METCS 777 A1 - Big Data Analytics
BU ID - U94893520

EMR, Zeppelin, and Jupyter: A Comprehensive Integration for Big Data Visualization.

Introduction

What is visualization of data? And how is it different from the visualization of big data? Data visualization refers to the practice of converting raw data into visual contexts such as charts, graphs, maps and other visual representations allowing for easier interpretation of the data being analyzed. Big data refers to extremely large datasets that often require specialized software and infrastructure to process and analyze due to their size, velocity, and variety. The visualization of big data is more complex than regular data visualization. The large volume of data presents challenges like real-time processing, scalability concerns, and the need for more advanced visualization techniques.

Data visualization plays a pivotal role when it comes to dissecting and understanding such extensive datasets. Big data visualizations can instantly communicate patterns, trends, and anomalies in the data that would otherwise be lost in just the numbers on the raw dataset . In the context of the NYC Taxi Trip Data, visualization allows city planners, researchers, taxi operators, or anyone interested in the taxi industry—to make sense of the data, gaining insights that can lead to improved services and better traffic management or even learning about the trends for personal use.

The objective of this project is to enable visualization of big data and dive deep into the NYC Taxi Trip Data, employing data analysis techniques and powerful visualization tools to understand the trends in the taxi industry and also provide a thorough analysis on the technologies used to achieve this. The aim is to not only create visualizations, but also the process of how it is done with big data, which is more complex.

METCS 777 A1 - Big Data Analytics
BU ID - U94893520

Technology

Amazon EMR (Elastic MapReduce):

Amazon EMR is a cloud-native big data platform, allowing processing of large datasets across clusters of Amazon EC2 instances that are highly customizable. They are designed to handle vast datasets like the NYC Taxi Trip Data and other forms of big data which cannot be processed on a regular laptop, EMR provides a framework that makes data analysis both fast and cost-effective. By leveraging the distributed processing capability of tools like Apache Hadoop and Spark, EMR ensures the ability for scalability and efficient data processing for big data visualization. EMR and Spark will be utilized in this project for preprocessing, cleaning, structuring and aggregating the large volumes of taxi trip data before visualization.

Apache Zeppelin

Apache Zeppelin is an open-source, web-based notebook that enables data-driven, interactive data analytics, and visualization with the added benefit of supporting multiple languages within the same notebook, such as Scala (with Apache Spark), Python (with PySpark), SQL and more. It's often compared with other notebook interfaces like Jupyter and Databricks. Apache Zeppelin is very useful when it comes to the visualization of Big Data, a vital component is the ability to directly integrate with the big data ecosystem like Apache Spark that is running on an EMR (Elastic Map Reduce) cluster and the use of SQL queries to immediately visualize data using the built in tools in Zeppelin.

Jupyter Notebook Integration:

Jupyter Notebook is an open-source web application that facilitates the creation and sharing of documents containing live code, equations, visualizations, and text. When integrated with Amazon EMR, Jupyter Notebooks become a powerful tool for performing data analysis in an interactive environment. The live code capability enables real-time data manipulation, while its support for various programming languages and extensive python libraries make it ideal for visual data analysis. However, using Jupyter for big data visualization comes at a cost and we will explore this later on.

METCS 777 A1 - Big Data Analytics
BU ID - U94893520

Visualization Tools using Jupyter:

Plotly: A high-level charting library, it comes with over 40 distinct types of statistical, financial, geographic, scientific, and 3D charts. It's well known for creating interactive visualizations, which can be important in making the vast dataset of NYC taxi trips more comprehensible.

Dash: Dash is a productive Python framework for building web applications and dashboards with interactive visualizations. For a dataset as rich as the NYC Taxi Trip Data, Dash can transform static insights into interactive narratives, enabling users to explore the data in a hands-on manner.

Folium: Visualization tool perfect for understanding geographical patterns in data, such as plotting routes of NYC taxi trips or highlighting busy pick-up spots

Process of Big Data Visualization

Cluster Setup

Before we begin to visualize the data, we need to set up an EMR cluster on AWS since a local machine is not capable of handling very large datasets. For this we choose to use EMR since it has built in support with Apache Zeppelin. When setting up the cluster, it is important to select Zeppelin 0.10.1 as an application to install, along with Spark and Hadoop.

