DATA SCIENCE B

Machine Learning Regression and Time Series

By Group 2

OUR TEAM MEMBERS

M. Kamal Jaza

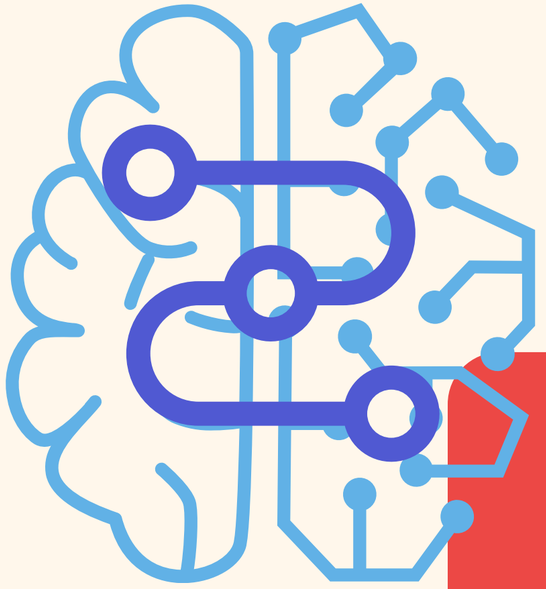
M. Irvan Arfandi

Risma Ashali

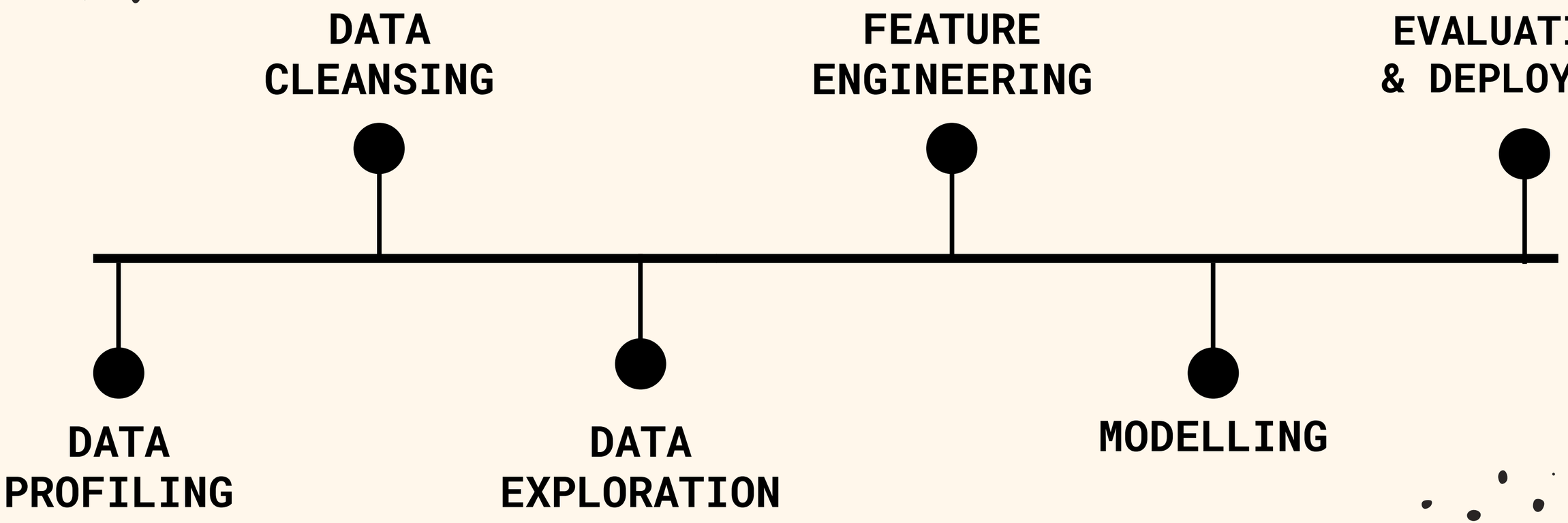
Rizal Maulana K.

Shafira Aisyah

Outline



Machine Learning Workflow


DATA CLEANSING
FEATURE ENGINEERING
EVALUATION & DEPLOYMENT
DATA PROFILING
DATA EXPLORATION
MODELLING

Requirements of Machine Learning

There are 3 Requirements of Machine Learning:

1. There must be data
2. The data must have a pattern
3. The algorithm is hard to be processed

Data Profiling

Definition

Data profiling is the process of reviewing source data, understanding structure, content and interrelationships, and identifying potential for data projects.

Example

●import packages

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split
from sklearn.ensemble import ExtraTreesClassifier

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

●load dataset

```
df = pd.read_csv('Titanic.csv')
```

Example

●preview dataset

```
df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

●info dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Example

- check missing value

```
df.isna().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

#Quiz 1

- load dataset

```
#Quiz 1
dt = pd.read_csv('data_telco.csv')
```

- preview dataset

```
dt.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSupp
0	7590-VHVEG	NaN	NaN	NaN	NaN	1	No	NaN	NaN	NaN	...	NaN	N
1	5575-GNVDE	NaN	NaN	NaN	NaN	34	Yes	NaN	NaN	NaN	...	NaN	N
2	3668-QPYBK	NaN	NaN	NaN	NaN	2	Yes	NaN	NaN	NaN	...	NaN	N
3	7795-CFOCW	NaN	NaN	NaN	NaN	45	No	NaN	NaN	NaN	...	NaN	N
4	9237-HQITU	NaN	NaN	NaN	NaN	2	Yes	NaN	NaN	NaN	...	NaN	N

5 rows × 21 columns

#Quiz 1

●info dataset

```
dt.info()
```

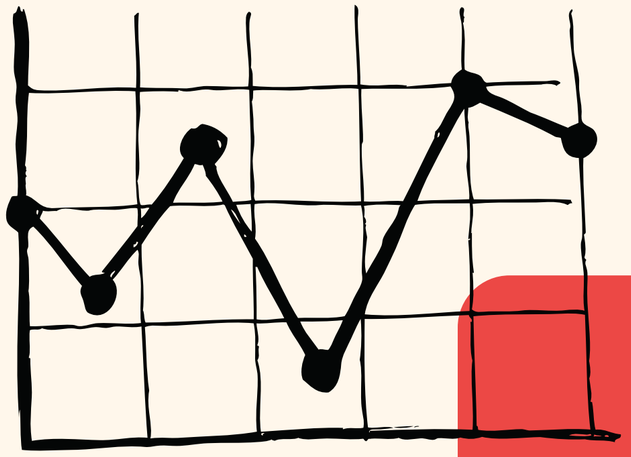
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            6034 non-null   object 
 2   SeniorCitizen     6034 non-null   float64
 3   Partner           6034 non-null   object 
 4   Dependents        6034 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     6034 non-null   object 
 8   InternetService   6034 non-null   object 
 9   OnlineSecurity    6034 non-null   object 
 10  OnlineBackup      6034 non-null   object 
 11  DeviceProtection  6034 non-null   object 
 12  TechSupport       6034 non-null   object 
 13  StreamingTV       6034 non-null   object 
 14  StreamingMovies   6034 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(2), int64(1), object(18)
memory usage: 1.1+ MB
```

#Quiz 1

- check missing value

```
dt.isna().sum()
```

```
customerID          0
gender           1009
SeniorCitizen    1009
Partner          1009
Dependents       1009
tenure              0
PhoneService        0
MultipleLines    1009
InternetService  1009
OnlineSecurity   1009
OnlineBackup     1009
DeviceProtection 1009
TechSupport      1009
StreamingTV      1009
StreamingMovies  1009
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```



Data Cleansing

Definition

Data cleansing is the process of detecting and correcting (or removing) 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.

Example

we will fill the empty data

```
val = df["Age"].median()
df['Age'] = df['Age'].fillna(val)
```

```
#Column Embarked
df['Embarked'] = df['Embarked'].fillna('Unknown')
```

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

```
S          644
C          168
Q           77
Unknown      2
Name: Embarked, dtype: int64
```

Example

we will delete the unnecessary column

```
#delete cabin column
df.drop('Cabin',axis=1, inplace=True)
```

we will merge or add new information based on column

```
#Column SibSp and Parch
df['Alone']=df['SibSp']+df['Parch']
```

```
df['Alone'][df['Alone']>0]='With Family'
df['Alone'][df['Alone']==0]='Without Family'
```

Example

result of data df

```
df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Alone
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S	With Family
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C	With Family
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S	Without Family
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S	With Family
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	S	Without Family

#Quiz 2

we will fill the empty data

```
#Quiz 2
dt['Partner'] = dt['Partner'].fillna('Unknown')
```

```
dt['Partner'].value_counts()
```

```
No         3111
Yes        2923
Unknown    1009
Name: Partner, dtype: int64
```

we will change the data type

```
#change data type
extr = dt['TotalCharges'].str.extract(r'^(\d{4})', expand=False)
dt['TotalCharges'] = pd.to_numeric(extr)
```

Exploratory Data Analysis

Definition

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns,to spot anomalies,to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

Example

Describe table

```
#Describe
df.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.361582	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	13.019697	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	22.000000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	35.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Example

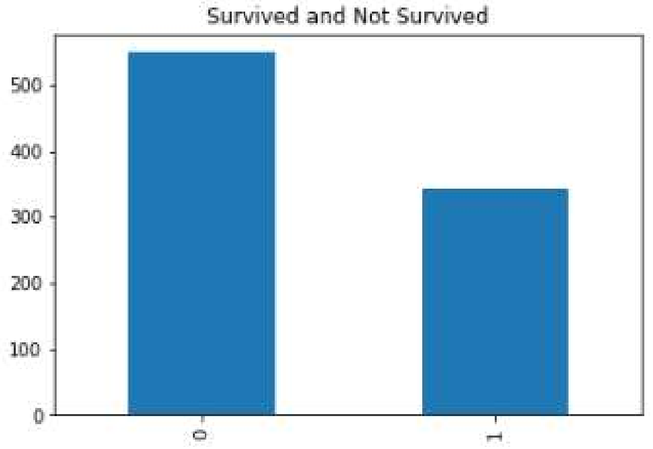
Change description row in a column and plot it

```
#Survived and non Survived
df['Survived'].value_counts()
```

```
0    549
1    342
Name: Survived, dtype: int64
```

```
df['Survived'].value_counts().plot(kind='bar');
plt.title('Survived and Not Survived')
```

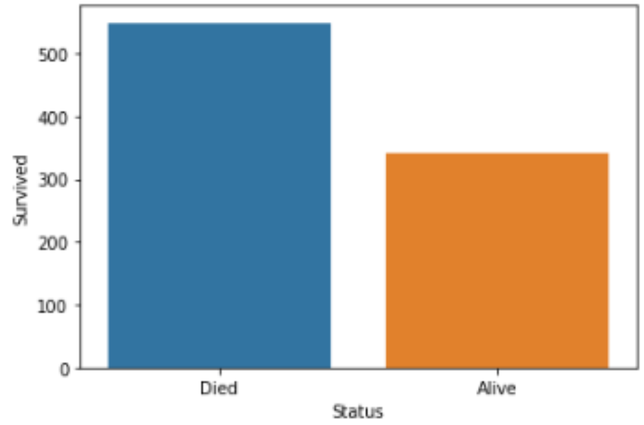
```
Text(0.5, 1.0, 'Survived and Not Survived')
```




