Basic Data Analysis for Data Science

Mohamad Irwan Afandi

OUTLINE

- *Introduction to Pandas*
- *Introduction to Preprocessing*
- *Basic Data Manipulation with Pandas*
- *Pandas Functionality*
- *Advance Data Manipulation with Pandas*
- *Data Cleansing with Pandas*
- *Data Visualization with Pandas*

What is Pandas?

- ***Open source library*** *use for* data manipulation and analysis in python.
- ***high-level data manipulation tool*** developed by Wes McKinney
- ***Built on the Numpy package*** and its key data structure is called the ***DataFrame***.
- DataFrames allow you to store and manipulate tabular data in ***rows*** of observations and ***columns*** of variables.
- Can ***load data from different data format***
- ***With pandas we can*** *handling missing value, reshaping data, take few data (indexing, slicing and subsetting), delete or insert, aggregation, transformation, merging, join, time series, etc*

Read Data in Pandas

When we try to read data from external source with the formal csv, xlsx, txt, json, xml, html, etc. The data will convert into ***same structure*** that called as ***DataFrame.*** The DataFrame looks like excel where data will be placed in ***rows and columns.*** The syntax below is used to read csv data in pandas.

```
import pandas as pd
```

```
df = pd.read_csv('insurance.csv') #read csv data and save in df variable
df.head()
```

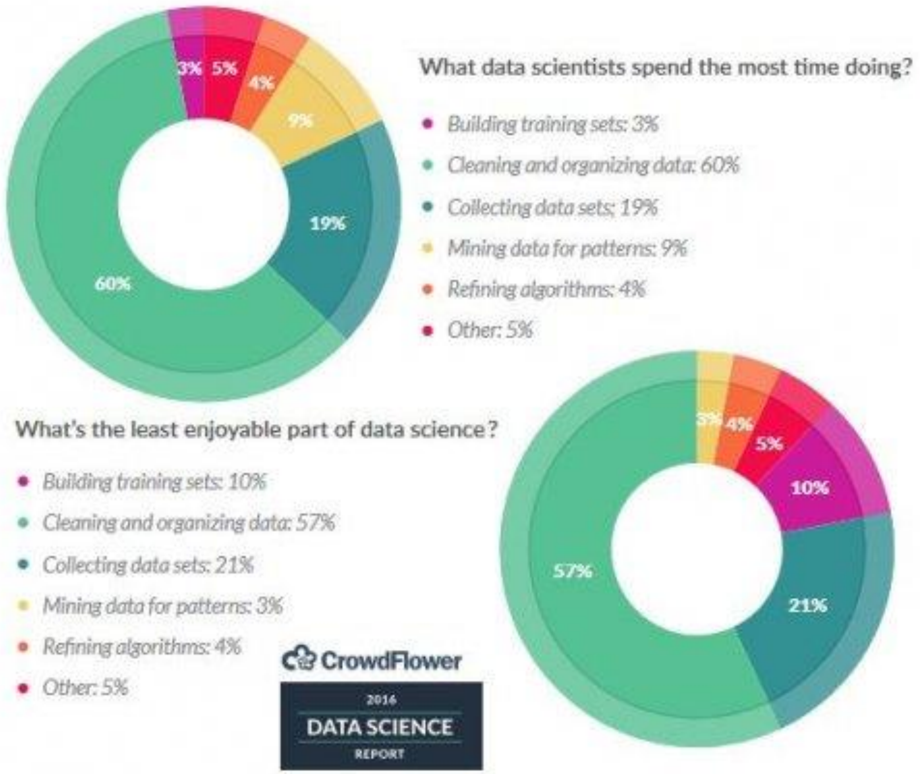
	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

Note: you can also add some parameter when read the data like na_values, skiprows, names, etc.

Data Preprocessing

*Data mining technique that involves transforming raw data into an **understandable format**. Real-world data is often **incomplete**, **inconsistent**, and/or **lacking in certain behaviors or trends**, and is likely to **contain many errors**. Data preprocessing is a proven method of **resolving such issues**.*

Row Data → Structure Data → Data Preprocessing → Exploratory Data Analysis (EDA) → Insight report (Visual Graphs)



Data scientists spend ***60% of their time on cleaning and organizing data***. **Collecting data sets comes second at 19%** of their time, meaning data scientists spend ***around 80% of their time on preparing and managing data for analysis.***

~ CrowdFlower

Basic Data Manipulation 1

1. Data Selection

```
df[['age','charges']].head()
```

	age	charges
0	19	16884.92400
1	18	1725.55230
2	28	4449.46200
3	33	21984.47061
4	32	3866.85520

Select by the column name

```
df.loc[1:3, ['sex','bmi']]
```

	sex	bmi
1	male	33.770
2	male	33.000
3	male	22.705

loc is label-based, which means that we have to specify the **name of the rows and columns** that we need to filter out.

```
df.iloc[1:3,0:2]
```

	age	sex
1	18	male
2	28	male

iloc is integer index-based. So here, we have to specify **rows and columns by their integer index**.

