

PySPARK DATAFRAME

IMPORTANT METHODS

SparkSession.builder.getOrCreate(): This initializes or retrieves an existing SparkSession, the main entry point for interacting with Spark. A SparkSession manages the environment and context for executing Spark jobs.

```
from pyspark.sql.session import SparkSession
spark = SparkSession.builder.getOrCreate()
```

Row: This creates row objects with named fields, which can be used to define structured data. For example, `Row(mobile="+1 234 567 8901", home="+1 234 567 8911")` represents a row with fields `mobile` and `home`.

```
from pyspark.sql import Row
```

```
import datetime
users = [
    {
        "id": 1,
        "first_name": "Corrie",
        "last_name": "Van den Oord",
        "email": "cvandenoord0@etsy.com",
        "phone_numbers": Row(mobile="+1 234 567 8901", home="+1 234 567 8911"),
        "courses": [1, 2],
        "is_customer": True,
        "amount_paid": 1000.55,
        "customer_from": datetime.date(2021, 1, 15),
        "last_updated_ts": datetime.datetime(2021, 2, 10, 1, 15, 0)
    },
    {
        "id": 2,
        "first_name": "Nikolaus",
        "last_name": "Brewitt",
        "email": "nbrewitt1@dailymail.co.uk",
        "phone_numbers": Row(mobile="+1 234 567 8923", home="+1 234 567 8934"),
        "courses": [3],
        "is_customer": True,
        "amount_paid": 900.0,
        "customer_from": datetime.date(2021, 2, 14),
        "last_updated_ts": datetime.datetime(2021, 2, 18, 3, 33, 0)
    },
    {
        "id": 3,
        "first_name": "Orelie",
        "last_name": "Penney",
        "email": "openney2@vistaprint.com",
```

```

"phone_numbers": Row(mobile="+1 714 512 9752", home="+1 714 512 6601"),
"courses": [2, 4],
"is_customer": True,
"amount_paid": 850.55,
"customer_from": datetime.date(2021, 1, 21),
"last_updated_ts": datetime.datetime(2021, 3, 15, 15, 16, 55)
},
{
  "id": 4,
  "first_name": "Ashby",
  "last_name": "Maddocks",
  "email": "amaddocks3@home.pl",
  "phone_numbers": Row(mobile=None, home=None),
  "courses": [],
  "is_customer": False,
  "amount_paid": None,
  "customer_from": None,
  "last_updated_ts": datetime.datetime(2021, 4, 10, 17, 45, 30)
},
{
  "id": 5,
  "first_name": "Kurt",
  "last_name": "Rome",
  "email": "krome4@shutterfly.com",
  "phone_numbers": Row(mobile="+1 817 934 7142", home=None),
  "courses": [],
  "is_customer": False,
  "amount_paid": None,
  "customer_from": None,
  "last_updated_ts": datetime.datetime(2021, 4, 2, 0, 55, 28)
}
]

```

```

import pandas as pd
pd.DataFrame(users)

```

	id	first_name	last_name	email	phone_numbers	courses	is_customer	amount_paid	customer_from	last_updated_ts
0	1	Corrie	Van den Oord	cvandenoord0@etsy.com	(+1 234 567 8901, +1 234 567 8911)	[1, 2]	True	1000.55	2021-01-15	2021-02-10 01:15:00
1	2	Nikolaus	Brewitt	nbrewitt1@dailymail.co.uk	(+1 234 567 8923, 1 234 567 8934)	[3]	True	900.00	2021-02-14	2021-02-18 03:33:00
2	3	Orelle	Penney	openney2@vistaprint.com	(+1 714 512 9752, +1 714 512 6601)	[2, 4]	True	850.55	2021-01-21	2021-03-15 15:16:55
3	4	Ashby	Maddocks	amaddocks3@home.pl	(None, None)	[]	False	NaN	None	2021-04-10 17:45:30
4	5	Kurt	Rome	krome4@shutterfly.com	(+1 817 934 7142, None)	[]	False	NaN	None	2021-04-02 00:55:28

createDataFrame: Converts various data sources (like lists, dictionaries, or Pandas DataFrames) into a Spark DataFrame, allowing distributed data manipulation.

```
users_df = spark.createDataFrame(pd.DataFrame(users))
```

show: Displays a specified number of rows from a DataFrame in the console. The **truncate** option controls whether long strings are shortened.

```
users_df.show(5)
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id|first_name| last_name| email| phone_numbers|courses|is_customer|amount_paid|customer_from| last_updated_ts|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1| Corrie|Van den Oord|cvandenoord0@etsy...|{+1 234 567 8901,...}| [1, 2]| true| 1000.55| 2021-01-15|2021-02-10 01:15:00|
| 2| Nikolaus| Brewitt|nbrewitt1@dailyma...|{+1 234 567 8923,...}| [3]| true| 900.0| 2021-02-14|2021-02-18 03:33:00|
| 3| Orelie| Penney|openney2@vistapri...|{+1 714 512 9752,...}| [2, 4]| true| 850.55| 2021-01-21|2021-03-15 15:16:55|
| 4| Ashby| Maddocks| amaddocks3@home.pl| {NULL, NULL}| []| false| NaN| NULL|2021-04-10 17:45:30|
| 5| Kurt| Rome|krome4@shutterfly...|{+1 817 934 7142,...}| []| false| NaN| NULL|2021-04-02 00:55:28|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
users_df.show(5,truncate=False)
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|id|first_name|last_name| email| phone_numbers|courses|is_customer|amount_paid|customer_from|last_updated_ts|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|1|Corrie|Van den Oord|cvandenoord0@etsy.com|{+1 234 567 8901, +1 234 567 8911}|[1, 2]|true|1000.55|2021-01-15|2021-02-10 01:15:00|
|2|Nikolaus|Brewitt|nbrewitt1@dailymail.co.uk|{+1 234 567 8923, 1 234 567 8934}|[3]|true|900.0|2021-02-14|2021-02-18 03:33:00|
|3|Orelie|Penney|openney2@vistaprint.com|{+1 714 512 9752, +1 714 512 6601}|[2, 4]|true|850.55|2021-01-21|2021-03-15 15:16:55|
|4|Ashby|Maddocks|amaddocks3@home.pl|{NULL, NULL}|[]|false|NaN|NULL|2021-04-10 17:45:30|
|5|Kurt|Rome|krome4@shutterfly.com|{+1 817 934 7142, NULL}|[]|false|NaN|NULL|2021-04-02 00:55:28|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

printSchema: Outputs the schema of a DataFrame, showing column names, data types, and whether null values are allowed, giving insight into the data's structure.

