**PYSPARK**

PySpark is the Python API for Apache Spark, an open source, distributed computing framework and set of libraries for real-time, large-scale data processing. If you're already familiar with Python and libraries such as Pandas

First, we need to create the new environment

->go to anaconda cmd

-> conda env list -> is used to see the list of envs

-> conda create - -name =test -> is used to create the env

-> conda activate test -> is used to activate the environment

-> conda deactivate -> vice versa

-> conda install -c anaconda ipykernel

-> python -m ipykernel install --user --name=trial -> is used to set the kernal to jupyter

-> jupyter.exe notebook -> to run the notebook from that env

!pip install pyspark -> it is used to install

Import pyspark -> used to test if it is installed or not

Then we will create a table using the excel and we will read that excel using the pandas

-> import pandas as pd

-> pd.read\_csv('name.csv') -> is used to read the csv file using pandas

-> from pyspark.sql import SparkSession

-> spark=SparkSession.builder.appName('practice').getOrCreate() // we do above 2 commands to create the spark session

-> spark -> is used to store the spark session

-> df\_pyspark=spark.read.csv(‘name.csv’)

-> df\_pyspark -> is used to see the schema

-> df\_pyspark.show() -> is used to see the table

-> spark.read.option(‘header’,’true’).csv(‘name.csv’) -> is used to read schema of headings of the table

-> spark.read.option('header', 'true').csv('name.csv').show() -> to see entire table from headings of the table

**### To print the schema**

df\_pyspark= sprak.read.option('header', 'true').csv('name.csv')

df\_pyspark.printSchema() // But it will take everything as a string to change that

df\_pyspark= sprak.read.option('header', 'true').csv('name.csv',inferSchema=True)

df\_pyspark.printSchema()

df\_pyspark**=**spark**.**read**.**csv('test1.csv',header**=True**,inferSchema**=True**)  
df\_pyspark**.**show()

***### Check the schema***

*df\_pyspark***.**printSchema()

type(df\_pyspark) -> it is used to see the dataframe (data frame is data structure)

**### Displaying the rows and columns**

df\_pyspark.columns -> to get the column names

df\_pyspark.head(2) -> to get the no of rows with details

df\_pyspark.select('name').show() -> is used to see column with values

df\_pyspark.select(['name','age']).show() -> we can get multiple columns

**### For getting the datatype of the columns**

df\_pyspark.select('tip').dtypes

df\_pyspark.dtypes-> used to get the data type

**### Adding columns to the table**

df\_pyspark=df\_pyspark.withColumn('Experence after 2 years',df\_pyspark['Experience']+2)

df\_pyspark.show()

from pyspark.sql.functions import lit

df\_pyspark=df\_pyspark.withColumn('Experence after 2 years',lit(2)) -> to add common value

**### Droping the Columns**

df\_pyspark=df\_pyspark.drop('Experence after 2 years')

df\_pyspark= df\_pyspark.drop('Experence after 2 years','salary') -> for multiple columns

**### For Renaming the Column**

df\_pyspark= df\_pyspark.withColumnRenamed('salary','money')

**### Droping the Rows**

df\_pyspark.na.drop().show()

df\_pyspark.na.drop(how='all').show() -> when how is all if only all values in row is null it will drop

df\_pyspark.na.drop(how='any',thresh=2).show() -> we set thresh as 2 so if 2 nulls in row it drop

df\_pyspark.na.drop(how='any',subset=['Experience']).show() -> all nulls in Specific col will be drop

**### Fill the missing values**

df\_pyspark.na.fill('missing').show() -> it will fill the null with the string only if its string value

df\_pyspark.na.fill(1,'age').show() -> to specify the column

df\_pyspark.na.fill(1,['age','Experience']).show() -> to specify the particular columns

**#Imputer -> we use this to replace null with mean median or mode**

from pyspark.ml.feature import Imputer

imputer = Imputer(

inputCols=['age', 'Experience', 'Salary'],

outputCols=["{}\_imputed".format(c) for c in ['age', 'Experience', 'Salary']]

).setStrategy("median")

imputer.fit(df\_pyspark).transform(df\_pyspark).show()

