Unit-Wise Concise Theory BA

UNIT-1

① Intro to BA : Role of Analytics for Data Driven Decision Making

→ Business Analytics means → "means utilizing tools, technique, and procedure to analyze past business data in order to gain insight and help in decision making & problem solving"

- Data is Everything, Analytics is necessary for survival
- Analytics can be used for process improvement (eg. using it to reduce time to deliver order), problem solving (predicting customer cancellation & frauds), decision making (switch to a new market etc).

② Introduction to concept of Big data Analytics

Big Data → refers to extremely large/complex sets of data that cannot be easily managed, processed or analysed with traditional data processing tools.

⇒ The goal of big data analytics is to extract valuable insights, patterns and knowledge

⇒ Big data is characterized by 3 Vs :-

① Volume (large amount of data), ② Velocity (how fast which data is generated & processed), ③ Variety (diff type of data like structured / unstructured)

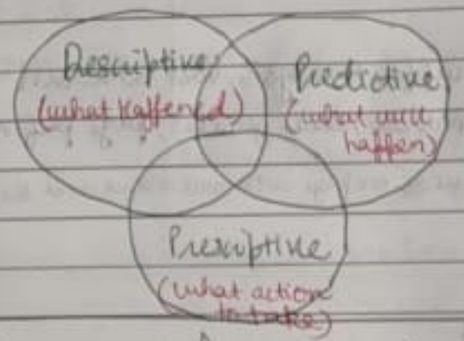
Sources of Big data:- Social media data, Satellite data, Banking, financial, healthcare data, E-commerce/retail, research data, Data from smart watches etc

- Structured data :- Has clear structure and follows a regular sequence. Organized formatted in a way which is easily readable by both Humans & machines. It generally follows tabular structure. Eg. ① Relational Database entries :- Employee data in company database with columns Name, ID, Department, Salary ② Spreadsheet :- Data in Excel or similar software like financial data with date, expense etc.

(Relational Database is used to handle complex, large datasets while spreadsheets are simpler, used for basic data manipulation & smaller datasets and calculations)

- Unstructured data :- lacks formal structure. Doesn't conform to fixed schema and is often human generated.
  Example :- word docs, pdfs, video clips, images, audio recordings etc

③ Types :- Descriptive Analytics, Predictive Analytics and Prescriptive Analytics



Descriptive is Numerical portion :- Covered in upcoming pages (one shot)

Descriptive	Predictive	Prescriptive
↓	↓	↓
Summarizes historical data to understand what happened in the past. [Tells you what happened in the past]	use historical data & statistical algorithms to make prediction about future outcomes. [Take educated guess of what will happen in the future]	goes a step further and recommends action to optimize or improve outcomes. gives advice/ action to achieve desired result [Specific action to achieve best outcome]
Eg. You run a e-commerce business, it will know value looking at the past data (sales) and answer ques. like :- ① what were our best selling products last year? ② which month had the highest sales?	Eg. Continuing with e-commerce one, it would forecast future sales of specific trends like sale of certain clothes ↑ during certain seasons.	Eg: Predictive analytics told us sale would go up during this period ∴ so prescriptive analytics might recommend ↑ing inventory of those products, running targeted ads or discounts

independent variable.

VIF = measures how much the variance of a regression coeff. is inflated due to multicollinearity in model. VIF=1 indicate no correlation among 'size' and other variable like bedroom, age etc. hence, no multicollinearity. As a rule of thumb VIF greater than 5-10 a problematic amount of multicollinearity.

Model Evaluation

mse = mean_squared_error(y_pred, y_test)
r2_score(y_train, model.predict(x_train))
r2_score(y_test, y_pred)

⇒ computes MSE b/w model predictions on test data and the actual value of the test data. [Actual value of y, and the value of y which model predicted.]. If MSE is less meaning our model is a better fit.

⇒ y_train :- actual value of y used in training
model.predict(x_train) :- predict values of y based on the training data of x. (MSE is ↓ if y_pred equal to y_test & is ↑ if they differ)

This line basically calculate R^2 for "training set" (x_trained

⇒ Similar to previous line it calculates R^2 but for "testing set". It compares the actual values (y_test) to the predicted values (y_pred). It measures how well unseen data (test data) is predicted by the model.

