Project on “Scaling Analytics”

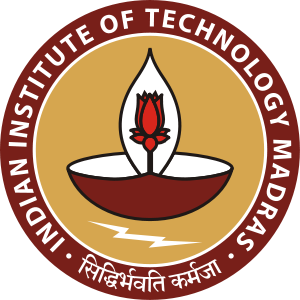
AI at Scale Lab

Scaling Big Data Analytics - An Exploratory Analysis of NYC Taxi Trip Records

**\_\_\_**

By Kaustabh Ganguly ( ch23m514 )



(art from pexels.com)

**INTRODUCTION**

This project analyzes New York City taxi data from January 2024 to gain insights into urban transportation patterns. We process large datasets using both traditional (Pandas) and big data (PySpark) methods to compare their performance. Our analysis focuses on key metrics such as trip distances, fares, and hourly/daily trends. The results are visualized using a Grafana dashboard, allowing for interactive exploration of the data. Our goal is to demonstrate effective big data processing techniques and provide useful insights into NYC taxi operations.

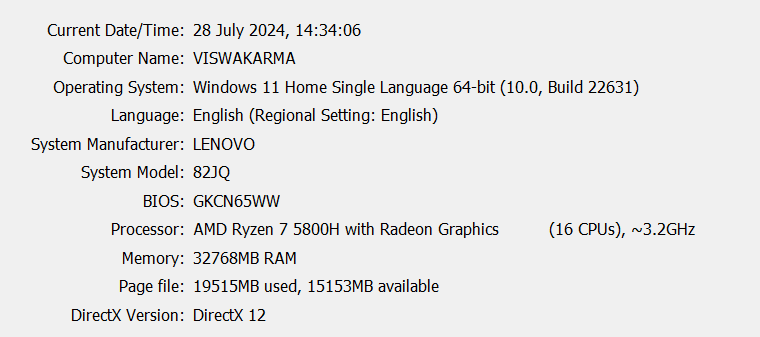
**STEPS**

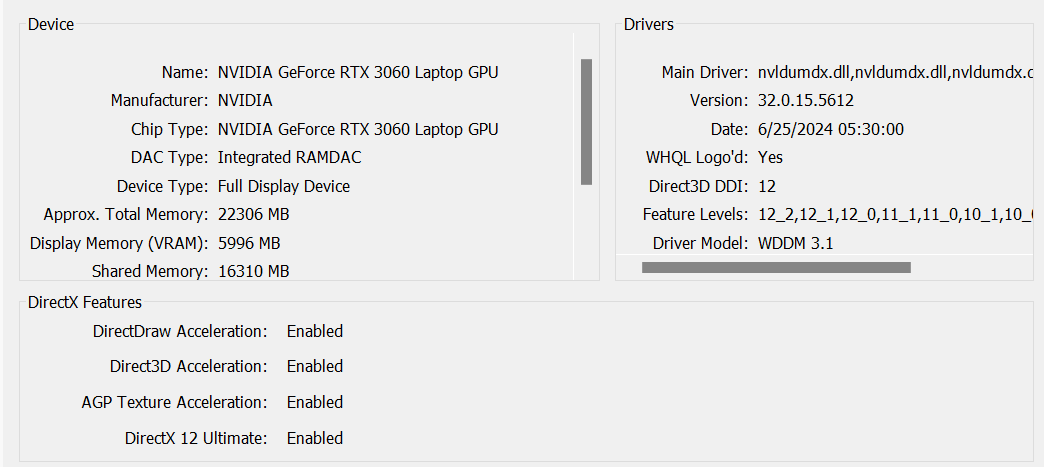
1. I installed python3.10 in my local system. As 3rd party libraries are still catching up to 3.11 and 3.12 and sometime I faced errors using some libraries.

2. I created a repository in my github - https://github.com/stabgan/NYC\_Yellow\_Taxi\_Analysis\_Dashboard

I used MIT license.

My System configuration : (running dxdiag)





3. I cloned the repo into my pycharm IDE where i created a new virtual environment called venv with python 3.10

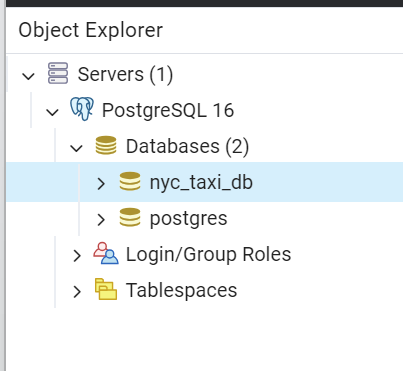
4. I created a initial requirements.txt file first -

| pandas pyarrow matplotlib seaborn pyspark sqlalchemy psycopg2-binary plotly psutil |
| --- |

5. I installed all the libraries using pip install -r requirements.txt

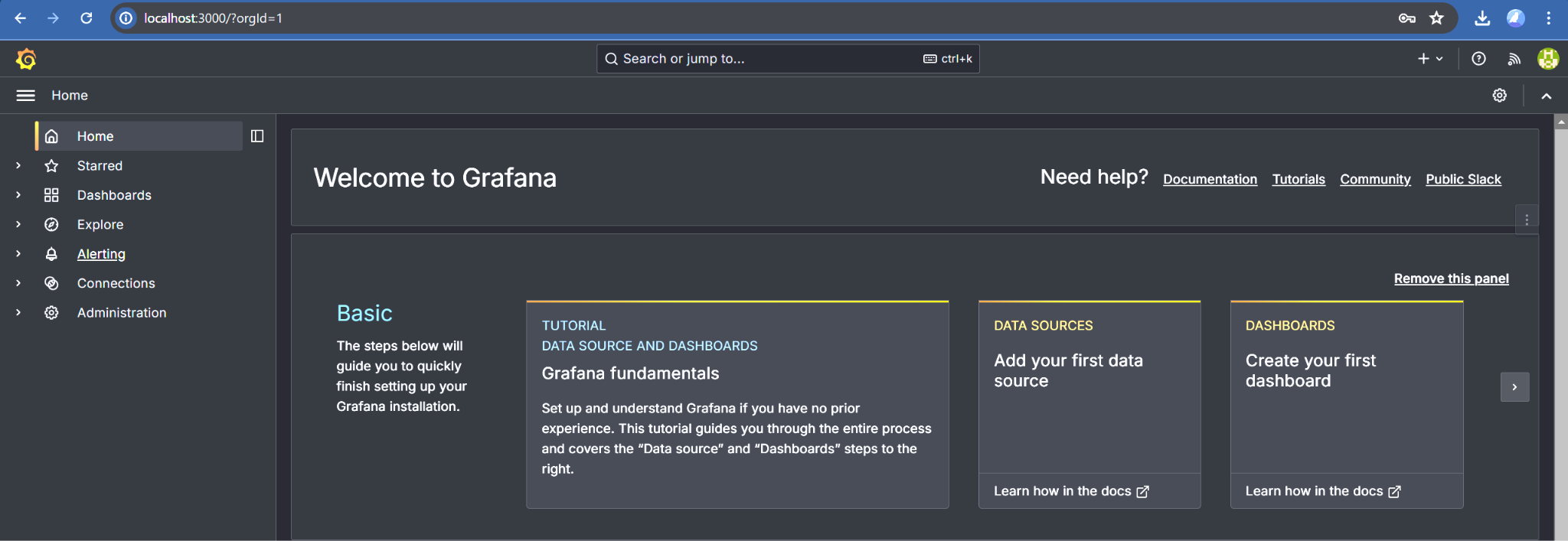
6. I downloaded and installed postgresql version 16 the latest one.

