#EDA#

① Numerical Features
② Categorical Features
③ Missing Value
④ Outliers
⑤ Temporal Variable
⑥ cleaning

#Feature Engineering#

① Handling Missing Value
② Handling Imbalanced Dataset
③ Treating the Outliers
④ Scaling the data {Standardisation / down / Normalisation}
⑤ Converting the categorical feature into numerical features.

#Feature Selection

① Coorelation
② K Neighbour
③ Chisquare
④ Genetic Algorithm
⑧ Feature Importance {Extra Tree Classifier}

① Handling Missing value
Types of Encoding

① Handling Missing Value.

i) Numerical feature → If Outliers → then replace with Median.
→ NO Outliers → then replace with Mean.

ii) Categorical feature → Replace with Mode.

Numerical Feature.

① Drop the entire row which has missig value. {for huge data only 1 million}
② Create a seperate model to handle missing value. {Much More Time}
③ Statistical Method, Mean, Median, Mode. {Best}

Categorical Feature.

① Drop the entire row which has missing value {for huge data only 1 million}
② Replace with Mode.
③ Apply classifier algorithm to predict. {Most Efficient} {KNN}
④ Apply Unsupervised ML. {Clustering}

Type of Encoding :-

① Nominal Encoding {Order Not Important} → One Hot Encoding
→ One Hot Encoding with many categorical {KDD Orange}
→ Mean Encoding {Pincode}

Ex:- Gender → Male
→ Female

② Ordinal Encoding {Order Important} → Label Encoding
→ Target Guided Ordinal Encoding

Ex:- Education.

	Rank
Phd	4
Master	3
12th	1
B.com	2

Day 1:- Feature Engineering. {Missing Value}

Lifecycle of Data Science Projects.

1. Data Collection strategy - from company site, 3rd party API, Surveys.
2. Feature Engineering - Handling Missing Value.

Why are ther Missing Value? {Survey → Depression Survey}

1. They hesitate to put down the information.
2. Survey informations are not that valid.
3. Men → salary {don't say} salary.
4. Women → Age {don't say} age.
5. People may have died

Note:- Dataset should be collected from multiple sources.

What are the different types of Missing Data?

① [Missing Completely at Random (MCAR)] :- A variable is missing completely at random (MCAR) if the probability of being missing is the same for all the observations. When data is MCAR, there is absolutely no relationship between the missing data and any other value, observed or missing, within the dataset. In other words, those missing data points are a random subset of the data. There is nothing systematic going on that makes some data more likely to be missing than other.

[Inshort :- NO relationship between missing value with any features.]

Example :- In [titanic] dataset, the colum [Embarked] has a missing value which has no relationship between dependent and other feature.

source/destination/Pickup/drop

② [Missing Data Not at Random (MNAR)] :- Systematic Missing value ther is absolutely some relationship between the data missing and any other values, observed or ~~mission~~ missing, within the dataset.

[Inshort :- Relationship between missing data with other feature.]

Example :- In [titantic] dataset, the [Age] column and [Cabin] column has on missing value. which has relationship between each ~~of them~~ other. which means if person is died then ~~the~~ we cannot find its cabin and Age.

③ Missing at Random :- Men {Hide Salary} women {Hide Age}

IMP.
All the ~~Rep~~ technique of handling Missing value?
① Mean / Median / Mode ② Random Sample Imputation ③ Capturing NAN values with a new feature ④ End of Distribution Imputation ⑤ Arbitary Imputation ⑥ Frequent Category Imputation.

① Mean/Median/Mode :- Data should be MCAR If no outlier use mean or if outlier than use median for categorical vorible use mode.

Advantages :- ① Easy to implement (Robust to outlier).
② Faster way to obtain the complete dataset.

Disadvantage :- ① Change or Distortion in the original variance ↑
② Impacts Correlation.

Day 2 :-

② Random Sample Imputation :- Data should be MCAR. Random Sample imputation consists of taking random observation from the dataset and we use this observation to replace nan value.