```
df_survived2 = pd.DataFrame(df['Survived'].value_counts())
df_survived2['Status'] = ['Died', 'Alive']
df_survived2
```

	Survived	Status
0	549	Died
1	342	Alive

```
sns.barplot(x='Status', y = 'Survived', data=df_survived2);
```



Multicollinearity

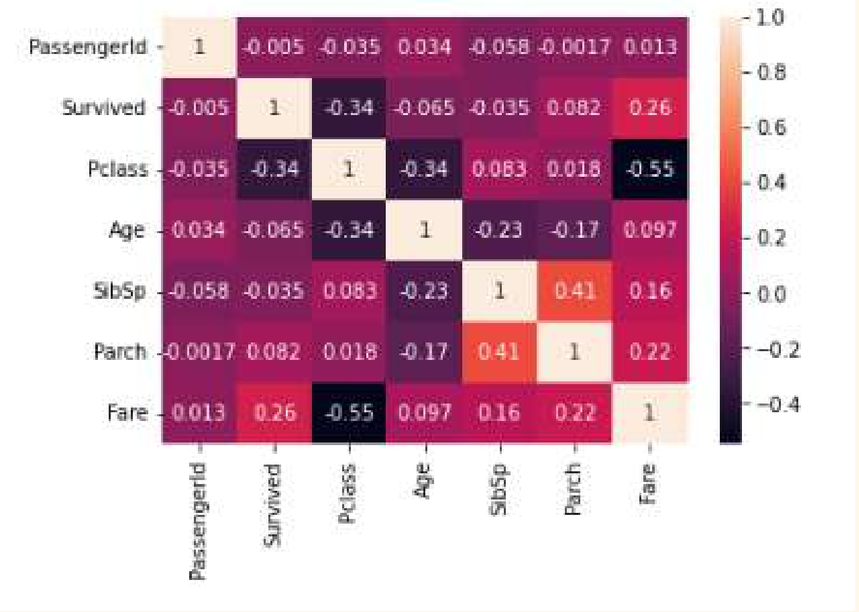
Multicollinearity occurs when one independent variable in a regression model is linearly correlated with another independent variable.

The way to detect multicollinearity in the regression model is by looking at the strength of the correlation between the independent variables. If there is a correlation between independent variables > 0.5, it can be indicated that there is multicollinearity.

Example

Checking Multicollinearity

```
#Correlation
plt.figure(figsize = (6,4))
sns.heatmap(df.corr(), annot = True);
```



there is multicollinearity in Pclass and Fare

#Quiz 3

Describe table

```
#Quiz 3
dt.describe()
```

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges
count	6034.000000	7043.000000	7043.000000	4139.000000
mean	0.161916	32.371149	64.761692	3628.141580
std	0.368404	24.559481	30.090047	2066.601044
min	0.000000	0.000000	18.250000	1001.000000
25%	0.000000	9.000000	35.500000	1743.000000
50%	0.000000	29.000000	70.350000	3205.000000
75%	0.000000	55.000000	89.850000	5237.500000
max	1.000000	72.000000	118.750000	8684.000000

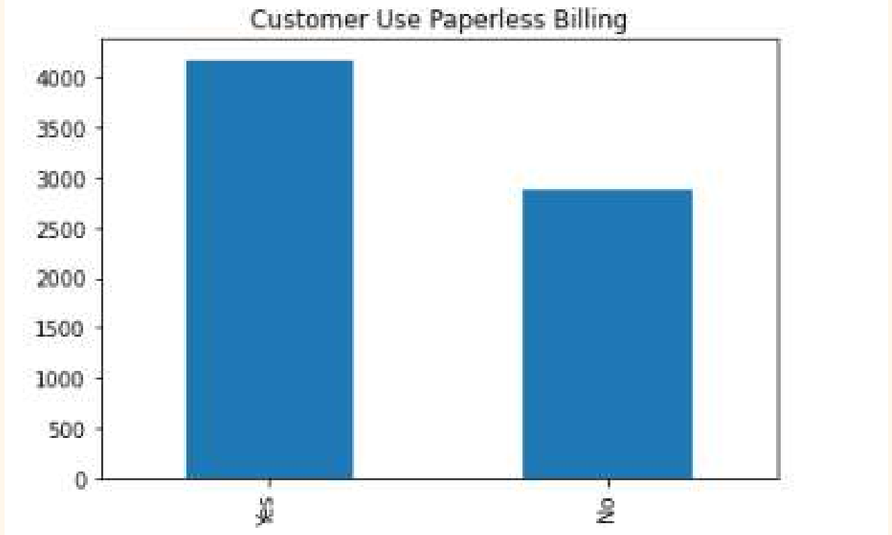
#Quiz 3

plotting the column PaperlessBilling

```
dt['PaperlessBilling'].value_counts()
```

```
Yes    4171
No     2872
Name: PaperlessBilling, dtype: int64
```

```
dt['PaperlessBilling'].value_counts().plot(kind='bar')
plt.title('Customer Use Paperless Billing');
```



#Quiz 3

Checking Multicollinearity

```
plt.figure(figsize = (10,5))
sns.heatmap(dt.corr(), annot = True);
```

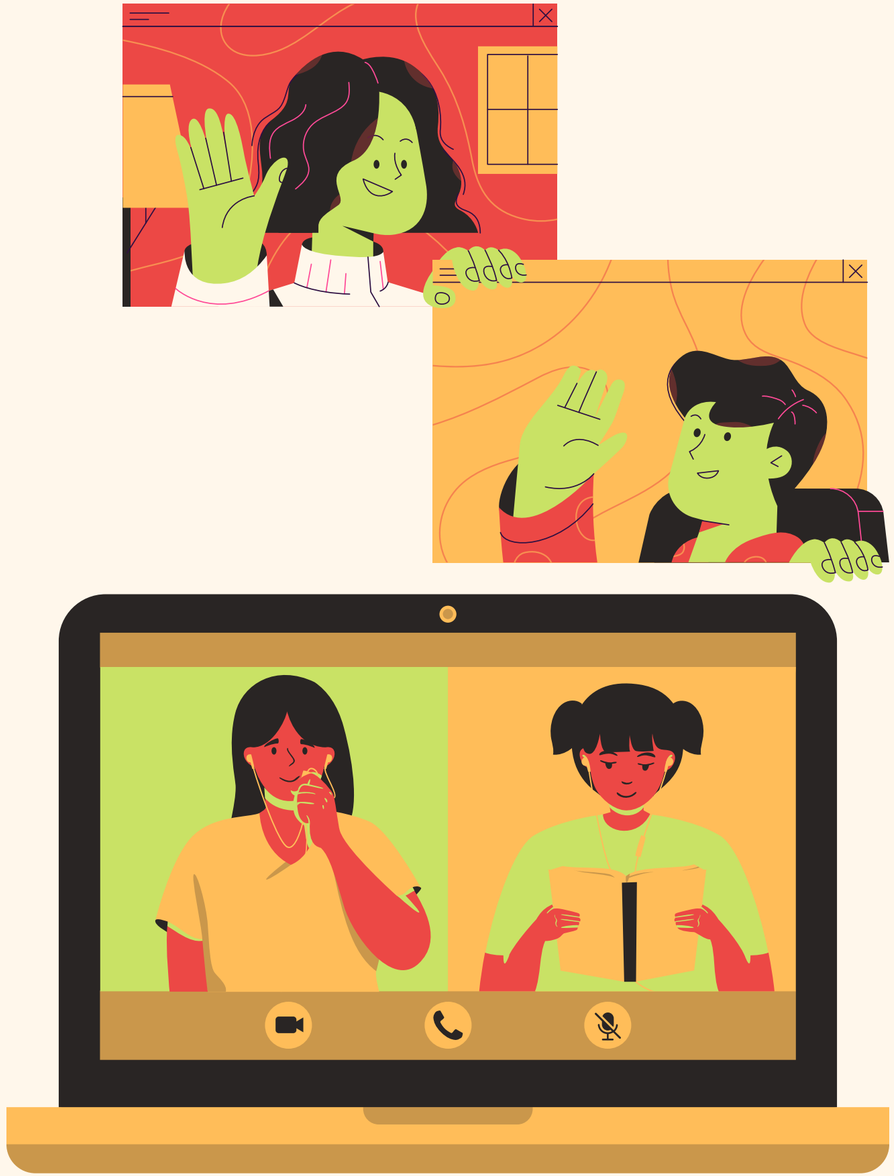
there are multicollinearity in tenure - TotalCharges, and MonthlyCharges - TotalCharges

Feature Engineering

Feature Engineering For Machine Learning

DEFINITION

Feature engineering is the process of selecting, manipulating, and transforming raw data into features that can be used for creating a predictive model using Machine learning or statistical Modelling.



One-Hot Encoding

One-hot encoding is a process of **converting categorical data variables** so they can be provided to machine learning algorithms to improve predictions.

Note : The Python library Pandas provides a function called *get_dummies* to enable one-hot encoding

Example:

```
# titanic dataset
embarked_enc = pd.get_dummies(df['Embarked'], prefix="Embarked", drop_first = False)
embarked_enc
```

Before one-hot encoding

Embarked
S
C
S
S
S
...
S

	Embarked_C	Embarked_Q	Embarked_S	Embarked_unknown
0	0	0	1	0
1	1	0	0	0
2	0	0	1	0
3	0	0	1	0
4	0	0	1	0
...	...	...	...	...
886	0	0	1	0

After one-hot encoding

Other Case

NUMERICAL DATA VARIABLES

To convert **numerical data variables** can use the **map()** function

Example:

```
# titanic dataset
df['Pclass'] = df['Pclass'].map({1:0, 2:1, 3:2})
```

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

```
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

Before

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

```
2    491
0    216
1    184
Name: Pclass, dtype: int64
```

After

Scaling

DEFINITION

Feature scaling is about transforming the values of different numerical features to fall within a similar range like each other. The feature scaling is used to prevent the supervised learning models from getting biased toward a specific range of values.

StandardScaler For Standardization

StandardScaler is a class from sklearn.preprocessing which is used for **standardization.**

Note : Standardization is used to center the feature columns at mean 0 with a standard deviation of 1 so that the feature columns have the same parameters as a standard normal distribution.