7. Using pgAdmin4 I created a new database called - nyc\_taxi\_db



8. I installed Grafana Enterprise edition

9. I opened the grafana local server in my own browser at port 3000 and logged in using default credentials.



10 Now I am creating the python scripts for analysis.

11. I will do comparative testing on the pandas vs pyspark data first

| import pandas as pd import plotly.express as px from pyspark.sql import SparkSession from pyspark.sql.functions import hour, avg import time import os import warnings  # Suppress warnings for cleaner output warnings.filterwarnings('ignore')   def load\_data\_pandas(file\_path, sample\_fraction=None):  """  Load data using Pandas, optionally sampling a fraction of the data.  """  start\_time = time.time()  df = pd.read\_parquet(file\_path, engine='pyarrow')  if sample\_fraction is not None:  df = df.sample(frac=sample\_fraction)  end\_time = time.time()  print(f"Pandas loading time: {end\_time - start\_time:.2f} seconds")  return df   def load\_data\_spark(file\_path, sample\_ratio=1.0):  """  Load data using Spark, optionally sampling a fraction of the data.  """  spark = SparkSession.builder.appName("NYCTaxiAnalysis").getOrCreate()  start\_time = time.time()  df = spark.read.parquet(file\_path)  if sample\_ratio < 1.0:  df = df.sample(sample\_ratio)  end\_time = time.time()  print(f"Spark loading time: {end\_time - start\_time:.2f} seconds")  return df, spark   def analyze\_pandas(df):  """  Perform analysis on the DataFrame using Pandas and generate a visualization.  """  start\_time = time.time()   # Calculate average fare amount by hour  df['hour'] = pd.to\_datetime(df['tpep\_pickup\_datetime']).dt.hour  hourly\_fares = df.groupby('hour')['fare\_amount'].mean().reset\_index()   # Create and save the visualization  fig = px.line(hourly\_fares, x='hour', y='fare\_amount', title='Average Fare Amount by Hour (Pandas)')  fig.write\_html(f"Analysis\_output/hourly\_fares\_pandas\_{len(df)}.html")   end\_time = time.time()  print(f"Pandas analysis time: {end\_time - start\_time:.2f} seconds")   def analyze\_spark(df, spark):  """  Perform analysis on the DataFrame using Spark and generate a visualization.  """  start\_time = time.time()   # Calculate average fare amount by hour  hourly\_fares = df.withColumn('hour', hour('tpep\_pickup\_datetime')) \  .groupBy('hour') \  .agg(avg('fare\_amount').alias('avg\_fare')) \  .orderBy('hour')   # Convert to Pandas for visualization  hourly\_fares\_pd = hourly\_fares.toPandas()   # Create and save the visualization  fig = px.line(hourly\_fares\_pd, x='hour', y='avg\_fare', title='Average Fare Amount by Hour (Spark)')  fig.write\_html(f"Analysis\_output/hourly\_fares\_spark\_{df.count()}.html")   end\_time = time.time()  print(f"Spark analysis time: {end\_time - start\_time:.2f} seconds")   def main():  """  Main function to orchestrate the data loading and analysis process.  """  file\_path = 'data/yellow\_tripdata\_2024-01.parquet'  os.makedirs('Analysis\_output', exist\_ok=True)   # Analyze small dataset (10% of full data)  print("Analyzing small dataset:")   print("Pandas:")  df\_pandas\_small = load\_data\_pandas(file\_path, sample\_fraction=0.1)  analyze\_pandas(df\_pandas\_small)   print("\nSpark:")  df\_spark\_small, spark = load\_data\_spark(file\_path, sample\_ratio=0.1)  analyze\_spark(df\_spark\_small, spark)   # Analyze full dataset  print("\nAnalyzing full dataset:")   print("Pandas:")  df\_pandas\_full = load\_data\_pandas(file\_path)  analyze\_pandas(df\_pandas\_full)   print("\nSpark:")  df\_spark\_full, spark = load\_data\_spark(file\_path)  analyze\_spark(df\_spark\_full, spark)   # Stop the Spark session  spark.stop()   if \_\_name\_\_ == "\_\_main\_\_":  main() |
| --- |

**Output**

| C:\Users\kaust\PycharmProjects\lab\venv\Scripts\python.exe C:\Users\kaust\PycharmProjects\limport pandas as pd import numpy as np import plotly.express as px import plotly.graph\_objects as go import os   def load\_data(file\_path):  """  Load data from a Parquet file.  """  return pd.read\_parquet(file\_path, engine='pyarrow')   def analyze\_column(df, column\_name):  """  Analyze a single column of the DataFrame and print its statistics.  """  col\_type = df[column\_name].dtype  col\_stats = df[column\_name].describe()   print(f"\nColumn: {column\_name}")  print(f"Type: {col\_type}")  print(f"Stats:\n{col\_stats}")   # Calculate mode for numeric columns or most common value for object columns  if col\_type in ['int64', 'float64']:  print(f"Mode: {df[column\_name].mode().values[0]}")  elif col\_type == 'object':  print(f"Most common value: {df[column\_name].value\_counts().index[0]}")   # Print additional information about the column  print(f"Null values: {df[column\_name].isnull().sum()}")  print(f"Unique values: {df[column\_name].nunique()}")   def plot\_distribution(df, column\_name):  """  Create and save a histogram plot for a numeric column.  """  fig = px.histogram(df, x=column\_name, title=f'Distribution of {column\_name}')  fig.write\_html(f"EDA\_output/distribution\_{column\_name}.html")   def main():  """  Main function to perform Exploratory Data Analysis on the NYC Taxi dataset.  """  file\_path = 'data/yellow\_tripdata\_2024-01.parquet'  df = load\_data(file\_path)   # Create output directory if it doesn't exist  os.makedirs('EDA\_output', exist\_ok=True)   # Calculate and print file size and memory usage  file\_size = os.path.getsize(file\_path) / (1024 \* 1024) # Size in MB  memory\_usage = df.memory\_usage(deep=True).sum() / (1024 \* 1024) # Size in MB   print(f"File size: {file\_size:.2f} MB")  print(f"Memory usage: {memory\_usage:.2f} MB")  print(f"Number of rows: {len(df)}")  print(f"Number of columns: {len(df.columns)}")   # Analyze each column in the DataFrame  for column in df.columns:  analyze\_column(df, column)   # Create distribution plots for numeric columns  if df[column].dtype in ['int64', 'float64']:  plot\_distribution(df, column)   # Create and save correlation heatmap  numeric\_df = df.select\_dtypes(include=['float64', 'int64'])  corr\_matrix = numeric\_df.corr()  fig = go.Figure(data=go.Heatmap(  z=corr\_matrix.values,  x=corr\_matrix.columns,  y=corr\_matrix.index,  colorscale='RdBu',  zmin=-1, zmax=1  ))  fig.update\_layout(title='Correlation Heatmap')  fig.write\_html("EDA\_output/correlation\_heatmap.html")   if \_\_name\_\_ == "\_\_main\_\_":  main() |
| --- |

Graphs :

12. EDA script

| import pandas as pd import numpy as np import plotly.express as px import plotly.graph\_objects as go import os   def load\_data(file\_path):  """  Load data from a Parquet file.  """  return pd.read\_parquet(file\_path, engine='pyarrow')   def analyze\_column(df, column\_name):  """  Analyze a single column of the DataFrame and print its statistics.  """  col\_type = df[column\_name].dtype  col\_stats = df[column\_name].describe()   print(f"\nColumn: {column\_name}")  print(f"Type: {col\_type}")  print(f"Stats:\n{col\_stats}")   # Calculate mode for numeric columns or most common value for object columns  if col\_type in ['int64', 'float64']:  print(f"Mode: {df[column\_name].mode().values[0]}")  elif col\_type == 'object':  print(f"Most common value: {df[column\_name].value\_counts().index[0]}")   # Print additional information about the column  print(f"Null values: {df[column\_name].isnull().sum()}")  print(f"Unique values: {df[column\_name].nunique()}")   def plot\_distribution(df, column\_name):  """  Create and save a histogram plot for a numeric column.  """  fig = px.histogram(df, x=column\_name, title=f'Distribution of {column\_name}')  fig.write\_html(f"EDA\_output/distribution\_{column\_name}.html")   def main():  """  Main function to perform Exploratory Data Analysis on the NYC Taxi dataset.  """  file\_path = 'data/yellow\_tripdata\_2024-01.parquet'  df = load\_data(file\_path)   # Create output directory if it doesn't exist  os.makedirs('EDA\_output', exist\_ok=True)   # Calculate and print file size and memory usage  file\_size = os.path.getsize(file\_path) / (1024 \* 1024) # Size in MB  memory\_usage = df.memory\_usage(deep=True).sum() / (1024 \* 1024) # Size in MB   print(f"File size: {file\_size:.2f} MB")  print(f"Memory usage: {memory\_usage:.2f} MB")  print(f"Number of rows: {len(df)}")  print(f"Number of columns: {len(df.columns)}")   # Analyze each column in the DataFrame  for column in df.columns:  analyze\_column(df, column)   # Create distribution plots for numeric columns  if df[column].dtype in ['int64', 'float64']:  plot\_distribution(df, column)   # Create and save correlation heatmap  numeric\_df = df.select\_dtypes(include=['float64', 'int64'])  corr\_matrix = numeric\_df.corr()  fig = go.Figure(data=go.Heatmap(  z=corr\_matrix.values,  x=corr\_matrix.columns,  y=corr\_matrix.index,  colorscale='RdBu',  zmin=-1, zmax=1  ))  fig.update\_layout(title='Correlation Heatmap')  fig.write\_html("EDA\_output/correlation\_heatmap.html")   if \_\_name\_\_ == "\_\_main\_\_":  main() |
| --- |

