Python for Data Analysis: Dataframe Basics and Data Cleansing

Sabtu, 1 April 2023

Arif Romadhan
Sr. Data Scientist

Program Zenius Studi Independen Bersertifikat
Zenius Bersama Kampus Merdeka

PT Cakra Syntesis Indonesia
Android Developer (2016)

Nawatech
Jr. ML Engineer (2017)

Bukalapak
Data Scientist (2018 - 2020)

Data Scientist Instructor
(Sept 2020 - Nov2020)

Data Scientist Instructor
(2021 - 2021)

Evermos
Sr. Data Scientist - Data Lead
(2021 - present)

Data Scientist Instructor
(2021- present)

Arif Romadhan

https://www.linkedin.com/in/arif-romadhan19/

1. Dataframe Basics
2. Dataframe Operations
3. Data Cleansing I
4. Data Cleansing II

Dataframe Basics

we will learn about python dataframe basics such as **data types**, **attributes** and **methods**

Python Dataframe

How to load data as python dataframe (CSV)

Importing Library

```
In [1]: import pandas as pd
```

Import Data from CSV

```
In [2]: df = pd.read_csv('example')
        df
```

Out[2]:

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

Export Data to CSV

```
In [3]: df.to_csv('example',index=False)
```

Python Dataframe

How to load data as python dataframe (HTML/links)

Import Data from Excel Files

```
In [4]: pd.read_excel('Excel_Sample.xlsx',sheet_name='Sheet1')
```

Out[4]:

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

Export Data to Excel Files

```
In [5]: df.to_excel('Excel_Sample.xlsx',sheet_name='Sheet1')
```

Import Data from HTML/Links

```
In [6]: df = pd.read_html('http://www.fdic.gov/bank/individual/failed/banklist.html')
```

```
In [8]: df[0].head()
```

Out[8]:

	Bank Name	City	ST	CERT	Acquiring Institution	Closing Date	Updated Date
0	Washington Federal Bank for Savings	Chicago	IL	30570	Royal Savings Bank	December 15, 2017	February 21, 2018
1	The Farmers and Merchants State Bank of Argonia	Argonia	KS	17719	Conway Bank	October 13, 2017	February 21, 2018
2	Fayette County Bank	Saint Elmo	IL	1802	United Fidelity Bank, fsb	May 26, 2017	July 26, 2017
3	Guaranty Bank, (d/b/a BestBank in Georgia & Mi...	Milwaukee	WI	30003	First-Citizens Bank & Trust Company	May 5, 2017	March 22, 2018
4	First NBC Bank	New Orleans	LA	58302	Whitney Bank	April 28, 2017	December 5, 2017

Dataframe Data Types

Pandas Type	Native Python Type	Description
object	string	The most general dtype. Will be assigned to your column if column has mixed types (numbers and strings).
int64	int	Numeric characters. 64 refers to the memory allocated to hold this character.
float64	float	Numeric characters with decimals. If a column contains numbers and NaNs, pandas will default to float64, in case your missing value has a decimal.
datetime64, timedelta[ns]	N/A (but see the datetime module in Python's standard library)	Values meant to hold time data. Look into these for time series experiments.

command **: df.dtypes**

Data Types :

- object
- int64
- float64
- Datetime64, timedelta[ns]

Dataframe Data Types

In [4]:
```
#Check a particular column type
df['salary'].dtype
```

Out[4]:
```
dtype('int64')
```

Dataframe Attributes

df.attribute	description
dtypes	list the types of the columns
columns	list the column names
axes	list the row labels and column names
ndim	number of dimensions
size	number of elements
shape	return a tuple representing the dimensionality
values	numpy representation of the data

Exercise 5.1

Using the given dataframe,

- How many records this data frame has ?
- How many elements are there ?
- What are the column names ?
- What types of columns we have in this data frame ?

Dataframe Methods

Unlike attributes, python methods have parenthesis.

df.method()	description
head([n]), tail([n])	first/last n rows
describe()	generate descriptive statistics (for numeric columns only)
max(), min()	return max/min values for all numeric columns
mean(), median()	return mean/median values for all numeric columns
std()	standard deviation
sample([n])	returns a random sample of the data frame
dropna()	drop all the records with missing values

Exercise 5.2

Using the given dataframe,

- Give the summary for the numeric columns in the dataset
- Calculate standard deviation for all numeric columns
- What are the mean values of the first 50 records in the dataset?

