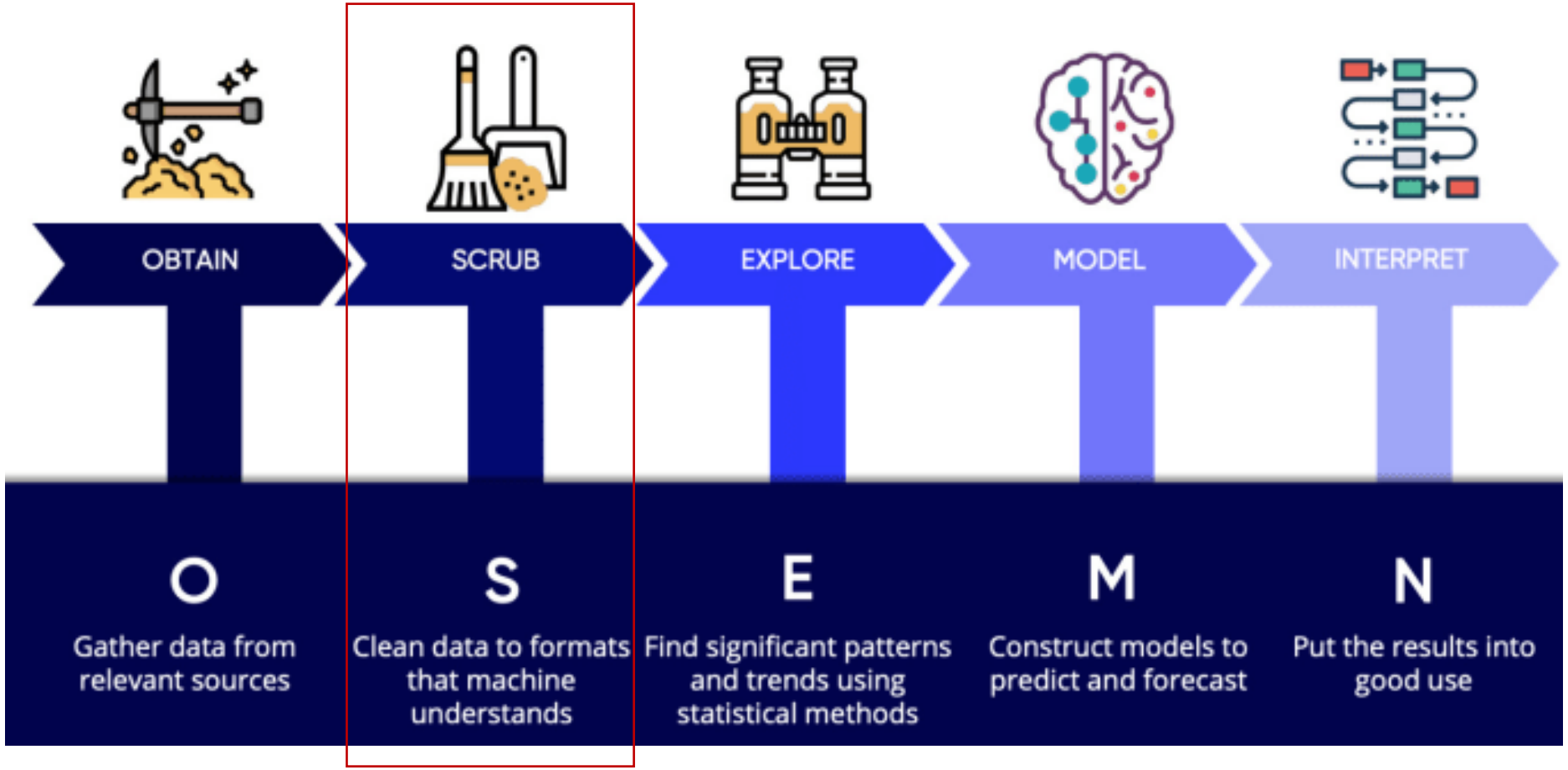
python

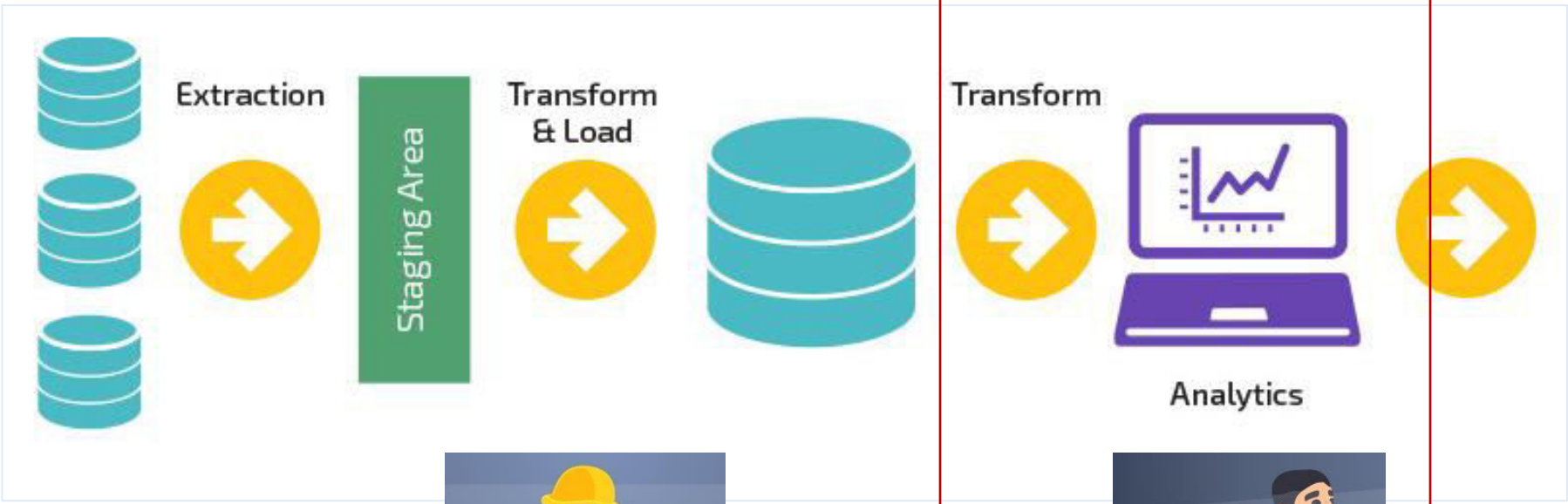
DATA SCIENCE

Data Cleaning Using Python

Data Cleaning Using Python



https://www.fiverr.com/ishant_agg/do-data-science-eda-and-visualisation-with-python-and-r



Data Engineer

Data Analyst

Data Scientist

EDA Objectives

1. Quickly describe a dataset
2. Clean data
3. Visualize data distributions
4. Calculate and visualize correlations

EDA Sub-Tasks

- Gather data
- Clean and prepare data for analysis ✓
- Explore Data
- Manipulate data
- Summarize data
- Visualize Data

Cleans the Data

What are we looking for ??

- Incomplete Data
- Missing Data
- Formatting
- Inaccurate Data

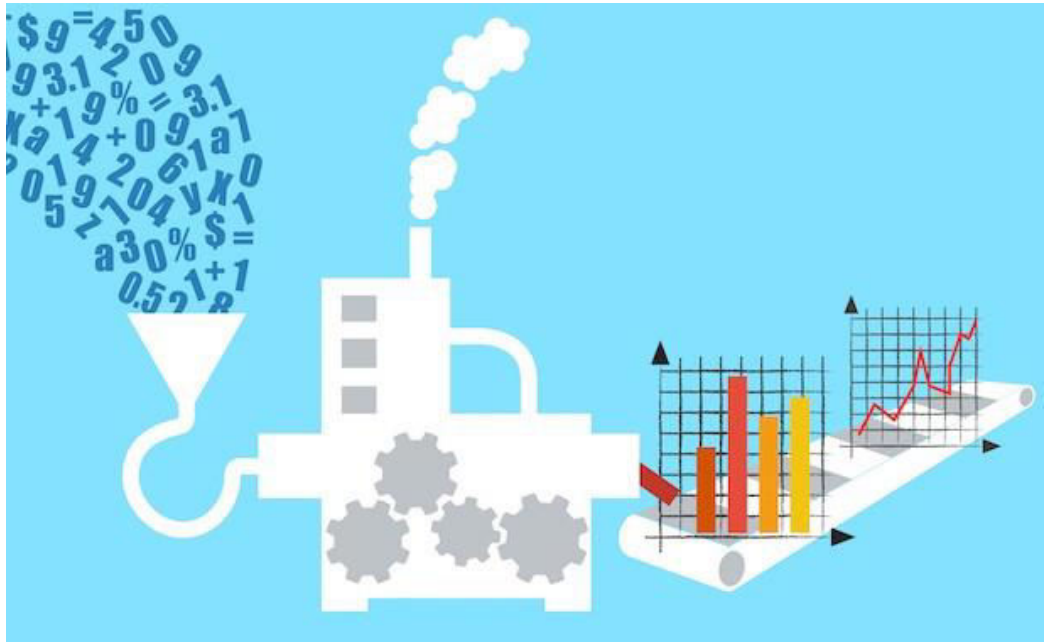
Python Pandas Library

pandas.pydata.org

Python for Data Analysis

Pandas Documentation (API Reference) :
https://pandas.pydata.org/docs/reference/index.html

Importing Dataset



Import Dataset

Using Pandas library, load a data file into a data structure, or container object, known as a Pandas' data frame.

```
[1]: 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 seaborn as sns # data vizualisation

     # to set backend of matplotlib to inline view visual :
     %matplotlib inline

[2]: df = pd.read_csv('dataset/BreadBasket_DMS.csv')
```

pandas data frame type variable

Pandas I/O

- Pandas I/O API is a set of reader and writer functions return a pandas object.
- The following table shows some available readers & writers.