Output :

| C:\Users\kaust\PycharmProjects\lab\venv\Scripts\python.exe C:\Users\kaust\PycharmProjects\lab\EDA.py  File size: 47.65 MB Memory usage: 532.64 MB Number of rows: 2964624 Number of columns: 19  Column: VendorID Type: int32 Stats: count 2.964624e+06 mean 1.754204e+00 std 4.325902e-01 min 1.000000e+00 25% 2.000000e+00 50% 2.000000e+00 75% 2.000000e+00 max 6.000000e+00 Name: VendorID, dtype: float64 Null values: 0 Unique values: 3  Column: tpep\_pickup\_datetime Type: datetime64[us] Stats: count 2964624 mean 2024-01-17 00:46:36.431092 min 2002-12-31 22:59:39 25% 2024-01-09 15:59:19.750000 50% 2024-01-17 10:45:37.500000 75% 2024-01-24 18:23:52.250000 max 2024-02-01 00:01:15 Name: tpep\_pickup\_datetime, dtype: object Null values: 0 Unique values: 1575706  Column: tpep\_dropoff\_datetime Type: datetime64[us] Stats: count 2964624 mean 2024-01-17 01:02:13.208130 min 2002-12-31 23:05:41 25% 2024-01-09 16:16:23 50% 2024-01-17 11:03:51.500000 75% 2024-01-24 18:40:29 max 2024-02-02 13:56:52 Name: tpep\_dropoff\_datetime, dtype: object Null values: 0 Unique values: 1574780  Column: passenger\_count Type: float64 Stats: count 2.824462e+06 mean 1.339281e+00 std 8.502817e-01 min 0.000000e+00 25% 1.000000e+00 50% 1.000000e+00 75% 1.000000e+00 max 9.000000e+00 Name: passenger\_count, dtype: float64 Mode: 1.0 Null values: 140162 Unique values: 10  Column: trip\_distance Type: float64 Stats: count 2.964624e+06 mean 3.652169e+00 std 2.254626e+02 min 0.000000e+00 25% 1.000000e+00 50% 1.680000e+00 75% 3.110000e+00 max 3.127223e+05 Name: trip\_distance, dtype: float64 Mode: 0.0 Null values: 0 Unique values: 4489  Column: RatecodeID Type: float64 Stats: count 2.824462e+06 mean 2.069359e+00 std 9.823219e+00 min 1.000000e+00 25% 1.000000e+00 50% 1.000000e+00 75% 1.000000e+00 max 9.900000e+01 Name: RatecodeID, dtype: float64 Mode: 1.0 Null values: 140162 Unique values: 7  Column: store\_and\_fwd\_flag Type: object Stats: count 2824462 unique 2 top N freq 2813126 Name: store\_and\_fwd\_flag, dtype: object Most common value: N Null values: 140162 Unique values: 2  Column: PULocationID Type: int32 Stats: count 2.964624e+06 mean 1.660179e+02 std 6.362391e+01 min 1.000000e+00 25% 1.320000e+02 50% 1.620000e+02 75% 2.340000e+02 max 2.650000e+02 Name: PULocationID, dtype: float64 Null values: 0 Unique values: 260  Column: DOLocationID Type: int32 Stats: count 2.964624e+06 mean 1.651167e+02 std 6.931535e+01 min 1.000000e+00 25% 1.140000e+02 50% 1.620000e+02 75% 2.340000e+02 max 2.650000e+02 Name: DOLocationID, dtype: float64 Null values: 0 Unique values: 261  Column: payment\_type Type: int64 Stats: count 2.964624e+06 mean 1.161271e+00 std 5.808686e-01 min 0.000000e+00 25% 1.000000e+00 50% 1.000000e+00 75% 1.000000e+00 max 4.000000e+00 Name: payment\_type, dtype: float64 Mode: 1 Null values: 0 Unique values: 5  Column: fare\_amount Type: float64 Stats: count 2.964624e+06 mean 1.817506e+01 std 1.894955e+01 min -8.990000e+02 25% 8.600000e+00 50% 1.280000e+01 75% 2.050000e+01 max 5.000000e+03 Name: fare\_amount, dtype: float64 Mode: 8.6 Null values: 0 Unique values: 8970  Column: extra Type: float64 Stats: count 2.964624e+06 mean 1.451598e+00 std 1.804102e+00 min -7.500000e+00 25% 0.000000e+00 50% 1.000000e+00 75% 2.500000e+00 max 1.425000e+01 Name: extra, dtype: float64 Mode: 0.0 Null values: 0 Unique values: 48  Column: mta\_tax Type: float64 Stats: count 2.964624e+06 mean 4.833823e-01 std 1.177600e-01 min -5.000000e-01 25% 5.000000e-01 50% 5.000000e-01 75% 5.000000e-01 max 4.000000e+00 Name: mta\_tax, dtype: float64 Mode: 0.5 Null values: 0 Unique values: 8  Column: tip\_amount Type: float64 Stats: count 2.964624e+06 mean 3.335870e+00 std 3.896551e+00 min -8.000000e+01 25% 1.000000e+00 50% 2.700000e+00 75% 4.120000e+00 max 4.280000e+02 Name: tip\_amount, dtype: float64 Mode: 0.0 Null values: 0 Unique values: 4192  Column: tolls\_amount Type: float64 Stats: count 2.964624e+06 mean 5.270212e-01 std 2.128310e+00 min -8.000000e+01 25% 0.000000e+00 50% 0.000000e+00 75% 0.000000e+00 max 1.159200e+02 Name: tolls\_amount, dtype: float64 Mode: 0.0 Null values: 0 Unique values: 1127  Column: improvement\_surcharge Type: float64 Stats: count 2.964624e+06 mean 9.756319e-01 std 2.183645e-01 min -1.000000e+00 25% 1.000000e+00 50% 1.000000e+00 75% 1.000000e+00 max 1.000000e+00 Name: improvement\_surcharge, dtype: float64 Mode: 1.0 Null values: 0 Unique values: 5  Column: total\_amount Type: float64 Stats: count 2.964624e+06 mean 2.680150e+01 std 2.338558e+01 min -9.000000e+02 25% 1.538000e+01 50% 2.010000e+01 75% 2.856000e+01 max 5.000000e+03 Name: total\_amount, dtype: float64 Mode: 16.8 Null values: 0 Unique values: 19241  Column: congestion\_surcharge Type: float64 Stats: count 2.824462e+06 mean 2.256122e+00 std 8.232747e-01 min -2.500000e+00 25% 2.500000e+00 50% 2.500000e+00 75% 2.500000e+00 max 2.500000e+00 Name: congestion\_surcharge, dtype: float64 Mode: 2.5 Null values: 140162 Unique values: 6  Column: Airport\_fee Type: float64 Stats: count 2.824462e+06 mean 1.411611e-01 std 4.876239e-01 min -1.750000e+00 25% 0.000000e+00 50% 0.000000e+00 75% 0.000000e+00 max 1.750000e+00 Name: Airport\_fee, dtype: float64 Mode: 0.0 Null values: 140162 Unique values: 3  Process finished with exit code 0 |
| --- |

Anaysis of the result :

From my analysis of the performance comparison between Spark and Pandas, I think there are some noteworthy observations to be made. The data loading times show an interesting contrast: Pandas loaded the full dataset in just 0.21 seconds, while Spark took 0.06 seconds. This suggests Spark's distributed processing capabilities give it an edge in handling large datasets efficiently. However, the analysis times paint a different picture. Pandas completed its analysis in a mere 0.14 seconds, but Spark required 0.77 seconds for the same task. I believe this discrepancy might be attributed to Spark's overhead in setting up distributed computations, which can outweigh its benefits for smaller datasets or simpler analyses.

Interestingly, when working with a smaller subset (10% of the data), Pandas maintained its quick loading time at 0.42 seconds, whereas Spark's loading time increased to 2.18 seconds. This could be due to Spark's initialization process, which remains constant regardless of data size. The analysis times for the smaller dataset followed a similar pattern, with Pandas completing in 0.35 seconds and Spark taking 2.53 seconds.