2. Data Filtering

```
df[(df['age']<20)&(df['smoker']=='yes')].head(2).reset_index()
```

	index	age	sex	bmi	children	smoker	region	charges
0	0	19	female	27.90	0	yes	southwest	16884.9240
1	57	18	male	31.68	2	yes	southeast	34303.1672

```
df[(df['age']>20)|(df['smoker']=='yes')].head(2)
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.9	0	yes	southwest	16884.924
2	28	male	33.0	3	no	southeast	4449.462

**Add with logic operator and (&) or (|)*

Basic Data Manipulation

2

3. Data Addition

Column

```
df_x = df.copy()
def grouped(x):
  return "over" if x>30 else "normal"
```

```
df_x["category"] = df['bmi'].apply(grouped)
df_x.head()
```

	age	sex	bmi	children	smoker	region	charges	category
0	19	female	27.900	0	yes	southwest	16884.92400	normal
1	18	male	33.770	1	no	southeast	1725.55230	over
2	28	male	33.000	3	no	southeast	4449.46200	over
3	33	male	22.705	0	no	northwest	21984.47061	normal
4	32	male	28.880	0	no	northwest	3866.85520	normal

Row

```
df1 = pd.DataFrame({'Name':['A','B'],
                    'Age':[20,15]})
df2 = pd.DataFrame({'Name':['D','E'],
                    'Age':[22,25]})
df_result = pd.concat([df1, df2], ignore_index = True)
df_result
```

	Name	Age
0	A	20
1	B	15
2	D	22
3	E	25

4. Data Deletion

```
df_result.drop([1,3], axis=0, inplace=True)
df_result
```

	Name	Age
0	A	20
2	D	22

```
df_result.drop(['Name','Age'], axis=1, inplace=True)
df_result
```

0
2

**Axis = 0 (row) and Axis = 1 (column)*

Basic Data Manipulation

3

5. Rename Column

```
new_name = {'Name':'passient',
            'Age':'age'}
df_result.rename(columns=new_name, inplace=True)
df_result
```

	passient	age
0	A	20
1	B	15
2	D	22
3	E	25

Make lower case of the column name:
column = str.lower

6. Data Sorting

```
df_result.sort_values(['age'])
```

	passient	age
1	B	15
0	A	20
3	D	22
2	C	25
4	E	25

```
df_result.sort_values(['age','passient'], ascending=False)
```

	passient	age
4	E	25
2	C	25
3	D	22
0	A	20
1	B	15

**ascending = False (mean descending), default True*

Basic Functionality on Pandas

1

df.head(n) : return first n record
df.tail(n)   : return last n record

```
df.head()
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

```
df.tail()
```

	age	sex	bmi	children	smoker	region	charges
1333	50	male	30.97	3	no	northwest	10600.5483
1334	18	female	31.92	0	no	northeast	2205.9808
1335	18	female	36.85	0	no	southeast	1629.8335
1336	21	female	25.80	0	no	southwest	2007.9450
1337	61	female	29.07	0	yes	northwest	29141.3603

mean, median, mode, min, max, standard deviation and count

```
mode = df['age'].mode()
median = df['age'].median()
mean = round(df['age'].mean(),2)
min = df['age'].min()
max = df['age'].max()
deviasi = round(df['age'].std(),2)
count = df['age'].count()
```

```
print("Mean: {} | Mode: {} | Median: {}". format(mean, mode[0], median))
print("Min: {} | Max: {}".format(min, max))
print("Standar Deviasi: {} | Count: {}".format(deviasi, count))
```

```
Mean: 39.21 | Mode: 18 | Median: 39.0
Min: 18 | Max: 64
Standar Deviasi: 14.05 | Count: 1338
```

Basic Functionality on Pandas

Gat quick summary of data (statistic)

```
df.describe()
```

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

Shape : data dimension
Columns : get all dataframe's columns

```
df.shape
```

```
(1338, 7)
```

```
df.columns
```

```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges'], dtype='object')
```

Data Engineering (1)

1 *Missing value check*

.isnull().sum() *: return the number nan value in data*
.notnull().sum() *: return the number of not nan value in data*

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

hotel_type	0
is_canceled	0
lead_time	0
arrival_date_year	0
arrival_date_month	0
arrival_date_week_number	0
arrival_date_day_of_month	0
stays_in_weekend_nights	0
stays_in_week_nights	0
adults	0
children	4
babies	0
meal_type	0
country_origin	488
market_segment	0
distribution_channel	0
is_repeated_guest	0
previous_cancellations	0
previous_bookings_not_canceled	0
reserved_room_type	0
assigned_room_type	0
booking_changes	0
deposit_type	0
agent	16340
company	112593

```
df.notnull().sum()
```

hotel_type	119390
is_canceled	119390
lead_time	119390
arrival_date_year	119390
arrival_date_month	119390
arrival_date_week_number	119390
arrival_date_day_of_month	119390
stays_in_weekend_nights	119390
stays_in_week_nights	119390
adults	119390
children	119386
babies	119390
meal_type	119390
country_origin	118902
market_segment	119390
distribution_channel	119390
is_repeated_guest	119390
previous_cancellations	119390
previous_bookings_not_canceled	119390
reserved_room_type	119390
assigned_room_type	119390
booking_changes	119390
deposit_type	119390
agent	103050
company	6797