Checking Multicollinearity with VIF

⇒ 'x' is a dataframe, with the columns :- Size, Bedroom and age. These features are used to predict y = price of house
x = df.drop(column = 'House price') is actuall this only.

⇒ Now,

• import lib to calculate VIF

Import Statsmodel.api as sm
from Statsmodel.stats.outliners_influence import variance_inflation_factor

Splitting the data.

x_train, x_test, y_train, y_test = sk.train_test_split (x, y, test size = 0.2, random state = 42)

output of the fnx

It returns four values :-

x_train :- 80% of x chosen randomly to train

x_test :- 20% of remaing x is stored here.

(similarly with y) :

y_train = corresponding to randomly chosen 80% x (x_train) data the rows of y. [it is also 80% hence]

y_test = corresponding to randomly chosen 20% of x (x_test) data the rows of y. [it is also 20% hence].

fnx

⇒ train test_split a fnx of sklearn lib. used to split arrays/matrices into random train & test subsets.

⇒ x = dataset with house price, y = house price column data

⇒ test size = 0.2 ⇒ specifies that for testing 20% of data will be used (of both x & y) and for training 80% of x & y data used

⇒ random state = 42 ensures data is selected i.e. 80% & 20% is selected randomly not like first 20% then 80% it should be random.

Creating and Training the model

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

⇒ A linear regression model by name "model" is created and trained (i.e. in it fit data) on data x_train & y_train.

Making prediction

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

⇒ The model is used to predict the house price (y) based on the training set (x_test). The predictions are stored in y_pred.

model Ev

check

• import

calculate

• Creating a dataframe for VIF values

```
vif = pd.DataFrame()
vif['Features'] = X.columns
```

→ a new dataframe is created by the name vif.

→ In that dataframe we have a colown named Features which is same / taken from columns in X.

• Calculating VIF for Each Feature

```
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
```

⇒ This line calculate VIF for every feature one by one.

at the end, after running this code we get :-

Features	VIF
Size	4.5
Bedrooms	7.2
Bathrooms	3.8
Age	1.2

⇨ VIF = 1 means no correlation b/w Size and bedrooms

⇨ VIF greater than 5-10 indicates there is a problematic amount of multicollinearity.

Multicollinearity

↳ refers to a situation in Statistical model (usually in multiple regression) where two or more independent variable (x) are highly correlated with each other.

⇒ When multicollinearity is present, it can cause problem in regression analysis.

Eg: y = price of house, x = size, no. of bedrooms etc. Now, larger houses (size ↑) tend to have more bedrooms so, these two variables are highly correlated. This is the multicollinearity. If we include both in regression model, its difficult to determine their individual effect on house prices also, model may favour one or unfavour another as they are highly correlated. ⇒ This is a problem.

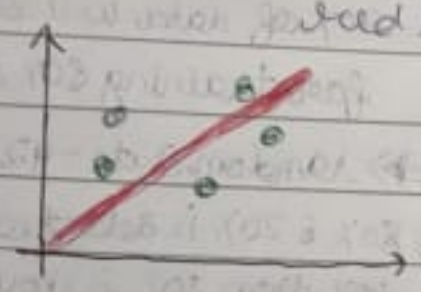
⇒ In order to detect this and its power, we use VIF (Variance inflation factor) for each

⇒ mse = mean_squared_error (y, y_predicted).
The line which the model draws actually calculate mse, b/w the actual y values and predicted one. It is a measure of how close a regression line is to a set of points.
The lower the MSE, the better the model performance

Plotting the result

for st. line ⇒
```
plt.scatter(x, y, colour='green')
plt.plot(x, y_predicted, colour='red')
plt.grid
plt.show
```

↓ red line b/w x any y_predicted.



Multiple (we have multiple independent variables ie. 'x').
↳ Here there are multiple x (independent variable) predicting a single y (=dependent variable). Eg. Predicting house price (y = house price) based on square area, no of bedrooms, age of house etc. (multiple x).

