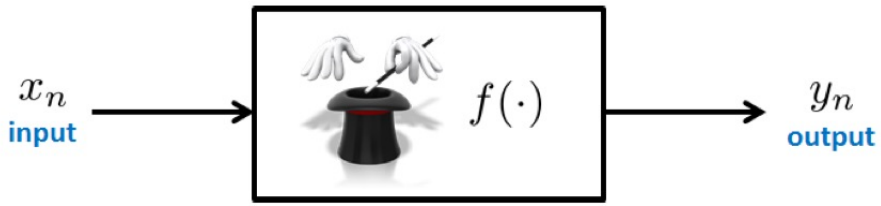
Probability & Statistics for DS & AI

Regression Analysis

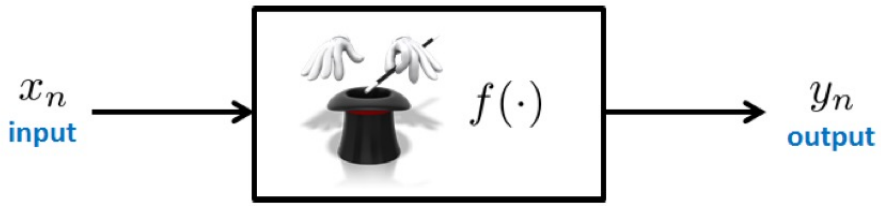
Michele Guindani

Summer



A Regression is a process for finding the best approximation to the relationship between some independent (input) data and some dependent (output) data.

$$y_n = f(x_n)$$

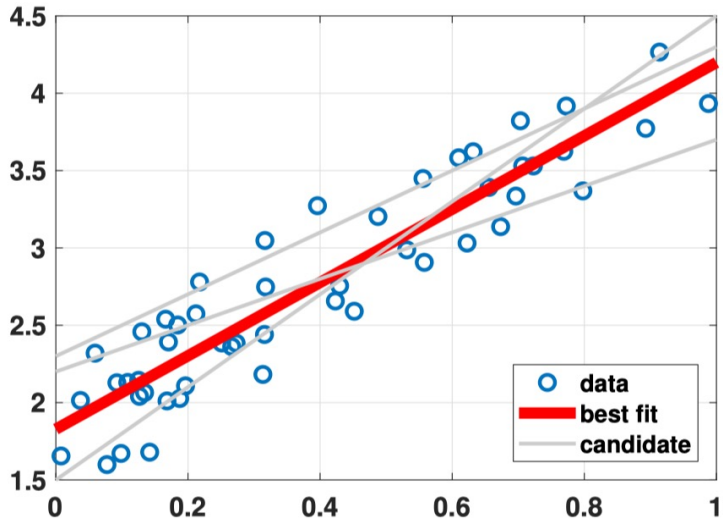


A Regression is a process for finding the best approximation to the relationship between some independent (input) data and some dependent (output) data.

$$y_n = f(x_n)$$

Finding the ``true" f(x) is infeasible in most problems.
The idea of regression is to add a **structure** to the problem

E.g.: simple linear regression $y_n = \alpha + \beta\, x_n + \epsilon_n$



Types of Regression models

- The simplest regression model is linear with a single predictor:

  Basic/Simple regression model: $y_i = a_i + bx_i + \varepsilon,\ i = 1, \ldots, n.$

Types of Regression models

- The simplest regression model is linear with a single predictor:

  Basic/Simple regression model: $y_i = a_i + bx_i + \varepsilon$, $i = 1, \ldots, n$.

  It assumes that the outcome y is continuous.

Types of Regression models

- The simplest regression model is linear with a single predictor:

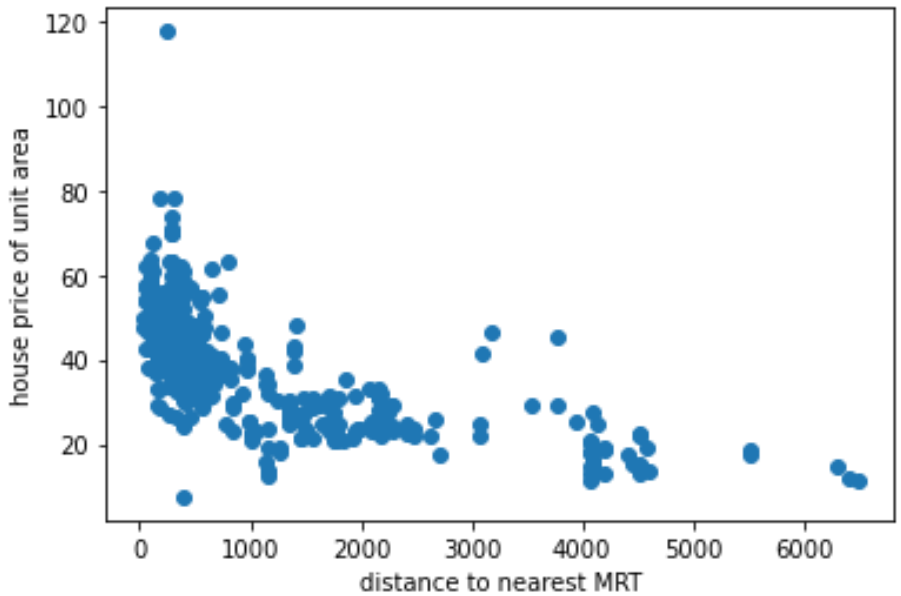
  Basic/Simple regression model: $y_i = a_i + bx_i + \varepsilon,\ i = 1, \ldots, n.$

It assumes that the outcome y is continuous.

Example (Real Estate Evaluation)

Cost of a house may depend on proximity to nearest MRT station

```
# scatter plot between house price and distance to nearest MRT
plt.plot(df['X3 distance to the nearest MRT station'], df['Y house price of unit area'], 'o')
plt.xlabel('distance to nearest MRT')
plt.ylabel('house price of unit area')
```



Example (Simple Linear Regression)

- Insurance companies would like to understand the association between healthcare costs and ageing.
- Businesses often use linear regression to understand the relationship between advertising spending and revenue.
- Medical researchers often use linear regression to understand the relationship between drug dosage and physiological measurements (e.g., blood pressure)
- In Sport statistics, one may want to assess the relationship between different training programs and performance.
- The pricing of a ridesharing service in Mumbai may depend on the weather (clear, clouds, drizzle, fog, haze, mist, rain, thunderstorm)

Goals of Linear Regression

- As pointed out already a few times, a good data analysis needs to be informed by the objective of the study.
- Regression analysis can be used both for estimation or for prediction, e.g. think of the following questions:

Goals of Linear Regression

- As pointed out already a few times, a good data analysis needs to be informed by the objective of the study.
- Regression analysis can be used both for estimation or for prediction, e.g. think of the following questions:
  - What's the best training program?

Goals of Linear Regression

- As pointed out already a few times, a good data analysis needs to be informed by the objective of the study.
- Regression analysis can be used both for estimation or for prediction, e.g. think of the following questions:
  - What's the best training program?
  - How much should I spend in advertising to increase the revenues so much?

Goals of Linear Regression

- As pointed out already a few times, a good data analysis needs to be informed by the objective of the study.
- Regression analysis can be used both for estimation or for prediction, e.g. think of the following questions:
  - What's the best training program?
  - How much should I spend in advertising to increase the revenues so much?
  - Where should I buy my house given the budget I have?

Goals of Linear Regression

- As pointed out already a few times, a good data analysis needs to be informed by the objective of the study.
- Regression analysis can be used both for estimation or for prediction, e.g. think of the following questions:
  - What's the best training program?
  - How much should I spend in advertising to increase the revenues so much?
  - Where should I buy my house given the budget I have?

⚠ The goal may change a bit how the analysis is conducted (since it may change its focus)

Learning and Prediction

- Here we are interested more in the statistical learning/prediction goal:
  - Given a set of data points, what is the relationship of the data in hand to data yet seen?
  - How should information from one dataset propagate to other data?
  - What's the best forecast of the status of a complex process in the future?

Types of Regression Models

- The simplest regression model is linear with a single predictor:

  Basic/Simple regression model: $y_i = a_i + bx_i + \varepsilon,\ i = 1, \ldots, n.$

Types of Regression Models

- The simplest regression model is linear with a single predictor:

  Basic/Simple regression model: $y_i = a_i + bx_i + \varepsilon,\ i = 1, \ldots, n.$

  It assumes that the outcome y is continuous.

  - Suppose that x is a categorical predictor.
  - To fix ideas, let $x \in \{0, 1\}$ (binary).
  - Then, we can consider $x = 0$ as a reference value for the regression:

Types of Regression Models

- The simplest regression model is linear with a single predictor:

  Basic/Simple regression model: $y_i = a_i + bx_i + \varepsilon,\ i = 1, \ldots, n.$

  It assumes that the outcome y is continuous.

  - Suppose that x is a categorical predictor.
  - To fix ideas, let $x \in \{0, 1\}$ (binary).
  - Then, we can consider $x = 0$ as a reference value for the regression:

    👉 α is the baseline model (when $x = 0$)

Types of Regression Models

- The simplest regression model is linear with a single predictor:

  Basic/Simple regression model: $y_i = a_i + bx_i + \varepsilon,\ i = 1, \ldots, n.$

  It assumes that the outcome y is continuous.

  - Suppose that x is a categorical predictor.
  - To fix ideas, let $x \in \{0, 1\}$ (binary).
  - Then, we can consider $x = 0$ as a reference value for the regression:

    👉 α is the baseline model (when $x = 0$)

    👉 β captures the deviation from the baseline model when x varies. More in detail, β captures the differences between the expected outcome for units (individuals) with $x = 1$ *vs* the expected outcome for units (individuals) with $x = 0$.

Types of Regression Models

- The simplest regression model is linear with a single predictor:

  Basic/Simple regression model: $y_i = a_i + bx_i + \varepsilon,\ i = 1, \ldots, n.$

  It assumes that the outcome y is continuous.

  - Suppose that x is a categorical predictor.
  - To fix ideas, let $x \in \{0, 1\}$ (binary).
  - Then, we can consider $x = 0$ as a reference value for the regression:

    👉 α is the baseline model (when $x = 0$)

    👉 β captures the deviation from the baseline model when x varies. More in detail, β captures the differences between the expected outcome for units (individuals) with $x = 1$ *vs* the expected outcome for units (individuals) with $x = 0$.

    🙄 In (almost all) textbooks, β is (sloppily!) defined as the effect on y of a unit (1) increase of x!! 😄

What if x is continuous

- If x is continuous, for the interpretation of the coefficients, it is important to identify a reference value
- This could be $x = 0$.
- Most often, even if x could assume the value 0, you may want to center the values x_i around their mean $\bar{x}$, or even standardize them.
  - In this case, the sample mean $\bar{x}$ becomes the reference value and the interpretation of the parameters changes as follow:
    - ★ $\alpha + b\bar{x}$ is the baseline model
    - ★ In particular, α as the intercept of the regression captures the "average" common value of the outcome which is unrelated to the covariate (predictor)
    - ★ β captures the difference between the expected outcome for units (individuals)with value of a covariate equal to $x + 1$ *vs* the expected outcome for units (individuals) with covariate euqal to x.
  - 🙄 In (almost all) textbooks, β is (sloppily!) defined as the effect on y of a unit (1) increase of x!! 😄

Types of Regression models

- Multiple regression considers additional predictors

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \varepsilon$$

  - Written in vector-matrix notation as $y = X\beta +$ error.

Types of Regression models

- Multiple regression considers additional predictors

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \varepsilon$$

  - Written in vector-matrix notation as $y = X\beta +$ error.

Table: Example of a general regression data format.