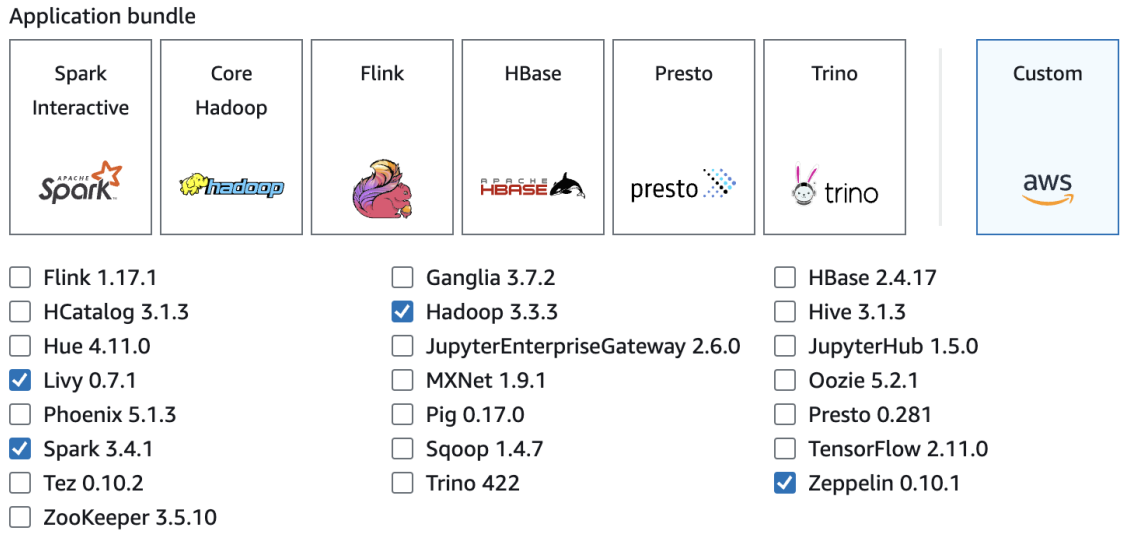


Figure 1: Cluster Setup Application Section

Next we need to choose machines that will be running the analysis of our data, for this we can choose any machine depending on the task or the size of the data. For this project we will be using c5.xlarge with 4 Cores and 8GB Memory for our master node and the same machines for our worker nodes but with 3 instances, this means we will have 3 worker machines and 1 master that will be utilized.

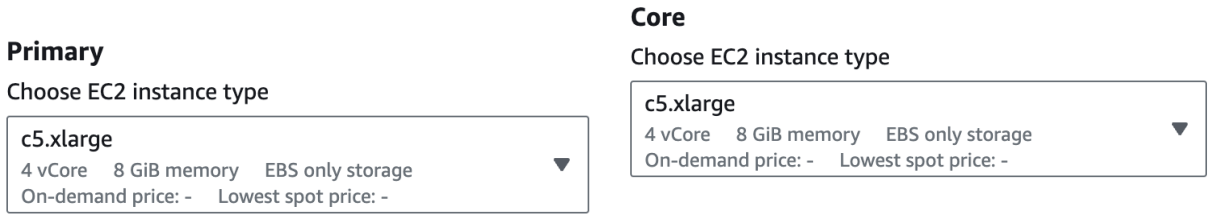


Figure 2: Machines Chosen for this Cluster

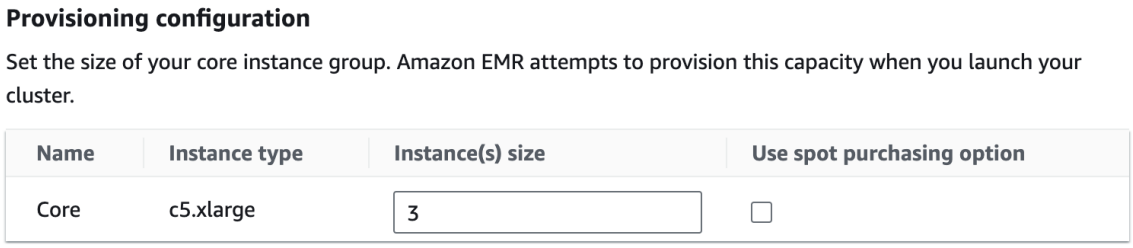


Figure 3: Number of core instances is set to 3

Now that we have the cluster set up, we can start it and then proceed with the next step which is to open Apache Zeppelin that is running on the EMR cluster.

Figure 4: Application UI Link for Zeppelin is listed under Applications

Dataset Loading and Preprocessing

Using spark we can load the dataset that is stored in an AWS S3 Bucket directly into the Zeppelin notebook

```
%spark.pyspark

#reading data from S3 bucket
df = spark.read.option("header", "false").csv("s3://psbigdata777/taxi-data-sorted-large.csv.bz2")

#Display the first few rows
df.show(5)
```

```
+--------------------+--------------------+-------------------+-------------------+---+----+----------+
|                 _c0|                 _c1|                _c2|                _c3|_c4| _c5|       _c6|
+--------------------+--------------------+-------------------+-------------------+---+----+----------+
|5EE2C4D3BF57BDB45...|E96EF8F6E6122591F...|2013-01-01 00:00:09|2013-01-01 00:00:36| 26|0.10|-73.992210|
|42730E78D8BE872B5...|6016A71F1D29D678E...|2013-01-01 00:01:00|2013-01-01 00:01:00|  0|0.01|  0.000000|
|CA6CD9BAED6A85E43...|77FFDF38272A60065...|2013-01-01 00:00:20|2013-01-01 00:01:22| 61|2.20|-73.970100|
```

After the dataset is loaded into the notebook, it has to be preprocessed. The dataset contains some missing values which have to be removed, column names should be changed for a clear understanding of what each column represents and the pick up and drop off timestamp should be converted to a date time format. The month, day and hour is extracted from this in order to group the data by these granularities later in the analysis.

A ‘profit_ratio’ column is added to the dataset which can provide more insight into the times when there is a higher profit ratio based on the surcharge amount.

The dataset has trip times in seconds, this is converted to minutes and is stored in a new column called ‘trip_duration’.

