56	**Log-Likelihood:**	-1582.9
No. Observations:	506	**AIC:**	3170.
Df Residuals:	504	**BIC:**	3178.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
RM	4.9069	0.070	69.906	0.000	4.769	5.045
LSTAT	-0.6557	0.031	-21.458	0.000	-0.716	-0.596

Omnibus:	145.153	**Durbin-Watson:**	0.834
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	442.157
Skew:	1.351	**Prob(JB):**	9.70e-97
Kurtosis:	6.698	**Cond. No.**	4.72

Prob(F-Statistic) tells the overall significance level of all the variables together. That the probability is close to zero implies that the overall regression is meaningful.

AIC & BIC Akaike's Information Criteria and Bayesian information criteria indicate model performance and complexity, respectively. *Lower values are better*.

Interpretation of results

- Regression analysis is a form of *inferential* statistics (สถิติเชิงอนุมาน).
  - The *p-values* help determine whether the relationships that you observe in your sample also exist in the larger population.
  - *If the p-value for a variable is less than your significance level (default 0.05 in OLS), include that variable in your regression model, meaning that the variable is significant.*

- Regression equation:

MEDV = 4.9069 x RM – 0.6557 LSTAT

OLS Regression Results

Dep. Variable:	MEDV	**R-squared (uncentered):**	0.948
Model:	OLS	**Adj. R-squared (uncentered):**	0.948
Method:	Least Squares	**F-statistic:**	4637.
Date:	Fri, 24 Jun 2022	**Prob (F-statistic):**	0.00
Time:	10:57:56	**Log-Likelihood:**	-1582.9
No. Observations:	506	**AIC:**	3170.
Df Residuals:	504	**BIC:**	3178.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
RM	4.9069	0.070	69.906	0.000	4.769	5.045
LSTAT	-0.6557	0.031	-21.458	0.000	-0.716	-0.596

Omnibus:	145.153	**Durbin-Watson:**	0.834
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	442.157
Skew:	1.351	**Prob(JB):**	9.70e-97
Kurtosis:	6.698	**Cond. No.**	4.72

Ref: https://statisticsbyjim.com/regression/interpret-coefficients-p-values-regression/

Run OLS again with Y-intercept

```
X = df[['RM', 'LSTAT']]      # Use both RM and LSTAT as predictors
X = sm.add_constant(X)  # Also, add an intercept (B0) to our model
y = target['MEDV']


model = sm.OLS(y, X).fit()
predictions = model.predict(X)
print(model.summary(), '\n')
print(predictions[:5].round(3))
```

- *P-value of the constant is higher than sig. level (0.05)!!*

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   MEDV   R-squared:                       0.639
Model:                            OLS   Adj. R-squared:                  0.637
Method:                 Least Squares   F-statistic:                     444.3
Date:                Fri, 24 Jun 2022   Prob (F-statistic):          7.01e-112
Time:                        10:57:56   Log-Likelihood:                -1582.8
No. Observations:                 506   AIC:                             3172.
Df Residuals:                     503   BIC:                             3184.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -1.3583      3.173     -0.428      0.669      -7.592       4.875
RM             5.0948      0.444     11.463      0.000       4.222       5.968
LSTAT         -0.6424      0.044    -14.689      0.000      -0.728      -0.556
==============================================================================
Omnibus:                      145.712   Durbin-Watson:                   0.834
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              457.690
Skew:                           1.343   Prob(JB):                    4.11e-100
Kurtosis:                       6.807   Cond. No.                         202.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

3. Use sklearn's LinearRegression()

```
from sklearn import linear_model
from sklearn import datasets

data = datasets.load_boston()
df   = pd.DataFrame(data.data, columns=data.feature_names)
target = pd.DataFrame(data.target, columns=['MEDV'])

lm = linear_model.LinearRegression(fit_intercept=True)  # default is True

results = lm.fit(X, y)          # fit the model to the training data
predictions = lm.predict(X) # predict to see accuracy or the training data
print(predictions[0:18])      # see only the first 18 predicted values
```

```
[28.94101368 25.48420566 32.65907477 32.40652    31.63040699 28.05452701
 21.28707846 17.78559653  8.10469338 18.24650673 17.99496223 20.73221309
 18.5534842  23.64474107 23.10895823 22.9239452  24.65257604 19.73611045]
```

https://stackoverflow.com/questions/27928275/find-p-value-significance-in-scikit-learn-linearregression

Interpret the results

```
R2 = lm.score(X,y)

print('R square:', R2)

print('Slope:', results.coef_)

print('Intercept:', results.intercept_)

print('No. of predictor(s):', results.n_features_in_)
```

```
R square: 0.6385616062603403
Slope: [ 0.         5.09478798 -0.64235833]
Intercept: -1.3582728118744853
No. of predictor(s): 3
```

OLS Regression Results

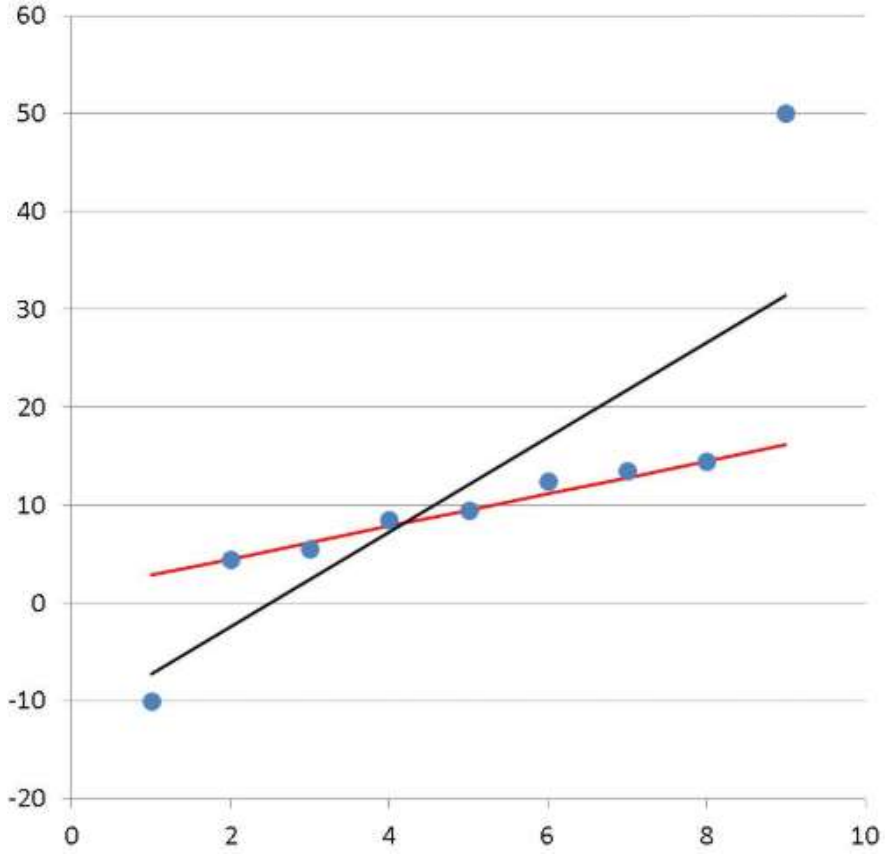
Dep. Variable:	MEDV	R-squared:	0.639
Model:	OLS	Adj. R-squared:	0.637
Method:	Least Squares	F-statistic:	444.3
Date:	Fri, 24 Jun 2022	Prob (F-statistic):	7.01e-112
Time:	10:57:56	Log-Likelihood:	-1582.8
No. Observations:	506	AIC:	3172.
Df Residuals:	503	BIC:	3184.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.3583	3.173	-0.428	0.669	-7.592	4.875
RM	5.0948	0.444	11.463	0.000	4.222	5.968
LSTAT	-0.6424	0.044	-14.689	0.000	-0.728	-0.556

Omnibus:	145.712	Durbin-Watson:	0.834
Prob(Omnibus):	0.000	Jarque-Bera (JB):	457.690
Skew:	1.343	Prob(JB):	4.11e-100
Kurtosis:	6.807	Cond. No.	202.

Limitations of Linear Regression

- Linearity
- Outliers
- Interdependence of variables (*multicollinearity*)