Subject	y	x_1	x_2	$\cdots$	x_r
1	y_1	x_{11}	x_{12}	$\cdots$	x_{1r}
2	y_2	x_{21}	x_{22}	$\cdots$	x_{2r}
3	y_3	x_{31}	x_{32}	$\cdots$	x_{3r}
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
n	y_n	x_{n1}	x_{n2}	$\cdots$	x_{nr}

Multiple Regression model

Example (Real Estate Evaluation)

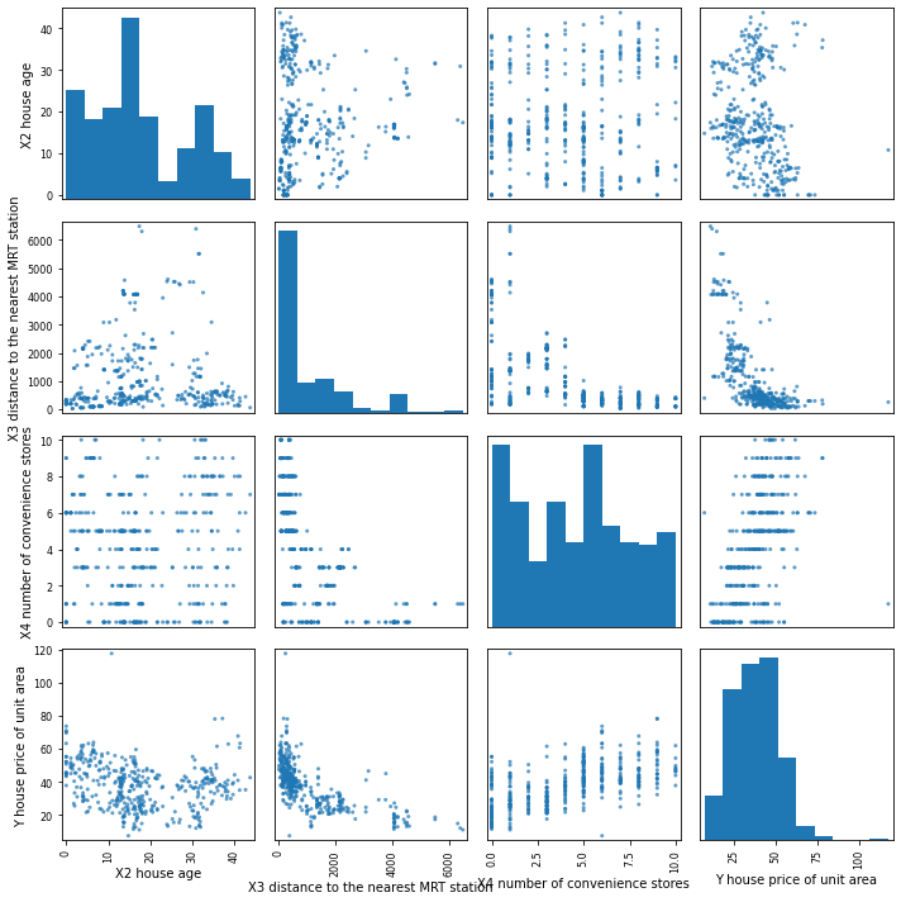
In addition to the distance from the MRT station,we may consider the number of convenience stores and/or the house age...(expert assessment)

Multiple Regression model

Example (Real Estate Evaluation)

In addition to the distance from the MRT station,we may consider the number of convenience stores and/or the house age...(expert assessment)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

axes = pd.plotting.scatter_matrix(
    df.iloc[:,[2,3,4,7]], alpha=0.7,
    figsize=(10,10))
plt.tight_layout()
plt.savefig('scatter_matrix.png')
```



Multiple Regression Model

Example

- An organization may be interested in finding the relationship between revenue generated from a product and features such as the price, money spent on promotion, competitors' price, and promotion expenses.

Multiple Regression Model

Example

- An organization may be interested in finding the relationship between revenue generated from a product and features such as the price, money spent on promotion, competitors' price, and promotion expenses.

- E-commerce companies such as Amazon, BigBasket, and Flipkart would like to understand the relationship between revenue and features such as
  (a) Number of customer visits to their portal.
  (b) Number of clicks on products.
  (c) Number of items on sale.
  (d) Average discount percentage.

Multiple Regression Model

Example

- An organization may be interested in finding the relationship between revenue generated from a product and features such as the price, money spent on promotion, competitors' price, and promotion expenses.
- E-commerce companies such as Amazon, BigBasket, and Flipkart would like to understand the relationship between revenue and features such as
  (a) Number of customer visits to their portal.
  (b) Number of clicks on products.
  (c) Number of items on sale.
  (d) Average discount percentage.
- Agricultural scientists often use linear regression to measure the effect of fertilizer and water on crop yields.

Multiple Regression Model

Example

- An organization may be interested in finding the relationship between revenue generated from a product and features such as the price, money spent on promotion, competitors' price, and promotion expenses.
- E-commerce companies such as Amazon, BigBasket, and Flipkart would like to understand the relationship between revenue and features such as
  (a) Number of customer visits to their portal.
  (b) Number of clicks on products.
  (c) Number of items on sale.
  (d) Average discount percentage.
- Agricultural scientists often use linear regression to measure the effect of fertilizer and water on crop yields.
- Prices of grains in agricultural markets vary every day. Its daily price fluctuations may depend on last day's temperature, last day's humidity, last day's sold out stock, last day's market arrivals, last day's price of substitute commodity, etc.

Models with Interaction

Sometimes, one may want to consider nonadditive models such as

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + \varepsilon$$

which contains an interaction between the input variables x_1 and x_2.

Example (FEV)

The FEV, which is an acronym for forced expiratory volume, is a measure of how much air a person can exhale (in litres) during a forced breath.

We have data on the FEV of 606 children, between the ages of 6 and 17. We want to assess the impact of smoking on the FEV of children.

Models with Interaction

Sometimes, one may want to consider nonadditive models such as

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + \varepsilon$$

which contains an interaction between the input variables x_1 and x_2.

Example (FEV)

The FEV, which is an acronym for forced expiratory volume, is a measure of how much air a person can exhale (in litres) during a forced breath.

We have data on the FEV of 606 children, between the ages of 6 and 17. We want to assess the impact of smoking on the FEV of children.

The dataset also provides additional information on these children: their age, their height, their gender and, most importantly, whether the child is a smoker or a non-smoker.

Example (FEV - continued)

Younger kids have generally lower FEV than older kids. Smoking may affect FEV differently at different ages:

$$\text{FEV}_i = \beta_1 + \beta_2\,\text{Age}_i + \beta_3\,\text{Smoker}_i + \beta_4\,\text{Age}_i \times \text{Smoker}_i + \varepsilon_i$$

Since the FEV values for smokers versus nonsmokers depends on age, we say that age modifies the effect of smoking or that age is an effect modifier of smoking.

Medical researchers tend to use the term effect modification, while statisticians refer to interaction between the two risk factors.

Steps in building a Regression Model

1. Collect/Extract Data

   Do not underestimate the experimental design phase!

Steps in building a Regression Model

1. Collect/Extract Data

   Do not underestimate the experimental design phase!

2. Pre-Process the Data

   -  Check if there are missing values. Identify outliers in the dataset. Decide what to do.

Steps in building a Regression Model

1. Collect/Extract Data

   Do not underestimate the experimental design phase!

2. Pre-Process the Data

   - 👁 Check if there are missing values. Identify outliers in the dataset. Decide what to do.
   - ➗ Many new variables (such as the log of a variable, ratio of variables or product of variables) can be derived (aka feature engineering) and also used in model building.

Steps in building a Regression Model

1. Collect/Extract Data

   Do not underestimate the experimental design phase!

2. Pre-Process the Data

   - 👁 Check if there are missing values. Identify outliers in the dataset. Decide what to do.
   - ➗ Many new variables (such as the log of a variable, ratio of variables or product of variables) can be derived (aka feature engineering) and also used in model building.
   - 🔨 Check if you can combine variables that provide the same information (e.g, instead of dummy variables for each comorbidity, a single variable indicating the number of comorbidites for each individual)

Steps in building a Regression Model

1. Collect/Extract Data

   Do not underestimate the experimental design phase!

2. Pre-Process the Data

   - 👁 Check if there are missing values. Identify outliers in the dataset. Decide what to do.
   - ➗ Many new variables (such as the log of a variable, ratio of variables or product of variables) can be derived (aka feature engineering) and also used in model building.
   - 🔨 Check if you can combine variables that provide the same information (e.g, instead of dummy variables for each comorbidity, a single variable indicating the number of comorbidites for each individual)
   - 💻 Categorical data must be pre-processed using dummy variables (part of feature engineering, factors) before it is used in the regression model.

Dummy Variables

	Age	YearsExperience	Salary	Gender	Classification	Job
0	22	1.1	39343	Female	Low	Assistant
1	22	1.3	46205	Male	TOP	Professor
2	23	1.5	37731	Female	TOP	Administrative
3	24	2.0	43525	Female	Medium	Assistant
4	25	2.2	39891	Male	Medium	Professor