Aggregations

Now that we have a fully preprocessed dataset, we are one step closer to data visualization. In order to extract data that will be insightful, we can aim to aggregate the data monthly, daily and hourly based on the metrics that we want to explore. In this example, the total number of rides, averages of fare, tip, duration, trip distance, total amount, profit ratio and toll amount and payment type is calculated using pyspark.sql functions.

```
%spark.pyspark
from pyspark.sql.functions import when
combined_aggregates = (df.groupBy("pickup_month", "pickup_day", "pickup_hour").agg(
    F.count("*").alias("total_rides"),
    F.avg("fare_amount").alias("avg_fare"),
    F.avg("tip_amount").alias("avg_tip"),
    F.avg("trip_duration").alias("avg_trip_duration"),
    F.avg("trip_distance").alias("avg_trip_distance"),
    F.avg("total_amount").alias("avg_total_amount"),
    F.avg("profit_ratio").alias("avg_profit_ratio"),
    F.avg("tolls_amount").alias("avg_toll_amount"),
    F.sum(when(F.col("payment_type") == "CSH", 1).otherwise(0)).alias("cash_payments"),
    F.sum(when(F.col("payment_type") == "CRD", 1).otherwise(0)).alias("card_payments")
)
.withColumn("cash_percentage", F.col("cash_payments") / F.col("total_rides") * 100)
.withColumn("card_percentage", F.col("card_payments") / F.col("total_rides") * 100)
.orderBy("pickup_month", "pickup_day", "pickup_hour"))
```

Took 1 sec. Last updated by anonymous at October 31 2023, 11:28:25 PM.

Figure 6: Combined Aggregation code on the large dataset

After we have created an aggregation, it can be registered as a temporary SQL table. This allows for SQL queries to run directly against the DataFrame using Spark SQL.

```
%spark.pyspark
combined_aggregates.createOrReplaceTempView("combined_aggregates_view")
```

Using the combined data, we can aim to see which hour of the day has the highest tips using the SQL query below. The query will output a table which can be immediately visualized using Zeppelin's built in visualization tool

```
%sql
SELECT pickup_hour, SUM(avg_tip*total_rides) / SUM(total_rides) AS avg_hourly_tip
FROM combined_aggregates_view
GROUP BY pickup_hour
ORDER BY avg_hourly_tip DESC;
```

Figure 7: SQL Query to group by pick up hour and view the average hourly tip

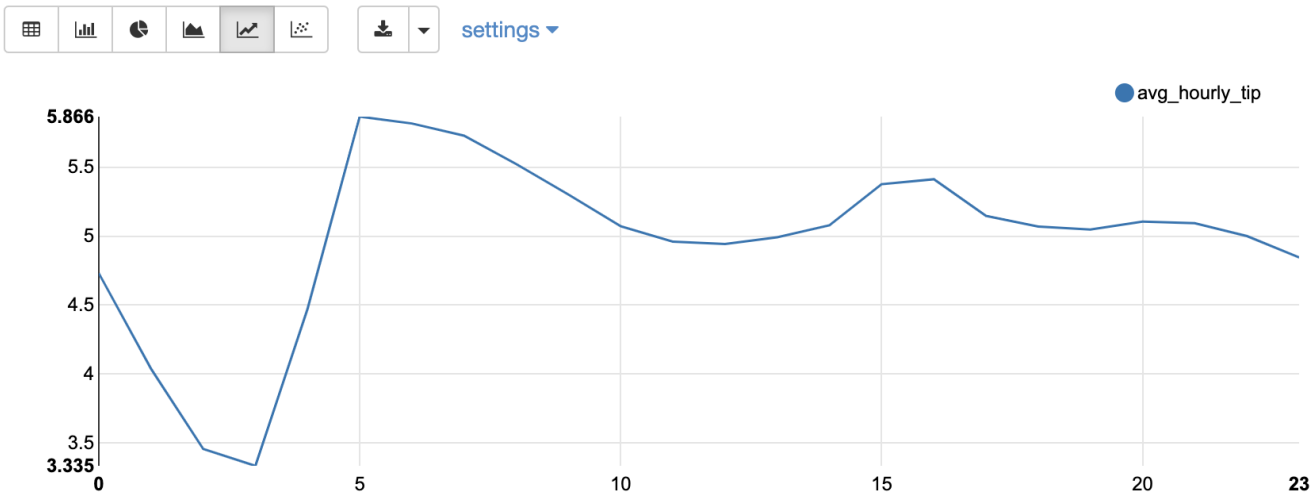
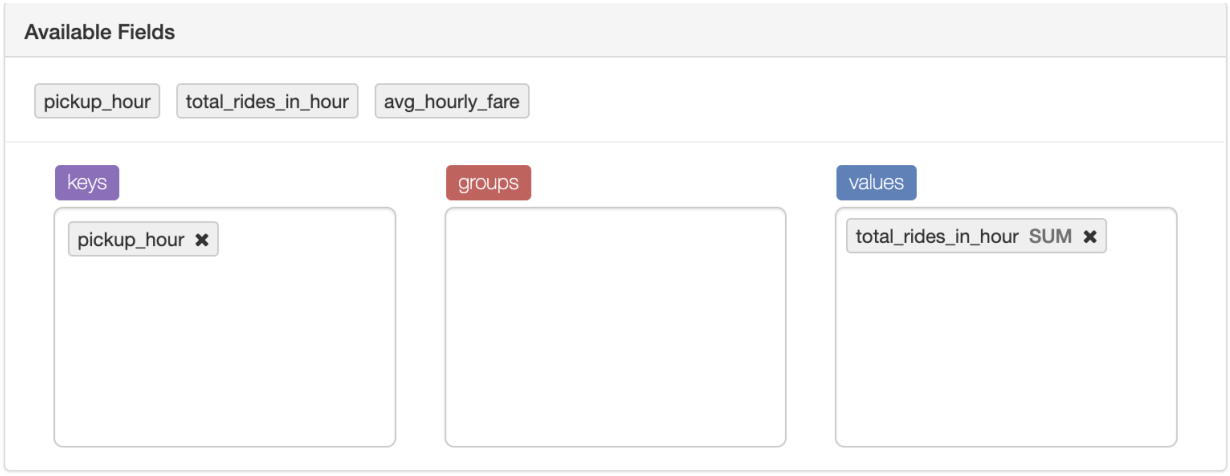


