

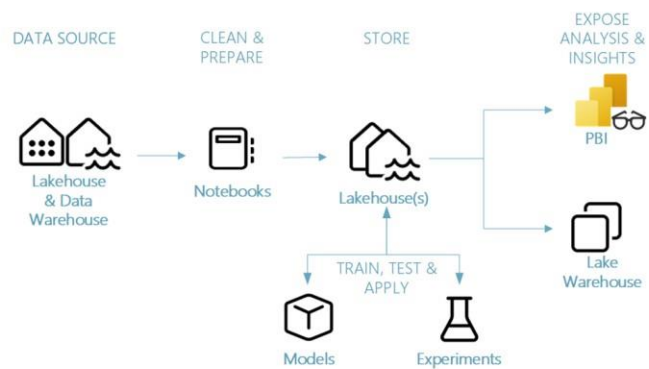
Fabric Data Science Tutorial

Data science

Published: July 2023

Data Science

- **Data Source:**
Access data from multiple sources – eg. Lakehouse, Data Warehouse, PBI Dataset
- **Explore, Clean & Prepare:**
Perform data transformation, exploration and featurization by leveraging built-in experiences on Spark as well as Python based tools like Data Wrangler and SemPy Library.
- **Models & Experiments:**
Iterate, build and track Machine Learning experiments and models using MLFlow. Leverage data science capabilities for model prediction at scale (PREDICT) to gain and share business insights.
- **Storage:**
Store data and insights in Lakehouse(s) (reference DE scenario)
- **Expose Analysis & Insights:**
Collaborate with other by sharing your findings and insights via Notebook, PBI Report and Lake Warehouse.



Contents

Introduction	3
Prerequisites.....	4
The Data Science end-to-end scenario	4
Sample Dataset.....	5
Import tutorial notebooks.....	6
Attach a lakehouse to the notebooks.....	8
Module 1: Ingest data into Fabric lakehouse using Apache Spark	9
Module 2: Explore and visualize data using notebooks.....	11
Module 3: Perform Data Cleansing and preparation using Apache Spark.....	19
Module 4: Train and register machine learning models.	21
Module 5: Perform batch scoring and save predictions to lakehouse.	26
Module 6: Create a Power BI report to visualize predictions.....	28
Summary	33

Introduction

Note – If you are new to Fabric, we would encourage you to go through this documentation to get an overview of different Fabric concepts and features: [Fabric - Overview](#)

The lifecycle of a Data science project typically includes (often, iteratively) the steps listed below:

- Business understanding
- Data acquisition
- Data exploration, cleansing, preparation, and visualization
- Model training and experiment tracking
- Model scoring and generating insights.

The goals and success criteria of each stage listed above depend on collaboration, data sharing and documentation. The Fabric Data science experience consists of multiple native-built features that enable collaboration, data acquisition, sharing, and consumption in a seamless way.

In this tutorial, you will take the role of a data scientist who has been given the task to explore, clean and transform a dataset containing taxicab trip data, and build a machine learning model to predict trip duration at scale on a large dataset.

In this tutorial, you will learn to perform the following activities:

- 1) Use the Fabric notebooks for data science scenarios.
- 2) Ingest data into Fabric lakehouse using Apache Spark.
- 3) Load existing data from the lakehouse delta tables.
- 4) Clean and transform Data using Apache Spark.
- 5) Create experiments and runs to Train a machine learning model.
- 6) Register and track trained models using MLflow and the Fabric UI.
- 7) Run scoring at scale and save predictions and inference results to the lakehouse.
- 8) Visualize predictions in PowerBI using DirectLake.

Prerequisites

1. Power BI Premium subscription. For more information, [How to purchase Power BI Premium](#).
2. A Power BI Workspace with assigned premium capacity.
3. An existing Fabric lakehouse. Create a lakehouse by following, [How to Create a lakehouse](#) in the lakehouse tutorial document.

The Data Science end-to-end scenario

In this tutorial we will showcase a simplified end-to-end data science scenario that involves:

- 1) Ingesting data from an external data source.
- 2) Data exploration and visualization.
- 3) Data cleansing, preparation, and feature engineering.
- 4) Model training and evaluation.
- 5) Model batch scoring and saving predictions for consumption.
- 6) Visualizing prediction results.

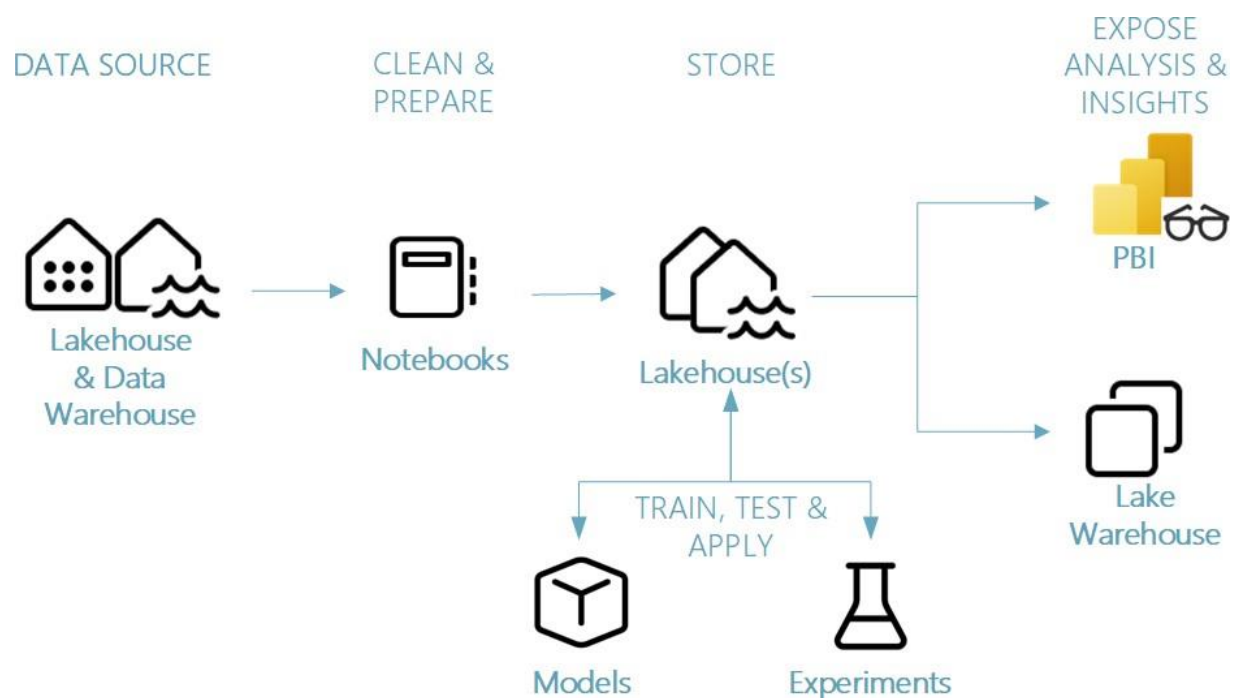


Figure 1: The Data Science end-to-end scenario

Different components of Data science scenario

Data Sources – Fabric makes it easy and quick to connect to Azure Data Services, other cloud platforms, and on-premises data sources to ingest data from. Using Fabric Notebooks you can ingest data from the inbuilt Lakehouse, Data Warehouse, Power BI Datasets as well as various Apache Spark and Python supported custom data sources. In this document we will focus on ingesting and loading data from Lakehouse.

Explore, Clean & Prepare – The Data Science experience on Fabric supports data cleansing, transformation, exploration and featurization by leveraging built-in experiences on Spark as well as Python based tools like Data Wrangler and SemPy Library. This tutorial will showcase data exploration using python library seaborn and data cleansing and preparation using Apache Spark .

Models & Experiments – Fabric enables you to train, evaluate and score machine learning models by using built-in Experiment and Model artifacts with seamless integration with [MLflow](#) for experiment tracking and model registration/deployment. Fabric also features capabilities for model prediction at scale (PREDICT) to gain and share business insights.

Storage – Fabric standardizes on [Delta Lake](#), that means all the engines of Fabric can interact with the same dataset stored in lakehouse. This storage layer allows you to store both structured and unstructured data that support both file-based storage as well as tabular format. The datasets and files stored can be easily accessed via all Fabric workload artifacts like notebooks and pipelines.

Expose Analysis and Insights – Data from Lakehouse can be consumed by Power BI, industry leading business intelligence tool, for reporting and visualization. Data persisted in the lakehouse can also be visualized in notebooks using Spark or Python native visualization libraries like matplotlib, seaborn, plotly, and more. Data can also be visualized using SemPy library that supports built-in rich, task-specific visualizations for the semantic data model, for dependencies and their violations, and for classification and regression use cases.

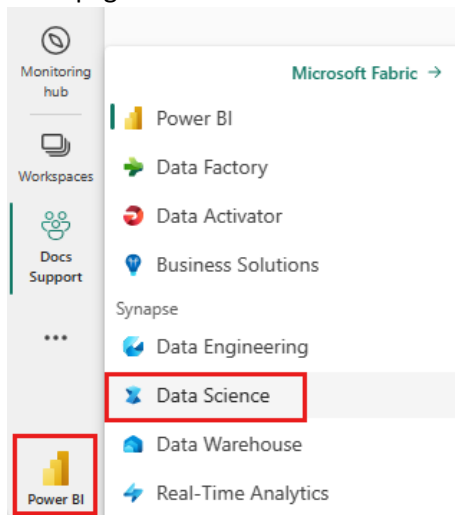
Sample Dataset

In this tutorial we will use the [NYC Taxi and Limousine yellow dataset](#), which is a large-scale dataset containing taxi trips in the city from 2009 to 2018. The dataset includes various features such as pick-up and drop-off dates, times, locations, fares, payment types, and passenger counts. The dataset can be used for various purposes such as analyzing traffic patterns, demand trends, pricing strategies, and driver behavior.

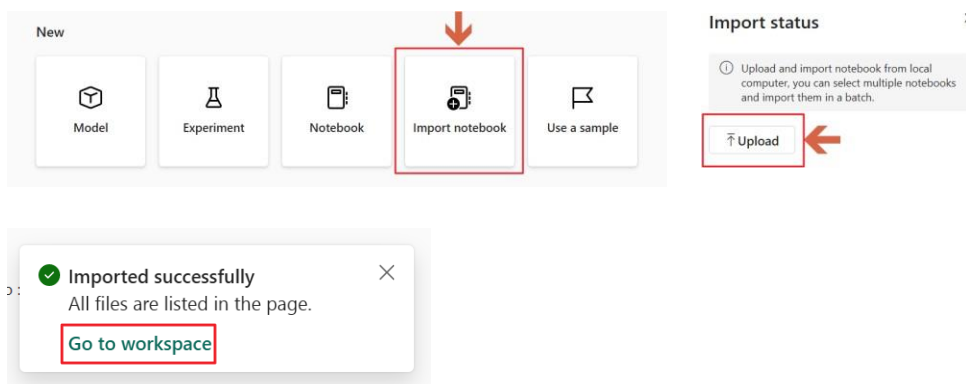
Import tutorial notebooks.