⇒ Consider, df = pd.read_csv('House.csv')
df

↓ Suppose we get :-

	House price	no. of bedroom	age of house	Sq. area
0	10,000	2	2	100
1	20000	4	4	121
2	30000	6	5	111

Importing the data

```
Import sklearn.model_selection as sk
x = df.drop(colown = 'House price')
y = df['House price']
```

→ this means x contain all colourms except House price

Calculate model parameters (intercept & coefficients)

```
model.intercept_
model.coef_[0]  or  model.coef_
```

(Indexing operator → Used to access the first element of the array. Coz, if we use array[:value] if we use [0] forward to it it will directly give [:value].

this it can give result

Predicting and Evaluating the data

```
y_predicted = model.predict(x)
r_square = r2_score(y, y_predicted)
mse = mean_squared_error(y, y_predicted)
```

⇒ model.predict(x) ⇒ uses the model to predict 'y' values for given 'x' values. This line basically say ⇒ Based on the rel^n you learned so far, tell me what 'y' would be for each 'x'. The result is stored in y_predicted.

⇒ r2_score ⇒ calculates coeff. of determination, k/a R^2. It gives you an idea of how well your independent variable (x, which you can change) explain the variance in your dependent variable (one which you are trying to predict).

$R^2 = 1$ → model perfectly fit. You could correctly predict y based on x.

$R^2 = 0.8$ → means 80% of the variance in y can be explained by x.

as R^2 ↓ so means your model is less & less accurate and at $R^2 = 0$ means model doesn't explain / predict variance in y.

requirement of such models).

-1 in reshape = tells Numpy to automatically calculate the no of rows based on the lenght of array & no of colown

1 ⇒ Specifies that there should be one colown (since, you have one feature or one independent variable (x))

```
X = np.array(adut).reshape(-1,1)
y = np.array(sale)
```

Creating and training the linear regression model

```
model = LinearRegression()
model.fit(X,y)
```

⇒ LinearRegression is the command that creats a linear regression model (which will predict value of a dependent variable (y), based on a independent variable(x)). It is stored in a variable named "model".

⇒ model.fit(X,y) → It is used to fit or train the model based on the data we provided.

⇒ X, y are the data we provided to the model. where, X (independent variable) —used→ to predict y (dependent variable).

⇒ we are tryying to predict "y" if we gave the model any 'x' is the purpose of this model.

⇒ when we run, Model.fit(X,y) we tell the model to look at the data 'x' & 'y' and learn the rel^n b/w them and try to figure out the changes in 'x' associated with changes in y.

⇒ After this the model finds the best fit line to represent rel^n b/w x & y.

All codes Unit-2

SIMPLE

Independent variable ⇒ I can change this variable, I have control over it.
dependent variable
↳ It is the output/response to the independent variable.

Eg. Sale = dependent variable, adult = independent var. (I can change its amount)
[y] [x]

```
Import numpy as np
Import pandas as pd
➡ Import matplotlib.pyplot as plt
★ ➡ From sklearn.linear_model import LinearRegression
★ ➡ From sklearn.metrics import r2_score, mean_squared_error
```

Data for regression

```
adult = [10,12,20,15,16,18,19,20]
Sale = [24,35,30,28,25,30,32,39]
```

⇒ out of this one is independent (I can change = adult) and one is dependent (= Sale).

Making the data compatible for linear regression.

↳ We convert the data into array since the models we use expect the data to be in array. As, it enable faster computation. Also it expect input (i.e. X) to be 2D array so thats why reshape (-1,1) for x variable. It transforms 1D array to 2D array (which is the

- To calculate stuff, on a particular column/row
  - ⇒ Stats.zscore(df['column name']) [in case of row use df.loc[0] ↑ index of row (1 minus row number)]
  - ⇒ df['column name'].mean()
  - df[' '].median()
    - .var()
    - .std()
    - .quartile(0.25) or .quartile([0.25, 0.75])
    - .skew/kurtosis()
- For graph of a particular column or row

  df[' '].hist() (you can include density, bins etc. too to the code.)

Shapiro Wilk Test for Normality

```
import scipy.stats as stats
exam_scores = [data.....];
```

→ shapiro_stat, shapiro_p_value = stats.shapiro(exam_scores)
print("shapiro-wilk test - statistic:", shapiro_stat)
print("shapiro wilk test - p-value:", shapiro_p_value)

⇓

formula for both shapiro-wilk test statistic and p-value is same.

