

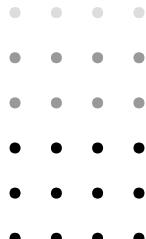
# SCALING

DATA WORKLOADS  
*using the best of both  
worlds:*

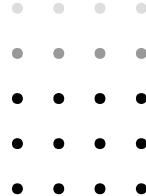
## PANDAS AND SPARK

---

Chengyin Eng (Senior Data Scientist)  
Hyukjin Kwon (Staff Software Engineer)



# AGENDA



---

## 01 pandas API on Spark

Swapping pandas for pyspark.pandas

---

## 02 pandas udf & function API

Writing distributed UDFs

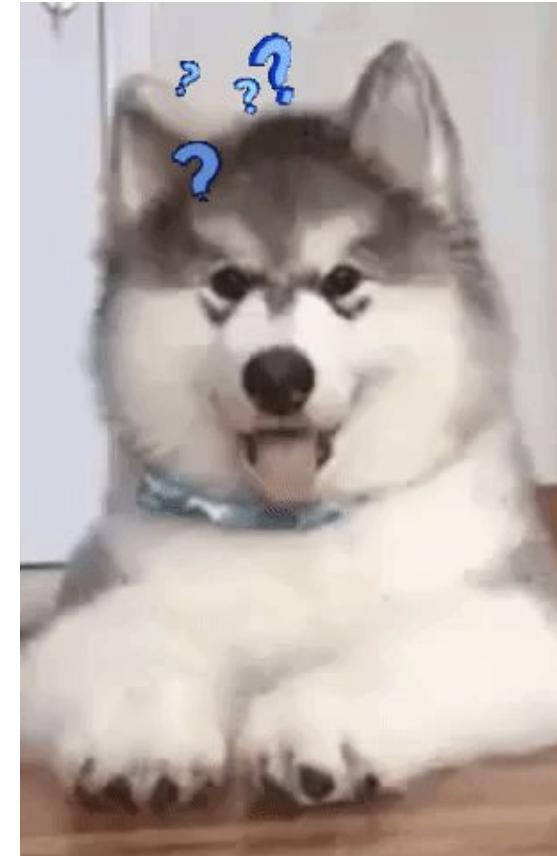
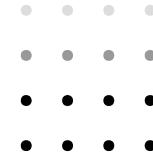
---

## 03 Spark Connect

Use Spark everywhere

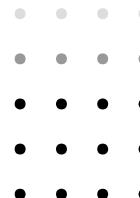
pandas is the de-facto library for  
data exploration and wrangling

Oh..will it scale?

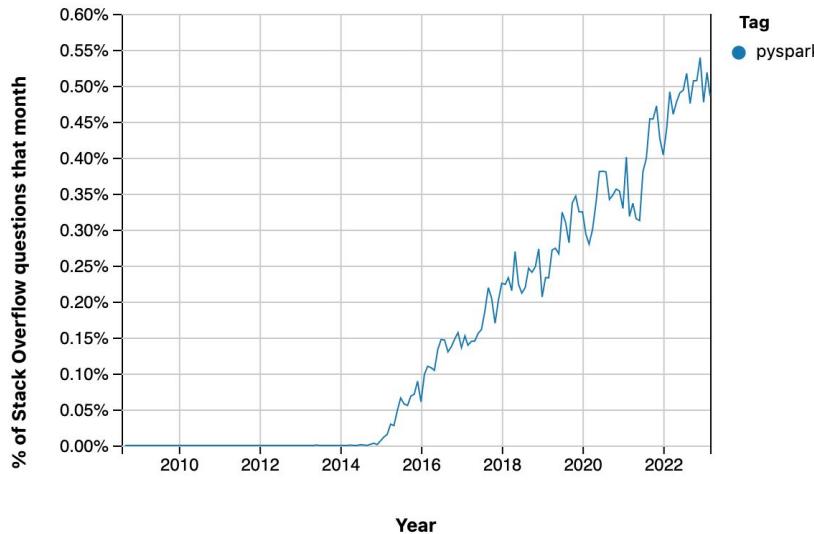
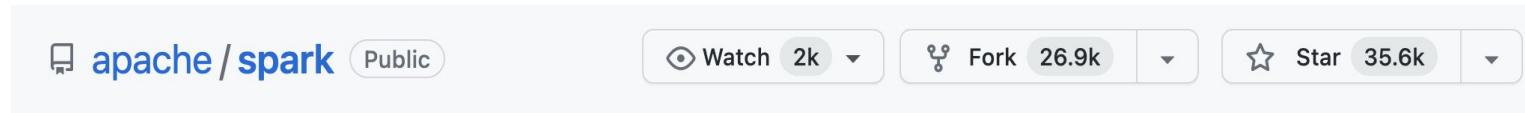