Example:

```
# titanic dataset
scaler = StandardScaler()

df[['Age']] = scaler.fit_transform(df[['Age']])
df[['Fare']] = scaler.fit_transform(df[['Fare']])
```

	Age	Fare
0	22.0	7.2500
1	38.0	71.2833
2	26.0	7.9250
3	35.0	53.1000
4	35.0	8.0500
...	...	...
886	27.0	13.0000

Before

	Age	Fare
0	-0.565736	-0.502445
1	0.663861	0.786845
2	-0.258337	-0.488854
3	0.433312	0.420730
4	0.433312	-0.486337
...	...	...
886	-0.181487	-0.386671

After

MinMaxScaler For Normalization

MinMaxScaler is a class from sklearn.preprocessing which is used for **normalization.**

Note : Normalization refers to the rescaling of the features to a range of [0, 1], which is a special case of min-max scaling.

Example:

```
# 50_Startups dataset
scaler =MinMaxScaler()

df['R&D Spend'] = scaler.fit_transform(df[['R&D Spend']])
df['Administration'] = scaler.fit_transform(df[['Administration']])
df['Marketing Spend'] = scaler.fit_transform(df[['Marketing Spend']])
df.head()
```

	R&D Spend	Administration	Marketing Spend
0	165349.20	136897.80	471784.10
1	162597.70	151377.59	443898.53
2	153441.51	101145.55	407934.54
3	144372.41	118671.85	383199.62
4	142107.34	91391.77	366168.42

Before

	R&D Spend	Administration	Marketing Spend
0	1.000000	0.651744	1.000000
1	0.983359	0.761972	0.940893
2	0.927985	0.379579	0.864664
3	0.873136	0.512998	0.812235
4	0.859438	0.305328	0.776136

After

Preprocessing
Modelling

Preprocessing Modeling

DEFINITION

Preprocessing is the most important aspect of data processing. When data is acquired as an output of an experiment, the next step is modeling the data to extract useful information.

Feature Selection

Feature selection is the process of **selecting the features** that contribute the most to the prediction variable or output that you are interested in, either automatically or manually.

Example:

```
df.drop('Name', axis=1, inplace = True) #too unique

df.drop('Ticket', axis=1, inplace = True) #too unique

df.drop('Fare', axis=1, inplace = True) #multicolinear

df.drop('PassengerId', axis=1, inplace = True) #identifier
```

Methods to perform Feature Selection

FEATURE IMPORTANCE

Feature importance assigns a score to each of your data’s features; the higher the score, the more important or relevant the feature is to your output variable.

CORRELATION STATISTICS

Correlation describes the relationship between the features and the target variable.

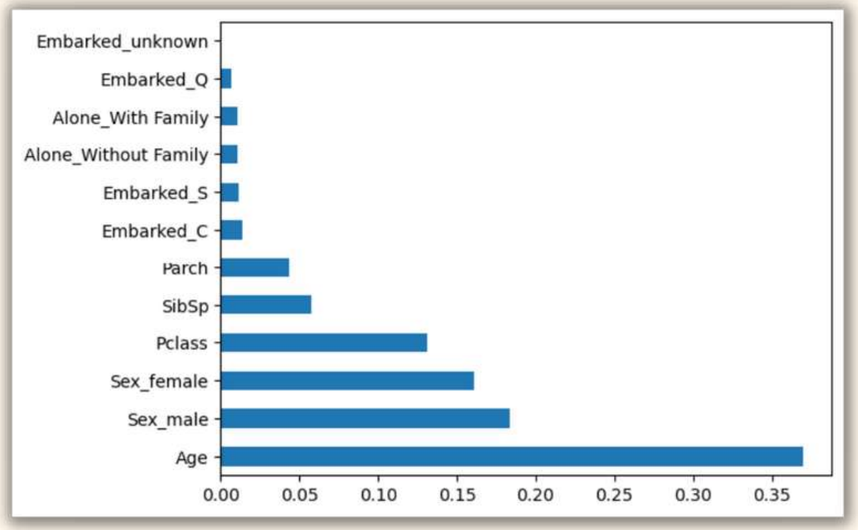
Feature Importance

Extra Trees Classifier is a type of ensemble learning technique which aggregates the results of multiple de-correlated decision trees collected in a “forest” to output it’s classification result.

Example:

```
X = df.drop(['Survived'], axis=1)
y = df['Survived']
```

```
from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier()
model.fit(X,y)

feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(23).plot(kind = 'barh')
plt.show()
```

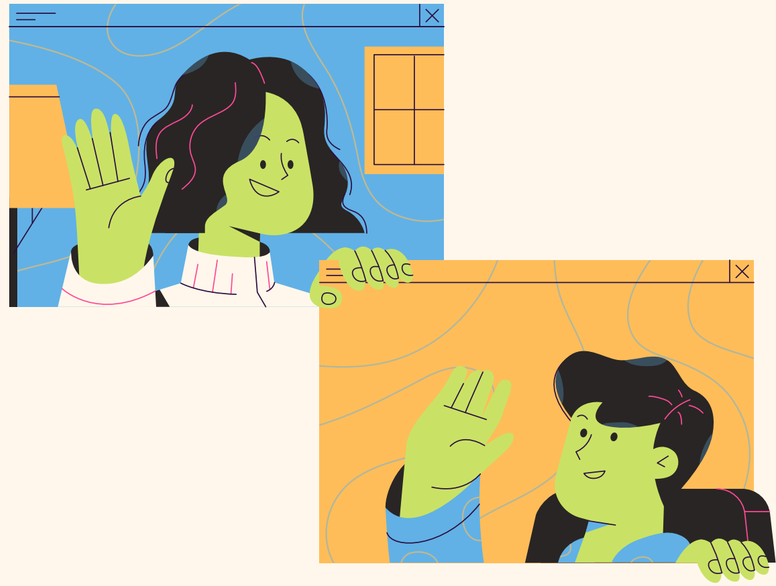


Train Test Split

DEFINITION

The train-test split is a technique for evaluating the performance of a machine learning algorithm.

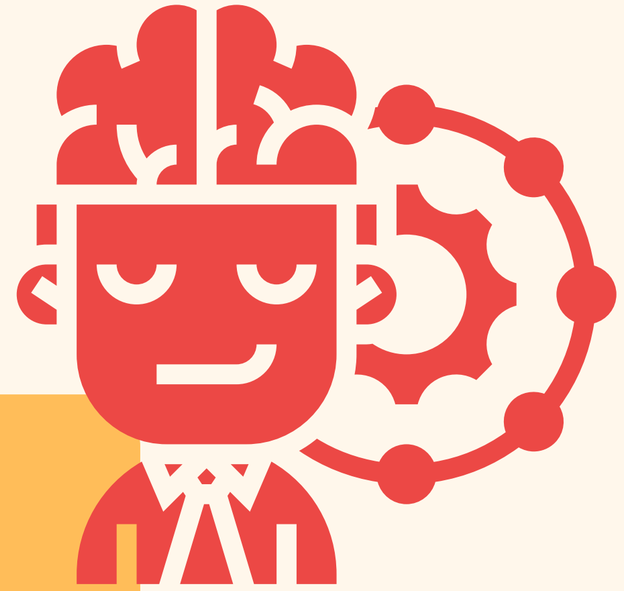
```
sklearn.model_selection.train_test_split(
*arrays, test_size=None, train_size=None,
random_state=None, shuffle=True,
stratify=None)
```

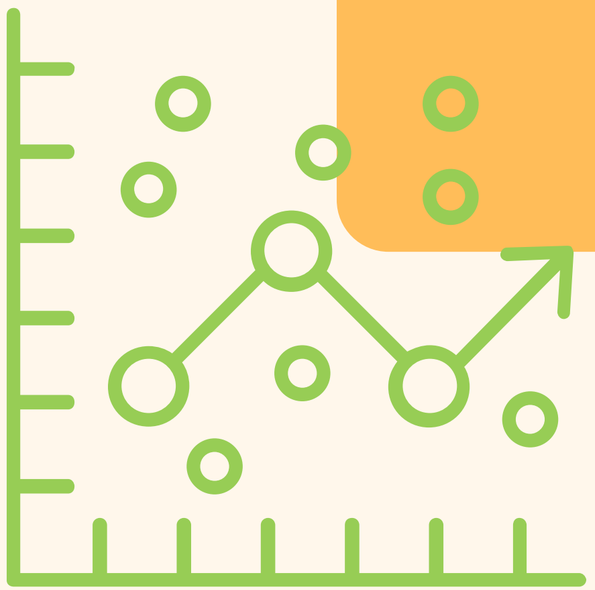


Example:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, random_state = 42)
```

Machine Learning
Regression





Regression

Situation:

- there is a single response variable Y, also called the dependent variable
- depends on the vakue of a set of input, aslo called independent varaibles (X1, X2, ..., Xr)

The simplest type of relationship these vatiables is a linear relationship

$$Y = \beta_0 + \beta_1 x_1 + \cdots + \beta_r x_r$$

Linear Regression Assumption

Relation between dependant variable and independent variable is linear

Error or residual of the model need to be normally distributed

There is no multicolineraity

Simple Linear Regression (SLR)

DEFINITION

Simple linear regression is a statistical method that allows us to summarize and study relationships between two continuous (quantitative) variables.

Data Profiling

Import packages

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_absolute_percentage_error

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

```
df = pd.read_csv('Salary_Data.csv')
```

```
df.head()
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

Load dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     float64
dtypes: float64(2)
memory usage: 608.0 bytes
```

```
df.isna().sum()
```

```
YearsExperience    0
Salary             0
dtype: int64
```

Check missing value

Data Cleansing

DATA IS CLEAN

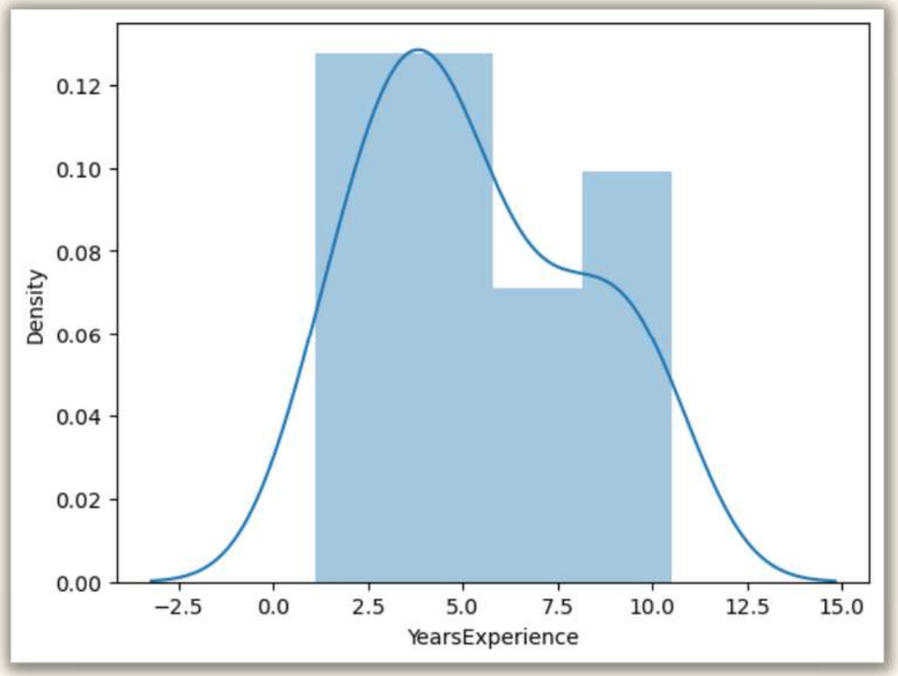
```
df.isna().sum()