We will utilize the notebook artifact in the Data Science experience to demonstrate various Fabric capabilities. The notebooks are available as jupyter notebook files that can be imported to your Fabric-enabled workspace.






1. Download the notebooks(.ipynb) files for this tutorial from the Scripts folder [Data Science Tutorial Source Code](#).
2. Switch to the Data Science experience using the workload switcher icon at the left corner of your homepage.



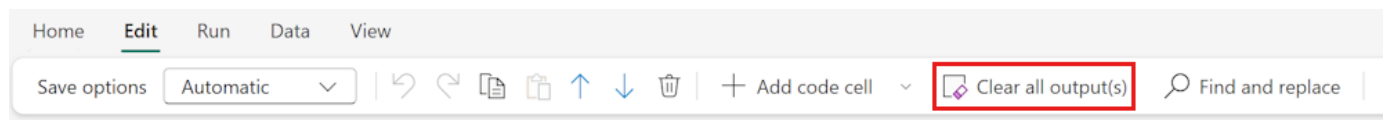
3. On the Data science experience homepage click on the **Import Notebook** button and upload the notebook files for modules 1- 5 downloaded in step 1. Once the notebooks are imported, click on the **Go to workspace** link in the import dialog box.



4. The imported notebooks would now be available in your workspace for use.

<div> <div>+ New</div> <div> <div>↑ Upload</div> <div>Create deployment pipeline</div> <div>Create app</div> <div>...</div> </div> </div>			
	Name	Type	Owner
	01 - Ingest data into Trident lakehouse using Apac...	Notebook	
	02 - Explore and Visualize Data using Notebooks	Notebook	
	03 - Perform Data Cleansing and preparation using...	Notebook	
	04 - Train and track machine learning models	Notebook	
	05 - Perform batch scoring and save predictions to ...	Notebook	

- If the imported notebook includes output, select the **Edit** menu, then select **Clear all outputs**.



Attach a lakehouse to the notebooks.

To demonstrate Fabric lakehouse features, the first five modules in this tutorial require attaching a default lakehouse to the notebooks. Below are the steps to add an existing lakehouse to a notebook in a Fabric-enabled workspace.

1. Open the notebook for the first module *“01 Ingest data into Lakehouse using Apache Spark”* in the workspace.
2. Click on the add lakehouse button on the left pane and select **Existing lakehouse** to open the **Data hub** dialog box.
3. Select the workspace and the lakehouse you intend to use with these tutorials and click the add button.
4. Once a lakehouse is added it will be visible in the lakehouse pane on notebook UI where *Tables* and *Files* stored in the lakehouse can be viewed.

Note: The steps defined above need to be performed for each notebook, before executing all notebook accompanying this tutorial

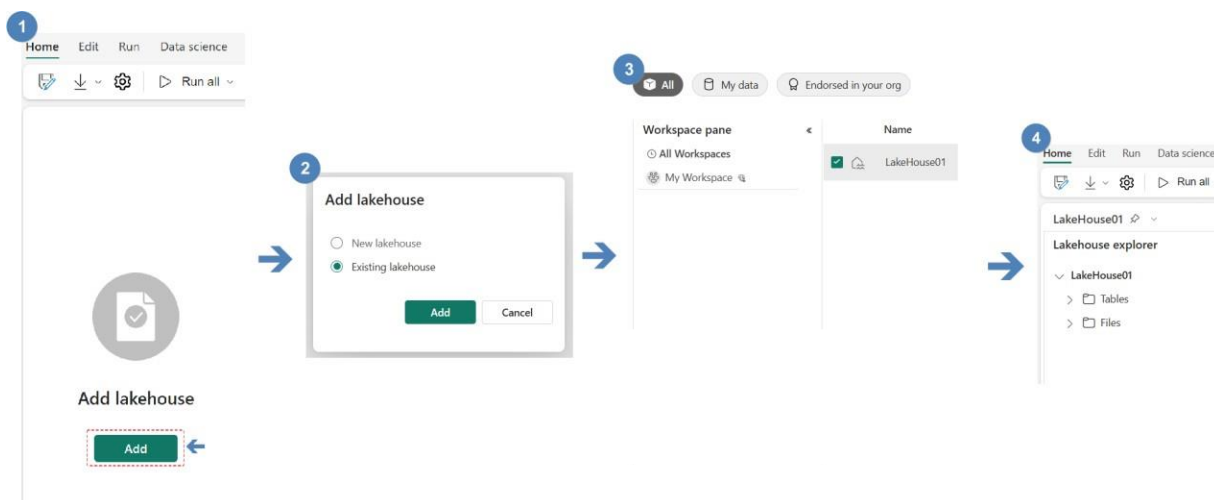


Figure 2: Attach a lakehouse to notebook.

Module 1: Ingest data into Fabric lakehouse using Apache Spark

In this module, we will ingest the NYC Taxi & Limousine Commission - yellow taxi trip Dataset to demonstrate data ingestion into Fabric lakehouses in delta lake format.

Lakehouse: A lakehouse is a collection of files/folders/tables that represent a database over a data lake used by the Spark engine and SQL engine for big data processing and that includes enhanced capabilities for ACID transactions when using the open-source Delta formatted tables.

Delta Lake: Delta Lake is an open-source storage layer that brings ACID transactions, scalable metadata management, and batch and streaming data processing to Apache Spark. A Delta Lake table is a data table format that extends Parquet data files with a file-based transaction log for ACID transactions and scalable metadata management.

In the following steps, you'll use the Apache spark to read data from **Azure Open Datasets** containers and write data into Fabric lakehouse delta table. [Azure Open Datasets](#) are curated public datasets that you can use to add scenario-specific features to machine learning solutions for more accurate models. Open Datasets are in the cloud on Microsoft Azure Storage and can be accessed by a variety of methods including, Apache Spark, REST API, Datafactory and other tools.

Note: The python commands/script used in each step of this tutorial can be found in the accompanying notebook: *01 Ingest data into Fabric lakehouse using Apache Spark*

1. In the first step of this module, we read data from “azureopendatastorage” storage container using anonymous since the container has public access. We will specifically load yellow cab data by specifying the directory and filter the data by year(puYear) and month(puMonth). In this tutorial, we will try to minimize the amount of data ingested and processed to speed up the execution.

```
# Azure storage access info for open datasets yellow cab
storage_account = "azureopendatastorage"
container = "nyctlc"

sas_token = r"" # Blank since container is Anonymous access

# Set Spark config to access blob storage
spark.conf.set("fs.azure.sas.%s.%s.blob.core.windows.net" % (container, storage_account), sas_token)

dir = "yellow"
year = 2016
months = "1,2,3,4"
wasbs_path = f"wasbs://{container}@{storage_account}.blob.core.windows.net/{dir}"
df = spark.read.parquet(wasbs_path)

# Filter data by year and months
filtered_df = df.filter(f"puYear = {year} AND puMonth IN ({months})")
```

2. Next, we will set spark configurations to enable Verti-Parquet engine and Optimize delta writes.

• **~VOrder** – Fabric includes Microsoft’s VertiParquet engine. VertiParquet writer optimizes the Delta Lake parquet files resulting in 3x-4x compression improvement and up to 10x performance acceleration over Delta Lake files not optimized using VertiParquet while still maintaining full Delta Lake and PARQUET format compliance.

• **Optimize write** – Spark in Fabric includes an Optimize Write feature that reduces the number of files written and targets to increase individual file size of the written data. It dynamically optimizes files during write operations generating files with a default 128 MB size. The target file size may be changed per workload requirements using configurations.

These configs can be applied at a session level(as `spark.conf.set` in a notebook cell) as demonstrated in the following code cell, or at workspace level which is applied automatically to all spark sessions created in the workspace. In this tutorial we will set these configurations using the code cell.

```
spark.conf.set("spark.sql.parquet.vorder.enabled", "true") # Enable Verti-Parquet write
spark.conf.set("spark.microsoft.delta.optimizeWrite.enabled", "true") # Enable automatic delta optimized write
```

Tip: The workspace level Apache Spark configurations can be set at:

- Workspace settings >> Data Engineering/Science >> Spark Compute >> Spark Properties >> Add.

3. In the next step, perform a spark dataframe write operation to save data into a lakehouse table named `nyctaxi_raw`.

```
1 table_name = "nyctaxi_raw"
2 filtered_df.write.mode("overwrite").format("delta").save(f"Tables/{table_name}")
3 print(f"Spark dataframe saved to delta table: {table_name}")
```

✓ 1 min 0 sec - Command executed in 1 min 1 sec 297 ms by

>  Spark jobs (9 of 9 succeeded)

Spark dataframe saved to delta table: nyctaxi_raw

Once the dataframe has been saved, you can navigate to the attached lakehouse artifact in your workspace and open the lakehouse UI to preview data in the `nyctaxi_raw` table created above.

Home Lakehouse

Get data New Power BI dataset Open notebook

A SQL endpoint for SQL querying and a default dataset for reporting were created and will be updated with any tables added to the lakehouse. You can access the SQL endpoint using the dropdown.

Lakehouse explorer

DSLakehouse

Tables

nyctaxi_raw

Files

nyctaxi_raw Showing <1000> rows

	vendorID	tpepPickup...	tpepDropo...	passenger...	tripDistance	startLon	startLat	endLon	endLat	rateCodeId	storeAndF...	paymentTy...
1	1	4/19/2016 ...	4/19/2016 ...	1	16.3	-73.968658...	40.7506904...	-73.787170...	40.6475601...	2	N	1
2	1	4/28/2016 ...	4/28/2016 ...	1	11.7	-73.984222...	40.7501182...	-73.806068...	40.6831741...	2	N	1
3	2	4/1/2016 5...	4/1/2016 5...	1	15.71	-74.000686...	40.7297401...	-73.788841...	40.6413917...	2	N	1
4	2	4/20/2016 ...	4/20/2016 ...	1	20.23	-73.790290...	40.6467132...	-73.972930...	40.7884292...	2	N	1
5	1	4/29/2016 ...	4/29/2016 ...	1	28.4	-73.796669...	40.6447296...	-73.952629...	40.7815513...	2	N	1
6	1	4/14/2016 ...	4/14/2016 ...	1	19.7	-73.790191...	40.6468887...	-73.972602...	40.7975616...	2	N	1
7	1	2/11/2016 ...	2/11/2016 ...	1	18.4	-73.781761...	40.6446990...	-73.989669...	40.7440376...	2	N	1
8	1	2/23/2016 ...	2/23/2016 ...	1	19	-73.776771...	40.6454811...	-73.999145...	40.7437248...	2	N	1
9	1	2/25/2016 ...	2/25/2016 ...	1	19.6	-73.790237...	40.6468391...	-73.969985...	40.7888641...	2	N	1
10	2	2/11/2016 ...	2/11/2016 ...	1	17.55	-73.789466...	40.6475028...	-73.976013...	40.7638473...	2	N	1
11	1	4/25/2016 ...	4/25/2016 ...	1	20.6	-73.783462...	40.6486053...	-73.938179...	40.8508338...	2	N	1
12	1	4/12/2016 ...	4/12/2016 ...	1	18.5	-73.781913...	40.6446914...	-73.987388...	40.7309532...	2	N	1
13	1	4/25/2016 ...	4/25/2016 ...	1	0.6	-73.801559...	40.6466636...	-73.790534...	40.6448554...	2	N	1
14	1	4/8/2016 4...	4/8/2016 6...	1	22	-73.790145...	40.6436042...	-74.006416...	40.7081832...	2	N	1
15	1	4/14/2016 ...	4/14/2016 ...	1	20.5	-73.776763...	40.6454391...	-73.953727...	40.7850685...	2	N	1

Module 2: Explore and visualize data using notebooks.