# How do I scale?

1. Just....throw a bigger (and bigger) instance
2. Downsample data
3. Switch to a distributed data processing framework

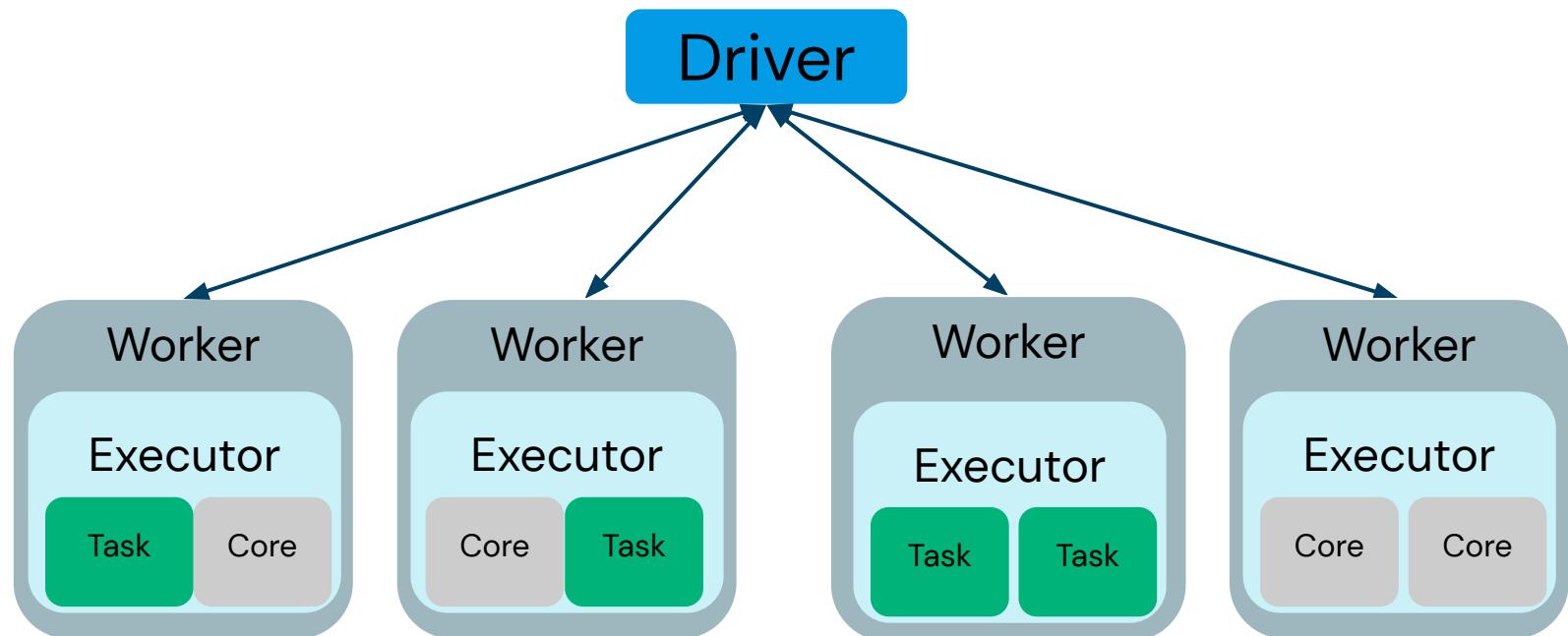


# Spark is popular (PySpark too!)



[Stack Overflow Trends](#)

# Spark: A distributed processing engine



# But...it's a new syntax

```
# pandas
import pandas as pd

pdf = pd.read_parquet("your_data.parquet")
pdf_filtered = pdf.loc[pdf["quantity"] > 10, ["date", "quantity"]]
pdf
```



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

df = spark.read.format("parquet").load("your_data.parquet")
df_filtered = df.select("date", "quantity").filter("quantity > 10")
df_filtered.show()
```

# Differences

	<b>Spark dataframe</b>	<b>pandas dataframe</b>
<b>Mutability</b>	No	Yes
<b>Processing</b>	Distributed	Single-machine <ul style="list-style-type: none"><li>• Driver-only</li></ul>
<b>Preserves row order</b>	No <ul style="list-style-type: none"><li>• No indices</li></ul>	Yes <ul style="list-style-type: none"><li>• Has indices</li></ul>
<b>Evaluation</b>	Lazy	Eager

# 01

## pyspark.pandas

Don't want to learn  
PySpark? No problem.



# Pyspark.pandas = easy transition to distributed environment

- Announced in April 2019 as "Koalas"
  - Integrated into the PySpark library in 2021
- Unifies pandas and Spark: no need to learn a new syntax!
- Scale up your code easily

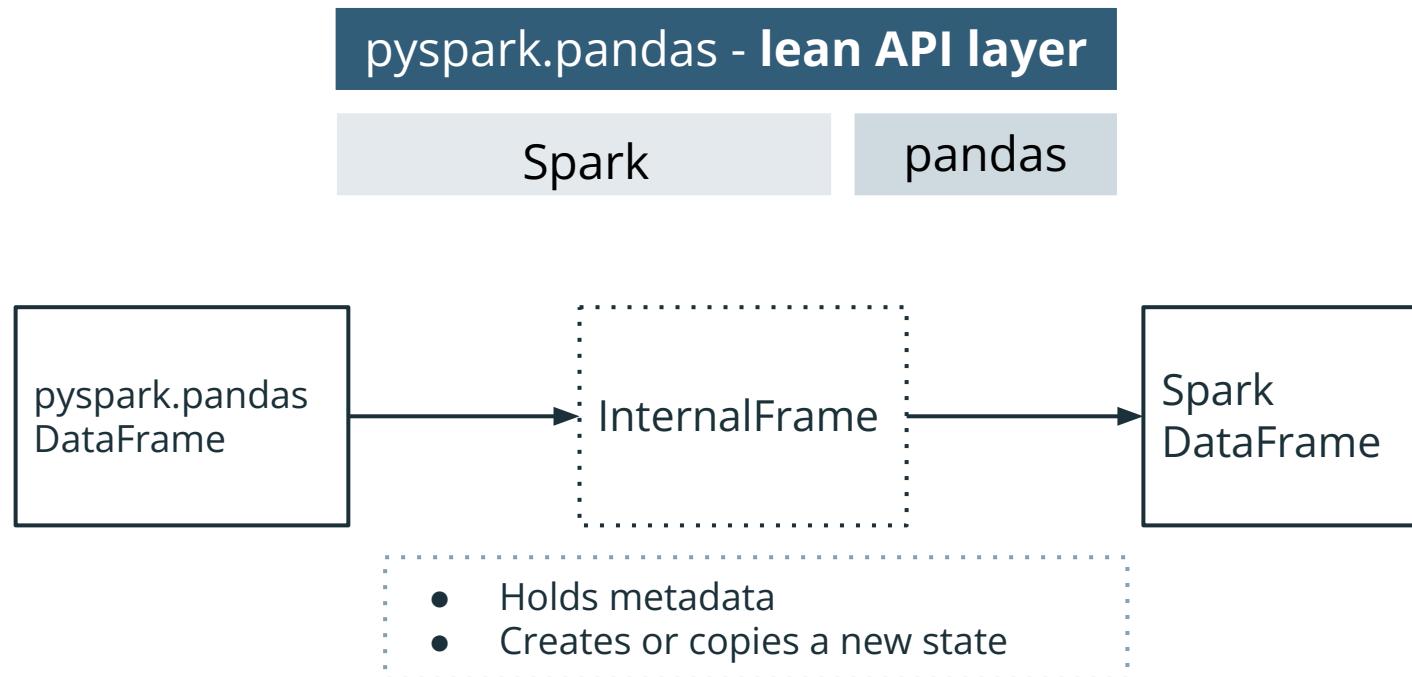


**Koalas**

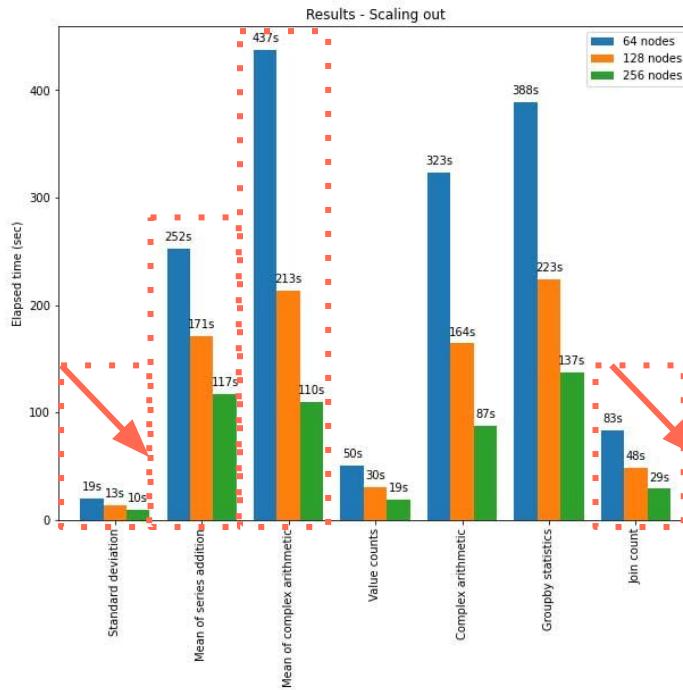
```
# import pandas as pd
import pyspark.pandas as ps