Advantages :- ① Easy to implement
② There is less distortion in variance ↓

Disadvantages :- ① Every situation randomness wont work.

③ Capturing NAN value with a new feature :- It works well if the data are not missing completely at random MNAR.

Advantages :- ①② Capture the Importance of missing values.

Disadvantages :- ① Creating Addition feature (Curse of Dimensionality)

Eg :-

Age	Survided
1	0
2	1
3	0
NaN	1
4	0

Age_new
0
0
0
1
0

Age	Survided	Age_new
1	0	0
2	1	0
3	0	0
2	1	1
4	0	0

Replace with Mean/Median

④ End of Distribution Imputation :- In this technique we will replace NaN value with far end of the distribution. [df.Age.mean() + 3 * df.Age.std()]

[df.Age.mean() + df.Age.std() * 3] ← same.

Eg. Age Survived

Age	Survived
22	0
38	1
26	0
Nan	1
30	0

Age	Survived
22	0
38	1
26	0
60	1
30	0

far end distribution

Eg:- Age Survided

Age	Survived
22	0
38	1
26	0
Nan	1
30	0

Age	Survived	Age_End_Distribution
22	0	22
38	1	38
26	0	26
23	1	60 → far end distribution
30	0	30

23 → median

Advantages :- ① Easy to implement.

② Captures the importance of missingess if there is one.

Disadvantage :- ① Distorts the original distribution of the variable.

② If missingess is not important, it may mask the predictive power of the original variable by distorting its distribution

③ If the number of NA is big, it will mask true outliers in the distribution.

④ If the number of NA is small, the replaced NA may be considered an outlier and pre-processed in a subsequent step of feature Engineering.

⑤ Arbitrary Value Imputation :- This technique was derived from kaggle competition it consists of replacing NaN with arbitrary value.

based on random choice or personal whim, rather than any reason or system. In these technique we will replace nan value with minimum or maximum value i.e. outlier value.

Eg:- Age survied

Age	survied
10	0
11	1
2 → min	0
13	1
15	0
100 → max	1

Nan → 11, Nan → 15

Age	survied
10	0
11	1
Nan	0
2	1
13	0
Nan	1
16	0
100	1

Age_min	Age_max
10	10
11	11
2	100
2	2
13	13
2	100
16	16
100	100

Day 3

Adva

Disadv

② If m

variab

③ Hard

Ham

① Fre

elem

Advant

Disadv

use

② I

②Capt

Note

Ha

②

③ M

①

②

No

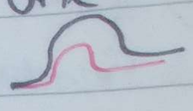
① On

② or

③ M

Day 3

Advantages :- ①② Capture the importance of missingness if there is one.

Disadvantages :- ① Distorts the original distribution of the variable  .
② If missingness is not important, it may mask the predictive power of the original variable by distorting its distribution.
③ Hard to decide which value to use.

Handling Missing Value in Categorical features :-

① Frequent Category Imputation :- Replacing Nan value with most frequent element i.e. Mode {for data missing 10-15%}

Advantages :- ①② Faster way to implement

Disadvantages :- ① Since we are using the more frequent labels, it may use them in an over represented way, if there are many nans.
② It distorts the relation of the most frequent label.

② Capturing Nan value with a new feature :- {Refer page no. 4} {1,0}

Note :- Suppose if we have more frequent categories, we will replace Nan with a new category i.e. {Missing} {for data missing more than 50%}

Handling Categorical variable :-

① One Hot Encoding
② One Hot Encoding with many category {KDD Org Orange Cup}
③ Mean Encoding {Pincode}
① Label Encoding
② Target Guided Ordinal Encoding

{order not imp → Nominal Encoding}
{Ordinal Encoding order Imp}

Nominal Encoding {Order Not Important}
① One Hot Encoding
② One Hot Encoding with many category (KDD)
③ Mean Encoding {Pincode}

P.T.O

Ordinal Encoding {Order Imp}
① Label Encoding
② Target Guided Ordinal Encoding

① One Hot encoding {Nominal} Ex. Gender → Male, Female

Gender	Gender_Male	Gender_Female
Male	1	0
Female	0	1
Female	0	1
Male.	1	0

pd.get_dummies(dataset).head()

drop_first = True.