The python runtime environment in Fabric notebooks comes with various pre-installed open-source libraries for building visualizations like matplotlib, seaborn, Plotly and more.

In this module we will use seaborn, a Python data visualization library that provides a high-level interface for building visuals on dataframes and arrays. You can learn more about seaborn [here](#).

Note: The python commands/script used in each step of this tutorial can be found in the accompanying notebook: *02 - Explore and Visualize Data using Notebooks*

In this tutorial you will learn to perform the following actions:

1. Read data stored from a delta table in the lakehouse.
2. Generate a random sample of the dataframe.
3. Convert a Spark dataframe to Pandas dataframe which is supported by python visualization libraries.
4. Perform exploratory data analysis using Seaborn on the New York taxi yellow cab dataset by visualizing a trip duration variable against other categorical and numeric variables.

Steps to follow.

1. To get started, let's read delta table(saved in module 1) from lakehouse and create a pandas dataframe on a random sample of the data.

```
data = spark.read.format("delta").load("Tables/nyctaxi_raw")
SEED = 1234
sampled_df = data.sample(True, 0.001, seed=SEED).toPandas()
```

Note: To minimize execution time in this tutorial, we are using a 1/1000 sample to explore and visualize ingested data.

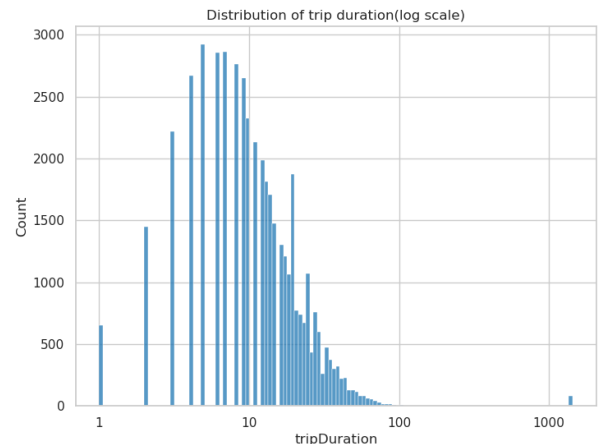
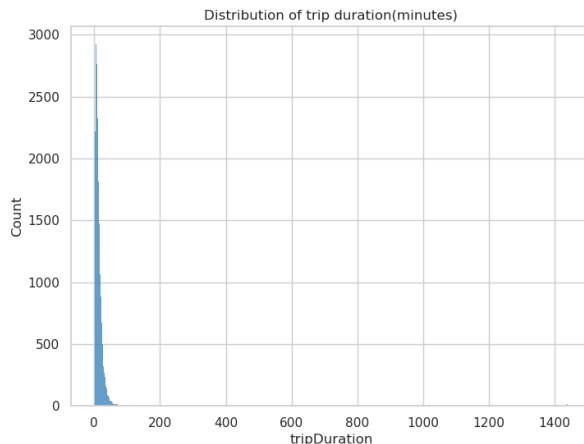
2. Import required libraries and function required for visualizations and set seaborn theme parameters that control aesthetics of the output visuals like style, color palette and size of the visual.

```
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
import numpy as np
sns.set_theme(style="whitegrid", palette="tab10", rc = {'figure.figsize':(9,6)})
```

3. Visualize distribution of trip duration(minutes) on linear and logarithmic scale by running below set of commands.

```
## Compute trip duration(in minutes) on the sample using pandas
sampled_df['tripDuration'] = (sampled_df['tpepDropoffDateTime'] - sampled_df['tpepPickupDateTime']).astype('timedelta64[m]')
sampled_df = sampled_df[sampled_df["tripDuration"] > 0]

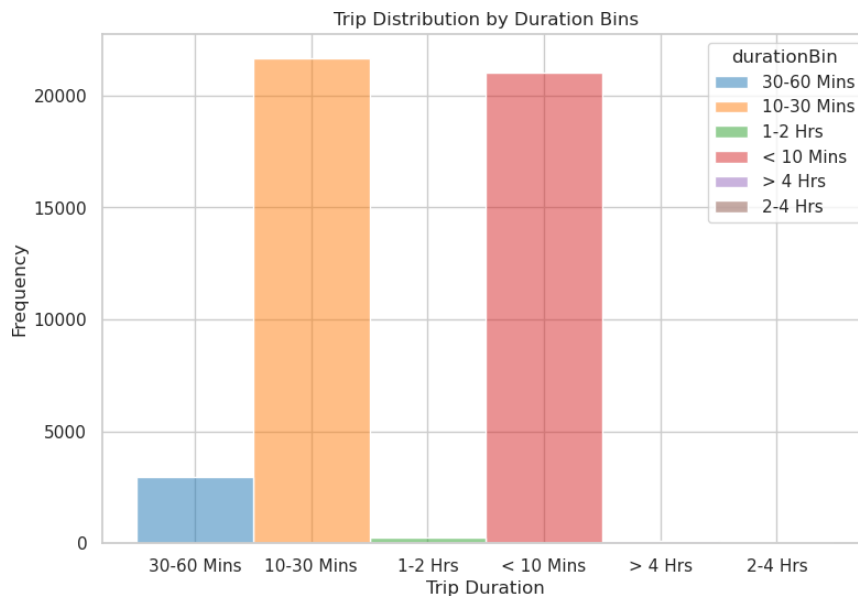
fig, axes = plt.subplots(1, 2, figsize=(18, 6))
sns.histplot(ax=axes[0], data=sampled_df,
             x="tripDuration",
             stat="count",
             discrete=True).set(title='Distribution of trip duration(minutes)')
sns.histplot(ax=axes[1], data=sampled_df,
             x="tripDuration",
             stat="count",
             log_scale=True).set(title='Distribution of trip duration(log scale)')
axes[1].xaxis.set_major_formatter(mticker.ScalarFormatter())
plt.show()
```



4. Create bins to segregate and understand distribution of tripDuration better by creating a durationBin column using pandas operations to classify trip durations into buckets of <10 Mins, 10-30 Mins, 30-60 Mins, 1-2 Hrs, 2-4 Hrs and > 4 Hrs. Visualize the binned column using seaborn histogram plot.

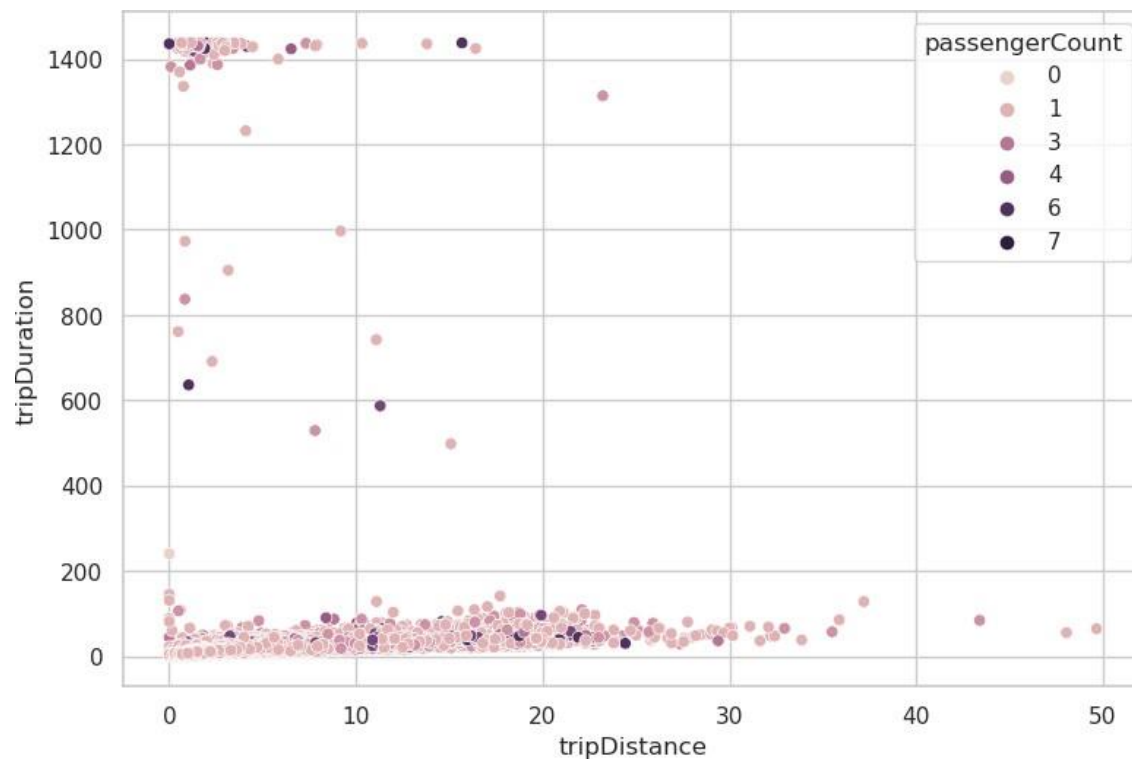
```
## Create bins for tripDuration column
sampled_df.loc[sampled_df['tripDuration'].between(0, 10, 'both'), 'durationBin'] = '< 10 Mins'
sampled_df.loc[sampled_df['tripDuration'].between(10, 30, 'both'), 'durationBin'] = '10-30 Mins'
sampled_df.loc[sampled_df['tripDuration'].between(30, 60, 'both'), 'durationBin'] = '30-60 Mins'
sampled_df.loc[sampled_df['tripDuration'].between(60, 120, 'right'), 'durationBin'] = '1-2 Hrs'
sampled_df.loc[sampled_df['tripDuration'].between(120, 240, 'right'), 'durationBin'] = '2-4 Hrs'
sampled_df.loc[sampled_df['tripDuration'] > 240, 'durationBin'] = '> 4 Hrs'

# Plot histogram using the binned column
sns.histplot(data=sampled_df, x="durationBin", stat="count", discrete=True, hue = "durationBin")
plt.title("Trip Distribution by Duration Bins")
plt.xlabel('Trip Duration')
plt.ylabel('Frequency')
```



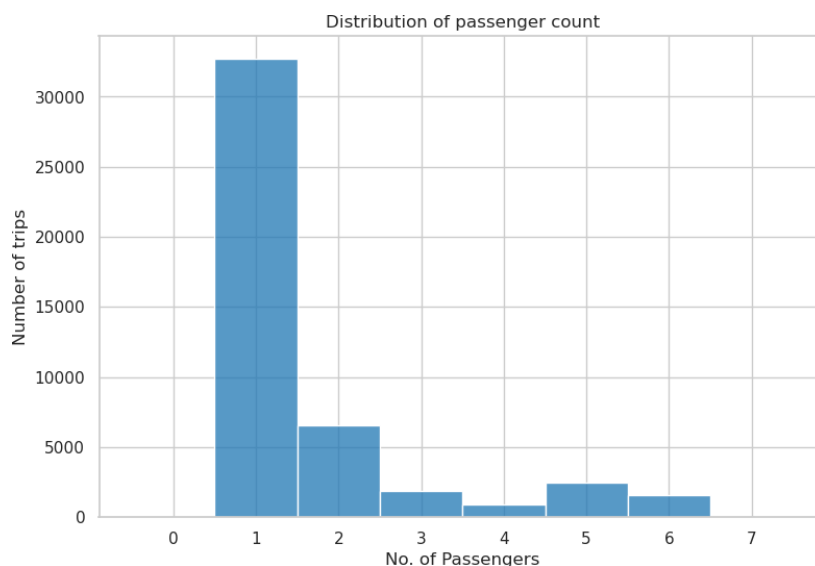
5. Visualize the distribution of tripDuration and tripDistance and classify by passengerCount using seaborn scatterplot by running below commands.

```
sns.scatterplot(data=sampled_df, x="tripDistance", y="tripDuration", hue="passengerCount")
plt.show()
```



6. Visualize the overall distribution of **passengerCount** column to understand the most common **passengerCount** instances in the trips.