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel

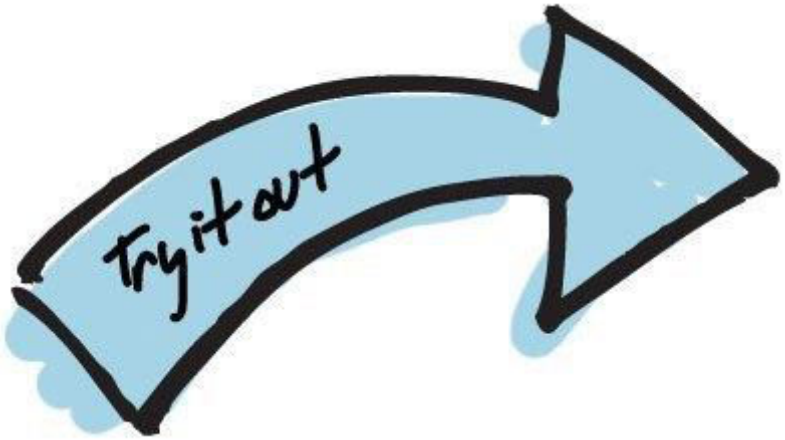
Pandas I/O Reader Usage Example

```
# Returns a DataFrame
pd.read_excel('path_to_file.xls', sheet_name='Sheet1')
```

```
with pd.ExcelFile('path_to_file.xls') as xls:
    df1 = pd.read_excel(xls, 'Sheet1')
    df2 = pd.read_excel(xls, 'Sheet2')
```

Using ExcelFile Class as a context, and load multiple sheet into different dataframe.

Importing Data – Example



- Open the jupyter notebook file for practicing by yourself :

- File : /script/importdata.ipynb
- Use Shift + Enter (or 'Run' button ▶) to run script in a cell.

Import Dataset – read_excel()

Using Pandas read_excel() function, load an online retail dataset in excel (*.xlsx) file into pandas' data frame.

```
[1]: import pandas as pd

[2]: ## ------- import data example-1 ------- ##
     #  import data from ms.excel file
     #  and save it into a pandas dataframe object named df

     df = pd.read_excel("D:\dataset\onlineretail.xlsx")
```

pandas data frame object

Get the New Dataset Overview

Using dataframe info(), simply read data overview.

```
[3]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 207719 entries, 0 to 207718
Data columns (total 8 columns):
InvoiceNo      207719 non-null object
StockCode      207719 non-null object
Description    206886 non-null object
Quantity       207719 non-null int64
InvoiceDate    207719 non-null datetime64[ns]
UnitPrice      207719 non-null float64
CustomerID     148101 non-null float64
Country        207719 non-null object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 12.7+ MB
```

207719 rows, 8 columns

Other data overview

Import Dataset – read_csv()

Load a csv file using pandas read_csv() function.

```
## ------- import data example-2 ------- ##
#  import data from a csv formatted file
#  and save it into a pandas dataframe object named df

filepath = "dataset/fifa.csv"
df = pd.read_csv(filepath)
```

New Dataset Overview

```
[5]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 286 entries, 0 to 285
Data columns (total 7 columns):
Date    286 non-null object
ARG     286 non-null float64
BRA     286 non-null float64
ESP     286 non-null float64
FRA     286 non-null float64
GER     286 non-null float64
ITA     286 non-null float64
dtypes: float64(6), object(1)
memory usage: 15.7+ KB
```

286 rows, 7 columns

Dataset Without Specific Index Column

Index column is column used as row label.

```
[5]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 286 entries, 0 to 285
Data columns (total 7 columns):
Date    286 non-null object
ARG     286 non-null float64
BRA     286 non-null float64
ESP     286 non-null float64
FRA     286 non-null float64
GER     286 non-null float64
ITA     286 non-null float64
dtypes: float64(6), object(1)
memory usage: 15.7+ KB
```

This is an example of dataset has no specific index column. It has a general range-index column.

Dataset With a Specific Index Column

Index column is column used as row label.

This is an example of dataset has specific index column.

Date is column used as row label (index)

```
[3]: df.info()
     df.head(5)

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3650 entries, 1981-01-01 to 1990-12-31
Data columns (total 1 columns):
Temp    3650 non-null object
dtypes: object(1)
memory usage: 57.0+ KB
```

[3]:

	Temp
Date	
1981-01-01	20,7
1981-01-02	17,9
1981-01-03	18,8
1981-01-04	14,6
1981-01-05	15,8

How to Set a Column as an Index?

```
[5]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 286 entries, 0 to 285
Data columns (total 7 columns):
Date    286 non-null object
ARG     286 non-null float64
BRA     286 non-null float64
ESP     286 non-null float64
FRA     286 non-null float64
GER     286 non-null float64
ITA     286 non-null float64
dtypes: float64(6), object(1)
memory usage: 15.7+ KB
```

How to set column ‘Date’ as an index?