② One Hot Encoding with many category in a feature {Nominal} {Mercedes.csv}

→ In this type of encoding we take first 10 row or more frequent repting element from each categorical column. {KDD Orange}

→ In mercedes dataset, we have column like $[X_0, X_1, X_2, X_3, X_4, X_5, X_6]$ with many category value. {Highly Cardinality}

→ In this only top 10 will replace with One Hot Encoding and rest of the value will become 0 which are not more frequent.

③ Ordinal Number Encoding:- {Ordinal} In this type of encoding, we assing rank to our data based upon priority.

Ex.

Education	Rank
12th	4
Graduation	3
PHD	2
Master	1

Day	Rank
Monday	1
Tuesday	2
Wednesday	3
Thursday	4
Friday	5
Saturday	6
Sunday	7

{Nominal}

④ Count or Freaueney Encoding :- In these encoding, we will replace the vaeue with its count.

Eg.

Country
India
USA
US
India
India
Cuba

Country
3
1
1
3
3
1

Advantages :- ① ② Not increase feature space.

Disadvantages :- ① It will provide same weight if the freavence are same.

i.e. If India has count = 3 and Cubas has count 3 then the data become inappropiate.

⑤ Target Guided {Ordinal Encoding} :- ① Ordering the labels according to the target ② Replace the labels by the joint probability of being 1 or 0.

→ In titiratic dataset, cabin column had vaue as E85, D 25, A 22, B33 and Nan.

→ we will replace Nan, with "Missing".

→ Then we will ~~with~~ take first letter of every column with [E, D, A, B, M] ↗ Missing .mean

→ Then sort the vaue [A, E, M, D, B] then assign rank to [A:1, E:2 ....]

→ Then replace in dataset [0, 1, 2, 3, 4, 5] ↑

⑥ Mean Encoding {Nominal} :- we will replace with mean.

```
m = dataset.groupby(['cabin'])['survived'].mean().to_dict()
print(m)
```

{ 'A' : 0.46, 'B' : 0.74, 'C' : 0.59, 'D' : 0.75 }

Now, we will replace in our dataset.

```
df['mean_ordinal_encode'] = df['cabin'].map(m)
```

survived	cabin	mean_ordinal_encode
0	M	0.09
1	C	0.59
1	M	0.69
0	C	0.59
0	M	0.69
1	A	0.46
0	B	0.74

① capturing information within the label therefore rendering more predictive feature

② Monotonic relationship between variable and target.

① Prone to overfitting
↗ lead to

⑦ Probability Ratio Encoding :- In titanic dataset, we will use column as "cabin" & "survived".

Survived	Cabin
0	NaN
1	C85
1	NaN
1	C123
0	NaN

we will replace Nan value with "Missing".

~~df[~~ {df['Cabin'].unique()}
['Missing', 'C85', 'C123', 'C4', 'A16', 'A27' ....]

```
df['Cabin'].astype(str).str[0] = df['Cabin']
df.head()
```

Survived	Cabin
0	M
1	C
1	M
1	C
0	M

```
df.Cabin.unique()
['M', 'C', 'E', 'G', 'D', 'A', 'B', 'F', 'T']

pob=df.groupby(['Cabin'])['Survived'].mean()
```

~~A 0.04~~ A 0.466
B 0.744
C 0.75
⋮ ⋮
T 0.00.00.

```
pob=
pd.Dataframe(pob)
```

Cabin	Survived
A	0.466
B	0.744
C	0.75
⋮	⋮
T	0.00

```
pob['Pob_ratio'] = prob['Survived']/pob['Died']
pob.head()
```

Cabin	Survived	Died	Probability_ratio
A	0.46	0.53	0.875
B	0.744	0.25	2.91
C	0.59	0.40	1.45
⋮	⋮	⋮	⋮

```
pob['Pob_ratio'].to_dict() = encoded.
```