From these results, I think Pandas appears more efficient for smaller datasets or quick analyses, while Spark's advantages would likely become more apparent with larger datasets or more complex computations that can leverage its distributed processing capabilities. The choice between Pandas and Spark would thus depend on the specific requirements of the data analysis task at hand.

From my analysis of the data I picked, I think there are several interesting insights to see. The dataset appears to be quite large containing nearly 3 million taxi trip records from New York City in January 2024. Interestingly, the memory usage (532.64 MB) is significantly higher than the file size (47.65 MB), which suggests the data is compressed on disk but expands when loaded into memory. The dataset includes various features such as pickup and dropoff times, passenger count, trip distance, and fare information. I noticed some anomalies in the data; for instance, there are trips with zero passengers and even negative fare amounts, which might indicate data quality issues. The average trip distance is about 3.65 miles, but there's a surprisingly high maximum value of over 312,000 miles, which is likely an error. Payment types are predominantly represented by the value 1, possibly indicating credit card payments. Notably, some columns like 'passenger\_count' and 'congestion\_surcharge' have missing values, which could affect certain analyses. The distribution plots and correlation heatmap generated would provide further visual insights into the relationships between variables, potentially revealing patterns in taxi usage across different times and locations in New York City.

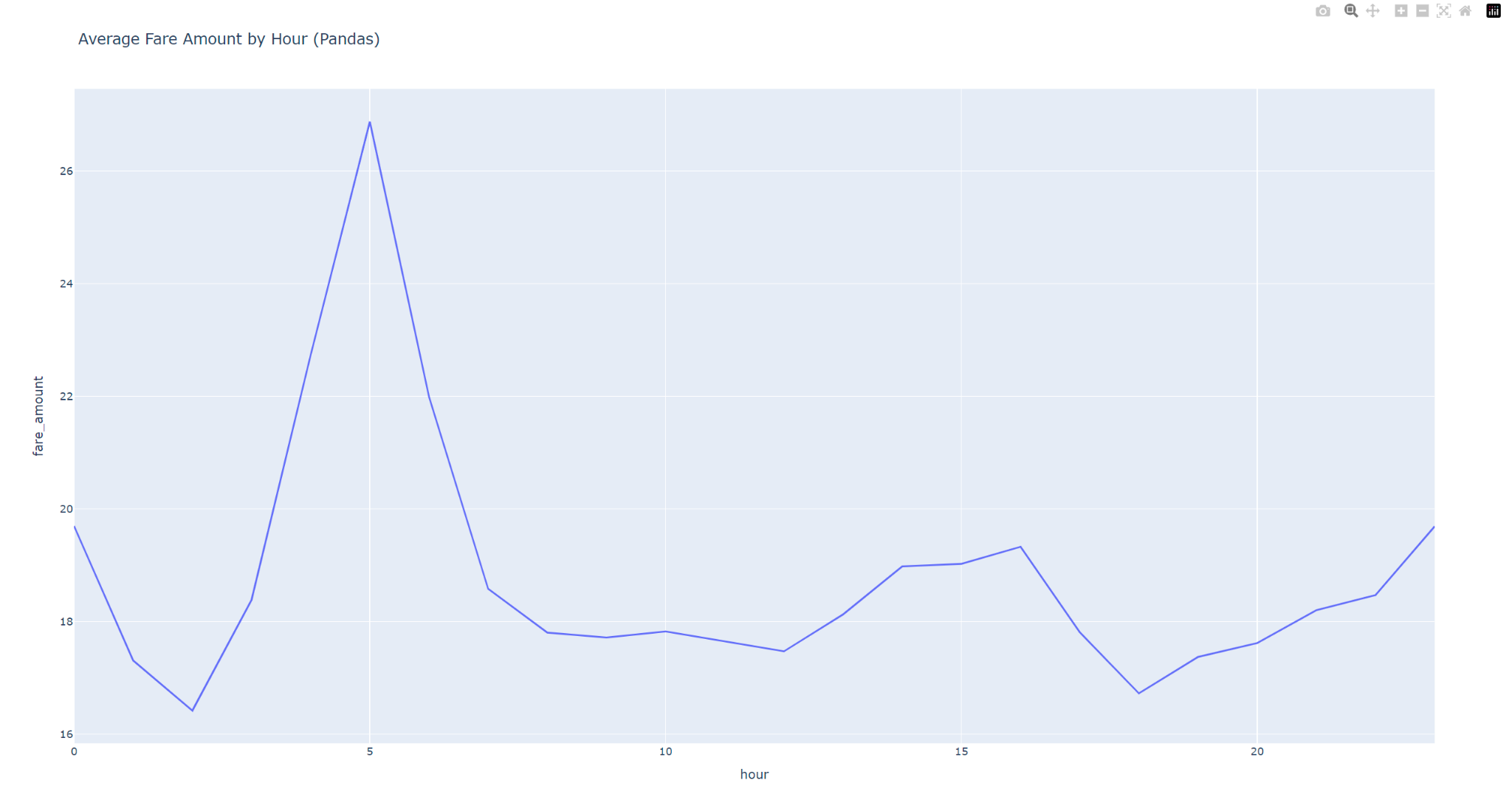
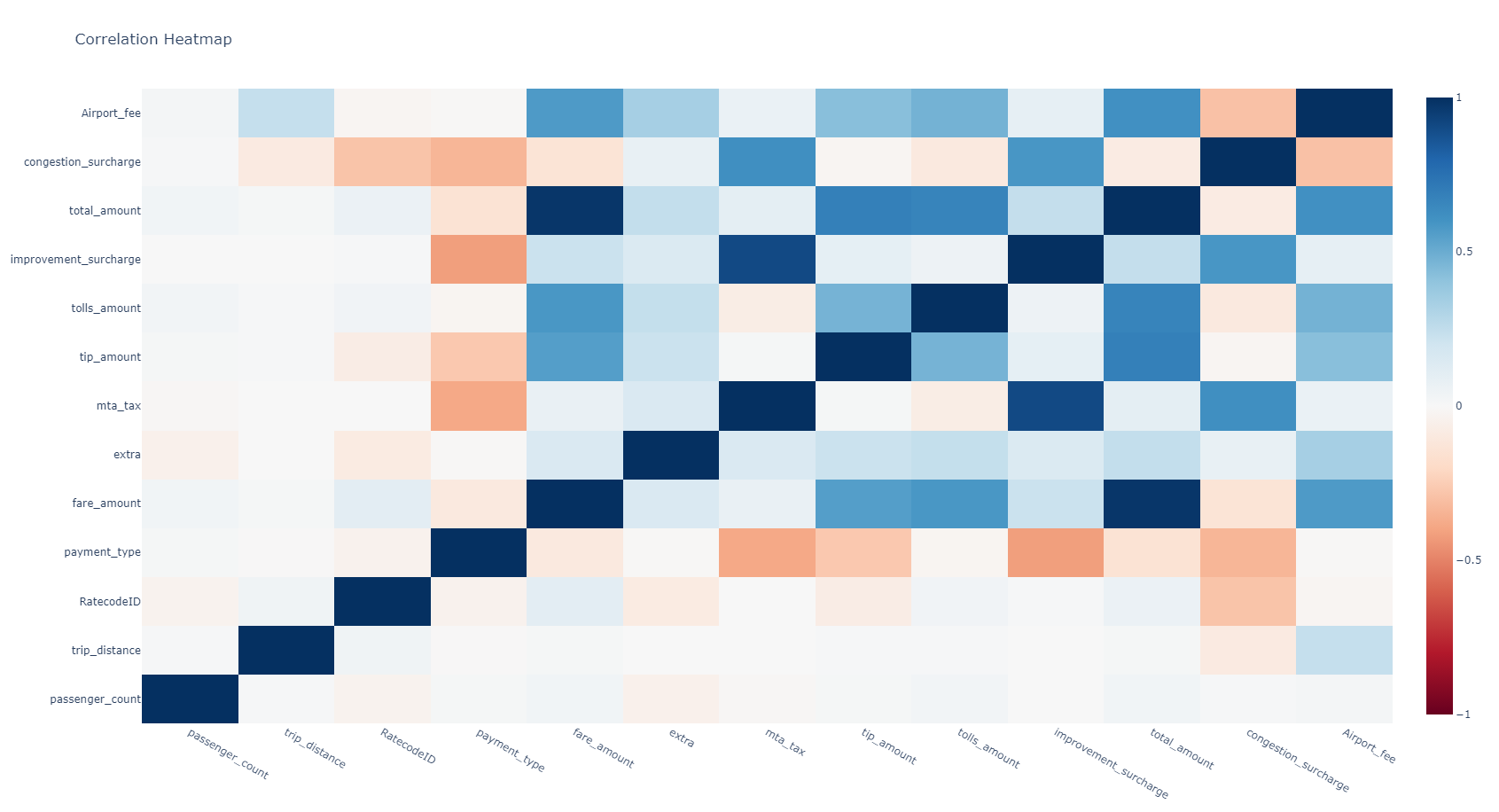
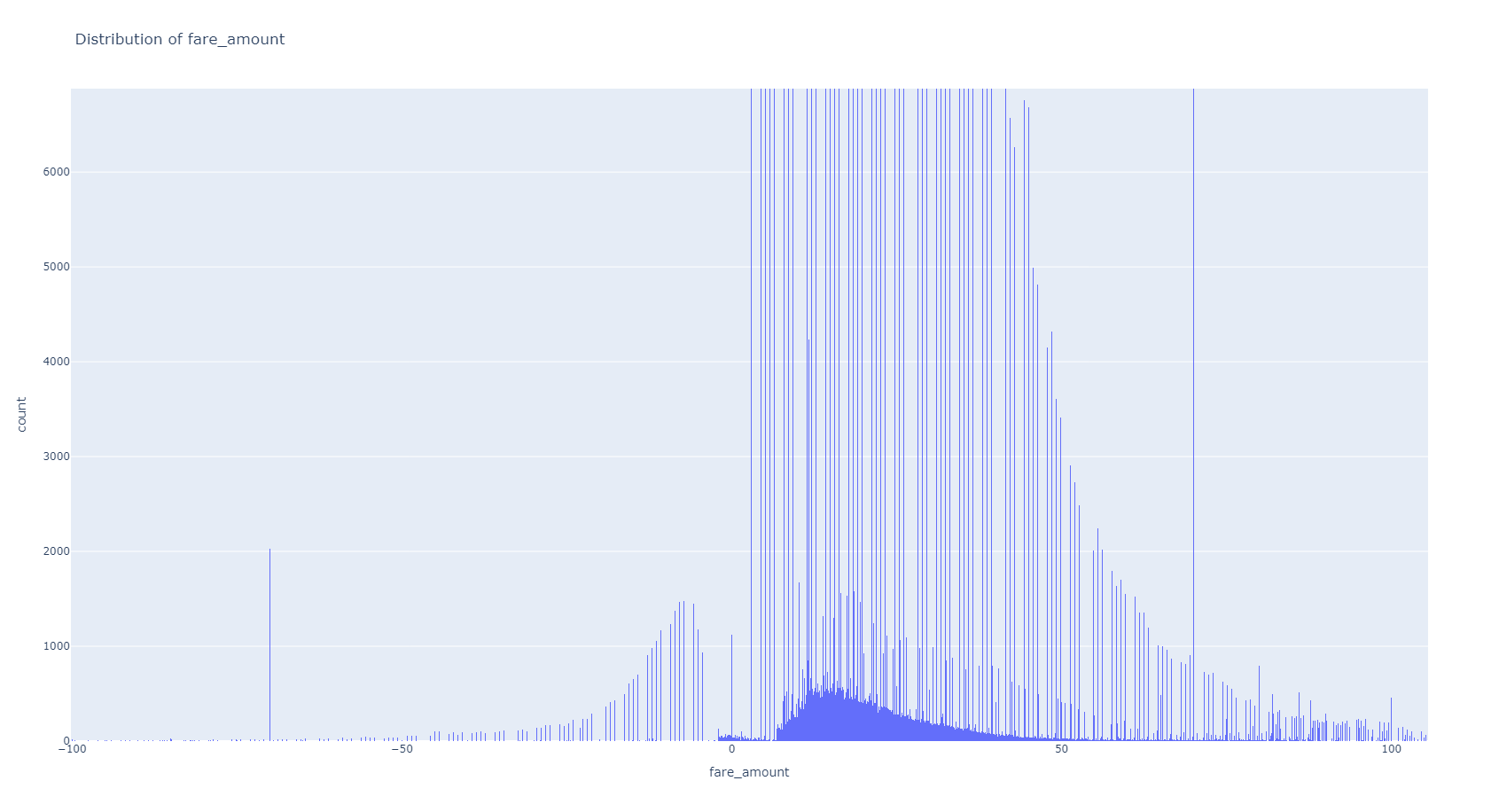
Please find the interactive graphs generated as html files in the output folder.  