We get the output as ⇒ output: shapiro wilk test statistic: —
p-value: —

Find correlation

→ df.corr()

→ It will give output like

	var1	var2	var3
var1	1	0.8	0.7
var2	-0.9	1	-0.2
var3	-0.2	0.14	1

→ observation (var1-1, 2-1, 3-3 for correlation 1 he rahega)

Correlation measures the strength and direction of association b/w two continous variables. → value of correlation (-1 to 1) → 1 = perfect +ve correlation, 0 = no correlation, -1 = perfect -ve correlation

You should have a df meaning either
df = pd.DataFrame(data) (you take)
or
df = pd.read_excel(" ")

- **No. of Rows & columns**

  df.shape

  ↓

  gives (Total no of rows, Total no. of columns)

- **Info on data types in the df**

  ↳ gives the type of data in every column

  df.datatypes

- **All columns names**

  ↳ gives the name of every column.

  df.columns

- **Dropping rows with missing value**

  df2 = df.dropna()

  drop/remove any row where there is a missing value

  df2

  ↳ on printing now we will get only that row which has all data points

Eg.

	Var 1	Var 2	Var 3
0	10	20	60.0
1	20	30	40.0
2	101	nan	69.1
3	100	20	70.0
4	400	10.1	nan

after dropna() and printing

2 & 4 are missing as they had missing values.

new Table

	Var 1	Var 2	Var 3
0	"	"	"
1	"	"	"
3	"	"	"

- **To access a particular row or column.**

  ★ iloc → In pands, stand for integer location. It gives us select rows and columns values. (Indexing in Python starts from '0' so, 0 means → 1st, 1 means → 2nd etc. In terms of Rows & Columns)

  df.iloc[0] = gives 1st row of df

  df.iloc[:,0] = gives 1st column of df

  df.iloc[0,0] = gives 1st element of 1st row & column

  df.iloc[:,[3,4]] = gives 4th and 5th column [Index 3 & 4 means 4th posn and 5th posn.]

→ NaN or nan = not a number

- To select a particular column or Row

Selected_column = df.column_name

OR

Selected_column = df.[["column1", "column2", "column3"]]

(for rows we use integer location iloc, alternatively iloc can also be used for columns.)

- To get a summary

Summary_Stats = df.describe()

↳ It will give out 8 things :-

- count (:- It means the total no. of entries without nan. in a column)
- mean
- std

And 5 point summary :- min, 25%, 50%, 75% and max

- Replacing missing values with True

↳ Tells us if any missing value

df.isnull() ⇒

	Var1	Var2	Var3
0	False	False	False
1	False	True	False
2	False	True	True

• True means there is a nan or missing value here in this point.

- Sum of the missing values

↳ Tell the total missing value in each column

df.isnull().sum()

like here,

Var1	0
Var2	1
Var3	2

output

Var3 has two datapoints at which there is no value.

★ Dataframes (df)

⇒ In the pandas library, we have a powerful tool for data analysis which provides data in structured form (like the one arranged in excel) k/a "Dataframe".

⇒ We always import excel in form of dataframes.

All commands of Dataframe ★

◉ Import Excel

```
import pandas as pd
df = pd.read_excel("path")
df
```

◉ Display rows

df.head() ⇒ gives first five rows of dataframe.

df.head(10) ⇒ gives 10 rows

df.tail(10) ⇒ gives the last 10 rows.

◉ Get info

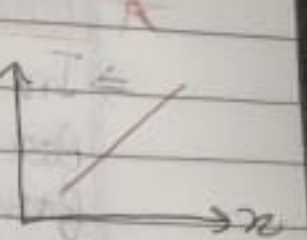
df.info ⇒ gives the info on non-null entries, Memory usage, Data type (like int = integers (whole no. (3,4 etc.)), float = real no. (2.34, 5.05 etc.))

String (Str) = text/characters, boolean (bool) = True or False data

etc.

For simple st. line graph

plt.plot(x, y, colour='red')

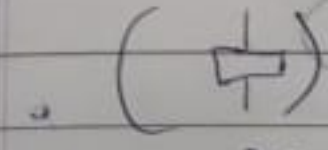


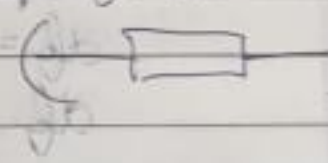
NOTE :- adding this code just after histogram code will give a frequency curve.