{'A':0.875, 'B':2.91, 'C':1.45 ...}

```
pob['Died'] = 1-pob['Survived']
```

Cabin	Survived	Died
A	0.46	0.53
B	0.744	0.25
C	0.59	0.40
⋮	⋮	⋮
T	0	1

```
df['Cabin_encode'] = df['Cabin'].map(encoded)
df.head()
```

Survived	Cabin	Cabin-encoded
0	M	0.48
1	C	1.45
1	M	0.48
1	C	1.45
0	M	⋮
⋮	⋮	⋮

Note :-
① Probabi
② Probabi
③ Pr(Survi
④ Dictiona
⑤ Replace

Transfo
Why trans
To scale
like Lin
Deep Le
Note :-
algo

Typ
① Nor
③ Sca
Transf
① Sta
scale.
df =

Note:-
① Probability of survived based on cabin → categorical feature
② Probability of Not survived → 1 - pr(survived)
③ pr(survived) / pr(Not survived)
④ Dictionary to map cabin with probability
⑤ Replace with the categorical feature.

Transformation of Features.

Why transformation of features are required? ~~To scale in a similar unit~~
To scale down the data and make calculation fast. Machine Learning algo^th like Linear Regression, Logistic Regression, KNN, Kmeans, Hierarchical Clustering and Deep Learning Techniques like ANN, CNN and RNN. we required feature transformation
Note:- we don't apply transformation to all ML algo only above ~~mention~~ mention algo ~~use~~ require transformation. whereas in DL we require transformation.

Types of Transformation:-

① Normalization & standardization ② Scaling to Min & Max value.
③ Scaling To Median & Quantiles ④ Gaussian Transformation { Logarithmic Transformation }{ Reciprocal }{ Square Root }{ Exponential (exponent) }{ Boxcox }.

① Standardization :- we try to bring all the variables or features to a similar scale. Standardization means centering the variable at zero. $z = \frac{x - \mu}{\sigma}$

df =

Survived	Pclass	Age	Fare
0	3	22	7.25
1	1	38	71.28
1	3	26	7.92
1	1	35	53.10
0	3	35	8.05

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
[-0.789, 0.827, .... ]
plt.hist(df_scaled['Survived'], bins=20)...
```

② Normalization :- {Mostly used in CNN DL}

→ Min Max scaling scales the value between 0 to 1.

$$X\text{-scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

```
from sklearn.preprocessing import MinMaxScaler
min_max = MinMaxScaler()
df_minmax = pd.DataFrame(min_max.fit_transform(df), columns=df.columns)
```

Survived	Pclass	Age	Fare
0.0	1.0	0.27	0.014
1.0	0.0	0.47	0.139
1.0	1.0	0.32	0.015
1.0	0.0	0.43	0.103
0.0	1.0	0.43	0.015

```
plt.hist(df_minmax['Pclass'], bins=20)
```

③ Robust Scaler :- It is used to scale the feature to median & quantiles

Scaling using median and quantiles consists of substracting the median to all the observation, and then dividing by the interquantile difference. The interquantile difference is the difference between the 75% & 25% quantile:

$$IQR = 75^{th}\text{ quantile} - 25^{th}\text{ quantile}$$

$$X\_scaled = (x - x.median)/IQR$$

```
from sklearn.preprocessing import RobustScaler
scaler = RobostScaler()
df_robust_scaler = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
df_robust_scaler.head()
```

Survived	Pclass	Age	Fare
0.0	0.0	-0.46	-0.31
1.0	-2.0	0.789	2.46
1.0	0.0	-0.15	-0.28
1.0	-2.0	0.53	1.67

```
plt.hist(df_robust_scaler['Age'], bins=20)
```

④ Gaussian

```
df = pd.
df['Age']
import
import
```

Note :- D
distribut

```
def plot
```

Logarith
import
df['
plot

Reciproc
df['
plot_

Square
df['
plot_

4) Gaussian Transformation :-

```
df=pd.read_csv('titanic.csv', usecols=['Age', 'Fare', 'Survived'])
df['Age']=df['Age'].fillna(df['Age'].median())

import scipy.stats as stat
import pylab
```

Note:- If we want to check whether feature is gaussian or normal distribution use Q-Q plot.