Basic Descriptive Stats

df.method()	description
describe	Basic statistics (count, mean, std, min, quantiles, max)
min, max	Minimum and maximum values
mean, median, mode	Arithmetic average, median and mode
var, std	Variance and standard deviation
sem	Standard error of mean
skew	Sample skewness
kurt	kurtosis

Dataframe Operations

we will learn about python dataframe operations such as : **filtering, slicing, selection, and sorting**

Dataframe Operations

Unlike method that use parenthesis, we commonly use [] for operations

Filtering

Slicing

Selection

Selection : loc and iloc

Sorting

Filtering

Filtering

To subset the data we can apply Boolean indexing. This indexing is commonly known as a filter. For example if we want to subset the rows in which the salary value is greater than $120K:

```
In [ ]: #Calculate mean salary for each professor rank:
        df_sub = df[ df['salary'] > 120000 ]
```

Any Boolean operator can be used to subset the data:

> greater; >= greater or equal;

< less; <= less or equal;

== equal; != not equal;

Filtering

Any Boolean operator can be used to subset the data:

> greater; >= greater or equal;

< less; <= less or equal;

== equal; != not equal;

```
In [ ]: #Select only those rows that contain female professors:
        df_f = df[ df['sex'] == 'Female' ]
```

Slicing

Slicing

When selecting one column, it is possible to use single set of brackets, but **the resulting object will be a Series** (not a DataFrame):

```
In [ ]: #Select column salary:
        df['salary']
```

When we need to select more than one column and/or make **the output to be a DataFrame**, we should use double brackets:

```
In [ ]: #Select column salary:
        df[['rank','salary']]
```

Selection

Selection

Selecting Columns

Method 1: Subset the data frame using column name:
df['sex']

Method 2: Use the column name as an attribute:
df.sex

Selecting Rows

If we need to select a range of rows, we can specify the range using ":"

```
In [ ]: #Select rows by their position:
        df[10:20]
```

Notice that the first row has a position 0, and the last value in the range is omitted:

So for 0:9 range the first 10 rows are returned.

Selection : loc and iloc

loc

If we need to select a range of rows, using their labels we can use method loc :

```
In [ ]: #Select rows by their labels:
        df_sub.loc[10:20,['rank','sex','salary']]
```

Out[ ]:

	rank	sex	salary
10	Prof	Male	128250
11	Prof	Male	134778
13	Prof	Male	162200
14	Prof	Male	153750
15	Prof	Male	150480
19	Prof	Male	150500

iloc

If we need to select a range of rows and/or columns, using their positions we can use method iloc :

```
In [ ]: #Select rows by their labels:
        df_sub.iloc[10:20,[0, 3, 4, 5]]
```

Out[ ]:

	rank	service	sex	salary
26	Prof	19	Male	148750
27	Prof	43	Male	155865
29	Prof	20	Male	123683
31	Prof	21	Male	155750
35	Prof	23	Male	126933
36	Prof	45	Male	146856
39	Prof	18	Female	129000
40	Prof	36	Female	137000
44	Prof	19	Female	151768
45	Prof	25	Female	140096

Selection : loc and iloc

iloc complete examples

```
df.iloc[0]  # First row of a data frame
df.iloc[i]  #(i+1)th row
df.iloc[-1] # Last row
```

```
df.iloc[:, 0]  # First column
df.iloc[:, -1] # Last column
```

```
df.iloc[0:7]        #First 7 rows
df.iloc[:, 0:2]     #First 2 columns
df.iloc[1:3, 0:2]   #Second through third rows and first 2 columns
df.iloc[[0,5], [1,3]]   #1st and 6th rows and 2nd and 4th columns
```

Sorting

Sorting

We can sort the data by a value in the column. By default the sorting will occur in ascending order and a new data frame is return.

In [ ]:
```
# Create a new data frame from the original sorted by the column Salary
df_sorted = df.sort_values( by ='service')
df_sorted.head()
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
55	AsstProf	A	2	0	Female	72500
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000

Sorting

We can sort the data using 2 or more columns:

In [ ]:
```
df_sorted = df.sort_values( by =['service', 'salary'], ascending = [True, False])
df_sorted.head(10)
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
52	Prof	A	12	0	Female	105000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
55	AsstProf	A	2	0	Female	72500
57	AsstProf	A	3	1	Female	72500
28	AsstProf	B	7	2	Male	91300
42	AsstProf	B	4	2	Female	80225
68	AsstProf	A	4	2	Female	77500