Figure 8: Visualization created of average hourly tip in each hour of the day from the entire dataset

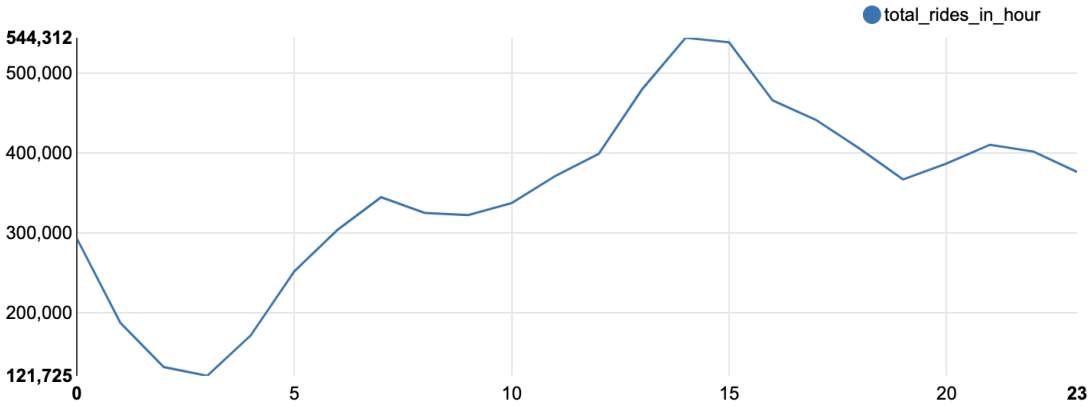
We can also include more metrics in the SQL query and Zeppelin has a feature to select which metrics that should be visualized. In the example below, we are selecting pick up hour, total number of rides and average hourly fare. This will allow for visualization of the total number of rides and hourly fare based on which metric we choose for each hour aggregated from the entire dataset.

Zeppelin provides a feature to select the available fields from the SQL query and it can be dragged into the relevant box for customizable visualizations

```
%sql
SELECT pickup_hour, SUM(total_rides) AS total_rides_in_hour,SUM(avg_fare*total_rides) / SUM(total_rides) AS avg_hourly_fare
FROM combined_aggregates_view
GROUP BY pickup_hour
ORDER BY total_rides_in_hour DESC;
```



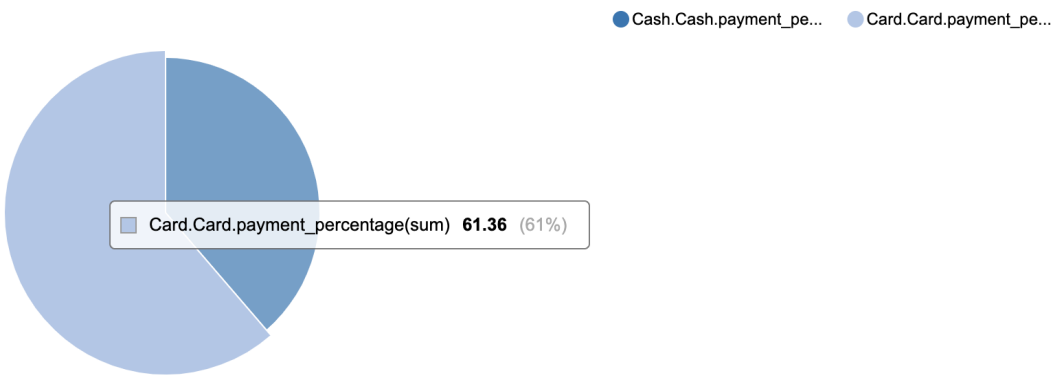
For example, we can first take a look at the total number of rides in each hour



This results in a line graph showing the total number of rides in each hour of the day across the entire dataset. A simple observation that can be made is that the total number of rides is at a maximum at hour 14 and 15 (2PM and 3PM).

If we change the values from total rides to average hourly fare, the graph in the Zeppelin notebook immediately changes to the new value which is average hourly fare.

Zeppelin not only has line graphs but also other types including pie charts, this can be very helpful in visualizing the total number of card or cash payments made in the entire dataset. Using SQL is a very efficient way to query the data into tables and then using the built in visualization tool can be used to view the data in a clear way if the table is not sufficient.



61% of the taxi rides are paid using a card, the rest being cash. This process allows us to draw immediate insights from the aggregated dataset, in this case we learned that over half of the trips are using a card as a payment type.

Dynamic Forms in Zeppelin

Zeppelin has a powerful feature which is the ability to create dynamic forms within notebooks. Dynamic forms allow users to interactively change the parameters of their analyses without having to modify the code itself.

