Applied Data Science with Microsoft Fabric

Tomaž Kaštrun, MVP

Sponsors


CONSORZIO
UNIVERSITARIO
DI PORDENONE
MOLTIPLICATORE DI VALORE


altitudo


beanTech
IT moves your business


[stesi]
Powered by Innovation


DATA
SATURDAYS


community

About

- Data scientist | BI Developer | Data Analyst
- SQL Server, SAS, R, Python, C#, SAP, SPSS
- 20+years experience MSSQL, DEV, BI, DM, DS
- Frequent community speaker, book author
- Avid coffee drinker & bicycle junkie

http://tomaztsql.wordpress.com

tomaz.kastrun@gmail.com

@tomaz_tsql

/in/tomaztsql

http://github.com/tomaztk

https://mvp.microsoft.com/PublicProfile/5002196

Session material:

github.com/tomaztk/microsoft-fabric

Agenda

- Fabric in General
- Tools comparison
- Tips from the field + Benchmarking
- Demo E2E:
  - Ingest data
  - Explore and visualize data (EDA)
  - Model building
  - Model registration with MLFlow
  - API

SaaS foundation

"The OneDrive for Data"

MICROSOFT FABRIC

Integration provides the following advantages:

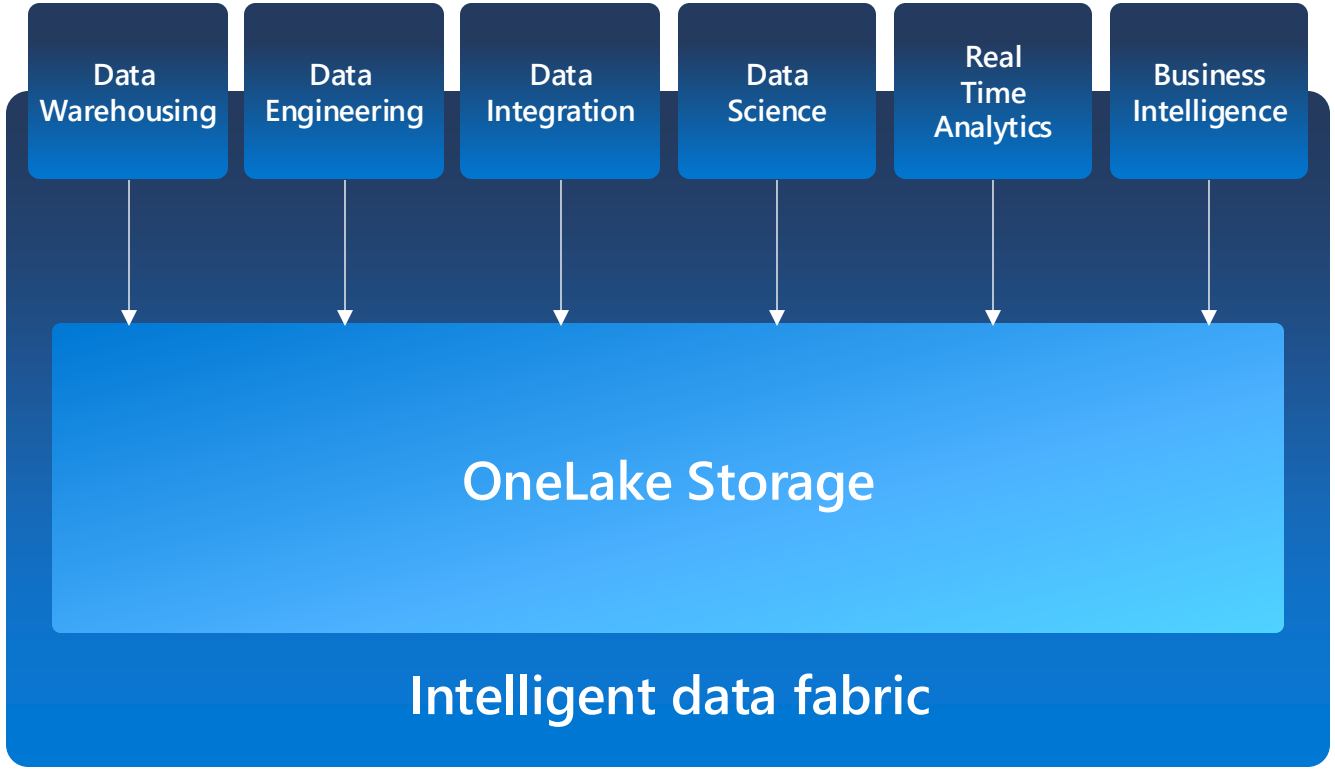
- centralized experience, administration and governance across all SaaS
- unified data lake (OneLake) to retain data
- shared experience and familiar environment across the board
- seamless integration and central configuration for all underlying services
- BI developers and data scientists can access and reuse all assets

