Transformations

Explore the packages in python

There are multiple packages in python

1. Standard Library Packages (Built-in)

These come with Python and don't require installation.

- os – Interact with the operating system.
- sys – Access system-specific parameters and functions.
- math – Mathematical functions like trigonometry and logarithms.
- datetime – Work with dates and times.
- json – Handle JSON data.
- re – Regular expressions for text processing.
- random – Generate random numbers.

2. Data Science & Machine Learning

Popular packages used for analysis and AI.

- numpy – Numerical computations, arrays, matrices.
- pandas – Data manipulation and analysis.
- matplotlib – Visualization and plotting.
- seaborn – Statistical data visualization.
- scipy – Scientific computing.
- scikit-learn – Machine learning algorithms.
- tensorflow / pytorch – Deep learning frameworks.

3. Web Development

Frameworks and tools for building web applications.

- flask – Lightweight web framework.
- django – Full-stack web framework.
- fastapi – High-performance web APIs.
- requests – Send HTTP requests easily.
- beautifulsoup4 – Web scraping and parsing HTML/XML.

4. Automation & Scripting

Packages to automate tasks.

- selenium – Automate web browsers.
- pyautogui – Automate mouse and keyboard actions.
- shutil – File operations (copy, move, delete).
- pyperclip – Copy and paste text to the clipboard.

5. Database Interaction

For handling databases in Python.

- sqlite3 – Built-in database support.
- sqlalchemy – SQL toolkit and ORM.
- pymysql – Interface for MySQL databases.

- psycopg2 – PostgreSQL database adapter.

6. Networking & APIs

For working with network-related tasks.

- socket – Low-level networking.
- asyncio – Asynchronous programming.
- httpx – Advanced HTTP client.
- websockets – WebSocket communication.

7. Cybersecurity & Cryptography

Packages for security-related tasks.

- cryptography – Encrypt and decrypt data.
- hashlib – Secure hash functions.
- scapy – Network packet manipulation.

8. Game Development

For building 2D/3D games.

- pygame – 2D game development.
- arcade – Modern game framework.
- panda3d – 3D game engine.

9. Cloud & DevOps

Interacting with cloud platforms.

- boto3 – AWS services.
- google-cloud – Google Cloud APIs.
- azure-storage-blob – Azure storage.

10. AI & NLP

Natural Language Processing and AI.

- nltk – Natural Language Toolkit.
- spacy – NLP and text processing.
- transformers – Pre-trained AI models.

Explore the packages in pyspark

1. Core Packages in PySpark

These are the main components of PySpark:

a) pyspark.sql (Structured Data & SQL)

- Provides functionalities to work with structured data using DataFrames and SQL.
- Supports reading from and writing to multiple data sources (CSV, Parquet, JSON, Hive, etc.).
- Common modules:
    - pyspark.sql.SparkSession – Entry point for working with DataFrames.
    - pyspark.sql.functions – Collection of built-in functions (e.g., col(), when(), lit(), etc.).
    - pyspark.sql.types – Defines data types for DataFrames (e.g., StringType, IntegerType).

- pyspark.sql.window – Window functions for advanced aggregations.

b) pyspark.ml (Machine Learning)

- Provides a distributed ML library with various algorithms.
- Uses a pipeline-based approach for ML workflows.

Key modules:

- pyspark.ml.classification – Logistic Regression, Decision Tree, etc.
- pyspark.ml.regression – Linear Regression, Random Forest, etc.
- pyspark.ml.clustering – K-Means, Gaussian Mixture Model.
- pyspark.ml.feature – Feature transformation and selection (e.g., VectorAssembler, StringIndexer).
- pyspark.ml.tuning – Hyperparameter tuning (CrossValidator, ParamGridBuilder).

c) pyspark.streaming (Real-time Data Processing)

- Handles real-time streaming data.
- Supports integration with sources like Kafka, HDFS, and sockets.

d) pyspark.graphframes (Graph Processing)

- Works with graph-based data.
- Supports algorithms like PageRank, BFS, and connected components.

e) pyspark.mllib (Legacy Machine Learning)

- Older ML library, replaced by pyspark.ml.
- Still used for RDD-based ML models.