1. Set column as index (row label) while importing data using pandas read_() function.

   or

2. Set column as index (row label) using dataframe set_index() function.

1. Set column as index (row label) while importing data

```
[6]: ## ------- import data example-3 ------- ##
     # load a dataset and set a column as an index (row column)
     # using pandas read_csv() function

     filepath = "dataset/fifa.csv"
     df = pd.read_csv(filepath, index_col="Date", parse_dates = True)

     # index_col="Date" :  column used as row label
     # parse_date = True : indicating row label as Date
```

index_col parameter set column(s) used as row label

Dataset Overview : index column

dataset has index column.

```
[7]: df.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 286 entries, 1993-08-08 to 2018-06-07
Data columns (total 6 columns):
ARG    286 non-null float64
BRA    286 non-null float64
ESP    286 non-null float64
FRA    286 non-null float64
GER    286 non-null float64
ITA    286 non-null float64
dtypes: float64(6)
memory usage: 15.6 KB
```

```
[8]: df.head()
```

[8]:

	ARG	BRA	ESP	FRA	GER	ITA
Date						
1993-08-08	5.0	8.0	13.0	12.0	1.0	2.0
1993-09-23	12.0	1.0	14.0	7.0	5.0	2.0
1993-10-22	9.0	1.0	7.0	14.0	4.0	3.0

Date is column used as row label (index)

2. Set column as index using dataframe set_index() function.

```
[9]: ## ------- import data example-4 ------- ##
     #  import data from a csv formatted file
     #  and set a column as an index using set_index() function

     filepath = "dataset/fifa.csv"
     df = pd.read_csv(filepath)
```

```
[10]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 286 entries, 0 to 285
Data columns (total 7 columns):
```

This dataframe has no specific index column

Set column as index using set_index() function.

```
df.set_index('Date', inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 286 entries, 1993-08-08 to 2018-06-07
Data columns (total 6 columns):
ARG    286 non-null float64
BRA    286 non-null float64
ESP    286 non-null float64
FRA    286 non-null float64
GER    286 non-null float64
ITA    286 non-null float64
dtypes: float64(6)
memory usage: 15.6+ KB
```

This dataframe has specific index column.

Dataset Overview : index column

dataset has index column.

```
[7]: df.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 286 entries, 1993-08-08 to 2018-06-07
Data columns (total 6 columns):
ARG    286 non-null float64
BRA    286 non-null float64
ESP    286 non-null float64
FRA    286 non-null float64
GER    286 non-null float64
ITA    286 non-null float64
dtypes: float64(6)
memory usage: 15.6 KB
```

```
[8]: df.head()
```

[8]:

	ARG	BRA	ESP	FRA	GER	ITA
Date						
1993-08-08	5.0	8.0	13.0	12.0	1.0	2.0
1993-09-23	12.0	1.0	14.0	7.0	5.0	2.0
1993-10-22	9.0	1.0	7.0	14.0	4.0	3.0

Date is column used as row label (index)

Understanding the Data Overview

Check the Data Overview

Objectives :

- Get a better look of the data
- Show data description
- Comparing multiple dataset
- Find inconsistent data dimension

Get Data Overview

- Get Data Overview using pandas head() or tail().

```
[4]: #dataset overview
     df.head(3)
```

default parameter = 5.
df.head() is same as df.head(5)

[4]:

	Date	Time	Transaction	Item
0	2016-10-30	09:58:11	1	Bread
1	2016-10-30	10:05:34	2	Scandinavian
2	2016-10-30	10:05:34	2	Scandinavian

Get Data Info

- Get Data Overview using pandas info().

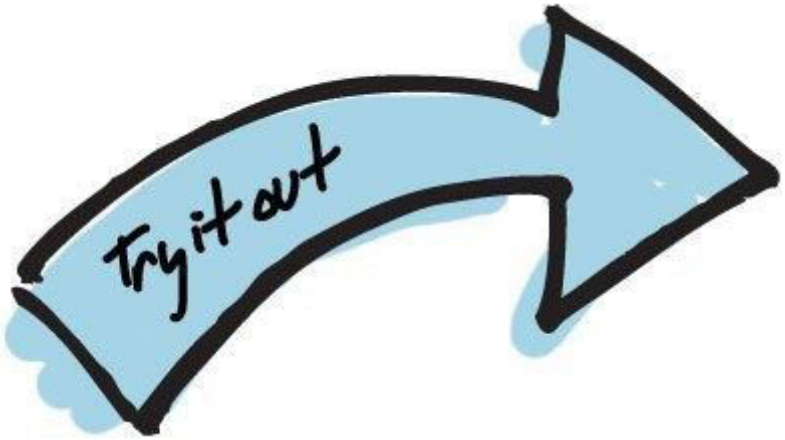
```
[5]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21293 entries, 0 to 21292
Data columns (total 4 columns):
Date           21293 non-null object
Time           21293 non-null object
Transaction    21293 non-null int64
Item           21293 non-null object
dtypes: int64(1), object(3)
memory usage: 665.5+ KB
```

