Practical 1

↳ 25 cells hidden

Practical 2

A. Program to demonstrate DataFrame Sorting operations

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
```

```
data = [
    ("James", "sales", "NY", 90000, 34, 10000),
    ("Micheal", "sales", "NY", 86000, 56, 20000),
    ("Robert", "sales", "CA", 81000, 30, 23000),
    ("Maria", "finance", "CA", 90000, 24, 23000),
    ("Jen", "finance", "NY", 79000, 53, 15000),
    ("Jeff", "marketing", "CA", 80000, 25, 18000),
    ("Kumar", "marketing", "NY", 91000, 50, 21000),
    ("Saif", "IT", "CA", 72000, 31, 22000),
    ("Raj", "IT", "NY", 65000, 29, 17000),
    ("Alex", "HR", "CA", 78000, 28, 11000),
    ("Sara", "HR", "NY", 75000, 27, 14000),
    ("Mona", "Legal", "CA", 87000, 45, 12000),
    ("Nina", "Legal", "NY", 93000, 38, 16000),
    ("John", "Operations", "CA", 82000, 33, 21000),
    ("David", "Operations", "NY", 84000, 49, 19000),
]
```

```
columns = ['Name', 'Department', 'location', 'salary', 'Age', 'bonus']

df = spark.createDataFrame(data = data, schema = columns)
df.show()
```

```
+-------+----------+--------+------+---+-----+
|   Name|Department|location|salary|Age|bonus|
+-------+----------+--------+------+---+-----+
|  James|     sales|      NY| 90000| 34|10000|
|Micheal|     sales|      NY| 86000| 56|20000|
| Robert|     sales|      CA| 81000| 30|23000|
|  Maria|   finance|      CA| 90000| 24|23000|
|    Jen|   finance|      NY| 79000| 53|15000|
|   Jeff| marketing|      CA| 80000| 25|18000|
|  Kumar| marketing|      NY| 91000| 50|21000|
|   Saif|        IT|      CA| 72000| 31|22000|
|    Raj|        IT|      NY| 65000| 29|17000|
|   Alex|        HR|      CA| 78000| 28|11000|
|   Sara|        HR|      NY| [illegible]000| 27|14000|
|   Mona|     Legal|      CA| [illegible]000| 45|12000|
|   Nina|     Legal|      NY| 93000| 38|16000|
|   John|Operations|      CA| 82000| 33|21000|
|  David|Operations|      NY| 84000| 49|19000|
+-------+----------+--------+------+---+-----+
```

```
# Sorting with Age
df.sort('Age').show()
df.sort(col('Age').desc()).show()
```

```
+-------+----------+--------+------+---+-----+
|   Name|Department|location|salary|Age|bonus|
+-------+----------+--------+------+---+-----+
|  Maria|   finance|      CA| 90000| 24|23000|
|   Jeff| marketing|      CA| 80000| 25|18000|
|   Sara|        HR|      NY| 75000| 27|14000|
|   Alex|        HR|      CA| 78000| 28|11000|
|    Raj|        IT|      NY| 65000| 29|17000|
| Robert|     sales|      CA| 81000| 30|23000|
|   Saif|        IT|      CA| 72000| 31|22000|
|   John|Operations|      CA| 82000| 33|21000|
|  James|     sales|      NY| 90000| 34|10000|
|   Nina|     Legal|      NY| 93000| 38|16000|
|   Mona|     Legal|      CA| 87000| 45|12000|
|  David|Operations|      NY| 84000| 49|19000|
|  Kumar| marketing|      NY| 91000| 50|21000|
|    Jen|   finance|      NY| 79000| 53|15000|
|Micheal|     sales|      NY| 86000| 56|20000|
+-------+----------+--------+------+---+-----+

