**df=dataframe name**

**1:Reading csv files**

ibm(dataframe name)=pd.read\_csv('filename.csv',na\_values=[ "?", " ", "null"])

**1a. Display specific columns**

import pandas as pd

df=pd.read\_csv('filename.csv',usecols=['col\_7','col\_8'])

df

**2.Dataframe head**

df.head()

**3.Dataframe tail**

df.tail()

**4.Dataframe summary**

df.describe()--Numeric

df.describe(include=”object”)—Num and Categoric

df.describe(include="all")

**5. Explore the data types of each column**

df.dtypes

**6.Check for unique values of each column**

df.nunique()

**7.Drop columns which are not significant**(hereUnnamed: 32,id are column names)

df.drop(['Unnamed: 32','id'],axis=1,inplace=True)

**8:Check for null values**

df.isna().sum()

print("No. of Null values in the train set :", df.isnull().sum().sum())-print null values

df.isnull()—True or False to check null values

df[df[‘col name’].isnull()]----dataframe with null values

**9:Remove null values** (here titanic=df)

titanic['Age']=titanic['Age'].fillna(titanic['Age'].median())---mean or median

titanic['Cabin'].fillna(titanic['Cabin'].mode()[0], inplace=True)-----mode

**10:Replace missing values using ffill,bfill**

(spx,dax=column name)

df.spx=df.spx.fillna(method= "ffill")

df.spx=df.spx.fillna(method= "bfill")

df.dax=df.dax.fillna(value=df.dax.mean())

**11:Converting to another data type**

df[cat\_cols] = df[cat\_cols].astype('int')------convert to int

**11a. Seperating categorical and numeric attributes**

cat\_attr = list(data.select\_dtypes(include=['category']).columns)

cat\_attr.pop()

num\_attr = list(data.columns.difference(cat\_attr))

num\_attr.pop()

or(here 5=n)

cat\_cols = data.columns[data.nunique() < 5]

num\_cols = data.columns[data.nunique() >= 5]

**12a:Convert all the categorical columns to Integer Format before dummification**

df['Gender'] = df['Gender'].map({'Female':0,'Male':1})

df['Married'] = df['Married'].map({'No':0, 'Yes':1}).astype(np.int)

df[cat\_cols] =df[cat\_cols].astype('int')

**12b.Dummification**

df = pd.get\_dummies(df, columns=cat\_cols, drop\_first=True)—better to use this\*

df = pd.get\_dummies(df, columns=cat\_cols, drop\_first=False)

or

df = pd.get\_dummies(columns=cat\_attr,data=df,drop\_first=True)

or

X\_encoded=pd.get\_dummies(df,columns=['sex', 'cp','fbs','restecg','exang','slope','ca','thal'])

X\_encoded.head()

Or

sex = pd.get\_dummies(train['Sex'],drop\_first=True)

embark = pd.get\_dummies(train['Embarked'],drop\_first=True)

df = pd.concat([df,sex,embark],axis=1)

**12c.Imputation**

mean\_imputer = Imputer(strategy='mean')

imputed\_df = pd.DataFrame(mean\_imputer.fit\_transform(df),columns=df.columns)

**12d. Select only the categorical variables**

object\_attrs = list(df.select\_dtypes("object").columns)

object\_attrs

**12e.Type-casting variables to correct data-types**

for attr in object\_attrs:

df[attr] = df[attr].astype("category")

**12f. Exclude target column from list of categorical columns before using it for dummification of independent categorical variables**

object\_attrs.remove("target column")

cat\_attrs = object\_attrs

**13:Remove warnings**

import warnings

warnings.filterwarnings('always')

warnings.filterwarnings('ignore')

**14: Dummify the Categorical columns**(can be done aftertrain test split)

df= pd.get\_dummies(df, columns=cat\_cols, drop\_first=True)

**# Train**

X\_train = pd.get\_dummies(X\_train, columns=cat\_cols, drop\_first=True)

**# Test**

X\_test = pd.get\_dummies(X\_test, columns=cat\_cols, drop\_first=True)

**14a.Scale the numeric attributes ["age", "bili", "alk", "sgot", "albu", "protime"]**

#num\_cols = ["age", "bili", "alk", "sgot", "albu", "protime"]

scaler = StandardScaler()

scaler.fit(X\_train.loc[:,num\_cols])

# scale on train

X\_train.loc[:,num\_cols] = scaler.transform(X\_train.loc[:,num\_cols])

#X\_train[num\_cols] = scaler.transform(X\_train[num\_cols])