YearsExperience    0
Salary             0
dtype: int64
```

Exploratory Data Analysis

```
df.describe()
```

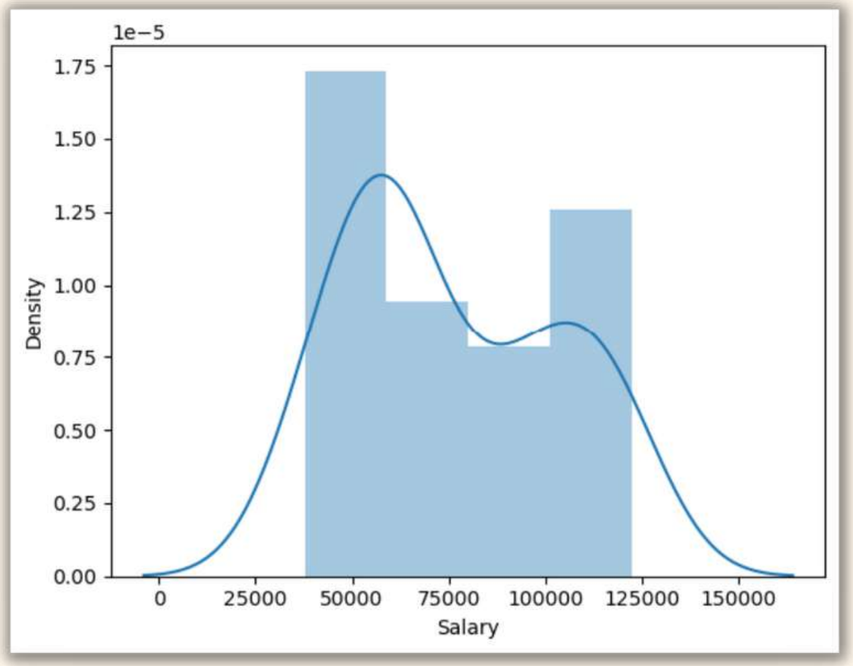
	YearsExperience	Salary
count	30.000000	30.000000
mean	5.313333	76003.000000
std	2.837888	27414.429785
min	1.100000	37731.000000
25%	3.200000	56720.750000
50%	4.700000	65237.000000
75%	7.700000	100544.750000
max	10.500000	122391.000000

from the table above can be seen that the average work experience of employees is 5 years



Histogram of YearsExperience

Histogram of Salary



Preprocessing Modeling

```
X = df.drop(['Salary'], axis=1)
y = df['Salary']
```

Defines X variable and y variable

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 1/3, random_state = 42)
```

Splitting Training and Test Set

Modeling

```
regressor = LinearRegression()
```

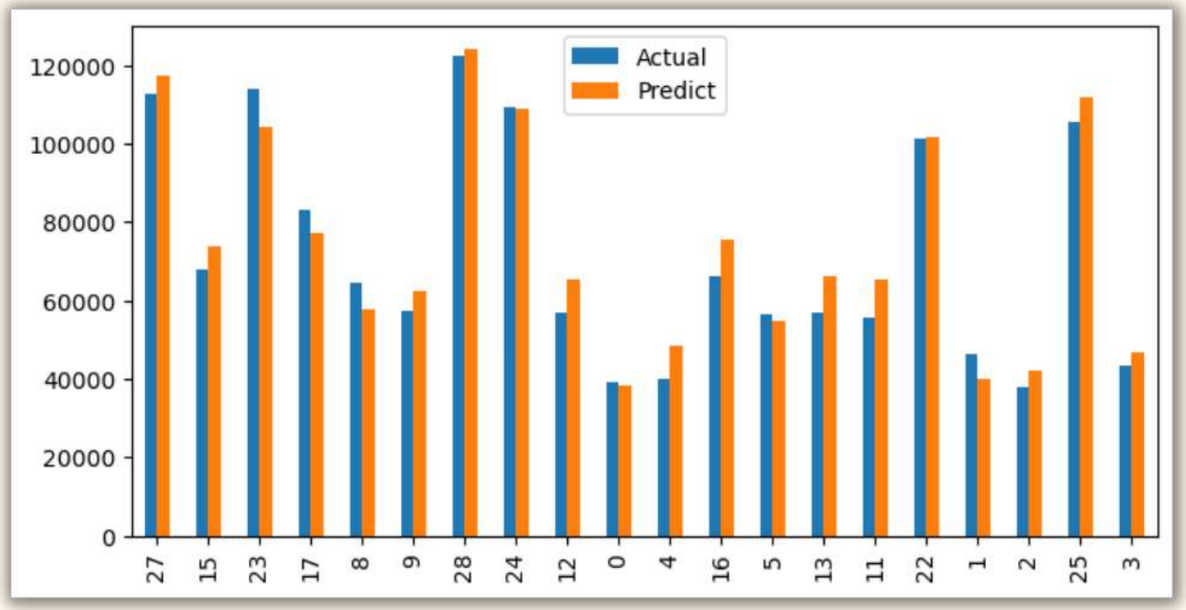
```
regressor.fit(X_train, y_train)
```

```
LinearRegression()
```

```
regressor.coef_
```

```
array([9337.63985893])
```

Fitting into training




```
y_pred = regressor.predict(X_test)
```

```
result = pd.DataFrame({'Actual':y_test, 'Predict':y_pred})
result
```

	Actual	Predict
27	112635.0	117549.897898
15	67938.0	73662.990561
23	113812.0	104477.202095
17	83088.0	77398.046504
8	64445.0	57789.002801
9	57189.0	62457.822730

Predict the reult

Plot the result

Evaluate Model

```
mean_absolute_error(y_test, y_pred)
```

```
5373.379510407303
```

```
mean_absolute_percentage_error(y_test, y_pred)
```

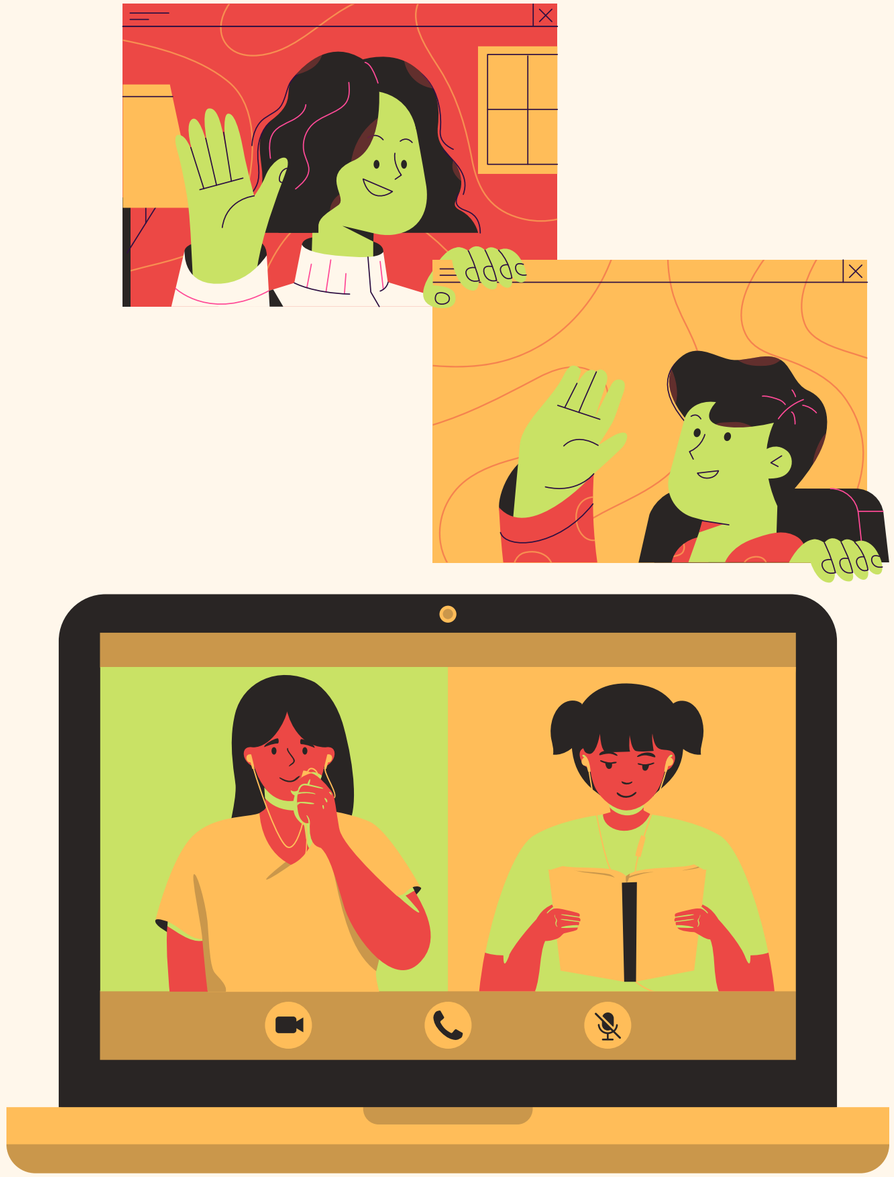
```
0.08804898617394867
```

Based on the results above, the accuracy of the model is **great** because the MAPE is below 10%

Multiple Linear Regression (MLR)

DEFINITION

Multiple linear regression is a statistical method to estimate the relationship between two or more independent variables and one dependent variable.