```
X = df[['Age', 'YearsExperience', 'Gender', 'Classification', 'Job']]
X = pd.get_dummies(data=X, drop_first=True)
X.head()
```

	Age	YearsExperience	Gender_Male	Classification_Medium	Classification_TOP	Job_Assistant	Job_Manager	Job_Professor	Job_Senior Manager
0	22	1.1	0	0	0	1	0	0	0
1	22	1.3	1	0	1	0	0	1	0
2	23	1.5	0	0	1	0	0	0	0
3	24	2.0	0	1	0	1	0	0	0
4	25	2.2	1	1	0	0	0	1	0

https://medium.com/analytics-vidhya/implementing-linear-regression-using-sklearn

Steps in building a Regression Model

③ Dividing Data into Training and Validation Datasets

- If the dataset is large enough, it is good practice to (randomly) divide the data into (at least) two subsets:
  training and testing dataset.

  Typically, $70 - 80\%$ of the data is used for training

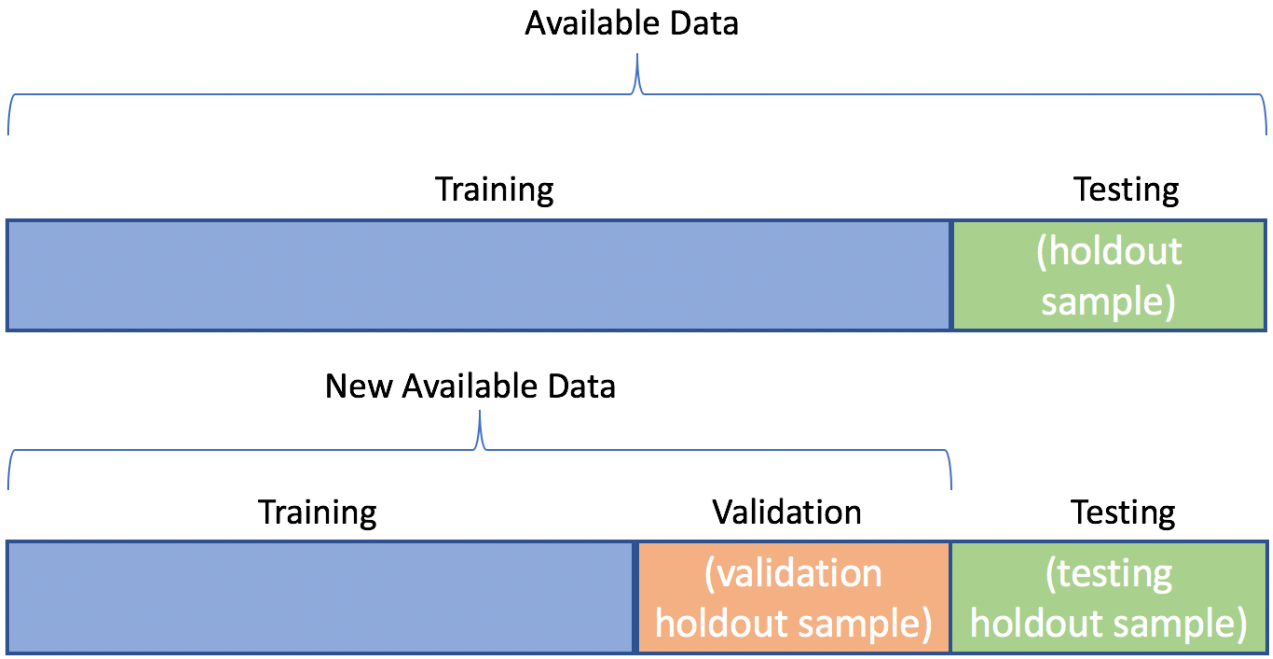
Steps in building a Regression Model

③ Dividing Data into Training and Validation Datasets

✂️ If the dataset is large enough, it is good practice to (randomly) divide the data into (at least) two subsets:
training and testing dataset.

Typically, $70 - 80\%$ of the data is used for training

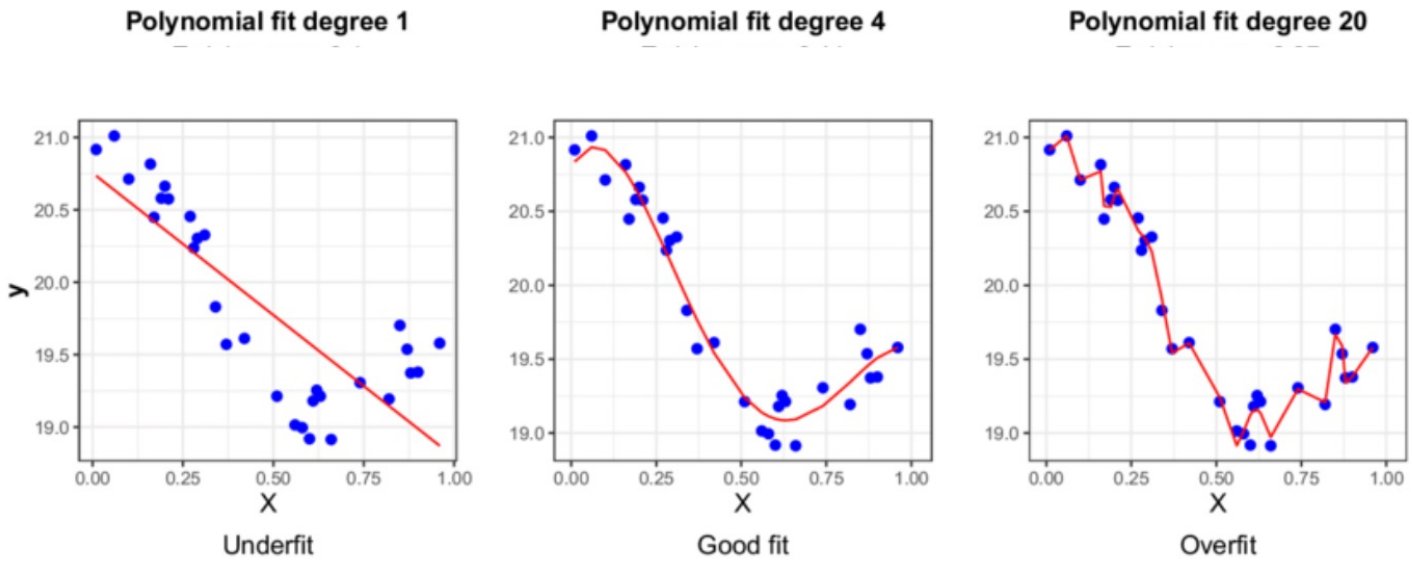
👉 Important to measure the performance of the model



Steps in building a Regression Model

③ Dividing Data into Training and Validation Datasets

👉 Also Important to check for overfitting

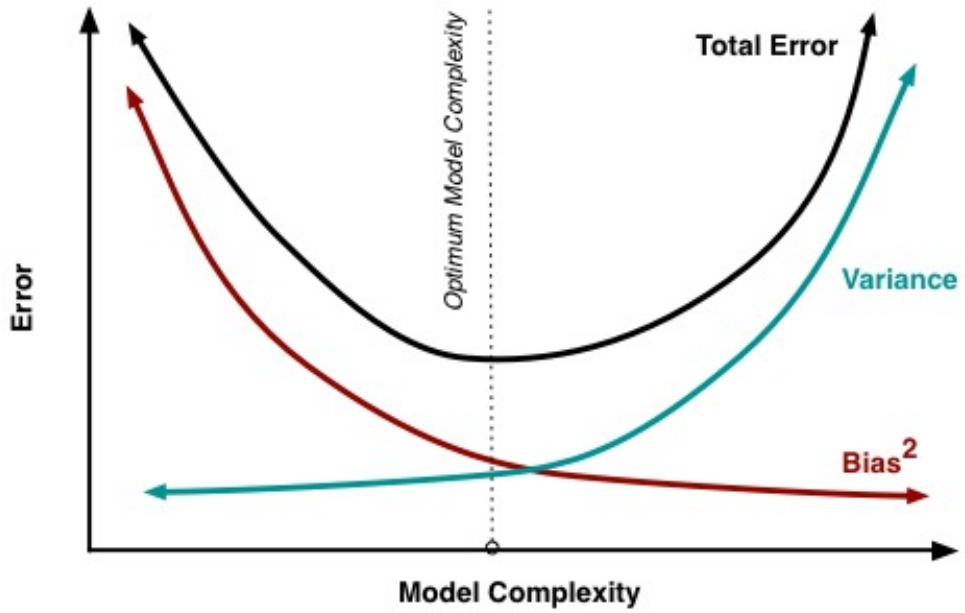


$$E(Y|X) = \beta_0 + \beta_1 x$$

$$E(Y|X) = \beta_0 + \beta_1\, x + \beta_2\, x^2 + \beta_3\, x^3 + \beta_4\, x^4$$

$$E(Y|X) = \beta_0 + \beta_1\, x + \beta_2\, x^2 + \beta_3\, x^3 + \beta_4\, x^4 + \ldots + \beta_{20}\, x^{20}$$

Bias-Variance Trade off



Overfitting improves if $N \uparrow$: Variance drops as N grows. Bias is unchanged.

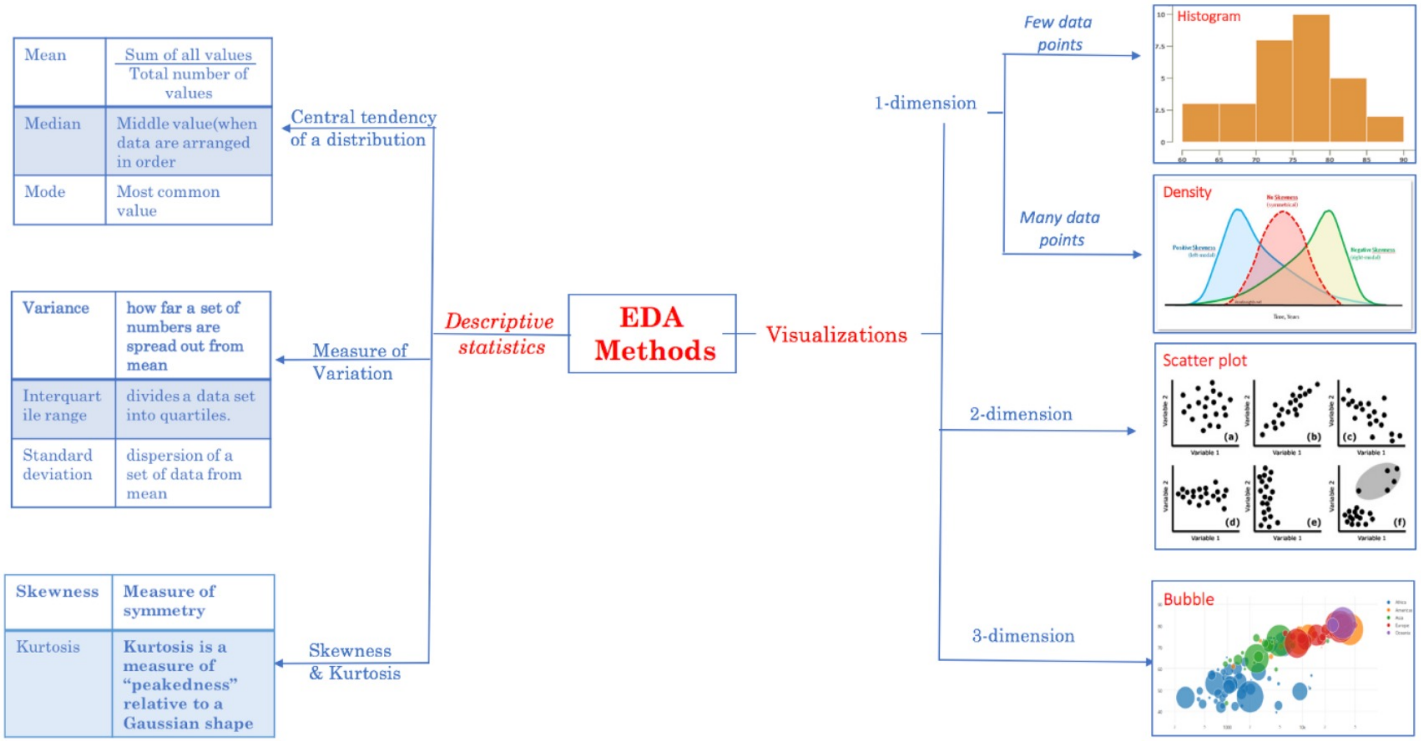
Overfitting worsens if the variance of the fit $\uparrow$. If the training noise grows, the fit will have more fluctuations, so the variance of the curve will grow.

Overfitting worsens if the the target function f is too complicated

Steps in building a Regression Model

④ Perform Descriptive Analysis or Data Exploration

- Scatter plots, Box plots, data visualization tools to understand/check validity of assumed relationships between variables



Explorative data analysis

https://towardsdatascience.com/analyze-the-data-through-data-visualization-using-seaborn

Steps in building a Regression Model

⑤ Model building

- 🧰 Choose an appropriate model for the outcome
- 🪚 Decide what variables to include in the model and which should not be considered (expert knowledge *vs* ML).

Steps in building a Regression Model

⑤ Model building

- 🧰 Choose an appropriate model for the outcome
- 🪚 Decide what variables to include in the model and which should not be considered (expert knowledge *vs* ML).
- 🪃 Are you going to expect any interactions?

Steps in building a Regression Model

⑤ Model building

- 🧰 Choose an appropriate model for the outcome
- 🪚 Decide what variables to include in the model and which should not be considered (expert knowledge *vs* ML).
- 🪃 Are you going to expect any interactions?
- 🔗 Think about possible confounding: have you considered all the variables (and related causal pathways) that can affect the outcome?
  - ★ A variable that is associated with both the response and the treatment variable, but is not caused by the treatment variable and vice versa, is called a confounding variable, or simply a confounder.

Steps in building a Regression Model

⑤ Model building

- 🧰 Choose an appropriate model for the outcome
- 🪚 Decide what variables to include in the model and which should not be considered (expert knowledge *vs* ML).
- 🪃 Are you going to expect any interactions?
- 🔗 Think about possible confounding: have you considered all the variables (and related causal pathways) that can affect the outcome?
  - ★ A variable that is associated with both the response and the treatment variable, but is not caused by the treatment variable and vice versa, is called a confounding variable, or simply a confounder.
- ⚖️ Avoid the curse of Hoarding: Fit can easily improved by adding more variables ⇨ loss of prediction performance, overfitting

Steps in building a regression model

⑥ Run and interpret the model

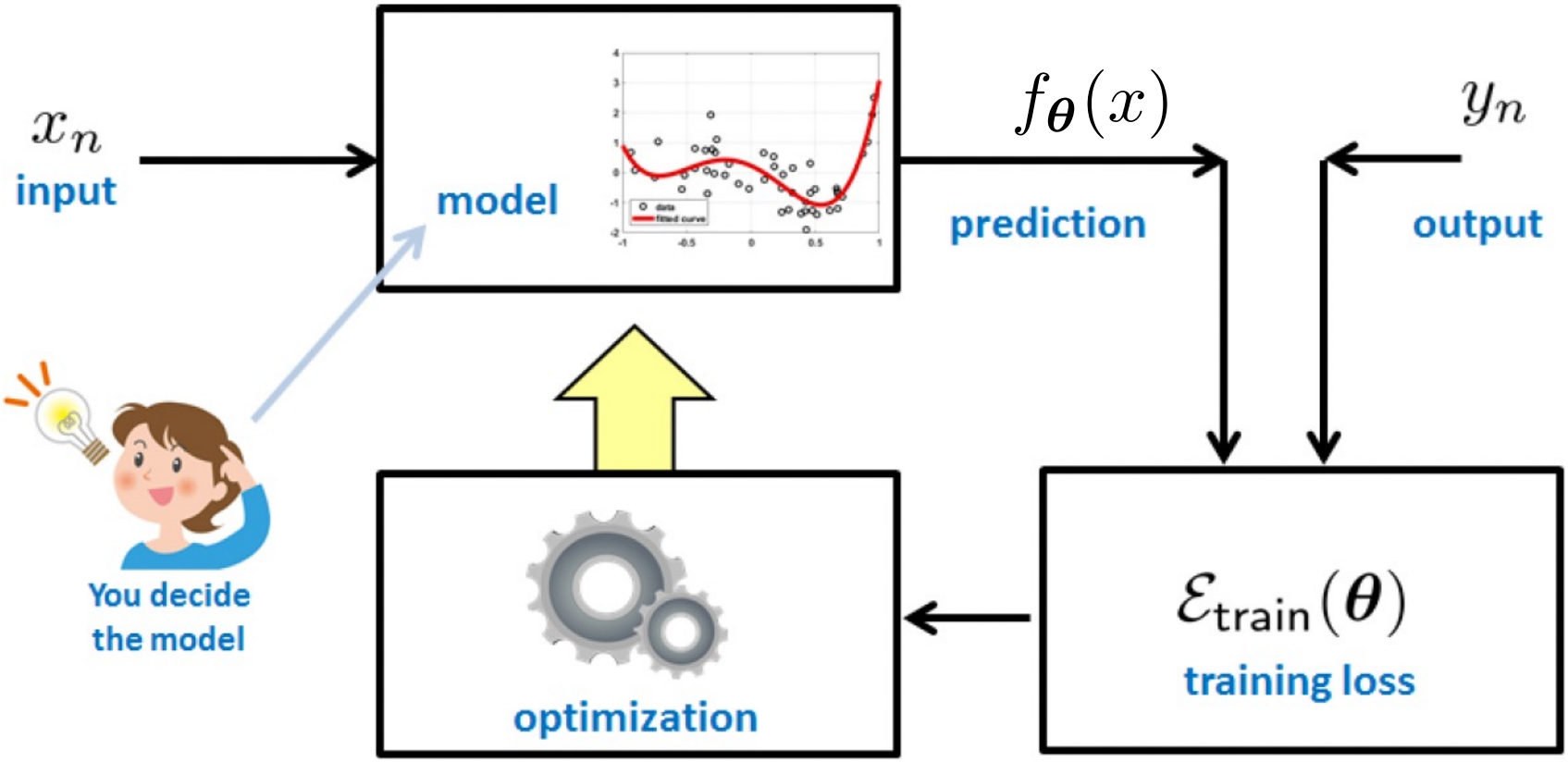
Steps in building a regression model

⑥ Run and interpret the model

- We won't go into the details, but we can use ordinary least squares/maximum likelihood estimation to fit the model
- Essentially, both methods consider a minimization of the sum-square error:

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \mathbb{R}^d}{\operatorname{argmin}} \underbrace{\sum_{n=1}^{N} \left(y_n - f_{\boldsymbol{\theta}} \left(x_n \right) \right)^2}_{\text{training loss } \mathcal{E}_{\text{train}} \left(\boldsymbol{\theta} \right)}$$

with $\boldsymbol{\theta}$ succinctly denoting all the unknown parameters of the model (e.g. coefficients, but also variances in MLE, etc).

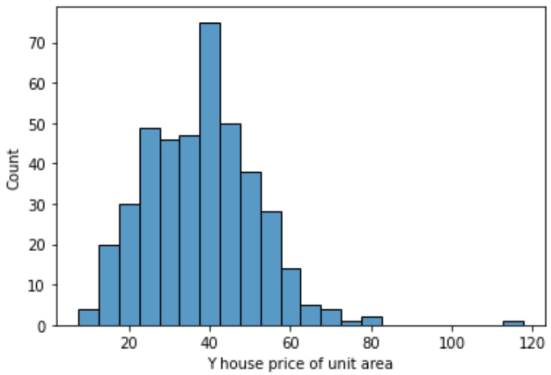

x_n
input
model
$f_{\boldsymbol{\theta}}(x)$
prediction
y_n
output
You decide
the model
optimization
$\mathcal{E}_{\text{train}}(\boldsymbol{\theta})$
training loss

Real Estate Evaluation Data

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt

df=pd.read_csv('Realestate.csv')
df.head()
```