2. External Packages for PySpark

Besides the core PySpark modules, there are additional packages that extend its capabilities:

Package Name	Description
delta	Delta Lake for ACID transactions on Spark.
koalas	Pandas-like API for PySpark DataFrames.
graphframes	Graph analytics library for PySpark.
spark-nlp	Natural Language Processing (NLP) in PySpark.
pysparkling	Integration of PySpark with H2O AI for ML models.

3. PySpark Data Sources

PySpark supports reading/writing from various data sources:

- **File formats**: CSV, JSON, Parquet, ORC, Avro.
- **Databases**: MySQL, PostgreSQL, MongoDB, Cassandra.
- **Cloud Storage**: AWS S3, Azure Blob, Google Cloud Storage.

4. Performance Optimization in PySpark

- **Use DataFrames instead of RDDs** (DataFrames are optimized).
- **Enable caching** (df.cache() to store intermediate results).
- **Use broadcast variables** (broadcast() for small lookup tables).
- **Optimize shuffle operations** (Use repartition() and coalesce()).

Explore the transformations

Check if we have mount connection to storage account

```
02:02 PM (4s)                                    1
dbutils.fs.ls("/mnt/container1")
```

```
[FileInfo(path='dbfs:/mnt/container1/CSV_Data/', name='CSV_Data/', size=0, modificationTime=0),
 FileInfo(path='dbfs:/mnt/container1/Delta_Data/', name='Delta_Data/', size=0, modificationTime=0),
 FileInfo(path='dbfs:/mnt/container1/Parquet_Data/', name='Parquet_Data/', size=0, modificationTime=0)]
```

Read the file from the storage account

```
1 minute ago (10s)                               2                                   Python
df=spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("/mnt/container1/CSV_Data/
Employee_2025-03-11T17_07_01.8945923Z (1).csv")
display(df)
(3) Spark Jobs
df: pyspark.sql.dataframe.DataFrame = [ID: integer, E_Name: string ... 2 more fields]
```

Table +

	ID	E_Name	E_City	E_Phonenumber
1	1	Robert	Toronto	2499791376
2	2	Ann	Brampton	2499799087
3	3	John	Montreal	2499793456
4	1	Robert	Toronto	2499791376
5	2	Ann	Toronto	2499799087
6	4	John	Vancouver	2499796785

6 rows | 10.49s runtime Refreshed now

We see the file have some duplicate values

Select only specific columns

To select only specific columns of the data frame, we need to use a function called as “col” which is in package pyspark.sql.fucntion, So we need to import this package before we use that function.

```
1 minute ago (<1s)                               4
from pyspark.sql.functions import col
```

Now we can select the columns only which we want
df_select=df.select(col("ID"), col("E_Name"))
display(df_select)

```
df_select=df.select(col("ID"), col("E_Name"))
display(df_select)
```

(1) Spark Jobs

df_select: pyspark.sql.dataframe.DataFrame = [ID: integer, E_Name: string]

	ID	E_Name
1	1	Robert
2	2	Ann
3	3	John
4	1	Robert
5	2	Ann
6	4	John

6 rows | 0.63s runtime

Refreshed 1 minute ago

Here we see df_select data frame will have data of ID and E_name only

We can rename the columns using same "**col**" function or else we have another function called as "**withColumnRename**" function

Renaming the Column:

To rename the columns we can use **col** function along with **alias**
df_rename_col= df.select(col("ID").alias("Employee_ID"), col("E_Name").alias("Employee_Name"), col("E_City").alias("Employee_City"), col("E_Phonenumber").alias("Employee_Phonenumber"))
display(df_rename_col)

```
df_rename_col= df.select(col("ID").alias("Employee_ID"), col("E_Name").alias("Employee_Name"), col("E_City").alias
("Employee_City"), col("E_Phonenumber").alias("Employee_Phonenumber"))
display(df_rename_col)
```