```
def plot_data(df, feature):
    plt.figure(figsize=(10,6))
    plt.subplot(1,2,1)
    df[feature].hist()
    plt.subplot(1,2,2)
    stat.probplot(df[feature], dist='norm', plot=pylab)
    plt.show()
```

Logarithmic Transformation

```
import numpy as np
df['Age_log']=np.log(df['Age'])
plot_data(df,'Age_log')
```

Reciprocal Transformation

```
df['Age_Reciprocal']=1/df.Age
plot_data(df,'Age_reciprol')
```

Square root Transformation

```
df['Age_square']=df.Age**(1/2)
plot_data(df,'Age_square')
```

Exponential Transformation

```
df['Age_Exprential']=df.Age**(1/1.2)
plot_data(df,'Age_exponial')
```

Box cox Transformation ↪ P70

```
stat.boxcox(df['Age'])
```

Box Cox Transformation :- The Box cox transformation is defined as:

$$T(Y) = (Y^{\exp(\lambda)} - 1) / \lambda$$

where Y is the response variable and λ is the transformation parameter. λ varies from -5 to 5. In the transformation, all values of λ are considered and the optimal value for a given variable is selected.

```
df['Age_Boxcox'], parameters = stat.boxcox(df['Age'])
print(parameters)
  0.79
plot_data(df, 'Age_Boxcox')
```

Handling

{pip install
we will use
output fea

```
X = df.dr
y = df.
from sklea
from sklea
from sklea
import ne
from sk
log-clas
grid =
CV = KF

from s
X-trai

clf = Gr
clf. fi

y-pre
print
print

[[8
[
0.
```

Igr

defined as:

...ion parameter.
...ues of λ
...ble is selected.

Handling Imbalanced Dataset

{pip install imbalanced-learn}

we will use CreditCard Fraud Detection which has no missing value and output feature is "Highly Imbalanced". Class 0 - 284315 1 - 492 {for Imbalance dataset it is not worth}

```
X = df.drop("Class", axis=1)
y = df.Class
```

{we will apply LogisticRegression to check what happen if our data is Highly Imbalance.}

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import KFold
import numpy as np
from sklearn.model_selection import GridSearchCV
```

DO not check accuracy if data is Highly Imbalanced.

```
log_class = LogisticRegression()
grid = {'C': 10.0**np.arange(-2,3), 'penalty': ['l1', 'l2']}
cv = KFold(n_splits=5, random_state=None, shuffle=False)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7)

clf = GridSearchCV(log_class, grid, cv=cv, n_jobs=-1, scoring='f1_macro')
clf.fit(X_train, y_train)
```

n_jobs=-1 use core of computer

```
y_pred = clf.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[85251   47]
 [   41  104]]
0.99
```

Ignore Bad model

	precision	recall	f1 score	support
	⋮	⋮	⋮	⋮

Ensemble technique are work well for Imbalance Dataset.

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(class_weight = class_weight)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Under Sampling → Best for small dataset worst for Big Dataset.

```
from collections import Counter
Counter(y_train)

from collections import Counter
from imblearn.under_sampling import NearMiss
ns = NearMiss(0.8)
X_train_ns, y_train_ns = ns.fit_sample(X_train, y_train)
print("The number of classes before fit {}".format(Counter(y_train)))
" " " " " " " " " " " " .format(Counter(y_train_ns)))
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier()
classifier.fit(X_train_ns, y_train_ns)

y_pred = classifier.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

→ 4 line above →

Note:- Interview how do you handle Imbalanced dataset? → If the dataset is small I will defenitly go with under sampling but i will be focusing on all the performance matrix recall precision f1 score apart from that i will also be focusing on based upon domain knowledge whether I should reduce with reducing FP with FN based i will be selecting ROC score that is the probability value 0.6 0.7 something like that finally

i will also be doing
be always involving
working i will go wit

Over sampling

