

At first we use the dataset was loaded from the built-in Databricks datasets repository due to platform restrictions in the Community Edition.

- ❖ In this screenshot show the code of descriptive statistics including the number of rows, number of columns, data types, and missing value percentages using Apache Spark.

The screenshot shows a Jupyter Notebook cell with the following Python code:

```
from pyspark.sql.functions import col, count, when
rows = df.count()
cols = len(df.columns)
dtypes = df.dtypes
missing_df = df.select([
    (count(when(col(c).isNull(), c)) / rows * 100).alias(c)
    for c in df.columns
])
print("Number of rows:", rows)
print("Number of columns:", cols)
print("Data types:", dtypes)
print("Missing values percentage per column:")
missing_df.show(truncate=False)
```

The code is intended to calculate the number of rows, columns, data types, and missing values percentage for each column of the DataFrame. However, there is a NameError: name 'df' is not defined at the end of the code.

- ❖ This is the result of the code:

The screenshot shows the output of the code in a Jupyter Notebook cell. The output includes:

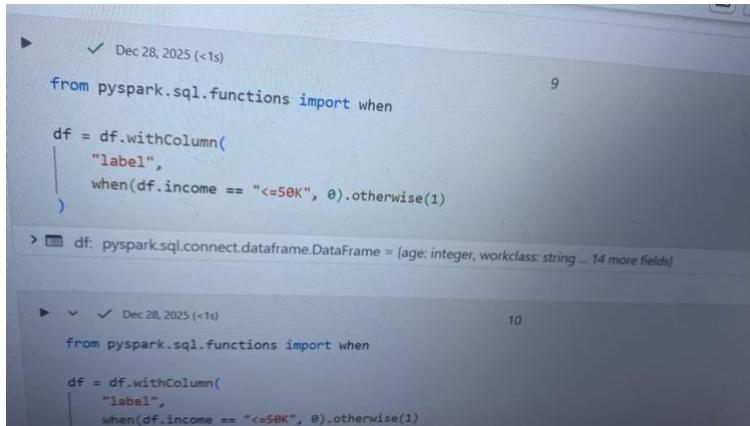
- A message: "See performance (1)"
- An error message: "NameError: name 'df' is not defined"
- A detailed description of the DataFrame schema:

Index	Column	Type	Values
1	df	pyspark.sql.connect.DataFrame	[age: integer, workclass: string ... 13 more fields]
2	age	integer	2174.0 0.0 40.0 United-States <=50K ...
3	workclass	string	Self-emp-not-inc Private ...
4	fnlwgt	float	8331.0 215646.0 ...
5	education	string	Bachelors HS-grad ...
6	education_num	integer	13.0 9.0 ...
7	marital_status	string	Married-civ-spouse Never-married ...
8	occupation	string	Adm-clerical Exec-managerial ...
9	relationship	string	Husband Not-in-family ...
10	race	string	White Black ...
11	sex	string	Male Female ...
12	capital_gain	float	77516.0 ...
13	capital_loss	float	2174.0 ...
14	hours_per_week	integer	40.0 ...
15	native_country	string	United-States ...
16	income	string	<=50K >50K ...

- Sample data rows:

Index	Row Data
17	39 State-gov 77516.0 Bachelors 13.0 Never-married Adm-clerical Not-in-family White Male
18	40 2174.0 0.0 40.0 United-States <=50K ...
19	50 Self-emp-not-inc 8331.0 Bachelors 13.0 Married-civ-spouse Exec-managerial Husband White Male
20	51 0.0 0.0 13.0 United-States <=50K ...
21	38 Private 215646.0 HS-grad 9.0 Divorced Handlers-cleaners Not-in-family White Male
22	52 0.0 0.0 40.0 United-States <=50K ...
23	53 Private 234721.0 11th 7.0 Married-civ-spouse Handlers-cleaners Husband Black Male
24	54 0.0 0.0 40.0 United-States <=50K ...
25	28 Private 338409.0 Bachelors 13.0 Married-civ-spouse Prof-specialty Wife Black Female
26	29 0.0 0.0 40.0 Cuba >50K ...