2 *Computing unique value*

.unique() : *return the unique value from the column.*
.value_counts() : *return unique value and its freq*

```
df['hotel_type'].unique()
```

```
array(['Resort Hotel', 'City Hotel'], dtype=object)
```

```
df['hotel_type'].value_counts()
```

```
City Hotel      79330
Resort Hotel    40060
Name: hotel_type, dtype: int64
```

Data Engineering (2)

3

Dropping and filling the missing value

.dropna*(axis=1, inplace = True) : delete the missing value by row (axis : 0) or by column (axis : 1)*

.fillna(....) *: replace the missing value with new value it can mean, mode or median*

```
df.dropna(axis=1, inplace=True)
df.isnull().sum()
```

```
hotel_type                        0
is_canceled                       0
lead_time                         0
arrival_date_year                 0
arrival_date_month                0
arrival_date_week_number          0
arrival_date_day_of_month         0
stays_in_weekend_nights           0
stays_in_week_nights              0
adults                            0
babies                            0
meal_type                         0
market_segment                    0
distribution_channel              0
is_repeated_guest                 0
previous_cancellations            0
previous_bookings_not_canceled    0
reserved_room_type                0
assigned_room_type                0
booking_changes                   0
deposit_type                      0
days_in_waiting_list              0
customer_type                     0
adr                               0
required_car_parking_spaces       0
```

```
df = df.apply(lambda x: x.fillna(x.median()) if x.dtype.kind in 'iuf' else x.fillna(df['num-of-doors'].mode()[0]))
```

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

```
symboling            0
normalized-losses    0
make                 0
fuel-type            0
aspiration           0
num-of-doors         0
body-style           0
drive-wheels         0
engine-location      0
wheel-base           0
length               0
width                0
height               0
curb-weight          0
engine-type          0
num-of-cylinders     0
engine-size          0
fuel-system          0
```

Advanced Data Manipulation

A pivot table is a table of statistics that summarizes the data of a more extensive table. In practical terms, a pivot table calculates a statistic on a breakdown of values

```
# 8. Which month has the highest number of cancellations?
booked = df[df['is_canceled'] == 1]
x = pd.pivot_table(booked, index=['arrival_date_year','arrival_date_month'], values='is_canceled', aggfunc='count')
x
# x.reindex(x['is_canceled'].sort_values(ascending=False).index)
```

		is_canceled
arrival_date_year	arrival_date_month	
2015	August	1598
	December	973
	July	1259
	November	486
	October	1732
	September	2094
2016	April	2061
	August	1825
	December	1398
	February	1337
	January	557

Compute a simple cross-tabulation of two (or more) factors. By default computes a **frequency** table of the factors

```
pd.crosstab(df['age'], df['smoker'])
```

smoker	no	yes
age		
18	57	12
19	50	18
20	20	9
21	26	2
22	22	6
23	21	7
24	22	6
25	23	5
26	25	3
27	19	9

Advanced Data Manipulation

Binning : grouping the continuous data into multiple buckets for further analysis (**pd.cut()**)

```
bin = [15, 22, 35, 50,100]
label = ['teenager','productive','superb','old']
df['age_category'] = pd.cut(df['age'], bin, labels=label)
df[['age','age_category']].head(7)
```

	age	age_category
0	19	teenager
1	18	teenager
2	28	productive
3	33	productive
4	32	productive
5	31	productive
6	46	superb

```
df[(df['age_category']=='productive')]['age'].tail()
```

```
1318    35
1320    31
1324    31
1328    23
1331    23
Name: age, dtype: int64
```