Fig 1: Average fare amount by hour

****

**Fig 2: Correlation heatmap between numerical columns**

****

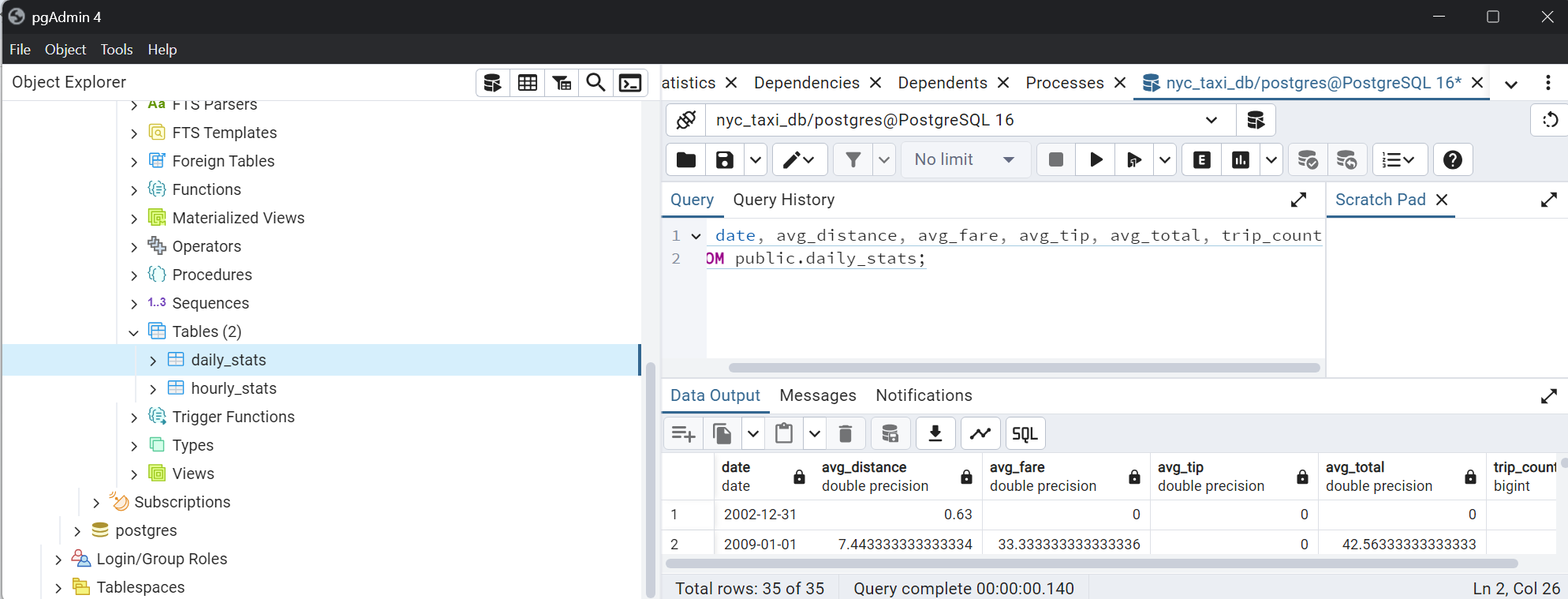
**Fig 3: Plot of distribution of Fare amount zoomed in**