pdf = ps.read_parquet("your_data.parquet")
pdf_filtered = pdf.loc[pdf["quantity"] > 10, ["data", "quantity"]]
pdf
```

# How does it work under the hood

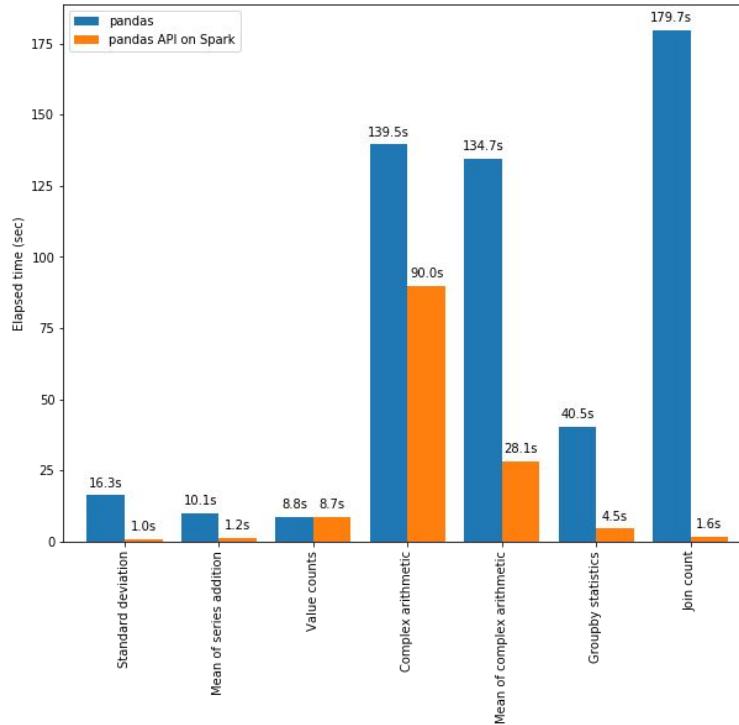


# Distributed cluster: it scales almost linearly with # of nodes



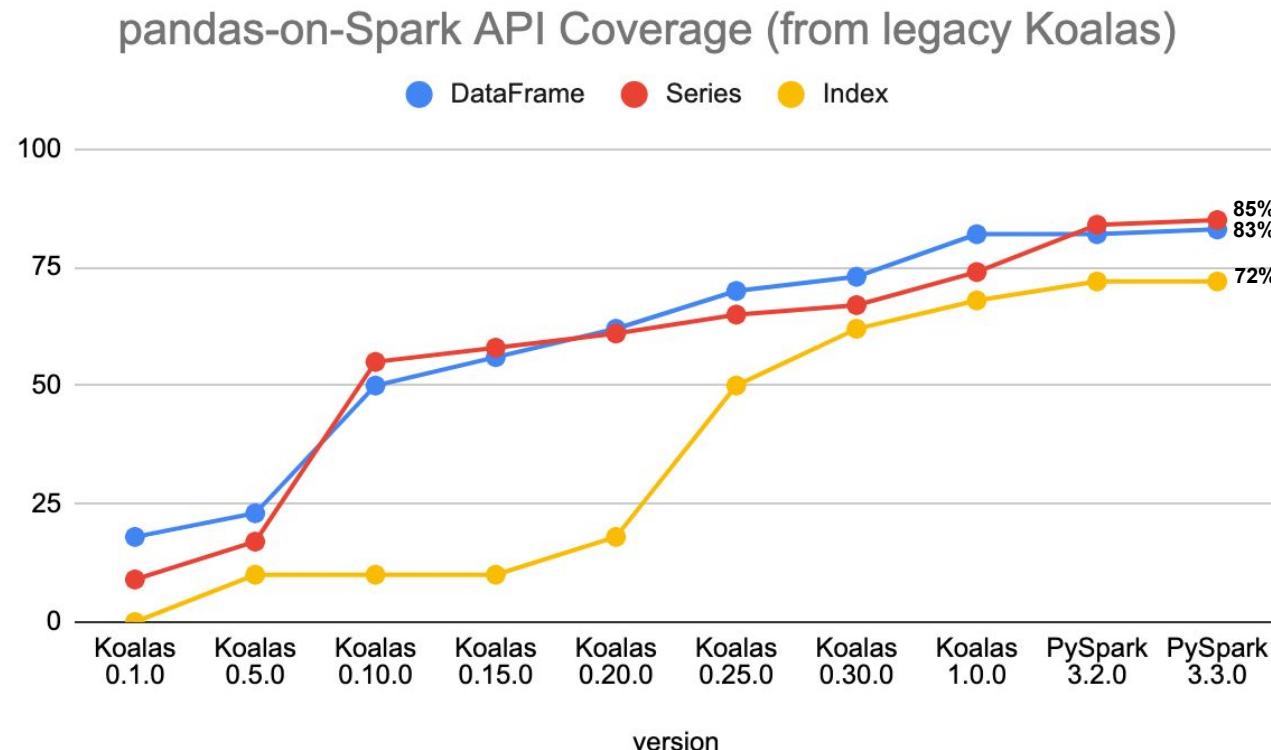
[Read blog post here](#)

# *Single-node cluster: it's faster than pandas*



[Read blog post here](#)

# API coverage



# Best practice: index choices

