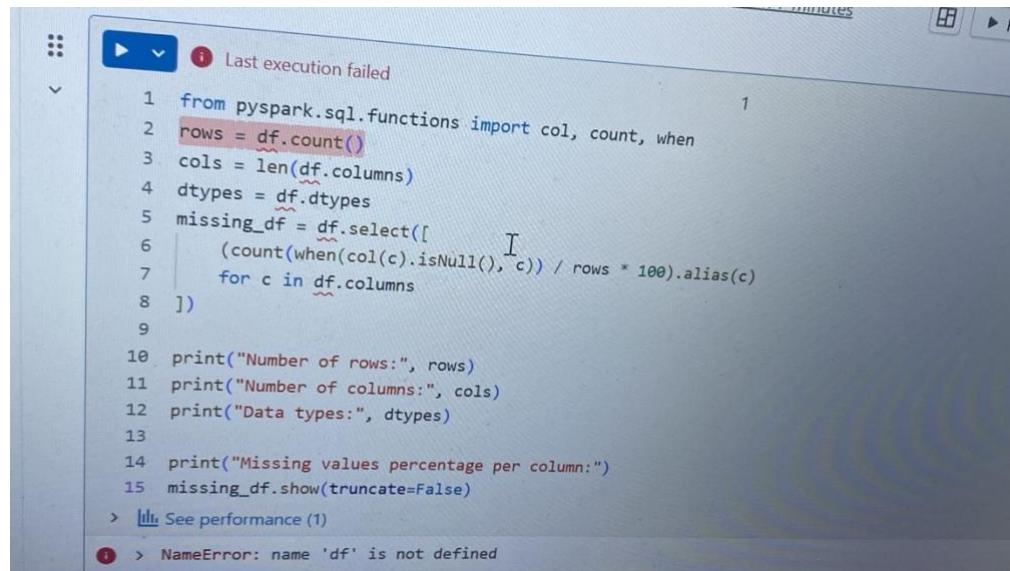


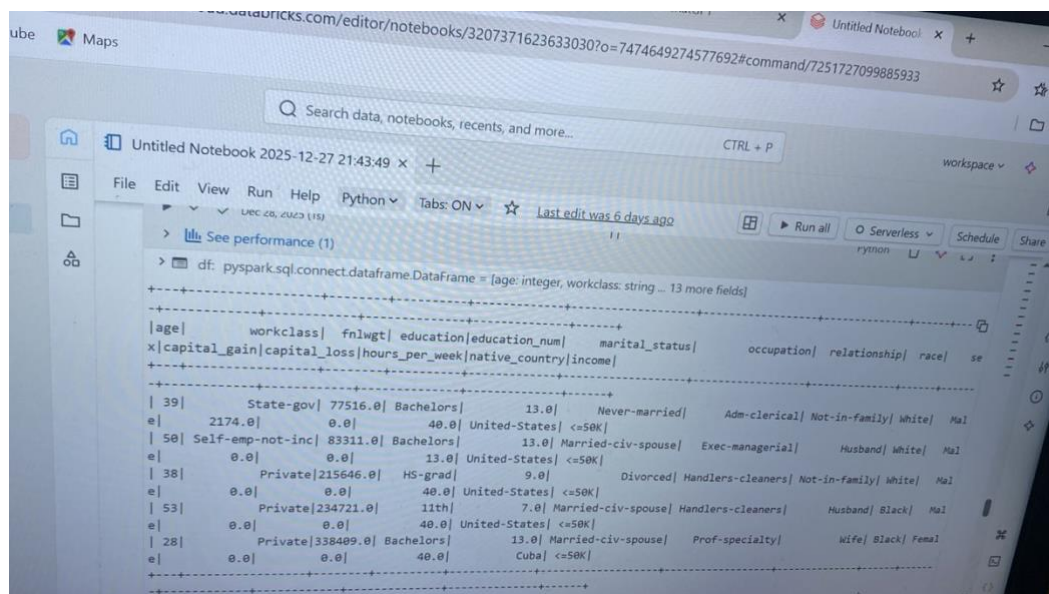
At first we use the dataset was loaded from the built-in Databricks datasets repository due to platform restrictions in the Community Edition.

- ❖ 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.



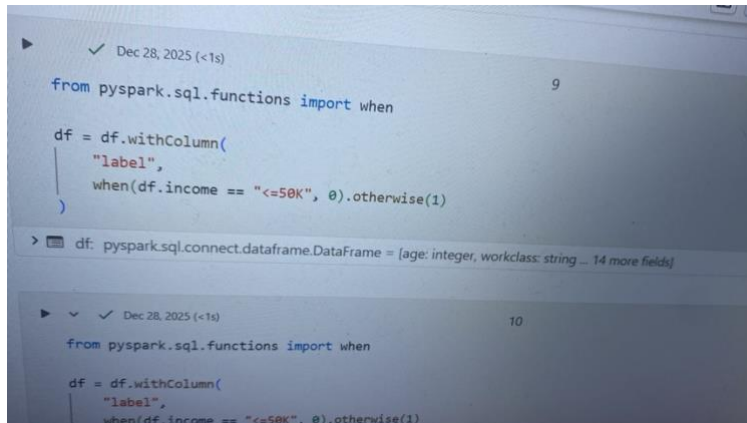
```
1 from pyspark.sql.functions import col, count, when
2 rows = df.count()
3 cols = len(df.columns)
4 dtypes = df.dtypes
5 missing_df = df.select([
6     (count(when(col(c).isNull(), c)) / rows * 100).alias(c)
7     for c in df.columns
8 ])
9
10 print("Number of rows:", rows)
11 print("Number of columns:", cols)
12 print("Data types:", dtypes)
13
14 print("Missing values percentage per column:")
15 missing_df.show(truncate=False)
> See performance \(1\)
NameError: name 'df' is not defined
```