(1) Spark Jobs

df_rename_col: pyspark.sql.dataframe.DataFrame = [Employee_ID: integer, Employee_Name: string ... 2 more fields]

	Employee_ID	Employee_Name	Employee_City	Employee_Phonenumber
1	1	Robert	Toronto	2499791376
2	2	Ann	Brampton	2499799087
3	3	John	Montreal	2499793456
4	1	Robert	Toronto	2499791376
5	2	Ann	Toronto	2499799087
6	4	John	Vancouver	2499796785

6 rows | 0.71s runtime

Refreshed now

Here we see all the columns names have changed

We can use **withColumnsRename** function as well to rename the columns
df_rename_with=df.withColumnRenamed("ID", "Emp_ID").withColumnRenamed("E_Name", "Emp_Name").withColumnRenamed("E_City", "Emp_City").withColumnRenamed("E_Phonenumber", "Emp_Phonenumber")
display(df_rename_with)

Just now (<1s) 9 Python

```python
df_rename_with=df.withColumnRenamed("ID", "Emp_ID").withColumnRenamed("E_Name", "Emp_Name").withColumnRenamed
("E_City", "Emp_City").withColumnRenamed("E_Phonenumber", "Emp_Phonenumber")
display(df_rename_with)
```

(1) Spark Jobs

df_rename_with: pyspark.sql.dataframe.DataFrame = [Emp_ID: integer, Emp_Name: string ... 2 more fields]

Table

	Emp_ID	Emp_Name	Emp_City	Emp_Phonenumber
1	1	Robert	Toronto	2499791376
2	2	Ann	Brampton	2499799087
3	3	John	Montreal	2499793456
4	1	Robert	Toronto	2499791376
5	2	Ann	Toronto	2499799087
6	4	John	Vancouver	2499796785

6 rows | 0.47s runtime Refreshed now

Removing unnecessary columns

To remove unnecessary columns, we have to use drop function to remove those columns
df_remove_columns=df.drop("E_City", "E_Phonenumber")
display(df_remove_columns)

Just now (<1s) 11 Python

```python
df_remove_columns=df.drop("E_City", "E_Phonenumber")
display(df_remove_columns)
```

(1) Spark Jobs

df_remove_columns: pyspark.sql.dataframe.DataFrame = [ID: integer, E_Name: string]

Table

	ID	E_Name
1	1	Robert
2	2	Ann
3	3	John
4	1	Robert
5	2	Ann
6	4	John

6 rows | 0.38s runtime Refreshed now

Identifying and Handling Duplicates

First, we can use **dropduplicates** function to remove any duplicates from the file
df_dup=df.dropDuplicates()
display(df_dup)
This will remove complete row which are duplicates

1 minute ago (2s) 13 Python

```
df_dup=df.dropDuplicates()
display(df_dup)
```

(2) Spark Jobs

df_dup: pyspark.sql.dataframe.DataFrame = [ID: integer, E_Name: string ... 2 more fields]

Table

	ID	E_Name	E_City	E_Phonenumber
1	2	Ann	Toronto	2499799087
2	4	John	Vancouver	2499796785
3	3	John	Montreal	2499793456
4	1	Robert	Toronto	2499791376
5	2	Ann	Brampton	2499799087

5 rows | 1.60s runtime Refreshed 1 minute ago

We can also use **drop_duplicates** to remove any duplicates from the file
df_dup2=df.drop_duplicates()
display(df_dup2)

Just now (1s) 14 Python

```
df_dup2=df.drop_duplicates()
display(df_dup2)
```

(2) Spark Jobs

df_dup2: pyspark.sql.dataframe.DataFrame = [ID: integer, E_Name: string ... 2 more fields]

Table

	ID	E_Name	E_City	E_Phonenumber
1	2	Ann	Toronto	2499799087
2	4	John	Vancouver	2499796785
3	3	John	Montreal	2499793456
4	1	Robert	Toronto	2499791376
5	2	Ann	Brampton	2499799087