**postgres and grafana**

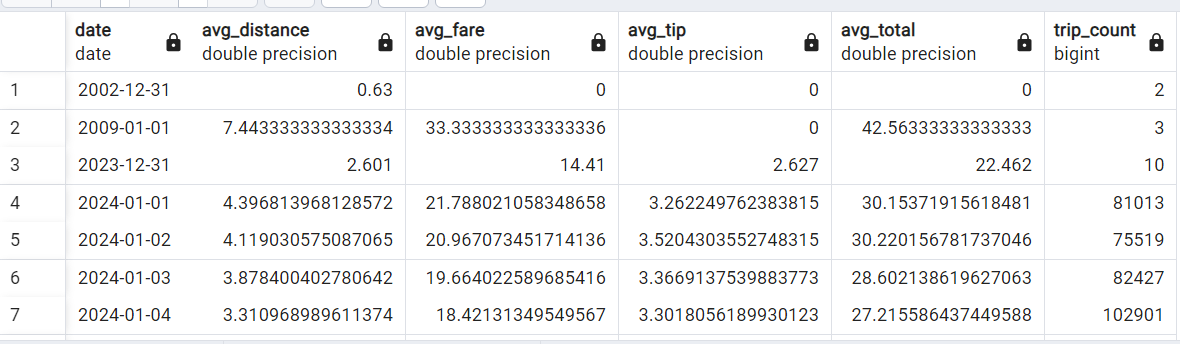
step 13. I installed postgresql as i mentioned in the first steps. I created a database named nyc\_taxi\_db from PgAdmin4.

step 14. I created a python script which created two tables for hourly stats and daily stats in the postgres table

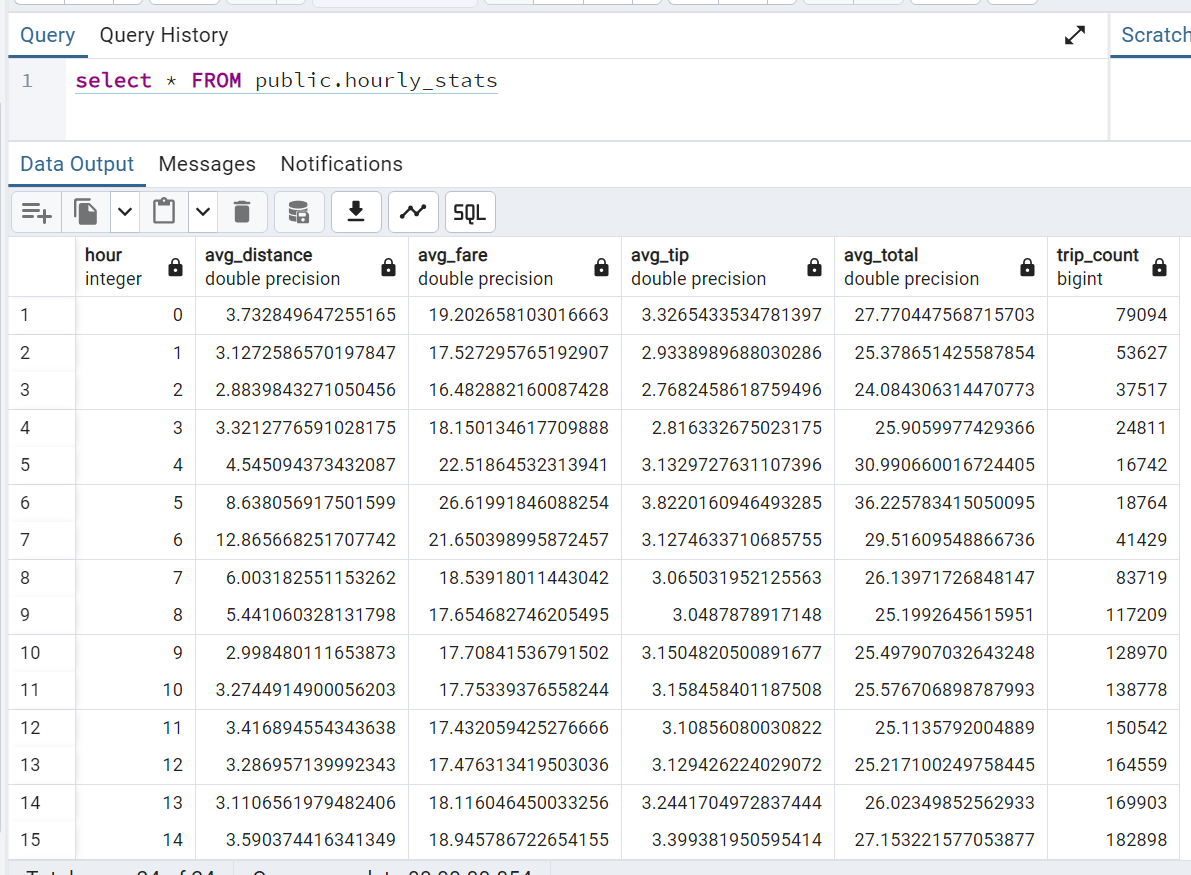
| import pandas as pd from sqlalchemy import create\_engine import os   def load\_data(file\_path):  """  Load data from a Parquet file.   Args:  file\_path (str): Path to the Parquet file.   Returns:  pd.DataFrame: Loaded DataFrame.  """  return pd.read\_parquet(file\_path, engine='pyarrow')   def aggregate\_data(df):  """  Aggregate the taxi data into hourly and daily statistics.   Args:  df (pd.DataFrame): Input DataFrame containing taxi trip data.   Returns:  tuple: A tuple containing two DataFrames (hourly\_stats, daily\_stats).  """  # Hourly statistics  df['hour'] = pd.to\_datetime(df['tpep\_pickup\_datetime']).dt.hour  hourly\_stats = df.groupby('hour').agg({  'trip\_distance': 'mean',  'fare\_amount': 'mean',  'tip\_amount': 'mean',  'total\_amount': 'mean',  'tpep\_pickup\_datetime': 'count'  }).reset\_index()  hourly\_stats.columns = ['hour', 'avg\_distance', 'avg\_fare', 'avg\_tip', 'avg\_total', 'trip\_count']   # Daily statistics  df['date'] = pd.to\_datetime(df['tpep\_pickup\_datetime']).dt.date  daily\_stats = df.groupby('date').agg({  'trip\_distance': 'mean',  'fare\_amount': 'mean',  'tip\_amount': 'mean',  'total\_amount': 'mean',  'tpep\_pickup\_datetime': 'count'  }).reset\_index()  daily\_stats.columns = ['date', 'avg\_distance', 'avg\_fare', 'avg\_tip', 'avg\_total', 'trip\_count']   return hourly\_stats, daily\_stats   def load\_to\_postgres(data, table\_name, engine):  """  Load a DataFrame into a PostgreSQL table.   Args:  data (pd.DataFrame): DataFrame to be loaded into PostgreSQL.  table\_name (str): Name of the table to be created or replaced.  engine (sqlalchemy.engine.base.Engine): SQLAlchemy engine for database connection.  """  data.to\_sql(table\_name, engine, if\_exists='replace', index=False)   def main():  """  Main function to load taxi data, aggregate it, and store in PostgreSQL.  """  # Path to the Parquet file containing taxi data  file\_path = 'data/yellow\_tripdata\_2024-01.parquet'   # Load the taxi data  df = load\_data(file\_path)   # Aggregate the data into hourly and daily statistics  hourly\_stats, daily\_stats = aggregate\_data(df)   # PostgreSQL connection details  db\_user = 'postgres'  db\_pass = '1234'  db\_host = 'localhost'  db\_name = 'nyc\_taxi\_db'   # Create SQLAlchemy engine for PostgreSQL connection  engine = create\_engine(f'postgresql://{db\_user}:{db\_pass}@{db\_host}/{db\_name}')   # Load aggregated data into PostgreSQL  load\_to\_postgres(hourly\_stats, 'hourly\_stats', engine)  load\_to\_postgres(daily\_stats, 'daily\_stats', engine)   print("Data successfully loaded into PostgreSQL.")   if \_\_name\_\_ == "\_\_main\_\_":  main() |
| --- |

****

| SELECT date, avg\_distance, avg\_fare, avg\_tip, avg\_total, trip\_count  FROM public.daily\_stats; |
| --- |