# scale on test

X\_test.loc[:,num\_cols] = scaler.transform(X\_test.loc[:,num\_cols])

**15.Check for value counts**

df[‘col name’].value\_counts()

**16.Categorical to numerical** (here vhigh,vhigh.1 are columns)

df['vhigh'],\_ = pd.factorize(df['vhigh'])

df['vhigh.1'],\_ = pd.factorize(df['vhigh.1'])

or

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

df['col name']=le.fit\_transform(df['col name'])

**17.Define X and y** (here class is target variable)

X= df.loc[:,df.columns!='class']

X

----------------------------------------

y=df.loc[:, "class"]

y

or

X=df.drop('target column', axis=1)

y=df["target column"]

or

**Get the Independent variables and dependent variable from data**

X, y = df.drop("target column", axis=1), df.target column

**17b.Check for X and y**

type(X)

**expected**:pandas.core.frame.DataFrame

type(y)

**expected**:pandas.core.series.Series

**18.Converting target variable yes or no to 1 and 0** (here diagnosis(M&B) is target variable)

df.diagnosis=[1 if each=="M" else 0 for each in df.diagnosis]

**18a.Students grade target variable**

def Grade(marks):

if marks >= 90:

grade = 'A'

elif marks >= 80:

grade = 'B'

elif marks >= 70:

grade = 'C'

elif marks >= 60:

grade = 'D'

else:

grade = 'F'

return grade

student["Grade\_math"] = student["math score"].apply(Grade)

student["Grade\_reading"] = student["reading score"].apply(Grade)

student["Grade\_writing"] = student["writing score"].apply(Grade)

student["Overall\_grade"] = student["Percentage"].apply(Grade)

student.head()

**19.Train test split**

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 122)

**Split the attributes into numerical and categorical types**

num\_attr=X\_train.select\_dtypes(['int64','float64']).columns

num\_attr

cat\_attr = X\_train.select\_dtypes('category').columns

cat\_attr

**20.Handling Numerical Attributes - Preprocessing**

**Imputation (Filling missing values)**

from sklearn.impute import SimpleImputer

num\_imputer = SimpleImputer(strategy='mean')

num\_imputer = num\_imputer.fit(X\_train[num\_attr])

X\_train\_num = num\_imputer.transform(X\_train[num\_attr])

X\_train\_num = pd.DataFrame(X\_train\_num, columns=num\_attr)

X\_test\_num = num\_imputer.transform(X\_test[num\_attr])

X\_test\_num = pd.DataFrame(X\_test\_num, columns=num\_attr)

**Handling Categorical Attributes - Preprocessing**

**Imputation (Filling missing values)**

cat\_imputer = SimpleImputer(strategy='most\_frequent')

cat\_imputer = cat\_imputer.fit(X\_train[cat\_attr])

X\_train\_cat = cat\_imputer.transform(X\_train[cat\_attr])

X\_train\_cat = pd.DataFrame(X\_train\_cat, columns= cat\_attr)

X\_test\_cat = cat\_imputer.transform(X\_test[cat\_attr])

X\_test\_cat = pd.DataFrame(X\_test\_cat, columns= cat\_attr)

**21.Encoding Categorical Attributes to Numeric – OneHotEncoding**

onehotencoder = OneHotEncoder(handle\_unknown='ignore')

onehotencoder = onehotencoder.fit(X\_train\_cat)

ohe\_cat\_col\_names = onehotencoder.get\_feature\_names(cat\_attr)

ohe\_cat\_col\_names

X\_train\_cat\_onehotencoded = onehotencoder.transform(X\_train\_cat).toarray()

X\_train\_cat\_onehotencoded = pd.DataFrame(X\_train\_cat\_onehotencoded, columns=ohe\_cat\_col\_names)

X\_train\_cat\_onehotencoded.head()

X\_test\_cat\_onehotencoded = onehotencoder.transform(X\_test\_cat).toarray()

X\_test\_cat\_onehotencoded = pd.DataFrame(X\_test\_cat\_onehotencoded, columns=ohe\_cat\_col\_names)

X\_test\_cat\_onehotencoded.head()

**Merging Numerical and Categorical Attributes**

X\_train = pd.concat([X\_train\_num, X\_train\_cat\_onehotencoded], axis=1)

X\_train.head()

X\_test = pd.concat([X\_test\_num, X\_test\_cat\_onehotencoded], axis=1)

X\_test.head()