For this example we had monthly aggregate data from a taxi dataset. Instead of creating separate visualizations or tables for each metric (like average fare, total rides, etc.), we used Zeppelin's dynamic forms feature to create an input metric box. This feature lets users select which metric they're interested in seamlessly. We are interested in understanding trends in avg__monthly_toll_amount, we would simply select that metric from the dropdown or input it in the chosen metric choice box and immediately see the results without altering any code. Another user might want to focus on the payment method trends. It can switch to cash_payments or card_percentage using the same method for quick visualization

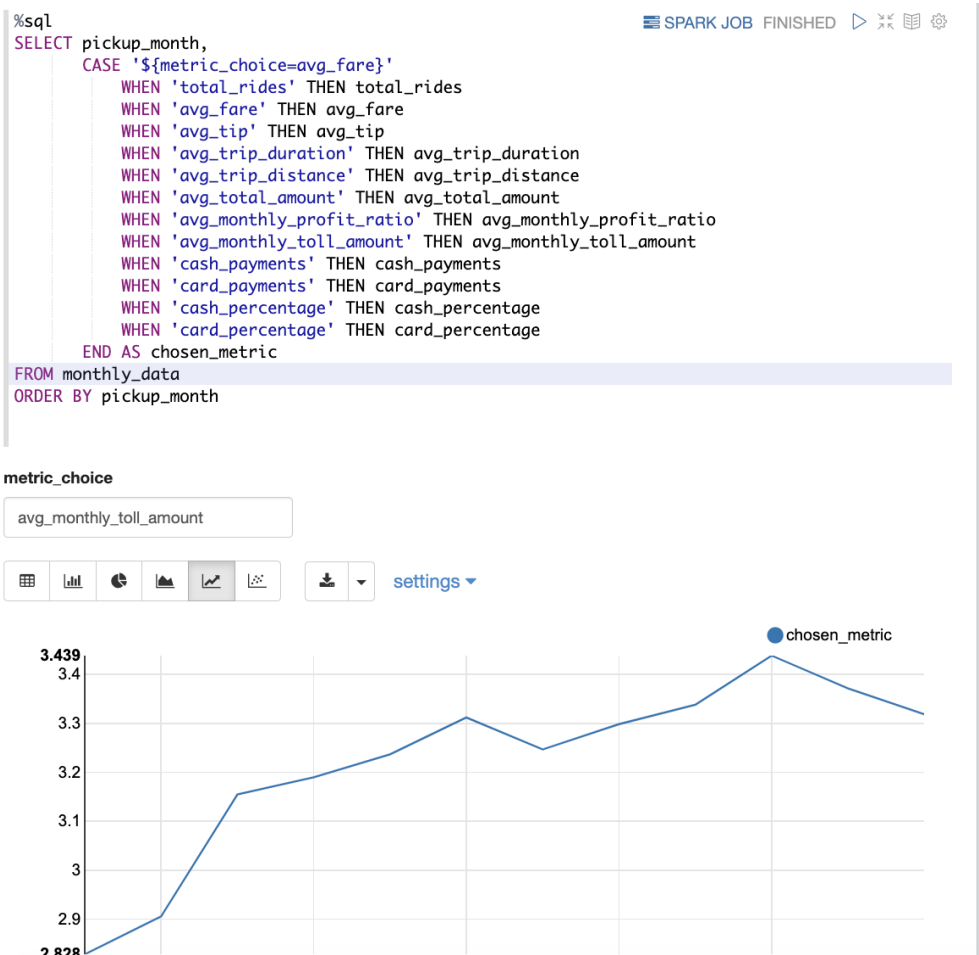


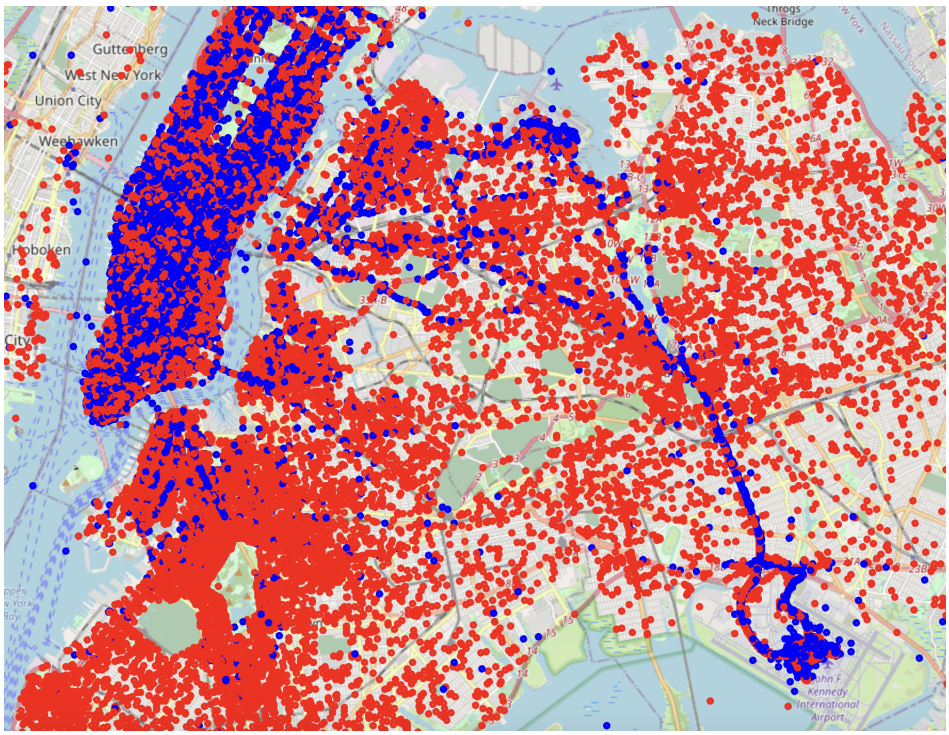
Figure: Dynamic Forms Example using monthly aggregate data for visualization of multiple metrics