Advanced Data Manipulation

Categorical Encoding Ordinal Data

to convert this kind of categorical text (**Ordinal / can be ranked**) data into model-understandable numerical data, we use the **Label Encoder function**.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['age_level'] = le.fit_transform(df['age_category'])
df.tail()
```

	age	sex	bmi	children	smoker	region	charges	age_category	age_level
1333	50	male	30.97	3	no	northwest	10600.5483	superb	2
1334	18	female	31.92	0	no	northeast	2205.9808	teenager	3
1335	18	female	36.85	0	no	southeast	1629.8335	teenager	3
1336	21	female	25.80	0	no	southwest	2007.9450	teenager	3
1337	61	female	29.07	0	yes	northwest	29141.3603	old	0

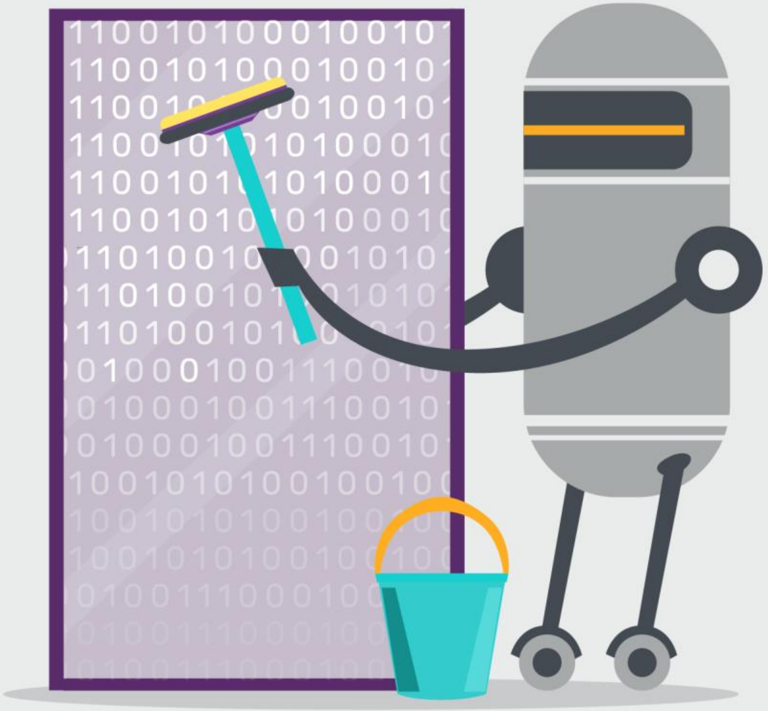
Categorical Encoding Nominal Data

For the nominal data that doesn't have the hierarchy, we shouldn't use the Label Encoding . We can use **pd.get_dummies function**

```
df = pd.get_dummies(df['age_category'])
```

```
df
```

	teenager	productive	superb	old
0	1	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	1	0	0
4	0	1	0	0



Data Cleansing

Data cleansing or data cleaning is the process of **detecting and correcting corrupt or inaccurate records from a record set, table, or database** and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data

- *Missing value checking and handling*
- *Duplicates checking*
- *Anomaly and outlier detection*
- *Data Type checking*

1

Missing Value

Why missing value exist?

- *Value are missing during data acquisition process.*
- *Value are delete accidentally*
- *Corrupt data*
- *Miss match between row and column*
- *Logic error on system*
- *The real value is not available*
- *User error*

Missing Value Checking

```
missing_data = df.isnull().sum().reset_index()
missing_data.columns = ['variable','count_missing']
missing_data['filling (%)'] = 100 - (missing_data['count_missing']/df.shape[0]*100)
missing_data.sort_values(['filling (%)'])
```

	variable	count_missing	filling (%)
24	company	112593	5.693107
23	agent	16340	86.313762
13	country_origin	488	99.591256
10	children	4	99.996650
0	hotel_type	0	100.000000
29	total_of_special_requests	0	100.000000
28	required_car_parking_spaces	0	100.000000
27	adr	0	100.000000
26	customer_type	0	100.000000
25	days_in_waiting_list	0	100.000000
22	deposit_type	0	100.000000

More then 80% data are missing, just **drop the feature**, because we can not analyst the data anymore.

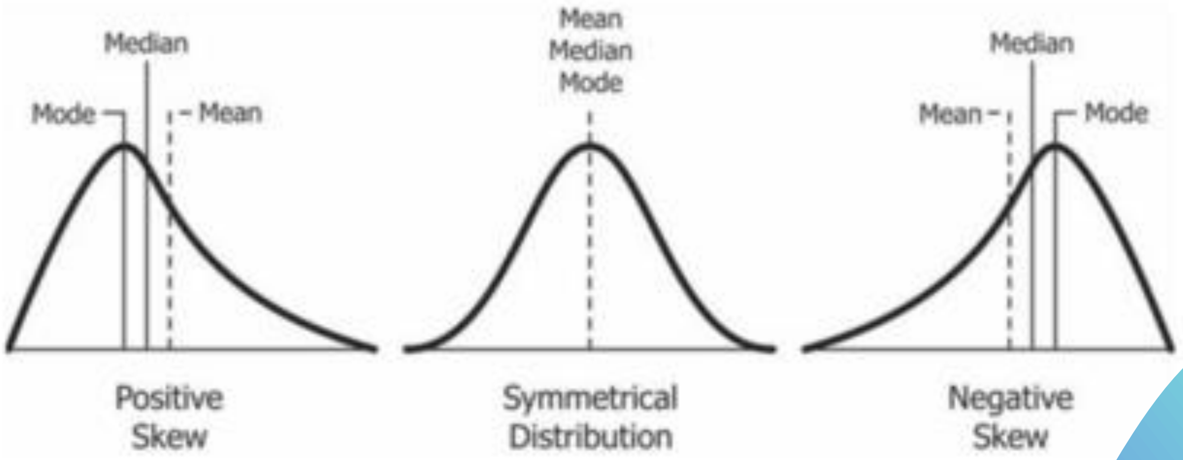
Missing Value Handling

Data Categorical

	age	sex	bmi
0	19	female	27.900
1	18	male	33.770
2	28	male	33.000
3	33	male	22.705
4	32	male	28.880

We can’t do math operations like mean and median. We handle this problem using **mode**

Data Numeric



Positive or negative skew : **median**
Symmetry : **mean, median, mode**

2

Data Duplication Handling

Data with the same values (data redundancy). How to handle this problem? we need to **drop this data**, because this data can skew our model to one side.

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