```
ps.set_option("compute.default_index_type", "sequence")
```

sequence

- Implements a sequence one-by-one
- Whole partition might end up in single node
- Avoid when data is large

Distributed  
-sequence  
(default)

- Implements a sequence one-by-one for each group-by and group-map approach
- Better than **sequence** when data is large

distributed

- Does not increase one-by-one
- Index is indeterministic
- ***Most performant***

[More on indices](#)

[More best practices](#)

# Should I learn PySpark or pyspark.pandas?

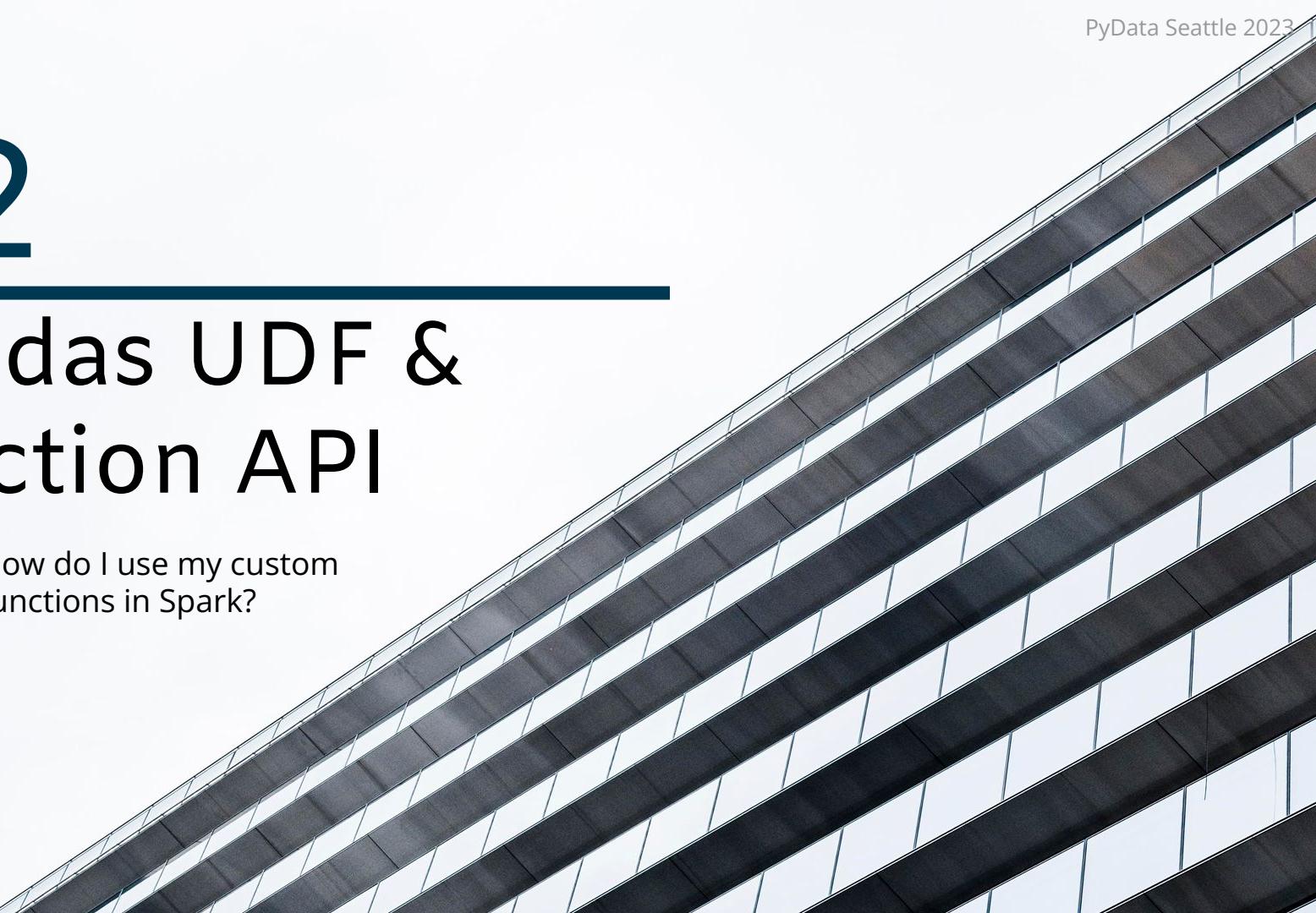
- If you know `pandas` but want to scale out to bigger data, then use `pyspark.pandas`
- If you are interested in learning PySpark and want to use Spark for streaming, then learn `PySpark`

# 02

---

## pandas UDF & function API

How do I use my custom  
functions in Spark?



• • •  
• • •  
• • •  
• • •  
• • •

# pandas friends in PySpark

## What is what?

- pandas UDF
  - Works as a function, internally an expression
  - Consistent with Scala UDFs and regular Python UDFs
  - Returns a regular PySpark column
  - e.g., `pandas_udf`
- pandas function APIs
  - Works as an API in DataFrame, query plan internally
  - Consistent with APIs such as map, mapGroups, etc.
  - e.g., `DataFrame.mapInPandas`

# pandas friends in PySpark

## What is what?

- pandas UDF