	No	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude	Y house price of unit area
0	1	2012.917	32.0	84.87882	10	24.98298	121.54024	37.9
1	2	2012.917	19.5	306.59470	9	24.98034	121.53951	42.2
2	3	2013.583	13.3	561.98450	5	24.98746	121.54391	47.3
3	4	2013.500	13.3	561.98450	5	24.98746	121.54391	54.8
4	5	2012.833	5.0	390.56840	5	24.97937	121.54245	43.1

```
import seaborn as sns
sns.histplot(data=df, x="Y house price of unit area")
```



Click here for some cute EDA

Original Data Source: Yeh, I. C., & Hsu, T. K. (2018). Building real estate valuation models with comparative approach through case-based reasoning. Applied Soft Computing, 65, 260-271

Steps in building a regression model

Example (Real Estate Evaluation)

We consider the following model

$$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \beta_3 X_{3i} + \varepsilon_i$$

where

Y: house price of unit area

X_1 : distance from MRT station

X_2: number of convenience stores

X_3: house age

The statsmodel library is used in Python for building statistical models.
To estimate the regression coefficient β_0, a constant term of 1 needs to be added as a separate column (intercept).

```
import numpy as np # linear algebra
import statsmodels.api as sm
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import matplotlib.pyplot as plt

df=pd.read_csv('Realestate.csv')

import statsmodels.api as sm
## build design matrix with covariates
## of interest
X = df[["X3 distance to the nearest MRT station", "X4 number of convenience stores", "X2 house age"]]
X = sm.add_constant(X)
X.head(5)
```

	const	X3 distance to the nearest MRT station	X4 number of convenience stores	X2 house age
0	1.0	84.87882	10	32.0
1	1.0	306.59470	9	19.5
2	1.0	561.98450	5	13.3
3	1.0	561.98450	5	13.3
4	1.0	390.56840	5	5.0

You may get a warning now using sm.add_constant – you can safely ignore it!

```
# creating outcome variable
Y = df['Y house price of unit area']

print(len(Y))
```

```
414
```

```
# the number of observation is large enough
# to consider splitting the model into
# train and test features

# train_X contains X features of the training set.
# train_y contains the values of response variable for the training set.
# test_X contains X features of the test set.
# test_y contains the values of response variable for the test set.

from sklearn.model_selection import train_test_split
train_X, test_X, train_Y, test_Y = train_test_split( X, Y,
train_size= 0.8,
random_state = 1234)

# train_size = 0.8 implies 80% of the data is used
# for training the model
# and the remaining 20% is used for
# validating the model.
# random_state = seed
```

```
# We fit the model using sm.OLS
# OLS (Ordinary Least Squares)
# minimizes the difference between the observation
# and the prediction

real_estate_lm=sm.OLS(train_Y, train_X).fit()
real_estate_lm.summary()
```

The output is organized in multiple sections:

Dep. Variable:	Y house price of unit area	**R-squared:**	0.580
Model:	OLS	**Adj. R-squared:**	0.576
Method:	Least Squares	**F-statistic:**	150.5
Date:	Wed, 17 Aug 2022	**Prob (F-statistic):**	2.77e-61
Time:	07:47:36	**Log-Likelihood:**	-1182.9
No. Observations:	331	**AIC:**	2374.
Df Residuals:	327	**BIC:**	2389.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	41.6149	1.474	28.233	0.000	38.715	44.515
X3 distance to the nearest MRT station	-0.0054	0.000	-11.371	0.000	-0.006	-0.004
X4 number of convenience stores	1.3750	0.208	6.625	0.000	0.967	1.783
X2 house age	-0.2017	0.042	-4.753	0.000	-0.285	-0.118

Omnibus:	48.005	**Durbin-Watson:**	2.134
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	110.958
Skew:	0.725	**Prob(JB):**	8.05e-25
Kurtosis:	5.438	**Cond. No.**	5.27e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.27e+03. This might indicate that there are strong multicollinearity or other numerical problems.

Resources to understand output of the model

- I have compiled for you a few online resources that provide a (reasonably good and exhaustive) description of the results of the model:

1. Toward DS Blog: Simple Explanation of Statsmodel Linear Regression Model Summary
2. GeeksforGeeks: Interpreting the results of Linear Regression using OLS Summary
3. Ordinary Least Squares in Python

- You may also find some other examples of regression analysis to look at those links

Steps in Building a Regression Model

⑦ Perform Model Diagnostics

👉 It is important to validate the regression model to ensure its validity and goodness of fit before it can be used for practical applications.

Often, the following steps are used to validate the simple linear regression models (not exhaustive):

- Residual analysis to validate the regression model assumptions.
- Co-efficient of determination (R-squared).
- Hypothesis tests for the regression coefficients.
- Omnibus tests for goodness of fit.
- Finding influential observations

Residual analysis

- Let $\widehat{Y}_i$ represents the fit (if on observed values) or prediction (if used to forecast future values) modeled by the estimated regression equation:

$$\widehat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1\, x_{1i} + \hat{\beta}_2\, x_{2i} + \ldots + \hat{\beta}_p\, x_{pi}$$

for p covariates (predictor) variables, $i = 1, \ldots, N$.

👉 Residuals or errors are the difference between the actual value of the outcome variable and the predicted value

$$\hat{\varepsilon}_i = \left(Y_i - \widehat{Y}_i \right)$$

Residual (error) analysis is important to check whether the assumptions of regression models have been satisfied. It is performed to check the following:

1. The residuals are normally distributed.
2. Variance of residual is constant (homoscedasticity).
3. The functional form of regression is correctly specified.
4. There are no outliers.

Residual plots

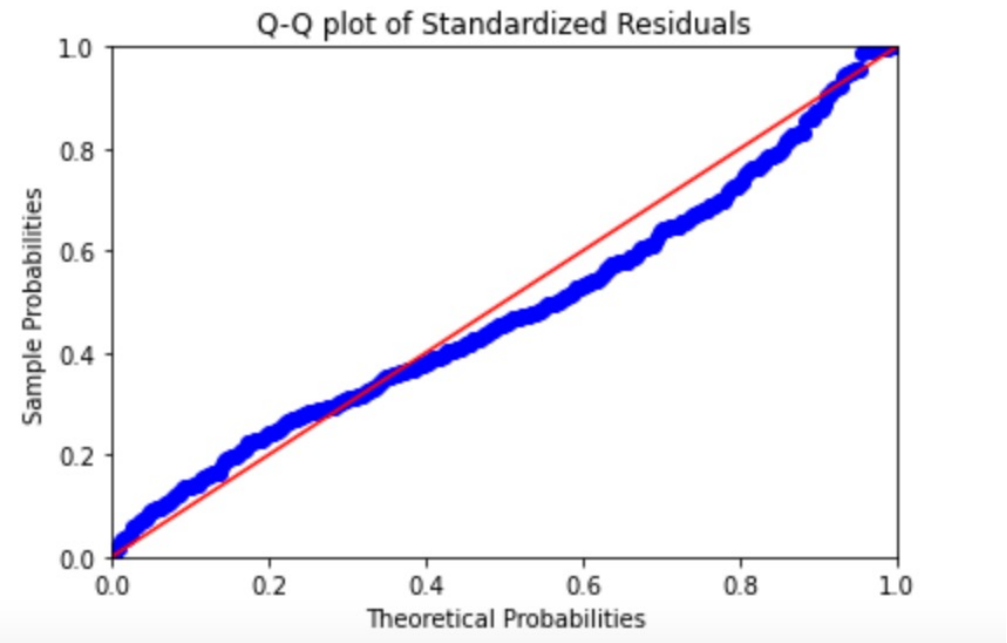
An important assumption of the regression model is that the residuals are normal and have constant variance homoscedasticity) across different values of the predicted value (Y).
To check for these assumptions, we can draw:

- a q-q plot for the normality assumption
- A residual plot, which is a plot between standardized residual value and standardized predicted value to test homoscedasticity.

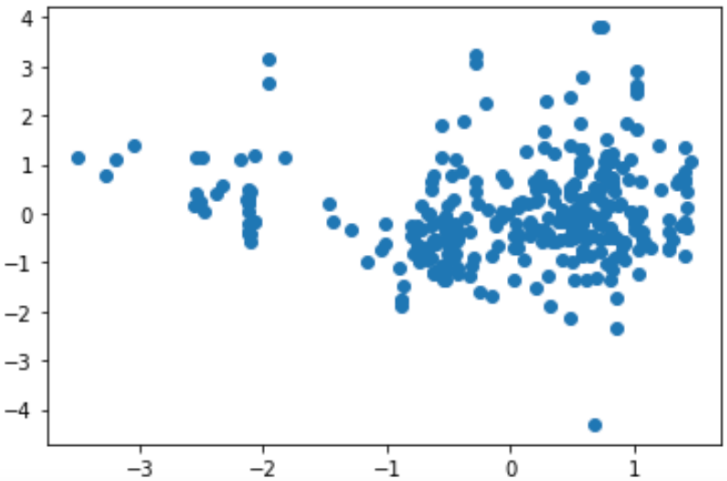
  If the constant variance assumption is adequate, a well-behaved plot will bounce randomly and form a roughly horizontal band around the residual = 0 line. And, no data points will stand out from the basic random pattern of the other residuals.