+-------+----------+--------+------+---+-----+
|   Name|Department|location|salary|Age|bonus|
+-------+----------+--------+------+---+-----+
|Micheal|     sales|      NY| 86000| 56|20000|
|    Jen|   finance|      NY| 79000| 53|15000|
|  Kumar| marketing|      NY| 91000| 50|21000|
|  David|Operations|      NY| 84000| 49|19000|
|   Mona|     Legal|      CA| 87000| 45|12000|
|   Nina|     Legal|      NY| 93000| 38|16000|
|  James|     sales|      NY| 90000| 34|10000|
|   John|Operations|      CA| 82000| 33|21000|
|   Saif|        IT|      CA| 72000| 31|22000|
| Robert|     sales|      CA| 81000| 30|23000|
|    Raj|        IT|      NY| 65000| 29|17000|
|   Alex|        HR|      CA| 78000| 28|11000|
|   Sara|        HR|      NY| 75000| 27|14000|
|   Jeff| marketing|      CA| 80000| 25|18000|
|  Maria|   finance|      CA| 90000| 24|23000|
+-------+----------+--------+------+---+-----+
```

```
# Sorting with multiple columns
df.sort('Department', 'location').show()
df.sort(col("Department"), col('location')).show()
```

```
+-------+----------+--------+------+---+-----+
|   Name|Department|location|salary|Age|bonus|
+-------+----------+--------+------+---+-----+
|   Alex|        HR|      CA| 78000| 28|11000|
|   Sara|        HR|      NY| 75000| 27|14000|
|   Saif|        IT|      CA| 72000| 31|22000|
|    Raj|        IT|      NY| 65000| 29|17000|
|   Mona|     Legal|      CA| 87000| 45|12000|
|   Nina|     Legal|      NY| 93000| 38|16000|
|   John|Operations|      CA| 82000| 33|21000|
|  David|Operations|      NY| 84000| 49|19000|
|  Maria|   finance|      CA| 90000| 24|23000|
|    Jen|   finance|      NY| 79000| 53|15000|
|   Jeff| marketing|      CA| 80000| 25|18000|
|  Kumar| marketing|      NY| 91000| 50|21000|
| Robert|     sales|      CA| 81000| 30|23000|
|  James|     sales|      NY| 90000| 34|10000|
|Micheal|     sales|      NY| 86000| 56|20000|
+-------+----------+--------+------+---+-----+

+-------+----------+--------+------+---+-----+
|   Name|Department|location|salary|Age|bonus|
+-------+----------+--------+------+---+-----+
|   Alex|        HR|      CA| 78000| 28|11000|
|   Sara|        HR|      NY| 75000| 27|14000|
|   Saif|        IT|      CA| 72000| 31|22000|
|    Raj|        IT|      NY| 65000| 29|17000|
|   Mona|     Legal|      CA| 87000| 45|12000|
|   Nina|     Legal|      NY| 93000| 38|16000|
|   John|Operations|      CA| 82000| 33|21000|
|  David|Operations|      NY| 84000| 49|19000|
|  Maria|   finance|      CA| 90000| 24|23000|
|    Jen|   finance|      NY| 79000| 53|15000|
|   Jeff| marketing|      CA| 80000| 25|18000|
|  Kumar| marketing|      NY| 91000| 50|21000|
| Robert|     sales|      CA| 81000| 30|23000|
|  James|     sales|      NY| 90000| 34|10000|
|Micheal|     sales|      NY| 86000| 56|20000|
+-------+----------+--------+------+---+-----+
```

```
# Sorting with orderby
df.orderBy('Age').show()
```

```
+-------+----------+--------+------+---+-----+
|   Name|Department|location|salary|Age|bonus|
+-------+----------+--------+------+---+-----+
|  Maria|   finance|      CA| 90000| 24|23000|
|   Jeff| marketing|      CA| 80000| 25|18000|
|   Sara|        HR|      NY| 75000| 27|14000|
|   Alex|        HR|      CA| 78000| 28|11000|
|    Raj|        IT|      NY| 65000| 29|17000|
| Robert|     sales|      CA| 81000| 30|23000|
|   Saif|        IT|      CA| 72000| 31|22000|
|   John|Operations|      CA| 82000| 33|21000|
|  James|     sales|      NY| 90000| 34|10000|
|   Nina|     Legal|      NY| 93000| 38|16000|
|   Mona|     Legal|      CA| 87000| 45|12000|
|  David|Operations|      NY| 84000| 49|19000|
|  Kumar| marketing|      NY| 91000| 50|21000|
|    Jen|   finance|      NY| 79000| 53|15000|
|Micheal|     sales|      NY| 86000| 56|20000|
+-------+----------+--------+------+---+-----+
```