DATA SATURDAYS

inOva community

OneLake for all Data

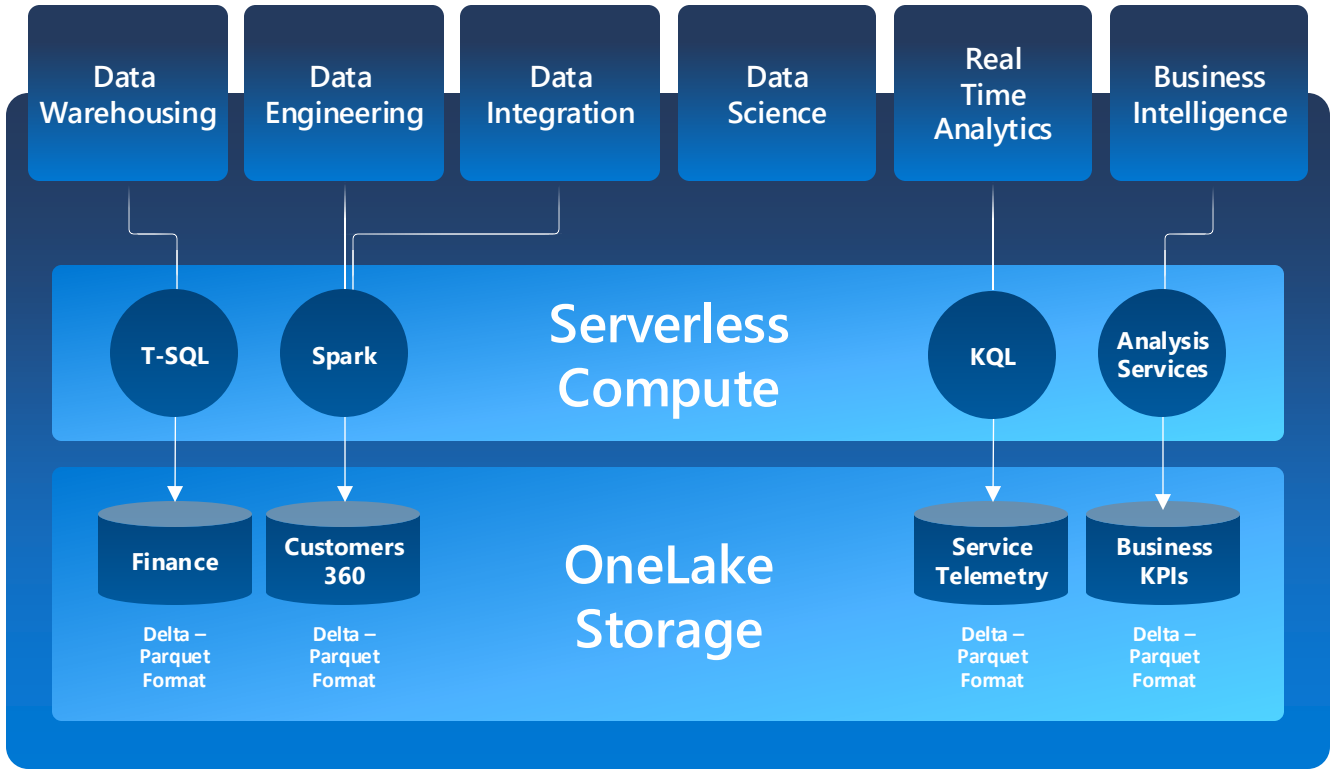
"The OneDrive for Data"



- A single SaaS lake for the whole organization
- Provisioned automatically with the tenant
- All workloads automatically store their data in the OneLake workspace folders
- All the data is organized in an intuitive hierarchical namespace
- The data in OneLake is automatically indexed for discovery, MIP labels, lineage, PII scans, sharing, governance and compliance

One Copy for all computes

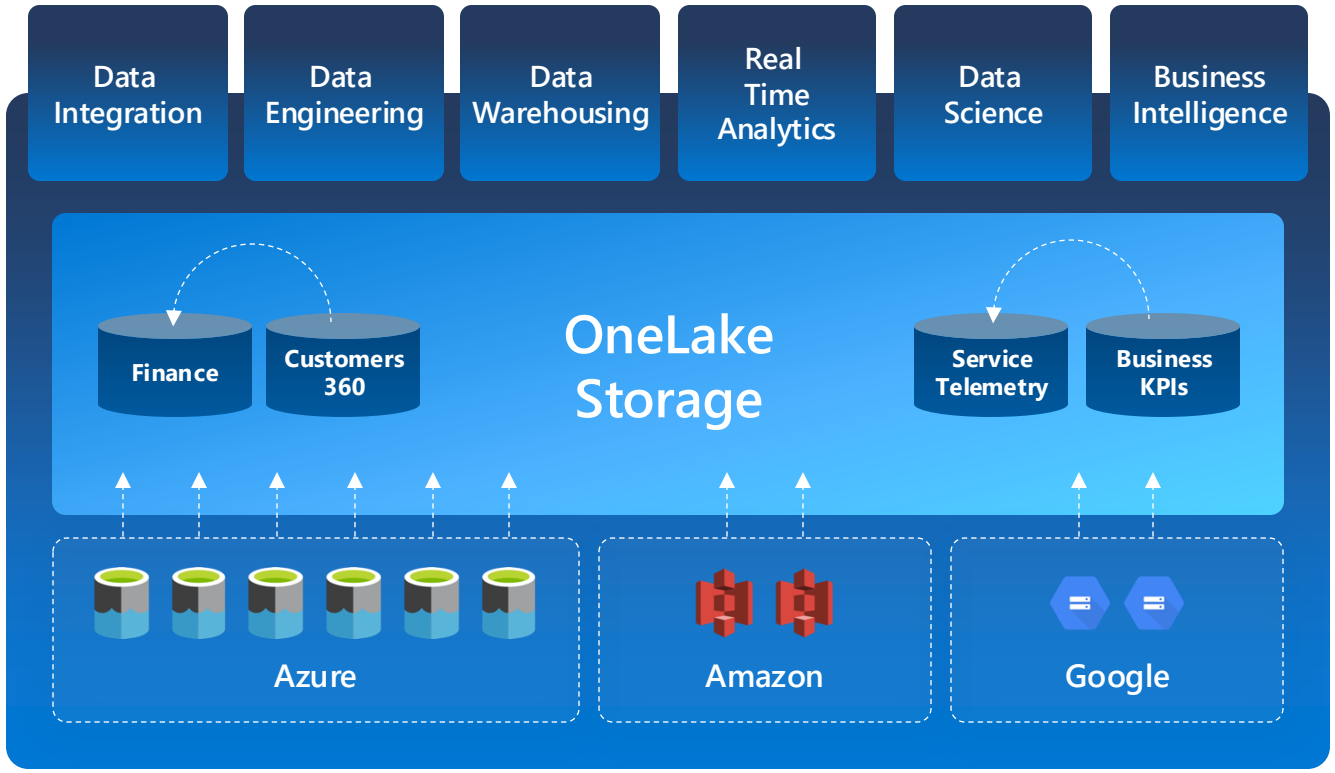
Real separation of compute and storage



- All the compute engines store their data automatically in OneLake
- The data is stored in a single common format
- Delta – Parquet, an open standards format, is the storage format for all tabular data in Fabric
- Once data is stored in the lake, it is directly accessible by all the engines without needing any import/export
- All the compute engines have been fully optimized to work with Delta Parquet as their native format
- Shared universal security model is enforced across all the engines