  If there is heteroscedasticity (non-constant variance of residuals), then a funnel type shape in the residual plot can be expected.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import scipy.stats as stats
import statsmodels.api as sm

qqplot=sm.ProbPlot(standardized_residuals)
qqplot.ppplot(line="45")
plt.title("Q-Q plot of Standardized Residuals")
plt.show()
```



```
# The following custom method get_standardized_values ()
# creates the standardized values of
# a series of values (variable).
# It subtracts values from mean and divides
# by standard deviation of
# the variable.

def get_standardized_values ( vals ) :
    return (vals - vals.mean())/vals.std()

standardized_predicted_values=get_standardized_values(real_estate_lm.fittedvalues)
standardized_residuals=get_standardized_values(real_estate_lm.resid)

# creating regression plots
plt.scatter(standardized_predicted_values,standardized_residuals)
plt.title("Residual Plot vs Fitted values (standardized)")
plt.show()
```



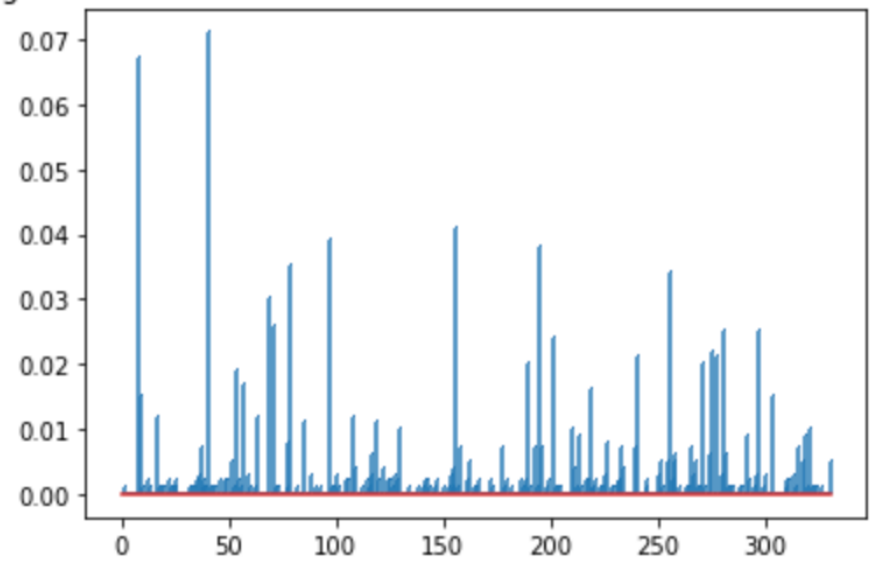
Influential variables: Cook's Distance

- Cook's distance measures how much the predicted value of the dependent variable changes for all the observations in the sample when a particular observation is excluded from the sample for the estimation of regression parameters.
- A Cook's distance value of more than 1 indicates highly influential observation.

Influential variables: Cook's Distance

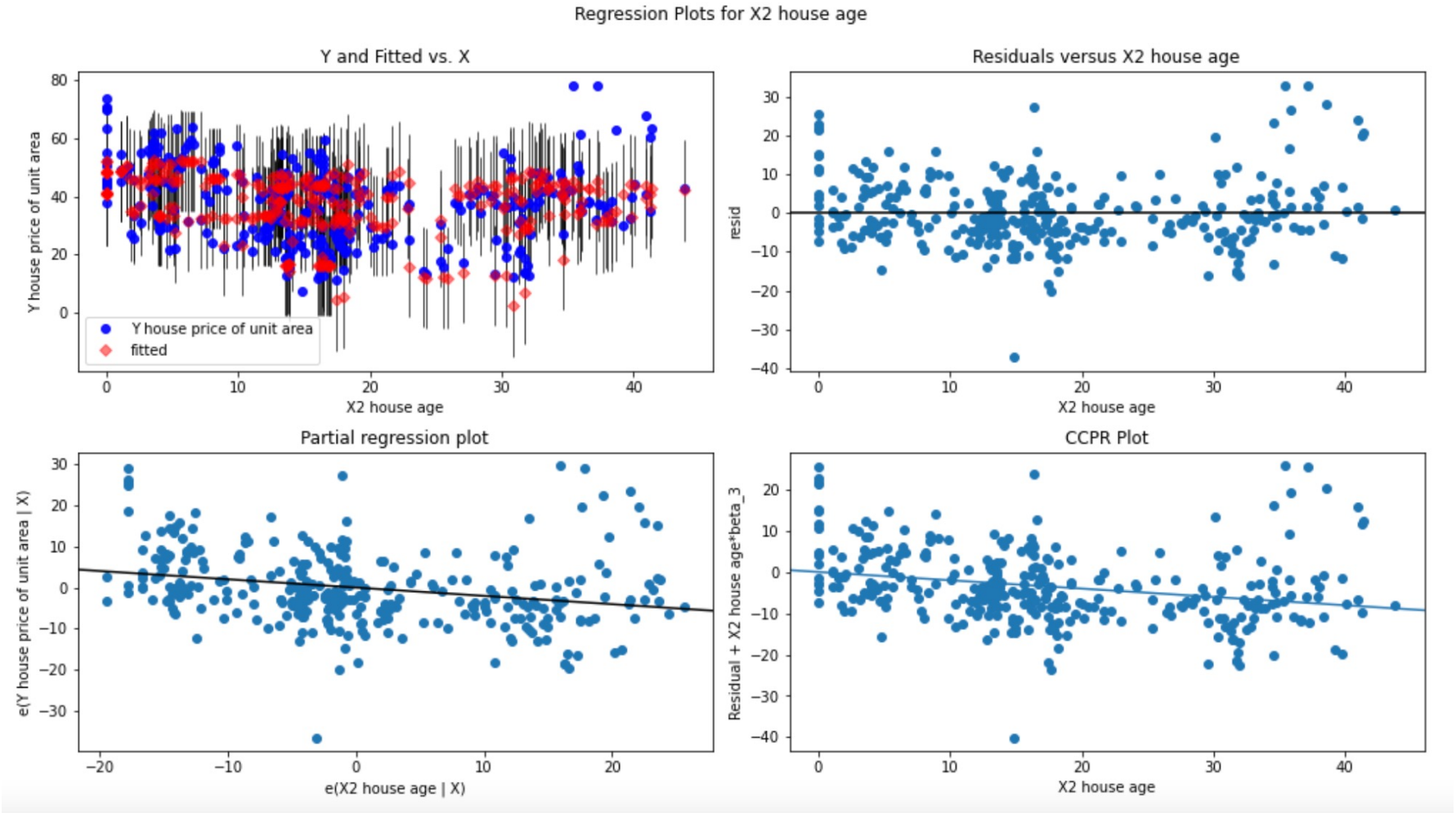
```
import numpy as np

real_estate_influence = real_estate_lm.get_influence()
(c, p) = real_estate_influence.cooks_distance

plt.stem(np.arange( len( train_X)),  np.round( c, 3),  markerfmt=",");
plt.title( "Figure 4.3 - Cooks distance for all observations in Real Estate Data set");
```

Figure 4.3 - Cooks distance for all observations in Real Estate Data set



Residual plots vs predictors

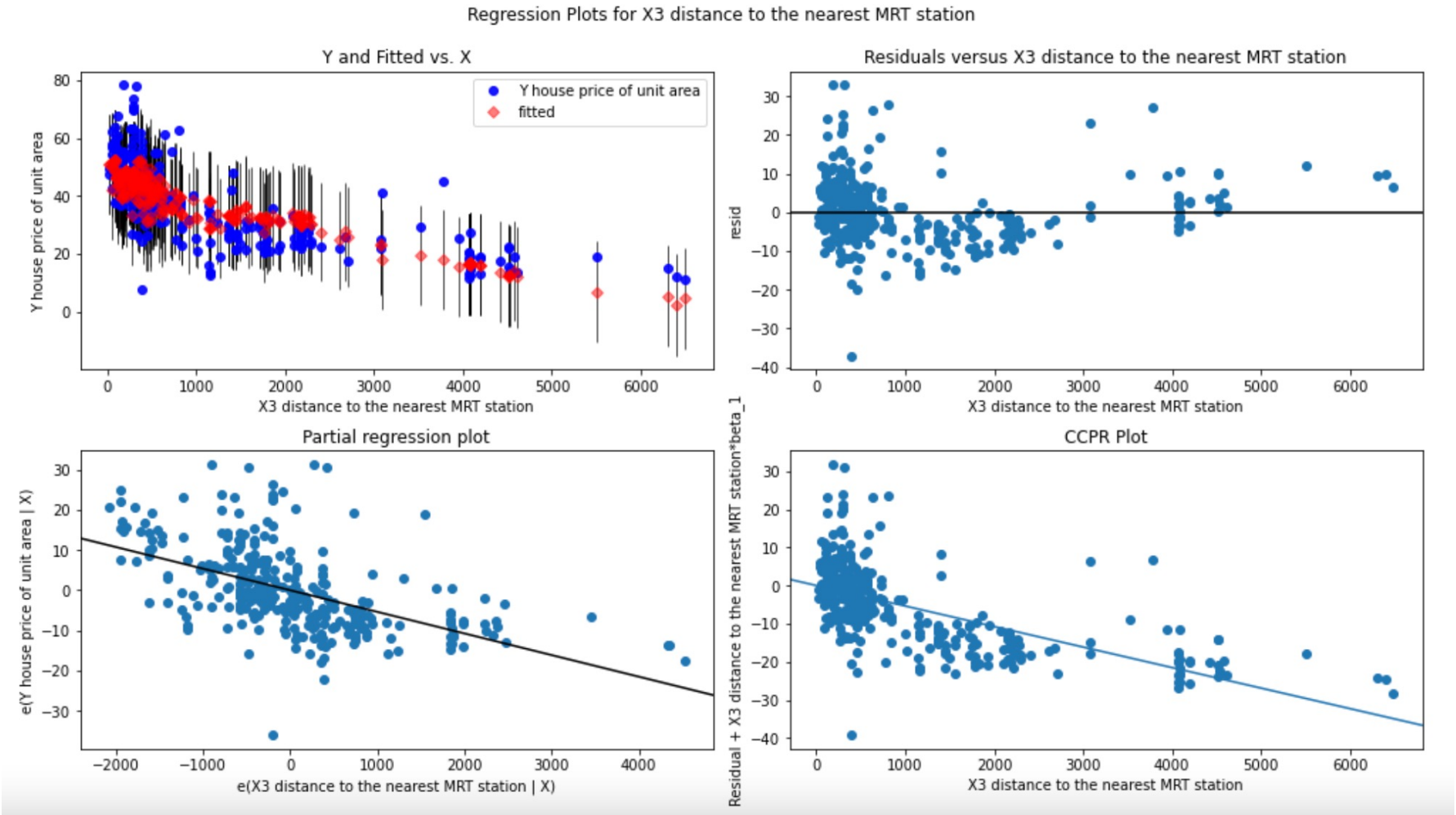
```
fig = plt.figure(figsize=(14, 8))
fig=sm.graphics.plot_regress_exog(real_estate_lm,
                                  'X2 house age', fig=fig)
```



Some explanation here

Residual plots vs predictors

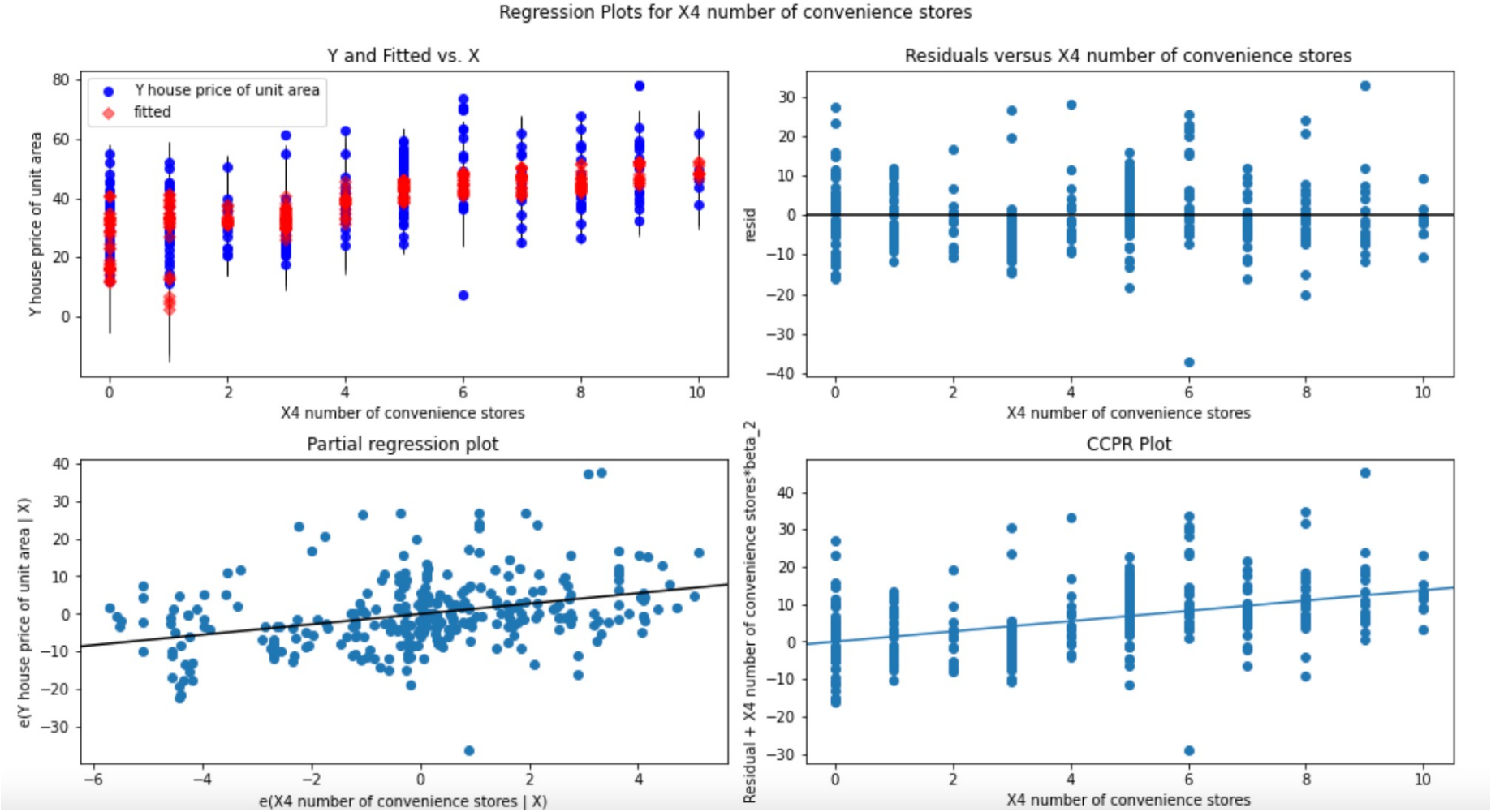
```
fig = plt.figure(figsize=(14, 8))
fig=sm.graphics.plot_regress_exog(real_estate_lm,
                                  'X3 distance to the nearest MRT station', fig=fig)
```



Some explanation here

Residual plots vs predictors