```
users_df.printSchema()
```

```
root
|-- id: long (nullable = true)
|-- first_name: string (nullable = true)
|-- last_name: string (nullable = true)
|-- email: string (nullable = true)
|-- phone_numbers: struct (nullable = true)
|   |-- mobile: string (nullable = true)
|   |-- home: string (nullable = true)
|-- courses: array (nullable = true)
|   |-- element: long (containsNull = true)
|-- is_customer: boolean (nullable = true)
|-- amount_paid: double (nullable = true)
|-- customer_from: date (nullable = true)
|-- last_updated_ts: timestamp (nullable = true)
```

filter: Filters DataFrame rows based on given conditions. It's similar to SQL's **WHERE** clause. For instance, **filter(users_df['amount_paid'] > 900)** only returns rows where **amount_paid** is greater than 900.

```
users_df.filter('amount_paid=900').show()
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id|first_name|last_name| email| phone_numbers|courses|is_customer|amount_paid|customer_from| last_updated_ts|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 2| Nikolaus| Brewitt|nbrewitt1@dailyma...|{+1 234 567 8923,...}| [3]| true| 900.0| 2021-02-14|2021-02-18 03:33:00|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

col: Refers to a specific column in the DataFrame, often used in expressions or for applying transformations.

```
from pyspark.sql.functions import col
type(col('amount_paid')),type('amount_paid')
```

```
↳ (pyspark.sql.column.Column, str)
```

isNotNull: Checks if values in a column are not null. Useful for filtering out null values, e.g., `(col('amount_paid').isNotNull())`.

```
users_df.filter(
    (users_df['amount_paid']>900) &
    (col('amount_paid').isNotNull())
).show()
```

```
↳ +-----+-----+-----+-----+-----+-----+-----+-----+
| id|first_name| last_name| email| phone_numbers|courses|is_customer|amount_paid|customer_from| last_updated_ts|
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1| Corrie|Van den Oord|cvandenoord0@etsy...|{+1 234 567 8901,...| [1, 2]| true| 1000.55| 2021-01-15|2021-02-10 01:15:00|
| 4| Ashby| Maddocks| amaddocks3@home.pl| {NULL, NULL}| []| false| NaN| NULL|2021-04-10 17:45:30|
| 5| Kurt| Rome|krome4@shutterfly...|{+1 817 934 7142,...| []| false| NaN| NULL|2021-04-02 00:55:28|
+-----+-----+-----+-----+-----+-----+-----+-----+
```

isnan: Identifies NaN (Not a Number) values in a column. When used with `~`, it can exclude rows with NaN values, often useful in filtering numerical columns.

```
from pyspark.sql.functions import isnan
```

```
users_df.filter(
    (users_df['amount_paid']>900) &
    (~ isnan(col('amount_paid'))))
).show()
```

```
↳ +-----+-----+-----+-----+-----+-----+-----+-----+
| id|first_name| last_name| email| phone_numbers|courses|is_customer|amount_paid|customer_from| last_updated_ts|
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1| Corrie|Van den Oord|cvandenoord0@etsy...|{+1 234 567 8901,...| [1, 2]| true| 1000.55| 2021-01-15|2021-02-10 01:15:00|
+-----+-----+-----+-----+-----+-----+-----+-----+
```

select: Selects specific columns from a DataFrame to work with, similar to a SQL **SELECT** statement. This can also include expressions or transformations on columns.

```
users_df.select('first_name','last_name','amount_paid').show()
```

```
↳ +-----+-----+-----+
|first_name| last_name|amount_paid|
+-----+-----+-----+
| Corrie|Van den Oord| 1000.55|
| Nikolaus| Brewitt| 900.0|
| Orelie| Penney| 850.55|
| Ashby| Maddocks| NaN|
| Kurt| Rome| NaN|
+-----+-----+-----+
```

fillna: Replaces null values with specified defaults for selected columns, providing a way to handle missing data. For example, `fillna(0, subset=['amount_paid'])` replaces nulls in `amount_paid` with 0.

```
users_df.fillna(0).show()
```

id	first_name	last_name	email	phone_numbers	courses	is_customer	amount_paid	customer_from	last_updated_ts
1	Corrie	Van den Oord	cvandenoord@etsy...	{+1 234 567 8901,...}	[1, 2]	true	1000.55	2021-01-15	2021-02-10 01:15:00
2	Nikolaus	Brewitt	nbrewitt1@dailyma...	{+1 234 567 8923,...}	[3]	true	900.0	2021-02-14	2021-02-18 03:33:00
3	Orelie	Penney	openney2@vistapri...	{+1 714 512 9752,...}	[2, 4]	true	850.55	2021-01-21	2021-03-15 15:16:55
4	Ashby	Maddocks	amaddocks3@home.pl	{NULL, NULL}	[]	false	0.0	NULL	2021-04-10 17:45:30
5	Kurt	Rome	krome4@shutterfly...	{+1 817 934 7142,...}	[]	false	0.0	NULL	2021-04-02 00:55:28

```
users_df_new=users_df.fillna(0,subset=['amount_paid'])
```

```
users_df_new.show()
```

	id	first_name	last_name	email	phone_numbers	courses	is_customer	amount_paid	customer_from	last_updated_ts
1	Corrie	Van den Oord	cvandenoord@etsy...	{+1 234 567 8901, ...}	[1, 2]	true	1000.55	2021-01-15	2021-02-10 01:15:00	
2	Nikolaus	Brewitt	nbrewitt1@daily...	{+1 234 567 8923, ...}	[3]	true	900.0	2021-02-14	2021-02-18 03:33:00	
3	Orelie	Penney	openney2@vistapri...	{+1 714 512 9752, ...}	[2, 4]	true	850.55	2021-01-21	2021-03-15 15:16:55	
4	Ashby	Maddocks	amaddocks3@home.pl	{NULL, NULL}	[]	false	0.0	NULL	2021-04-10 17:45:30	
5	Kurt	Rome	kromez4@shutterfly...	{+1 817 934 7142, ...}	[]	false	0.0	NULL	2021-04-02 00:55:28	

Overview of Narrow and Wide Transformations

Let us get an overview of Narrow and Wide Transformations.