```
data.drop_duplicates()
```

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

```
data.drop_duplicates(subset = 'ColumnA')
```

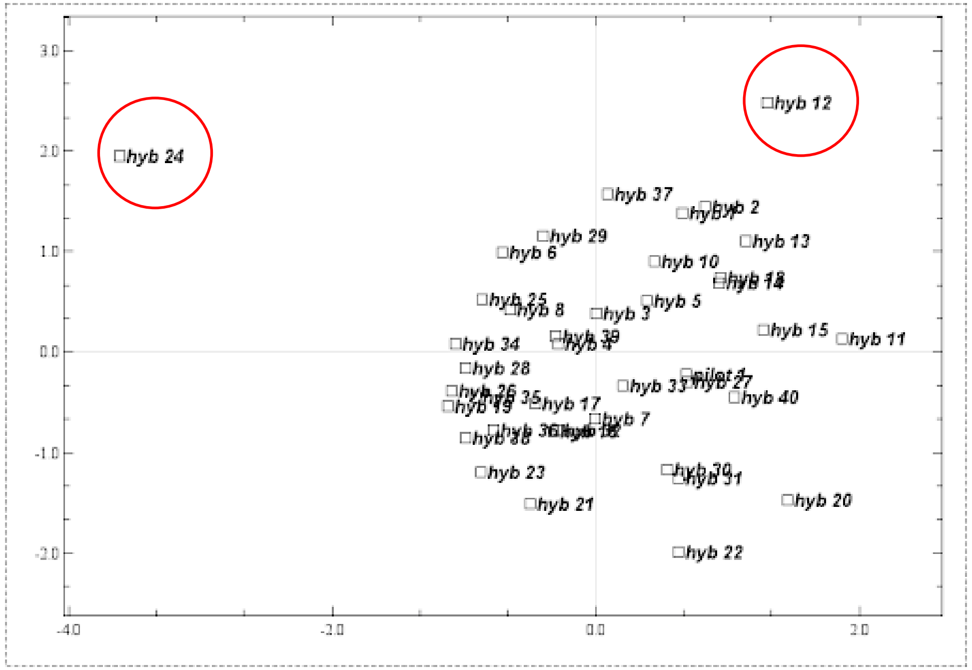
	ColumnA	ColumnB
0	111	444
1	222	555
2	333	666

3

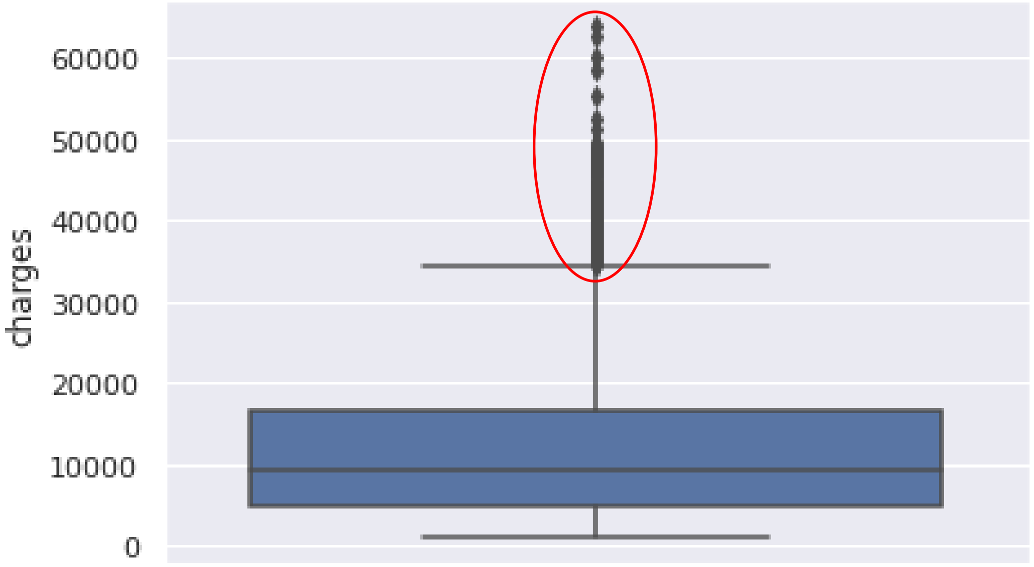
Outlier and Anomalies Data Handling

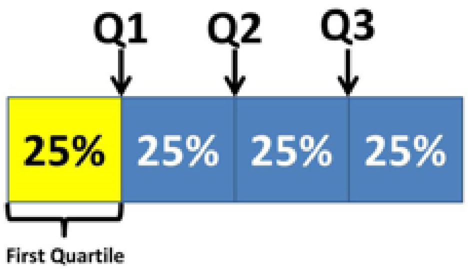
We can check the outlier or anomaly on data using scatter plot, box plot , or using IQR method.

Scatter plot



Box-plot





IQR

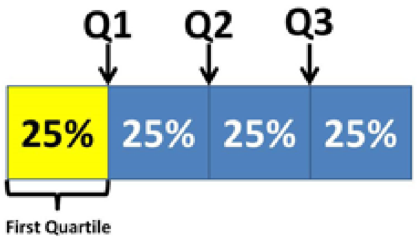
Boundary Limit:

- *Higher Outlier : Q3 + (1.5 * Q3)*
- *Lower Outlier : Q1 – (1.5 * Q1)*

Outlier and Anomalies Data Handling

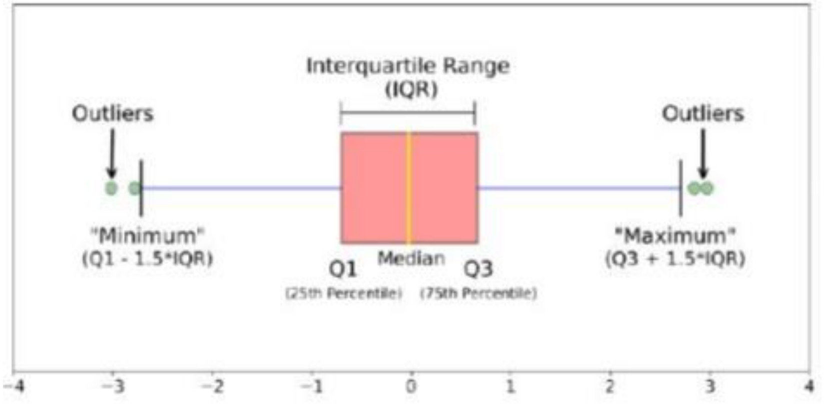
We can check the outlier or anomaly on data using scatter plot, box plot , or using IQR method.

IQR



Boundary Limit:

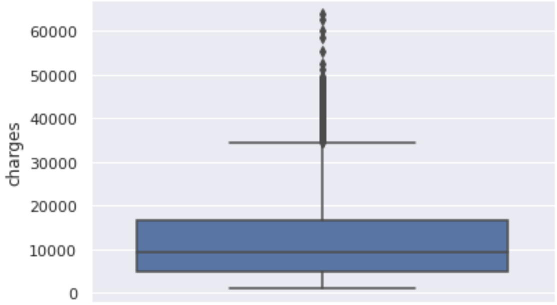
- *Higher Outlier : Q3 + (1.5 * Q3)*
- *Lower Outlier : Q1 – (1.5 * Q1)*