```
from imblearn.over
os = RandomOverS
X_train_ns, y_
print(" "
print(" "
from sklearn.ense
classifier = Rand
classifier.fit(
```

→ 4 line

SMOTETomek

```
from imblearn.c
os = SMOTETom
X_train_ns, y_
print(" "
print(" "
from sklearn.e
classifier = Ran
classifier.fit(
```

→ 4 line

Ensemble T

```
from imblear
easy = EasyE
easy.(X_t
easy.
y_pred = eas
```

← 3 line

i will also be doing SMOTE technique i will be Oversampling technique i will be always moving around understanding performancing matrix finally it is not working i will go with DT, RandomForest, XGBOOT where I can also provide class weight.

Oversampling

```
from imblearn.over_sampling import RandomOverSampler
os = RandomOverSampler (0.75)
X_train_ns, y_train_ns = os.fit_sample (X_train, y_train)
print ( -1, - 11 - 11
print ( -11 - 11 - 11
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier()
classifier.fit (X_train_ns, y_train_ns)
```

→ 4 ure

SMOTETomek

```
from imblearn.combine import SMOTETomek
os = SMOTETomek (0.75)
X_train_ns, y_train_ns = os.fit_sample (X_train, y_train)
print(-11-11)
print(-11-11)
from sklearn.ensemble import RandomForestClassifier.
classifier = RandomForestClassifier()
classifier.fit ( X_train_ns, y_train_ns)
```

→ 4 ure

Ensemble technique

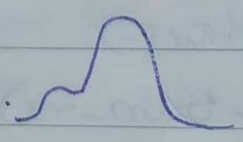
```
from imblearn. ensemble import EasyEnsembleClassifier
easy = EasyEnsembleClassifier()
easy. ( X_train, y_train)
easy.
y_prcd = easy. predict (X_test)
```

→ 3 ure

Outlier Handling:-

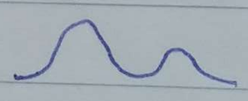
1. Naive Bayes Classifier - Not sensitive to outliers
2. SVM - Not sensitive to outliers
3. Linear Regression - Sensitive to outliers
4. Logistic Regression - Sensitive to outliers
5. Decision Tree Regressor or classifier - Not Sensitive to outliers.
6. Ensemble (RF, XGBoost, GB) - Not sensitive to outliers.
7. KNN - Not sensitive to outliers.
8. Kmeans - Sensitive to outliers.
9. Hierarichal - Sensitive to outliers.
10. PCA - Sensitive to outliers.
11. Neural Network - sensitive to outliers.
12. DBScan - Sensitive to outlier.

```
import pandas as pd
df = pd.read_csv('titanic.csv')
df.head()
df['Age'].isnull().sum()   OP - 177
import seaborn as sns
sns.distplot(df['Age'].dropna())
```



Now, we will add outlier in age column for demonstration.

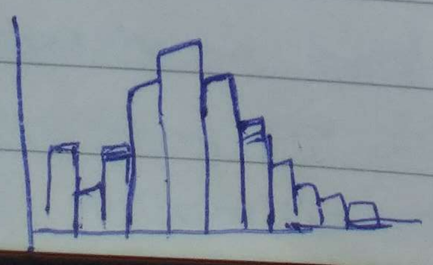
```
sns.distplot(df['Age'].fillna(100))
```



NOTE:- If the data is normally distributed we will use IQR to remove outlier.

Gaussian Distributed

```
figure = df.Age.hist(bins=50)
figure.set_title('Age')
figure.set_xlabel('Age')
figure.set_ylabel('NO of Passengers')
```