Data Profiling

Import packages

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_absolute_percentage_error

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

Load dataset

```
df = pd.read_csv('50_Startups.csv')
df.head()
```

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   R&D Spend        50 non-null     float64
 1   Administration   50 non-null     float64
 2   Marketing Spend  50 non-null     float64
 3   State            50 non-null     object
 4   Profit           50 non-null     float64
```

Check missing value

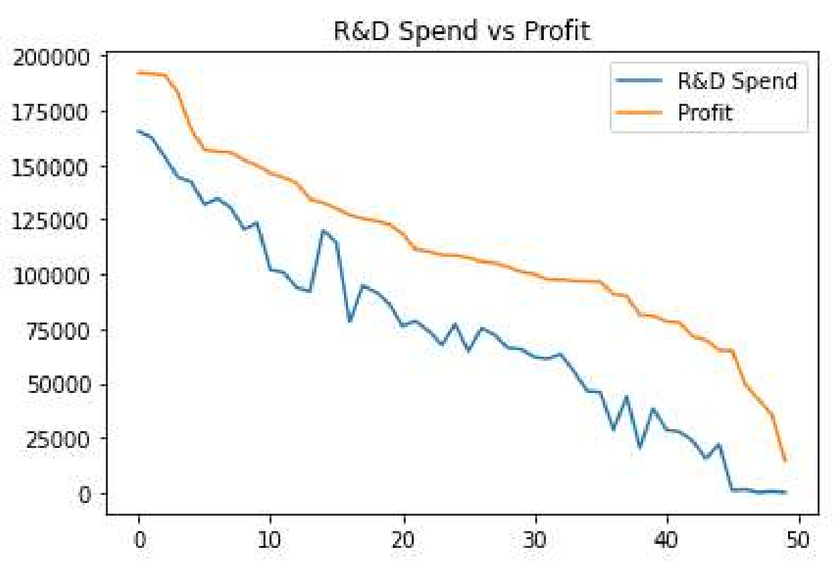
Data Cleansing

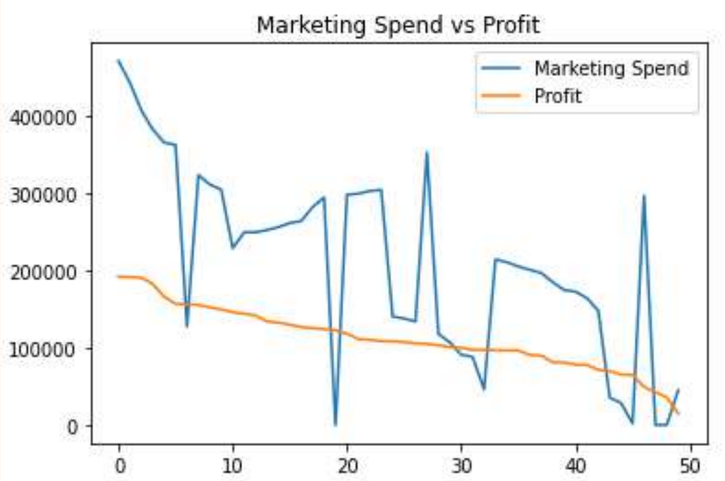
DATA ALREADY CLEAN

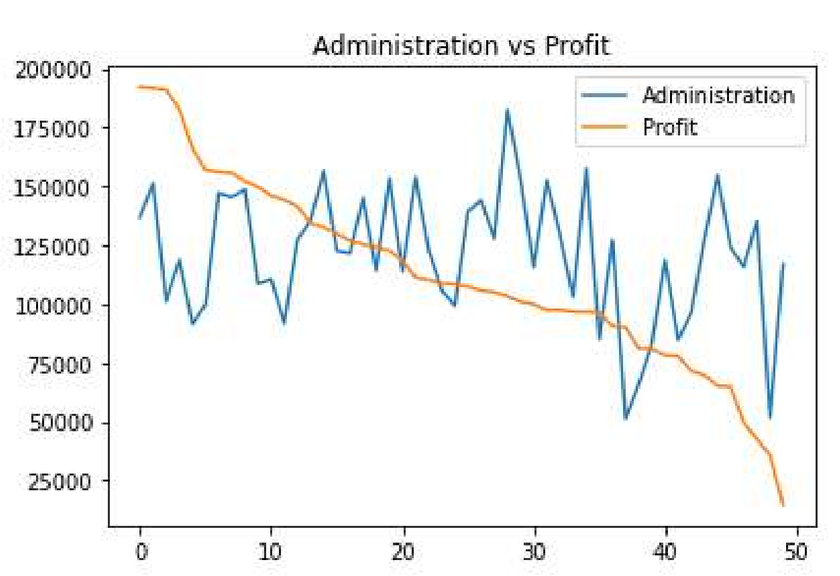
```
df.isna().sum()
```

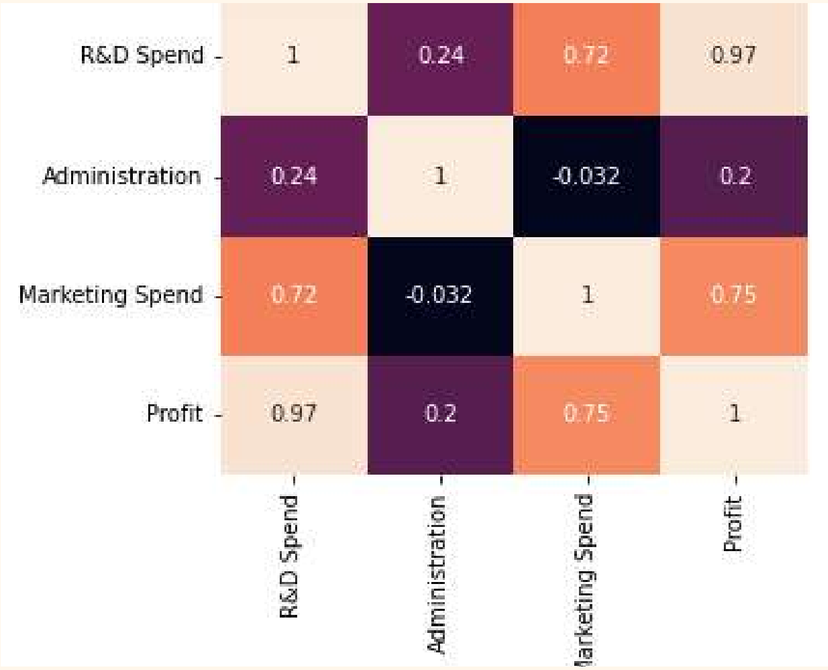
```
R&D Spend          0
Administration     0
Marketing Spend    0
State              0
Profit             0
```

Exploratory Data Analysis









In this heatmap we can see multicolinear between R&D Spend and Marketing Spend so we need to remove 1 of these

Feature Engineering

In this Dataset we can see State field is filled with datatype object so we need to change it to numeric by using one hot enconding

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

```
State = pd.get_dummies(df['State'], prefix='State',drop_first=False)
df.drop(['State'],axis=1,inplace=True)
df = pd.concat([df,State], axis=1)
```

	R&D Spend	Administration	Marketing Spend	Profit	State_California	State_Florida	State_New York
0	165349.20	136897.80	471784.10	192261.83	0	0	1
1	162597.70	151377.59	443898.53	191792.06	1	0	0
2	153441.51	101145.55	407934.54	191050.39	0	1	0
3	144372.41	118671.85	383199.62	182901.99	0	0	1
4	142107.34	91391.77	366168.42	166187.94	0	1	0

Feature Engineering

	R&D Spend	Administration	Marketing Spend	Profit	State_California	State_Florida	State_New York
0	165349.20	136897.80	471784.10	192261.83	0	0	1
1	162597.70	151377.59	443898.53	191792.06	1	0	0
2	153441.51	101145.55	407934.54	191050.39	0	1	0
3	144372.41	118671.85	383199.62	182901.99	0	0	1
4	142107.34	91391.77	366168.42	166187.94	0	1	0

after doing one hot encoding we need to scaling data to equalize the values

```
scaler = MinMaxScaler()
df['R&D Spend'] = scaler.fit_transform(df[['R&D Spend']])
df['Administration'] = scaler.fit_transform(df[['Administration']])
df['Marketing Spend'] = scaler.fit_transform(df[['Marketing Spend']])
df.head()
```

	R&D Spend	Administration	Marketing Spend	Profit	State_California	State_Florida	State_New York
0	1.000000	0.651744	1.000000	192261.83	0	0	1
1	0.983359	0.761972	0.940893	191792.06	1	0	0
2	0.927985	0.379579	0.864664	191050.39	0	1	0
3	0.873136	0.512998	0.812235	182901.99	0	0	1
4	0.859438	0.305328	0.776136	166187.94	0	1	0

Preprocessing Modeling

Remove Multicolinear field

```
df.drop('Marketing Spend',axis=1,inplace=True) #Remove Multicolinear
```

Defines X variable and y variable

```
X = df.drop(['Profit'], axis=1)
y = df['Profit']
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.8,random_state=1)
```

Splitting Training and Test Set

Modeling

```
model = LinearRegression()
model.fit(X_train, y_train)
```

```
LinearRegression()
```

Fitting model into training

```
df.drop('State_California', axis=1, inplace = True)
df.drop('State_Florida', axis=1, inplace = True)
df.drop('State_New York', axis=1, inplace = True)
df.drop('Administration', axis=1, inplace = True)
```

Remove useless field

See importance field

```
#get importance
importance = model.coef_
```

```
for i,v in enumerate(importance):
    print('Feature : %0d, score: %5f '%(i,v))
```

```
Feature : 0, score: 140813.624037
Feature : 1, score: -222.895459
Feature : 2, score: -219.309465
Feature : 3, score: 265.423501
Feature : 4, score: -46.114036
```

```
X = df.drop(['Profit'], axis=1)
y = df['Profit']
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.8,random_state=0)
```

```
model = LinearRegression()
model.fit(X_train, y_train)
```

```
LinearRegression()
```

Spliting test and train data and fitting again

Modeling

```
y_pred =  model.predict(X_test)
```

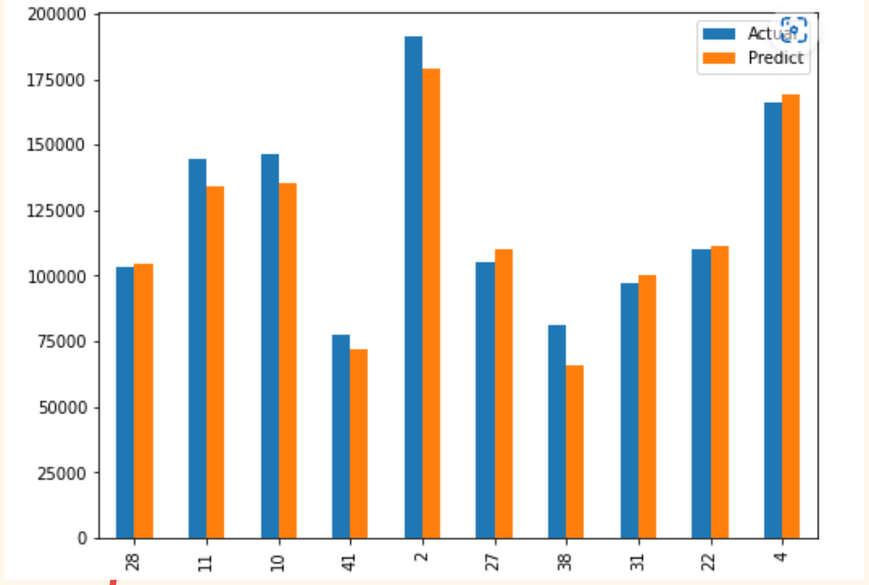
```
result = pd.DataFrame({'Actual': y_test, 'Predict': y_pred})
result
```

	Actual	Predict
28	103282.38	104667.278060
11	144259.40	134150.834106
10	146121.95	135207.800195
41	77798.83	72170.544289
2	191050.39	179090.586025
27	105008.31	109824.773866
38	81229.06	65644.277738

Predict the result