**22. Is the data balanced w.r.t target column?**

df['Column name'].value\_counts()/df.shape[0]\*100

df['Column name'].value\_counts()/len(df)\*100

**23. Type-casting variables to correct data-types**

for attr in object\_attrs:

df[attr] = df[attr].astype("category")

or

cat\_attr=['Education', 'Family', 'CDAccount', 'Online','CreditCard',

'SecuritiesAccount']

for cols in cat\_attr :

df[cols]=df[cols].astype('category')

**23.Machine learning models**

**SVM**

import numpy as np

from sklearn.datasets.samples\_generator import make\_blobs

from sklearn.model\_selection import train\_test\_split

from matplotlib import pyplot as plt

from sklearn.svm import LinearSVC

from sklearn.metrics import confusion\_matrix

from sklearn import svm

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score,f1\_score

linear\_svm = SVC(kernel='linear', C=1, random\_state=0)

linear\_svm.fit(X=X\_train, y= y\_train)

train\_predictions = linear\_svm.predict(X\_train)

test\_predictions = linear\_svm.predict(X\_test)

**### Train data accuracy**

print("TRAIN Conf Matrix : \n", confusion\_matrix(y\_train, train\_predictions))

print("\nTRAIN DATA ACCURACY",accuracy\_score(y\_train,train\_predictions))

print("\nTrain data f1-score for class '1'",f1\_score(y\_train,train\_predictions,pos\_label=1))

print("\nTrain data f1-score for class '2'",f1\_score(y\_train,train\_predictions,pos\_label=0))

**### Test data accuracy**

print("\n\n--------------------------------------\n\n")

print("TEST Conf Matrix : \n", confusion\_matrix(y\_test, test\_predictions))

print("\nTEST DATA ACCURACY",accuracy\_score(y\_test,test\_predictions))

print("\nTest data f1-score for class '1'",f1\_score(y\_test,test\_predictions,pos\_label=1))

print("\nTest data f1-score for class '2'",f1\_score(y\_test,test\_predictions,pos\_label=0))

**Decision tree**

from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier(criterion = 'entropy', random\_state = 0)

classifier.fit(X\_train, y\_train)

from sklearn.metrics import accuracy\_score,f1\_score

**### Train data accuracy**

print("TRAIN Conf Matrix : \n", confusion\_matrix(y\_train, train\_predictions))

print("\nTRAIN DATA ACCURACY",accuracy\_score(y\_train,train\_predictions))

print("\nTrain data f1-score for class '1'",f1\_score(y\_train,train\_predictions,pos\_label=1))

print("\nTrain data f1-score for class '2'",f1\_score(y\_train,train\_predictions,pos\_label=0))

**### Test data accuracy**

print("\n\n--------------------------------------\n\n")

print("TEST Conf Matrix : \n", confusion\_matrix(y\_test, test\_predictions))

print("\nTEST DATA ACCURACY",accuracy\_score(y\_test,test\_predictions))

print("\nTest data f1-score for class '1'",f1\_score(y\_test,test\_predictions,pos\_label=1))

print("\nTest data f1-score for class '2'",f1\_score(y\_test,test\_predictions,pos\_label=0))

**Naive Bayes**

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(X\_train, y\_train)

train\_predictions = classifier.predict(X\_train)

test\_predictions = classifier.predict(X\_test)

**### Train data accuracy**

from sklearn.metrics import accuracy\_score,f1\_score

print("TRAIN Conf Matrix : \n", confusion\_matrix(y\_train, train\_predictions))

print("\nTRAIN DATA ACCURACY",accuracy\_score(y\_train,train\_predictions))

print("\nTrain data f1-score for class '1'",f1\_score(y\_train,train\_predictions,pos\_label=1))

print("\nTrain data f1-score for class '2'",f1\_score(y\_train,train\_predictions,pos\_label=0))

**### Test data accuracy**

print("\n\n--------------------------------------\n\n")

print("TEST Conf Matrix : \n", confusion\_matrix(y\_test, test\_predictions))

print("\nTEST DATA ACCURACY",accuracy\_score(y\_test,test\_predictions))

print("\nTest data f1-score for class '1'",f1\_score(y\_test,test\_predictions,pos\_label=1))

print("\nTest data f1-score for class '2'",f1\_score(y\_test,test\_predictions,pos\_label=0))

**Logistic Regression**

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0)

classifier.fit(X\_train, y\_train)

train\_predictions = classifier.predict(X\_train)

test\_predictions = classifier.predict(X\_test)