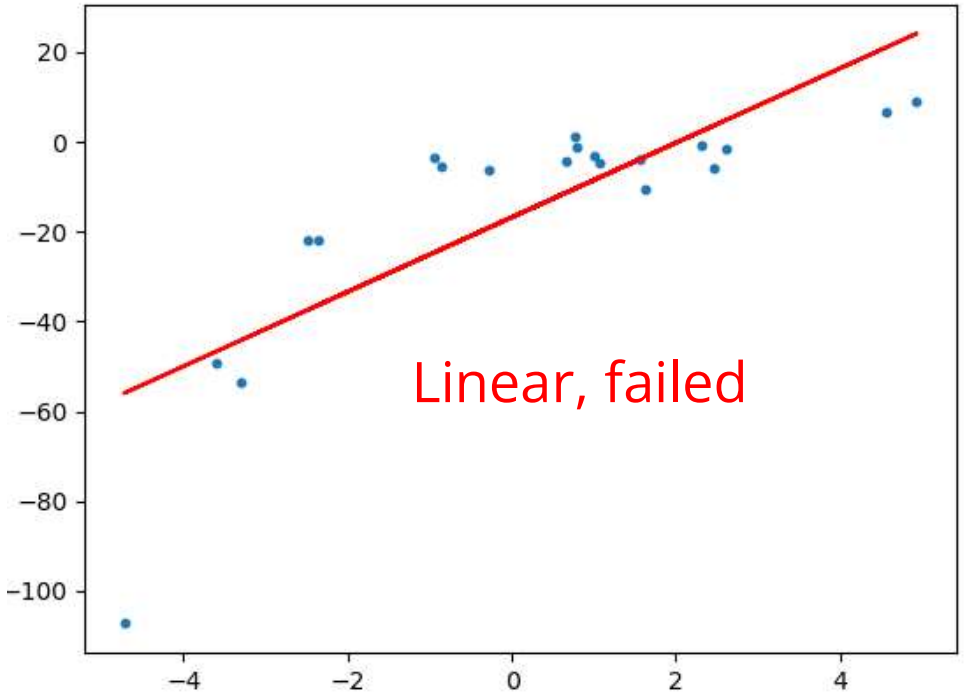


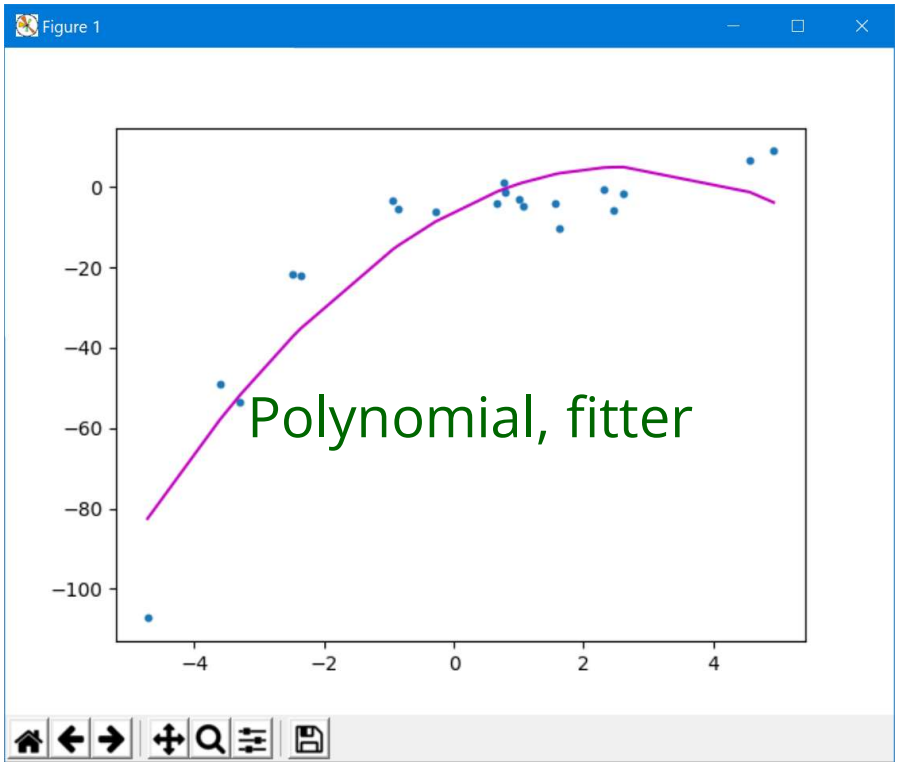
Contents

- Concept of regression analysis
- Linear Regression
- Multiple Linear Regression
- Polynomial Regression
- Overfitting and Underfitting
- More advanced regressors
- Case studies

Polynomial Regression

- For data which can’t be estimated with linear relationship.



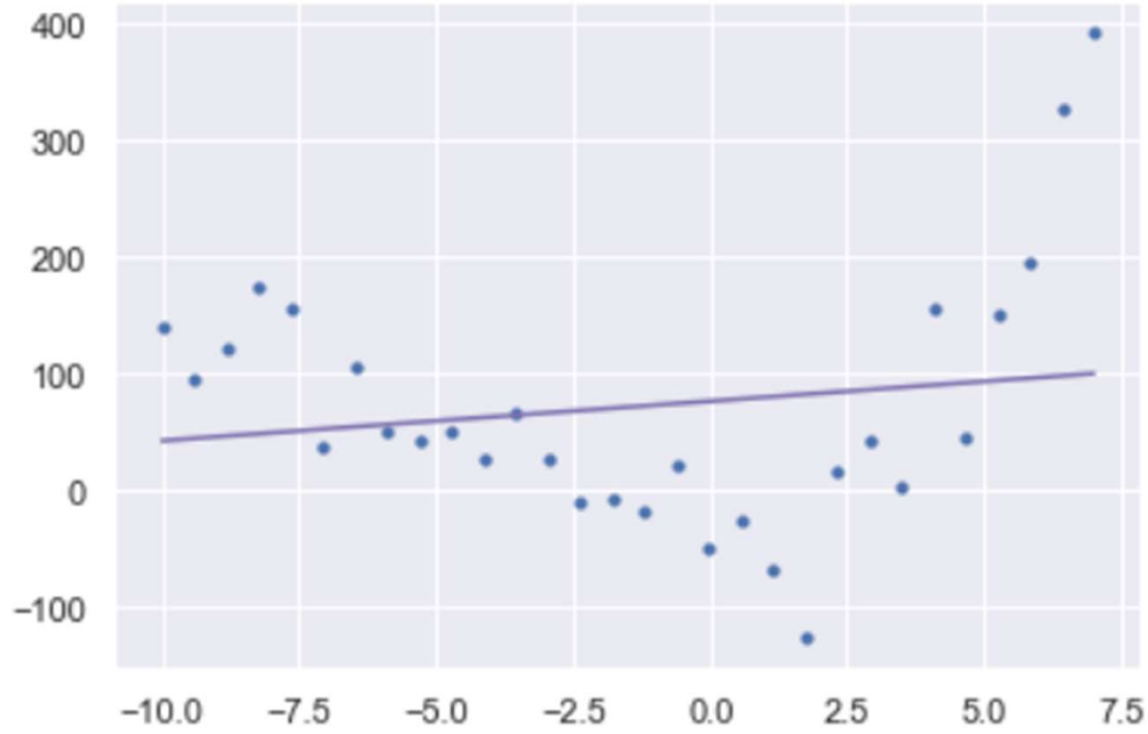


Example: Polynomial Regression (see the .ipynb file)

```
# Polynomial Regression
# https://towardsdatascience.com/polynomial-regression-bbe8b9d97491
#
import operator
import numpy as np
import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures

np.random.seed(0)
x = np.linspace(-10, 7, 30)
y = (0.4 * x**3) + 5 * (x**2) - x + np.random.normal(-40, 40, 30)

# transforming the data to include another axis
x = x[:, np.newaxis]
y = y[:, np.newaxis]
```

If try using linear regression model...

```
RMSE = 106.09937177295845
R2 = 0.025615244877509102
```



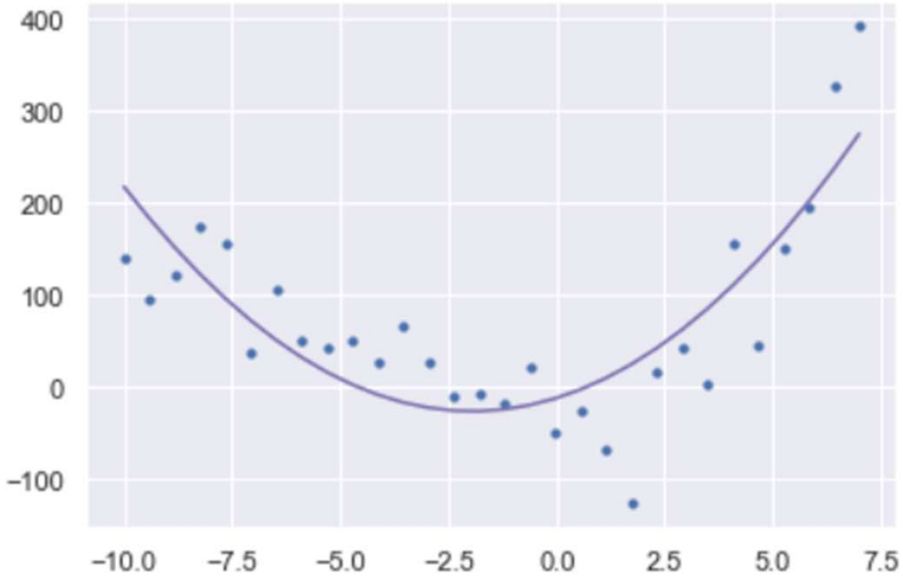
Example: Polynomial Regression

```
pf= PolynomialFeatures(degree=2)
x_poly = pf.fit_transform(x)

model = LinearRegression()
model.fit(x_poly, y)
y_poly_pred = model.predict(x_poly)

rmse = np.sqrt(mean_squared_error(y, y_poly_pred))
r2 = r2_score(y, y_poly_pred)
print('RMSE is', rmse)
print('R2 is', r2)

plt.scatter(x, y, s=10)
# sort the values of x before line plot
sort_axis = operator.itemgetter(0)
sorted_zip = sorted(zip(x,y_poly_pred), key=sort_axis)
x, y_poly_pred = zip(*sorted_zip)
plt.plot(x, y_poly_pred, color='m')
plt.show()
```