```
@pandas_udf("long")
def pandas_plus_one(v: pd.Series) -> pd.Series:
    return v + 1
```

- pandas function APIs

```
def pandas_plus_one(v: pd.DataFrame) -> pd.DataFrame:
    return v + 1

df.mapInPandas(pandas_plus_one, df.schema)
```

# pandas UDF

- Annotate the type you want to take as the input and output
- PySpark decides how to execute them
- They are (de)serialized and executed in the vectorized manner

```
@pandas_udf("int")
def my_udf1(v: pd.Series) -> pd.Series:
    return v + 1
```

# Type what you want

```
@pandas_udf("int")
def my_udf1(v: pd.Series) -> pd.Series:
    return v + 1 # Plus one for each value
```

```
@pandas_udf("int")
def my_udf2(v: pd.Series) -> int:
    return v.sum() # Sum of the whole column
```

```
@pandas_udf("int")
def my_udf3(pdf: pd.DataFrame) -> pd.Series:
    return pdf.sum(axis=0) # Sum of each row
```

# pandas function API

- (Optionally) annotate the type you want to take as the input and output
- PySpark decides how to execute them based on which API you choose
- They are (de)serialized and executed in the vectorized manner

```
def pandas_filter(iterator: Iterator[pd.DataFrame]) ->
    Iterator[pd.DataFrame]:
    for pdf in iterator:
        yield pdf[pdf.id == 1]

df.mapInPandas(pandas_filter, df.schema)
```

# Call what you want

```
def subtract_mean(pdf: pd.DataFrame) -> pd.DataFrame:
    return pdf.assign(v=pdf.v - v.mean())
df.groupby("id").applyInPandas(subtract_mean, df.schema)
```

```
def asof_join(left: pd.DataFrame, right: pd.DataFrame) -> pd.DataFrame:
    return pd.merge_asof(left, right, on="time", by="id")
df1.groupby("id").cogroup(df2.groupby("id"))
    .applyInPandas(asof_join, "time int, id int, v1 double, v2 string")
```

```
def pandas_filter(itr: Iterator[pd.DataFrame]) -> Iterator[pd.DataFrame]:
    for pdf in itr:
        yield pdf[pdf.id == 1]
df.mapInPandas(pandas_filter, df.schema)
```

# ML: Train on grouped instances

```
def train_my_model(pdf: pd.DataFrame) -> pd.DataFrame:
    x, y = pdf[["feature_1"]], pdf["label"]
    rf = RandomForestClassifier()
    rf.fit(x, y)

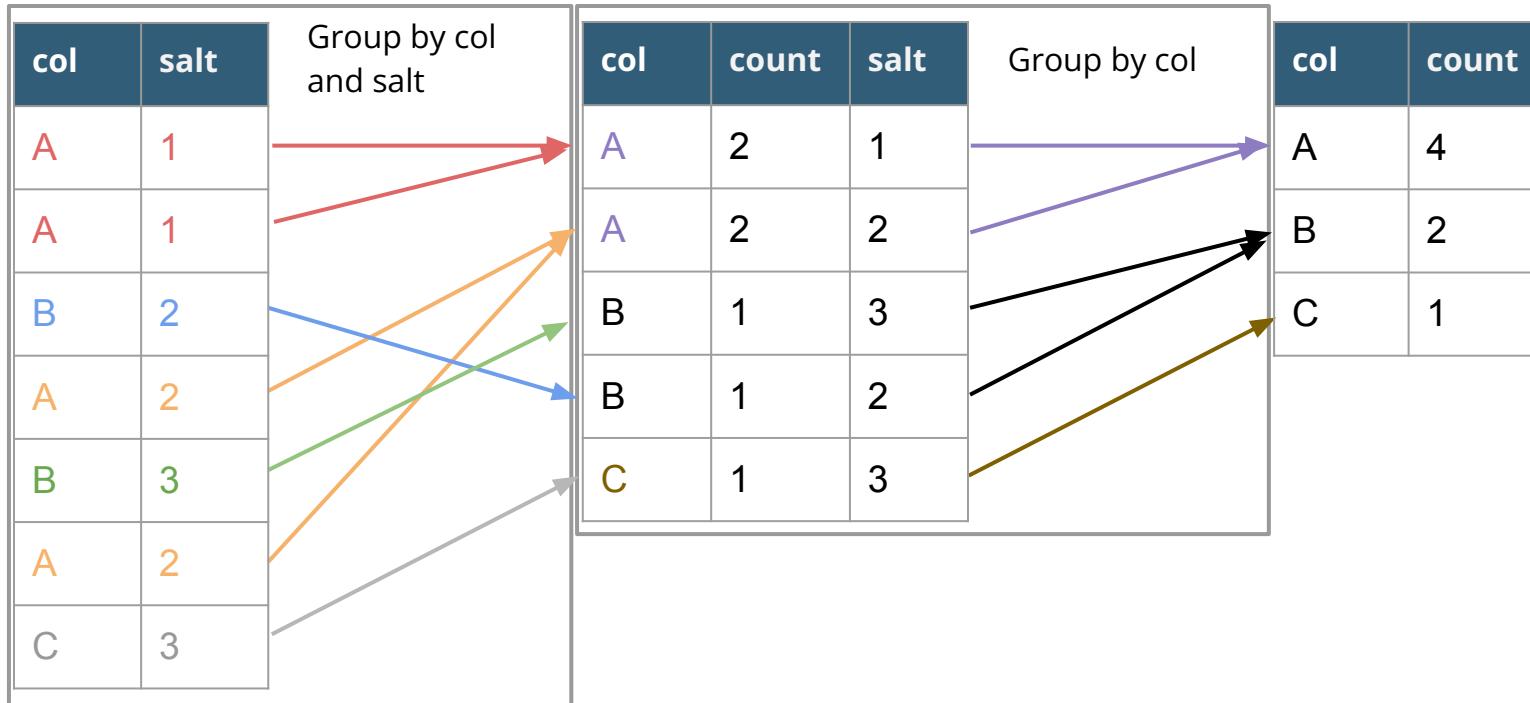
    # Evaluate the model
    predictions = rf.predict(X)
    d_id, n_used, auc = (
        pdf["device_id"].iloc[0], pdf.shape[0], auc(y, predictions)
    )

    return pd.DataFrame(
        [[device_id, n_used, auc]], columns=["d_id", "n_used", "auc"])
```

# ML: Train on grouped instances