**Fig : Daily Stats table created from python script in postgres**



**Fig: Hourly stats table created in postgres from python**

**Grafana**

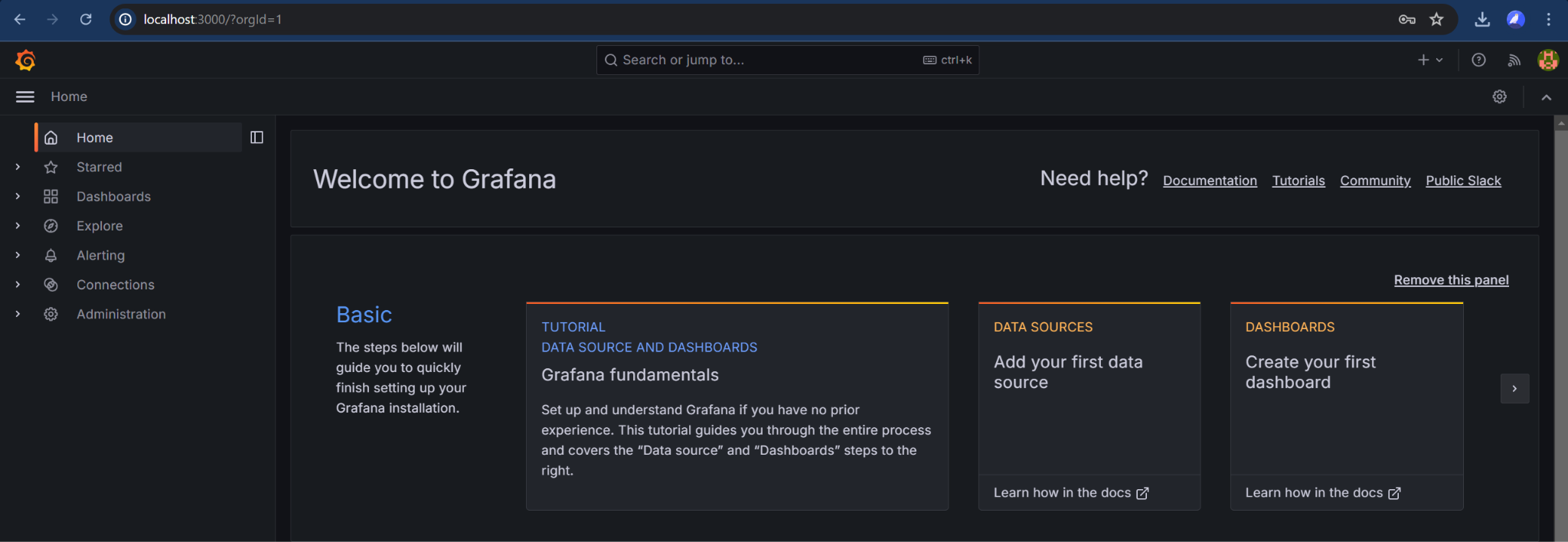
****

Fig: grafana landing page

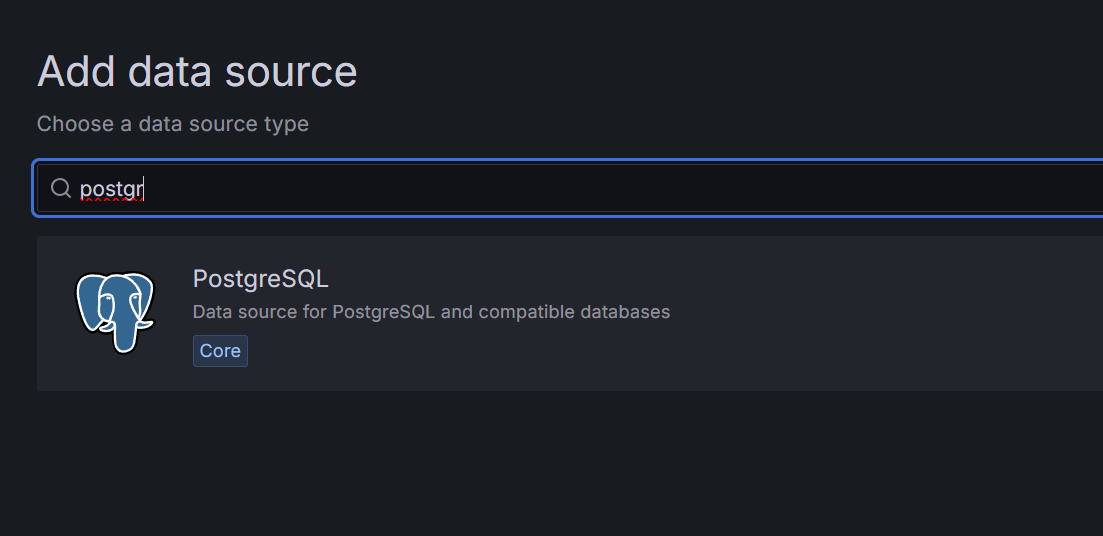