```
df.orderBy(col('Age').desc()).show()
```

```
+-------+----------+--------+------+---+-----+
|   Name|Department|location|salary|Age|bonus|
+-------+----------+--------+------+---+-----+
|Micheal|     sales|      NY| 86000| 56|20000|
|    Jen|   finance|      NY| 79000| 53|15000|
|  Kumar| marketing|      NY| 91000| 50|21000|
|  David|Operations|      NY| 84000| 49|19000|
|   Mona|     Legal|      CA| 87000| 45|12000|
|   Nina|     Legal|      NY| 93000| 38|16000|
|  James|     sales|      NY| 90000| 34|10000|
|   John|Operations|      CA| 82000| 33|21000|
|   Saif|        IT|      CA| 72000| 31|22000|
| Robert|     sales|      CA| 81000| 30|23000|
|    Raj|        IT|      NY| 65000| 29|17000|
|   Alex|        HR|      CA| 78000| 28|11000|
|   Sara|        HR|      NY| 75000| 27|14000|
|   Jeff| marketing|      CA| 80000| 25|18000|
|  Maria|   finance|      CA| 90000| 24|23000|
+-------+----------+--------+------+---+-----+
```

B. pyspark program to demonstrate drop rows with NULL values

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
```

```
spark = SparkSession.builder.appName('exams').getOrCreate()
```

```
file_path = '/content/Book1.csv'
df = spark.read.options(header=True, inferSchema=True).csv(file_path)
df.show()
```

```
+---+-------+--------+------+-----+-----------+
| id|zipcode|    type|  city|state|population |
+---+-------+--------+------+-----+-----------+
|  1|    704|standard|  NULL|   PR|      30100|
|  2|    704|    NULL|Bhopal|   PR|       NULL|
|  3|    709|    NULL|Mumbai|   PR|       3700|
|  4|  76166|  unique|  Pune|   TX|      84000|
|  5|  76177|standard| Delhi|   TX|       NULL|
+---+-------+--------+------+-----+-----------+
```

```
# Drop NA/NULL values
df.dropna().show()
df.na.drop(how='any').show()
```

```
+---+-------+------+----+-----+-----------+
| id|zipcode|  type|city|state|population |
+---+-------+------+----+-----+-----------+
|  4|  76166|unique|Pune|   TX|      84000|
+---+-------+------+----+-----+-----------+

+---+-------+------+----+-----+-----------+
| id|zipcode|  type|city|state|population |
+---+-------+------+----+-----+-----------+
|  4|  76166|unique|Pune|   TX|      84000|
+---+-------+------+----+-----+-----------+
```

```
# Drop NA/NULL values
df.dropna(subset=['population ']).show()
df.na.drop(subset=['population ']).show()
```

```
+---+-------+--------+------+-----+-----------+
| id|zipcode|    type|  city|state|population |
+---+-------+--------+------+-----+-----------+
|  1|    704|standard|  NULL|   PR|      30100|
|  3|    709|    NULL|Mumbai|   PR|       3700|
|  4|  76166|  unique|  Pune|   TX|      84000|
+---+-------+--------+------+-----+-----------+

+---+-------+--------+------+-----+-----------+
| id|zipcode|    type|  city|state|population |
+---+-------+--------+------+-----+-----------+
|  1|    704|standard|  NULL|   PR|      30100|
|  3|    709|    NULL|Mumbai|   PR|       3700|
|  4|  76166|  unique|  Pune|   TX|      84000|
+---+-------+--------+------+-----+-----------+
```

C. program to demonstrate Pyspark split() columns with Options

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import split, col
```

```
spark = SparkSession.builder.appName('Exams').getOrCreate()
```

```
data = [
    ('James,Simth', '1991-04-01'),
    ('Michael,Rose', '2000-05-19'),
    ('Robert,Williams', '1978-09-12'),
    ('Maria,Jones', '1994-03-25'),
    ('Jen,Brown', '1988-07-17'),
    ('Jeff,Davis', '1992-06-04'),
    ('Kumar,Patel', '1985-10-30'),
    ('Saif,Khan', '1997-11-11'),
    ('Raj,Singh', '1990-02-02'),
    ('Alex,Johnson', '1983-08-20')
]
```