```
result.plot(kind='bar',figsize=(8,6))
```

```
<AxesSubplot:>
```



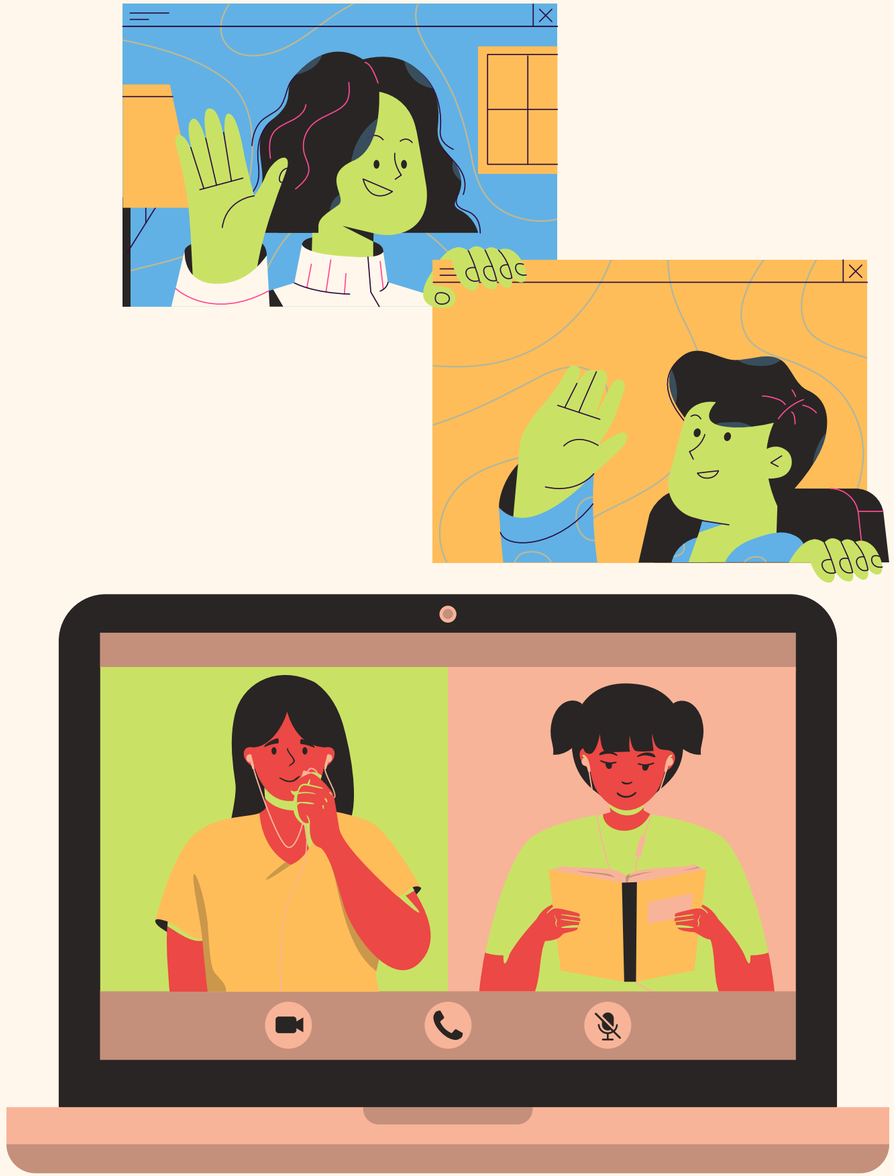
Plot the result

Time Series

Time Series

DEFINITION

- A Time Series is typically defined as a series of values that one or more variables take over successive time periods. For example, sales volume over a period of successive years, average temperature in a city over months etc.
- Therefore, Time Series forecast is about forecasting a variable’s value in future, based on it’s own past values.



MONTHLY-CAR-SALES DATASET

Data Profiling

Step 1 : Import the data and preview it.

```
#Load dataset
df = pd.read_csv('monthly-car-sales.csv')
```

```
#preview dataset
df.head()
```

	Month	Sales
0	1960-01	6550
1	1960-02	8728
2	1960-03	12026
3	1960-04	14395
4	1960-05	14587

Step 2 : Check is there any missing value or not

```
#info dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 108 entries, 0 to 107
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Month   108 non-null    object
 1   Sales   108 non-null    int64
dtypes: int64(1), object(1)
memory usage: 1.8+ KB
```

If the data have **missing value**, it **must be imputated** before continue the analysis

Data Cleansing

Step 1 : Change date column to datetime format

```
df['Month'] = pd.to_datetime(df['Month'])
df
```

	Month	Sales
0	1960-01-01	6550
1	1960-02-01	8728
2	1960-03-01	12026
3	1960-04-01	14395
4	1960-05-01	14587
...	...	...
103	1968-08-01	16722
104	1968-09-01	14385
105	1968-10-01	21342
106	1968-11-01	17180
107	1968-12-01	14577

108 rows × 2 columns

Step 2 : Change the column name and check the format of each column

```
df.columns = ['ds', 'y']
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 108 entries, 0 to 107
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype         
---  ------  --------------  -----         
 0   ds      108 non-null    datetime64[ns]
 1   y       108 non-null    int64         
dtypes: datetime64[ns](1), int64(1)
memory usage: 1.8 KB
```

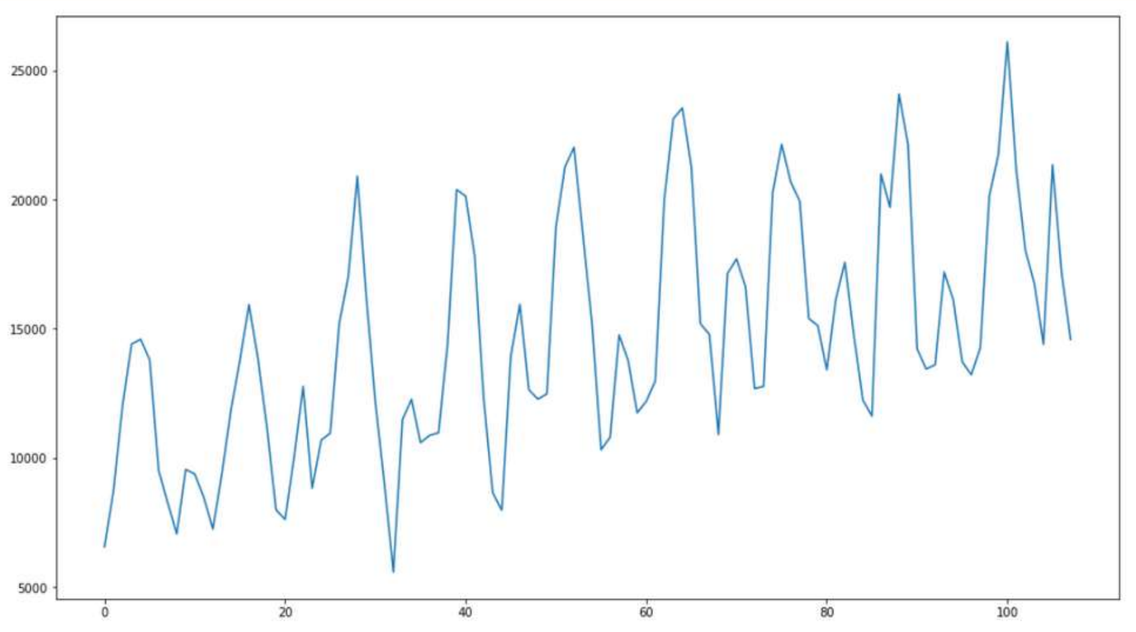
The change in the name of the column is do to make it **easier to call the specific column.**

MONTHLY-CAR-SALES DATASET

Exploratory Data Analysis

- Before analyze the data, its importance to investigate the data and summarize the key insights.
- One of them is view the time series plot to know the pattern

```
plt.figure(figsize=(16,9))
df['y'].plot()
plt.show()
```



The data pattern of the plot is **seasonal** and there is a trend

Preprocessing

- Train-test split

```
train = df.drop(df.index[-12:]) #drop last 12 months
test = df.drop(df.index[:-12]) #choose last 12 months
```

Splitting Training and Test Set

```
X_train = pd.DataFrame(train['ds'])
y_train = pd.DataFrame(train['y'])

X_test = pd.DataFrame(test['ds'])
y_test = pd.DataFrame(test['y'])
```

Name the column for the date with 'ds' and data with 'y'

In machine learning, train/test split splits the data randomly. But, in time series data the values at the rear of the dataset if for testing and everything else for training.

Modelling

Modelling the train data

```
model = Prophet()
model.fit(train)
```

```
11:09:33 - cmdstanpy - INFO - Chain [1] start processing
11:09:34 - cmdstanpy - INFO - Chain [1] done processing
```

```
<prophet.forecaster.Prophet at 0x268de392b20>
```

```
y_pred = model.predict(X_test)
```

```
yhat = y_pred['yhat'].values
```

```
yhat
```

```
array([14512.98439689, 14930.90562927, 20842.17617187, 23000.67913103,
       23350.15715933, 20972.26825767, 17359.53927678, 14785.20570325,
       13521.40595927, 17058.06396199, 17418.70122152, 15808.94518601])
```

- To modelling time series data, we used **prophet** module.
- Prophet models time series as a generalized additive model (GAM) combining the trend function, seasonality function, holiday effects, and an error term in one model

Evaluate Model

```
MAPE = mean_absolute_percentage_error(y_test, yhat)
MAPE
```

```
0.07187183098630126
```

MAPE is 7,1% then the predictions are on average 7,1% away from the actual values they were aiming for.

MAPE expresses **accuracy as a percentage of the error.** Because this number is a percentage, it can be easier to understand than the other statistics.

Data Profiling

SHAMPO DATASET

Import dataset

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

	Month	Sales
0	1-01	266.0
1	1-02	145.9
2	1-03	183.1
3	1-04	119.3
4	1-05	180.3

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36 entries, 0 to 35
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Month   36 non-null     object 
 1   Sales   36 non-null     float64
dtypes: float64(1), object(1)
memory usage: 704.0+ bytes
```

There is no missing value from the data. Analysis can be continued

Data Cleansing

SHAMPO DATASET

```
date = pd.date_range(start='1/1/2001', end='12/31/2003', freq='M')
date
```

```
DatetimeIndex(['2001-01-31', '2001-02-28', '2001-03-31', '2001-04-30',
               '2001-05-31', '2001-06-30', '2001-07-31', '2001-08-31',
               '2001-09-30', '2001-10-31', '2001-11-30', '2001-12-31',
               '2002-01-31', '2002-02-28', '2002-03-31', '2002-04-30',
               '2002-05-31', '2002-06-30', '2002-07-31', '2002-08-31',
               '2002-09-30', '2002-10-31', '2002-11-30', '2002-12-31',
               '2003-01-31', '2003-02-28', '2003-03-31', '2003-04-30',
               '2003-05-31', '2003-06-30', '2003-07-31', '2003-08-31',
               '2003-09-30', '2003-10-31', '2003-11-30', '2003-12-31'],
              dtype='datetime64[ns]', freq='M')
```