- ❖ This is the result of the code:



	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex
39	State-gov	77516.0	Bachelors	13.0	Never-married	Adm-clerical	Not-in-family	White	Mal	
50	Self-emp-not-inc	83311.0	Bachelors	13.0	Married-civ-spouse	Exec-managerial	Husband	White	Mal	
38	Private	215646.0	HS-grad	9.0	Divorced	Handlers-cleaners	Not-in-family	White	Mal	
53	Private	234721.0	11th	7.0	Married-civ-spouse	Handlers-cleaners	Husband	Black	Mal	
28	Private	338409.0	Bachelors	13.0	Married-civ-spouse	Prof-specialty	Wife	Black	Female	

❖ Preparing the label before each ML:



```
from pyspark.sql.functions import when

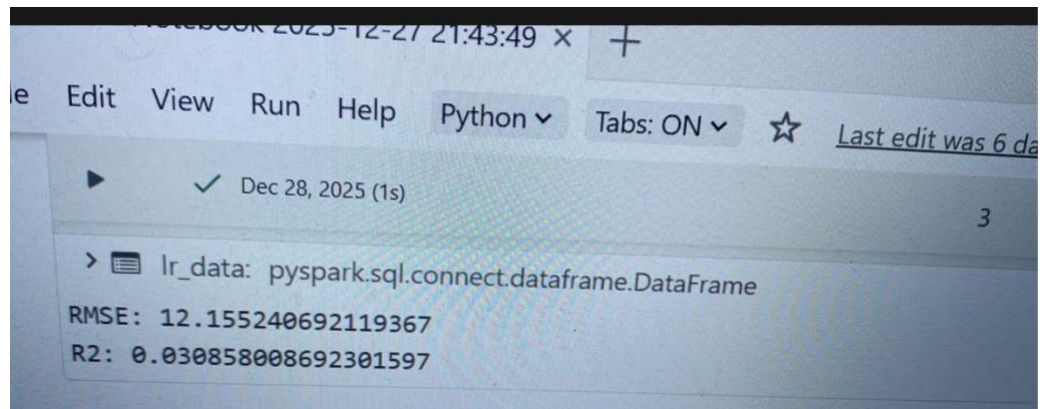
df = df.withColumn(
    "label",
    when(df.income == "<=50K", 0).otherwise(1)
)
```

> df: pyspark.sql.connect.dataframe.DataFrame = [age: integer, workclass: string ... 14 more fields]

```
from pyspark.sql.functions import when

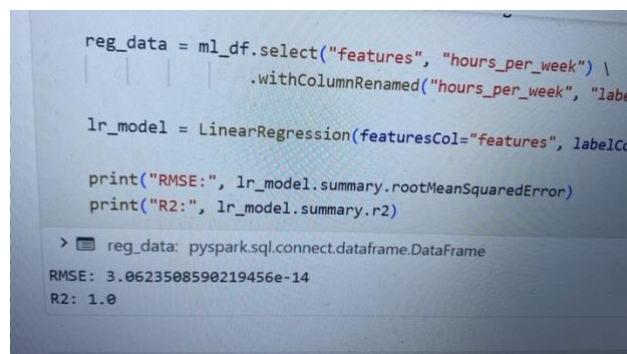
df = df.withColumn(
    "label",
    when(df.income == "<=50K", 0).otherwise(1)
)
```

- 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:



```
lr_data: pyspark.sql.connect.dataframe.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.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 shown that the algorithm successfully divided the dataset into three distinct clusters.

```

> k_df: pyspark.sql.connect.dataframe.DataFrame = [age: integer, education_num: double ... 1 more field]
> k_ready: pyspark.sql.connect.dataframe.DataFrame
> clusters: pyspark.sql.connect.dataframe.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

```

- 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.

```

> df_dt: pyspark.sql.connect.dataframe.DataFrame = [age: integer, workclass: string ... 14 more fields]
> data: pyspark.sql.connect.dataframe.DataFrame
> train: pyspark.sql.connect.dataframe.DataFrame
> test: pyspark.sql.connect.dataframe.DataFrame
> pred: pyspark.sql.connect.dataframe.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

```


❖ 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.

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

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]

	1.2_3 Parallelism (p)	1.2 Execution Time (sec)	1.2 Speedup	1.2 Efficiency
1	1	0.878960927327474	1	1
2	2	1.3829665184020996	0.63556197176995...	0.3177809858849796
3	4	1.0455437501271565	0.84067350335227...	0.21016837583806916
4	8	1.1150545279184978	0.78826721502871...	0.09853340187858951

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]