Taking One Copy to the next level

Shortcuts



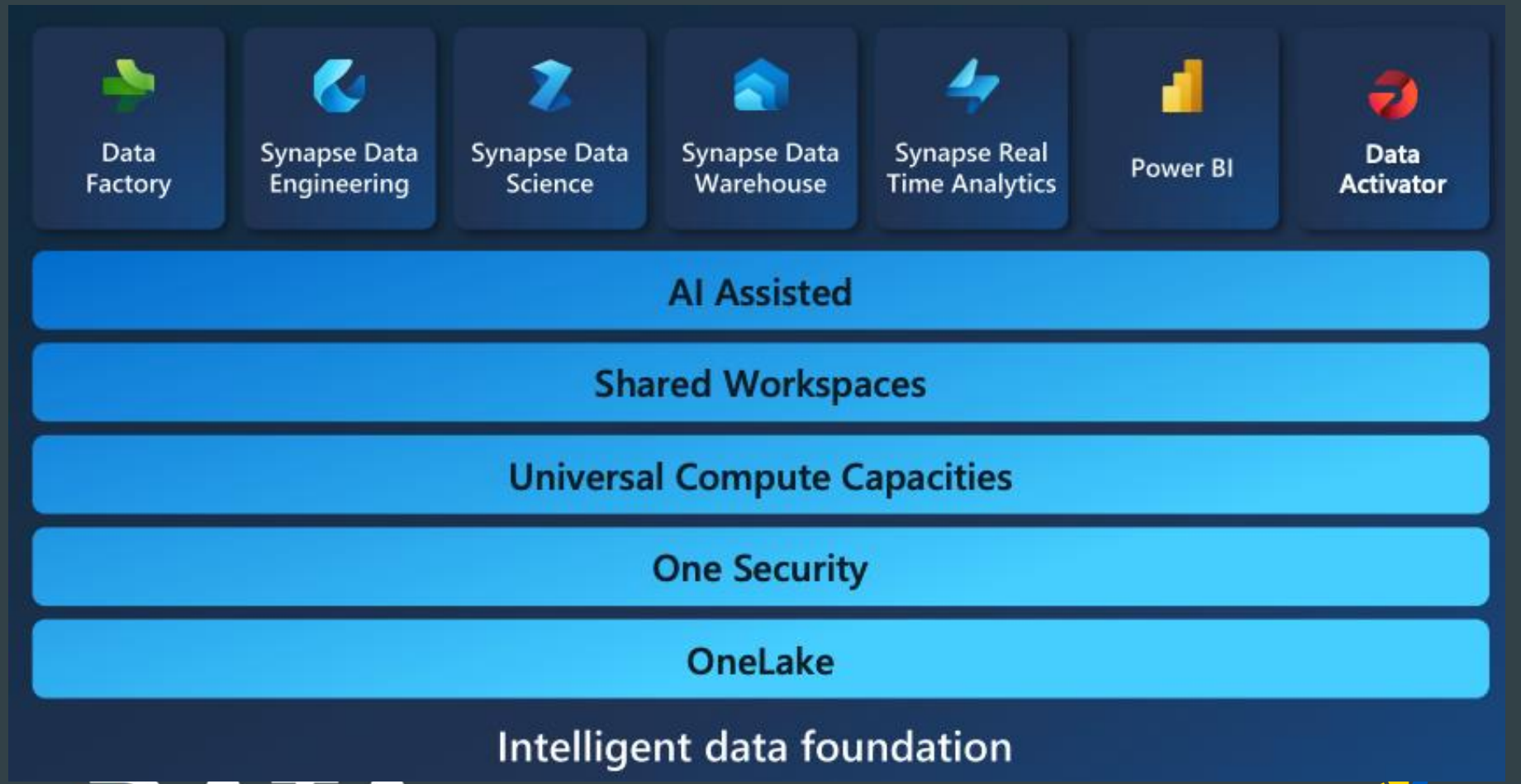
Sharing data in OneLake is as easy as sharing files in OneDrive, removing the needs for data duplication

With **shortcuts**, data throughout OneLake can be composed together without any data movement

Shortcuts also allow instant linking of data already existing in Azure and in other clouds, without any data duplication and movement, making **OneLake the first multi-cloud data lake**

With support for industry standard APIs, OneLake data can be directly accessed by any application or service

Microsoft Fabric in a gist with “All services”



Source: Microsoft

OneLake and lakehouse

Where home is!

Unified location!

- Built on top of ADLS Gen2
- Tenant-wide store
- One security with “one drive” feels like
- Simplifies for end user, with no need to “understand” the resource groups, RBAC, Resource manager, redundancy policies and regions

seamless integration and central configuration for all underlying services

DATA SATURDAYS

inn0va community

OneLake organisation

- is hierarhical and there is no

  need to up-front provisioning

- one per tenant and single file

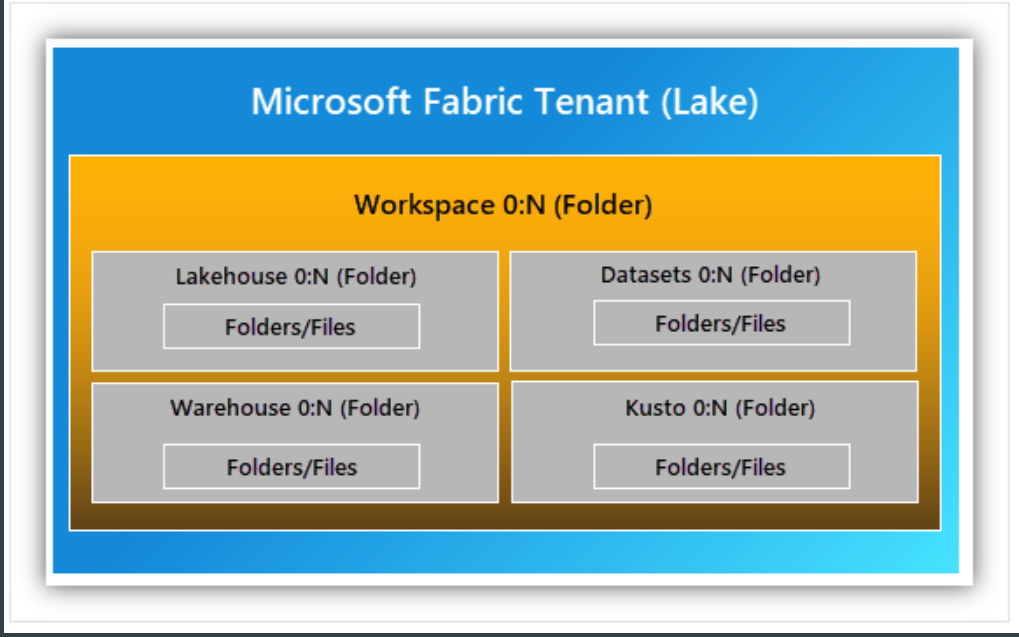
system namespace across users and regions

- divided into manageble containers

- user can create n-number of workspaces within tenant (WS can be understood as folder)

- every developer / business unit can create WS and ingest, process and analyse data, automatically the compute will be prewired to this WS

Use Shortcut feature to access data in Azure Data Lake storage without migration



.. and Fabric is - a complete analytics platform

Each analytics project depends on several supporting systems. These supporting systems often have a unique set of requirements and often require input from several other vendors. Integrating the different products of these vendors can be a difficult, fragile, and expensive task.