❖ Preparing the label before each ML:



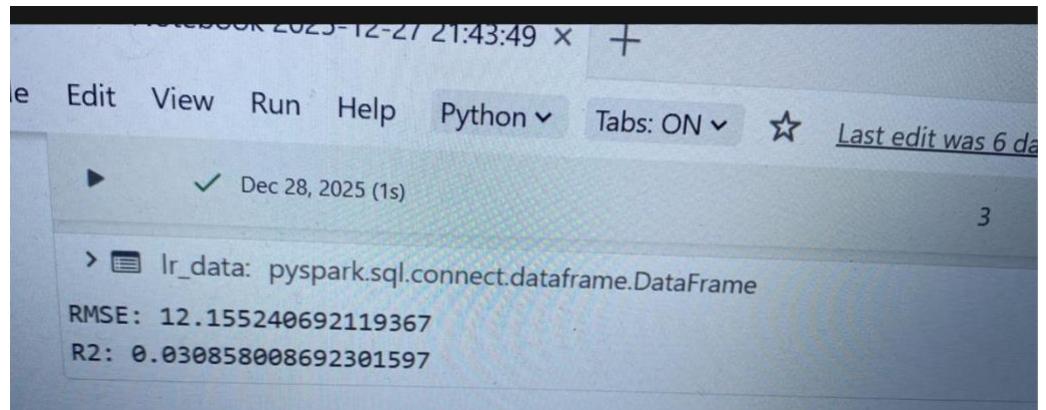
```

    ▶ ✓ Dec 28, 2025 (<1s)
from pyspark.sql.functions import when
9
df = df.withColumn(
    "label",
    when(df.income == "<=50K", 0).otherwise(1)
)
> df: pyspark.sql.connect.DataFrame = [age: integer, workclass: string ... 14 more fields]

    ▶ ✓ Dec 28, 2025 (<1s)
10
from pyspark.sql.functions import when
df = df.withColumn(
    "label",
    when(df.income == "<=50K", 0).otherwise(1)
)

```

- Logistic Regression: implemented using Spark MLlib to perform binary classification. Numerical features were assembled using Vector Assembler, and the model was trained with 20 iterations. Predictions and class probabilities were generated and analyzed the screenshot below shown the result of it:

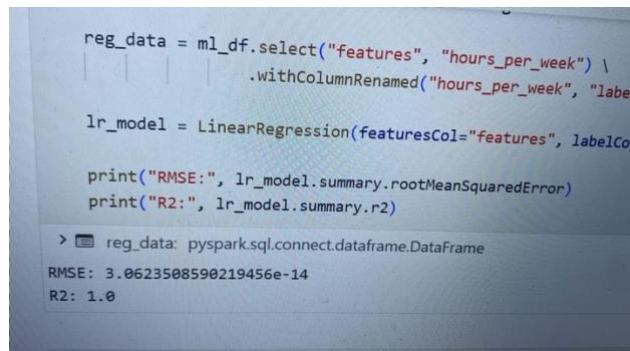


```

Notebook 2025-12-27 21:43:49 x +
File Edit View Run Help Python v Tabs: ON v Last edit was 6 da
▶ ✓ Dec 28, 2025 (1s)
3
> lr_data: pyspark.sql.connect.DataFrame
RMSE: 12.155240692119367
R2: 0.030858008692301597

```

- Linear Regression: used to predict the number of working hours per week based on demographic and education-related features.