Example: Polynomial Regression: Output

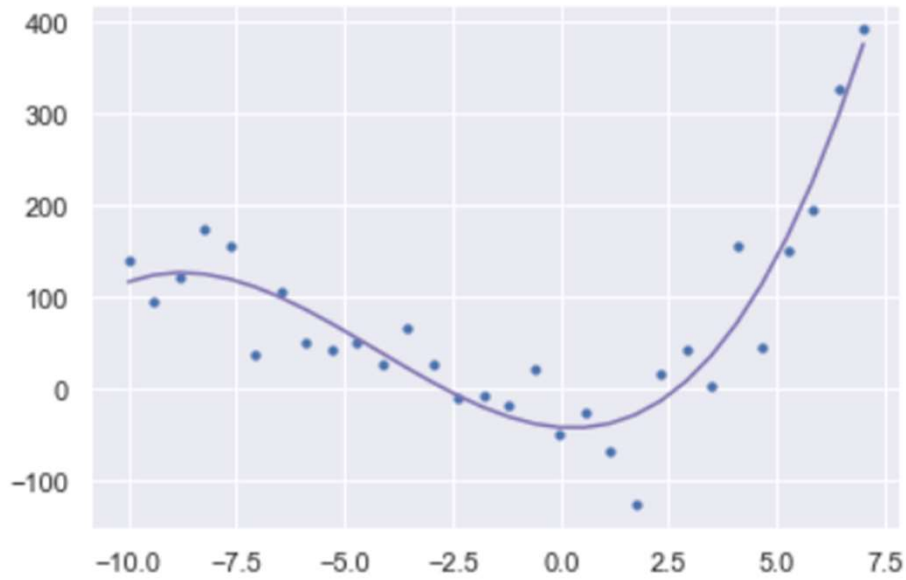
degree = 2

```
RMSE = 61.325185386369434
R2 = 0.6744763134609137
```



degree = 3

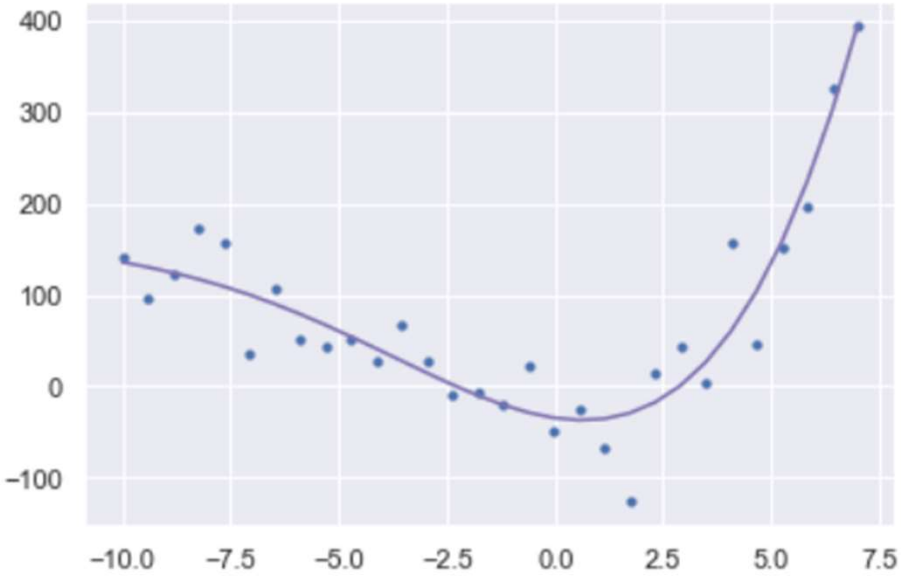
```
RMSE = 39.79425460213335
R2 = 0.8629289927465575
```



Example: Polynomial Regression: Output

degree = 4

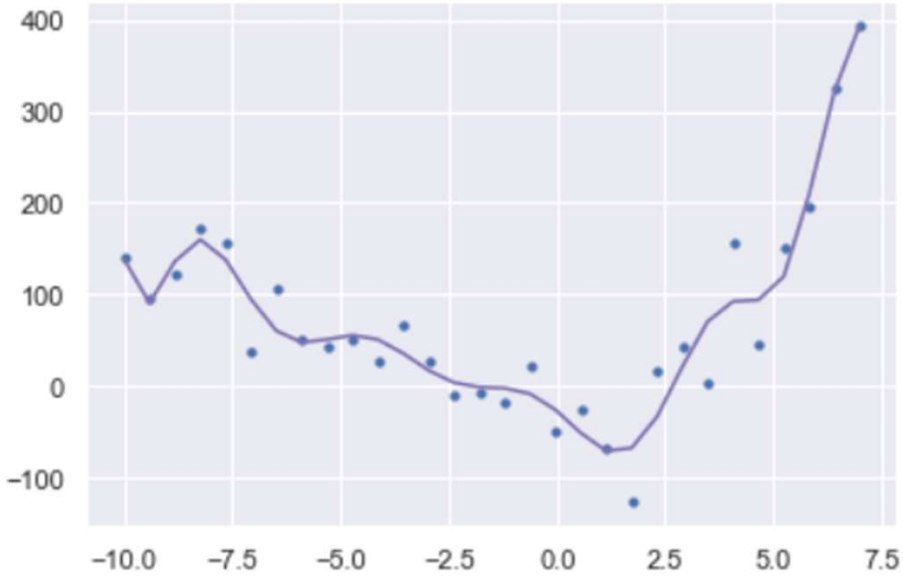
```
RMSE = 38.813421093582704
R2 = 0.8696026685450101
```



degree = 15

```
RMSE = 31.02820145763909
R2 = 0.9166667996701975
```

Looks good?

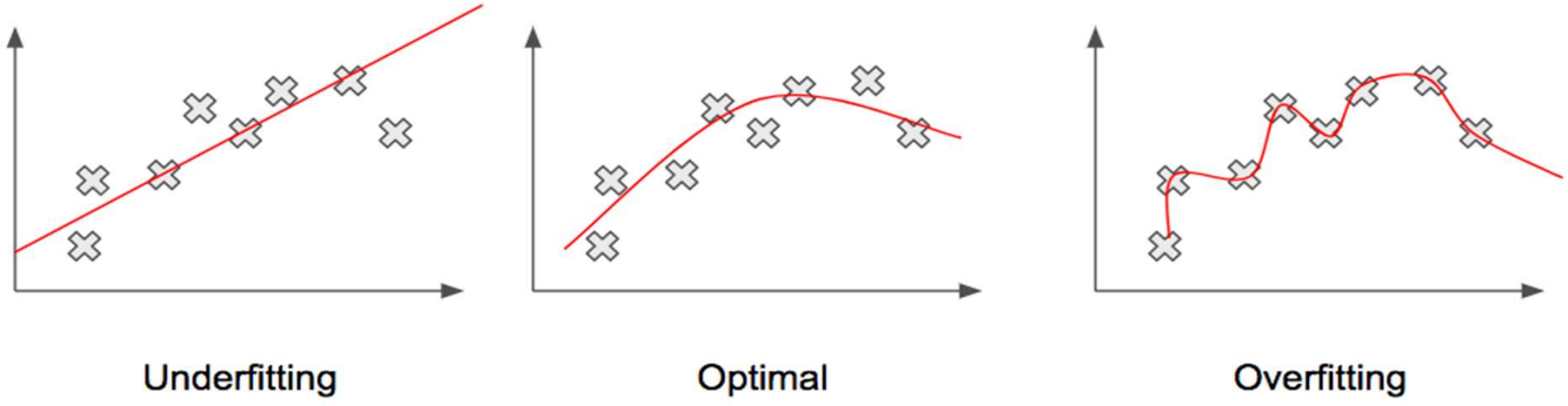


Contents

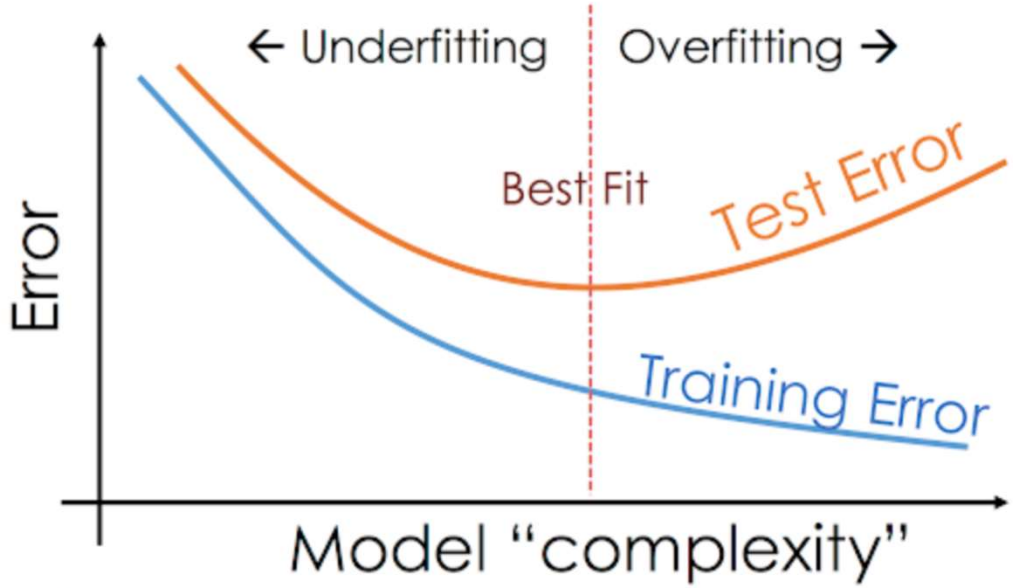
- Concept of regression analysis
- Linear Regression
- Multiple Linear Regression
- Polynomial Regression
- Overfitting and Underfitting
- More advanced regressors
- Case studies

Overfitting and Underfitting

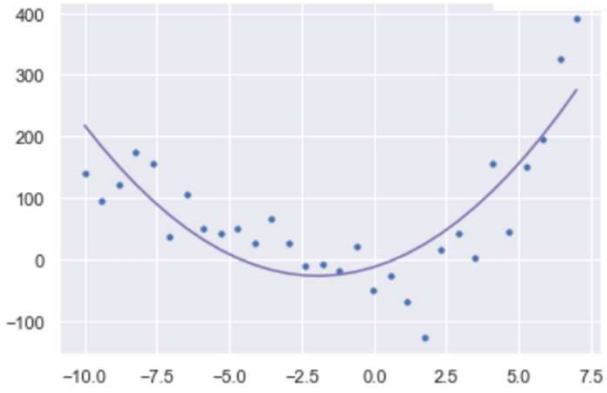
- In machine learning, learning from training data is a kind of *inductive learning* (แนวอุปนัย)
- A good ML algorithm should be good at induction (or *generalization*) from training examples.
- Major problems of induction are *overfitting* and *underfitting*
- *Overfitting* หมายถึงสถานการณ์ที่เส้นโค้งประมาณค่าจะเหมาะกับข้อมูลฝึกฝน (training data) ทั้งหมด (ซึ่งรวมถึง noise) มากเกินไป และมีประสิทธิภาพต่ำในการพยากรณ์ข้อมูลทดสอบ (ที่ยังไม่พบ) (less generalization ability).
- *Underfitting* หมายถึงสถานการณ์ที่โมเดลไม่สามารถเรียนรู้หรือจับหาทิศทางแนวโน้มของข้อมูลได้