**### Train data accuracy**

from sklearn.metrics import accuracy\_score,f1\_score

print("TRAIN Conf Matrix : \n", confusion\_matrix(y\_train, train\_predictions))

print("\nTRAIN DATA ACCURACY",accuracy\_score(y\_train,train\_predictions))

print("\nTrain data f1-score for class '1'",f1\_score(y\_train,train\_predictions,pos\_label=1))

print("\nTrain data f1-score for class '2'",f1\_score(y\_train,train\_predictions,pos\_label=0))

**### Test data accuracy**

print("\n\n--------------------------------------\n\n")

print("TEST Conf Matrix : \n", confusion\_matrix(y\_test, test\_predictions))

print("\nTEST DATA ACCURACY",accuracy\_score(y\_test,test\_predictions))

print("\nTest data f1-score for class '1'",f1\_score(y\_test,test\_predictions,pos\_label=1))

print("\nTest data f1-score for class '2'",f1\_score(y\_test,test\_predictions,pos\_label=0))

**Random forest classifier**

from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n\_estimators = 10, criterion = 'entropy', random\_state = 0)

classifier.fit(X\_train, y\_train)

train\_predictions = classifier.predict(X\_train)

test\_predictions = classifier.predict(X\_test)

**### Train data accuracy**

from sklearn.metrics import accuracy\_score,f1\_score

print("TRAIN Conf Matrix : \n", confusion\_matrix(y\_train, train\_predictions))

print("\nTRAIN DATA ACCURACY",accuracy\_score(y\_train,train\_predictions))

print("\nTrain data f1-score for class '1'",f1\_score(y\_train,train\_predictions,pos\_label=1))

print("\nTrain data f1-score for class '2'",f1\_score(y\_train,train\_predictions,pos\_label=0))

**### Test data accuracy**

print("\n\n--------------------------------------\n\n")

print("TEST Conf Matrix : \n", confusion\_matrix(y\_test, test\_predictions))

print("\nTEST DATA ACCURACY",accuracy\_score(y\_test,test\_predictions))

print("\nTest data f1-score for class '1'",f1\_score(y\_test,test\_predictions,pos\_label=1))

print("\nTest data f1-score for class '2'",f1\_score(y\_test,test\_predictions,pos\_label=0))

**XGBoost**

from numpy import loadtxt

from xgboost import XGBClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

model = XGBClassifier()

model.fit(X\_train, y\_train)

train\_predictions = classifier.predict(X\_train)

test\_predictions = classifier.predict(X\_test)

**### Train data accuracy**

from sklearn.metrics import accuracy\_score,f1\_score

print("TRAIN Conf Matrix : \n", confusion\_matrix(y\_train, train\_predictions))

print("\nTRAIN DATA ACCURACY",accuracy\_score(y\_train,train\_predictions))

print("\nTrain data f1-score for class '1'",f1\_score(y\_train,train\_predictions,pos\_label=1))

print("\nTrain data f1-score for class '2'",f1\_score(y\_train,train\_predictions,pos\_label=0))

**### Test data accuracy**

print("\n\n--------------------------------------\n\n")

print("TEST Conf Matrix : \n", confusion\_matrix(y\_test, test\_predictions))

print("\nTEST DATA ACCURACY",accuracy\_score(y\_test,test\_predictions))

print("\nTest data f1-score for class '1'",f1\_score(y\_test,test\_predictions,pos\_label=1))

print("\nTest data f1-score for class '2'",f1\_score(y\_test,test\_predictions,pos\_label=0))

**Linear Regression**

from sklearn import linear\_model

from sklearn.linear\_model import LinearRegression

reg = linear\_model.LinearRegression()

reg.fit(X\_train, y\_train)

train\_predictions = classifier.predict(X\_train)

test\_predictions = classifier.predict(X\_test)

**### Train data accuracy**

from sklearn.metrics import accuracy\_score,f1\_score

print("TRAIN Conf Matrix : \n", confusion\_matrix(y\_train, train\_predictions))

print("\nTRAIN DATA ACCURACY",accuracy\_score(y\_train,train\_predictions))

print("\nTrain data f1-score for class '1'",f1\_score(y\_train,train\_predictions,pos\_label=1))

print("\nTrain data f1-score for class '2'",f1\_score(y\_train,train\_predictions,pos\_label=0))