In this analysis, data will be used from specific range with a monthly range, then a range is carried out to sort the data according to the date needed

```
df['Time_Stamp'] = pd.DataFrame(date, columns=['Date'])
df.head()
```

	Month	Sales	Time_Stamp
0	1-01	266.0	2001-01-31
1	1-02	145.9	2001-02-28
2	1-03	183.1	2001-03-31
3	1-04	119.3	2001-04-30
4	1-05	180.3	2001-05-31

Change date column to datetime format

```
df.columns = ['y','ds']
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36 entries, 0 to 35
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   y       36 non-null     float64
 1   ds      36 non-null     datetime64[ns]
dtypes: datetime64[ns](1), float64(1)
memory usage: 704.0 bytes
```

Rename the column with 'y' for data, and 'ds' for date

Exploratory Data Analysis

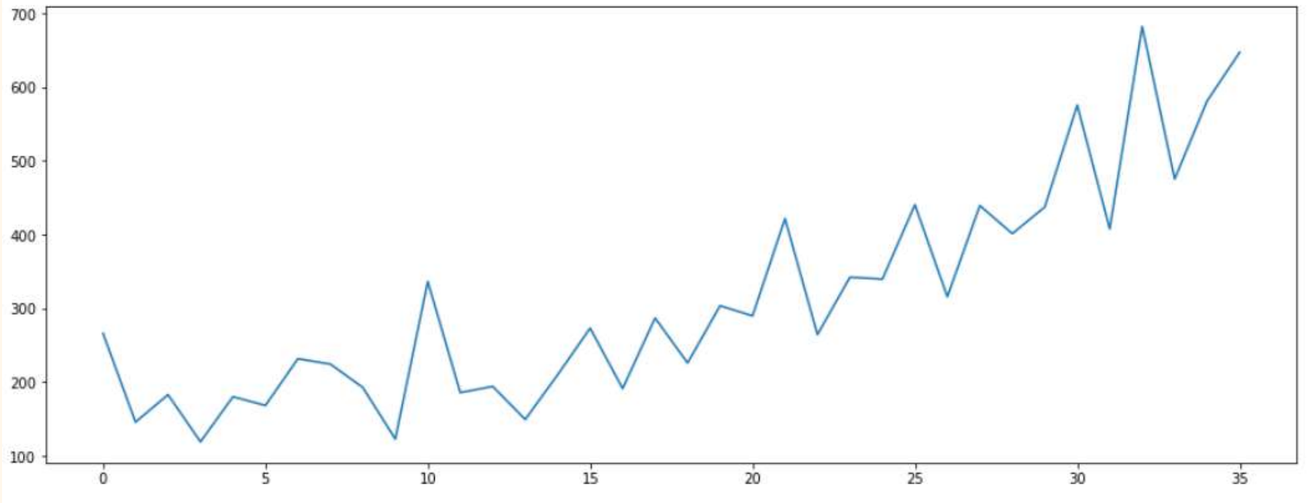
SHAMPO DATASET

To find out insights from the data, descriptive stats can also be displayed

```
df.describe()
```

	y
count	36.000000
mean	312.600000
std	148.937164
min	119.300000
25%	192.450000
50%	280.150000
75%	411.100000
max	682.000000

```
plt.figure(figsize=(16,6))
df['y'].plot()
```



The data pattern of the plot is **trend**

Preprocessing Modeling

SHAMPO DATASET

```
train = df.drop(df.index[-6:])
test = df.drop(df.index[:-6])
```

Splitting Training and Test Set

```
X_train = pd.DataFrame(train['ds'])
y_train = pd.DataFrame(train['y'])
```

```
X_test = pd.DataFrame(test['ds'])
y_test = pd.DataFrame(test['y'])
```

```
y_test
```

	y
30	575.5
31	407.6
32	682.0
33	475.3
34	581.3
35	646.9

Name the column for the date with 'ds' and data with 'y'

In time series data the values at the rear of the dataset if for testing and everything else for training.

Modeling

SHAMPO DATASET

Modelling the train data

```
model = Prophet()
model.fit(train)
```

```
11:13:40 - cmdstanpy - INFO - Chain [1] start processing
11:13:40 - cmdstanpy - INFO - Chain [1] done processing
```

```
y_pred = model.predict(X_test)
```

```
yhat = y_pred['yhat'].values
```

```
yhat
```

```
array([274.34717641, 441.04077116, 448.13720697, 773.23177526,
       240.82343363, 533.44942883])
```

Also, dont forget to predict the x_test and name it with yhat

Evaluate Model

SHAMPO DATASET

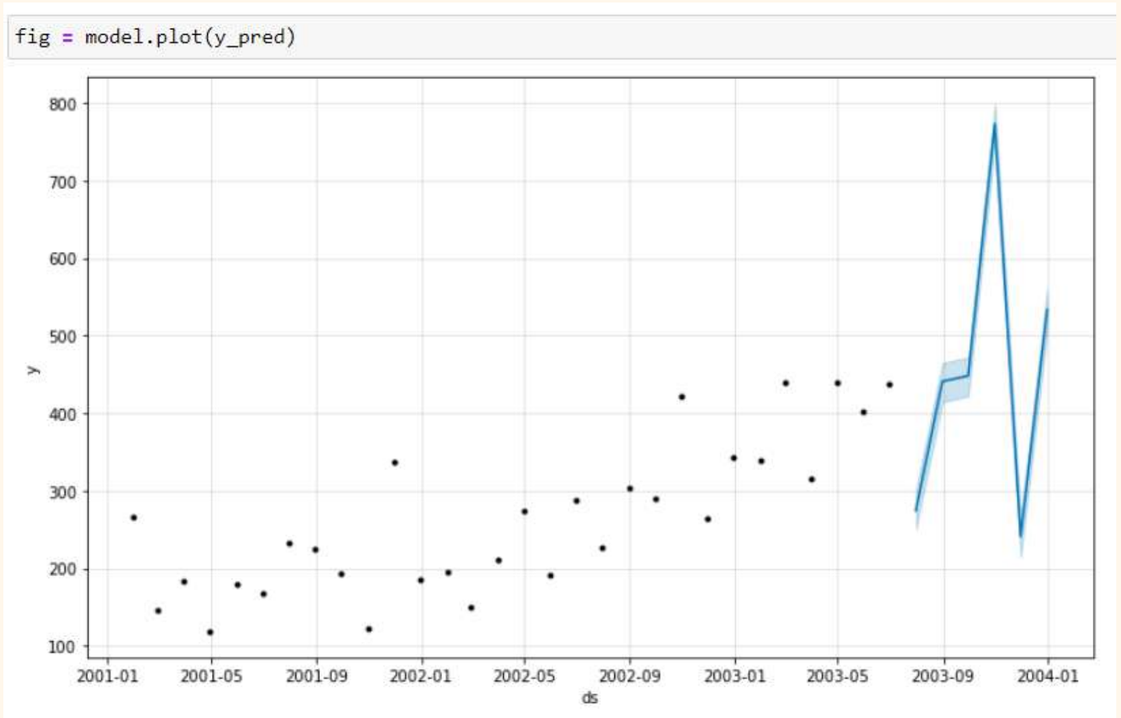
```
MAPE = mean_absolute_percentage_error(y_test, yhat)
MAPE
```

```
0.38935997239687486
```

MAPE is 38,936% then the predictions are on average 38,936% away from the actual values they were aiming for.