Boxplot

→ plt.boxplot(data1) OR plt.boxplot(data1, vert=False)

↓ For vertical boxplot ↓ For Horizontal boxplot





→ plt.title('....')

→ plt.show

In case we are drawing scatter plot of two different columns (independent and dependent)

then, plt.scatter(df['column name1'], df['column name2'])
plt.title('title')
plt.xlabel('column 1')
plt.ylabel('column 2')
plt.show.

q1 = np.percentile(data, 25)

q2 = np.percentile(data, 50)

q3 = np.percentile(data, 75)

- Graphs.

For Histogram

Import matplotlib.pyplot as plt

plt.hist(data1) OR plt.hist(data1, bins=10)

remains same

plt.xlabel('.....')
plt.ylabel('.....')
plt.title('.....')
plt.show

OR

plt.hist(data1, bins=10, density=True)

alpha = 0.5

For transparency of graph

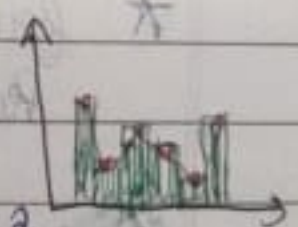
For frequency curve

↳ Frequency curve is drawn above histogram so, add the following line at the end of histogram code.

⇒ Histogram code ⇒ then

plt.plot(x, y, 'r-', linewidth=2)

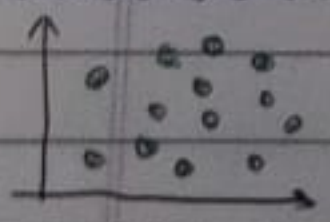
'r-' means red colour line of width = 2



Or

You can simply type "colour='red'"

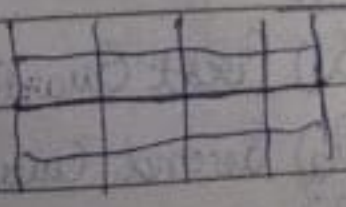
For Scatter plot



plt.scatter(x, y, colour='green')

green dots of (x,y) points will be shown

⭐ plt.grid ⇒ can be used to make grids on graphs. Just add this line at end or anywhere.



- **For Sorting data → Rising or ↓sing order**

Ascending / Rising ⇒ sorted(data)

Just directly these commands no '-' sign.

Reading / ↓sing ⇒ sorted(data, reverse=True)

- **For Range**

Range = max(data) - min(data) [compute value b/w min and max. value]

- **For Standard deviation, Variance, Skewness, Kurtosis, Z-score**

★ Std = np.std(data)
★ var = np.var(data)

Both, std and var are numpy lib

★ from scipy.stats import skew
skew_value = skew(data)

★ from scipy.stats import Kurtosis
Kurtosis_value = kurtosis(data)

We have to import these two from scipy.stats hence no np. ⇒ in there formula.

Import scipy.stats as st
Z_Score = st.zscore(data)

⇒ we import scipy lib so st.zscore(data), tells they are i.e. zscore is from scipy lib/module.

- **For Percentile**

⇒ we use 'numpy' lib to calculate percentiles.

NOTE

(Q_1) First Quartile corresponds to 25th percentile
(Q_2) Second Quartile " " 50th percentile aka Median
(Q_3) Third Quartile " " 75th percentile.

★ List of Import Statement

- from Statistics import mode
- from scipy.stats import skew
- from scipy.stats import kurtosis
- from sklearn.metrics import r2_score, mean_squared_error
- from statsmodel.stats.outliners_influence import variance_inflation_factor

★ Basic fnx codes

- For Mean, Median, Mode → define, data = [9, 10, 12, 13, 8, 7, 5, 4, 9, 10, 11]

① mean_value = np.mean(data)

To print just write ⇒ mean_value or print(mean_value). (np.mean is the main formula. The variable in which it is stored i.e. mean_value name can be changed to any.

② Median_value = np.median(data)

③ from statistics import mode

then, mode_value = mode(data) (* In formula of mode there is no np.) [As, it is not called from numpy library]