```
figure = df.b
df['Age'].de
```

Assuming the
the bounda

```
upper_bound
lower_boun
print(upper_
-13.88 , 7
```

```
IQR = df.
17.875
```

```
lower_tence
higher_fen
```

```
lower_bri
higher_fenc
```

Skewed

```
figure=d
figure.set
figure.set
figure.set
df.boxplo
df['Fare
```

figure = df. boxplot (column = 'Age')
df['Age'].describe ()

Assuming the age follows a Gaussian Distribution we will calculate the boundaries which differentiates the outliers.

upper-boundary = df['Age'].mean + 3 * df['Age'].std()
lower-boundary = df['Age'].mean - 3 * df['Age'].std()
print(upper-boundary), print(lower-boundary), print(df['Age'].mean())
-13.88 , 73.27, 29.69

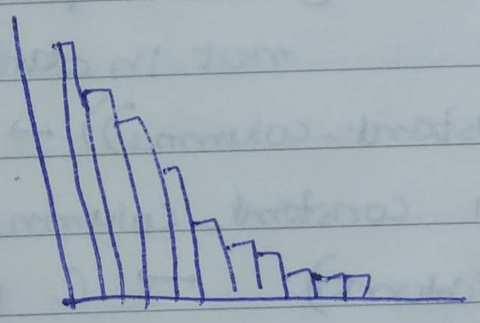
IQR = df.Age.quantile(0.75) - df.Age.quantile(0.25)
17.875

lower-tence = df['Age'].quantile(0.25) - (IQR * 1.5) {-6.68}
higher-fence = df['Age'].quantile(0.75) + (IQR * 1.5) {64.81}

lower-bifence = df['Age'].quantile(0.25) - (IQR * 3) {-33.5}
higher-fence = df['Age'].quantile(0.75) + (IQR * 3) {91.6}
{For Extreme Outlier.}

Skewed Feature.

figure = df.Fare.hist(bins = 50)
figure.set_title('Fare')
figure.set_xlabel('Fare')
figure.set_ylabel('NO of passenger')
df.boxplot(column="Fare")
df['Fare'].describe()



Feature selection

Feature selection - {Dropping constant feature} :- In this step we will be removing the features which have constant features which are acutally not important for solving the problem statements.

```
import pandas as pd
data = pd.DataFrame({"A":[1,2,4,1,2,4], "B":[4,5,6,7,8,9],
                     "C":[0,0,0,0,0,0], "D":[1,1,1,1,1,1]})
```

① Variance Threshold :- Feature selector that removes all low-variance feature. This feature selection algorithm looks only at the feature(X), not the desired outputs(y), and can thus be used for unsupervised learning.

```
# It will remove zero variance feature.
from sklearn.feature_selection import VarianceThreshold
var_thres = VarianceThreshold(threshold=0)
var_thres.fit(data) → VarianceThreshold(threshold=0)

var_thres.get_support() → array([True, True, False, False])
data.columns[var_thres.get_support()] → Index(['A', 'B'], dtype='object')

constant_columns = [column for column in data.columns if column
                    not in data.columns[var_thres.get_support()]]
print(len(constant_columns)) → 2
for feature in constant_columns:
    print(feature) → C D

data.drop(constant_columns, axis=1)
```

A	B
1	4
2	5
4	6
1	7
2	8
4	9

② Feature selec

removing the
we will take b
test split.

```
X_train.corr()
import seaborn as
plt.figure(figsi
cor = X_train.c
sns.heatmap(c
plt.show()
```

Now, we will re

Note:- If inde
do not drop
with other indep

```
def correlation
    col_corr =
    corr_matr
    for i in
        for j
        if
    return
corr_features
len(set(co
corr_featur
X_train.dr
X_test.dro
```

② Feature selection - with Correlation :- In this step we will be removing the features which are highly correlated.

we will take boston dataset and clipt feature in x & y then train test clipt.

```
x_train.corr()
import seaborn as sns
plt.figure(figsize=(12,10))
cor = x_train.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.CMRmap_r)
plt.show()
```

Now, we will remove feature which are highly correlated.

Note:- If independent feature is highly corelated with dependend feature do not drop that column. If one independent feature is highly corelated with other independent feature then we any one column.