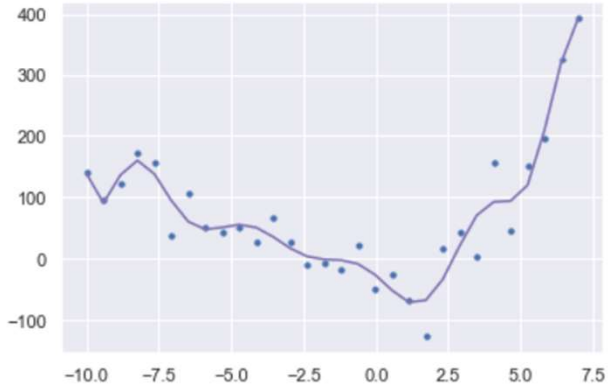
Model complexity vs overfitting or underfitting



RMSE = 61.325185386369434
R2 = 0.6744763134609137



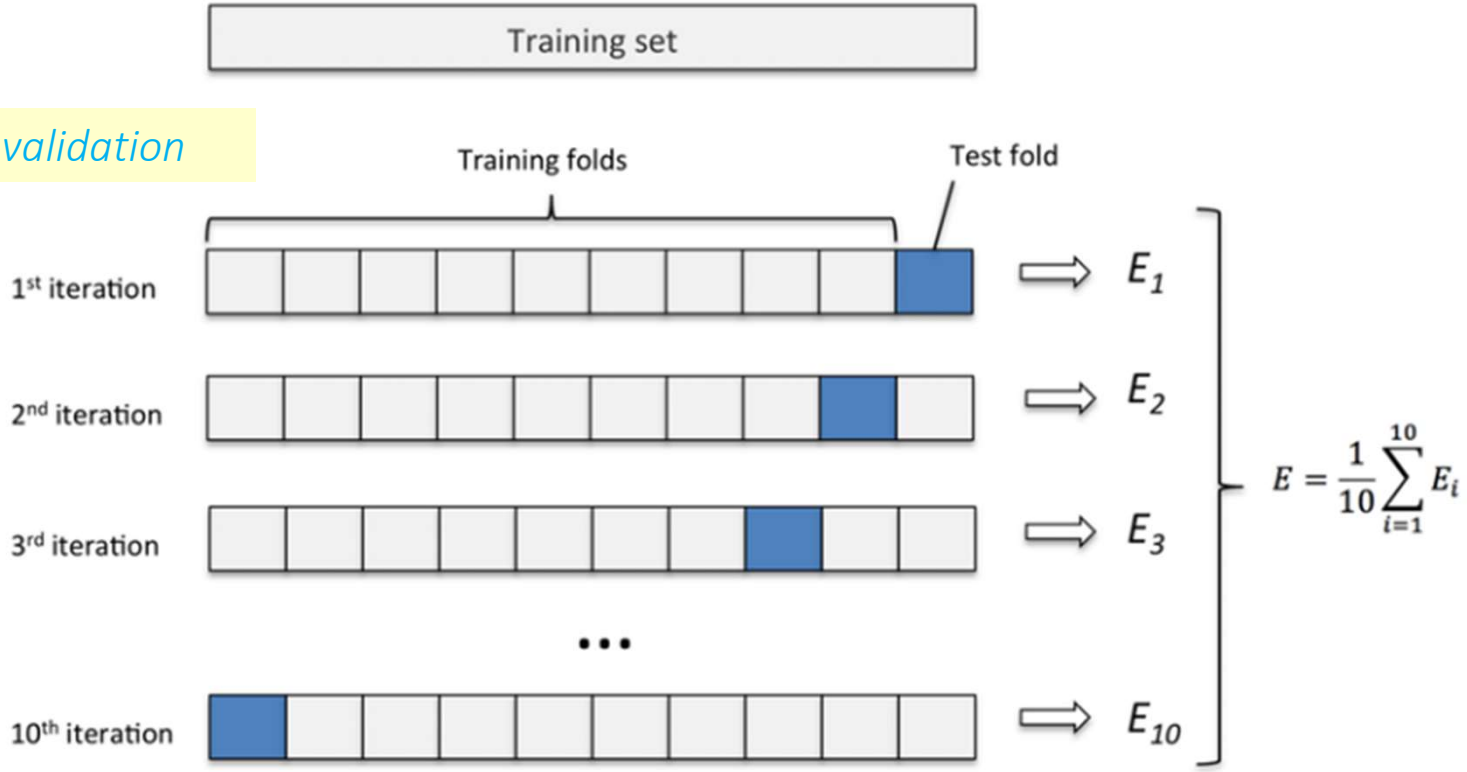
RMSE = 31.02820145763909
R2 = 0.9166667996701975



Handling of overfitting and Underfitting

- One efficient and popular method to handle overfitting for *supervised learning (classification and regression)* is a resampling technique by using *"k-fold cross validation"*

10-fold cross validation



sklearn's KFold()

- A good tutorial: https://machinelearningmastery.com/k-fold-cross-validation/

sklearn.model_selection.KFold

class `sklearn.model_selection.` **`KFold`**(*n_splits=5, *, shuffle=False, random_state=None*) [source]

K-Folds cross-validator

Provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds (without shuffling by default).

Each fold is then used once as a validation while the k - 1 remaining folds form the training set.

Read more in the User Guide.

Parameters:

n_splits : *int, default=5*

Number of folds. Must be at least 2.

Changed in version 0.22: `n_splits` default value changed from 3 to 5.

shuffle : *bool, default=False*

Whether to shuffle the data before splitting into batches. Note that the samples within each split will not be shuffled.

random_state : *int or RandomState instance, default=None*

When `shuffle` is True, `random_state` affects the ordering of the indices, which controls the randomness of each fold. Otherwise, this parameter has no effect. Pass an int for reproducible output across multiple function calls. See Glossary.

Contents

- Concept of regression analysis
- Linear Regression
- Multiple Linear Regression
- Polynomial Regression
- Overfitting and Underfitting
- More advanced regressors
- Case studies

More advance regressors

- RandomForestRegressor()

sklearn.ensemble.RandomForestRegressor

class **sklearn.ensemble.RandomForestRegressor**(*n_estimators=100, *, criterion='squared_error', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=1.0, max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, ccp_alpha=0.0, max_samples=None*) [source]

A random forest regressor.

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the `max_samples` parameter if `bootstrap=True` (default), otherwise the whole dataset is used to build each tree.

- XGBRegressor

Scikit-Learn Wrapper interface for XGBoost.

class **xgboost.XGBRegressor**(**, objective='reg:squarederror', **kwargs*)

Bases: `XGBModel`, `RegressorMixin`

Implementation of the scikit-learn API for XGBoost regression.

Parameters:

- **n_estimators** (*int*) – Number of gradient boosted trees. Equivalent to number of boosting rounds.
- **max_depth** (*Optional[int]*) – Maximum tree depth for base learners.
- **max_leaves** – Maximum number of leaves; 0 indicates no limit.
- **max_bin** – If using histogram-based algorithm, maximum number of bins per feature
- **grow_policy** – Tree growing policy. 0: favor splitting at nodes closest to the node, i.e. grow depth-wise. 1: favor splitting at nodes with highest loss change.
- **learning_rate** (*Optional[float]*) – Boosting learning rate (xgb's "eta")

More advance regressors

- SVR() – Support Vector Regressor
- MLPRegressor

sklearn.svm.SVR

class **sklearn.svm.SVR**(**, kernel='rbf', degree=3, gamma='scale', coef0=0.0, tol=0.001, C=1.0, epsilon=0.1, shrinking=True, cache_size=200, verbose=False, max_iter=-1*) [source]

Epsilon-Support Vector Regression.

The free parameters in the model are C and epsilon.

sklearn.neural_network.MLPRegressor

class **sklearn.neural_network.MLPRegressor**(*hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000*) [source]

Multi-layer Perceptron regressor.

This model optimizes the squared error using LBFGS or stochastic gradient descent.

Contents

- Concept of regression analysis
- Linear Regression
- Multiple Linear Regression
- Polynomial Regression
- Overfitting and Underfitting
- More advanced regressors
- Case studies