```
Q1 = df['charges'].quantile(0.25)
Q3 = df['charges'].quantile(0.75)
IQR = Q3-Q1
Lb = Q1-(1.5*IQR)
Ub = Q3+(1.5*IQR)
len(df[(df['charges']<Lb) | (df['charges']>Ub)])
# print(Q1,' ', Q3,' ',' ',IQR,' ',Lb,' ', Ub)
```

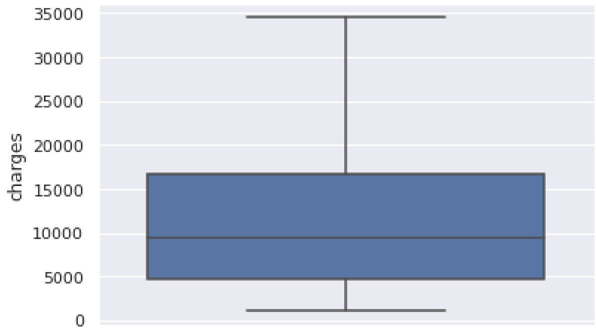
```
sns.boxplot(y=df['charges'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f4c0ad438>
```




```
sns.boxplot(y=df['charges'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1f4c1bdf60>
```




```
#IQR
def outlier(x):
  if x > Ub:
    return Ub
  elif x < Lb:
    return Lb
  else:
    return x

df["charges"] = df["charges"].apply(outlier)
sns.boxplot(y=df['charges'])
```

4

Data Type Checking

We can check the attribute data type using **.dtypes** or **.info()**

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

```
df.dtypes
```

```
age           int64
sex          object
bmi         float64
children      int64
smoker       object
region       object
charges     float64
dtype: object
```

Change attribute data type use **.astype(data type)**

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

To Datetime

```
from datetime import datetime as dtime
data['columnC'] = pd.to_datetime(data['columnC'], format="%Y-%m-%d")
```

introduction to matplotlib

by Jason Lee

bokeh

Seaborn

Data Visualization with Python

Create Data Visualization

1. Import library

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

2. Select the data you want to visualize

```
ax = df['make'].value_counts().plot(kind='bar', figsize=(20,10), color ='orange')
plt.xlabel('Brand', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.title('The  number of car sales by Brand' , fontsize=16)

for p in ax.patches:
    ax.annotate(str(p.get_height()), (p.get_x() * 1.005, p.get_height() * 1.005))
```

1