Microsoft Fabric alleviates this problem by providing teams with a single solution that offers a uniform user interface, architecture, and a variety of other tools necessary for insights to be extracted from data and presented.

.. and Fabric is Lake-centric and open

Fabric solves this problem by introducing a built-in software as a service (SaaS), multi-cloud data lake called **OneLake**.

Entirety of Fabric's workloads are wired into **OneLake**.

The built-in integration of **OneLake** helps to remove ubiquitous and disorganized data silos, which arise when team members configure their own segregated storage accounts.

OneLake offers the entire team a single, unified storage unit that makes discovering and sharing data simple.

.. and Fabric is Artificial intelligence

Since Copilot is integrated into every Microsoft Fabric data experience, users can utilize conversational language to:

- Build machine learning models
- Develop dataflows and data pipelines
- Generate code and entire functions
- Visualize results

It’s even possible for users to create their own custom conversational language experiences that combine their data **with Azure OpenAI** Service models and then publish them as plug-ins.

.. and Fabric is Empowerment for all business users

Teams within an organization aspire to drive a data-driven culture when everyone is empowered **to make better decisions using data**.

Microsoft Fabric helps **to foster this culture** by making analytics accessible to all.

Fabric is **deeply integrated** with the typical, everyday Microsoft 365 applications, for uncovering and applying insights.

.. and Fabric is cost reduction through unified capacities

Fabric alleviates *this* problem by reducing the **complexity of purchasing** and **managing resources**.

With Fabric, it's possible to purchase a single pool of compute to power all workloads (e.g., data integration, data science, etc.).

This **all-inclusive model** significantly reduces costs since any unused compute resources in a workload may be utilized by any of the workloads.

.. and Fabric is OneLake: The Heart of Fabric

At the heart of Microsoft Fabric lies OneLake. **OneLake** is effectively where all data utilized within Fabric is stored. As the name suggests, OneLake is a single, unified, logical data lake that's responsible for supporting all of your Fabric workloads.

A common comparison is often drawn between OneLake and OneDrive:

"Similar to how Office stores Word, Excel, and PowerPoint files in OneDrive, Fabric stores lakehouses, warehouses, and other items in OneLake."

.. and where does Fabric stand with others?

Recap key features:

- Unifies Data Platform (OneLake on ADLS2, Shortcuts)
- Lakehouse Architecture (Data warehouse, Data Lakehouse, Parquet, Delta)
- Data integration and Orchestration, Data Science, Real-time Analytics
- Power BI Integration
- Built-in governance and security
- Compute
- OSS technologies (Spark, MLFlow, Delta Lake)
- Collaborative workspace (cross-teams, notebooks, git integration)

..and Competition?

Feature / Product	Databricks	Microsoft Fabric	Azure Machine Learning	HDInsight
Primary Offering	Unified platform for data engineering, ML, and analytics	Data Fabric platform for data integration, analytics, and governance	Machine learning platform for building, training, and deploying models	Managed cloud service for big data analytics (Hadoop, Spark, etc.)
Data Engineering	5/5: Strong in ETL with Apache Spark, notebooks, and ML	4/5: Good integration but lacks Databricks' power	3/5: ML focused, limited for ETL	4/5: Built for big data, but not focused on ML
Machine Learning	5/5: Supports scalable ML with MLflow, AutoML, Spark MLlib	3/5: Limited advanced ML capabilities, better for data integration	5/5: ML-centric, strong AutoML, pipelines, distributed training	3/5: Supports ML but lacks advanced tools like Databricks
Analytics	5/5: Great for exploratory data analytics, Spark SQL	4/5: Strong reporting and Power BI integration	5/5: More ML-oriented, not ideal for interactive analytics	4/5: Focused on big data processing, not analytics per se
Programming Languages	5/5: Supports Python, SQL, Scala, R, Notebooks functionality	4/5: Primarily SQL and Power Query, Python, Spark	5/5: Python, R, Scala and SDK support for deep learning	5/5: Supports Python, Scala, and others for big data
Compute Scaling	5/5: Auto-scaling clusters, Pools, Photon, Multi	4/5: Elastic pools, easy scaling, but less compute-intensive	5/5: Scalable compute for distributed ML training, AK8s	4/5: Auto-scaling but more oriented to big data than ML
Integration	4/5: Integrates well with Azure, data lakes, MLflow	5/5: Best for Microsoft ecosystem, Power BI, Azure Synapse, DWH	5/5: Deep Azure ecosystem integration, supports Git, MLOps	4/5: Integrates well with Azure and other data services
Ease of Use	4/5: Excellent for data scientists, data engineers, but complex	5/5: User-friendly, low-code/no-code tools	4/5: Powerful, but setup can be complex for non-experts	3/5: More complex for beginners, focused on big data engineers
Maturity	5/5 On Market almost 10y	3/5 Rapid development, changing focus	5/5 Mature with 10+Years	5/5 Mature

Comparison – Strengths vs. Weaknesses

1. **Databricks**:
    1. **Strengths**: Best for data engineering, real-time analytics, and machine learning with a strong Apache Spark backbone and excellent multi-cloud support. It offers strong integration with big data tools and machine learning workflows (e.g., MLflow) for automated pipelines.
    2. **Weaknesses**: Can be complex and costly at a large scale.
2. **Microsoft Fabric**:
    1. **Strengths**: Excels in data governance, reporting (with Power BI integration), and ease of use with low-code tools. It integrates deeply with the Microsoft ecosystem.
    2. **Weaknesses**: Limited advanced machine learning capabilities compared to Databricks and Azure ML.
3. **Microsoft Azure Machine Learning**:
    1. **Strengths**: Best for advanced machine learning workflows, AutoML, MLOps, and hyperparameter tuning. It has deep Azure integration and strong deployment capabilities for ML models.
    2. **Weaknesses**: More focused on ML and less suitable for general data engineering tasks.
4. **HDInsight**:
    1. **Strengths**: Excellent for big data batch processing and Hadoop/Spark workloads, offering scalability at a lower cost.
    2. **Weaknesses**: Lacks modern machine learning and data engineering features seen in Databricks or Azure ML. Installation process and maintanenc