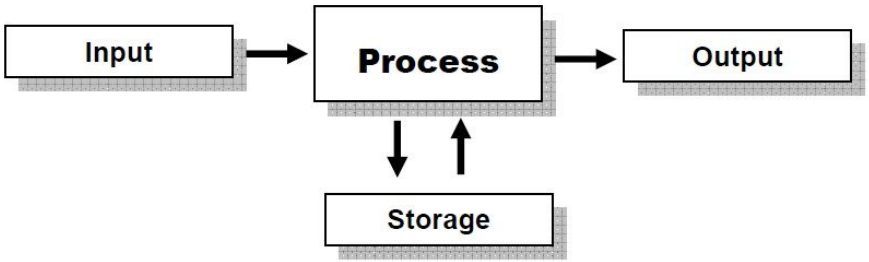
Data Cleansing I

we will learn about the **Data Analytics Basics** and **why Data Cleansing is very important in Data Analysis.**

Data Analytics Basics

Data Analytics

is **science and techniques** of **analyzing raw data** in order to **make conclusions** about that information



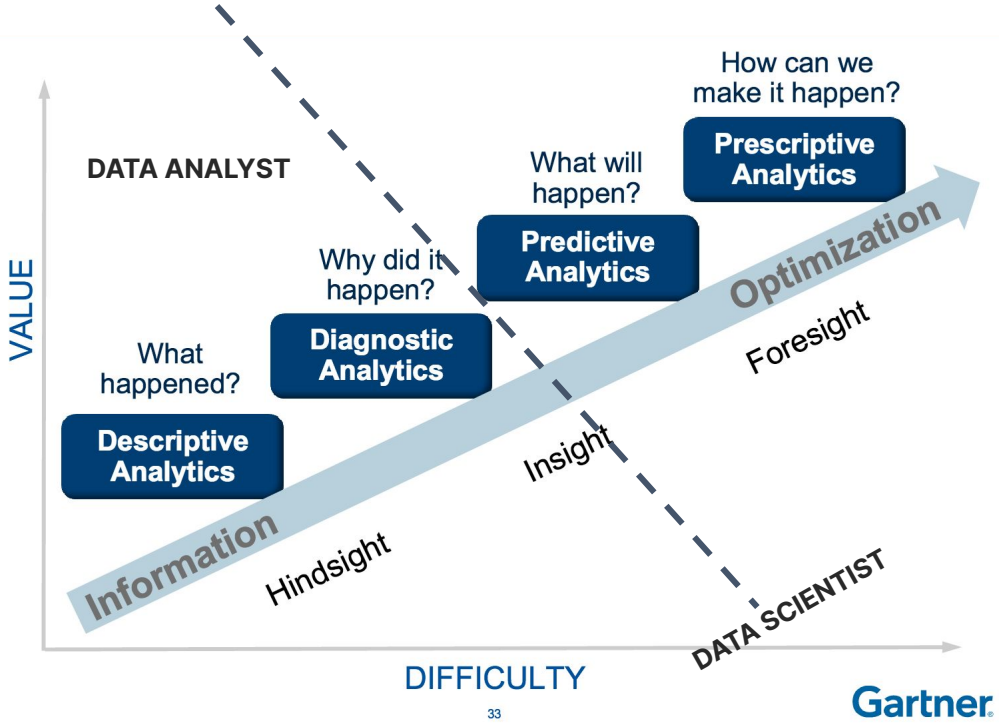
Data Analytics Basics

Most of the cases,

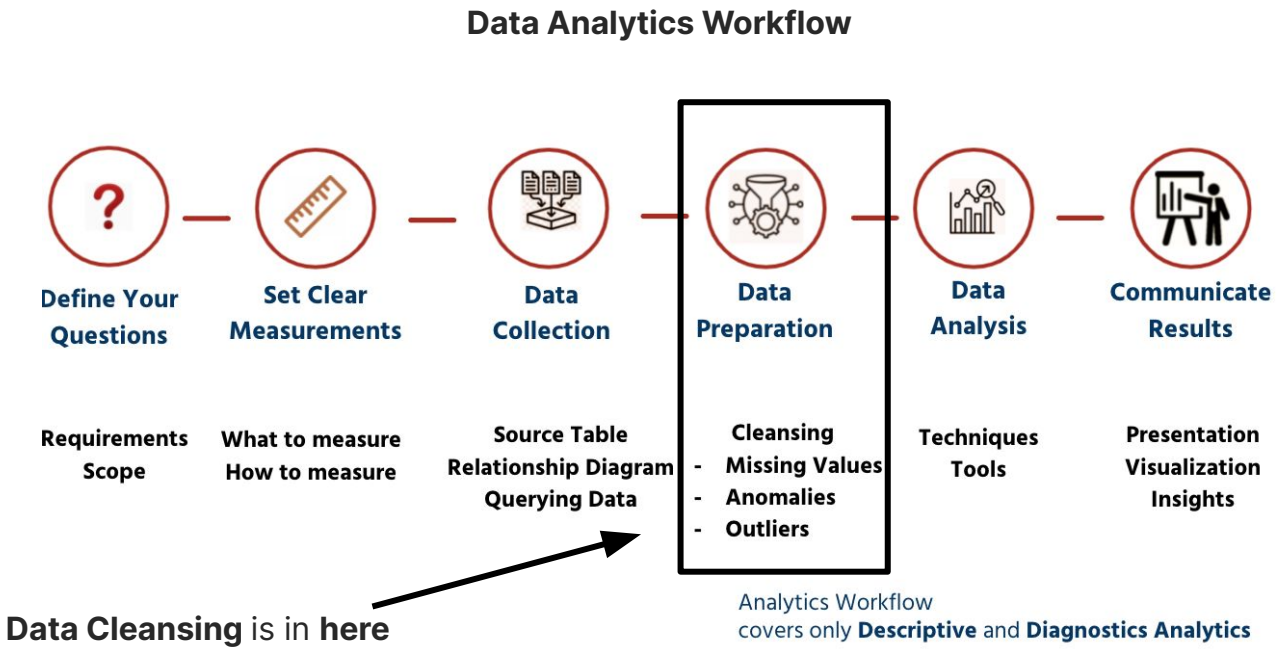
Data Analyst only covers the process of doing the **Descriptive and Diagnostic Analytics**