Here are the functions related to narrow transformations. Narrow transformations doesn't result in shuffling. These are also known as row level transformations.

- `df.select`
- `df.filter`
- `df.withColumn`
- `df.withColumnRenamed`
- `df.drop`

Here are the functions related to wide transformations.

- `df.distinct`
- `df.union` or any set operation
- `df.join` or any join operation
- `df.groupBy`
- `df.sort` or `df.orderBy`

Any function that results in shuffling is a wide transformation. For all the wide transformations, we have to deal with a group of records based on a key.

alias: Renames a DataFrame or column temporarily, useful for self-joins or chaining operations without permanently renaming.

```
users_df.alias('u').select('u.*').show()
```

id	first_name	last_name	email	phone_numbers	courses	is_customer	amount_paid	customer_from	last_updated_ts
1	Corrie	Van den Oord	cvandenoord@etsy...	{+1 234 567 8901,...}	[1, 2]	true	1000.55	2021-01-15	2021-02-10 01:15:00
2	Nikolaus	Brewitt	nbrewitt1@dailyma...	{+1 234 567 8923,...}	[3]	true	900.0	2021-02-14	2021-02-18 03:33:00
3	Orelie	Penney	openney2@vistapri...	{+1 714 512 9752,...}	[2, 4]	true	850.55	2021-01-21	2021-03-15 15:16:55
4	Ashby	Maddocks	amaddocks3@home.pl	{NULL, NULL}	[]	false	NaN	NULL	2021-04-10 17:45:30
5	Kurt	Rome	krome4@shutterfly...	{+1 817 934 7142,...}	[]	false	NaN	NULL	2021-04-02 00:55:28

concat: Concatenates multiple columns or literals to form a new column. Commonly used to combine **first_name** and **last_name** into a full name.

```
from pyspark.sql.functions import col, concat, lit

users_df.select(
    col('id'),
    'first_name',
    'last_name',
    concat('first_name', lit(', '), 'last_name').alias('full_name')
).show()
```

```

+---+-----+-----+-----+
| id|first_name| last_name| full_name|
+---+-----+-----+-----+
| 1| Corrie|Van den Oord|Corrie, Van den Oord|
| 2| Nikolaus| Brewitt| Nikolaus, Brewitt|
| 3| Orelie| Penney| Orelie, Penney|
| 4| Ashby| Maddocks| Ashby, Maddocks|
| 5| Kurt| Rome| Kurt, Rome|
+---+-----+-----+-----+

```

lit: Creates a constant column (literal) that can be used within expressions. For instance, **lit(', ')** is used to add a comma separator between concatenated names.

```
users_df.\
withColumn(
    'full_name',
    concat(col('first_name'), lit(', '), col('last_name'))
).show()
```

```

+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id|first_name| last_name| email| phone_numbers|courses|is_customer|amount_paid|customer_from| last_updated_ts| full_name|
+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1| Corrie|Van den Oord|cvandenoord@etsy...|{+1 234 567 8901,...}| [1, 2]| true| 1000.55| 2021-01-15|2021-02-10 01:15:00|Corrie, Van den Oord|
| 2| Nikolaus| Brewitt|nbrewitt1@dailyma...|{+1 234 567 8923,...}| [3]| true| 900.0| 2021-02-14|2021-02-18 03:33:00| Nikolaus, Brewitt|
| 3| Orelie| Penney|openney2@vistapri...|{+1 714 512 9752,...}| [2, 4]| true| 850.55| 2021-01-21|2021-03-15 15:16:55| Orelie, Penney|
| 4| Ashby| Maddocks| amaddocks3@home.pl| {NULL, NULL}| []| false| NaN| NULL|2021-04-10 17:45:30| Ashby, Maddocks|
| 5| Kurt| Rome|krome4@shutterfly...|{+1 817 934 7142,...}| []| false| NaN| NULL|2021-04-02 00:55:28| Kurt, Rome|
+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

withColumn: Adds a new column or replaces an existing one in the DataFrame. It can take an expression, enabling you to perform transformations. For example, **withColumn('full_name', concat(...))** adds a **full_name** column.

selectExpr: Allows SQL-style expressions within a **select** statement. For example, **selectExpr('concat(first_name, ", ", last_name) AS full_name')** creates a new column directly from an SQL expression.

```
users_df.selectExpr('id', 'first_name', 'last_name', 'concat(first_name, ", ", last_name) AS
full_name').show()
```

```

+---+-----+-----+-----+
| id|first_name| last_name| full_name|
+---+-----+-----+-----+
| 1| Corrie|Van den Oord|Corrie, Van den Oord|
| 2| Nikolaus| Brewitt| Nikolaus, Brewitt|
| 3| Orelie| Penney| Orelie, Penney|
| 4| Ashby| Maddocks| Ashby, Maddocks|
| 5| Kurt| Rome| Kurt, Rome|
+---+-----+-----+-----+

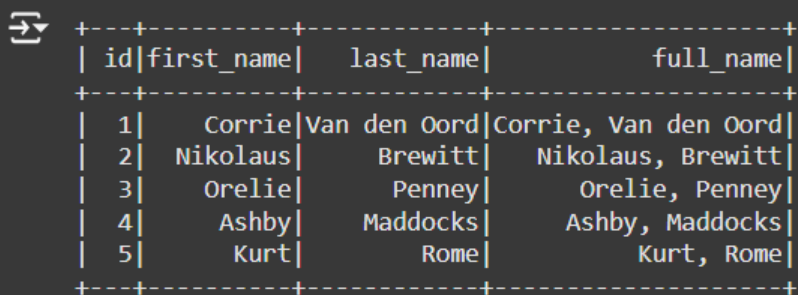
```

createOrReplaceTempView: Registers a DataFrame as a temporary SQL table. This allows you to query it using Spark SQL syntax within the SparkSession's SQL context.