```
fig = plt.figure(figsize=(14, 8))
fig=sm.graphics.plot_regress_exog(real_estate_lm,
                                  'X4 number of convenience stores', fig=fig)
```

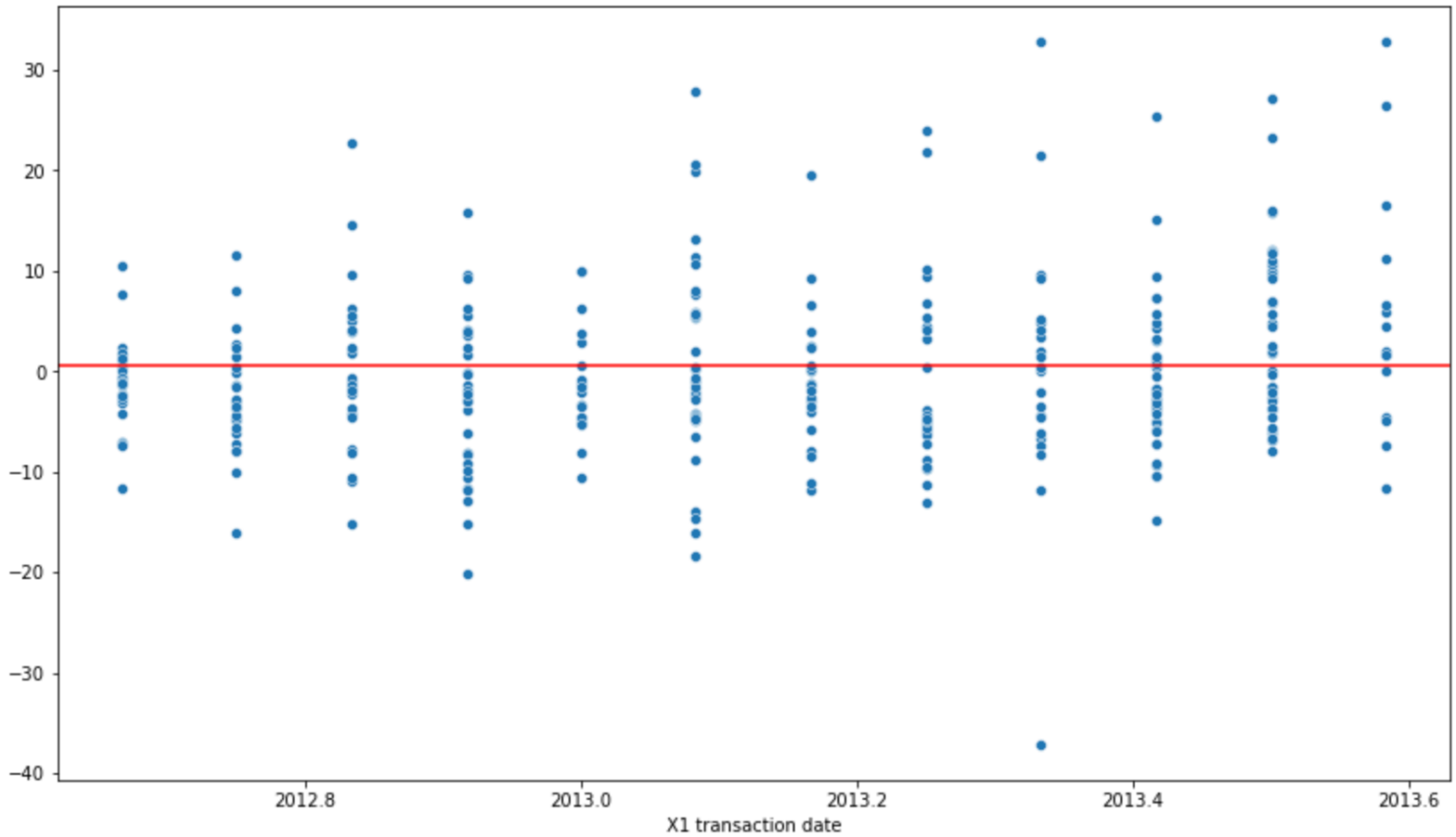


Some explanation here

Residual plots vs predictors

```
fig = plt.figure(figsize=(14, 8))
sns.scatterplot(y=real_estate_lm.resid, x=df["X1 transaction date"])
plt.axhline(y=0.5, color='r', linestyle='-')
```

```
<matplotlib.lines.Line2D at 0x7f9c10529af0>
```



Steps in Building a Regression model

⑧ Making Predictions and Measuring Accuracy

- We can look at the prediction made on the validation (or test) data and the accuracy of prediction.
- The model variable has a method predict(), which takes the X parameters and returns the predicted values.
- We can look at some metrics for the prediction:
  - ★ R-Squared represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model. We want it closer to 1 as possible.
  - ★ $RMSE$ of the prediction : $\sqrt{(Y - \hat{Y}_{pred})^2}$

```
pred_Y = real_estate_lm.predict(test_X)

from sklearn.metrics import r2_score, mean_squared_error

np.abs(r2_score(test_Y, pred_Y))

# the model explains 40% of the variance in the validation set
```

```
0.40212213055141119
```

```
import numpy

np.sqrt(mean_squared_error(test_Y, pred_Y))

# RMSE means the average error the model makes in predicting the outcome.
# The smaller the value of  RMSE, the better the model is.
```

```
11.317783679283815
```

Calculating Prediction Intervals

The regression equation gives us the point estimate of the outcome variable for a given value of the independent variable.

In many applications, we would be interested in knowing the interval estimate of Y_i for a given value of explanatory variable.

The function

`wls_prediction_std()`

returns the prediction interval while making a prediction.

It takes significance value (α) to calculate the interval.

An α-value of 0.1 returns the prediction at confidence interval of 90%.

```
# different way to get the predictions, with more info
sm_pred = real_estate_lm.get_prediction(sm.add_constant(test_X)).summary_frame(alpha=0.1)
sm_pred.head()
```

	mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower	obs_ci_upper
101	39.513083	1.104122	37.691804	41.334362	25.080188	53.945978
181	49.189110	0.843148	47.798315	50.579905	34.804197	63.574022
64	32.725680	1.017715	31.046931	34.404429	18.310077	47.141282
149	44.611243	1.039763	42.896127	46.326360	30.191361	59.031126
270	39.452383	1.085690	37.661508	41.243258	25.023294	53.881473

```
# Let's examine what percentage
# of target values in the test data
# were within the prediction intervals

testing = pd.DataFrame({'obs': test_Y , 'ci_lower': sm_pred['obs_ci_lower'], 'ci_upper': sm_pred['obs_ci_upper']})
testing.describe()
```

	obs	ci_lower	ci_upper
count	83.000000	83.000000	83.000000
mean	38.439759	23.325634	52.135837
std	14.726084	9.673607	9.631401
min	15.500000	-7.686398	21.618364
25%	28.500000	17.896967	46.720935
50%	38.500000	24.413189	53.317288
75%	46.650000	30.050495	58.806450
max	117.500000	37.884160	66.742789

```
testing.loc[:,"obs"]
```

```
101     32.9
181     55.9
64      25.3
149     39.7
270    117.5
       ...
108     34.1
138     43.5
221     38.5
207     34.2
393     40.3
Name: obs, Length: 83, dtype: float64
```

```
percent_correct = np.mean(
    testing.loc[:,"obs"].between(testing.loc[:,"ci_lower"], testing.loc[:,"ci_upper"])
)
print(f"{percent_correct:.2%} of the prediction intervals contain true target.")
```

```
95.18% of the prediction intervals contain true target.
```

Alternative modeling....

Alternative modeling....

```
### Alternative modeling

import numpy as np # linear algebra
import statsmodels.api as sm
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import matplotlib.pyplot as plt

df=pd.read_csv('Realestate.csv')

import statsmodels.api as sm

## create new variables for X3
## (distance from nearest MRT station)
## we introduce a dummy variable
## -> two different slopes
## depending if the distance is
## less than 1500 or greater than 1500

## a variable for  distance less <= 1500
df["Distance_MRT_less_than_1500"] = (df["X3 distance to the nearest MRT station"] <= 1500) * 1

## the previous one returns 0 and 1 (dummy)
## could work, it gives categorical value only
## but we want still to consider the actual measurements
## not just the info about < or >= 1500 meters
## so we do

df["Distance_MRT_less_than_1500"] = (df["X3 distance to the nearest MRT station"] <= 1500) * \
                                     df["X3 distance to the nearest MRT station"]

## See how the results change

## a variable for  distance less > 1500
df["Distance_MRT_more_than_1500"] = (df["X3 distance to the nearest MRT station"] > 1500) * 1

## similar considerations

df["Distance_MRT_more_than_1500"] = (df["X3 distance to the nearest MRT station"] > 1500) * \
df["X3 distance to the nearest MRT station"]
```

```
## Let's consider a variable for time
## we are going to consider a quadratic function
## of time

## the transaction date is expressed as a fraction
## of the year, more precisely the months are represented
## as a fraction of a year.
## So, for instance, June is the 6th month of the year,
## hence it is 6/12 = 0.5
## corresponding to 2013.500

## the following code is simply to represent the
## weird notation as continuous time

print(df['X1 transaction date'].dtype)
## changing to string vector
df['String time'] = df['X1 transaction date'].astype('string')
print(df['String time'].dtype)

## changing the cells content
## considering only the last part
## adding 1 to cells starting with 2013

df[['String time']]=df[['String time']].apply(lambda s: s.str.replace('2012.', '', regex=True))
df[['String time']]=df[['String time']].apply(lambda s: s.str.replace('2013.', '1', regex=True))
#print(df[['String time']])

## changing back to float
## changing to string vector
df['time'] = df['String time'].astype('float')
print(df['time'].dtype)

## Express time as fraction of the year
df['time']=df['time']/1000
print(df['time'])

df['time_squared']=np.square(df['time'])
```

```
## build design matrix with covariates
## of interest
X = df[['time', 'time_squared', "X3 distance to the nearest MRT station", "X4 number of convenience stores",
        "X2 house age", 'Distance_MRT_less_than_1500', 'Distance_MRT_more_than_1500']]
X = sm.add_constant(X)
X.head(5)
```

	const	time	time_squared	X3 distance to the nearest MRT station	X4 number of convenience stores	X2 house age	Distance_MRT_less_than_1500	Distance_MRT_more_than_1500
0	1.0	0.917	0.840889	84.87882	10	32.0	84.87882	0.0
1	1.0	0.917	0.840889	306.59470	9	19.5	306.59470	0.0
2	1.0	1.583	2.505889	561.98450	5	13.3	561.98450	0.0
3	1.0	0.015	0.000225	561.98450	5	13.3	561.98450	0.0
4	1.0	0.833	0.693889	390.56840	5	5.0	390.56840	0.0