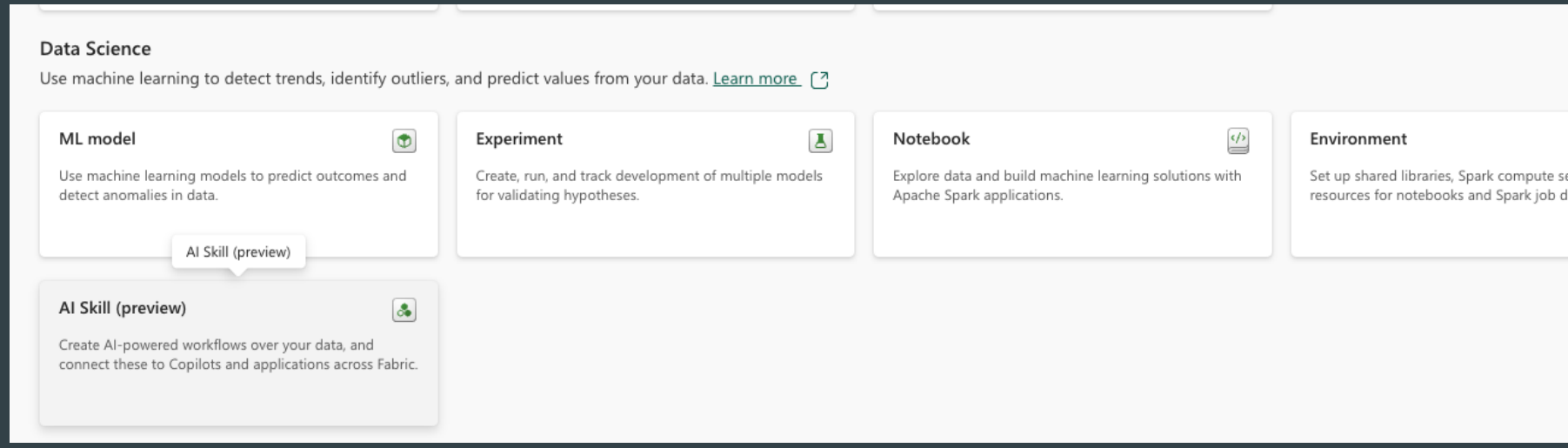
Tips from the field

- Compute
- Languages
- One version of data == one copy
- Transactional and analytical system and data “copying”; ACID
- Spark based and file system management
- SynapseML for PySpark!

Benchmarking Demo --> Pandas, Koalas, Polars are all nice and cute bear-like animals, but go with PySpark!

Data science in Fabric ☺

Fabric – Data Science assets



Ingest data

Benchmark and Comparison

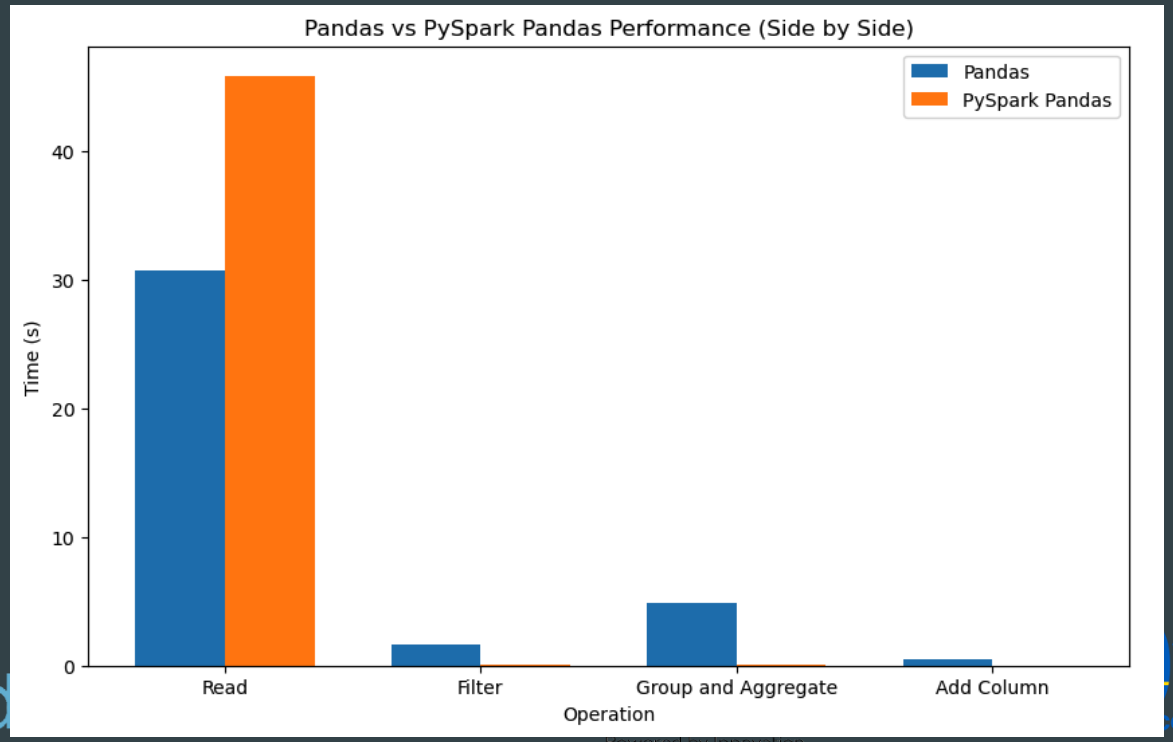
Task	Pandas (Python)	PySpark Pandas (Koalas)
Data Loading (100M rows)	Slower: Pandas reads data from a CSV or Parquet file into memory. Loading large datasets can result in high memory consumption and may even fail if the dataset is too large for the available RAM.	Faster: PySpark Pandas is distributed and can load data faster by splitting the work across multiple nodes. Suitable for datasets exceeding memory limits of a single machine.
Filtering (e.g., amount > 500)	Fast for smaller datasets, but performance degrades as the dataset size grows (100M rows may take time). Single-threaded operation.	Significantly faster as operations are distributed across nodes. Spark's parallelism optimizes filtering, even on large datasets.
Grouping and Aggregation	Performance drops on large datasets due to memory limitations and single-threaded execution. Pandas groups and aggregates data in memory.	Spark-based grouping and aggregation are highly optimized for distributed computing, leading to faster performance even with massive datasets.
Adding New Columns	Quick for small-to-medium datasets but slower for larger datasets due to memory limits.	More efficient when scaling because operations are distributed across the cluster. Adding new columns is faster with large datasets.
Handling Null Values	Quick for small datasets, but performance suffers with large datasets.	Spark handles missing values efficiently at scale, outperforming Pandas for larger datasets.