```
users_df.createOrReplaceTempView('users')
```

spark.sql(): Executes a SQL query on DataFrames that have been registered as temporary views or tables within the SparkSession. This allows you to use SQL syntax for querying and manipulating DataFrames.

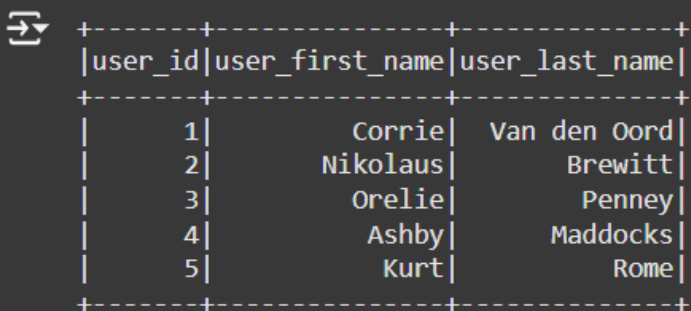
```
spark.sql("""
  SELECT id, first_name, last_name,
         concat(first_name, ' ', last_name) AS full_name
  FROM users
""").\
  show()
```



id	first_name	last_name	full_name
1	Corrie	Van den Oord	Corrie, Van den Oord
2	Nikolaus	Brewitt	Nikolaus, Brewitt
3	Orelie	Penney	Orelie, Penney
4	Ashby	Maddocks	Ashby, Maddocks
5	Kurt	Rome	Kurt, Rome

withColumnRenamed: Renames a single column in a DataFrame, allowing clearer names for specific use cases or to avoid column name conflicts in joins.

```
users_df.\
  select('id','first_name','last_name').\
  withColumnRenamed('id','user_id').\
  withColumnRenamed('first_name','user_first_name').\
  withColumnRenamed('last_name','user_last_name').\
  show()
```



user_id	user_first_name	user_last_name
1	Corrie	Van den Oord
2	Nikolaus	Brewitt
3	Orelie	Penney
4	Ashby	Maddocks
5	Kurt	Rome

read.csv: Reads a CSV file and loads it into a DataFrame. The **header=True** option treats the first line as column names, and the **schema** parameter can enforce a specific structure.

```
spark.read.csv(path='Customers.csv').show()
```


	_c0	_c1	_c2	_c3	_c4	_c5	_c6	_c7	_c8	_c9
	CustomerKey	Gender	Name	City	State Code	State	Zip Code	Country	Continent	Birthday
	301	Female	Lilly Harding	WANDEARAH EAST	SA	South Australia	5523	Australia	Australia	7/3/1939
	325	Female	Madison Hull	MOUNT BUDD	WA	Western Australia	6522	Australia	Australia	9/27/1979
	554	Female	Claire Ferres	WINJALLOK	VIC	Victoria	3380	Australia	Australia	5/26/1947
	786	Male	Jai Poltpalingada	MIDDLE RIVER	SA	South Australia	5223	Australia	Australia	9/17/1957
	1042	Male	Aidan Pankhurst	TAWONGA SOUTH	VIC	Victoria	3698	Australia	Australia	11/19/1965
	1086	Male	Hayden Clegg	TEMPLERS	SA	South Australia	5371	Australia	Australia	1/20/1954
	1133	Male	Nicholas Caffyn	JUBILEE POCKET	QLD	Queensland	4802	Australia	Australia	11/22/1969
	1256	Male	Lincoln Jenks	KULLOGUM	QLD	Queensland	4660	Australia	Australia	3/12/1950
	1314	Male	Isaac Israel	EDITH RIVER	NT	Northern Territory	852	Australia	Australia	12/21/1965
	1568	Male	Luke Virtue	KOTTA	VIC	Victoria	3565	Australia	Australia	7/25/1975
	1585	Female	Anna Hallstrom	GREEN LAKE	VIC	Victoria	3401	Australia	Australia	8/12/1990
	1626	Female	Mary Leach	NEDS CORNER	VIC	Victoria	3496	Australia	Australia	10/17/1985
	1642	Female	Ruby Ambrose	TARLO	NSW	New South Wales	2580	Australia	Australia	3/3/1965
	1677	Male	Dean Innes	ELLESMERE	QLD	Queensland	4610	Australia	Australia	11/22/1964
	1817	Female	Laura Le Rennetel	BEAUCHAMP	VIC	Victoria	3579	Australia	Australia	7/10/1986
	1863	Female	Chelsea Watkins	NAMBOUR DC	QLD	Queensland	4560	Australia	Australia	9/22/1984
	1945	Male	Angus Beaurepaire	GREENFIELDS	WA	Western Australia	6210	Australia	Australia	9/20/1974
	2238	Female	Isabelle Cochran	BRUCKNELL	VIC	Victoria	3268	Australia	Australia	10/31/1969
	2248	Male	Rory Spargo	JUNG	VIC	Victoria	3401	Australia	Australia	12/15/1987