**### Test data accuracy**

print("\n\n--------------------------------------\n\n")

print("TEST Conf Matrix : \n", confusion\_matrix(y\_test, test\_predictions))

print("\nTEST DATA ACCURACY",accuracy\_score(y\_test,test\_predictions))

print("\nTest data f1-score for class '1'",f1\_score(y\_test,test\_predictions,pos\_label=1))

print("\nTest data f1-score for class '2'",f1\_score(y\_test,test\_predictions,pos\_label=0))

**k-nearest neighbor(KNN)**

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import load\_iris

knn = KNeighborsClassifier(n\_neighbors=7)

knn.fit(X\_train, y\_train)

train\_predictions = knn.predict(X\_train)

test\_predictions = knn.predict(X\_test)

**### Train data accuracy**

from sklearn.metrics import accuracy\_score,f1\_score

print("TRAIN Conf Matrix : \n", confusion\_matrix(y\_train, train\_predictions))

print("\nTRAIN DATA ACCURACY",accuracy\_score(y\_train,train\_predictions))

print("\nTrain data f1-score for class '1'",f1\_score(y\_train,train\_predictions,pos\_label=1))

print("\nTrain data f1-score for class '2'",f1\_score(y\_train,train\_predictions,pos\_label=0))

**### Test data accuracy**

print("\n\n--------------------------------------\n\n")

print("TEST Conf Matrix : \n", confusion\_matrix(y\_test, test\_predictions))

print("\nTEST DATA ACCURACY",accuracy\_score(y\_test,test\_predictions))

print("\nTest data f1-score for class '1'",f1\_score(y\_test,test\_predictions,pos\_label=1))

print("\nTest data f1-score for class '2'",f1\_score(y\_test,test\_predictions,pos\_label=0))

**K-Means**

import numpy as np

import pandas as pd

from matplotlib import pyplot as plt

from sklearn.datasets.samples\_generator import make\_blobs

from sklearn.cluster import KMeans

from sklearn.cluster import KMeans

kmeans = KMeans(n\_clusters=2)

kmeans.fit(X\_train, y\_train)

train\_predictions = kmeans.predict(X\_train)

test\_predictions = kmeans.predict(X\_test)

**### Train data accuracy**

from sklearn.metrics import accuracy\_score,f1\_score

print("TRAIN Conf Matrix : \n", confusion\_matrix(y\_train, train\_predictions))

print("\nTRAIN DATA ACCURACY",accuracy\_score(y\_train,train\_predictions))

print("\nTrain data f1-score for class '1'",f1\_score(y\_train,train\_predictions,pos\_label=1))

print("\nTrain data f1-score for class '2'",f1\_score(y\_train,train\_predictions,pos\_label=0))

**### Test data accuracy**

print("\n\n--------------------------------------\n\n")

print("TEST Conf Matrix : \n", confusion\_matrix(y\_test, test\_predictions))

print("\nTEST DATA ACCURACY",accuracy\_score(y\_test,test\_predictions))

print("\nTest data f1-score for class '1'",f1\_score(y\_test,test\_predictions,pos\_label=1))

print("\nTest data f1-score for class '2'",f1\_score(y\_test,test\_predictions,pos\_label=0))

**For unseendata**

**24.Reading unseendata csv files**

unseendata=pd.read\_csv('df.csv', na\_values=["?", "", "null"])

**25. Prediction**

(Here id is not significant which is used for prediction)

unseendata\_ids = unseendata.loc[:, "id"]

unseendata.drop(["id"], axis = 1, inplace=True)

**26. Target column** (here class is target variable)

X\_unseendata=unseendata.loc[:,unseendata.columns!='class']

y\_unseendata=unseendata.loc[:, "class"]

**27. Final output csv**

final\_output=pd.DataFrame({'id':unseendata\_ids, 'prediction':unseendata\_predictions})

id\_n\_prediction = ["id", "prediction"]

final\_output = final\_output.loc[:,id\_n\_prediction]

#final\_output.to\_csv("final\_output01.csv")

final\_output

**28. Download final csv to specific path**

(Here /kaggle/working/final\_output01.csv is path to get download)

final\_output.to\_csv("/kaggle/working/final\_output01.csv")

**29. How to avoid Unnamed: 0 columns**

df = pd.read\_csv("file.csv", index\_col=0)

**30. Title mapping**

train\_test\_data = [train, test] # combining train and test dataset

for dataset in train\_test\_data:

dataset['Title'] = dataset['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)