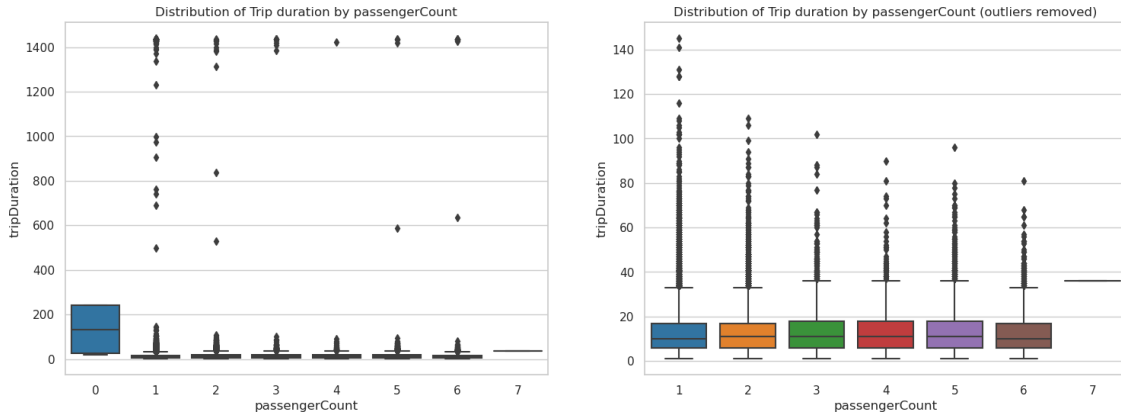
```
sns.histplot(data=sampled_df, x="passengerCount", stat="count", discrete=True)
plt.title("Distribution of passenger count")
plt.xlabel('No. of Passengers')
plt.ylabel('Number of trips')
```



- Create boxplots to visualize the distribution of **tripDuration** by passenger count. A boxplot is a useful tool to understand the variability, symmetry, and outliers of the data.

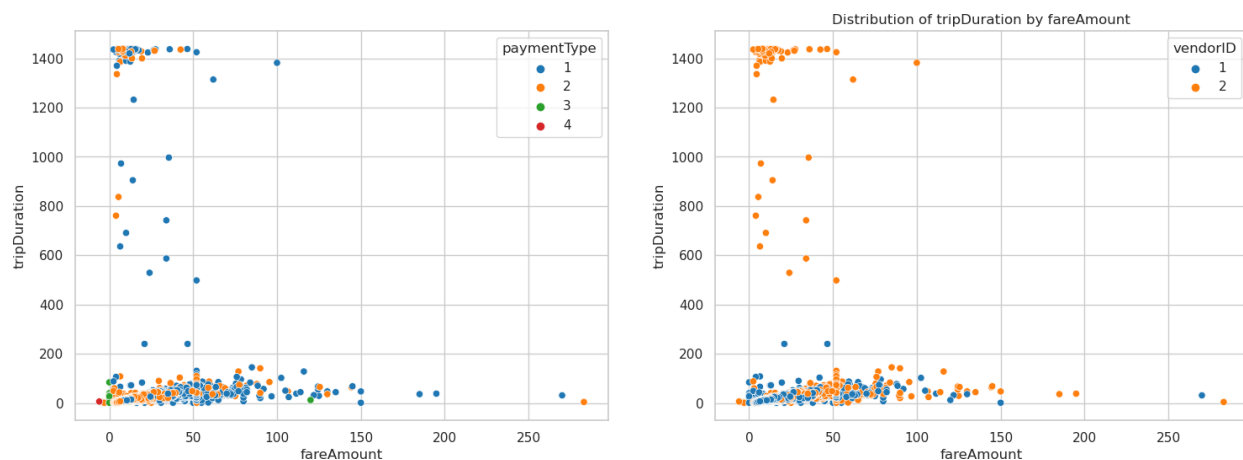
```
fig, axes = plt.subplots(1, 2, figsize=(18, 6))
sns.boxplot(ax=axes[0], data=sampled_df, x="passengerCount", y="tripDuration").set(title='Distribution of Trip duration by passengerCount')
sampleddf_clean = sampled_df[(sampled_df["passengerCount"] > 0) & (sampled_df["tripDuration"] < 180)]
sns.boxplot(ax=axes[1], data=sampleddf_clean, x="passengerCount", y="tripDuration").set(title='Distribution of Trip duration by passengerCount (outliers removed)')
plt.show()
```

In the first figure we visualize tripDuration without removing any outliers whereas in the second figure we are removing trips with duration greater than 3 hours and zero passengers.



- Analyze the relationship of **tripDuration** and **fareAmount** classified by **paymentType** and **VendorId** using seaborn scatterplots.

```
f, axes = plt.subplots(1, 2, figsize=(18, 6))
sns.scatterplot(ax =axes[0], data=sampled_df, x="fareAmount", y="tripDuration", hue="paymentType")
sns.scatterplot(ax =axes[1], data=sampled_df, x="fareAmount", y="tripDuration", hue="vendorID")
plt.title("Distribution of tripDuration by fareAmount")
plt.show()
```

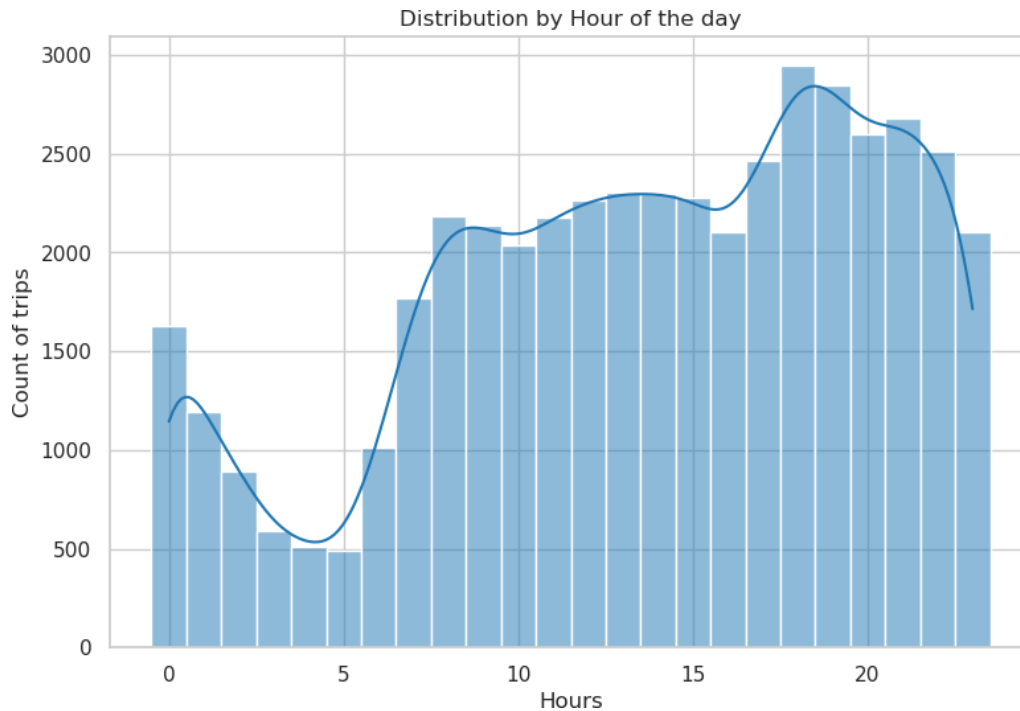


- Analyze the frequency of the taxi trips by hour of the day using a histogram of trip counts.

```

sampled_df['hour'] = sampled_df['tpepPickupDateTime'].dt.hour
sampled_df['dayofweek'] = sampled_df['tpepDropoffDateTime'].dt.dayofweek
sampled_df['dayname'] = sampled_df['tpepDropoffDateTime'].dt.day_name()
sns.histplot(data=sampled_df, x="hour", stat="count", discrete=True, kde=True)
plt.title("Distribution by Hour of the day")
plt.xlabel('Hours')
plt.ylabel('Count of trips')
plt.show()

