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(here Unnamed: 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 after train 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