```
customers_df=spark.read.csv(path='Customers.csv',header=True)
customers_df.show()
```

	CustomerKey	Gender	Name	City	State Code	State	Zip Code	Country	Continent	Birthday
	301	Female	Lilly Harding	WANDEARAH EAST	SA	South Australia	5523	Australia	Australia	7/3/1939
	325	Female	Madison Hull	MOUNT BUDD	WA	Western Australia	6522	Australia	Australia	9/27/1979
	554	Female	Claire Ferres	WINJALLOK	VIC	Victoria	3380	Australia	Australia	5/26/1947
	786	Male	Jai Poltpalingada	MIDDLE RIVER	SA	South Australia	5223	Australia	Australia	9/17/1957
	1042	Male	Aidan Pankhurst	TAWONGA SOUTH	VIC	Victoria	3698	Australia	Australia	11/19/1965
	1086	Male	Hayden Clegg	TEMPLERS	SA	South Australia	5371	Australia	Australia	1/20/1954
	1133	Male	Nicholas Caffyn	JUBILEE POCKET	QLD	Queensland	4802	Australia	Australia	11/22/1969
	1256	Male	Lincoln Jenks	KULLOGUM	QLD	Queensland	4660	Australia	Australia	3/12/1950
	1314	Male	Isaac Israel	EDITH RIVER	NT	Northern Territory	852	Australia	Australia	12/21/1965
	1568	Male	Luke Virtue	KOTTA	VIC	Victoria	3565	Australia	Australia	7/25/1975
	1585	Female	Anna Hallstrom	GREEN LAKE	VIC	Victoria	3401	Australia	Australia	8/12/1990
	1626	Female	Mary Leach	NEDS CORNER	VIC	Victoria	3496	Australia	Australia	10/17/1985
	1642	Female	Ruby Ambrose	TARLO	NSW	New South Wales	2580	Australia	Australia	3/3/1965
	1677	Male	Dean Innes	ELLESMERE	QLD	Queensland	4610	Australia	Australia	11/22/1964
	1817	Female	Laura Le Rennetel	BEAUCHAMP	VIC	Victoria	3579	Australia	Australia	7/10/1986
	1863	Female	Chelsea Watkins	NAMBOUR DC	QLD	Queensland	4560	Australia	Australia	9/22/1984
	1945	Male	Angus Beaurepaire	GREENFIELDS	WA	Western Australia	6210	Australia	Australia	9/20/1974
	2238	Female	Isabelle Cochran	BRUCKNELL	VIC	Victoria	3268	Australia	Australia	10/31/1969
	2248	Male	Rory Spargo	JUNG	VIC	Victoria	3401	Australia	Australia	12/15/1987
	2435	Female	Lilian Hall	BLAKEBROOK	NSW	New South Wales	2480	Australia	Australia	4/1/1968

only showing top 20 rows

StructType and **StructField**: Define a schema explicitly, with each field given a name and data type. This is useful when data files lack a header or need more complex structures.

```
from pyspark.sql.types import StructType, StructField, StringType, IntegerType
```

```
# Define the schema
schema = StructType([
    StructField("CustomerKey", IntegerType()),
    StructField("Gender", StringType()),
    StructField("Name", StringType()),
    StructField("City", StringType()),
    StructField("State Code", StringType()),
    StructField("State", StringType()),
    StructField("Zip Code", StringType()),
    StructField("Country", StringType()),
    StructField("Continent", StringType()),
```



```
StructField("Birthday", StringType())
])
```



```
spark.read.csv(path='Customers.csv',header=True,schema=schema).printSchema()
```



```
root
|-- CustomerKey: integer (nullable = true)
|-- Gender: string (nullable = true)
|-- Name: string (nullable = true)
|-- City: string (nullable = true)
|-- State Code: string (nullable = true)
|-- State: string (nullable = true)
|-- Zip Code: string (nullable = true)
|-- Country: string (nullable = true)
|-- Continent: string (nullable = true)
|-- Birthday: string (nullable = true)
```

cast: Changes the data type of a column. For example, converting a **StringType** date column into a **DateType** allows for easier date manipulation.

```
customers_df=customers_df.withColumn('Birthday_1',col('Birthday').cast('date'))
```

to_date: Converts a string column to a date column using a specified format. This function helps ensure that dates are consistently formatted and recognized.

```
from pyspark.sql.functions import to_date
customers_df=customers_df.withColumn('Birthday_1',to_date(col('Birthday'),'M/d/yyyy'))
customers_df.show()
```



customer_id	Gender	Name	City	State Code	State	Zip Code	Country	Continent	Birthday	Birthday_1
301	Female	Lilly Harding	WANDEARAH EAST	SA	South Australia	5523	Australia	Australia	7/3/1939	1939-07-03
325	Female	Madison Hull	MOUNT BUDD	WA	Western Australia	6522	Australia	Australia	9/27/1979	1979-09-27
554	Female	Claire Ferres	WINDJALLOK	VIC	Victoria	3380	Australia	Australia	5/26/1947	1947-05-26
786	Male	Jai Poltpalingada	MIDDLE RIVER	SA	South Australia	5223	Australia	Australia	9/17/1957	1957-09-17
1042	Male	Aidan Pankhurst	TAWONGA SOUTH	VIC	Victoria	3698	Australia	Australia	11/19/1965	1965-11-19
1086	Male	Hayden Clegg	TEMPLERS	SA	South Australia	5371	Australia	Australia	1/20/1954	1954-01-20
1133	Male	Nicholas Caffyn	JUBILEE POCKET	QLD	Queensland	4802	Australia	Australia	11/22/1969	1969-11-22
1256	Male	Lincoln Jenks	KULLOGUM	QLD	Queensland	4660	Australia	Australia	3/12/1950	1950-03-12
1314	Male	Isaac Israel	EDITH RIVER	NT	Northern Territory	852	Australia	Australia	12/21/1965	1965-12-21
1568	Male	Luke Virtue	KOTTA	VIC	Victoria	3565	Australia	Australia	7/25/1975	1975-07-25
1585	Female	Anna Hallstrom	GREEN LAKE	VIC	Victoria	3401	Australia	Australia	8/12/1990	1990-08-12
1626	Female	Mary Leach	NEDS CORNER	VIC	Victoria	3496	Australia	Australia	10/17/1985	1985-10-17
1642	Female	Ruby Ambrose	TARLO	NSW	New South Wales	2580	Australia	Australia	3/3/1965	1965-03-03
1677	Male	Dean Innes	ELLESMERE	QLD	Queensland	4610	Australia	Australia	11/22/1964	1964-11-22
1817	Female	Laura Le Rennetel	BEAUCHAMP	VIC	Victoria	3579	Australia	Australia	7/10/1986	1986-07-10
1863	Female	Chelsea Watkins	NAMBOUR DC	QLD	Queensland	4560	Australia	Australia	9/22/1984	1984-09-22
1945	Male	Angus Beaupaire	GREENFIELDS	WA	Western Australia	6210	Australia	Australia	9/20/1974	1974-09-20
2238	Female	Isabelle Cochran	BRUCKNELL	VIC	Victoria	3268	Australia	Australia	10/31/1969	1969-10-31
2248	Male	Rory Spargo	JUNG	VIC	Victoria	3401	Australia	Australia	12/15/1987	1987-12-15
2435	Female	Lilian Hall	BLAKEBROOK	NSW	New South Wales	2480	Australia	Australia	4/1/1968	1968-04-01

only showing top 20 rows

drop: Removes specified columns from a DataFrame, which can simplify the schema if certain columns are no longer needed.