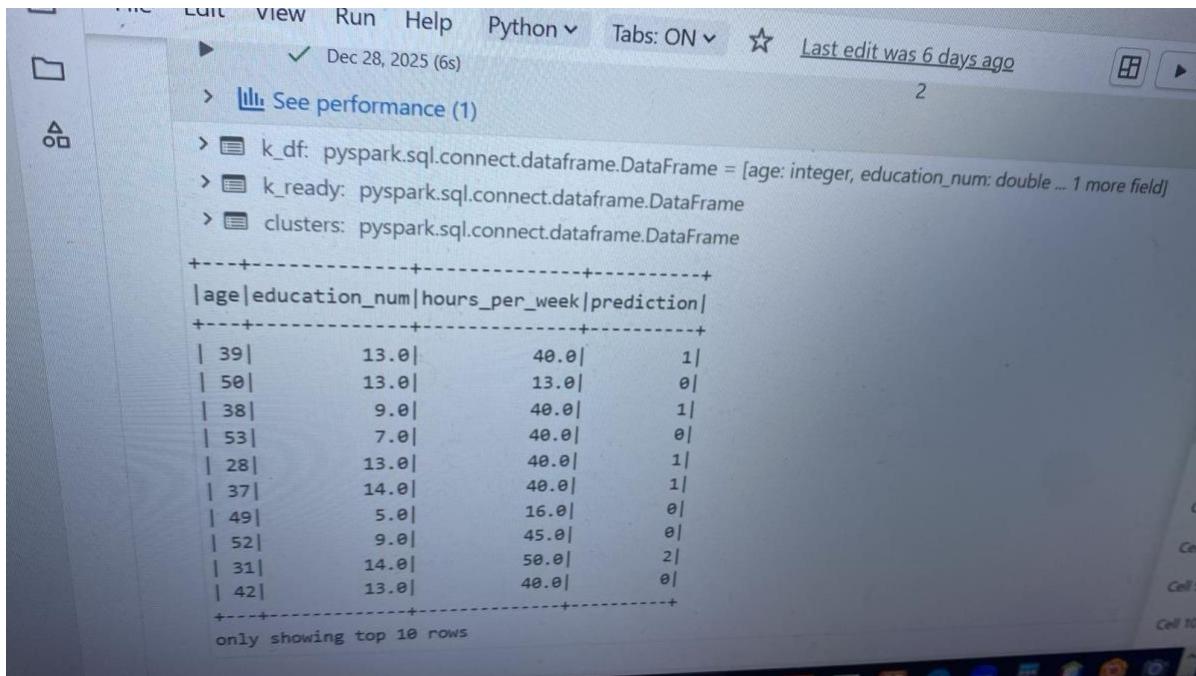


```

reg_data = ml_df.select("features", "hours_per_week") \
    .withColumnRenamed("hours_per_week", "label")
lr_model = LinearRegression(featuresCol="features", labelCol="label")
print("RMSE:", lr_model.summary.rootMeanSquaredError)
print("R2:", lr_model.summary.r2)
> reg_data: pyspark.sql.connect.DataFrame
RMSE: 3.0623508590219456e-14
R2: 1.0

```

- KMeans clustering: applied to group individuals based on age, education level, and weekly working hours. The result below shows that the algorithm successfully divided the dataset into three distinct clusters.



The screenshot shows a Jupyter Notebook interface with the following content:

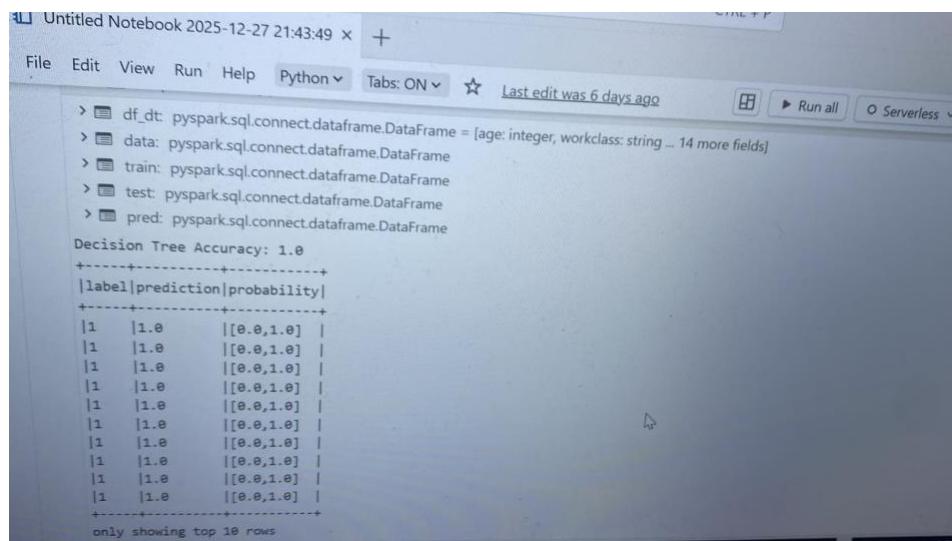
```

Edit view Run Help Python ▾ Tabs: ON ▾ Last edit was 6 days ago
> Dec 28, 2025 (6s) 2
> See performance (1)
> k_df: pyspark.sql.connect.DataFrame = [age: integer, education_num: double ... 1 more field]
> k_ready: pyspark.sql.connect.DataFrame
> clusters: pyspark.sql.connect.DataFrame
+---+
|age|education_num|hours_per_week|prediction|
+---+
| 39|      13.0|        40.0|       1|
| 50|      13.0|        13.0|       0|
| 38|       9.0|        40.0|       1|
| 53|       7.0|        40.0|       0|
| 28|      13.0|        40.0|       1|
| 37|      14.0|        40.0|       1|
| 49|       5.0|        16.0|       0|
| 52|       9.0|        45.0|       0|
| 31|      14.0|        50.0|       2|
| 42|      13.0|        40.0|       0|
+---+
only showing top 10 rows