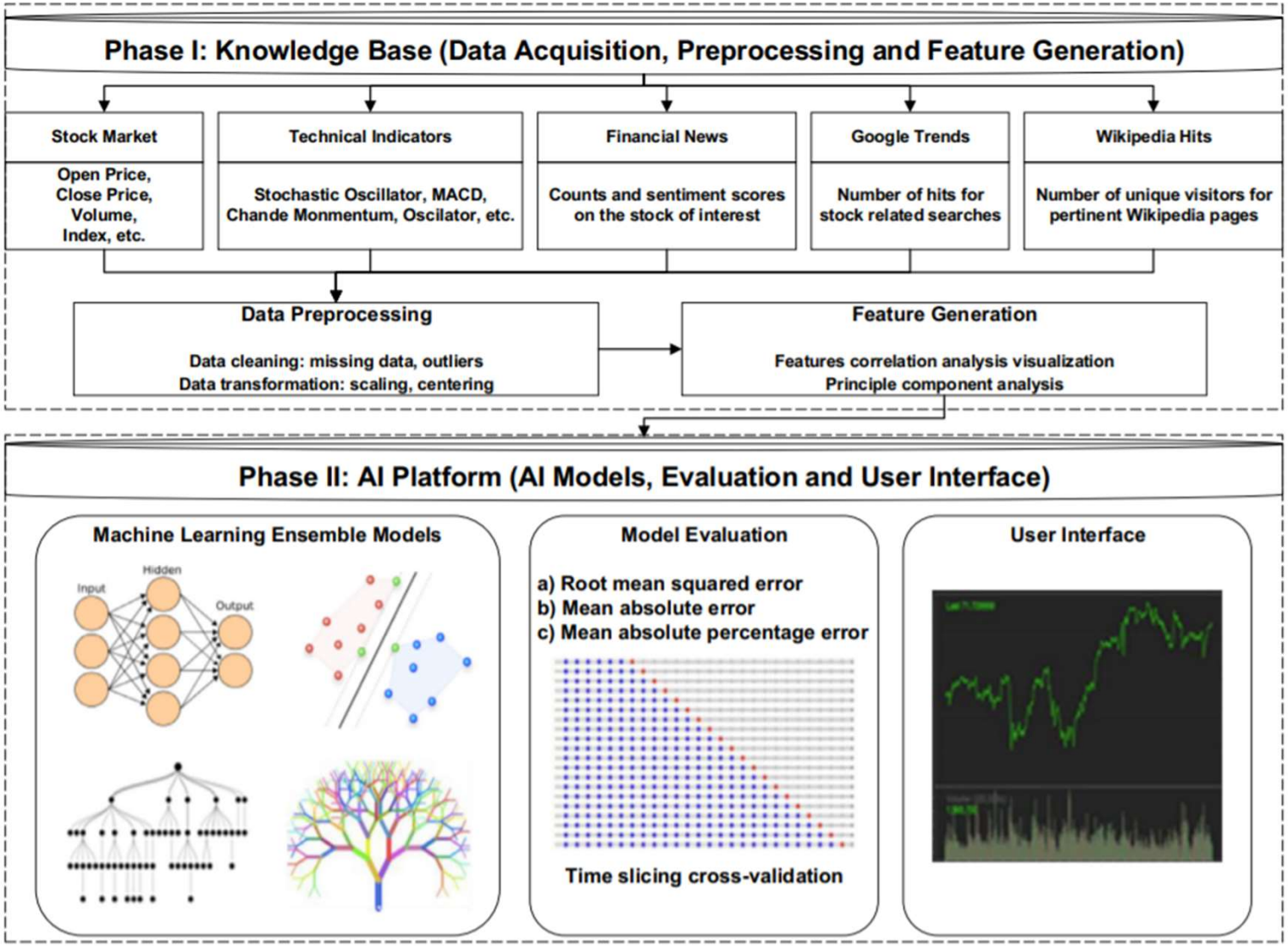
Case Study 1: Classification for stock trend prediction

- Bin Weng, Lin Lu, et al. 2021. Predicting short-term stock prices using ensemble methods and online data sources. Expert Systems With Applications 112, pg. 258–273.
- With the ubiquity of the Internet, platforms such as: Google, Wikipedia and the like can provide insights pertaining to firms' financial performance as well as capture the collective interest of traders through search trends, number of web page visitors and/or financial news sentiment. Information emanating from these platforms can significantly affect, or be affected by, changes in the stock market. The overarching goal of this paper is to develop a financial expert system that incorporates these features to predict short term stock prices. Our expert system is comprised of two main modules: a knowledge base and an artificial intelligence (AI) platform.
- The "knowledge base" for our expert system captures: (a) historical stock prices; (b) several well-known technical indicators; (c) counts and sentiment scores of published news articles for a given stock; (d) trends in Google searches for the given stock ticker; and (e) number of unique visitors for pertinent Wikipedia pages. (continued on next page)

Classification for stock trend prediction

- Bin Weng, Lin Lu, et al. 2021. Predicting short-term stock prices using ensemble methods and online data sources. Expert Systems With Applications. 112, pg. 258–273.
- (continued from previous page) Once the data is collected, we use a structured approach for data preparation. Then, the AI platform trains four machine learning ensemble methods: (a) a neural network regression ensemble; (b) a support vector regression ensemble; (c) a boosted regression tree; and (d) a random forest regression. In the cross-validation phase, the AI platform picks the “best” ensemble for a given stock.
- To evaluate the efficacy of our expert system, we first present a case study based on the Citi Group stock ($C) with data collected from 01/01/2013 - 12/31/2016. We show the expert system can predict the 1-day ahead $C stock price with a mean absolute percent error (MAPE) ≤ 1.50% and the 1–10 day ahead with a MAPE ≤ 1.89%, which is better than the reported results in the literature. We show that the use of features extracted from online sources does not substitute the traditional financial metrics, but rather supplements them to improve upon the prediction performance of machine learning based methods.
- To highlight the utility and generalizability of our expert system, we predict the 1-day ahead price of 19 additional stocks from different industries, volatilities and growth patterns. We report an overall mean for the MAPE statistic of 1.07% across our five different machine learning models, including a MAPE of under 0.75% for 18 of the 19 stocks for the best ensemble (boosted regression tree).

- Bin Weng, Lin Lu, et al. 2021. Predicting short-term stock prices using ensemble methods and online data sources. *Expert Systems With Applications*. 112, pg. 258–273.



Ensemble Classifier for Stock Trading Recommendation

Worasucheep, C. (2021)

- This paper presents a heterogeneous *ensemble classifier* for *price trend prediction* of a stock, in which the prediction results are subsequently used in *trading recommendation*. The proposed ensemble model is based on Support vector machine, Artificial neural networks, Random forest, Extreme gradient boosting, and Light gradient boosting machine. A feature selection is performed to choose an optimal set of 45 technical indicators as input attributes of the model. Each base classifier is executed with an extensive hyperparameter tuning to improve performance. The prediction results from five base classifiers are aggregated through a modified majority voting among three classifiers with the highest accuracies, to obtain final prediction result. The performance of proposed ensemble classifier is evaluated using daily historical prices of 20 stocks from Stock Exchange of Thailand, with 3 overlapping datasets of 5-year intervals during 2014–2020 for different market conditions. The experimental results show that the proposed ensemble classifier clearly outperforms buy-and-hold strategy, individual base classifiers, and the ensemble with straightforward majority voting in terms of both trading return and Sharpe ratio.

Model of Ensemble Classifier for Stock Trading Recommendation

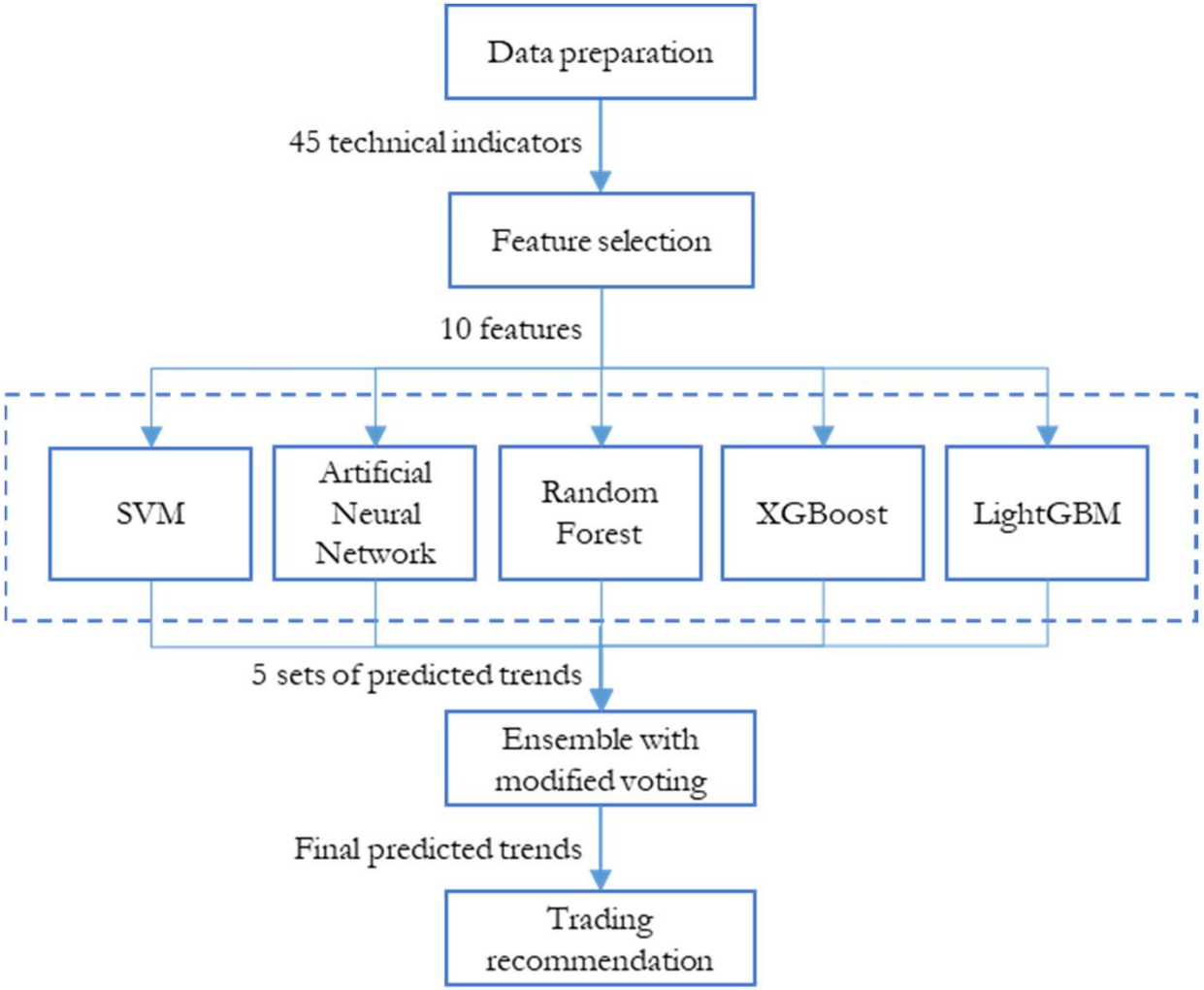


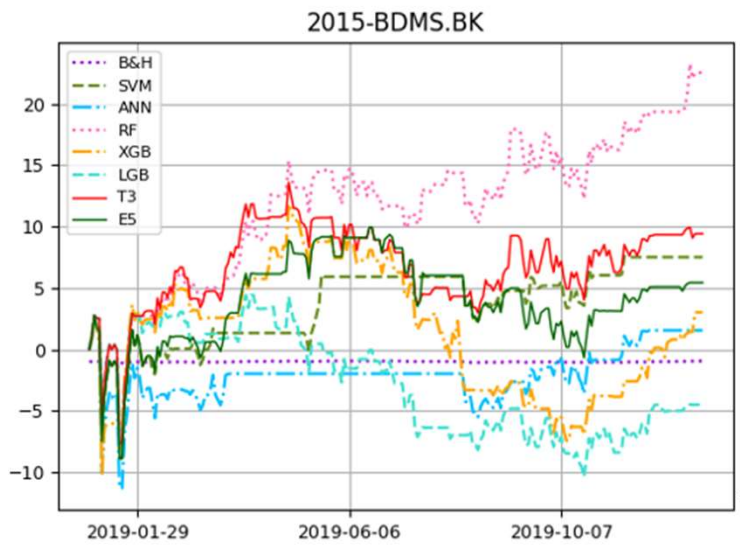
Table 2. Final decision of trend prediction results.