and on the other hand,

Data Scientist covers whole process of doing the **Descriptive, Diagnostic, Predictive and Prescriptive Analytics**



Data Analytics Basics



Data Analytics Basics

Why do we need Data Cleansing ?

- Preparation for data analysis
- Data almost never comes in clean
- Diagnose your data for problems

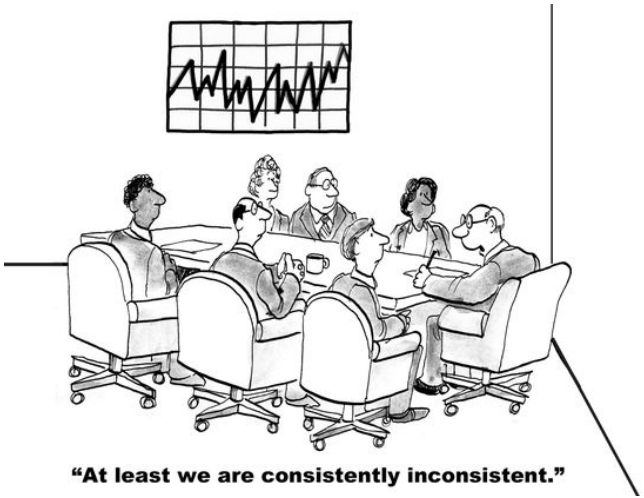
“Garbage in, garbage out”

Your analysis is as good as your data.

Data Analytics Basics

Common Data Problems

- Inconsistent column names
- Missing data
- Outliers
- Duplicate rows
- Untidy
- Need to process columns
- Column types can signal unexpected data values



Data Analytics Basics

Example :

	Continent	Country	female literacy	fertility	population
0	ASI	Chine	90.5	1.769	1.324655e+09
1	ASI	Inde	50.8	2.682	1.139965e+09
2	NAM	USA	99.0	2.077	3.040600e+08
3	ASI	Indonésie	88.8	2.132	2.273451e+08
4	LAT	Brésil	90.2	1.827	NaN

Problems :

- Column name inconsistencies
- Missing data
- Country names are in French

Data Cleansing II

we will learn and practice how to do **Data Cleansing** such as : **Missing Values Handling, Categorical Data Encoding** and **Anomalies & Outlier Handling**

Missing Values Handling

Missing Values Handling

Let's start with creating dataframe with NaN values

```
In [1]: import numpy as np
        import pandas as pd

In [2]: df = pd.DataFrame({'A':[1,2,np.nan],
                           'B':[5,np.nan,np.nan],
                           'C':[1,2,3]})

In [3]: df
```

Out[3]:

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

Missing Values Handling

dropna() method

```
In [4]: df.dropna()
```

Out[4]:

	A	B	C
0	1.0	5.0	1

```
In [5]: df.dropna(axis=1)
```

Out[5]:

	C
0	1
1	2
2	3

```
In [6]: df.dropna(thresh=2)
```

Out[6]:

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2

Drop is not recommended to use, especially for production/real data. We need to maintain our data or our samples.