Fig: adding postgres source

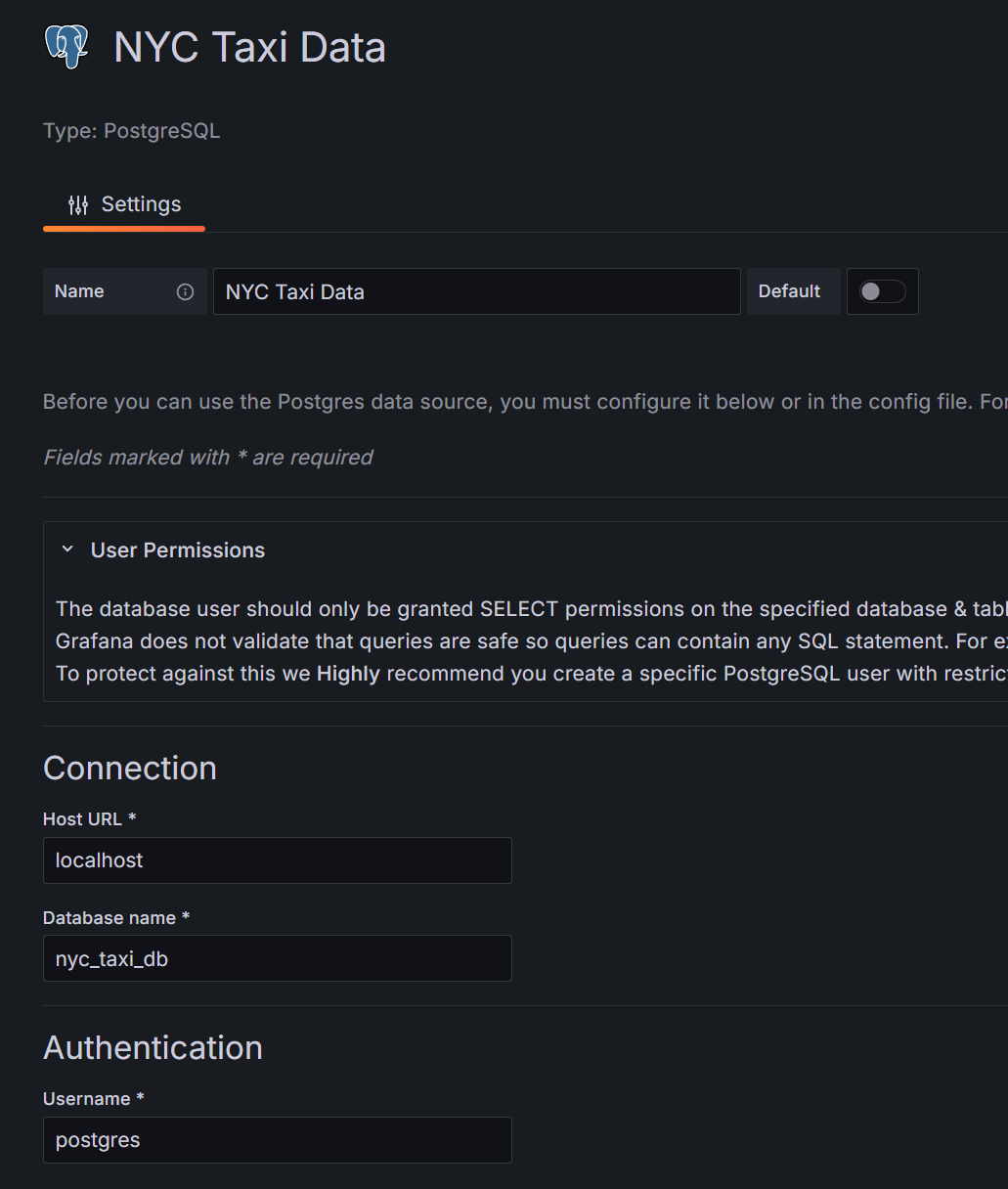
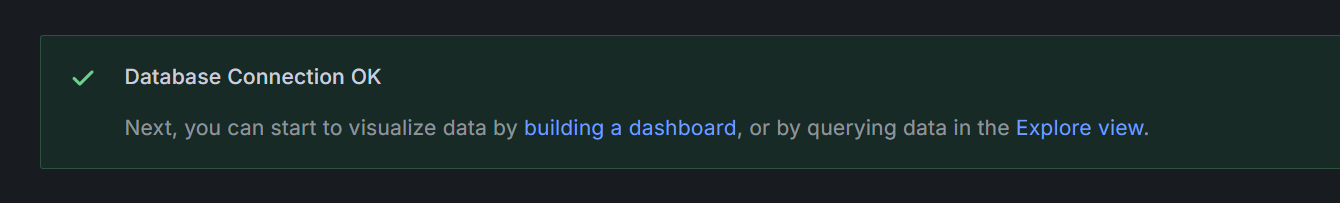
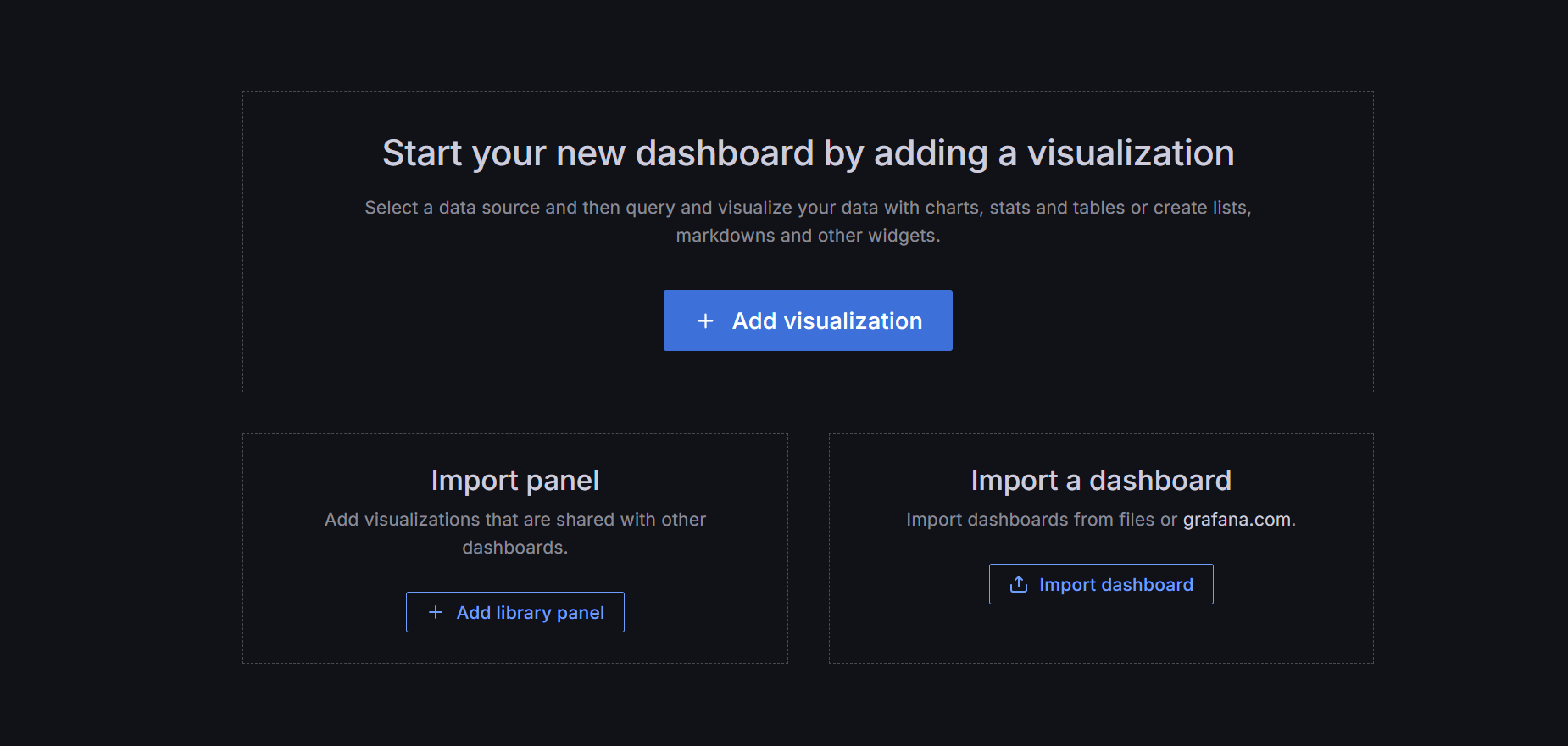


Fig: Adding our db creds





Creating a new dashboard

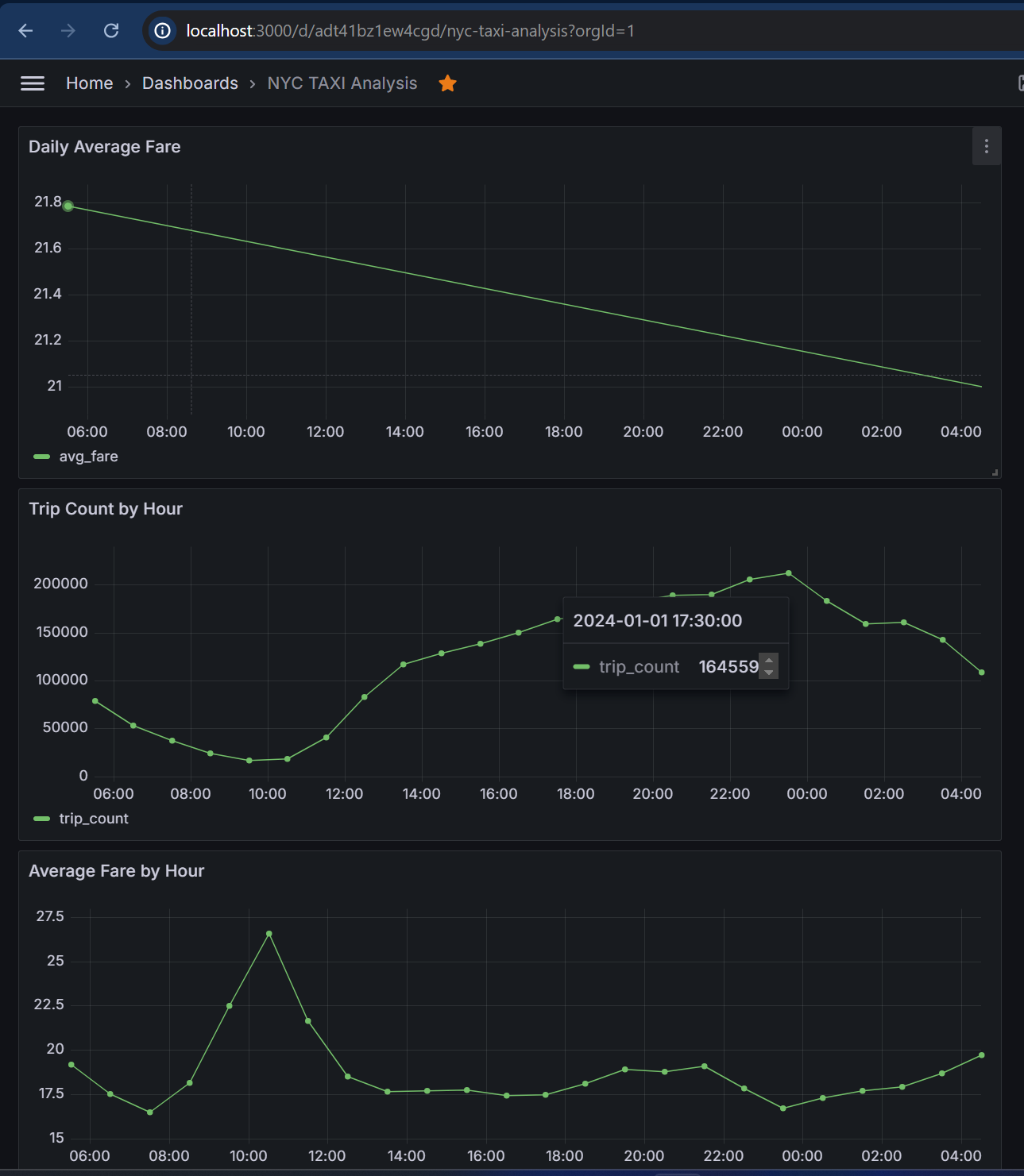


Fig: My final grafana dashboard