Time Series Visualization

Zeppelin allowed for seamless visualization of big data, by integrating with the EMR cluster and querying data using SQL we were able to visualize basic trends directly from the large dataset. This is the advantage Apache Zeppelin has over Jupyter notebook. While Jupyter can also run SQL commands, further configuration and installation is needed and its performance is also slower when compared to Zeppelin. While it is possible to use more advanced visualization tools like Plotly on Zeppelin after initial data visualization using SQL queries, it might be a better alternative to aggregate the data, run SQL queries and use Zeppelin's built in visualization tool to identify the metrics and granularities to be further investigated. Using Plotly and Dash might be better suited with Jupyter since it is deeply integrated into the python ecosystem. Once we have explored the large dataset and performed aggregates, this can be saved as a parquet file to the AWS S3 bucket which can be loaded locally using Jupyter Notebook to use Plotly and Dash for more customizable visualizations. If the dataset is still too large for producing more visualizations, the Jupyter notebook can also be integrated with the EMR cluster for more processing power and memory to generate and render geospatial visualizations.

Geospatial Visualization

Pick Up and Drop Off Points Map

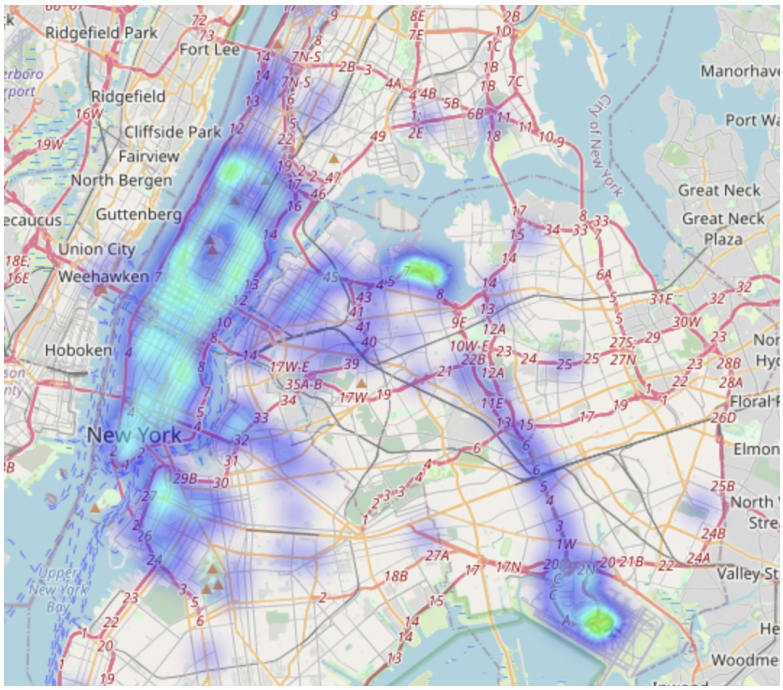


METCS 777 A1 - Big Data Analytics
BU ID - U94893520

This visualization presents a geospatial representation of taxi activity in the New York metropolitan area, mapped using the Folium library. The blue points scattered throughout the region indicate pick-up locations, while the more densely plotted red points indicate drop-off locations. A noticeable concentration of red points in specific neighborhoods suggests high drop-off demand in those areas, potentially indicating commercial hubs, tourist attractions, or transport junctions. The areas with a predominance of blue might indicate residential zones or areas with significant establishments where people initiate their journeys. For example we can see many blue points in JFK Airport. The map provides an intuitive snapshot of taxi transit dynamics, highlighting popular destinations and originating points across the city. It is also interactive and can be zoomed in for a more detailed view.

The geospatial visualizations were run on a Jupyter Notebook with the Folium library imported. Jupyter is capable and a very efficient tool to use for data visualization, especially when using python based libraries.