Missing Values Handling

fillna() method

```
In [7]: df.fillna(value='FILL VALUE')
```

Out[7]:

	A	B	C
0	1	5	1
1	2	FILL VALUE	2
2	FILL VALUE	FILL VALUE	3

```
In [10]: df['A'].fillna(value=df['A'].mean())
Out[10]: 0    1.0
         1    2.0
         2    1.5
         Name: A, dtype: float64
```

Result :

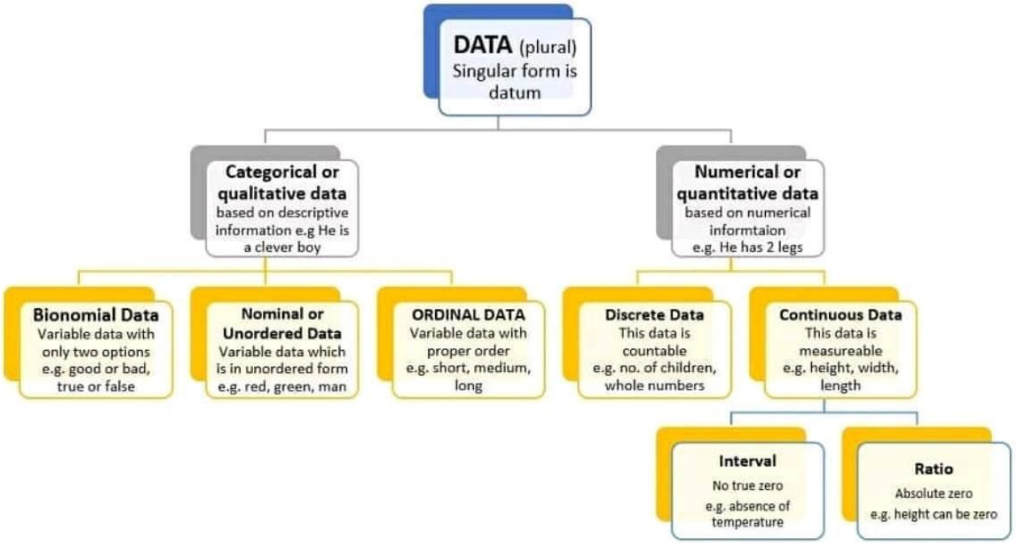
```
In [11]: df
```

Out[11]:

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

Categorical Data Encoding

Categorical Data Encoding



Categorical Data Encoding

Before

```
In [2]: df = pd.read_csv('telco_customer_churn.csv')
```

```
In [3]: df.head()
```

Out[3]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtecti
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	No	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	No	
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	No	

After

```
In [19]: df_dummy = pd.get_dummies(df)
```

```
In [20]: df_dummy.head()
```

Out[20]:

	tenure	MonthlyCharges	TotalCharges	gender_Male	SeniorCitizen_Yes	Partner_Yes	Dependents_Yes	PhoneService_Yes	MultipleLines_No phone service	MultipleLines_Ye
0	1	29.85	29.85	0	0	1	0	0	1	
1	34	56.95	1889.50	1	0	0	0	1	0	
2	2	53.85	108.15	1	0	0	0	1	0	
3	45	42.30	1840.75	1	0	0	0	0	1	
4	2	70.70	151.65	0	0	0	0	1	0	

Categorical Data Encoding

Checking and Converting Data Types

Data Frame

	ColumnA	ColumnB
0	111	444
1	222	555
2	333	666
3	111	444
4	222	777

Type of Column A

```
data['ColumnA'].dtype
```

```
dtype('int64')
```

To string

```
data['ColumnA'] = data['ColumnA'].astype('str')
data['ColumnA'].dtype
```

```
dtype('O')
```

To float

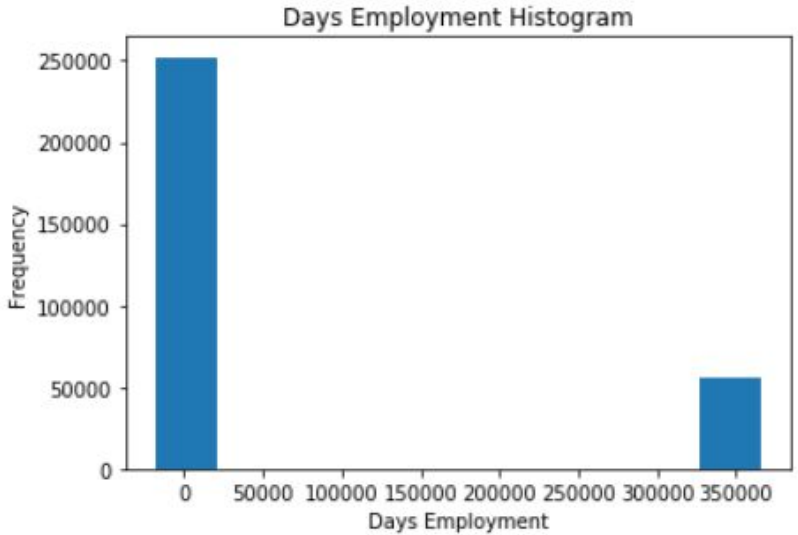
```
data['ColumnA'] = data['ColumnA'].astype('float64')
data['ColumnA'].dtype
data
```