Ingest data

Benchmark and Comparison

Pandas is optimized for small to medium datasets, working best on single machines with data that fits in memory. It has a more intuitive API but struggles when handling datasets exceeding a few million rows due to memory and CPU limitations.

PySpark Pandas (Koalas) is designed for distributed processing across clusters. This allows for faster data operations on large datasets. It is ideal for handling big data and scales out efficiently, making it faster than Pandas in most large-scale scenarios.

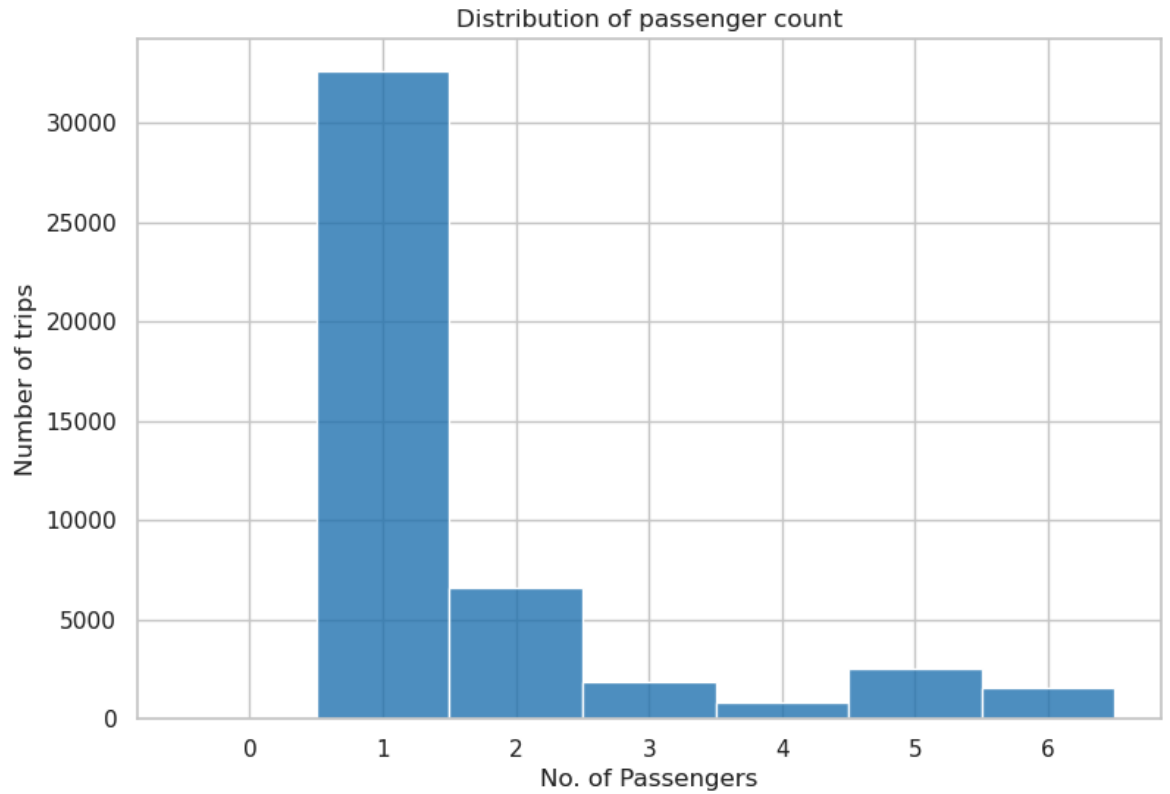


Explore and visualize data (EDA)

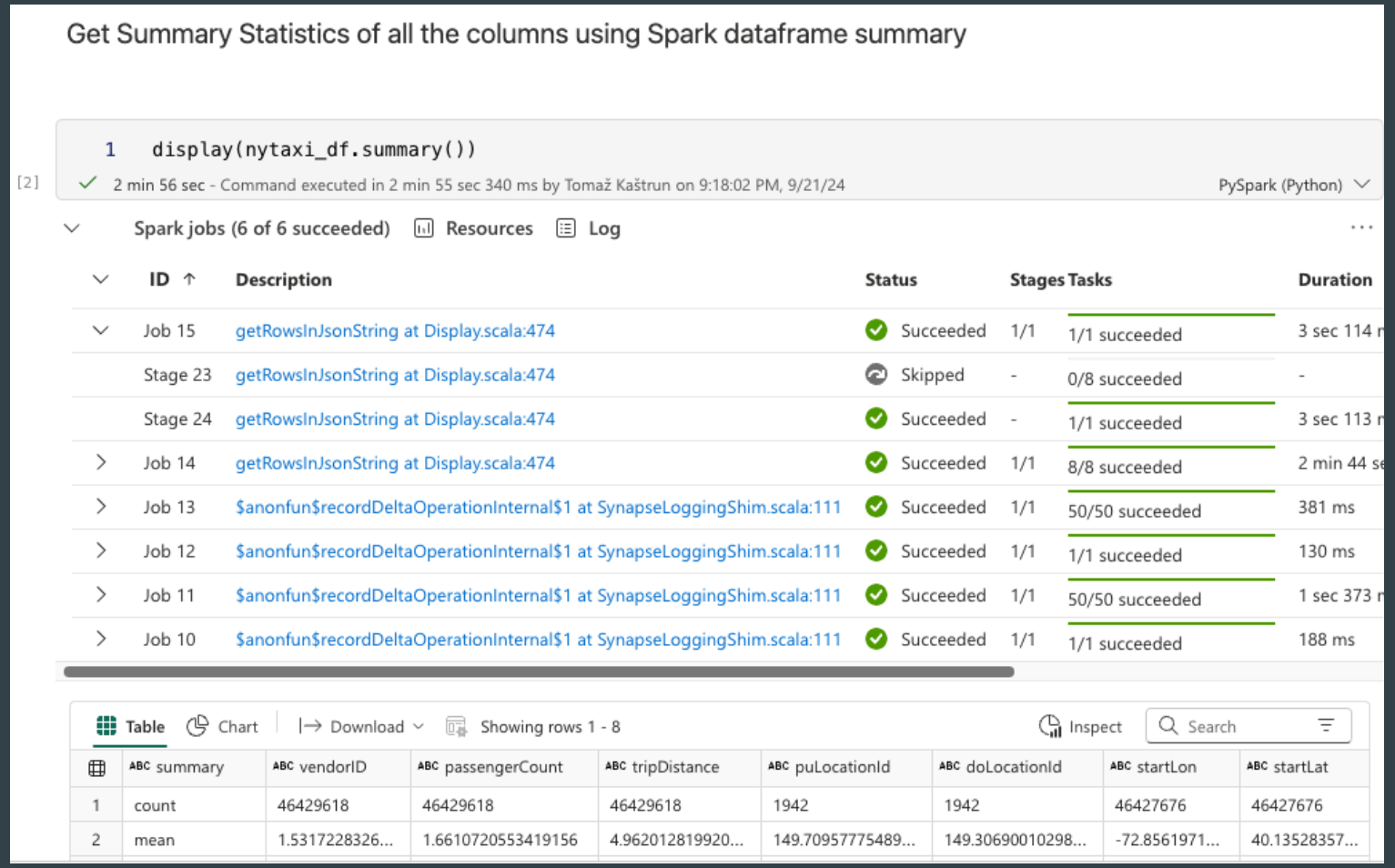
```
1  sns.histplot(data=sampled_df, x="passengerCount", stat="count", discrete=True)
2  plt.title("Distribution of passenger count")
3  plt.xlabel('No. of Passengers')
4  plt.ylabel('Number of trips')
```