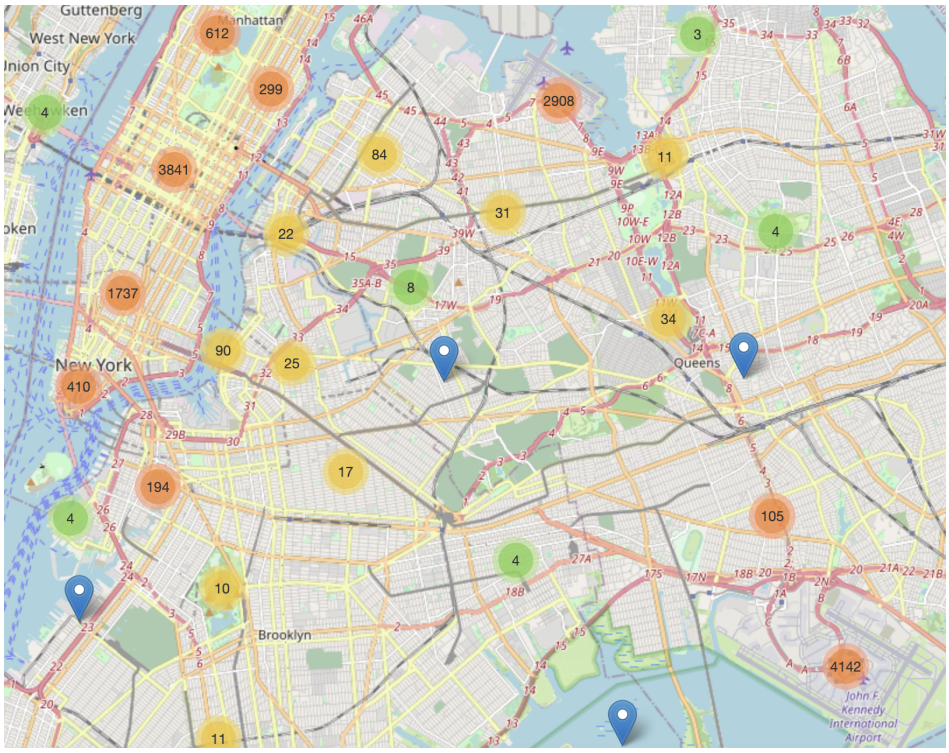
Heat Map



METCS 777 A1 - Big Data Analytics
BU ID - U94893520

This heat map generated by Folium provides a more generalized representation of taxi pick-up activity in the New York metropolitan area. Unlike the previous point-based map, which distinctly marked each pick-up and drop-off, this visualization uses gradient coloring to depict density. Areas with a more intense blue hue indicate higher concentrations of taxi pick-ups, suggesting they are popular origination zones.

Cluster Map



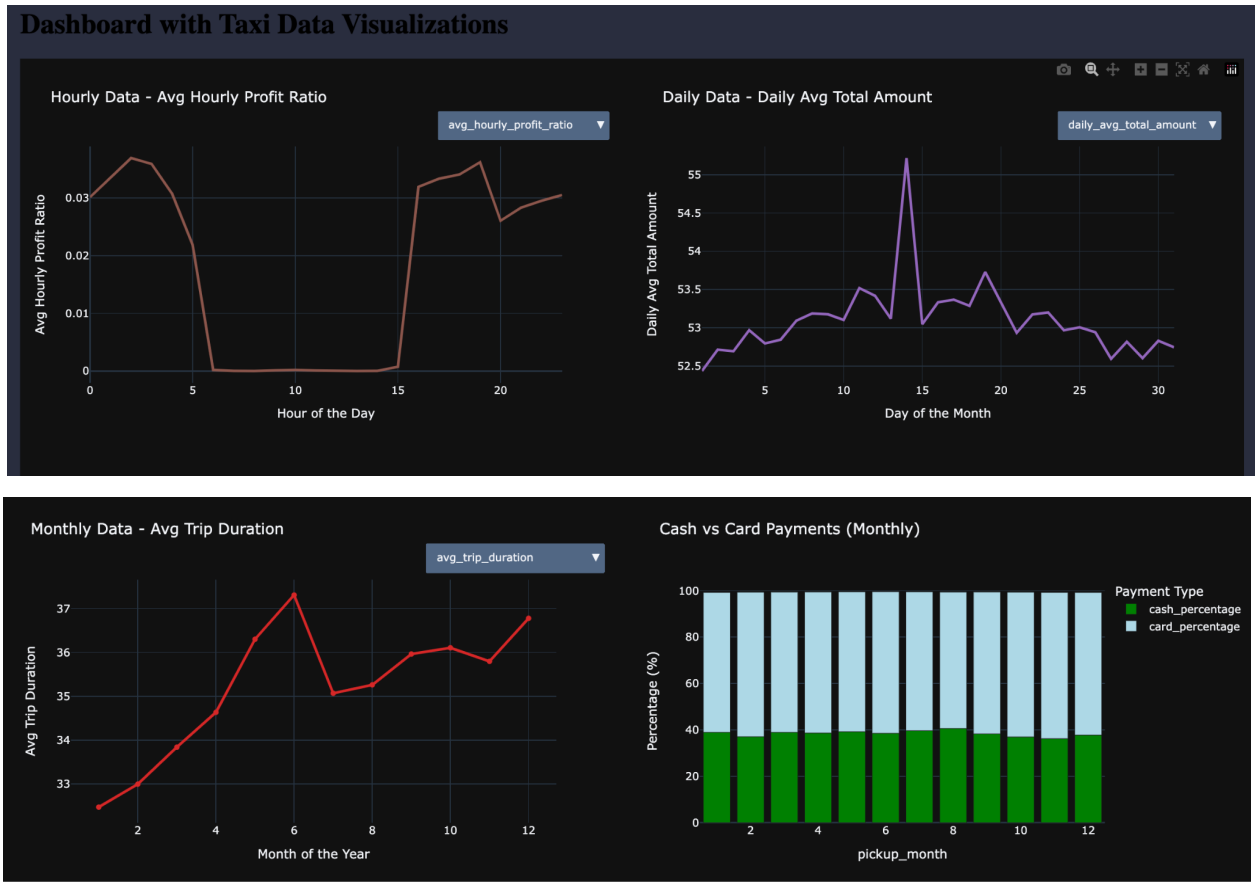
This cluster map provides a holistic view of activity concentrations across the New York metropolitan area. Unlike the earlier maps, this visualization aggregates close-proximity data points into clusters, represented by circles with numbers indicating the count of aggregated points. When we zoom in, these clusters break down to reveal individual data points or smaller clusters, providing a dynamic level of detail.