21293 rows, 4 columns

Other data overview

Get Data Overview – Example



- Open the jupyter notebook file for practicing by yourself :

- File : /script/dataoverview.ipynb
- Use Shift + Enter (or 'Run' button▶) to run script in a cell.

Data Overview – Example1

```
[2]: ## ------- data overview example-1 ------- ##
     #  import data from a csv flight delays dataset

     filepath = "dataset/flight_delays.csv"
     df = pd.read_csv(filepath, index_col = 'Month')
```

```
[3]: df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 12 entries, 1 to 12
Data columns (total 14 columns):
AA    12 non-null float64
AS    12 non-null float64
B6    12 non-null float64
DL    12 non-null float64
EV    12 non-null float64
F9    12 non-null float64
HA    12 non-null float64
MQ    12 non-null float64
NK    12 non-null float64
OO    12 non-null float64
UA    12 non-null float64
US    6 non-null float64
VX    12 non-null float64
WN    12 non-null float64
```

incomplete or missing data

Display Data Preview using head() or tail()

```
df.tail()
```

Month	AA	AS	B6	DL	EV	F9	HA	MQ	NK	OO	UA	US	VX
8	3.193907	2.503899	9.280950	0.653114	5.154422	9.175737	7.448029	1.896565	20.519018	5.606689	5.014041	NaN	5.106221
9	-1.432732	-1.813800	3.539154	-3.703377	0.851062	0.978460	3.696915	-2.167268	8.000101	1.530896	-1.794265	NaN	0.070998
10	-0.580930	-2.993617	3.676787	-5.011516	2.303760	0.082127	0.467074	-3.735054	6.810736	1.750897	-2.456542	NaN	2.254278
11	0.772630	-1.916516	1.418299	-3.175414	4.415930	11.164527	-2.719894	0.220061	7.543881	4.925548	0.281064	NaN	0.116370
12	4.149684	-1.846681	13.839290	2.504595	6.685176	9.346221	-1.706475	0.662486	12.733123	10.947612	7.012079	NaN	13.498720

some rows has incomplete data on 'US' flight code.

Which Rows (Months) has Incomplete Data?

```
[7]: df[df['US'].isnull()]
```

[7]:

Month	AA	AS	B6	DL	EV	F9	HA	MQ	NK	OO	UA	US	VX
7	3.870440	0.377408	5.841454	1.204862	6.926421	14.464543	2.001586	3.980289	14.352382	6.790333	10.262551	NaN	7.135773
8	3.193907	2.503899	9.280950	0.653114	5.154422	9.175737	7.448029	1.896565	20.519018	5.606689	5.014041	NaN	5.106221
9	-1.432732	-1.813800	3.539154	-3.703377	0.851062	0.978460	3.696915	-2.167268	8.000101	1.530896	-1.794265	NaN	0.070998
10	-0.580930	-2.993617	3.676787	-5.011516	2.303760	0.082127	0.467074	-3.735054	6.810736	1.750897	-2.456542	NaN	2.254278
11	0.772630	-1.916516	1.418299	-3.175414	4.415930	11.164527	-2.719894	0.220061	7.543881	4.925548	0.281064	NaN	0.116370
12	4.149684	-1.846681	13.839290	2.504595	6.685176	9.346221	-1.706475	0.662486	12.733123	10.947612	7.012079	NaN	13.498720

Data Overview – Example2

```
[8]: ## ------- data overview example-2 ------- ##

     filepath = "D:\dataset\onlineretail.xlsx"
     df = pd.read_excel(filepath)
```

```
[9]: df.info()

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 207719 entries, 0 to 207718
     Data columns (total 8 columns):
     InvoiceNo      207719 non-null object
     StockCode      207719 non-null object
     Description    206886 non-null object
     Quantity       207719 non-null int64
     InvoiceDate    207719 non-null datetime64[ns]
     UnitPrice      207719 non-null float64
     CustomerID     148101 non-null float64
     Country        207719 non-null object
```

missing or incomplete data

More Detail Data Overview

```
[10]: df.isnull().any()
```

```
[10]: InvoiceNo      False
      StockCode      False
      Description     True
      Quantity       False
      InvoiceDate    False
      UnitPrice      False
      CustomerID      True
      Country        False
      dtype: bool
```

```
[11]: df.isnull().sum()
```

```
[11]: InvoiceNo          0
      StockCode          0
      Description      833
      Quantity           0
      InvoiceDate        0
      UnitPrice          0
      CustomerID     59618
      Country            0
      dtype: int64
```