	ColumnA	ColumnB
0	111.0	444
1	222.0	555
2	333.0	666
3	111.0	444
4	222.0	777

Anomalies and Outlier Handling

Anomalies and Outlier Handling

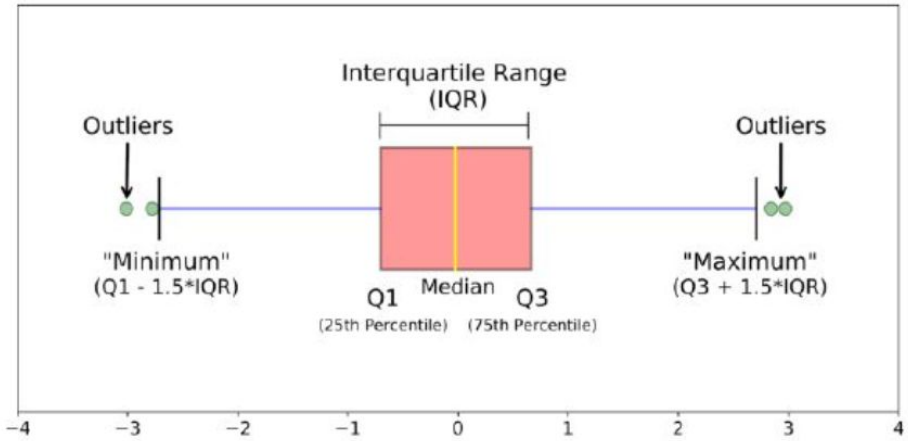
Anomalies vs Outlier



Anomalies and Outliers Handling

Outliers

An outlier is a data point that lies an abnormal distance from other values in the data.



Basic Outlier Formula :

1. Lower Bound = Q1 - 1.5 x IQR

2. Upper Bound = Q3 + 1.5 x IQR

3. IQR = Q3 - Q1

The box plot is a useful graphical display for describing the behavior of the data in the middle as well as at the ends of the distributions

Anomalies and Outliers Handling

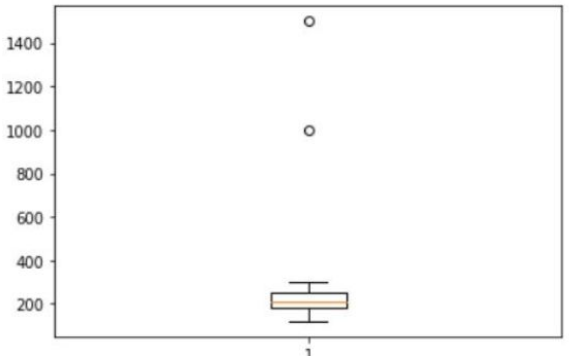
Outliers Handling

```
print(data.shape)
data.head()
```

```
(13, 1)
```

	ColumnA
0	300
1	250
2	171
3	184
4	150

```
import matplotlib.pyplot as plt
plt.boxplot(data['ColumnA'])
plt.show()
```




```
Q1 = data['ColumnA'].quantile(0.25)
Q3 = data['ColumnA'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5*IQR
upper_bound = Q3 + 1.5*IQR
```

```
outliers = data[data['ColumnA']>upper_bound]
outliers
```