A primary advantage of the cluster map is its ability to offer a clear, uncluttered overview of data density across large areas. For instance, the circle with "3841" near Manhattan's west side suggests an incredibly high activity concentration, possibly denoting a major transportation or

commercial hub. The cluster map efficiently communicates the scale of data points in specific areas without overwhelming amounts of individual points. It's particularly useful when dealing with large datasets, as it condenses information into digestible chunks, making it easier to identify patterns, hotspots, and areas of interest at different zoom levels.

Dashboards

Using Dash in combination with Plotly to put together a dashboard of visualization for presentation



Here we can see a Dash app that contains insightful information about taxi data aggregated monthly, daily and hourly for various metrics that can be changed with the click of a button. The interactivity allows for efficient capture of the data visually and it can be very useful for the visualization of big data.

METCS 777 A1 - Big Data Analytics
BU ID - U94893520

Evaluation of Technologies

In this paper we have covered the use of Apache Zeppelin for big data visualization by connecting directly to the EMR cluster to work with the large dataset. We also looked at Jupyter for implementing more customizable visualizations on geospatial data and aggregated data using Plotly, Folium and Dash. Apache Zeppelin's primary strength emerged in its ability to directly connect with EMR (Elastic MapReduce) clusters. This direct engagement allowed for a more streamlined visualization process for the large dataset. Additionally, Zeppelin's inbuilt interpreters for diverse big data frameworks ensured that even those unfamiliar with the intricacies of big data operations could intuitively interact with the massive volume of data.

Zeppelin’s dynamic forms added another layer of interactivity, allowing for more adaptive querying—a critical feature when visualizing such vast datasets in a notebook using SQL.
On the other hand, Jupyter's extensibility became evident when diving deeper into the intricacies of the NYC dataset. The geospatial data in the taxi trips was rendered using the Folium library within Jupyter, offering invaluable insights into spatial patterns, such as popular pickup or drop-off zones. Furthermore, the combination of Jupyter with Plotly and Dash transformed the visualizations from static representations to dynamic and interactive experiences. With Dash, the static data presentations evolved into immersive web applications, enabling readers to engage hands-on with the patterns, trends, and anomalies within the taxi data.

Zeppelin’s feature to connect directly to databases like MongoDB and Postgres makes it a very efficient tool to visualize and understand large sets of data by using SQL queries, this is not possible using Jupyter Notebook unless additional configurations and installations have been made. Zeppelin ensures the scalability of big data analysis and visualization and it can even be integrated with live databases where there is live data being fed into the database.

In conclusion, Zeppelin provided an experience tailored for comprehensive big data frameworks and the use of SQL in understanding the data. Jupyter's strengths, especially when enhanced by libraries like Plotly, Folium, and Dash, afforded a deeper, more customizable dive into the NYC taxi trip dataset. However, it is not recommended that Jupyter be used directly on the large data since it requires extra configurations that can be avoided by using Zeppelin. The project's

success relied on harnessing the unique strengths of each tool, ensuring that big data data visualizations were both insightful and user-friendly.

Code Examples:

Code Example 1: Creating Interactive Plotly graph on aggregated data with dropdown menu for different metrics. Trends in each day of the month across the entire dataset

```
fig_1 = go.Figure()


metrics = [
'daily_rides', 'daily_avg_fare', 'daily_avg_tip', 'daily_trip_duration',
'daily_avg_total_amount', 'avg_daily_profit_ratio', 'avg_daily_toll_amount',
]


colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b', '#e377c2']

# Add each metric as a separate trace to the figure
for i, metric in enumerate(metrics):
fig_1.add_trace(go.Scatter(
x=daily_data['pickup_day'],
y=daily_data[metric],
name=metric,
visible=False,
line=dict(color=colors[i], width=3),
marker=dict(size=6)
))


fig_1.data[0].visible = True


buttons = []
for i, metric in enumerate(metrics):
visible = [False] * len(metrics)
```

```
visible[i] = True
buttons.append(
dict(
label=metric,
method="update",
args=[{"visible": visible},
{"title": f"Daily Data - {metric.replace('_', ' ').title()}",
"xaxis": {"title": "Day of the Month"},
"yaxis": {"title": metric.replace('_', ' ').title()}}
]
)
)

# Update layout
fig_1.update_layout(
updatemenus=[
dict(
direction="down",
x=1.1,
y=1.15,
showactive=True,
buttons=buttons
)
],
template="plotly_dark",
xaxis_title="Day of the Month",
yaxis_title=metrics[0].replace('_', ' ').title(),
title=f"Daily Data - {metrics[0].replace('_', ' ').title()}",
margin=dict(t=100)
)

# Display the figure
fig_1.show()
```

METCS 777 A1 - Big Data Analytics
BU ID - U94893520

Code Example 2: SQL query running in a Zeppelin notebook querying data from the dataset

```
%sql
SELECT
    pickup_hour,
    AVG(hourly_avg_fare) as avg_fare
FROM hourly_aggregates_view
GROUP BY pickup_hour
ORDER BY avg_fare DESC
LIMIT 5;
```