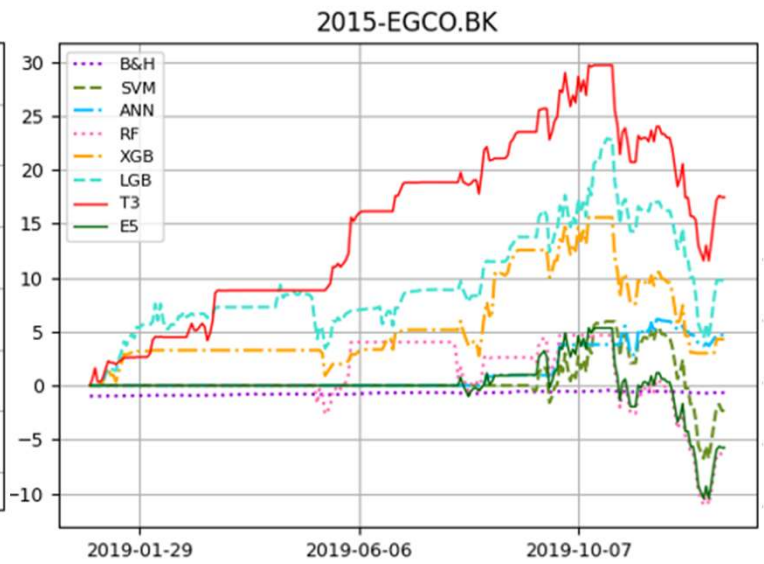
Classifier 1	Classifier 2	Classifier 3	Ensemble T3
-1	-1	Any	-1
-1	Any	-1	-1
Any	-1	-1	-1
0	0	Any	0
0	Any	0	0
Any	0	0	0
1	1	Any	1
1	Any	1	1
Any	1	1	1
-1	0	1	Prediction result from the base classifier with highest accuracy
0	1	-1	
1	-1	0	

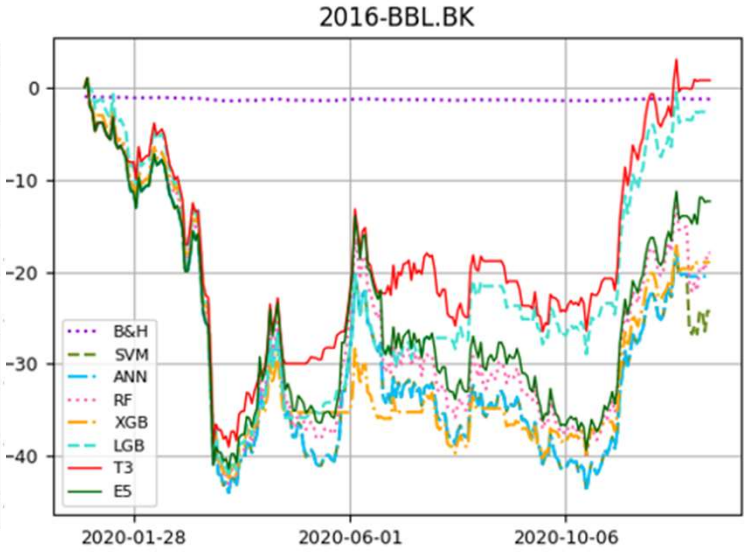
Technical indicators used

Indicator	Description	Number of indicator(s)
RSI10, RSI15, RSI20, RSI25	Relative Strength Index (10, 15, 20, and 25 days)	4
SLOWK, SLOWD	Stochastic Oscillators (%K, %D)	2
ADX10, ADX20	Average Directional Movement Index (10-day and 20-day)	2
BBAND10, BBAND20, BBAND30	Bollinger Band (10-day, 20-day, 30-day)	3
DISP1, DISP2, DISP5, DISP10, DISP20, DISP50, DISP100	Disparity n = 1, 2, 5, 10, 20, 50, 100 days	7
AROONOSC	Aroon Oscillator (14-day)	1
CCI14, CCI28	Commodity Channel Index (14-day and 28-day)	2
CMO14	Chande Momentum Oscillator (14-day)	1
DMI14	Directional Movement Index (14-day)	1
MACD(12, 26)	Moving Average Convergence and Divergence	3
MFI10, MFI14, MFI20	Money Flow Index (10-day, 14-day, 20-day)	3
PPO12:26, PPO9:20	Percentage Price Oscillator (12-26-day, 9-20-day)	2
ROC5, ROC10, ROC20, ROC40, ROC80	Rate of Change (5-day, 10-day, 20-day, 40-day, 80-day)	5
TRIX30	Triple Smooth EMA (30-day)	1
WILLR10, WILLR15, WILLR20	William %R (10-day, 15-day, 20-day)	3
ADOSC(3, 10)	Accumulation/Distribution Oscillator (3-10-day)	1
OBV	On Balance Volumn	1
NATR10, NATR15, NATR20	Normalized Average True Range (10-day, 15-day, 20-day)	3

Some few outputs (comparisons of returns)







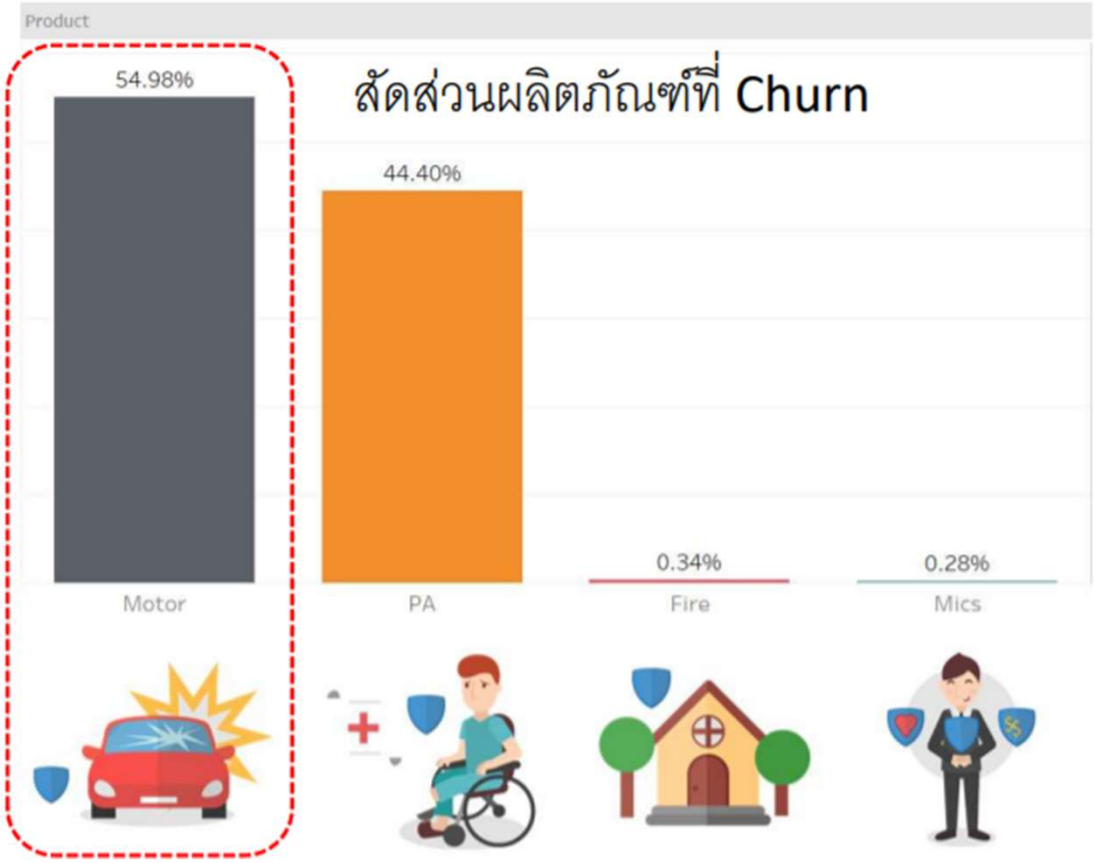
Summary of results

Table 6. Averages of return surplus ($\Delta X = R_X - R_{BH}$), Sharpe ratios (*Sr*), and accuracy (*Ac*).