```

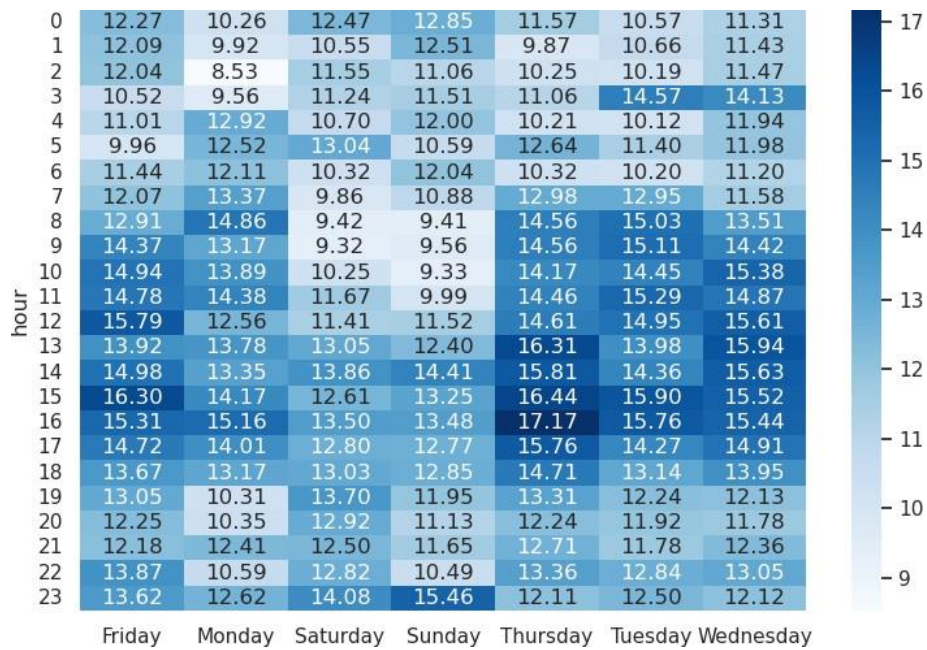


10. Analyze average taxi trip duration by hour and day together by using a seaborn heatmap. The below cell creates a pandas pivot table by grouping the trips by hour and **dayName** columns and getting a mean of the tripDuration values. This pivot table is used to create a heatmap using seaborn as below.

```

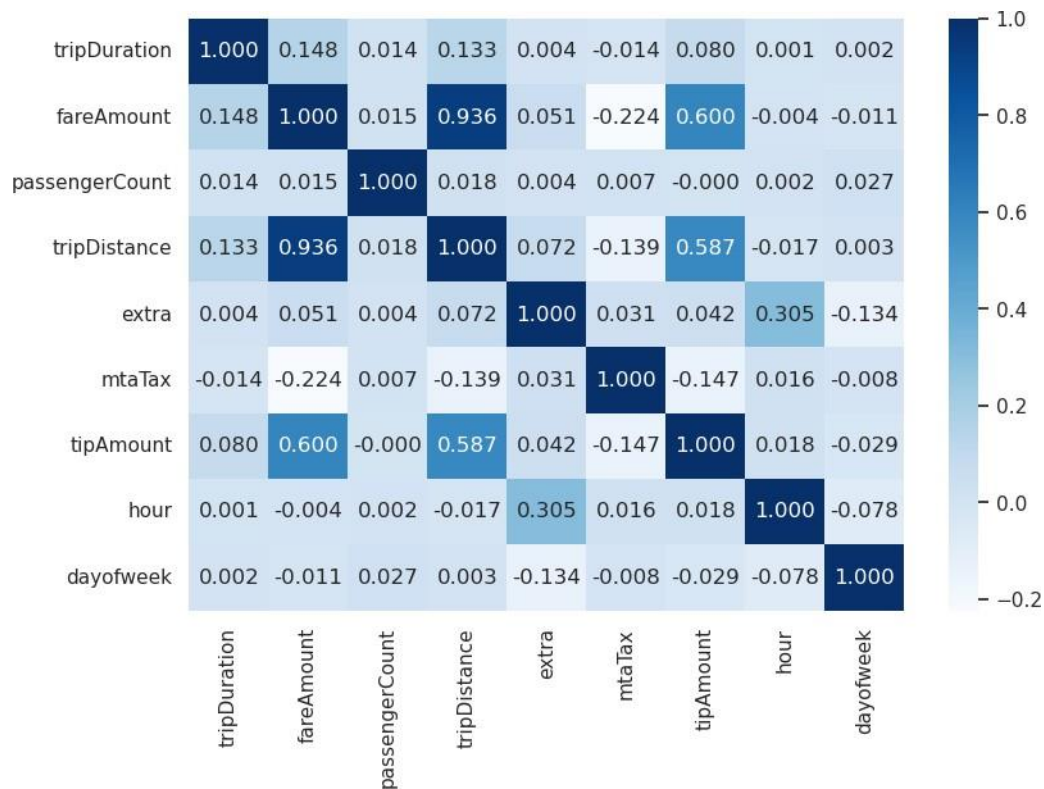
pv_df = sampled_df[sampled_df["tripDuration"]<180]\
    .groupby(["hour", "dayname"]).mean("tripDuration")\
    .reset_index().pivot("hour", "dayname", "tripDuration")
sns.heatmap(pv_df, annot=True, fmt='.2f', cmap="Blues").set(xlabel=None)

```

- Finally let's create a correlation plot which is a useful tool for exploring the relationships among numerical variables in a dataset. It displays the data points for each pair of variables as a scatterplot and calculates the correlation coefficient for each pair. The correlation coefficient indicates how strongly and in what direction the variables are related. A positive correlation means that the variables tend to increase or decrease together, while a negative correlation means that they tend to move in opposite directions. In this example we are generating correlation plot for a subset of columns in the dataframe for analysis.

```
cols_to_corr = ['tripDuration', 'fareAmount', 'passengerCount', 'tripDistance', 'extra', 'mtaTax',
                'improvementSurcharge', 'tipAmount', 'hour', "dayofweek"]
sns.heatmap(data = sampled_df[cols_to_corr].corr(),annot=True,fmt='.3f', cmap="Blues")
```



Summary of observations from exploration data analysis

- Some trips in the sample data have a passenger count of 0 but most trips have a passenger count between 1-6.
- tripDuration column has outliers with a comparatively small number of trips having **tripDuration** of greater than 3 hours.
- The outliers for TripDuration are specifically present for vendorId 2.
- Some trips have zero tripdistance and hence they can be cancelled and filtered out from any modeling.
- A small number of trips have no passengers(0) and hence can be filtered out.
- fareAmount column contains negative outliers which can be removed from model training.
- The number of trips starts rising around 16:00 hours and peaks between 18:00 - 19:00 hours.

Module 3: Perform Data Cleansing and preparation using Apache Spark.

The [NYC Yellow Taxi dataset](#) contains over 1.5 Billion trip records with each month of trip data running into millions of records which makes processing these records computationally expensive and often non-feasible using non-distributed processing engines.

In this tutorial we will demonstrate usage of Apache Spark notebook to clean and prepare the taxi trips dataset due to Sparks optimized distribution engine that makes it ideal for processing large volume of data.

Tip: For datasets of relatively small size, the Data Wrangler UI which is a notebook-based graphical user interface tool that provide interactive exploration and data cleansing experience for users working with pandas dataframes on Fabric notebooks.

In the following steps you will read the raw NYC Taxi data from lakehouse deltalake table(saved in module 1) and perform various operations to clean and transform data in order to make the dataset more conducive for training machine learning models.

Note: The python commands/script used in each step of this tutorial can be found in the accompanying notebook: *03 - Perform Data Cleansing and preparation using Apache Spark*

Steps to follow.

1. Load NYC yellow taxi Data from lakehouse delta table: **nyctaxi_raw** using spark.read command.

```
nyctaxi_df = spark.read.format("delta").load("Tables/nyctaxi_raw")
```

2. To aid the data cleansing process, next we will utilize Apache Spark's built-in summary feature that generates summary statistics, which are numerical measures that describe aspects of a column in the dataframe, such as count, mean, standard deviation, min, and max. You can use the command below to view the summary statistics of all columns in the **nyctaxi** dataset.

```
display(nyctaxi_df.summary())
```

Note: Generating Summary statistics is a computationally expensive process and can take considerable amount of execution time based on the size of the dataframe. In this tutorial this step takes between 2-3 minutes.

Table Chart Export results ▾					
Index	summary	vendorID	passengerCount	tripDistance	puLocationId
1	count	46429618	46429618	46429618	1942
2	mean	1.5317228326108563	1.6610720553419156	4.962012819920075	149.70957775489185
3	stddev	0.49899265250498853	1.3139169916343387	4204.500118805805	67.92532418822987
4	min	1	0	-3390583.8	10
5	25%	1.0	1	1.0	100.0
6	50%	2.0	1	1.7	142.0
7	75%	2.0	2	3.1	229.0
8	max	2	9	1.90726288E7	97

- In this step we will clean the **nytaxi_df** dataframe and add additional columns derived from the values of existing columns.

Below are the set of operations performed in this step:

Add derived Columns.

- pickupDate - convert datetime to date for visualizations and reporting.
- weekDay - day number of the week
- weekDayName - day names abbreviated.
- dayOfMonth - day number of month
- pickupHour - hour of pickup time
- tripDuration - representing duration in minutes of the trip.
- timeBins - Binned time of the day

Filter Conditions

- fareAmount is between and 100.
- tripDistance greater than 0.
- tripDuration is less than 3 hours (180 minutes).
- passengerCount is between 1 and 8.
- startLat, startLon, endLat, endLon are not NULL.
- Remove outstation trips(outliers) tripDistance>100.

```
from pyspark.sql.functions import col, when, dayofweek, date_format, hour, unix_timestamp, round, dayofmonth, lit
nytaxidf_prep = nytaxi_df.withColumn('pickupDate', col('tpepPickupDateTime').cast('date'))\
    .withColumn("weekDay", dayofweek(col("tpepPickupDateTime")))\
    .withColumn("weekDayName", date_format(col("tpepPickupDateTime"), "EEEE"))\
    .withColumn("dayOfMonth", dayofmonth(col("tpepPickupDateTime")))\
    .withColumn("pickupHour", hour(col("tpepPickupDateTime")))\
    .withColumn("tripDuration", (unix_timestamp(col("tpepDropoffDateTime")) - unix_timestamp(col("tpepPickupDateTime")))/60)\
    .withColumn("timeBins", when((col("pickupHour") >=7) & (col("pickupHour")<=10), "MorningRush")\
    .when((col("pickupHour") >=11) & (col("pickupHour")<=15), "Afternoon")\
    .when((col("pickupHour") >=16) & (col("pickupHour")<=19), "EveningRush")\
    .when((col("pickupHour") <=6) | (col("pickupHour")>=20), "Night"))\
    .filter("""fareAmount > 0 AND fareAmount < 100 and tripDistance > 0 AND tripDistance < 100
            AND tripDuration > 0 AND tripDuration <= 180
            AND passengerCount > 0 AND passengerCount <= 8
            AND startLat IS NOT NULL AND startLon IS NOT NULL AND endLat IS NOT NULL AND endLon IS NOT NULL""")
```

Note: Apache Spark uses Lazy evaluation paradigm which delays the execution of transformations until an action is triggered. This allows Spark to optimize the execution plan and avoid unnecessary computations. In this step the definitions of the transformations and filters are created. The actual cleansing and transformation will be triggered once data is written(an action) in next step.

- Once the cleaning steps are defined and assigned to a dataframe named: **nytaxidf_prep** we will write the cleansed and prepared data to a new delta table (**nyctaxi_prep**) in the attached lakehouse using below set of commands.

```
table_name = "nyctaxi_prep"
nytaxidf_prep.write.mode("overwrite").format("delta").save(f"Tables/{table_name}")
print(f"Spark dataframe saved to delta table: {table_name}")
```

The cleansed and prepared data produced in this module is now available in the lakehouse as delta table and can be used for further processing and generating insights.

Module 4: Train and register machine learning models.

In this module you will learn to train machine learning models to predict the total ride duration (tripDuration) of yellow taxi trips in New York City based on various factors such as pickup and drop-off locations, distance, date, time, number of passengers, and rate code. Once a model is trained, you will register the trained models, and log hyperparameters used and evaluation metrics using Fabric's native integration with the MLflow framework.

Tip: MLflow is an open-source platform for managing the end-to-end machine learning lifecycle with features for tracking experiments, packaging ML models and artifacts, and model registry. You can learn more about MLflow [here](#).

Note: The python commands/script used in each step of this tutorial can be found in the accompanying notebook: *04 - Train and track machine learning models*

In the following steps you will load cleansed and prepared data from lakehouse delta table and use it to train a regression model to predict tripDuration variable. You will also use the Fabric MLflow integration to create and track experiments and register the trained model, model hyperparameters and metrics.

Steps to follow.