```
customers_df=customers_df.\
drop('Birthday').\
```

```
withColumnRenamed('Birthday_1','Birthday')
```

```
customers_df.show()
```

customer_id	Gender	Name	City	State	Code	State	Zip	Code	Country	Continent	Birthday
301	Female	Lilly Harding	WANDEARAH EAST	SA	South Australia	5523	Australia	Australia	1939-07-03		
325	Female	Madison Hull	MOUNT BUDD	WA	Western Australia	6522	Australia	Australia	1979-09-27		
554	Female	Claire Ferres	WINJALLOK	VIC	Victoria	3380	Australia	Australia	1947-05-26		
786	Male	Jai Poltpalingada	MIDDLE RIVER	SA	South Australia	5223	Australia	Australia	1957-09-17		
1042	Male	Aidan Pankhurst	TAWONGA SOUTH	VIC	Victoria	3698	Australia	Australia	1965-11-19		
1086	Male	Hayden Clegg	TEMPLERS	SA	South Australia	5371	Australia	Australia	1954-01-20		
1133	Male	Nicholas Caffyn	JUBILEE POCKET	QLD	Queensland	4802	Australia	Australia	1969-11-22		
1256	Male	Lincoln Jenks	KULLOGUM	QLD	Queensland	4660	Australia	Australia	1950-03-12		
1314	Male	Isaac Israel	EDITH RIVER	NT	Northern Territory	852	Australia	Australia	1965-12-21		
1568	Male	Luke Virtue	KOTTA	VIC	Victoria	3565	Australia	Australia	1975-07-25		
1585	Female	Anna Hallstrom	GREEN LAKE	VIC	Victoria	3401	Australia	Australia	1990-08-12		
1626	Female	Mary Leach	NEDS CORNER	VIC	Victoria	3496	Australia	Australia	1985-10-17		
1642	Female	Ruby Ambrose	TARLO	NSW	New South Wales	2580	Australia	Australia	1965-03-03		
1677	Male	Dean Innes	ELLESMERE	QLD	Queensland	4610	Australia	Australia	1964-11-22		
1817	Female	Laura Le Renne	BEAUCHAMP	VIC	Victoria	3579	Australia	Australia	1986-07-10		
1863	Female	Chelsea Watkins	NAMBOUR DC	QLD	Queensland	4560	Australia	Australia	1984-09-22		
1945	Male	Angus Beaurepaire	GREENFIELDS	WA	Western Australia	6210	Australia	Australia	1974-09-20		
2238	Female	Isabelle Cochran	BRUCKNELL	VIC	Victoria	3268	Australia	Australia	1969-10-31		
2248	Male	Rory Spargo	JUNG	VIC	Victoria	3401	Australia	Australia	1987-12-15		
2435	Female	Lilian Hall	BLAKEBROOK	NSW	New South Wales	2480	Australia	Australia	1968-04-01		

only showing top 20 rows

year, month, date_format: Extract date components like year and month, or format a date to show specific information like the day of the week (e.g., **date_format('Birthday', 'EEEE')** extracts the weekday name).

```
from pyspark.sql.functions import year,month,date_format
```

```
customers_df=customers_df.withColumns({'year':year('Birthday'),'month':month('Birthday'),'week_name':date_format('Birthday','EEEE')})
```

```
customers_df.show()
```

customer_id	Gender	Name	City	State	Code	State	Zip	Code	Country	Continent	Birthday	year	month	week_name
301	Female	Lilly Harding	WANDEARAH EAST	SA	South Australia	5523	Australia	Australia	1939-07-03	1939	7	Monday		
325	Female	Madison Hull	MOUNT BUDD	WA	Western Australia	6522	Australia	Australia	1979-09-27	1979	9	Thursday		
554	Female	Claire Ferres	WINJALLOK	VIC	Victoria	3380	Australia	Australia	1947-05-26	1947	5	Monday		
786	Male	Jai Poltpalingada	MIDDLE RIVER	SA	South Australia	5223	Australia	Australia	1957-09-17	1957	9	Tuesday		
1042	Male	Aidan Pankhurst	TAWONGA SOUTH	VIC	Victoria	3698	Australia	Australia	1965-11-19	1965	11	Friday		
1086	Male	Hayden Clegg	TEMPLERS	SA	South Australia	5371	Australia	Australia	1954-01-20	1954	1	Wednesday		
1133	Male	Nicholas Caffyn	JUBILEE POCKET	QLD	Queensland	4802	Australia	Australia	1969-11-22	1969	11	Saturday		
1256	Male	Lincoln Jenks	KULLOGUM	QLD	Queensland	4660	Australia	Australia	1950-03-12	1950	3	Sunday		
1314	Male	Isaac Israel	EDITH RIVER	NT	Northern Territory	852	Australia	Australia	1965-12-21	1965	12	Tuesday		
1568	Male	Luke Virtue	KOTTA	VIC	Victoria	3565	Australia	Australia	1975-07-25	1975	7	Friday		
1585	Female	Anna Hallstrom	GREEN LAKE	VIC	Victoria	3401	Australia	Australia	1990-08-12	1990	8	Sunday		
1626	Female	Mary Leach	NEDS CORNER	VIC	Victoria	3496	Australia	Australia	1985-10-17	1985	10	Thursday		
1642	Female	Ruby Ambrose	TARLO	NSW	New South Wales	2580	Australia	Australia	1965-03-03	1965	3	Wednesday		
1677	Male	Dean Innes	ELLESMERE	QLD	Queensland	4610	Australia	Australia	1964-11-22	1964	11	Sunday		
1817	Female	Laura Le Renne	BEAUCHAMP	VIC	Victoria	3579	Australia	Australia	1986-07-10	1986	7	Thursday		
1863	Female	Chelsea Watkins	NAMBOUR DC	QLD	Queensland	4560	Australia	Australia	1984-09-22	1984	9	Saturday		
1945	Male	Angus Beaurepaire	GREENFIELDS	WA	Western Australia	6210	Australia	Australia	1974-09-20	1974	9	Friday		
2238	Female	Isabelle Cochran	BRUCKNELL	VIC	Victoria	3268	Australia	Australia	1969-10-31	1969	10	Friday		
2248	Male	Rory Spargo	JUNG	VIC	Victoria	3401	Australia	Australia	1987-12-15	1987	12	Tuesday		
2435	Female	Lilian Hall	BLAKEBROOK	NSW	New South Wales	2480	Australia	Australia	1968-04-01	1968	4	Monday		

only showing top 20 rows

distinct: Returns a new DataFrame with only unique rows, removing duplicates, which is useful for checking unique values in specific columns.