```
from sklearn.model_selection import train_test_split
train_X, test_X, train_Y, test_Y = train_test_split( X, Y,
train_size= 0.8,
random_state = 1234)
real_estate_lm=sm.OLS(train_Y, train_X).fit()
real_estate_lm.summary()
```

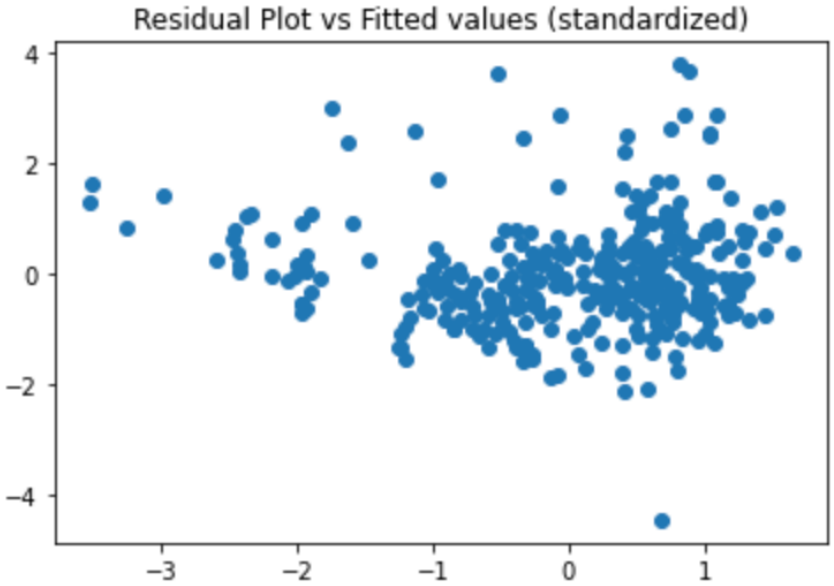
OLS Regression Results

Dep. Variable:	Y house price of unit area	**R-squared:**	0.604
Model:	OLS	**Adj. R-squared:**	0.597
Method:	Least Squares	**F-statistic:**	82.51
Date:	Thu, 18 Aug 2022	**Prob (F-statistic):**	2.76e-62
Time:	18:34:58	**Log-Likelihood:**	-1173.0
No. Observations:	331	**AIC:**	2360.
Df Residuals:	324	**BIC:**	2387.
Df Model:	6		
Covariance Type:	nonrobust		

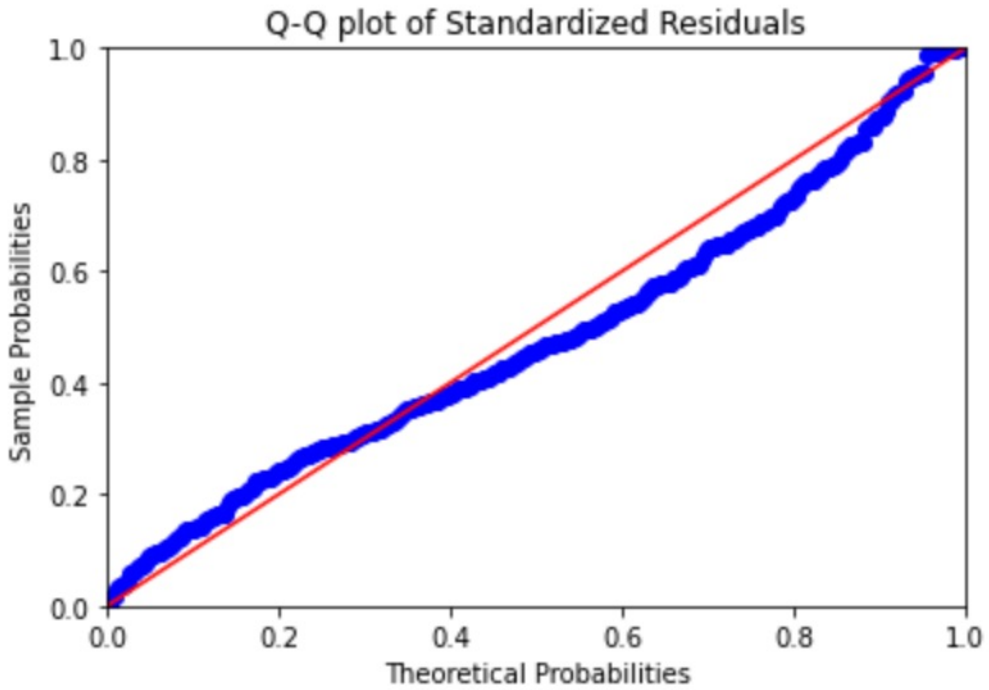
	coef	std err	t	P>|t|	[0.025	0.975]
const	47.0902	1.959	24.036	0.000	43.236	50.944
time	-7.9123	3.232	-2.448	0.015	-14.270	-1.555
time_squared	5.4068	2.223	2.432	0.016	1.033	9.780
X3 distance to the nearest MRT station	-0.0061	0.001	-8.680	0.000	-0.008	-0.005
X4 number of convenience stores	1.0023	0.222	4.521	0.000	0.566	1.439
X2 house age	-0.1748	0.042	-4.164	0.000	-0.257	-0.092
Distance_MRT_less_than_1500	-0.0058	0.001	-5.579	0.000	-0.008	-0.004
Distance_MRT_more_than_1500	-0.0003	0.000	-0.778	0.437	-0.001	0.001

Omnibus:	50.496	**Durbin-Watson:**	2.092
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	128.369
Skew:	0.727	**Prob(JB):**	1.33e-28
Kurtosis:	5.682	**Cond. No.**	1.13e+16

```python
def get_standardized_values ( vals ) :
    return (vals - vals.mean())/vals.std()

standardized_predicted_values=get_standardized_values(real_estate_lm.fittedvalues)
standardized_residuals=get_standardized_values(real_estate_lm.resid)

# creating regression plots
plt.scatter(standardized_predicted_values,standardized_residuals)
plt.title("Residual Plot vs Fitted values (standardized)")
plt.show()
```



```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import scipy.stats as stats
import statsmodels.api as sm

qqplot=sm.ProbPlot(standardized_residuals)
qqplot.ppplot(line="45")
plt.title("Q-Q plot of Standardized Residuals")
plt.show()
```



```
pred_Y = real_estate_lm.predict(test_X)

from sklearn.metrics import r2_score, mean_squared_error

np.abs(r2_score(test_Y, pred_Y))
```

```
0.4157775032891291
```

```
# different way to get the predictions, with more info
sm_pred = real_estate_lm.get_prediction(sm.add_constant(test_X)).summary_frame(alpha=0.1)
sm_pred.head()
```

	mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower	obs_ci_upper
101	41.005013	1.404271	38.688570	43.321456	26.854707	55.155319
181	48.803958	0.900602	47.318352	50.289564	34.765715	62.842201
64	31.219008	1.247825	29.160633	33.277383	17.108652	45.329365
149	43.997159	1.255250	41.926536	46.067782	29.885011	58.109307
270	42.253405	1.346003	40.033078	44.473733	28.118516	56.388295

```
testing = pd.DataFrame({'obs': test_Y , 'ci_lower': sm_pred['obs_ci_lower'], 'ci_upper': sm_pred['obs_ci_upper']})
testing.describe()
```

	obs	ci_lower	ci_upper
count	83.000000	83.000000	83.000000
mean	38.439759	23.652055	51.889113
std	14.726084	10.106569	10.047894
min	15.500000	-7.692443	20.984599
25%	28.500000	17.315751	45.560625
50%	38.500000	24.227528	52.508529
75%	46.650000	31.003856	59.124887
max	117.500000	40.690760	69.114796

```
percent_correct = np.mean(
    testing.loc[:,"obs"].between(testing.loc[:,"ci_lower"], testing.loc[:,"ci_upper"])
)
print(f"{percent_correct:.2%} of the prediction intervals contain true target.")
```

```
95.18% of the prediction intervals contain true target.
```

In the end, lots of work but no gain...

Baseball salaries

- We consider a data frame called "Hitters" with 20 variables and 322 observations of major league players.
- We want to predict a baseball player's salary on the basis of various statistics associated with performance in the previous year.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt

df = pd.read_csv("https://raw.githubusercontent.com/kirenz/datasets/master/Hitters.csv")
df.head()
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWalks	League	Division	PutOuts	Assists	Errors	Salary	NewLea
0	293	66	1	30	29	14	1	293	66	1	30	29	14	A	E	446	33	20	NaN	
1	315	81	7	24	38	39	14	3449	835	69	321	414	375	N	W	632	43	10	475.0	
2	479	130	18	66	72	76	3	1624	457	63	224	266	263	A	W	880	82	14	480.0	
3	496	141	20	65	78	37	11	5628	1575	225	828	838	354	N	E	200	11	3	500.0	
4	321	87	10	39	42	30	2	396	101	12	48	46	33	N	E	805	40	4	91.5	

Lasso Regression

We can build a model to understand what features of players are influencing their sold price or predict the player's auction prices in future. However, all columns are not features.

Lasso regression relies upon the linear regression model but additionaly performs a so called L1 regularization, to prevent overfitting.

Thus, we can fit a model containing all possible predictors and use lasso to perform variable selection by using a technique that regularizes the coefficient estimates (it shrinks the coefficient estimates towards zero):

$$\widehat{\beta} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} \left(y_i - \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} \right) \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

where
$\lambda = 0 \Rightarrow$ same coefficients as usual linear regression
$\lambda = \infty \Rightarrow$ all coefficients are zero
$\lambda \in (0, \infty) \Rightarrow$ coefficients are in between 0 and the case of usual linear regression

```
#Note that the salary is missing for some of the players:

print(df.isnull().sum())
```

```
AtBat          0
Hits           0
HmRun          0
Runs           0
RBI            0
Walks          0
Years          0
CAtBat         0
CHits          0
CHmRun         0
CRuns          0
CRBI           0
CWalks         0
League         0
Division       0
PutOuts        0
Assists        0
Errors         0
Salary        59
NewLeague      0
dtype: int64
```

```
#We simply drop the missing cases:

df = df.dropna()
```

```
# Encode categorical as dummies

dummies = pd.get_dummies(df[['League', 'Division','NewLeague']])
print(dummies.head())
```

	League_A	League_N	Division_E	Division_W	NewLeague_A	NewLeague_N
1	0	1	0	1	0	1
2	1	0	0	1	1	0
3	0	1	1	0	0	1
4	0	1	1	0	0	1
5	1	0	0	1	1	0

```
#Our label y is:

Y = df['Salary']

#We drop the column with the outcome variable (Salary),
#and categorical columns for which we already created dummy variables:

X_numerical = df.drop(['Salary', 'League', 'Division', 'NewLeague'], axis=1).astype('float64')

# Make a list of numerical features
list_numerical = X_numerical.columns
list_numerical
```

```
Index(['AtBat', 'Hits', 'HmRun', 'Runs', 'RBI', 'Walks', 'Years', 'CAtBat',
       'CHits', 'CHmRun', 'CRuns', 'CRBI', 'CWalks', 'PutOuts', 'Assists',
       'Errors'],
```

```
# Create all features
X = pd.concat([X_numerical, dummies[['League_N', 'Division_W', 'NewLeague_N']]], axis=1)

# split data

from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=1234)
```

Standardization

- Lasso performs best when all numerical features are centered around 0 and have variance in the same order.
- If a feature has a variance that is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

```
## Standardization
## Notice: You do this after splitting
## otherwise it defies the purpose of splitting
## big source of mistakes!