Plot of the prediction value.

```
fig = model.plot(y_pred)
```



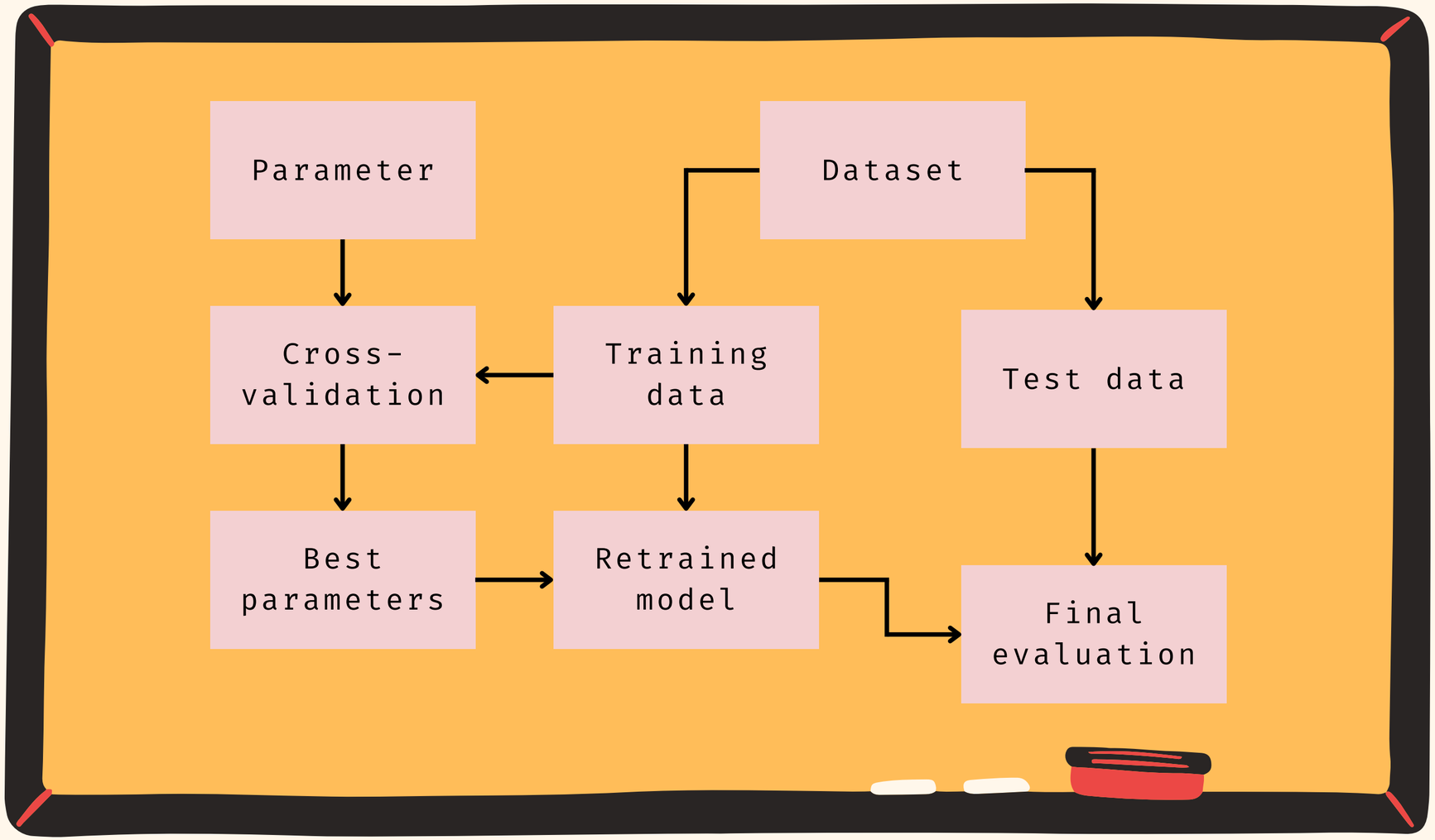
Cross Validation

Cross Validation

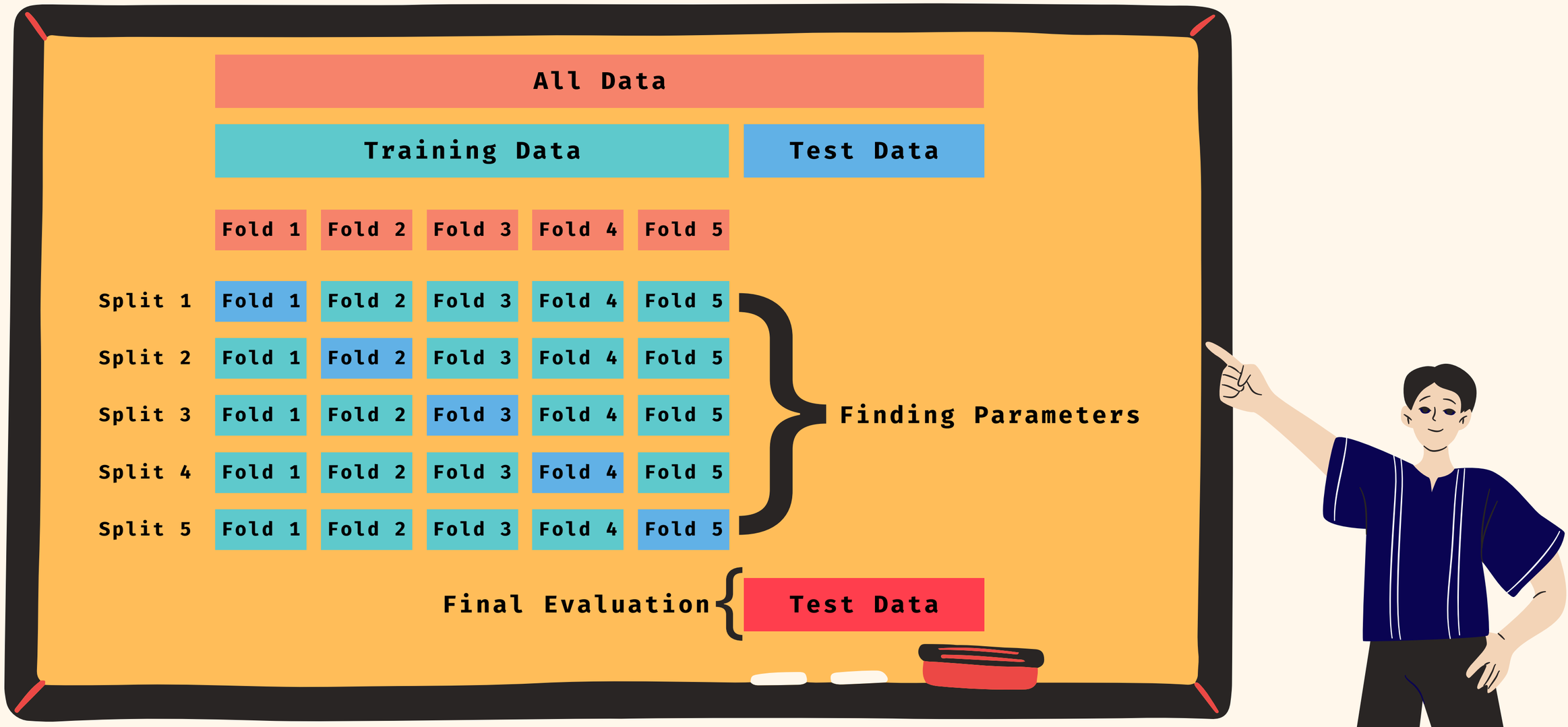
Cross-validation (CV) is a technique used to assess a machine learning model and test its performance (or accuracy). It involves reserving a specific sample of a dataset on which the model isn't trained. Later on, the model is tested on this sample to evaluate it.

Cross-validation is used to protect a model from overfitting, especially if the amount of data available is limited. It's also known as rotation estimation or out-of-sample testing and is mainly used in settings where the model's target is prediction.

Modelling Workflow



Cross Validation



Cross Validation

The picture at the previous slide shows how one fold in each step iteratively is held out for testing while the remaining folds are used to build the model. Each step calculates the prediction error. Upon the completion of the final step, a list of computed error are produced of which we can take the mean.

STARTUPS DATASET

First Evaluate Model

```
In [33]: mean_absolute_error(y_test, y_pred)

Out[33]: 6290.45960484629

In [34]: mape = mean_absolute_percentage_error(y_test, y_pred)
         mape

Out[34]: 0.0923794036122996 3
```

MAPE is 0,923%

STARTUPS DATASET

Building a Model With Cross Validation

- R square is a value that pays attention to how much the independent variable affects the dependent variable.
- If the value of R square is small, then the error component is large.
- If the value of R square is large, then the error component is small.

```
model = LinearRegression()
scores = cross_val_score(model, X_train, y_train, scoring = 'r2', cv=3)
scores
```

```
array([0.96052255, 0.97121584, 0.84760897])
```

Hyperparameter Tuning

Hyperparameter Tuning

There is a list of different machine learning models. They all are different in some way or the other, but what makes them different is nothing but input parameters for the model. These input parameters are named as **Hyperparameters**. These hyperparameters will define the architecture of the model, and the best part about these is that you get a choice to select these for your model. Of course, you must select from a specific list of hyperparameters for a given model as it varies from model to model.

Often, we are not aware of optimal values for **hyperparameters** which would generate the best model output. So, what we tell the model is to explore and select the optimal model architecture automatically. This selection procedure for hyperparameter is known as **Hyperparameter Tuning**.

Hyperparameter Tuning

STARTUPS DATASET

```
model.get_params()
```

```
{'copy_X': True,
 'fit_intercept': True,
 'n_jobs': None,
 'normalize': 'deprecated',
 'positive': False}
```

```
parameters = {"copy_X": [True, False],
              "fit_intercept": [True, False],
              "n_jobs": [None, -1],
              "normalize": [True, False],
              "positive": [True, False]
              }
```

We make parameter for model

Hyperparameter Tuning

STARTUPS DATASET

search = RandomizedSearchCV(estimator = regressor, param_distributions = parameters, cv=3, scoring = "r2")

```
grid = GridSearchCV(estimator = model, param_grid = parameters, cv=3)
```

```
best_model = grid.fit(X_train, y_train)
```

Get best parameter

```
best_model.best_params_
```

```
{'copy_X': True,
 'fit_intercept': True,
 'n_jobs': None,
 'normalize': True,
 'positive': True}
```

Do the GridSearch and get best parameters

Modelling with Hyperparameter Tuning

STARTUPS DATASET

Model with best parameter

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

Train with best parameter

```
regressor_new = model_new.fit(X_train, y_train)
```

```
y_pred_new = model_new.predict(X_test)
```

Modelling with best parameters and train it

Last Evaluate Model

STARTUPS DATASET

```
mape_new = mean_absolute_percentage_error(y_test, y_pred_new)
mape_new
```

```
0.0923794036122996
```

Difference of mape

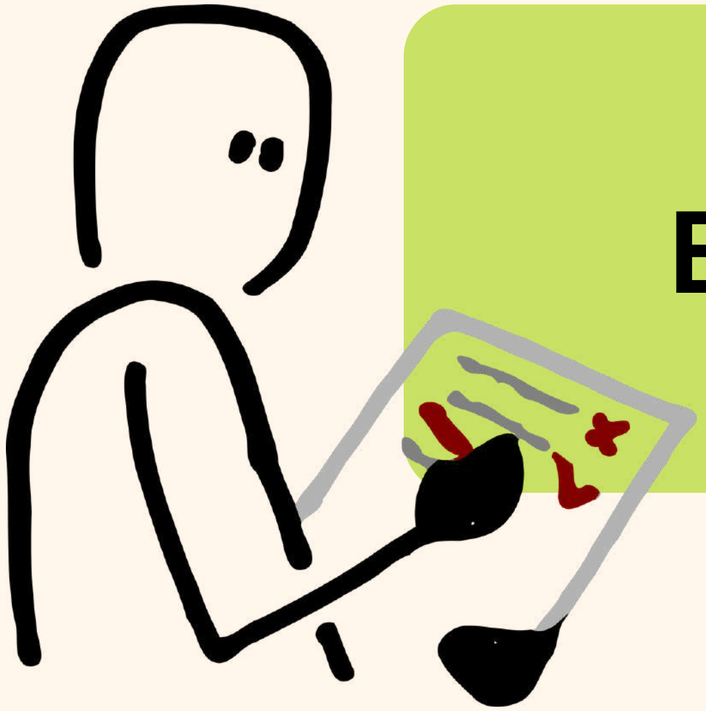
```
mape - mape_new
```

```
2.7755575615628914e-17
```

```
mape > mape_new
```

```
True
```

The old/first mape is bigger than new/last mape



Evaluate Model

MAPE

DEFINITION

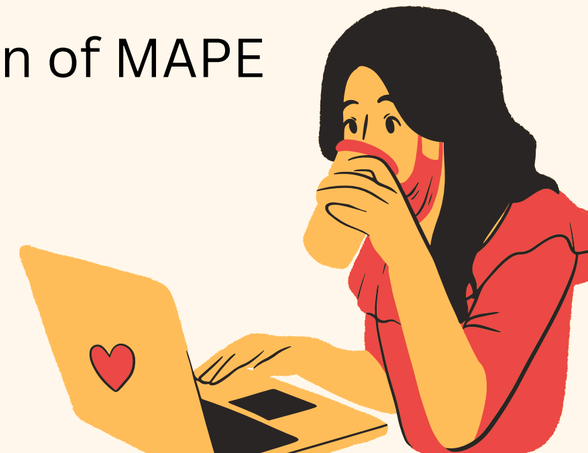
The Mean Absolute Percentage Error (MAPE) can be used in machine learning to measure the **accuracy of a model.**

The formula of MAPE

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

MAPE Score	Interpretation of Score
> 50 %	Poor
20% – 50%	Relatively good
10% – 20%	Good
< 10%	Great

Interpretation of MAPE



Thank You For Reading