```
def correlation(dataset, threshold):
    col_corr = set()
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i,j])> threshold:
                colname = corr_matrix.columns[i]
                col_corr.add(colname)
    return col_corr

corr_features = correlation(X_train, 0.7)
len(set(corr_feature))

corr_features

X_train.drop(corr_feature, axis=1)
X_test.drop(corr_feature, axis=1)
```

(3) Feature selection - Information Gain - Mutual Information in Classification Problem Statements.

Mutual Information estimate mutual information for a discrete target variable. Mutual Information between two random variable is a non-negative value, which measure the dependency between the variable. It is equal to zero if and only if two random variables are independant, and higher value mean higher dependency. { use wine data, X & y, train test split.

```
from sklearn.feature_selection import mutual_info_classif {0 to 1}
mutul-info= mutual_info_classif (X_train, y_train)
mutul-info
```

```
from sklearn.feature_selection import SelectKBest
sel_five_cols = SelectKBest(mutual_info_classif, K=5)
sel_five_cols.fit (X_train, y_train)
X_train.columns [sel_five_cols.get_support()]
```

Difference Between Information Gain and Mutual Information

$I(X;Y) = H(X) - H(X|Y)$ and $IG(S,a) = H(S) - H(S|a)$

As such, mutual information is sometimes used as a synonym for information gain. Technically, they calculate the same quantity if applied to the same data. →

(4) Feature selection

Problem statement

Mutual Information

we will use

from sklearn.feat

mutual_info = m

mutual_info

mutual_info =

mutual_info *

mutual_info.so

from sklearn.fea

selected_top_c

selected_top_c

selected_top_

X_train.colu

(5) Chi square

Compare c

This score

→ This score

values for

non-negati

document

→ Recall th

variables,

to be in

Square s

variable

mation in

a discrete target
able is a non-negative
k. It is equal to zero
and higher value
put.

sif {0 to 1}

rmation

ym for
uantity if } →

④ Feature selection - Information Gain - mutual information in Regression
Problem Statements.

Mutual Information estimate mutual information for a continuous target variable.
we will we having dataset in that only numerical feature x & y, train test.

```
from sklearn.feature_selection import mutual_info_regression
mutual_info = mutual_info_regression(X_train.fillna(0), y_train)
mutual_info
mutual_info = pd.Series(mutual_info)
mutual_info.index = X_train.columns
mutual_info.sort_values(ascending=False)
```

```
from sklearn.feature_selection import SelectPercentile
selected_top_columns = SelectPercentile(mutual_info_regression, percentile=20)
selected_top_columns.fit(X_train.fillna(0), y_train)
```

```
selected_top_columns.get_support()
```

```
X_train.columns[selected_top_columns.get_support()]
```

⑤ Chi square Test for feature selection.

Compare chi-squared stats between each non-negative feature & class.
This score can be used to evaluate categorical variable in a classification test.
→ This score can be used to select the n_feature features with the highest values for the test chi-squared statistic from X, which must contain only non-negative features such as booleans or frequencies (e.g. terms counts in document classification), relative to the class.
→ Recall that the chi-squared test measure dependence between stochastic variables, so using this function "weed out" the feature that are the most likely to be independent of class and therefore irrelevant for classification. The Chi Square statistic is commonly used for testing relationship between categorical variable.

→ It compares the observed distribution classes of target Y among the different category of the feature, against the expected distribution of the target class, regardless of the feature categories.
→ We will use titanic dataset categorical feature with feature engineering of categorical data. X & Y, test train split.

```
from sklearn.feature_selection import chi2
f_p_value = chi2(X_train, y_train)

f_p_values ([... ]), ([.... ])
import pandas as pd
p_values = pd.Series(f_p_value[1])
p_values.index = X_train.columns
p_values
p_values.sort_index(ascending = False)
```

Observation

Sex column is the most important column when compared to the output feature Survived.