```
customers_df.select('state').distinct().show()
```

```

+-----+
| state |
+-----+
| Palermo |
| Worcester |
| Utah |
| Charnwood |
| North Kesteven |
| Hawaii |
| Manitoba |
| Arun |
| Waveney |
| Firenze |
| Stroud |
| Pays de la Loire |
| Nuneaton & Bedworth |
| New Forest |
| Newmarket |
| Haute-Normandie |
| Guildford |
| Minnesota |
| Matera |
| Central Bedfordshire |
+-----+
only showing top 20 rows

```

```
customers_df.select('state').distinct().count()
```

```

⇄ 512

```

orderBy: Sorts the DataFrame by specified columns, in ascending order by default. Sorting enables easier analysis of ordered data, like dates or numerical columns.

```
customers_df.select('year').distinct().orderBy('year').show()
```

```
customers_df.select('year').distinct().orderBy('year', ascending=False).show()
```

```

⇄ +----+
| year |
+----+
| 1935 |
| 1936 |
| 1937 |
| 1938 |
| 1939 |
| 1940 |
| 1941 |
| 1942 |
| 1943 |
| 1944 |
| 1945 |
| 1946 |
| 1947 |
| 1948 |
| 1949 |
| 1950 |
| 1951 |
| 1952 |
| 1953 |
| 1954 |
+----+
only showing top 20 rows

```


```

⇄ +----+
| year |
+----+
| 2002 |
| 2001 |
| 2000 |
| 1999 |
| 1998 |
| 1997 |
| 1996 |
| 1995 |
| 1994 |
| 1993 |
| 1992 |
| 1991 |
| 1990 |
| 1989 |
| 1988 |
| 1987 |
| 1986 |
| 1985 |
| 1984 |
| 1983 |
+----+
only showing top 20 rows

```

groupBy: Groups the DataFrame by one or more columns, commonly used with aggregation functions (e.g., **count**) for data summarization, similar to SQL **GROUP BY**.

```
customers_df.groupBy('year').count().show()
```




```

+----+-----+
|year|count|
+----+-----+
|1959| 243|
|1990| 221|
|1975| 217|
|1977| 247|
|1974| 212|
|1955| 235|
|1978| 217|
|1961| 251|
|1942| 212|
|1939| 212|
|1944| 216|
|1952| 212|
|1956| 235|
|1997| 218|
|1988| 213|
|1994| 219|
|1968| 264|
|1951| 241|
|1938| 217|
|1973| 222|
+----+-----+
only showing top 20 rows

```

read.json: Reads a JSON file and loads it into a DataFrame. Specifying **schema** ensures the DataFrame's structure, particularly if the JSON has inconsistent fields.

```
spark.read.json(path='customers.json').show()
```



```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Birthday | City | Continent | Country | Gender | Name | State | State Code | Zip Code | customer_id | month | week_name | year |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1939-07-03 | WANDEARAH EAST | Australia | Australia | Female | Lilly Harding | South Australia | SA | 5523 | 301 | 7 | Monday | 1939 |
| 1979-09-27 | MOUNT BUDD | Australia | Australia | Female | Madison Hull | Western Australia | WA | 6522 | 325 | 9 | Thursday | 1979 |
| 1947-05-26 | WINJALLOK | Australia | Australia | Female | Claire Ferres | Victoria | VIC | 3380 | 554 | 5 | Monday | 1947 |
| 1957-09-17 | MIDDLE RIVER | Australia | Australia | Male | Jai Poltpalingada | South Australia | SA | 5223 | 786 | 9 | Tuesday | 1957 |
| 1965-11-19 | TAWONGA SOUTH | Australia | Australia | Male | Aidan Pankhurst | Victoria | VIC | 3698 | 1042 | 11 | Friday | 1965 |
| 1954-01-20 | TEMPLERS | Australia | Australia | Male | Hayden Clegg | South Australia | SA | 5371 | 1086 | 1 | Wednesday | 1954 |
| 1969-11-22 | JUBILEE POCKET | Australia | Australia | Male | Nicholas Caffyn | Queensland | QLD | 4802 | 1133 | 11 | Saturday | 1969 |
| 1950-03-12 | KULLOGUM | Australia | Australia | Male | Lincoln Jenks | Queensland | QLD | 4660 | 1256 | 3 | Sunday | 1950 |
| 1965-12-21 | EDITH RIVER | Australia | Australia | Male | Isaac Israel | Northern Territory | NT | 852 | 1314 | 12 | Tuesday | 1965 |
| 1975-07-25 | KOTTA | Australia | Australia | Male | Luke Virtue | Victoria | VIC | 3565 | 1568 | 7 | Friday | 1975 |
| 1990-08-12 | GREEN LAKE | Australia | Australia | Female | Anna Hallstrom | Victoria | VIC | 3401 | 1585 | 8 | Sunday | 1990 |
| 1985-10-17 | NEDS CORNER | Australia | Australia | Female | Mary Leach | Victoria | VIC | 3496 | 1626 | 10 | Thursday | 1985 |
| 1965-03-03 | TARLO | Australia | Australia | Female | Ruby Ambrose | New South Wales | NSW | 2580 | 1642 | 3 | Wednesday | 1965 |
| 1964-11-22 | ELLESMERE | Australia | Australia | Male | Dean Innes | Queensland | QLD | 4610 | 1677 | 11 | Sunday | 1964 |
| 1986-07-10 | BEAUCHAMP | Australia | Australia | Female | Laura Le Renne | Victoria | VIC | 3579 | 1817 | 7 | Thursday | 1986 |
| 1984-09-22 | NAMBOUR DC | Australia | Australia | Female | Chelsea Watkins | Queensland | QLD | 4560 | 1863 | 9 | Saturday | 1984 |
| 1974-09-20 | GREENFIELDS | Australia | Australia | Male | Angus Beaurepaire | Western Australia | WA | 6210 | 1945 | 9 | Friday | 1974 |
| 1969-10-31 | BRUCKNELL | Australia | Australia | Female | Isabelle Cochran | Victoria | VIC | 3268 | 2238 | 10 | Friday | 1969 |
| 1987-12-15 | JUNG | Australia | Australia | Male | Rory Spargo | Victoria | VIC | 3401 | 2248 | 12 | Tuesday | 1987 |
| 1968-04-01 | BLAKEBROOK | Australia | Australia | Female | Lilian Hall | New South Wales | NSW | 2480 | 2435 | 4 | Monday | 1968 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 20 rows

```