**### Filtering the data**

df\_pyspark.filter('Salary>=20000').show() -> get the data with salary >=20000

df\_pyspark.filter('Salary>=20000').select(['Name','age']).show() -> to get specific columns

df\_pyspark.filter('Salary>=20000 and Salary <25000').select(['Name','age']).show() -> multi condi

df\_pyspark.filter(df\_pyspark['sex']=='Male').show()

**#OR**

df\_pyspark.filter((df\_pyspark['salary']>=20000)&(df\_pyspark['salary']<=25000)).select(['Name','age']).show()

**#Except operation (except the condition all the data will be displayed )**

df\_pyspark.filter(~(df\_pyspark['salary']>=20000)).select(['Name','age']).show()

**### GROUP BY AND AGGREGATE**

df\_pyspark.groupBy('name').sum('Salary').show() -> to get sum

df\_pyspark.groupBy('Departments').count().show()

df\_pyspark.agg({'Salary':'sum'}).show() -> we can also do direct aggregate functions

df\_pyspark.agg({'Salary':'sum','Departments':'count'}).show() -> we can also do for the multiple

You can do max min mean avg etc we can check by pressing the tab

**### Predicting the Salary based on the age and experience**

from pyspark.ml.feature import VectorAssembler

featureassembler=VectorAssembler(inputCols=["age","Experience"],outputCol="Independent Features")

output=featureassembler.transform(training)

output.show()

finalized\_data=output.select("Independent Features","Salary")

finalized\_data.show()

**\*\*\*All we did till here is we combined the columns age and experience and created a new column and then we created a variable named finalized data and we stored the only needed columns which is new column and salary**

from pyspark.ml.regression import LinearRegression

##train test split

train\_data,test\_data=finalized\_data.randomSplit([0.75,0.25])

regressor=LinearRegression(featuresCol='Independent Features', labelCol='Salary')

regressor=regressor.fit(train\_data)

regressor.coefficients

regressor.intercept

pred\_results=regressor.evaluate(test\_d

pred\_results.predictions.show()

pred\_results.meanAbsoluteError,pred\_results.meanSquaredError

**DATA BRICK**

Databricks is an industry-leading, cloud-based data engineering tool used for **processing and transforming massive quantities of data and exploring the data through machine learning models**. Recently added to Azure, it's the latest big data tool for the Microsoft cloud.

Click on create a cluster

Name it

You can select the version you want

Click Create

Once it created you can find it on compute tab

Now you need to create the data to work with our cluster

Click on create the table

Then you can upload the data set

After upload click create table in notebook

**# File location and type**

file\_location = "/FileStore/tables/test1.csv"

file\_type = "csv"

**# CSV options**

df\_pyspark=spark.read.csv(file\_location,header=True,inferSchema=True)

**### Handling Categorical Features**

from pyspark.ml.feature import StringIndexer

indexer=StringIndexer(inputCol="sex",outputCol="sex\_indexed")

df\_pyspark\_i=indexer.fit(df\_pyspark).transform(df\_pyspark)

df\_pyspark\_i.show()

indexer=StringIndexer(inputCols=["sex",'day','time'],outputCols=["sex\_indexed",'day\_indx','time\_indx']) -> for multiple cols

**### taking all the converted features into list format**

from pyspark.ml.feature import VectorAssembler

featureassembler=VectorAssembler(inputCols=['tip','size','sex\_indexed','smoker\_indx','day\_indx',

'time\_indx'],outputCol="Independent Features")

result=featureassembler.transform(df\_pyspark\_i)

result.show()

**### we will select all the independent features and dependent features**

fianl\_result=result.select('Independent Features','total\_bill')

from pyspark.ml.regression import LinearRegression

##train test split

train\_data,test\_data=final\_result.randomSplit([0.75,0.25])

regressor=LinearRegression(featuresCol='Independent Features', labelCol='total\_bill')

regressor=regressor.fit(train\_data)

regressor.coefficients -> to see the coeffient

Regressor.intercept

pred\_results=regressor.evaluate(test\_data)

pred\_results.predictions.show()

pred\_results.r2,pred\_results.meanAbsoluteError,pred\_results.meanSquaredError