2 Columns has ‘Null’ values

NaN Value = Not Available

```
[12]: df[df['Description'].isnull()].head()
```

[12]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
622	536414	22139	NaN	56	2010-12-01 11:52:00	0.0	NaN	United Kingdom
1970	536545	21134	NaN	1	2010-12-01 14:32:00	0.0	NaN	United Kingdom
1971	536546	22145	NaN	1	2010-12-01 14:33:00	0.0	NaN	United Kingdom
1972	536547	37509	NaN	1	2010-12-01 14:33:00	0.0	NaN	United Kingdom
1987	536549	85226A	NaN	1	2010-12-01 14:34:00	0.0	NaN	United Kingdom

Try to Find Another Invalid, Missing, or Duplicate Data

?

```
[12]: df[df['Description'].isnull()].head()
```

[12]:

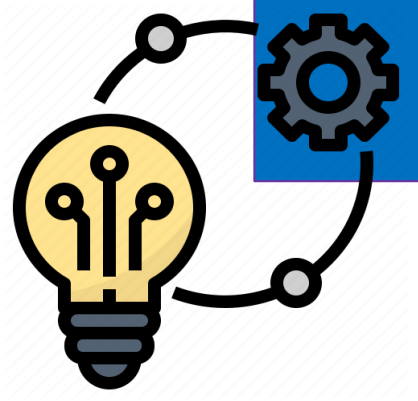
	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
622	536414	22139	NaN	56	2010-12-01 11:52:00	0.0	NaN	United Kingdom
1970	536545	21134	NaN	1	2010-12-01 14:32:00	0.0	NaN	United Kingdom
1971	536546	22145	NaN	1	2010-12-01 14:33:00	0.0	NaN	United Kingdom
1972	536547	37509	NaN	1	2010-12-01 14:33:00	0.0	NaN	United Kingdom
1987	536549	85226A	NaN	1	2010-12-01 14:34:00	0.0	NaN	United Kingdom

Promo/Free Item ?

Price = 0.0 ??

Data Cleaning

Data Cleaning Technique



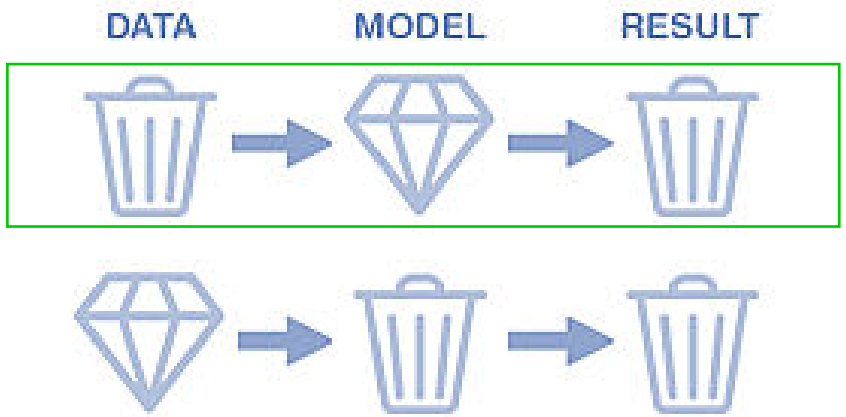
The steps and techniques for data cleaning will vary from dataset to dataset.

Common Steps :

- fixing structural errors
- handling outliers
- handling missing data
- filtering observations

Garbage In Garbage Out

- Better data leads to better algorithm.