1. In the first step, we will Import the mlflow library and create an experiment named **nyctaxi_tripduration** to log the runs and models produced as part of the training process.

```
# Create Experiment to Track and register model with mlflow
import mlflow
print(f"mlflow library version: {mlflow.__version__}")
EXPERIMENT_NAME = "nyctaxi_tripduration"
mlflow.set_experiment(EXPERIMENT_NAME)
```

2. Read Cleansed and prepared data from lakehouse delta table: **nyctaxi_prep** and create a fractional random sample from the data(to reduce computation time in this tutorial).

```
SEED = 1234
# note: From the perspective of the tutorial, we are sampling training data to speed up the execution.
training_df = spark.read.format("delta").load("Tables/nyctaxi_prep").sample(fraction = 0.5, seed = SEED)
```

Tip: A seed in machine learning is a value that determines the initial state of a pseudo-random number generator. A seed is used to ensure that the results of machine learning experiments are reproducible. By using the same seed, you can get the same sequence of numbers and thus the same outcomes for data splitting, model training, and other tasks that involve randomness.

3. Perform random split to get train and test datasets and define categorical and numeric features by executing the below set of commands. We will also cache the train and test dataframes to improve the speed of downstream processes.


```

TRAIN_TEST_SPLIT = [0.75, 0.25]
train_df, test_df = training_df.randomSplit(TRAIN_TEST_SPLIT, seed=SEED)

# Cache the dataframes to improve the speed of repeatable reads
train_df.cache()
test_df.cache()

print(f"train set count:{train_df.count()}")
print(f"test set count:{test_df.count()}")

categorical_features = ["storeAndFwdFlag", "timeBins", "vendorID", "weekDayName", "pickupHour", "rateCodeId", "paymentType"]
numeric_features = ['passengerCount', "tripDistance"]

```

***Tip:** Apache Spark caching is a feature that allows you to store intermediate data in memory or disk and reuse it for multiple queries or operations. Caching can improve the performance and efficiency of your Spark applications by avoiding reprocessing of data that is frequently accessed. You can use different methods and storage levels to cache your data, depending on your needs and resources. Caching is especially useful for iterative algorithms or interactive analysis that require repeated*

4. In this step we will define the steps to perform additional feature engineering and train the model using [Spark ML](#) pipelines and Microsoft [SynapseML](#) library. The algorithm used for this tutorial, [LightGBM](#) is a fast, distributed, high performance gradient boosting framework based on decision-tree algorithms. It is an open-source project developed by Microsoft and supports regression, classification, and many other machine learning scenarios. Its main advantages are faster training speed, lower memory usage, better accuracy, and support for distributed learning.

```

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 a pipeline steps for training a LightGBMRegressor regressor model
def lgbm_pipeline(categorical_features, numeric_features, hyperparameters):
    # String indexer
    stri = StringIndexer(inputCols=categorical_features,
                        outputCols=[f"{feat}Idx" for feat in categorical_features]).setHandleInvalid("keep")
    # encode categorical/indexed columns
    ohe = OneHotEncoder(inputCols=stri.getOutputCols(),
                      outputCols=[f"{feat}Enc" for feat in categorical_features])

    # convert all feature columns into a vector
    featurizer = VectorAssembler(inputCols=ohe.getOutputCols() + numeric_features, outputCol="features")

    # Define the LightGBM regressor
    lgr = LightGBMRegressor(
        objective = hyperparameters["objective"],
        alpha = hyperparameters["alpha"],
        learningRate = hyperparameters["learning_rate"],
        numLeaves = hyperparameters["num_leaves"],
        labelCol="tripDuration",
        numIterations = hyperparameters["iterations"],
    )
    # Define the steps and sequence of the SPark ML pipeline
    ml_pipeline = Pipeline(stages=[stri, ohe, featurizer, lgr])
    return ml_pipeline

```

5. Define Training Hyperparameters as a python dictionary for the initial run of the lightgbm model by executing the below cell.

```
# Default hyperparameters for LightGBM Model
LGBM_PARAMS = {"objective": "regression",
               "alpha": 0.9,
               "learning_rate": 0.1,
               "num_leaves": 31,
               "iterations": 100}
```

Tip: Hyperparameters are the parameters that you can change to control how a machine learning model is trained. Hyperparameters can affect the speed, quality and accuracy of the model. Some common methods to find the best hyperparameters are by testing different values, using a grid or random search, or using a more advanced optimization technique. The hyperparameters for the LightGBM model in this tutorial have been pre-tuned using a distributed grid search(not covered as part of this tutorial) run using the [hyperopt](#)

- In this step we create a new run in the defined experiment using Mlflow and fit the defined pipeline on the training dataframe, and then generate predictions on the test dataset using the below set of commands.

```
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")
```

- Once a model is trained and predictions generated on the test set, we can compute model statistics for evaluating performance of the trained LightGBMRegressor model by leveraging SynapseML library utility **ComputeModelStatistics** which helps evaluate various types of models based on the algorithm. Once the metrics are generated, we will also convert them into a python dictionary object for logging purposes. The metrics on which a regression model is evaluated are MSE(Mean Square Error), RMSE(Root Mean Square Error), R^2 and MAE(Mean Absolute Error).

```
1 from synapse.ml.train import ComputeModelStatistics
2 import json
3 lg_metrics = ComputeModelStatistics(
4     evaluationMetric="regression", labelCol="tripDuration", scoresCol="prediction"
5 ).transform(lg_predictions)
6 lg_metrics_dict = json.loads(lg_metrics.toJSON().first())
7 display(lg_metrics)
```

✓ 5 sec - Command executed in 5 sec 61 ms by Abid Nazir Guroo on 9:25:27 PM, 4/17/23

Table Chart ↳ Export results ▾

Index	mean_squared_error	root_mean_squared_error	R^2	mean_absolute_error
1	25.721591239516997	5.071645811718026	0.7613537489434428	3.2594670228763087

- Next, we will define a general function to register the trained LightGBMRegressor model with default hyperparameters under the created experiment using Mlflow. We will also log associated

hyperparameters used and metrics for model evaluation in the experiment run and terminate the run in the Mlflow experiment in the end.

```
from mlflow.models.signature import ModelSignature
from mlflow.types.utils import _infer_schema

# Define a function to register a spark model
def register_spark_model(run, model, model_name, signature, metrics, hyperparameters):
    # log the model, parameters and metrics
    mlflow.spark.log_model(model, artifact_path = model_name, signature=signature, registered_model_name = model_name, dfs_tmpdir="Files/tmp/mlflow")
    mlflow.log_params(hyperparameters)
    mlflow.log_metrics(metrics)
    model_uri = f"runs:/{run.info.run_id}/{model_name}"
    print(f"Model saved in run {run.info.run_id}")
    print(f"Model URI: {model_uri}")
    return model_uri

# Define Signature object
sig = ModelSignature(inputs=_infer_schema(train_df.select(categorical_features + numeric_features)),
                    outputs=_infer_schema(train_df.select("tripDuration")))

ALGORITHM = "lightgbm"
model_name = f"{EXPERIMENT_NAME}_{ALGORITHM}"

# Call model register function
model_uri = register_spark_model(run = run,
                                model = lg_model,
                                model_name = model_name,
                                signature = sig,
                                metrics = lg_metrics_dict,
                                hyperparameters = LGBM_PARAMS)

mlflow.end_run()
```

9. Once the default model is trained and registered, we will define tuned hyperparameters (tuning hyperparameters not covered in this tutorial) and remove the *paymentType* categorical feature since *paymentType* is usually selected at the end of a trip, we hypothesize that it shouldn't be useful to predict trip duration.

```
# Tuned hyperparameters for LightGBM Model
TUNED_LGBM_PARAMS = {"objective": "regression",
                     "alpha": 0.08373361416254149,
                     "learning_rate": 0.0801709918703746,
                     "num_leaves": 92,
                     "iterations": 200}

# Remove paymentType
categorical_features.remove("paymentType")
```

10. After defining new hyperparameters and updating feature list, we will fit the lightgbm pipeline with tuned hyperparameters 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, TUNED_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")
```

11. Generate model evaluation metrics for the new LightGBM regression model with optimized hyperparameters and updated features.

```
lg_metrics_tn = ComputeModelStatistics(
    evaluationMetric="regression", labelCol="tripDuration", scoresCol="prediction"
).transform(lg_predictions)
lg_metrics_tn_dict = json.loads(lg_metrics_tn.toJSON().first())
display(lg_metrics_tn)
```


12. In the final step of the module, we register the 2nd LightGBM regression model, and log the metrics and hyperparameters to the Mlflow experiment and end the run.

```
# Define Signature object
sig = ModelSignature(inputs=_infer_schema(train_df.select(categorical_features + numeric_features)),
                    outputs=_infer_schema(train_df.select("tripDuration")))
model_uri = register_spark_model(run = run,
                                model = lg_model,
                                model_name = model_name,
                                signature = sig,
                                metrics = lg_metrics_dict,
                                hyperparameters = TUNED_LGBM_PARAMS)

mlflow.end_run()
```


>  Spark jobs (2 of 2 succeeded)

Table Chart → Export results ▾				
Index	mean_squared_error	root_mean_squared_error	R^2	mean_absolute_error
1	25.444472646953216	5.0442514456511	0.7637293020097541	3.241663446115354

At the end of the module, we now have two runs of the lightgbm regression model trained and registered in the Mlflow model registry and the model is also available in the workspace as Fabric model artifact.

Note: if you do not see your model artifact in the list, please refresh your browser.

In order to view the model in the UI:

- Navigate to your currently active Fabric workspace.
- Click on the model artifact named: ***nyctaxi_tripduration_lightgbm*** to open the model UI.
- On the model UI you can view the properties and metrics of a given run, compare performance of various runs and download various file artifacts associated with the trained model.

Here is a layout of the various features within the model UI on a Fabric workspace.

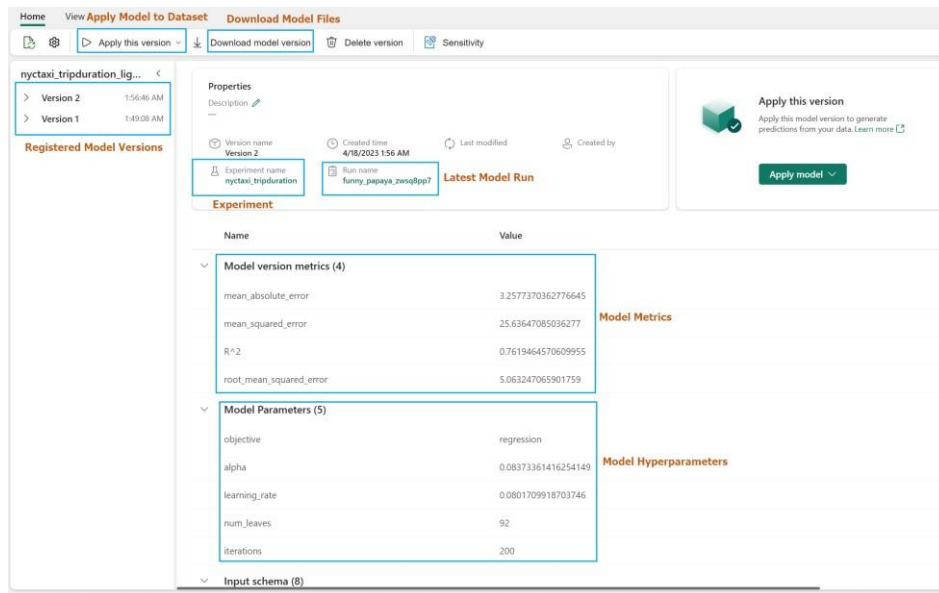


Figure 3: Model Artifact UI

Module 5: Perform batch scoring and save predictions to lakehouse.

In this module, you will learn to import a trained and registered LightGBMRegressor model from the Fabric MLflow model registry and perform batch predictions on a test dataset loaded from lakehouse.

Note: The python commands/script used in each step of this tutorial can be found in the accompanying notebook: *05 - Perform batch scoring and save predictions to lakehouse*

Steps to follow.

1. Read a random sample of cleansed data from lakehouse table: **nyctaxi_prep** filtered for puYear=2016 and puMonth=3.
2. Import the required pyspark.ml and synapse.ml libraries and load the trained and registered LightGBMRegressor model using the **run_uri** copied from the final step of "Module 4: Train and register machine learning models".