```
columns = ['Name', 'dob']

df = spark.createDataFrame(data=data, schema=columns)
```

```
df.show()
```

```
+---------------+----------+
|           Name|       dob|
+---------------+----------+
|    James,Simth|1991-04-01|
|   Michael,Rose|2000-05-19|
|Robert,Williams|1978-09-12|
|    Maria,Jones|1994-03-25|
|      Jen,Brown|1988-07-17|
|     Jeff,Davis|1992-06-04|
|    Kumar,Patel|1985-10-30|
|      Saif,Khan|1997-11-11|
|      Raj,Singh|1990-02-02|
|   Alex,Johnson|1983-08-20|
+---------------+----------+
```

```
df_split = df.withColumn("New_name", split(col('Name'), ","))
df_split.show()
```

```
+---------------+----------+------------------+
|           Name|       dob|          New_name|
+---------------+----------+------------------+
|    James,Simth|1991-04-01|    [James, Simth]|
|   Michael,Rose|2000-05-19|   [Michael, Rose]|
|Robert,Williams|1978-09-12|[Robert, Williams]|
|    Maria,Jones|1994-03-25|    [Maria, Jones]|
|      Jen,Brown|1988-07-17|      [Jen, Brown]|
|     Jeff,Davis|1992-06-04|     [Jeff, Davis]|
|    Kumar,Patel|1985-10-30|    [Kumar, Patel]|
|      Saif,Khan|1997-11-11|      [Saif, Khan]|
|      Raj,Singh|1990-02-02|      [Raj, Singh]|
|   Alex,Johnson|1983-08-20|   [Alex, Johnson]|
+---------------+----------+------------------+
```

```
df2 = df_split.withColumn('firstname', df_split['New_name'].getItem(0))\
  .withColumn('lastname', df_split['New_name'].getItem(1)).select('firstname',

df2.show()
```

```
+---------+--------+----------+
|firstname|lastname|       dob|
+---------+--------+----------+
|    James|   Simth|1991-04-01|
|  Michael|    Rose|2000-05-19|
|   Robert|Williams|1978-09-12|
|    Maria|   Jones|1994-03-25|
|      Jen|   Brown|1988-07-17|
|     Jeff|   Davis|1992-06-04|
|    Kumar|   Patel|1985-10-30|
|     Saif|    Khan|1997-11-11|
|      Raj|   Singh|1990-02-02|
|     Alex| Johnson|1983-08-20|
+---------+--------+----------+
```

```
df2_split = df2.withColumn('New_dob', split(col('dob'), '-'))
df2_split.show()
```

```
+---------+--------+----------+--------------+
|firstname|lastname|       dob|       New_dob|
+---------+--------+----------+--------------+
|    James|   Simth|1991-04-01|[1991, 04, 01]|
|  Michael|    Rose|2000-05-19|[2000, 05, 19]|
|   Robert|Williams|1978-09-12|[1978, 09, 12]|
|    Maria|   Jones|1994-03-25|[1994, 03, 25]|
|      Jen|   Brown|1988-07-17|[1988, 07, 17]|
|     Jeff|   Davis|1992-06-04|[1992, 06, 04]|
|    Kumar|   Patel|1985-10-30|[1985, 10, 30]|
|     Saif|    Khan|1997-11-11|[1997, 11, 11]|
|      Raj|   Singh|1990-02-02|[1990, 02, 02]|
|     Alex| Johnson|1983-08-20|[1983, 08, 20]|
+---------+--------+----------+--------------+
```

```
df3 = df2_split.select('firstname', 'lastname', col('New_dob').getItem(0).alias
                       col('New_dob').getItem(1).alias('month'),
                       col('New_dob').getItem(2).alias('date'))
df3.show()
```

```
+---------+--------+----+-----+----+
|firstname|lastname|year|month|date|
+---------+--------+----+-----+----+
|    James|   Simth|1991|   04|  01|
|  Michael|    Rose|2000|   05|  19|
|   Robert|Williams|1978|   09|  12|
|    Maria|   Jones|1994|   03|  25|
|      Jen|   Brown|1988|   07|  17|
|     Jeff|   Davis|1992|   06|  04|
|    Kumar|   Patel|1985|   10|  30|
|     Saif|    Khan|1997|   11|  11|
|      Raj|   Singh|1990|   02|  02|
|     Alex| Johnson|1983|   08|  20|
+---------+--------+----+-----+----+
```

D. program to demonstrate pyspark concatenation columns with Options

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