```
# `spark_df` is Spark DataFrame
model_directories_df = spark_df.groupby("device_id")
    .applyInPandas(
        train_my_model,
        schema="device_id INT, n_used INT, auc FLOAT")
```

# OOM (Out-of-Memory)? Salt it



# OOM (Out-of-Memory)? Salt it

```
df.groupby(col).applyInPandas(subtract_mean, df.schema)
```



```
(df.withColumn("salt", (rand * n).cast(IntegerType()))
  .groupBy("salt", col)
  .applyInPandas(subtract_mean, df.schema)
  .groupBy(col)
  .agg(...))
```

# Debug? Performance?

- How to debug implementation errors?
  - Make sure it works on a single node before distributing in Spark
  - If you are using iterators (i.e. `mapInPandas`), use Python closures
  - Remote Debugging, Memory Profiler and Python Profilers
- Make sure your functions are optimized! No for-loops
- Make sure your data is properly partitioned
  - `pyspark_df.repartition("{n or col_name}").mapInPandas(...)`
- Use Spark UI to debug performance issues
  - Add print statements in your functions
  - Go to your Jobs tab -> Stages tab and check the standard errors/outputs

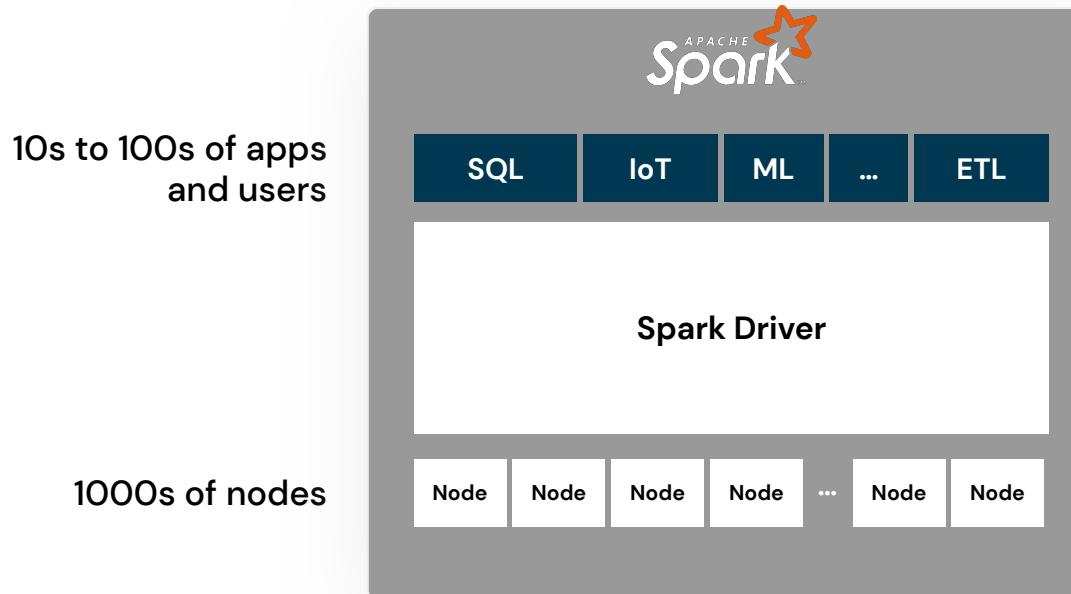
# 03

## Spark Connect

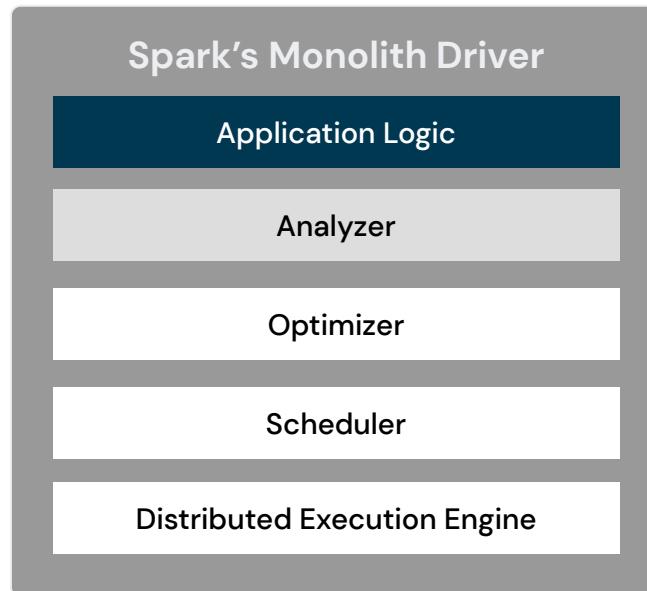
Run Spark everywhere



# Spark is often associated with data centers and large clusters



# If you zoom in ...



# Data applications don't just live in data centers anymore

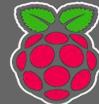
Interactive Development (notebooks / IDEs)



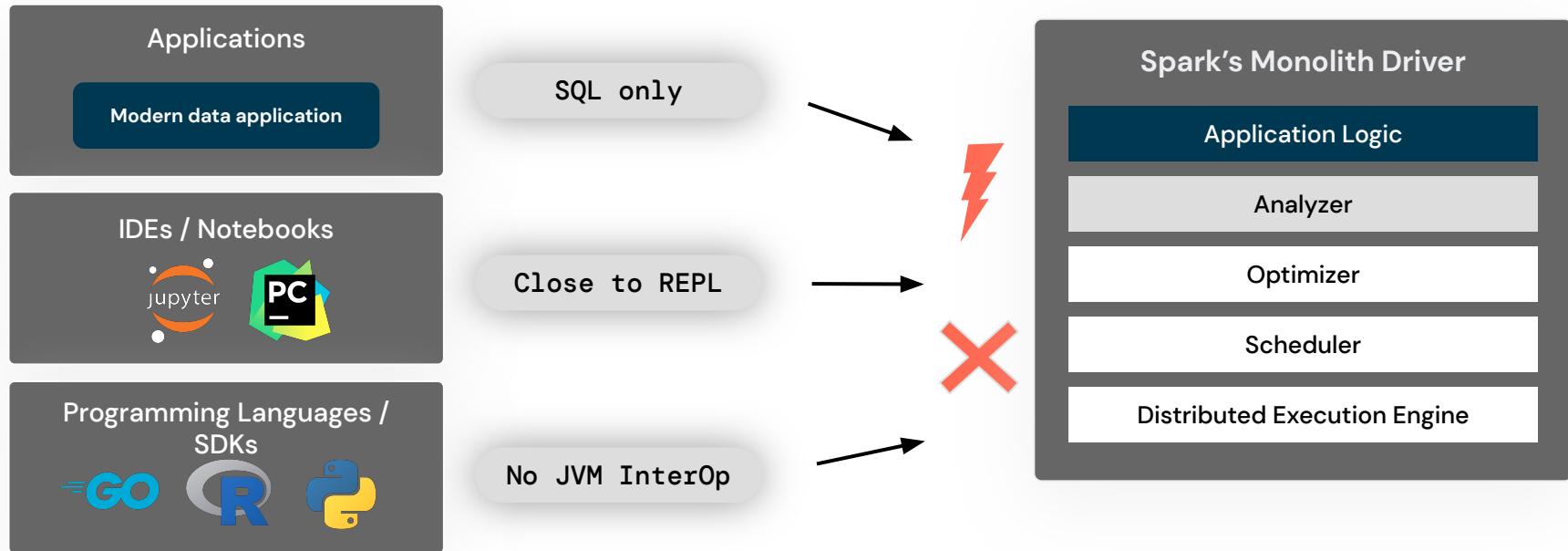
Application Servers / Web Services



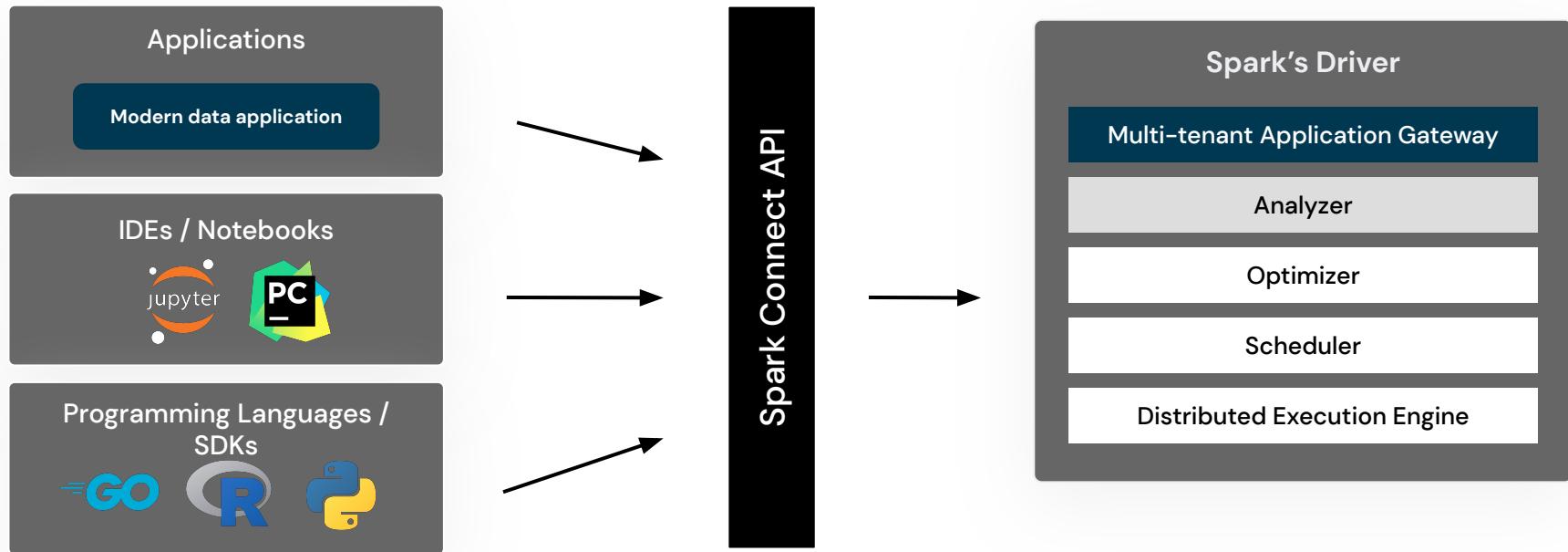
Edge Devices



# How to embed Spark in applications?



# Thin client, with the full power of Apache Spark



DataFrame API

```
spark.read.table("logs")
    .select("id",
extract_profile("blob").write.insertInto("profiles")
```

Translated into logical  
parse plan