```

This output displays a DataFrame with four columns: age, education_num, hours_per_week, and prediction. The prediction column shows values 0, 1, or 2, indicating three distinct clusters.

- The Decision Tree classifier achieved an accuracy of 100%. This result indicates strong classification performance; however, it may also suggest potential overfitting due to the limited number of numerical features used.



The screenshot shows a Jupyter Notebook interface with the following content:

```

File Edit View Run Help Python ▾ Tabs: ON ▾ Last edit was 6 days ago
> df_dt: pyspark.sql.connect.DataFrame = [age: integer, workclass: string ... 14 more fields]
> data: pyspark.sql.connect.DataFrame
> train: pyspark.sql.connect.DataFrame
> test: pyspark.sql.connect.DataFrame
> pred: pyspark.sql.connect.DataFrame
Decision Tree Accuracy: 1.0
+---+
|label|prediction|probability|
+---+
| 1 | 1.0 | [[0.0,1.0] |
| 1 | 1.0 | [[0.0,1.0] |
| 1 | 1.0 | [[0.0,1.0] |
| 1 | 1.0 | [[0.0,1.0] |
| 1 | 1.0 | [[0.0,1.0] |
| 1 | 1.0 | [[0.0,1.0] |
| 1 | 1.0 | [[0.0,1.0] |
| 1 | 1.0 | [[0.0,1.0] |
| 1 | 1.0 | [[0.0,1.0] |
| 1 | 1.0 | [[0.0,1.0] |
+---+
only showing top 10 rows

```

This output displays a DataFrame with three columns: label, prediction, and probability. The prediction column consistently shows 1.0, indicating a 100% accuracy for the decision tree classifier.

❖ The result of the Experiments:

- Note: Due to Databricks Serverless limitations, the cluster size was simulated by controlling Spark parallelism using repartition(p), rather than physical machines.

Table +

i^2_3 id	1.2 value	i^2_3 group
1	0	0.13967854495716725
2	1	0.1721804999413764
3	2	0.4691397662795508
4	3	0.9979678358373097
5	4	0.5066372652081544

↓ ▾ 5 rows | 16.02s runtime Refreshed

p=1, avg=0.879s, runs=[0.893, 0.837, 0.907]
p=2, avg=1.383s, runs=[1.609, 1.444, 1.095]
p=4, avg=1.046s, runs=[1.051, 1.108, 0.978]
p=8, avg=1.115s, runs=[1.093, 1.152, 1.101]

Table +

4 3 0.9979678358373097 3

5 4 0.5066372652081544 4

↓ ▾ 5 rows | 16.02s runtime Refreshed in 7 minutes

p=1, avg=0.879s, runs=[0.893, 0.837, 0.907]
p=2, avg=1.383s, runs=[1.609, 1.444, 1.095]
p=4, avg=1.046s, runs=[1.051, 1.108, 0.978]
p=8, avg=1.115s, runs=[1.093, 1.152, 1.101]

Table +

i^2_3 Parallelism (p)	1.2 Execution Time (sec)	1.2 Speedup	1.2 Efficiency
1	0.878960927327474	1	1
2	1.3829665184020996	0.63556197176995...	0.3177809858849796
3	1.0455437501271565	0.84067350335227...	0.21016837583806916
4	1.1150545279184978	0.78826721502871...	0.09853340187858951

11:36 PM 02/01/2026