	ColumnA
11	1000
12	1500

Anomalies and Outliers Handling

Outliers Handling

1. Removing outlier

2. Fill with Upper/Lower Bound

```
print(data.shape)
data.head()
```

```
(13, 1)
```

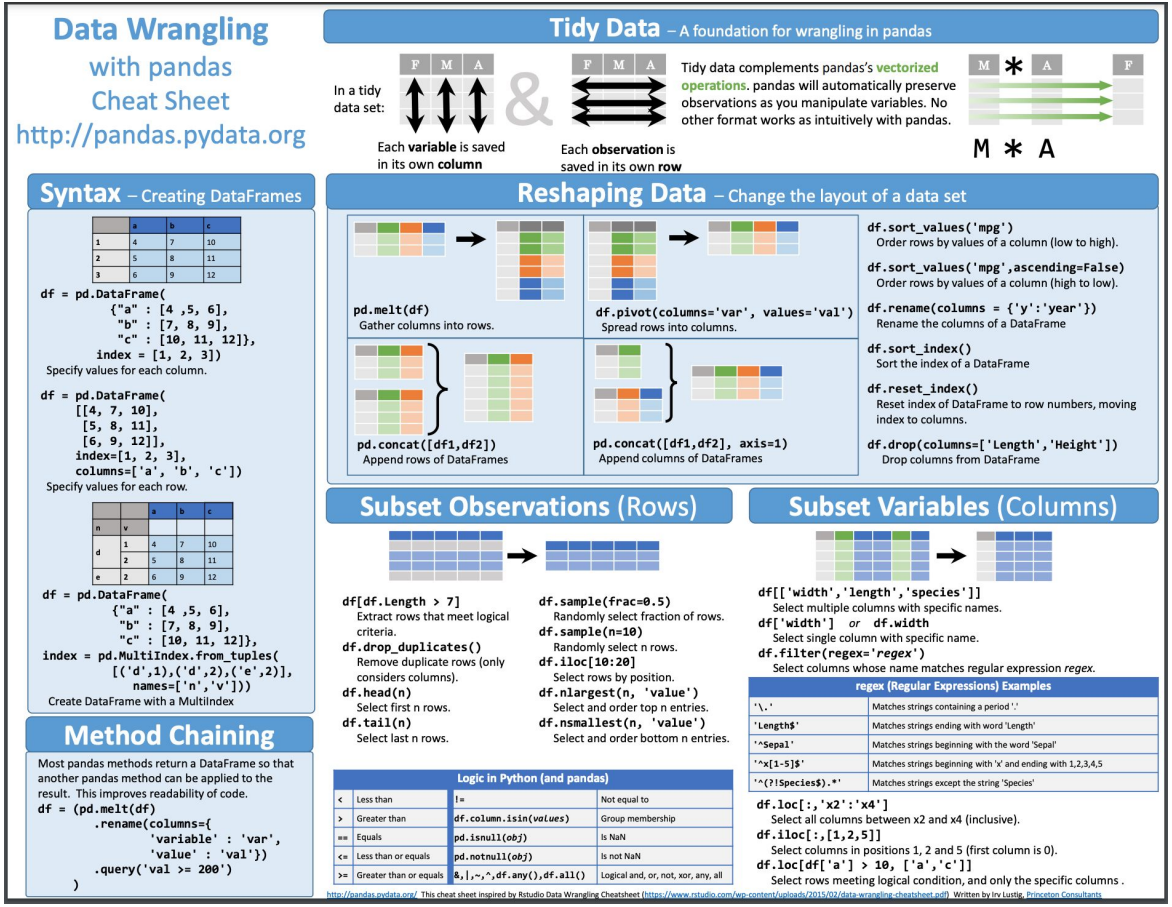
	ColumnA
0	300
1	250
2	171
3	184
4	150

```
data.loc[(data['ColumnA']>upper_bound),'ColumnA'] = upper_bound
```

```
data.describe()
```

	ColumnA
count	13.000000
mean	227.153846
std	71.247042
min	120.000000
25%	184.000000
50%	210.000000
75%	250.000000
max	349.000000

Cheatsheet



Data Wrangling with pandas Cheat Sheet

http://pandas.pydata.org

Tidy Data – A foundation for wrangling in pandas

In a tidy data set:

Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements pandas's **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

M * A

Syntax – Creating DataFrames

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
       index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```

Specify values for each row.

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
         names=['n','v']))
```

Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
    )
```

Reshaping Data – Change the layout of a data set

pd.melt(df)
Gather columns into rows.

df.pivot(columns='var', values='val')
Spread rows into columns.

pd.concat([df1,df2])
Append rows of DataFrames

pd.concat([df1,df2], axis=1)
Append columns of DataFrames

df.sort_values('mpg')
Order rows by values of a column (low to high).

df.sort_values('mpg',ascending=False)
Order rows by values of a column (high to low).

df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

df.sort_index()
Sort the index of a DataFrame

df.reset_index()
Reset index of DataFrame to row numbers, moving index to columns.

df.drop(columns=['Length','Height'])
Drop columns from DataFrame

Subset Observations (Rows)

df[df.Length > 7]
Extract rows that meet logical criteria.

df.drop_duplicates()
Remove duplicate rows (only considers columns).

df.head(n)
Select first n rows.

df.tail(n)
Select last n rows.

df.sample(frac=0.5)
Randomly select fraction of rows.

df.sample(n=10)
Randomly select n rows.

df.iloc[10:20]
Select rows by position.

df.nlargest(n, 'value')
Select and order top n entries.

df.nsmallest(n, 'value')
Select and order bottom n entries.