```
spark = SparkSession.builder.appName('Exams').getOrCreate()
```

```
data = [
    ("James", "Smith", "NY"),
    ("Anna", "Brown", "CA"),
    ("Robert", "Williams", "TX"),
]
```

```
columns = ['firstname', 'lastname', 'state']

df = spark.createDataFrame(data=data, schema=columns)
df.show()
```

```
+---------+--------+-----+
|firstname|lastname|state|
+---------+--------+-----+
|    James|   Smith|   NY|
|     Anna|   Brown|   CA|
|   Robert|Williams|   TX|
+---------+--------+-----+
```

```
# concat
df_concat = df.withColumn('Name', concat(col('firstname'), col('lastname')))
df_concat.select(col('Name'), col('state')).show()
```

```
+--------------+-----+
|          Name|state|
+--------------+-----+
|    JamesSmith|   NY|
|     AnnaBrown|   CA|
|RobertWilliams|   TX|
+--------------+-----+
```

```
# concat_ws
df_concat_ws = df.withColumn('Name', concat_ws("-", col('firstname'), col('las
df_concat_ws.select('Name', 'state').show()
```

```
+---------------+-----+
|           Name|state|
+---------------+-----+
|    James-Smith|   NY|
|     Anna-Brown|   CA|
|Robert-Williams|   TX|
+---------------+-----+
```

E. program to demonstrate pyspark fillna, fill and replace NULL values

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, avg
```

```
spark = SparkSession.builder.appName('exams').getOrCreate()
```

```
data = [
    ("Alice", 25, None),
    ("Bob", None, "California"),
    ("Charlie", 30, "Texas"),
    (None, 22, "Nevada"),
    ("David", None, None)
]
```

```
columns = ['Name', 'Age', 'location']
df = spark.createDataFrame(data=data, schema=columns)
df.show()
```

```
+-------+----+----------+
|   Name| Age|  location|
+-------+----+----------+
|  Alice|  25|      NULL|
|    Bob|NULL|California|
|Charlie|  30|     Texas|
|   NULL|  22|    Nevada|
|  David|NULL|      NULL|
+-------+----+----------+
```

```
# fillna
df.fillna({'Age':0, 'Name':'unknown', 'location':'unknown'}).show()
```

```
+-------+---+----------+
|   Name|Age|  location|
+-------+---+----------+
|  Alice| 25|   unknown|
|    Bob|  0|California|
|Charlie| 30|     Texas|
|unknown| 22|    Nevada|
|  David|  0|   unknown|
+-------+---+----------+
```

```
# fill
df.na.fill({'Age':0, 'location':'unknown', 'Name':'unknown'}).show()
```

```
+-------+---+----------+
|   Name|Age|  location|
+-------+---+----------+
|  Alice| 25|   unknown|
|    Bob|  0|California|
|Charlie| 30|     Texas|
|unknown| 22|    Nevada|
|  David|  0|   unknown|
+-------+---+----------+
```

```
# replace
df.replace({30:35}).show()
```

```
+-------+----+----------+
|   Name| Age|  location|
+-------+----+----------+
|  Alice|  25|      NULL|
|    Bob|NULL|California|
|Charlie|  35|     Texas|
|   NULL|  22|    Nevada|
|  David|NULL|      NULL|
+-------+----+----------+
```

Practical 3

A. pyspark program to demonstrate various Array Type Operations

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import array_contains, explode, size
```

```
spark = SparkSession.builder.appName('Exams').getOrCreate()
```

```
data = [
    (1, ["apple", "banana", "cherry"]),
    (2, ["banana", "orange"]),
    (3, ["apple"]),
    (4, [])
]
```

```
columns = ['id', 'fruits']

df = spark.createDataFrame(data=data, schema=columns)
df.show()
```

```
+---+--------------------+
| id|              fruits|
+---+--------------------+
|  1|[apple, banana, c...|
|  2|    [banana, orange]|
|  3|             [apple]|
|  4|                  []|
+---+--------------------+
```

```
# explode
df.select('id', explode('fruits').alias('fruit')).show()
```

```
+---+------+
| id| fruit|
+---+------+
|  1| apple|
|  1|banana|
|  1|cherry|
|  2|banana|
|  2|orange|
|  3| apple|
+---+------+
```