```

schema="""customer_id string,
Gender string,
Name string,
City string,
`State Code` string,
State string,
`Zip Code` string,
Country string,
Continent string,
Birthday string
"""

```

```
customers_json_df=spark.read.json(path='customers.json',schema=schema)
```

schema: This is a schema definition in PySpark using a multi-line string. Each line specifies a column name followed by its data type, allowing you to define a structured schema in a convenient format. For instance, `customer_id string` means the `customer_id` column will be treated as a `StringType`. This schema helps ensure data consistency and provides explicit structure when reading data, especially if the JSON file lacks a clear structure or includes fields with inconsistent types.

```
spark.read.json(path='customers.json', schema=schema):
```

- This line reads a JSON file (**customers.json**) into a DataFrame.
- By passing the **schema** parameter, Spark enforces this predefined structure on the JSON data, rather than inferring it automatically. This is particularly useful when you want to control data types or handle inconsistent data.
- Specifying **schema=schema** ensures each column is parsed with the intended data type, reducing potential errors during analysis.

withColumnsRenamed: Allows renaming multiple columns at once using a dictionary mapping of old names to new names, streamlining the process of updating column names.

```
customers_json_df.withColumnsRenamed({'State Code':'State_Code','Zip
Code':'zip_code'}).show()
```

customer_id	Gender	Name	City	State_code	State	zip_code	Country	Continent	Birthday
301	Female	Lilly Harding	WANDERARH EAST	SA	South Australia	5523	Australia	Australia	1939-07-03
325	Female	Madison Hull	MOUNT BUDD	WA	Western Australia	6522	Australia	Australia	1979-09-27
554	Female	Claire Ferres	WINJALLOK	VIC	Victoria	3380	Australia	Australia	1947-05-26
786	Male	Jai Poltpalingada	MIDDLE RIVER	SA	South Australia	5223	Australia	Australia	1957-09-17
1042	Male	Aidan Pankhurst	TAWONGA SOUTH	VIC	Victoria	3698	Australia	Australia	1965-11-19
1086	Male	Hayden Clegg	TEMPLERS	SA	South Australia	5371	Australia	Australia	1954-01-20
1133	Male	Nicholas Caffyn	JUBILEE POCKET	QLD	Queensland	4802	Australia	Australia	1969-11-22
1256	Male	Lincoln Jenks	KULLOGUM	QLD	Queensland	4660	Australia	Australia	1950-03-12
1314	Male	Isaac Israel	EDITH RIVER	NT	Northern Territory	852	Australia	Australia	1965-12-21
1568	Male	Luke Virtue	KOTTA	VIC	Victoria	3565	Australia	Australia	1975-07-25
1585	Female	Anna Hallstrom	GREEN LAKE	VIC	Victoria	3401	Australia	Australia	1990-08-12
1626	Female	Mary Leach	NEDS CORNER	VIC	Victoria	3496	Australia	Australia	1985-10-17
1642	Female	Ruby Ambrose	TARLO	NSW	New South Wales	2580	Australia	Australia	1965-03-03
1677	Male	Dean Innes	ELLESMERE	QLD	Queensland	4610	Australia	Australia	1964-11-22
1817	Female	Laura Le Rennetel	BEAUCHAMP	VIC	Victoria	3579	Australia	Australia	1986-07-10
1863	Female	Chelsea Watkins	NAMBOUR DC	QLD	Queensland	4560	Australia	Australia	1984-09-22
1945	Male	Angus Beaurepaire	GREENFIELDS	WA	Western Australia	6210	Australia	Australia	1974-09-20
2238	Female	Isabelle Cochran	BRUCKNELL	VIC	Victoria	3268	Australia	Australia	1969-10-31
2248	Male	Rory Spargo	JUNG	VIC	Victoria	3401	Australia	Australia	1987-12-15
2435	Female	Lilian Hall	BLAKEBROOK	NSW	New South Wales	2480	Australia	Australia	1968-04-01

SUMMARY

1. **Creating a SparkSession:** This is the entry point for interacting with Spark, allowing us to create DataFrames and execute transformations.
2. **Working with Rows and DataFrames:** Rows provide structured data elements, and DataFrames are distributed collections of these rows, supporting operations similar to SQL tables.
3. **Schema Definition:** We can define explicit schemas using **StructType** or as a multi-line string to specify data types and enforce structure on incoming data, especially useful for files with inconsistent formats (like JSON).
4. **Basic DataFrame Operations:**
 - **select, filter, withColumn:** These functions allow us to select specific columns, filter rows based on conditions, and create or modify columns.
 - **fillna:** Replaces null values, helping in data cleaning.
 - **distinct and groupBy:** Used for de-duplication and aggregating data, enabling easier summarization.
5. **Column Expressions:**
 - **concat, col, lit:** These functions support column manipulations like concatenating strings or adding constant values.
 - **Date Functions:** Functions like **year, month,** and **date_format** help extract specific parts of date columns.
6. **SQL Queries on DataFrames:** Registering DataFrames as temporary views enables us to run SQL queries, combining the power of SQL and PySpark.
7. **Reading External Data:**
 - **read.csv and read.json:** These functions allow us to load CSV and JSON files into DataFrames with optional schema definitions, which can simplify or standardize data loading.

Overall, these operations showcase PySpark's flexibility for data manipulation and analysis, from basic transformations to complex operations involving schemas, SQL queries, and external data sources.