## Also, big note: You do not standardize the
## categorical/dummy variables!


from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(X_train[list_numerical])

X_train[list_numerical] = scaler.transform(X_train[list_numerical])

X_test[list_numerical] = scaler.transform(X_test[list_numerical])

X_train.head()

## Sometimes you may want to standardize the Y variable too
## depends on the problem
## changes the interpretation
## Standardizing Y explictly by subtracting mean and
## dividing by standard deviation
## Y_train=(Y_train-Y_train.mean())/Y_train.std()
## test_Y=(Y_test-Y_test.mean())/Y_test.std()
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWalks	PutOuts	
242	-1.287110	-1.163171	-0.876250	-1.291505	-1.285880	-1.217354	0.981845	0.065155	-0.093905	-0.324129	-0.302885	-0.089830	-0.231998	0.343238	-1
269	-0.812597	-1.015467	-0.988293	-1.177840	-0.732641	-1.171055	0.556263	0.012143	-0.090603	-0.383248	-0.144651	-0.229090	-0.207448	-0.570460	
187	1.770865	2.360631	2.148914	2.383684	1.590963	-0.337679	-0.933274	-0.338479	-0.222689	-0.418720	-0.315802	-0.408581	-0.669798	0.478865	-1
296	-1.359605	-1.437479	-1.100336	-1.215728	-1.322763	-0.985860	-0.082110	-0.837906	-0.830285	-0.690669	-0.822799	-0.724237	-0.751630	-0.584737	-1

Lasso in Python

- First, we apply lasso regression on the training set with an arbitrarily regularization parameter α of 1. (just to see how it works)

```
from sklearn.linear_model import Lasso

reg = Lasso(alpha=1)
lasso_fit=reg.fit(X_train, Y_train)

# The importance of a feature
# is the absolute value of its coefficient

importance = np.abs(lasso_fit.coef_)
print(importance)
```

```
[3.19712930e+02 3.21364360e+02 3.99044558e+01 2.69020780e+01
 6.69081277e+00 9.44264745e+01 6.14637927e+01 2.18933747e+02
 0.00000000e+00 5.30431620e+00 4.84313317e+02 1.13137936e+02
 1.03814824e+02 9.79126380e+01 4.52136394e+01 4.02325968e-01
 2.74613349e+01 1.31585594e+02 1.55668301e+00]
```

```
#The features that survived
#the Lasso regression are:

np.array(X_train.columns)[importance > 0]
```

```
array(['AtBat', 'Hits', 'HmRun', 'Runs', 'RBI', 'Walks', 'Years',
       'CAtBat', 'CHmRun', 'CRuns', 'CRBI', 'CWalks', 'PutOuts',
       'Assists', 'Errors', 'League_N', 'Division_W', 'NewLeague_N'],
      dtype=object)
```

```
## The features that have been discarded are

np.array(X_train.columns)[importance == 0]
```

```
array(['CHits'], dtype=object)
```

Model Evaluation

```
#We print the R2 -score for the training and test set.

print('R squared training set', round(reg.score(X_train, Y_train)*100, 2))
print('R squared test set', round(reg.score(X_test, Y_test)*100, 2))
```

```
R squared training set 56.17
R squared test set 37.69
```

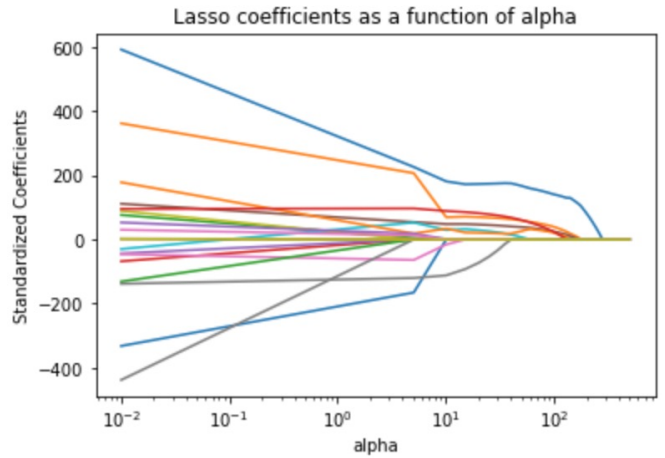
```
#MSE of the prediciton

from sklearn.metrics import mean_squared_error

# Training data
pred_train = reg.predict(X_train)
mse_train = mean_squared_error(Y_train, pred_train)
print('MSE training set', round(mse_train, 2))

# Test data
pred = reg.predict(X_test)
mse_test =mean_squared_error(Y_test, pred)
print('MSE test set', round(mse_test, 2))
```

```
MSE training set 99079.74
MSE test set 90338.93
```

Role of α

To better understand the role of α, we can plot the lasso coefficients as a function of alpha

```
import numpy as np
import matplotlib.pyplot as plt

alphas = np.linspace(0.01,500,100)
lasso = Lasso(max_iter=10000)
coefs = []

for a in alphas:
    lasso.set_params(alpha=a)
    lasso.fit(X_train, Y_train)
    coefs.append(lasso.coef_)

ax = plt.gca()

ax.plot(alphas, coefs)
ax.set_xscale('log')
plt.axis('tight')
plt.xlabel('alpha')
plt.ylabel('Standardized Coefficients')
plt.title('Lasso coefficients as a function of alpha');
```



Lasso with optimal α

To find the optimal value of alpha, we use scikit learns lasso linear model with iterative fitting along a regularization path (LassoCV). The best model is selected by cross-validation.

```
from sklearn.linear_model import LassoCV
```

```
from sklearn.linear_model import LassoCV

# Lasso with 5 fold cross-validation
model = LassoCV(cv=5, random_state=0, max_iter=10000)

# Fit model
model.fit(X_train, Y_train)
```

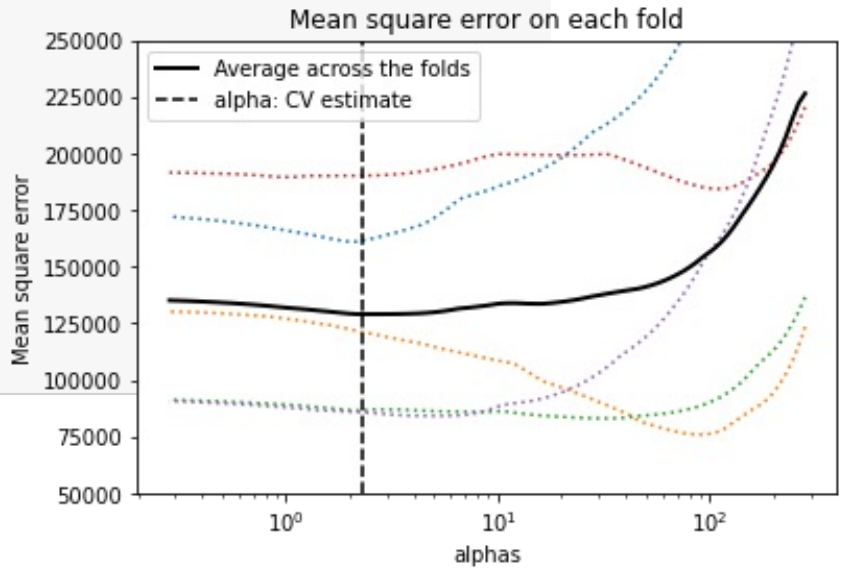
```
LassoCV(cv=5, max_iter=10000, random_state=0)
```

```
# Show best value of penalization
# chosen by cross validation:
model.alpha_
```

```
2.286940792280632
```

Best model

```
# Lasso Path
# plot results of cross-validation
# with mean squared erros

plt.semilogx(model.alphas_, model.mse_path_, ":")
plt.plot(
    model.alphas_ ,
    model.mse_path_.mean(axis=-1),
    "k",
    label="Average across the folds",
    linewidth=2,
)
plt.axvline(
    model.alpha_, linestyle="--", color="k", label="alpha: CV estimate"
)

plt.legend()
plt.xlabel("alphas")
plt.ylabel("Mean square error")
plt.title("Mean square error on each fold")
plt.axis("tight")

ymin, ymax = 50000, 250000
plt.ylim(ymin, ymax);
```



Lasso Path

```
#Use best value for our final model:

# Set best alpha
lasso_best = Lasso(alpha=model.alpha_)
lasso_best.fit(X_train, Y_train)

#Show model coefficients and names:

print(list(zip(lasso_best.coef_, X)))
```

```
[(-284.15273590203395, 'AtBat'), (283.86632961425016, 'Hits'), (21.52103580801291, 'HmRun'), (-0.0, 'Runs'), (0.0, 'R
BI'), (79.28866353851899, 'Walks'), (-78.88525623814488, 'Years'), (-32.66477135218706, 'CAtBat'), (0.0, 'CHits'), (5
5.19829936733721, 'CHmRun'), (319.6893149324783, 'CRuns'), (23.91018168259842, 'CRBI'), (-61.393277642187286, 'CWalk
s'), (99.01764885332823, 'PutOuts'), (34.10233368699217, 'Assists'), (0.0, 'Errors'), (20.269652983191556, 'League_
N'), (-127.96366984225787, 'Division_W'), (2.5384144813099363, 'NewLeague_N')]
```

```
# Model Evaluation
print('R squared training set', round(lasso_best.score(X_train, Y_train)*100, 2))
print('R squared test set', round(lasso_best.score(X_test, Y_test)*100, 2))

mean_squared_error(Y_test, lasso_best.predict(X_test))
```

```
R squared training set 55.36
R squared test set 37.05

91272.28539337942
```

Predicting the Sold Price (Auction Price) of Players

- The Indian Premier League (IPL) is a professional league for Twenty 20 (T20) cricket championships that was started in 2008 in India.

  IPL was initiated by the BCCI with eight franchises comprising players from across the world.

  The first IPL auction was held in 2008 for ownership of the teams for 10 years, with a base price of USD 50 million. The franchises acquire players through an English auction that is conducted every year. However, there are several rules imposed by the IPL. For example, only international players and popular Indian players are auctioned.

  The performance of the players could be measured through several metrics. We have data on the performances of 130 players who played in at least one season of the IPL (2008-2011) measured through various performance metrics.

- Example from the book by Pradhan, Manaranjan; U Dinesh Kumar. Machine Learning using Python (but solved differently)

TABLE 4.3 Metadata of IPL dataset

Data Code	Description
AGE	Age of the player at the time of auction classified into three categories. Category 1 (L25) means the player is less than 25 years old, category 2 means that the age is between 25 and 35 years (B25– 35) and category 3 means that the age is more than 35 (A35).
RUNS-S	Number of runs scored by a player.
RUNS-C	Number of runs conceded by a player.
HS	Highest score by a batsman in IPL.
AVE-B	Average runs scored by a batsman in IPL.
AVE-BL	Bowling average (number of runs conceded/number of wickets taken) in IPL.
SR-B	Batting strike rate (ratio of the number of runs scored to the number of balls faced) in IPL.
SR-BL	Bowling strike rate (ratio of the number of balls bowled to the number of wickets taken) in IPL.
SIXERS	Number of six runs scored by a player in IPL.
WKTS	Number of wickets taken by a player in IPL.

Data Code	Description
ECON	Economy rate of a bowler (number of runs conceded by the bowler per over) in IPL.
CAPTAINCY EXP	Captained either a T20 team or a national team.
ODI-SR-B	Batting strike rate in One-Day Internationals.
ODI-SR-BL	Bowling strike rate in One-Day Internationals.
ODI-RUNS-S	Runs scored in One-Day Internationals.
ODI-WKTS	Wickets taken in One-Day Internationals.
T-RUNS-S	Runs scored in Test matches.
T-WKTS	Wickets taken in Test matches.
PLAYER-SKILL	Player's primary skill (batsman, bowler, or allrounder).
COUNTRY	Country of origin of the player (AUS: Australia; IND: India; PAK: Pakistan; SA: South Africa; SL: Sri Lanka; NZ: New Zealand; WI: West Indies; OTH: Other countries).
YEAR-A	Year of Auction in IPL.
IPL TEAM	Team(s) for which the player had played in the IPL (CSK: Chennai Super Kings; DC: Deccan Chargers; DD: Delhi Daredevils; KXI: Kings XI Punjab; KKR: Kolkata Knight Riders; MI: Mumbai Indians; PWI: Pune Warriors India; RR: Rajasthan Royals; RCB: Royal Challengers Bangalore). A + sign is used to indicate that the player has played for more than one team. For example, CSK+ would mean that the player has played for CSK as well as for one or more other teams.

See code.

Ridge Regression

Ridge Regression

- With high-dimensional data ($n << p$), linear regression becomes very difficult/impractical.
- One major problem is multicollinearity of the predictors.
- Multicollinearity is the existence of a correlation between independent variables in modeled data. It can cause inaccuracy in the regression coefficient estimates.
- It can also magnify the standard errors in the regression coefficients, produce deceiving results and p-values and increase the redundancy of a model, making its predictability inefficient and less reliable.
- Ridge regression provides an L^2 regularization, the objective function being:

$$\sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \alpha \sum_{j=1}^{p} \beta_j^2$$

- This technique works very well to avoid over-fitting issues.