```
InsertInto `profiles`
+- Project
    +- UnresolvedTable `logs`
```

Unresolved logical plan is sent  
to the server via gRPC/protobuf  
(language agnostic)

Results streamed back to  
the client via gRPC/Arrow  
(language agnostic)

## Spark Server

Analyzer

Optimizer

Scheduler

Distributed  
Execution Engine

# Power of Apache Spark, everywhere, with a thin client

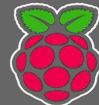
Interactive Development (notebooks / IDEs)



Application Servers / Web Services



Edge Devices



Power of Apache Spark,  
everywhere, with a thin client



# Example: Jupyter visualization

The screenshot shows a Jupyter Notebook interface running in a browser window. The title bar indicates the window is titled "Untitled3.ipynb (3) - JupyterLab". The address bar shows the URL "localhost:8888/lab/tree/Untitled3.ipynb". The left sidebar displays a file tree with several files and folders, including "build", "databricks\_connect.egg-info", "dist", "docs", "lib", "pyspark", "pyspark.egg-info", "target", "test\_coverage", "test\_support", "coverage.xml", "MANIFEST.in", "mypy.ini", "README.md", "run-tests", "run-tests-with-coverage", "run-tests.py", "setup.cfg", "setup.py", "test.py", "Untitled.ipynb", "Untitled1.ipynb", "Untitled2.ipynb", and "Untitled3.ipynb". The file "Untitled3.ipynb" is highlighted at the bottom of the tree. The main area contains three code cells:

```
[1]: from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()

[4]: import pywalker as pyg

[9]: df = spark.read.table("samples.mytaxi.trips").filter("trip_distance < 10")
df = (df.select(
    df["pickup_zip"].cast("string"),
    df["dropoff_zip"].cast("string"),
    df["trip_distance"],
    df["fare_amount"]))

pdf = df.toPandas()
```

The status bar at the bottom shows "Mode: Command" and "Ln 1, Col 1 Untitled3.ipynb".

# Example: IDE

The screenshot shows the PyCharm IDE interface with the following details:

- Title Bar:** demo\_dbconnect - main.py
- Project Bar:** demo\_dbconnect (selected), main.py, loader.py, dev.env
- Code Editor:** The main window displays the content of `main.py`. The code includes PySpark imports, a transformation function `transform_df`, a test class `MyTest` with a `setUp` method, and a `test_simple_df` test case. A red circle highlights the line `res = df.transform(transform_df)`, indicating a potential error or warning.
- Toolbars and Status Bar:** Standard PyCharm toolbars and status bar at the bottom showing "Tests passed: 0 (moments ago)" and system information.

```
demo_dbconnect - main.py
demo_dbconnect > main.py
Project main.py × loader.py × dev.env ×
1 import pyspark.sql.connect.functions as F
2
3
4 def transform_df(input):
5     """Perform a complex transformation."""
6     return input.where(F.col("trip_distance") < F.lit(10)).limit(11)
7
8 class MyTest(unittest.TestCase):
9
10     def setUp(self) -> None:
11         self.spark = SparkSession.builder.getOrCreate()
12
13     def test_simple_df(self):
14         df = self.spark.read.table("samples.nyctaxi.trips")
15
16         res = df.transform(transform_df)
17         self.assertEqual(10, len(res.collect()))
18
19
20
21 if __name__ == "__main__":
22     unittest.main()
```

# Example: Run in your browser

```

def spark_session():
    with open("cluster.json") as f:
        config = json.load(f)

    host = config["workspaceUrl"]
    clusterId = config["clusterId"]
    token = config["token"]

    connStr = f"sc:///{host}:443;/token={token};x-databricks-cluster-id={clusterId}"

    return SparkSession.builder.remote(connStr).getOrCreate()

spark = spark_session()

app = Dash(__name__)

app.layout = html.Div(children=[
    html.H1(children="NYC Taxi Cockpit: Plotly x Databricks Demo"),

    html.Div([
        html.H2("Predicting fare and trip time with a Machine Learning Model (on Databricks)"),
        html.Span([
            "Distance [miles]: ",
            dcc.Input(id="trip-distance", value="1", type="number")
        ]),
        ...
    ])
])

```

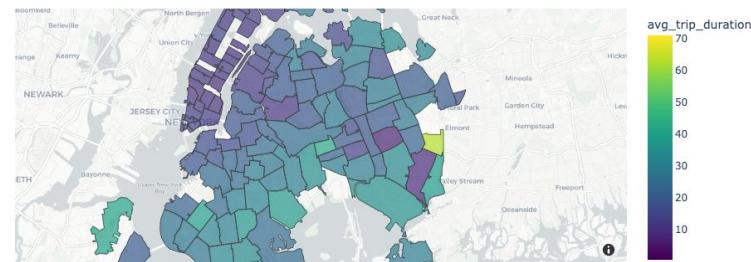
## NYC Taxi Cockpit: Plotly x Databricks Demo

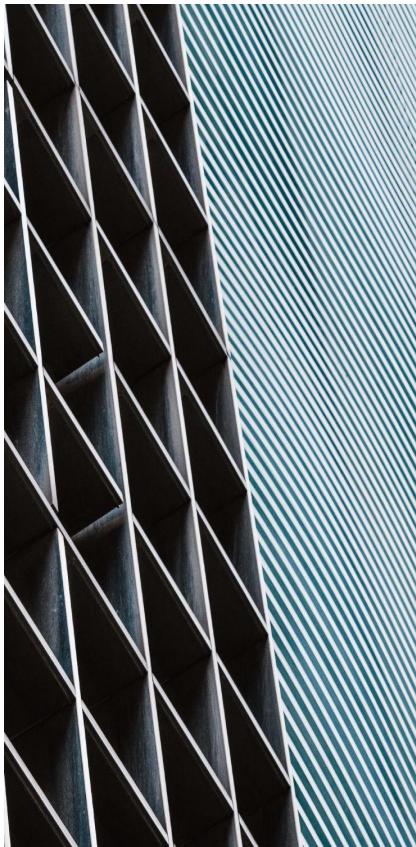
### Predicting fare and trip time with a Machine Learning Model (on Databricks)

Distance [miles]:   
 Estimated Time [min]: 10.180399283601904  
 Estimated Fare [dollars]: 8.72553449821577

### NYC Taxi analysis (data processing on Databricks)

dropoff_zip	x ▾
avg_trip_duration	x ▾



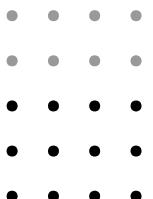


# QUESTIONS?

---



*Slides:*  
[bit.ly/pandas-spark](https://bit.ly/pandas-spark)



CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon** and infographics & images by **Freepik**