```
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>"
loaded_model = mlflow.spark.load_model(run_uri, dfs_tmpdir="Files/tmp/mlflow")
```

Enter run_id from module



- Run model transform on the input dataframe to generate predictions and remove unnecessary vector features created for model training using the below commands.

```
# Generate predictions by applying model transform on the input dataframe
predictions = loaded_model.transform(input_df)
cols_toremove = ['storeAndFwdFlagIdx', 'timeBinsIdx', 'vendorIDIdx', 'paymentTypeIdx', 'vendorIDEnc',
                 'rateCodeIdx', 'paymentTypeEnc', 'weekDayEnc', 'pickupHourEnc', 'storeAndFwdFlagEnc', 'timeBinsEnc', 'features', 'weekDayNameIdx',
                 'pickupHourIdx', 'rateCodeIdx', 'weekDayNameEnc']
output_df = predictions.withColumnRenamed("prediction", "predictedtripDuration").drop(*cols_toremove)
```

- Save predictions to lakehouse delta table: **nyctaxi_pred** for downstream consumption and analysis.

```
table_name = "nyctaxi_pred"
output_df.write.mode("overwrite").format("delta").save(f"Tables/{table_name}")
print(f"Output Predictions saved to delta table: {table_name}")
```

- You can preview the final predicted data by various methods including SparkSQL queries that can be executed using the `%%sql` magics command which tells the notebook engine that the cell is a SparkSQL script.