Title map

Mr : 0

Miss : 1

Mrs: 2

Others: 3

title\_mapping = {"Mr": 0, "Miss": 1, "Mrs": 2,

"Master": 3, "Dr": 3, "Rev": 3, "Col": 3, "Major": 3, "Mlle": 3,"Countess": 3,

"Ms": 3, "Lady": 3, "Jonkheer": 3, "Don": 3, "Dona" : 3, "Mme": 3,"Capt": 3,"Sir": 3 }

for dataset in train\_test\_data:

dataset['Title'] = dataset['Title'].map(title\_mapping)

**31.To check unique values in columns**

print(df['col\_name'].unique())

**32.To add new column to df**

df['new\_feature']=value

**33.To handle years**

df['number\_years']=df['current year']-df['year']

**34. To check important features**

Print(model.feature\_importance\_)

**35.** **Are you using train\_test\_split with a classification problem?**

**Be sure to set "stratify=y" so that class proportions are preserved when splitting.Especially important if you have class imbalance!**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, random\_state=0, stratify=y)

**36. Need to impute missing values for a categorical feature?**

**Two options:**

**Impute the most frequent value**

**Impute the value "missing", which treats it as a separate category**

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='most\_frequent')

imputer.fit\_transform(X)

or

imputer = SimpleImputer(strategy='constant', fill\_value='missing')

imputer.fit\_transform(X)

**37. Display as word cloud -display dataframe in word picture**

**# making a word cloud for df**

from wordcloud import WordCloud

from wordcloud import STOPWORDS

plt.rcParams['figure.figsize'] = (15, 12)

stopwords = set(STOPWORDS)

wordcloud = WordCloud(background\_color = 'gray',

max\_words = 200,

stopwords = stopwords,

width = 1200,

height = 800,

random\_state = 42).generate(str(df['col\_name']))

plt.title('Wordcloud for Shop Names', fontsize = 30)

plt.axis('off')

plt.imshow(wordcloud, interpolation = 'bilinear')

**38. Using LightGBM model**

**from lightgbm import LGBMRegressor**

model\_lgb = LGBMRegressor( n\_estimators=200,

learning\_rate=0.03,

num\_leaves=32,

colsample\_bytree=0.9497036,

subsample=0.8715623,

max\_depth=8,

reg\_alpha=0.04,

reg\_lambda=0.073,

min\_split\_gain=0.0222415,

min\_child\_weight=40)

model\_lgb.fit(x\_train, y\_train)

y\_pred\_lgb = model\_lgb.predict(x\_test)

**39. Change specific values in column( eg:14+ as 15, 12a as 14)**

df["col\_name"].replace({"14+":"15"}, inplace=True)

or

train\_data.replace({"non-stop": 0, "1 stop": 1, "2 stops": 2, "3 stops": 3, "4 stops": 4}, inplace = True)

**40. You can then use to\_numeric in order to convert the values in the dataset into a float format. But since 3 of those values are non-numeric, you’ll get ‘NaN’ for those 3 values.Here is the code that you may then use to get the NaN values:**

train\_df = train\_df.apply (pd.to\_numeric, errors='coerce')

**41. Drop the Rows with NaN Values in Pandas DataFrame**

df = df.apply (pd.to\_numeric, errors='coerce')

df = df.dropna()

df.dropna(inplace=True)-better to use

**42. Reset the Index**

df.reset\_index(drop=True)

**43. EDA using pandas profiling**

!pip install <https://github.com/pandas-profiling/pandas-profiling/archive/master.zip>

import pandas\_profiling as pp

from pandas\_profiling import ProfileReport

profile = pp.ProfileReport(df, title='Pandas Profiling Report', explorative=True)

profile.to\_file("profile.html")

profile.to\_notebook\_iframe()

**44. Extract dataframe to excel**

df.to\_csv(‘Test1.csv’)

**45. Create own Dataframe**

df=pd.DataFrame(np.arange(0,20).reshape(5,4),index=['Row1','Row2','Row3','Row4','Row5'],columns=["Column1","Column2","Column3","Coumn4"])

**46. Accessing the elements two ways**

loc

iloc

df.loc[‘Row1’]

df.iloc[:,:]

**46. Read CSV file separated by ;(semicolon)**

df=pd.read\_csv(‘file.csv’,sep=’;’)

**47. To display all the column**

pd.set\_option('display.max\_columns' ,None)

**48. Rename columns of Dataframe**

df.columns=[‘new col name’,’new col name’]

**49. Concat columns to datframe with steps( here State is dummified with new df1 name)**

df1=pd.get\_dummies(df['State'],drop\_first=True)

df=pd.concat([df1,df],axis=1)

df.drop('State',axis=1,inplace=True)

**50. Pycaret for automl**