```
fig= plt.figure(figsize=(10,7))
plt.scatter(df['price'], df['horsepower'])
plt.xlabel('Price', fontsize=14)
plt.ylabel('horsepower', fontsize=14)
plt.title("The relation between price and house power", fontsize=16)
plt.show()
```

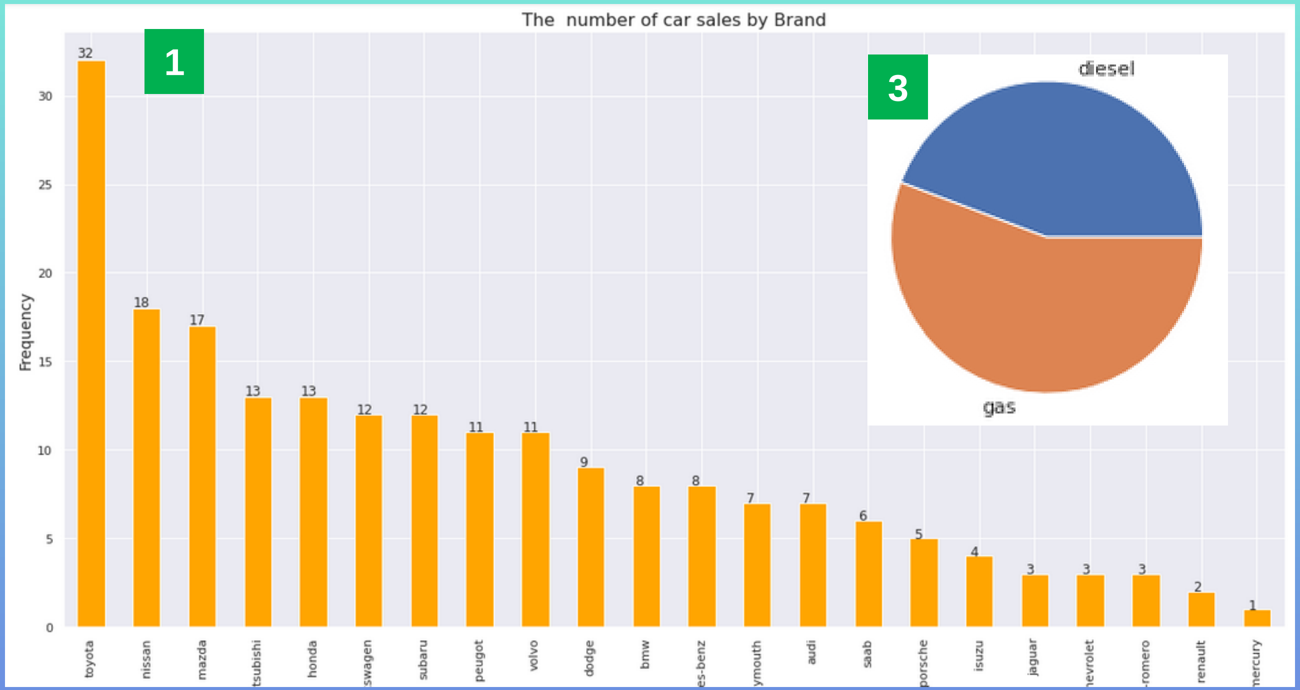
2

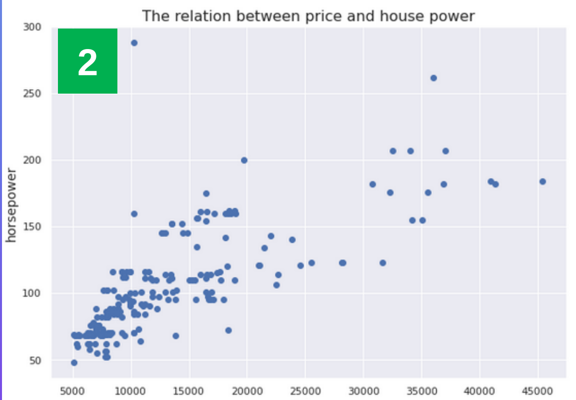
```
from matplotlib.pyplot import pie, axis, show
sums = df['horsepower'].groupby(df['fuel-type']).mean()
axis('equal');
pie(sums, labels=sums.index);
show()
```

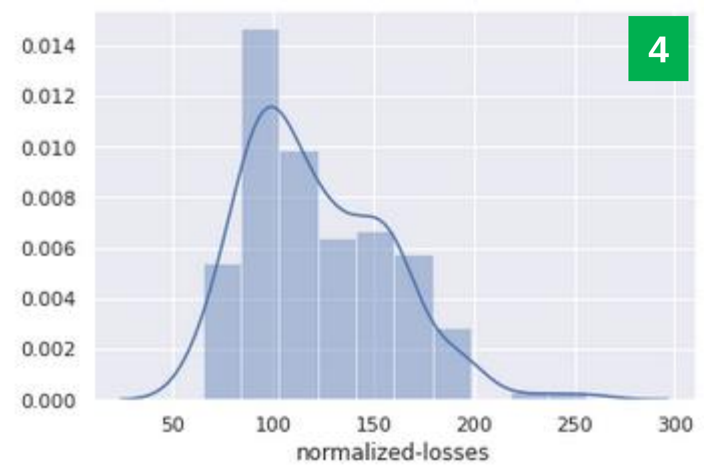
3