```
%%sql
SELECT * FROM nyctaxi_pred LIMIT 20
```

weekDayName	dayofMonth	pickupHour	tripDuration	timeBins	predictedtripDuration
Friday	6	16	59.78333333333333	EveningRush	61.58005626051534
Monday	2	11	35.81666666666667	Afternoon	42.417611111723566
Monday	2	10	0.2166666666666667	MorningRush	33.3357211596686
Wednesday	4	13	51.45	Afternoon	43.7793609451249
Sunday	1	20	54.21666666666667	Night	46.82853219745318
Friday	6	7	62.15	MorningRush	61.88564862320572
Friday	6	13	45.5	Afternoon	50.32821615322531
Monday	2	13	48.7	Afternoon	48.23349522219192
Saturday	7	18	57.4	EveningRush	47.96713768906313
Monday	2	22	36.36666666666667	Night	34.03186812788601
Wednesday	4	12	69.6	Afternoon	51.387055239584086
Saturday	7	15	39.13333333333333	Afternoon	43.54207101473043
Friday	6	13	25.1	Afternoon	35.83451867516566
Friday	6	15	39.35	Afternoon	59.92015611554394

- The **nyctaxi_pred** delta table containing predictions can also be viewed from the lakehouse UI by navigating to the lakehouse artifact in the active Fabric workspace.

Home

Get data

New Power BI dataset

Open notebook

Lakehouse

A SQL endpoint for SQL querying and a default dataset for reporting were created and will be updated with any tables added to the lakehouse. You can access the SQL endpoint using the dropdown.

Lakehouse explorer

DSLakehouse

Tables

nyctaxi_pred

nyctaxi_prep

nyctaxi_raw

Files

nyctaxi pred

Showing <1000> rows

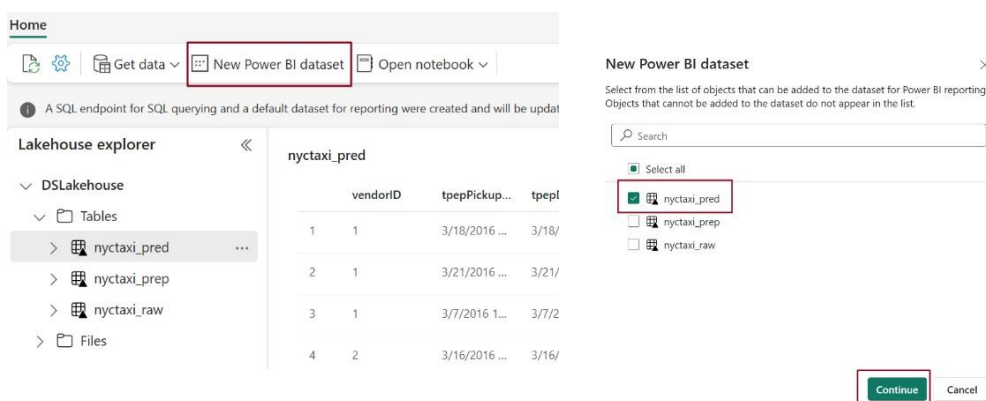
	vendorID	tpepPickup...	tpepDropo...	passenger...	tripDistance	startLon	startLat	endLon	endLat	rateCodeId	storeAndF...	paymentTy...
1	1	3/16/2016 ...	3/16/2016 ...	1	16.1	-73.971786...	40.7513732...	-73.790054...	40.6460886...	2	N	1
2	1	3/21/2016 ...	3/21/2016 ...	1	17.4	-73.983001...	40.7618598...	-73.781791...	40.6444511...	2	N	1
3	1	3/7/2016 1...	3/7/2016 1...	1	11.5	-74.005256...	40.7237205...	-74.005249...	40.7237167...	2	N	1
4	2	3/16/2016 ...	3/16/2016 ...	1	16.08	0	0	0	0	2	N	1
5	1	3/20/2016 ...	3/20/2016 ...	1	28.2	-73.951118...	40.7744331...	-73.790390...	40.6467666...	2	Y	1
6	1	3/11/2016 ...	3/11/2016 ...	1	21.1	-73.980346...	40.7739067...	-73.785751...	40.6433143...	2	N	1
7	1	3/25/2016 ...	3/25/2016 ...	1	18	-73.982315...	40.7569236...	-73.776206...	40.6454315...	2	N	1
8	1	3/28/2016 ...	3/28/2016 ...	1	20.7	-73.790802...	40.6466712...	-73.974700...	40.7567443...	2	N	1
9	2	3/26/2016 ...	3/26/2016 ...	1	19.3	-73.780288...	40.6454315...	-73.979141...	40.7640800...	2	N	1
10	2	3/28/2016 ...	3/28/2016 ...	1	18.46	-73.788063...	40.6415863...	-73.988159...	40.7336654...	2	N	1
11	2	3/9/2016 1...	3/9/2016 1...	1	19.31	-73.776687...	40.6453170...	-73.985176...	40.7583732...	2	N	1
12	2	3/19/2016 ...	3/19/2016 ...	1	16.63	-73.785949...	40.6517143...	-73.981430...	40.7560806...	2	N	1
13	1	3/25/2016 ...	3/25/2016 ...	1	16.3	-73.992431...	40.7600288...	-74.182769...	40.6878433...	3	N	1
14	2	3/25/2016 ...	3/25/2016 ...	1	21.56	-73.979431...	40.7658500...	-74.177230...	40.6951293...	3	N	1
15	2	3/3/2016 2...	3/3/2016 3...	1	19.83	-73.989583...	40.7630462...	-74.177452...	40.6906814...	3	N	1

Module 6: Create a Power BI report to visualize predictions.

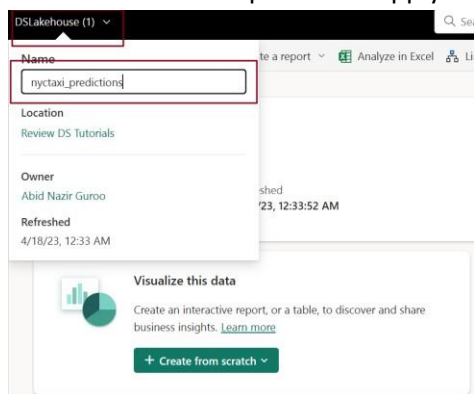
In this module, we will use Fabric DirectLake feature that enables direct connectivity from Power BI datasets to Lakehouse tables in direct query mode with automatic data refresh. In the following steps you will use the prediction data produced in *“Module 5: Perform Batch Scoring and save predictions to lakehouse.”*

Steps to follow.

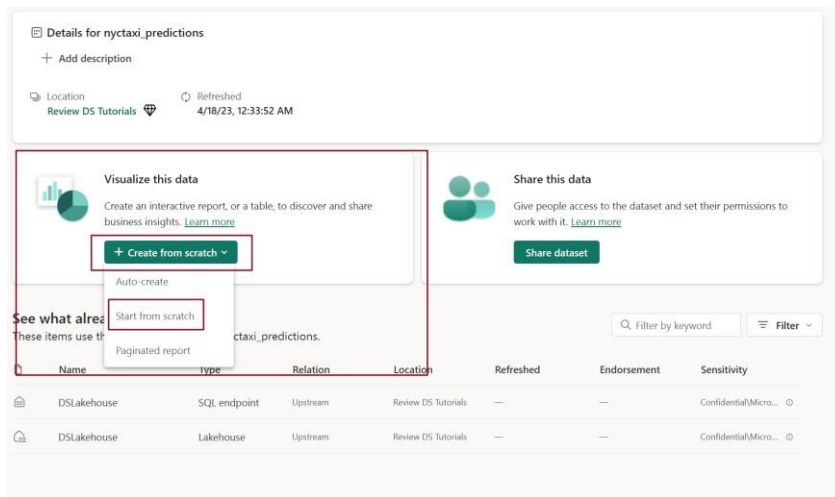
1. Navigate to the default lakehouse artifact in the workspace, that you used as part of the previous modules and open the lakehouse UI.
2. Click on the “New Power BI dataset” icon on the top ribbon and select **nyctaxi_pred** and click continue to create a new Power BI dataset linked to the predictions data produced in module 5.



3. Once the UI for the new dataset loads, rename the dataset by clicking on the dropdown at top left corner of the dataset page and enter a more user-friendly name: **nyctaxi_predictions** and click outside the drop down to apply the name change.



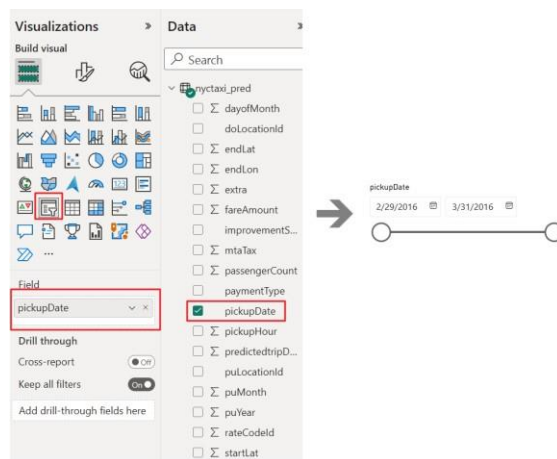
4. On the dataset pane in the section titled: **Visualize the data**, click on **Create from scratch** option and then select **Start from Scratch** to open the Power BI report authoring page.



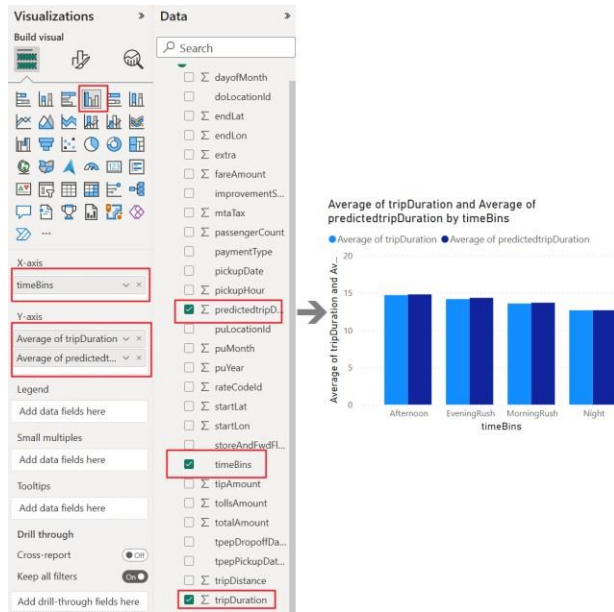
You can now create various visuals to generate insights from the prediction dataset.

Sample Visuals to analyze predictedTripDuration.

1. Create a Slicer visualization for pickupDate.
 - Select the slicer option from the visualizations pane and select **pickupDate** from the data pane and drop it on the created slicer visualization field of the date slider visual.

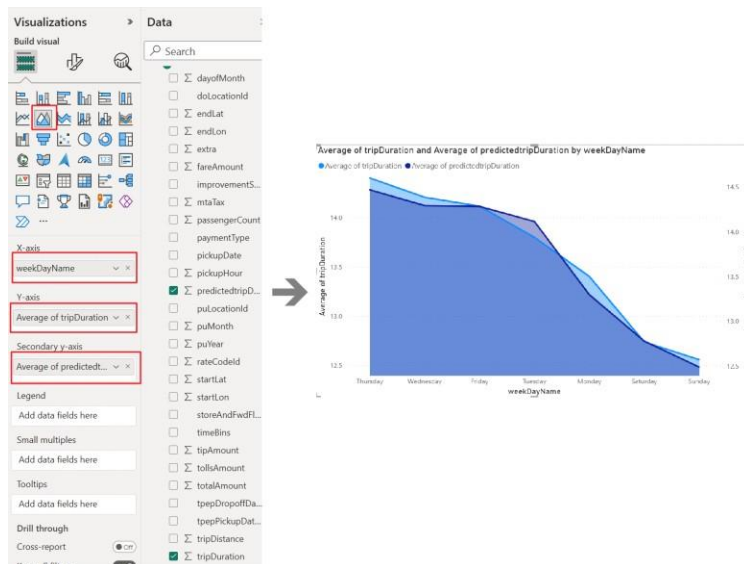


2. Visualize Average tripDuration and predictedTripDuration by timeBins using a clustered column chart.
 - Add a clustered column chart, add **timeBins** to X-axis, **tripDuration** and **predictedTripDuration** to Y-axis and change the aggregation method to Average.



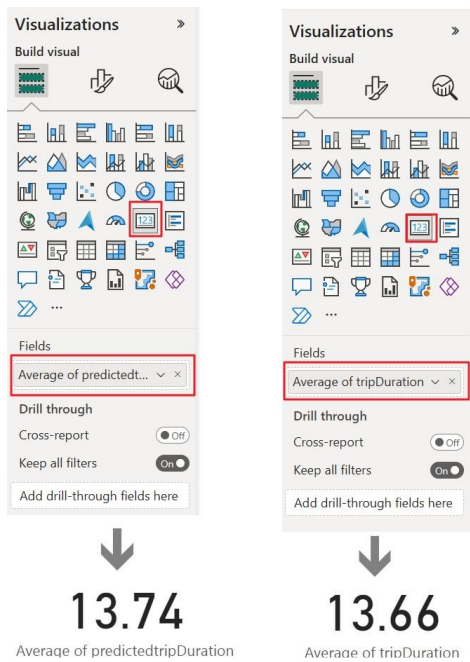
3. Visualize Average tripDuration and predictedTripDuration by weekDayName.

- Add an area chart visual and add **weekDayName** onto X-axis, **tripDuration** to Y-axis and **predictedTripDuration** to secondary Y-axis. Switch aggregation method to Average for both Y-axes.

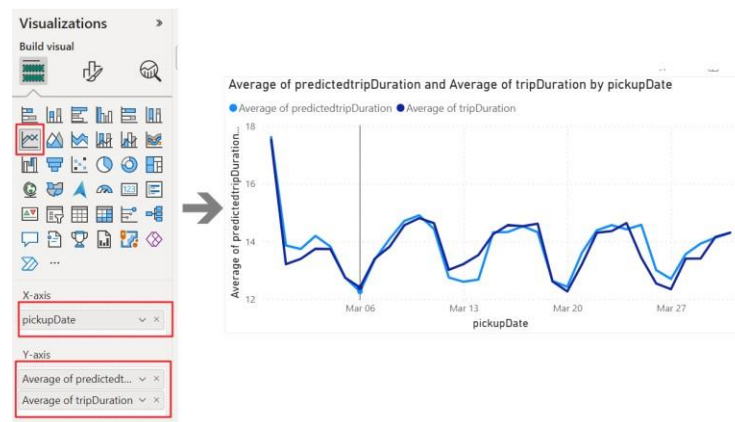


4. Add Card visuals for overall predictedTripDuration and tripDuration.

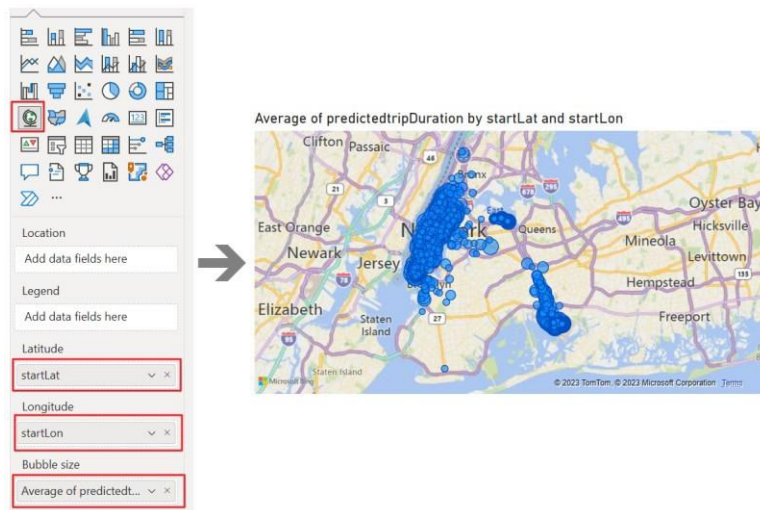
- Add a Card Visual and add predictedTripDuration to the fields and switch aggregation method to Average.
- Add a Card Visual and add TripDuration to the fields and switch aggregation method to Average.



5. Visualize Average tripDuration and predictedTripDuration by pickupDate using line chart.
 - Add a line chart visual and add **pickupDate** onto X-axis, **tripDuration** and **predictedTripDuration** to Y-axis and switch aggregation method to Average for both fields.



6. Visualize Average predictedTripDuration using a map visual.
 - Add a map chart visual and add **startLat** to Latitude **and startLon** to Longitude fields.
 - Add **predictedTripDuration** to bubble size field and switch the aggregation method of predictedTripDuration to Average.



Once all the visuals are added, you can reshape the visuals and re-align the layout based on your preferences.

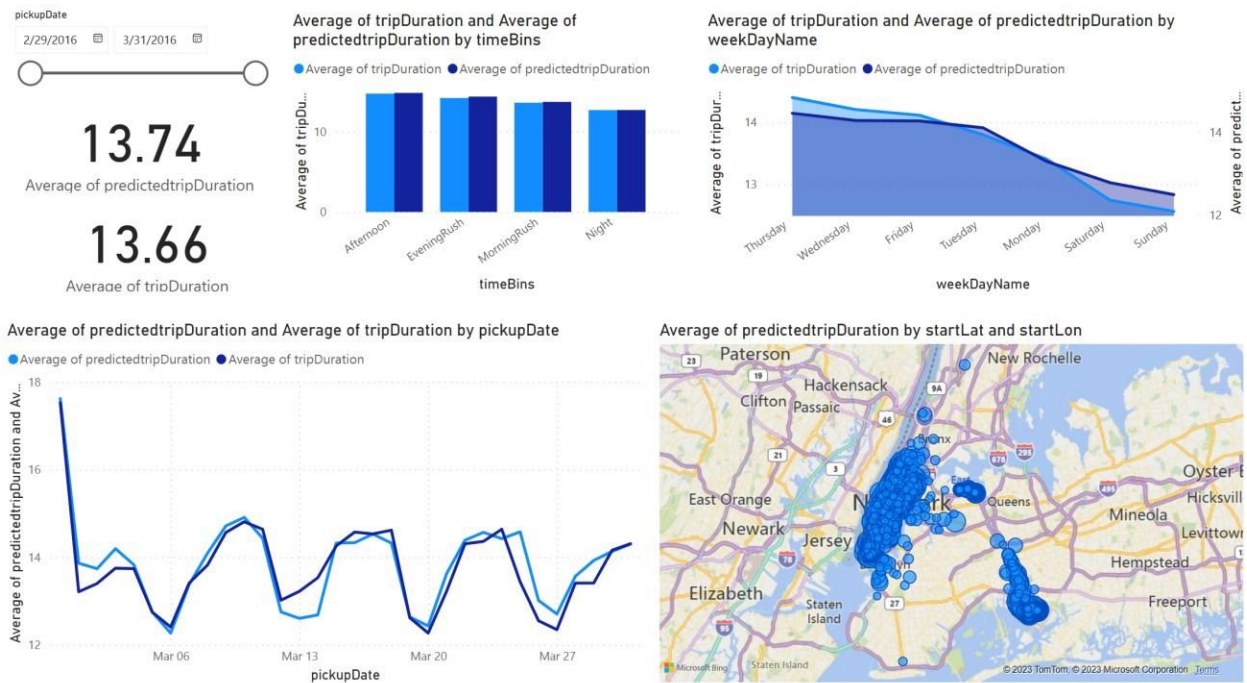


Figure 4: Visualizing predictedTripDuration in Power BI

Summary

In the above set of tutorials, we demonstrated a sample end-to-end scenario on the Fabric data science experience by implementing each step from data ingestion, cleansing and preparation to training machine learning models and generating insights, and consuming those insights using visualization tools like Power BI.