5 rows | 0.70s runtime Refreshed now

We can also use **Distinct ()**
df_dup_distinct=df.distinct()
display(df_dup_distinct)

Just now (1s) 17 Python

```
df_dup_distinct=df.distinct()
display(df_dup_distinct)
```

(2) Spark Jobs

df_dup_distinct: pyspark.sql.dataframe.DataFrame = [ID: integer, E_Name: string ... 2 more fields]

Table

	ID	E_Name	E_City	E_Phonenumber
1	2	Ann	Toronto	2499799087
2	4	John	Vancouver	2499796785
3	3	John	Montreal	2499793456
4	1	Robert	Toronto	2499791376
5	2	Ann	Brampton	2499799087

5 rows | 0.54s runtime Refreshed now

But by using **dropduplicate** or **drop_duplicates fucniton** we can specify for which columns we want to remove the duplicates but we cannot specify this for **distinct** function

df_sepc_dup= df.dropDuplicates(["ID"])
display(df_sepc_dup)

Just now (1s) 19 Python

```
df_sepc_dup= df.dropDuplicates(["ID"])
display(df_sepc_dup)
```

(2) Spark Jobs

df_sepc_dup: pyspark.sql.dataframe.DataFrame = [ID: integer, E_Name: string ... 2 more fields]

Table

	ID	E_Name	E_City	E_Phonenumber
1	1	Robert	Toronto	2499791376
2	2	Ann	Brampton	2499799087
3	3	John	Montreal	2499793456
4	4	John	Vancouver	2499796785

4 rows | 0.70s runtime Refreshed now

df_sepc_dup1=df.drop_duplicates(["E_Name"])
display(df_sepc_dup1)

Just now (<1s) 20 Python

```
df_sepc_dup1=df.drop_duplicates(["E_Name"])
display(df_sepc_dup1)
```

(2) Spark Jobs

df_sepc_dup1: pyspark.sql.dataframe.DataFrame = [ID: integer, E_Name: string ... 2 more fields]

Table

	ID	E_Name	E_City	E_Phonenumber
1	2	Ann	Brampton	2499799087
2	3	John	Montreal	2499793456
3	1	Robert	Toronto	2499791376

Applying Filter Conditions

df_filter = df.filter(df.E_Name == "John")
display(df_filter)

Just now (1s) 22 Python

```
df_filter = df.filter(df.E_Name == "John")
display(df_filter)
```

(1) Spark Jobs

df_filter: pyspark.sql.dataframe.DataFrame = [ID: integer, E_Name: string ... 2 more fields]

Table

	ID	E_Name	E_City	E_Phonenumber
1	3	John	Montreal	2499793456
2	4	John	Vancouver	2499796785

Can Also use where function to filter the data

df_where= df.where(df.E_Name == "John")
display(df_where)

Just now (<1s) 24 Python

```
df_where= df.where(df.E_Name == "John")
display(df_where)
```

(1) Spark Jobs

df_where: pyspark.sql.dataframe.DataFrame = [ID: integer, E_Name: string ... 2 more fields]

Table

	ID	E_Name	E_City	E_Phonenumber
1	3	John	Montreal	2499793456
2	4	John	Vancouver	2499796785

We can use filter function to get only specified data, can filter unwanted data for this data frame

Sorting Data (ORDER BY)

df_order= df.orderBy(df.E_Name)
display(df_order)

Just now (1s) 26 Python

```
df_order= df.orderBy(df.E_Name)
display(df_order)
```

(1) Spark Jobs

df_order: pyspark.sql.dataframe.DataFrame = [ID: integer, E_Name: string ... 2 more fields]

Table

	ID	E_Name	E_City	E_Phonenumber
1	2	Ann	Toronto	2499799087
2	2	Ann	Brampton	2499799087
3	3	John	Montreal	2499793456
4	4	John	Vancouver	2499796785
5	1	Robert	Toronto	2499791376
6	1	Robert	Toronto	2499791376

6 rows | 0.57s runtime Refreshed now

We can use **orderby** to sort the data based on whatever condition pr column we want.