Subset Variables (Columns)

df[['width','length','species']]
Select multiple columns with specific names.

df['width'] *or* **df.width**
Select single column with specific name.

df.filter(regex='*regex*')
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples

`'\.'`	Matches strings containing a period '.'
`'Length$'`	Matches strings ending with word 'Length'
`'^Sepal'`	Matches strings beginning with the word 'Sepal'
`'^x[1-5]$'`	Matches strings beginning with 'x' and ending with 1,2,3,4,5
`'^(?!Species$).*'`	Matches strings except the string 'Species'

df.loc[:,'x2':'x4']
Select all columns between x2 and x4 (inclusive).

df.iloc[:,[1,2,5]]
Select columns in positions 1, 2 and 5 (first column is 0).

df.loc[df['a'] > 10, ['a','c']]
Select rows meeting logical condition, and only the specific columns .

Logic in Python (and pandas)

<	Less than	!=	Not equal to
>	Greater than	df.column.isin(*values*)	Group membership
==	Equals	pd.isnull(*obj*)	Is NaN
<=	Less than or equals	pd.notnull(*obj*)	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

Cheatsheet

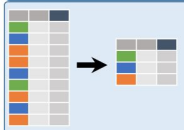
Summarize Data

df['w'].value_counts()
Count number of rows with each unique value of variable
len(df)
# of rows in DataFrame.
df['w'].nunique()
# of distinct values in a column.
df.describe()
Basic descriptive statistics for each column (or GroupBy)

pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

sum()
Sum values of each object.
count()
Count non-NA/null values of each object.
median()
Median value of each object.
quantile([0.25,0.75])
Quantiles of each object.
apply(*function*)
Apply function to each object.
min()
Minimum value in each object.
max()
Maximum value in each object.
mean()
Mean value of each object.
var()
Variance of each object.
std()
Standard deviation of each object.

Group Data



df.groupby(by="col")
Return a GroupBy object, grouped by values in column named "col".

df.groupby(level="ind")
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group.
Additional GroupBy functions:
size()
Size of each group.
agg(*function*)
Aggregate group using function.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

shift(1)
Copy with values shifted by 1.
rank(method='dense')
Ranks with no gaps.
rank(method='min')
Ranks. Ties get min rank.
rank(pct=True)
Ranks rescaled to interval [0, 1].
rank(method='first')
Ranks. Ties go to first value.
shift(-1)
Copy with values lagged by 1.
cumsum()
Cumulative sum.
cummax()
Cumulative max.
cummin()
Cumulative min.
cumprod()
Cumulative product.

Handling Missing Data

df.dropna()
Drop rows with any column having NA/null data.
df.fillna(value)
Replace all NA/null data with value.

Make New Columns

df.assign(Area=lambda df: df.Length*df.Height)
Compute and append one or more new columns.
df['Volume'] = df.Length*df.Height*df.Depth
Add single column.
pd.qcut(df.col, n, labels=False)
Bin column into n buckets.

pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

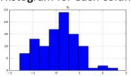
max(axis=1)
Element-wise max.
clip(lower=-10,upper=10)
Trim values at input thresholds
min(axis=1)
Element-wise min.
abs()
Absolute value.

Windows

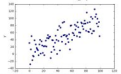
df.expanding()
Return an Expanding object allowing summary functions to be applied cumulatively.
df.rolling(n)
Return a Rolling object allowing summary functions to be applied to windows of length n.

Plotting

df.plot.hist()
Histogram for each column



df.plot.scatter(x='w',y='h')
Scatter chart using pairs of points



Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

+

bdf

x1	x3
A	T
B	F
D	T

=

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

pd.merge(adf, bdf, how='left', on='x1')
Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

pd.merge(adf, bdf, how='right', on='x1')
Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

pd.merge(adf, bdf, how='inner', on='x1')
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

pd.merge(adf, bdf, how='outer', on='x1')
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

adf[adf.x1.isin(bdf.x1)]
All rows in adf that have a match in bdf.

x1	x2
C	3

adf[~adf.x1.isin(bdf.x1)]
All rows in adf that do not have a match in bdf.

ydf

x1	x2
A	1
B	2
C	3

+

zdf

x1	x2
B	2
C	3
D	4

=

Set-like Operations

x1	x2
B	2
C	3

pd.merge(ydf, zdf)
Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

pd.merge(ydf, zdf, how='outer')
Rows that appear in either or both ydf and zdf (Union).

x1	x2
A	1

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('_merge == "left_only"')
.drop(columns=['_merge'])
```
Rows that appear in ydf but not zdf (Setdiff).

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

Thanks!
Any Questions?