```
df['normalized-losses'] = df['normalized-losses'].astype(float)
sns.distplot(df['normalized-losses'], bins=10)
plt.show()
```

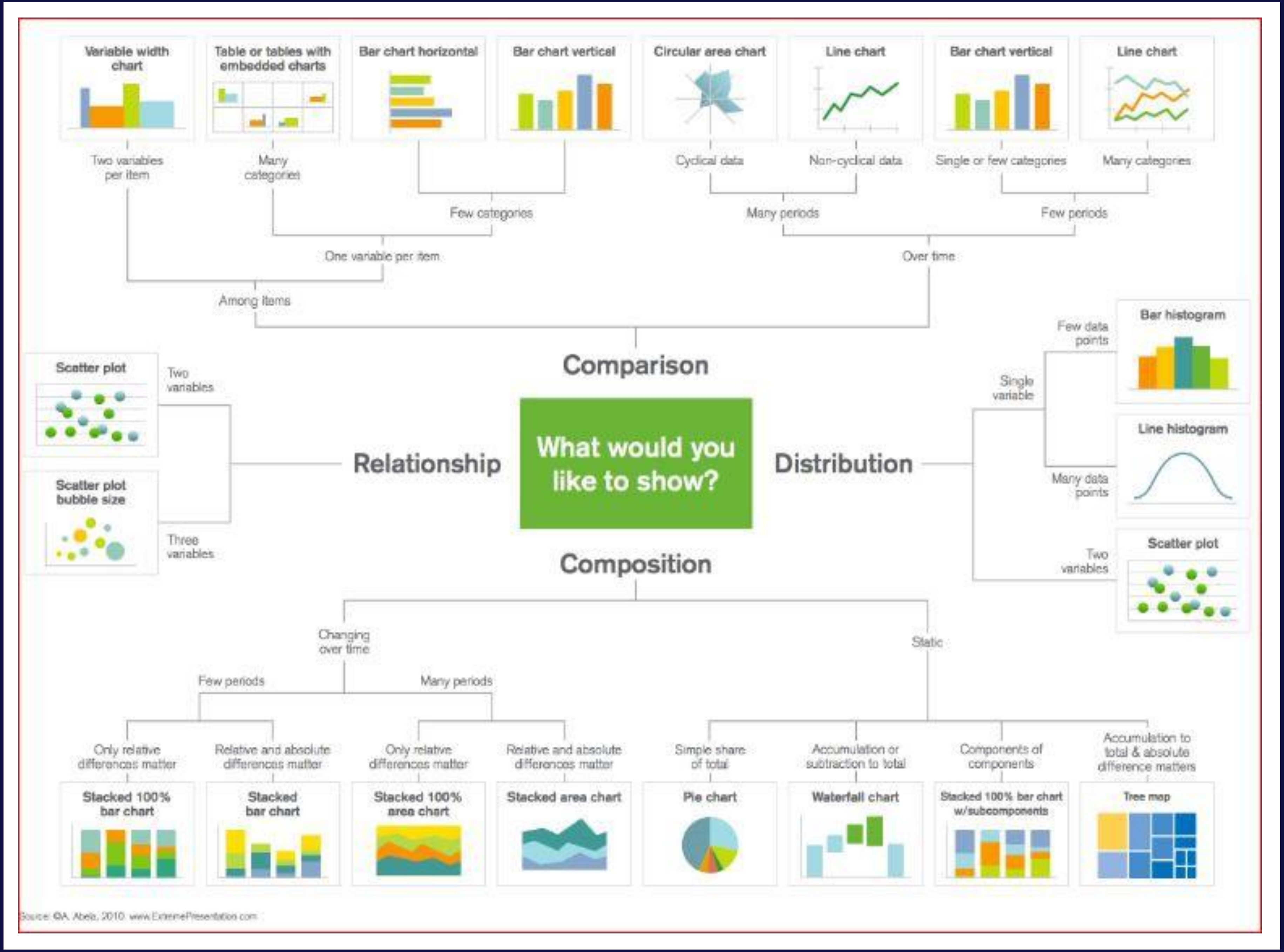
4







Choose The Right Chart



Thank You

Don’t forget to say Alhamdulillah for today

Reference

- https://www.w3schools.com/python/
- https://www.geeksforgeeks.org/data-preprocessing-in-data-mining/
- https://towardsdatascience.com/data-preprocessing-concepts-fa946d11c825
- https://www.ktvn.com/story/41067230/the-top-10-types-of-data-visualization-made-simple
- https://www.programmer-books.com/wp-content/uploads/2019/04/Python-for-Data-Analysis-2nd-Edition.pdf