[6] ✓ PySpark (Python)

```
Text(0, 0.5, 'Number of trips')
```



Data Cleaning and using Apache Spark



Details for Stage 24 (Attempt 0)

Resource Profile Id: 0
Total Time Across All Tasks: 3 s
Locality Level Summary: Node local: 1
Shuffle Read Size / Records: 6.8 MiB / 8
Associated Job Ids: 15

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Summary Metrics for 1 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	3 s	3 s	3 s	3 s	3 s
GC Time	0.4 s	0.4 s	0.4 s	0.4 s	0.4 s
Shuffle Read Size / Records	6.8 MiB / 8	6.8 MiB / 8	6.8 MiB / 8	6.8 MiB / 8	6.8 MiB / 8

- Aggregated Metrics by Executor

Tasks (1)

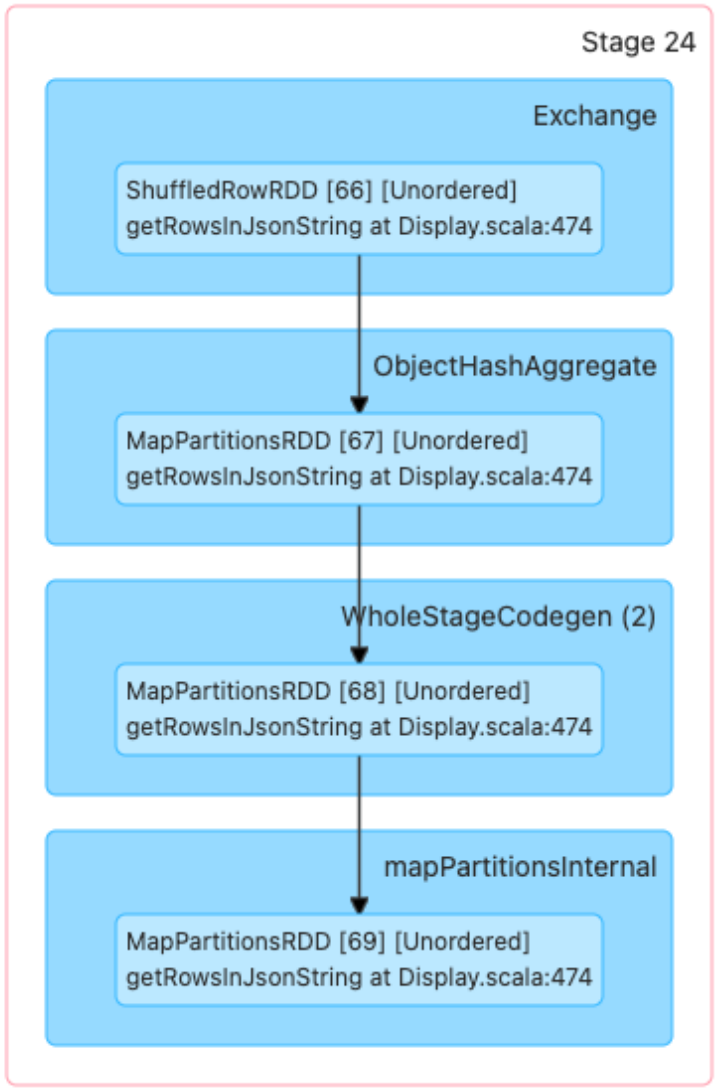
Show 20 entries Search:

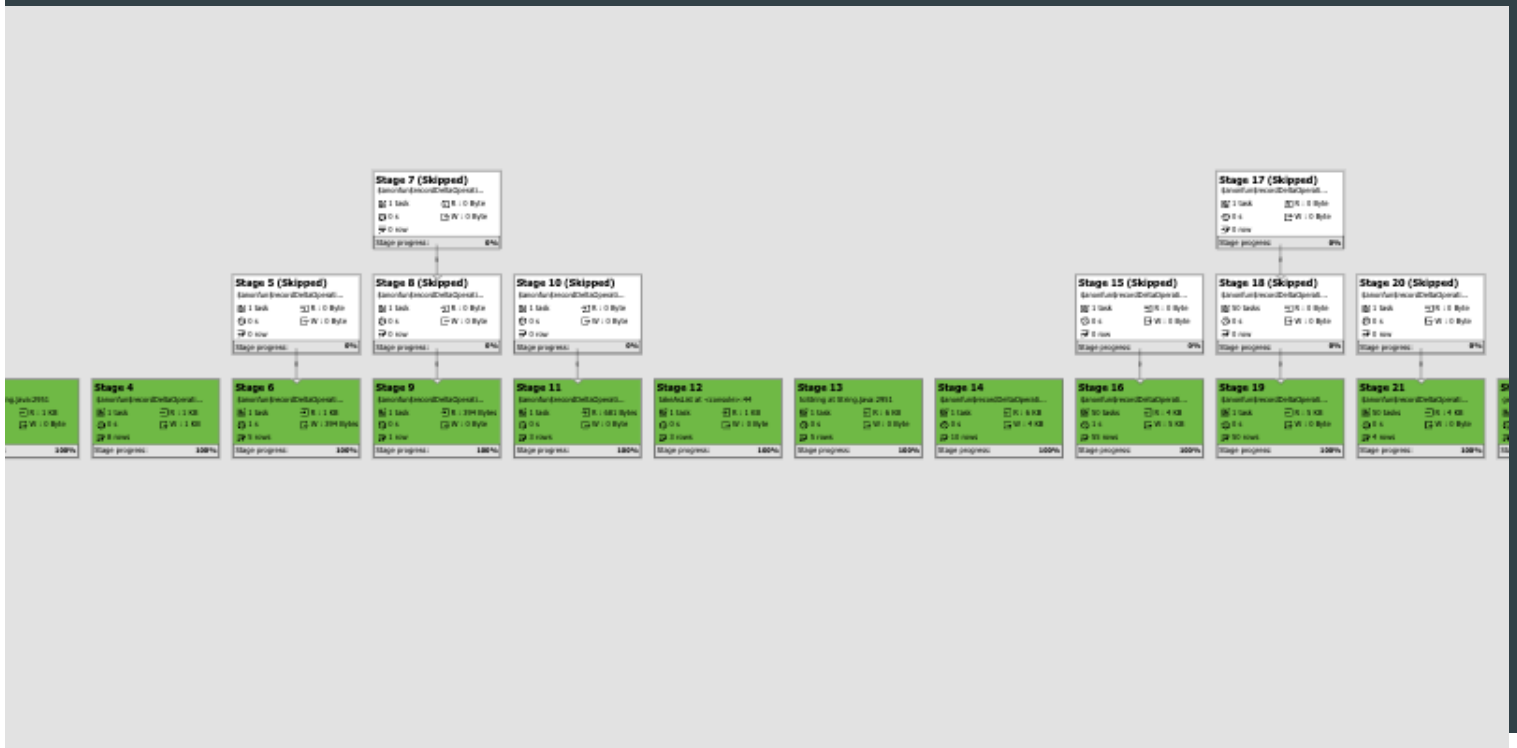
Index	Task ID	Attempt	Status	Locality level	Executor ID	Host	Logs	Launch Time	Duration	GC Time	Shuffle Read Size / Records	Error
0	121	0	SUCCESS	NODE_LOCAL	1	vm-e2886599	stdout stderr	2024-09-21 21:17:56	3 s	0.4 s	6.8 MiB / 8	