Metric	Dataset	SVM	ANN	RF	XGB	LGB	T3	E5
Average Δ	A. 2014 – 2018	12.113	11.506	12.720	12.074	10.340	12.375	12.878
	B. 2015 – 2019	0.453	0.822	4.625	5.787	6.928	10.006	7.632
	C. 2016 – 2020	7.966	8.366	13.356	14.115	15.696	18.150	15.466
$\overline{\Delta}$	Grand average	6.844	6.898	10.234	10.659	10.988	**13.510**	11.992
Average *Sr*	A. 2014 – 2018	0.394	0.040	0.476	0.265	-0.298	0.755	0.916
	B. 2015 – 2019	-0.063	-0.587	0.710	**1.134**	**1.006**	**1.261**	**1.160**
	C. 2016 – 2020	-0.063	-0.009	0.710	0.910	**1.071**	**1.586**	**1.225**
$\overline{Sr}$	Grand average	0.089	-0.185	0.632	0.770	0.593	**1.201**	**1.100**
Average *Ac*	A. 2014 – 2018	0.412	0.414	0.427	0.415	0.419	0.423	0.445
	B. 2015 – 2019	0.401	0.400	0.413	0.408	0.417	0.497	0.463
	C. 2016 – 2020	0.427	0.430	0.443	0.440	0.443	0.481	0.476
$\overline{Ac}$	Grand average	0.414	0.414	0.428	0.421	0.426	**0.467**	0.461

Churn Prediction of a Motor Insurance Firm

Problem Statement



Objective

- เพื่อทำนายโอกาสที่จะ Churn ของลูกค้า**ก่อนที่กรมธรรม์จะหมดอายุ**
- เพื่อนำผลการทำนายไปปรับใช้กับลูกค้ากลุ่มที่ผลการทำนายเป็น Churn และ**ลดจำนวนการ Churn ของลูกค้ากลุ่มนี้**
- เพื่อหา Insight และนำไปประยุกต์ใช้ในงานอื่นๆที่เกี่ยวข้องต่อไป

Scope

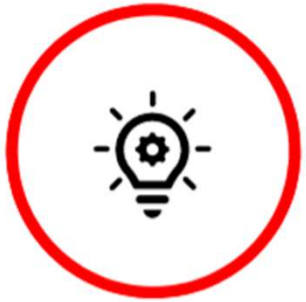
- ประกันภาคสมัครใจ

Churn Prediction of a Motor Insurance Firm

Tools

Microsoft Access / Excel

Process



1 Collect and Know your Data

- Explore the dataset
- Data Understanding (Data Visualization)
- Data Preparation (Scrub Data)

2 Model Building

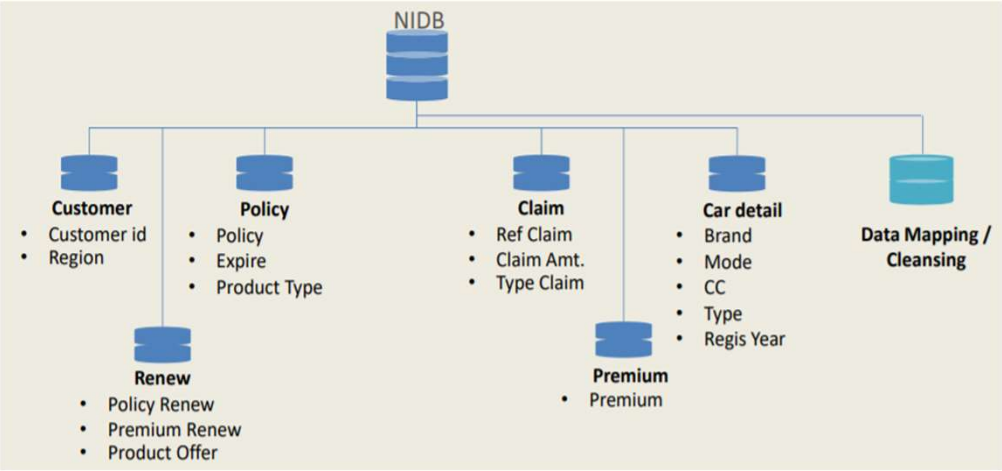
- Feature / Attribute Selection
  - Forward Selection
  - Backward Selection
  - Evolutionary
- Churn Prediction
  - Logistic Regression
  - Decision Tree
  - Random Forest
- Model Selection

3 Result

4 ประโยชน์และการนำไปใช้

Churn Prediction of a Motor Insurance Firm

- Step 2: Explore Data



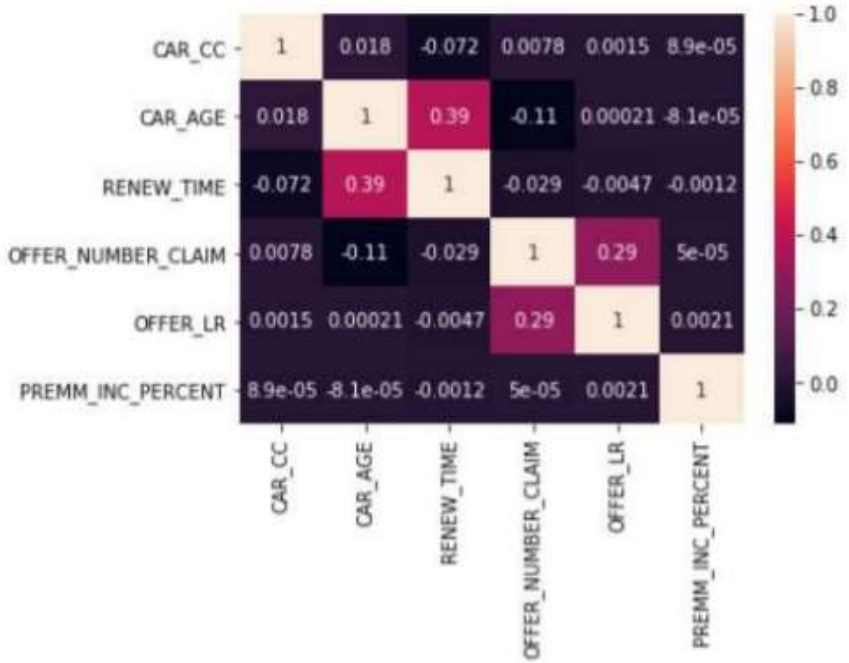
Step 1: Data Acquisition

Variable	Feature / Attribute	Description	Data Type
ID	Pol_ID	เลขกรมธรรม์ภาคสมัครใจ ที่หมดความคุ้มคร	Nominal / Polynomial
Y	CHURN_STATUS	สถานะการ CHURN (Churn or Renew)	Nominal / Binary
X1	INS_GENDER	เพศผู้เอาประกันภัย (Male or Female)	Nominal / Binary
X2	LOB	Line of Business : ประเภทการทำประกันภัย	Nominal / Polynomial
X3	SALE_CHANNEL	ช่องทางการขาย / ที่มาของกรมธรรม์ที่หมดอ	Nominal / Polynomial
X4	REGION	พื้นที่ / ภูมิภาค ในการซื้อประกัน i.e. กรุงเท	Nominal / Polynomial
X5	CAR_BRAND	ยี่ห้อรถ i.e. TOYOTA, HONDA, ISUZU, NI	Nominal / Polynomial
X6	CAR_TYPE	กลุ่มรถ i.e. City Car, Pickup , Pickup-Van, S	Nominal / Polynomial
X7	COMPULSORY	กรมธรรม์ภาคสมัครใจมีการซื้อ พรบ. ด้วยหร	Nominal / Binary
X8	CAR_CC	CC รถยนต์	Ordinary / Polynomial
X9	CAR_AGE	อายุรถยนต์ตอนออกใบเตือนต่ออายุประกันภ	Interval
X10	RENEW_TIME	จำนวนครั้งที่ต่ออายุ i.e. 1, 2, 3, 4, ...	Interval
X11	OFFER_CLAIM	มีการเคลม กรณีที่ผู้เอาประกันเป็นฝ่ายผิดหรื	Nominal / Binary
X12	OFFER_NUMBER_CLAIM	จำนวนครั้งในการเคลม (เฉพาะผู้เอาประกันเ	Interval
X13	OFFER_LR	Loss Ratio(เฉพาะผู้เอาประกันเป็นฝ่ายผิด) =	Ratio
X14	PREMM_CHANGE	%ส่วนต่าง ของเบี้ยฯประกันภัยปีที่แล้วกับเบี้	Ratio

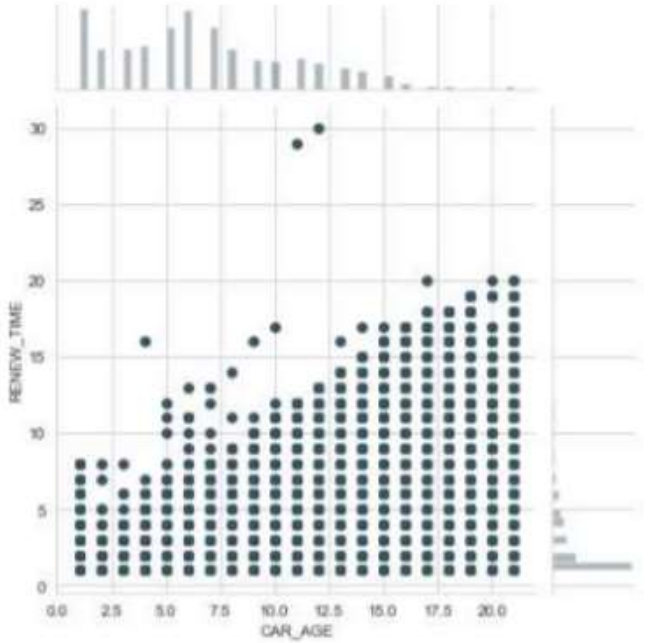
Churn Prediction of a Motor Insurance Firm

- Step 3: Find correlation coefficient of features

```
In [10]: sns.heatmap(df.corr(), annot=True)
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x1698a914208>
```



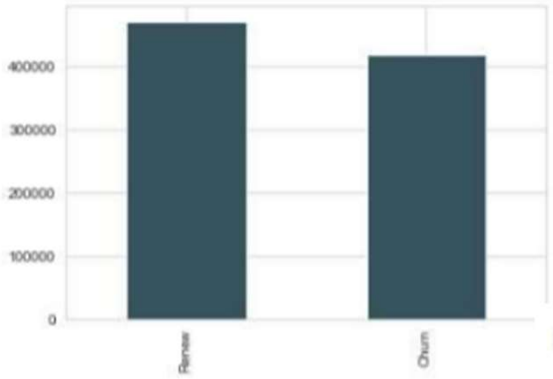

```
In [12]: #Data Visualization
         sns.set_palette("GnBu_d")
         sns.set_style('whitegrid')

         sns.jointplot(x='CAR_AGE', y='RENEW_TIME', data=df)
Out[12]: <seaborn.axisgrid.JointGrid at 0x169895b1ac8>
```