Fixing Structural Error

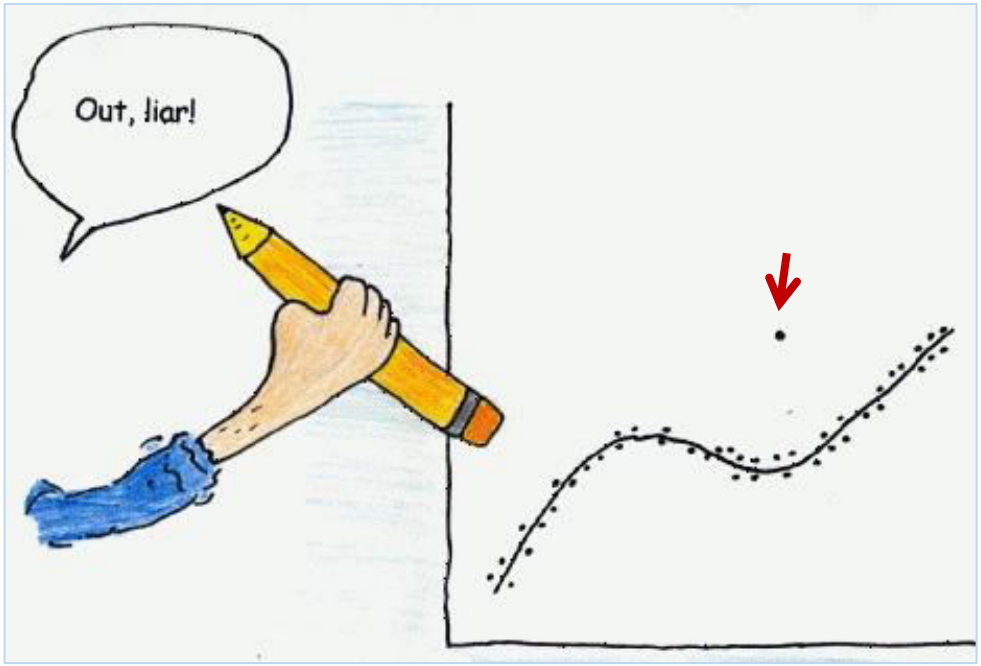
- This step is mostly a concern for categorical features.
- Check for possibility of:
  - typos
  - inconsistent capitalization
  - misslabeled classes

Fixing Structural Error - example

Inconsistent capitalization

Misslabeled classes

Handling Outliers

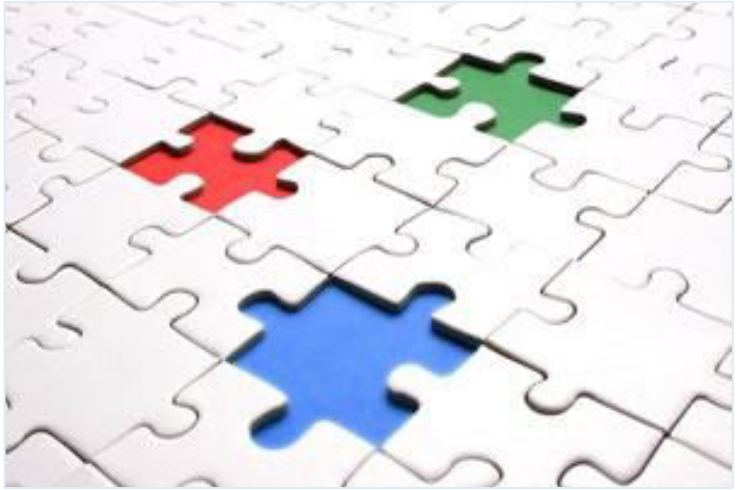


Outliers can cause problems with certain types of data models. For example, linear regression is sensitive to outliers.

We must have a good reason for removing an outlier.

"Outliers are innocent until proven guilty."

Handling Missing Data



Most commonly ways of dealing with missing data :

1. Dropping observations that have missing values
2. Imputing the missing values based on other observations

Imputing Missing Data

Missingness is informative

Imputing the missing values based on other observations.

Common methods :

- Mean or Median Imputation
- Multivariate Imputation
- Random Forest

We cannot simply imputing missing values because it still leads to a loss in information, no matter how sophisticated your imputation method is.

Pandas DataFrame fillna() Function

- fillna() function is used to remove missing values using specified method

```
DataFrame.fillna(self, value=None, method=None, axis=None, inplace=False, limit=None, downcast=None)
```

Name	Description	Type/Value/Default Value
value	Value to use to fill holes (e.g. 0).	scalar, dict, Series, or DataFrame
method	Method to use for filling holes in reindexed Series. ffill: propagate last valid observation forward to fill gap. bfill: use next valid observation to fill gap.	{‘backfill’, ‘bfill’, ‘pad’, ‘ffill’}
axis	Axis along which to fill missing values.	{**0** or ‘**index**’, 1 or ‘columns’}
limit	If method is specified, this is the maximum number of consecutive NaN values to forward/backward fill.	int
inplace	If True, fill in-place.	bool
downcast	A dict of item->dtype of what to downcast if possible	dict

Dropping Missing Data

Missingness is informative

Dropping observations that have missing values.

Even that most algorithms do not accept missing values, still better to tell the algorithm if a value was missing.

We cannot simply dropping observation and its missing values because when you drop observations, you drop information.

Pandas Dataframe dropna() Function

- dropna() function is used to remove missing values

```
DataFrame.dropna(self, axis=0, how='any', thresh=None, subset=None, inplace=False)
```

Name	Description	Type/Value/Default Value
axis	Determine if rows or columns which contain missing values are removed.	{**0 or 'index'**, 1 or 'columns'}
how	Determine if row or column is removed when we have at least one NA or all NA.	{**'any'**, 'all'}
thresh (ops)	Keep rows/columns with at least that many non-NA values	int
subset (ops)	Define in which columns to look for missing values.	array-like
inplace	If True, do operation inplace and return None.	bool

Handling Missing Data – Other Option

Tell the algorithm that the value was missing. How to do so?