Showing 1 to 1 of 1 entries Previous 1 N

Resource Profile Id: 0
Total Time Across All Tasks: 3 s
Locality Level Summary: Node local: 1
Shuffle Read Size / Records: 6.8 MiB / 8
Associated Job Ids: 15

▾ DAG Visualization





Model registratioin with MLFlow

```
1   TRAIN_TEST_SPLIT = [0.75, 0.25]
2   train_df, test_df = training_df.randomSplit(TRAIN_TEST_SPLIT, seed=SEED)
3
4   # Cache the dataframes to improve the speed of repeatable reads
5   train_df.cache()
6   test_df.cache()
7
8   print(f"train set count:{train_df.count()}")
9   print(f"test set count:{test_df.count()}")
10
11  categorical_features = ["storeAndFwdFlag","timeBins","vendorID","weekDayName
12  numeric_features = ['passengerCount', "tripDistance"]
```

[3] ✓ 1 min 2 sec - Command executed in 1 min 1 sec 864 ms by Tomaž Kaštrun on 9:26:34 PM, 9/21/24

Spark jobs (6 of 6 succeeded) Resources Log

ID	Description	Status	Stage
Job 15	count at NativeMethodAccessorImpl.java:0	Succeeded	1/1
Job 14	count at NativeMethodAccessorImpl.java:0	Succeeded	1/1
Job 13	count at NativeMethodAccessorImpl.java:0	Succeeded	1/1
Job 12	count at NativeMethodAccessorImpl.java:0	Succeeded	1/1
Job 11	$anonfun$recordDeltaOperationInternal$1 at SynapseLoggingShim.scala:111	Succeeded	1/1
Job 10	$anonfun$recordDeltaOperationInternal$1 at SynapseLoggingShim.scala:111	Succeeded	3/3

Model building

Fit the defined pipeline on the training dataframe and generate predictions on the test dataset

```
if mlflow.active_run() is None:
    mlflow.start_run()
run = mlflow.active_run()
print(f"Active experiment run_id: {run.info.run_id}")
lg_pipeline = lgbm_pipeline(categorical_features,numeric_features,LGBM_PARAMS)
lg_model = lg_pipeline.fit(train_df)

# Get Predictions
lg_predictions = lg_model.transform(test_df)
## Caching predictions to run model evaluation faster
lg_predictions.cache()
print(f"Prediction run for {lg_predictions.count()} samples")
```

- Command executed in 8 sec 329 ms by Tomaž Kaštrun on 9:33:56 PM, 9/21/24

Compute Model Statistics for evaluating performance of the trained LightGBMRegressor model

```
from synapse.ml.train import ComputeModelStatistics
lg_metrics = ComputeModelStatistics(
    evaluationMetric="regression", labelCol="tripDuration", scoresCol="prediction"
).transform(lg_predictions)
display(lg_metrics)
```

- Command executed in 851 ms by Tomaž Kaštrun on 9:28:19 PM, 9/21/24

Model Assessment

```
1  lg_metrics_tn = ComputeModelStatistics(
2      evaluationMetric="regression", labelCol="tripDuration", scoresCol="prediction"
3  ).transform(lg_predictions_tn)
4  display(lg_metrics_tn)
```

Table | Chart | Download | Showing rows 1 - 1

	1.2 mean_squared_error	1.2 root_mean_squared_error	1.2 R^2	1.2 mean_absolute_error
1	25.36957482259526	5.036821897049295	0.7641619...	3.239484827129292

Model Consumption with API

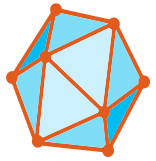
Get the trained and registered model to generate predictions

```python
import mlflow
from pyspark.ml.feature import OneHotEncoder, VectorAssembler, StringIndexer
from pyspark.ml import Pipeline
from synapse.ml.core.platform import *
from synapse.ml.lightgbm import LightGBMRegressor

## Define run_uri to fetch the model
# run_uri = "<Enter the run_uri from module 04 here>"
run_uri = "runs:/b1bb91e0-55cf-4302-86d5-53b8aba63d13/nyctaxi_tripduration_lightgbm"
loaded_model = mlflow.spark.load_model(run_uri, dfs_tmpdir="Files/tmp/mlflow")
```

PySpark (Pyt

SynapseML and Microsoft AI services



Distributed ML
Model Training

MLflow
support

Cognitive
Services

OpenAI
LLMs

Thanks!

http://tomaztsql.wordpress.com

tomaz.kastrun@gmail.com

@tomaz_tsql

/in/tomaztsql

http://github.com/tomaztk

https://mvp.microsoft.com/PublicProfile/5002196