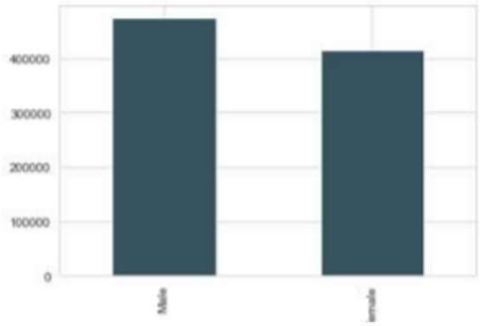
Churn Prediction of a Motor Insurance Firm

- Step 4: Visualize data

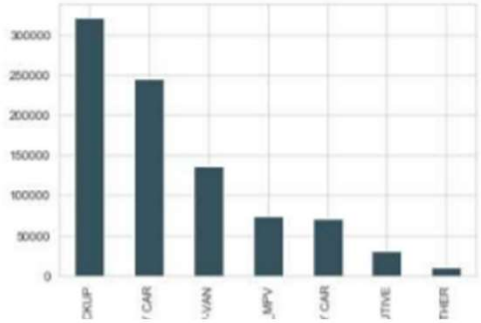
```
In [24]: #สัดส่วนการต่ออายุและ Churn
         pd.value_counts(df['CHURN']).plot.bar()
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x1698aa1f6d8>
```



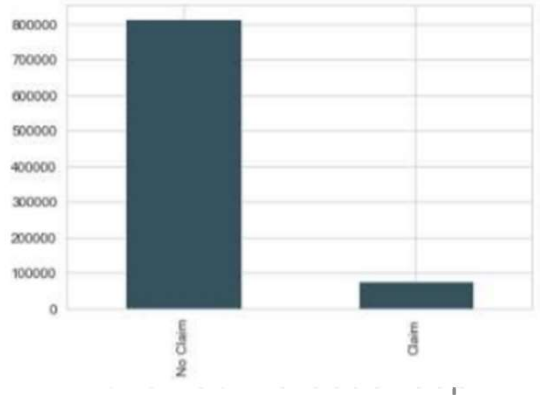

```
In [23]: #สัดส่วนประชากร ชาย/หญิง
         pd.value_counts(df['INS_GENDER']).plot.bar()
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x1698a9c42b0>
```



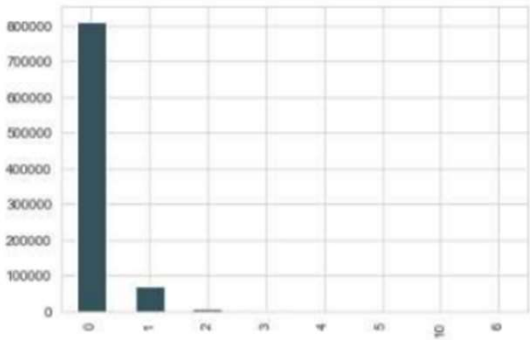

```
In [21]: #สัดส่วนประเภทรถยนต์
         pd.value_counts(df['CAR_TYPE']).plot.bar()
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x1698a161c88>
```




```
In [20]: #สัดส่วนในการเคลม
         pd.value_counts(df['OFFER_CLAIM']).plot.bar()
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1698a011860>
```




```
In [19]: #จำนวนครั้งในการแจ้งเคลม
         pd.value_counts(df['OFFER_NUMBER_CLAIM']).plot.bar()
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1698a095208>
```



Churn Prediction of a Motor Insurance Firm

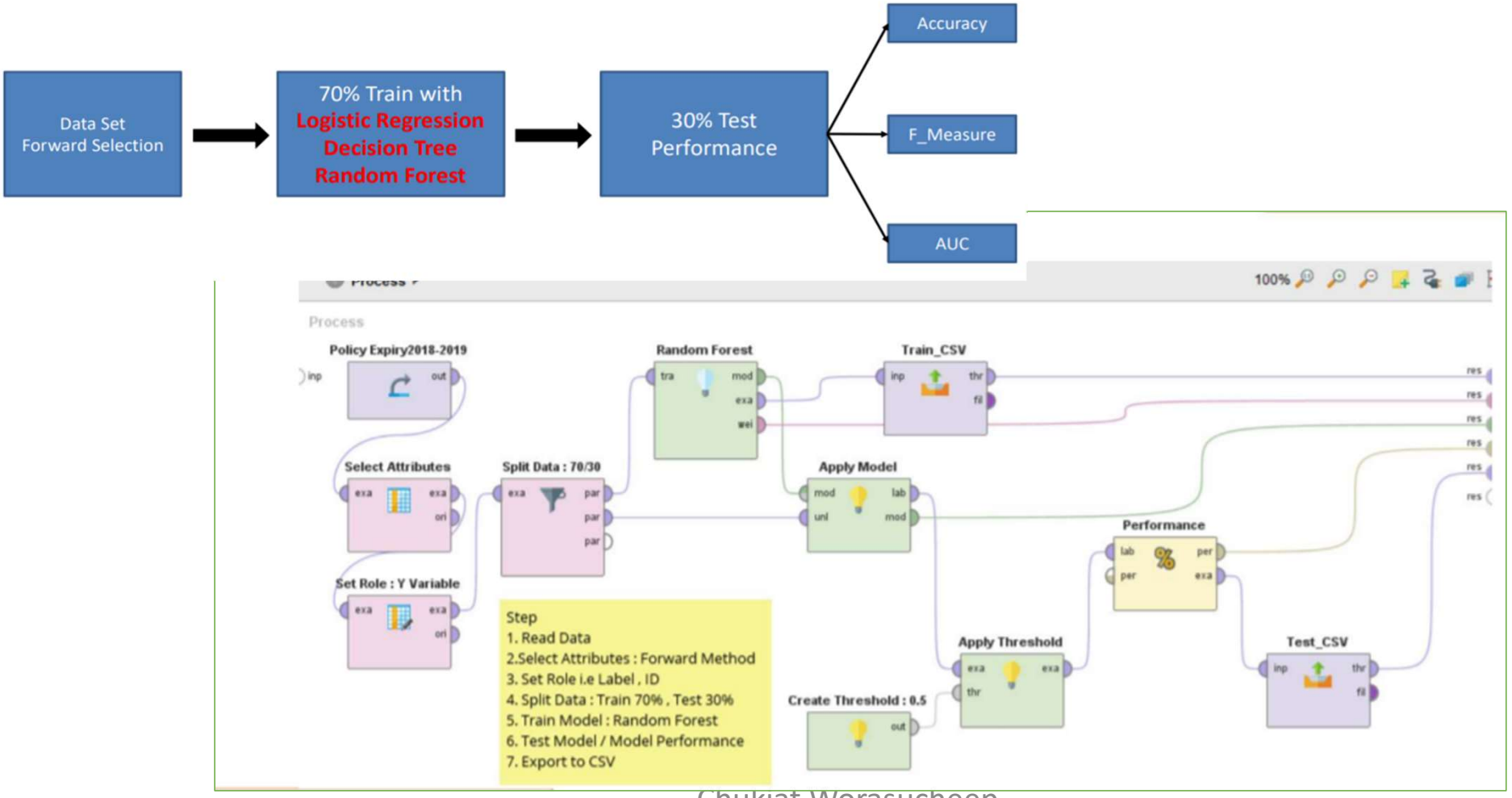
- Step 5: Feature Selection

Attribute	Forward	Backward	Evolutionary
INS_GENDER		x	
LOB	x	x	x
SALE_CHANNEL	x	x	x
REGION			x
CAR_BRAND		x	
CAR_TYPE		x	x
COMPULSORY			
CAR_CC		x	x
CAR_AGE			
RENEW_TIME	x	x	x
OFFER_CLAIM	x	x	x
OFFER_NUMBER_CLAIM	x		x
OFFER_LR			
PREMM_INC_PERCENT	x	x	x
Number of Attribute	**6**	**9**	**9**

Method	Accuracy
Forward	76.67%
Backward	76.87%
Evolutionary	76.88%

Churn Prediction of a Motor Insurance Firm

- Step 6: Classification with 3 classifiers



Churn Prediction of a Motor Insurance Firm

- Step 7: Performance Evaluation of 3 classifiers

	Logistics Regression	Decision Tree	Random Forest
Accuracy	75.37%	75.88%	76.73%
F_Measure	73.35%	74.63%	74.09%
AUC	0.812	0.795	0.827

Random Forest

Confusion Matrix

Positive Class : CHURN

Accuracy 76.73%

	Total	Actual **CHURN**	Actual **RENEW**	
Pred. **CHURN**	**122,770 (43%)**	94,922 (1)	27,848 (2)	**77.32%**
Pred. **RENEW**	**162,631**	38,558 (3)	124,073 (4)	**78.15%**
		71.11%	**81.67%**	

Churn Prediction of a Motor Insurance Firm

- Step 8: Analysis of results

Predict: CHURN

1 = True Positive

Pred.Churn Churn		94,922
Product	SUPER LITE	40%
Channel	AL	69%
Renew Time	1	99%
Status Claim	No Claim	89%
%Premm Change	-2%	35%

2 = False Positive

Pred.Churn Renew		27,848
Product	Type 1	43%
Channel	AL	52%
Renew Time	1	100%
Status Claim	No Claim	91%
%Premm Change	-2%	19%

Predict: RENEW

3 = False Negative

Pred.Renew Churn		38,558
Product	Type 1	55%
Channel	Branch	46%
Renew Time	2	32%
Status Claim	No Claim	91%
%Premm Change	0%	23%

4 = True Negative

Pred.Renew Renew		124,073
Product	Type 1	67%
Channel	Tbroker	44%
Renew Time	2	25%
Status Claim	No Claim	93%
%Premm Change	0%	35%

What we've learnt from this session

- Concept of regression analysis
- Linear Regression
- Multiple Linear Regression
- Polynomial Regression
- Overfitting and Underfitting
- More advanced regressors
- Case studies