Missing Categorical Data

Simply label the missing data as 'Missing'

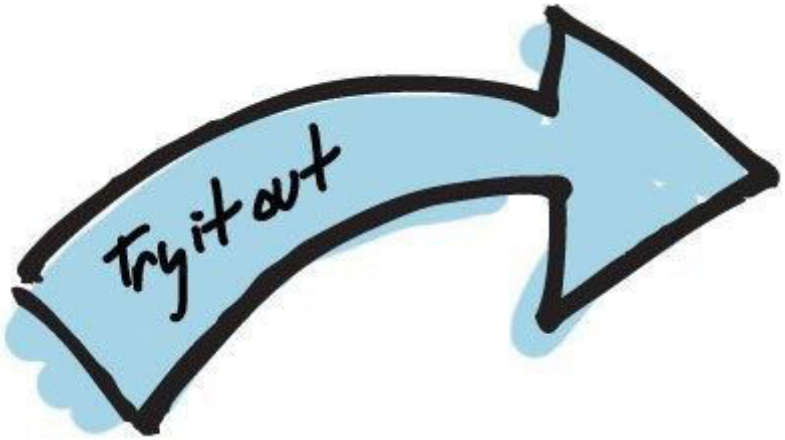
- This means adding a new class for the feature.

Missing Numerical Data

Flag and fill the missing values.

- Flag the observation with an indicator variable of missingness.
- Fill the original missing value with 0 just to meet the technical requirement of no missing values.

Data Cleaning – Example



- Open the jupyter notebook file for practicing by yourself :

- File : /script/datacleaning.ipynb
- Use Shift + Enter (or 'Run' button▶) to run script in a cell.

Data Cleaning – Example1

```
[2]: ## ------- data cleaning example-1 ------- ##
     #  import data from a *.xlsx retail transact
     path = "D:/dataset/onlineretail.xlsx"
     df = pd.read_excel(path)
```

```
[3]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 207719 entries, 0 to 207718
Data columns (total 8 columns):
InvoiceNo      207719 non-null object
StockCode      207719 non-null object
Description    206886 non-null object
Quantity       207719 non-null int64
InvoiceDate    207719 non-null datetime64[ns]
UnitPrice      207719 non-null float64
CustomerID     148101 non-null float64
Country        207719 non-null object
```

Incomplete data

Columns with Missing Value

```
[5]: df.isnull().any()

[5]: InvoiceNo      False
     StockCode      False
     Description     True
     Quantity       False
     InvoiceDate    False
     UnitPrice      False
     CustomerID      True
     Country        False
     dtype: bool
```

```
[6]: df.isnull().sum()

[6]: InvoiceNo          0
     StockCode          0
     Description      833
     Quantity           0
     InvoiceDate        0
     UnitPrice          0
     CustomerID     59618
     Country            0
     dtype: int64
```

Display Some Rows With Missing Value

```
[8]: df[df['CustomerID'].isnull()].head()
```

[8]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
622	536414	22139	NaN	56	2010-12-01 11:52:00	0.00	NaN	United Kingdom
1443	536544	21773	DECORATIVE ROSE BATHROOM BOTTLE	1	2010-12-01 14:32:00	2.51	NaN	United Kingdom
1444	536544	21774	DECORATIVE CATS BATHROOM BOTTLE	2	2010-12-01 14:32:00	2.51	NaN	United Kingdom
1445	536544	21786	POLKADOT RAIN HAT	4	2010-12-01 14:32:00	0.85	NaN	United Kingdom
1446	536544	21787	RAIN PONCHO RETROSPOT	2	2010-12-01 14:32:00	1.66	NaN	United Kingdom

Dropping Missing Values

```
[11]: dfcleaned = df.dropna(axis=0)
      # df.dropna(axis=0, inplace=True)

[12]: print('df: ' , len(df), ' | ', 'dfcleaned: ', len(dfcleaned))

      df:  207719  |  dfcleaned:  148101

[13]: print('df - dfcleaned = ' , len(df)-len(dfcleaned))

      df - dfcleaned =  59618
```

```
[6]: df.isnull().sum()

[6]: InvoiceNo          0
     StockCode          0
     Description      833
     Quantity           0
     InvoiceDate        0
     UnitPrice          0
     CustomerID     59618
     Country            0
     dtype: int64
```

Python Pandas Library

pandas.pydata.org

Python for Data Analysis

Pandas Documentation (API Reference) :
https://pandas.pydata.org/docs/reference/index.html

Data Cleaning – Exercise

- File : datacleaning_exercise.pdf

Solution :

- File : /script/datacleaning_exercise.ipynb
- Use Shift + Enter (or ‘Run’ button ▶) to run script in a cell.

inixindo

"Continuous learning, keep up to date"