```
# size
df.select('id', size('fruits').alias('num_fruits')).show()
```

```
+---+----------+
| id|num_fruits|
+---+----------+
|  1|         3|
|  2|         2|
|  3|         1|
|  4|         0|
+---+----------+
```

```
# array_contains
df.select("id", array_contains('fruits', 'banana').alias('is_available')).show
```

```
+---+------------+
| id|is_available|
+---+------------+
|  1|        true|
|  2|        true|
|  3|       false|
|  4|       false|
+---+------------+
```

B. program to demonstrate pyspark convert array columns to a string with options

↳ 8 cells hidden

C. pyspark program to demonstrate converting a string column to an array column

↳ 7 cells hidden

D. pyspark program to demonstrate converting Map to column

↳ 7 cells hidden

programme to demonstrate use of explode an array & map

↳ 7 cells hidden

program to demonstrate use of explode on nested array

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode
```

```
spark = SparkSession.builder.appName('exams').getOrCreate()
```

```
data = [
    (1, [[1,2,3], [4,5]]),
    (2, [[2,3,4], [3,6]])
]

columns = ['id', 'array']
```

```
df = spark.createDataFrame(data=data, schema=columns)
```

```
df.show()
```

```
+---+-------------------+
| id|              array|
+---+-------------------+
|  1|[[1, 2, 3], [4, 5]]|
|  2|[[2, 3, 4], [3, 6]]|
+---+-------------------+
```

```
# exploding outer array
df2 = df.select('id', explode(df.array).alias('outerarray'))
df2.show()
```

```
+---+----------+
| id|outerarray|
+---+----------+
|  1| [1, 2, 3]|
|  1|    [4, 5]|
|  2| [2, 3, 4]|
|  2|    [3, 6]|
+---+----------+
```

```
# exploding inner arry
df2.select('id', explode(df2.outerarray).alias('innerarray')).show()
```

```
+---+----------+
| id|innerarray|
+---+----------+
|  1|         1|
|  1|         2|
|  1|         3|
|  1|         4|
|  1|         5|
|  2|         2|
|  2|         3|
|  2|         4|
|  2|         3|
|  2|         6|
+---+----------+
```

Practical 4

↳ 27 cells hidden

Practical 5

↳ 36 cells hidden

Practical 6

↳ 22 cells hidden

Practical 7

↳ 21 cells hidden

Practical 8 --- Pending

A. Write a program to demonstrate PySpark SQL ex.

↳ 12 cells hidden

B. Write a program to demonstrate Pyspark SQL expr() function

↳ 6 cells hidden

C. Write a program to demonstrate Pypspark Select Columns from DataFrame

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
```

```
spark = SparkSession.builder.appName('Exams').getOrCreate()
```

```
data = [
    ("James", "Smith", "USA", "CA", 90000, 1989),
    ("Michael", "Rose", "USA", "NY", 120000, 1999),
    ("Robert", "Williams", "USA", "CA", 95000, 1992),
    ("Maria", "Jones", "USA", "FL", 88000, 1994),
    ("Jen", "Brown", "USA", "NY", 99000, 1999)
]
cols = ["firstname", "lastname", "country", "state", "salary", "JoinYear"]
```

```
df = spark.createDataFrame(data, cols)
```

```
df.select("firstname").show()
```

```
+---------+
|firstname|
+---------+
|    James|
|  Michael|
|   Robert|
|    Maria|
|      Jen|
+---------+
```

```
df.select("firstname", 'lastname').show()
```

```
+---------+--------+
|firstname|lastname|
+---------+--------+
|    James|   Smith|
|  Michael|    Rose|
|   Robert|Williams|
|    Maria|   Jones|
|      Jen|   Brown|
+---------+--------+
```

```
df.select(col('firstname'), col('lastname')).show()
```

```
+---------+--------+
|firstname|lastname|
+---------+--------+
|    James|   Smith|
|  Michael|    Rose|
|   Robert|Williams|
|    Maria|   Jones|
|      Jen|   Brown|
+---------+--------+
```

```
df.select('firstname', 'lastname').where('salary > 100000').show()
```

```
+---------+--------+
|firstname|lastname|
+---------+--------+
|  Michael|    Rose|
+---------+--------+
```

Practical 9

↳ 18 cells hidden