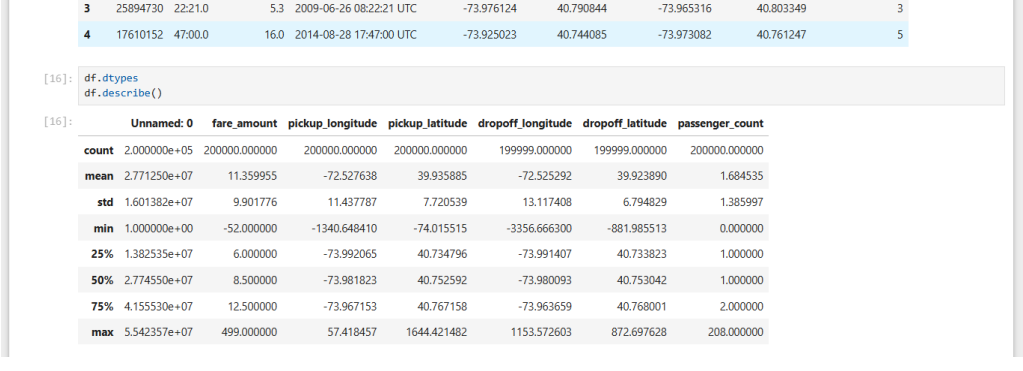
Data loading process

The raw Uber Fares dataset was loaded into a Python environment using Pandas for preprocessing. The cleaned and enhanced CSV file was later imported into Power BI using the “Get Data” → “Text/CSV” option.

```
[4]: import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns

[8]: import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     %matplotlib inline

     print("Libraries imported successfully!")

     Libraries imported successfully!

[10]: df = pd.read_csv("uber.csv")

[12]: df.shape

[12]: (200000, 9)

[14]: df.info()
      df.head()

      <class 'pandas.core.frame.DataFrame'>
      RangeIndex: 200000 entries, 0 to 199999
      Data columns (total 9 columns):
       #   Column             Non-Null Count   Dtype
      ---  ------             --------------   -----
       0   Unnamed: 0         200000 non-null  int64
       1   key                200000 non-null  object
       2   fare_amount        200000 non-null  float64
       3   pickup_datetime    200000 non-null  object
       4   pickup_longitude   200000 non-null  float64
       5   pickup_latitude    200000 non-null  float64
       6   dropoff_longitude  199999 non-null  float64
       7   dropoff_latitude   199999 non-null  float64
       8   passenger_count    200000 non-null  int64
```



3	25894730	22:21.0	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	3
4	17610152	47:00.0	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	5

```
[16]: df.dtypes
      df.describe()
```

[16]:

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	2.000000e+05	200000.000000	200000.000000	200000.000000	199999.000000	199999.000000	200000.000000
mean	2.771250e+07	11.359955	-72.527638	39.935885	-72.525292	39.923890	1.684535
std	1.601382e+07	9.901776	11.437787	7.720539	13.117408	6.794829	1.385997
min	1.000000e+00	-52.000000	-1340.648410	-74.015515	-3356.666300	-881.985513	0.000000
25%	1.382535e+07	6.000000	-73.992065	40.734796	-73.991407	40.733823	1.000000
50%	2.774550e+07	8.500000	-73.981823	40.752592	-73.980093	40.753042	1.000000
75%	4.155530e+07	12.500000	-73.967153	40.767158	-73.963659	40.768001	2.000000
max	5.542357e+07	499.000000	57.418457	1644.421482	1153.572603	872.697628	208.000000

2. Data Cleaning Steps

Missing values were identified and removed to ensure dataset quality. Outliers like negative fare amounts or zero-distance trips were filtered out to improve the reliability of the analysis.

[18]:
```
df.isnull().sum()
df_clean = df.dropna()
```

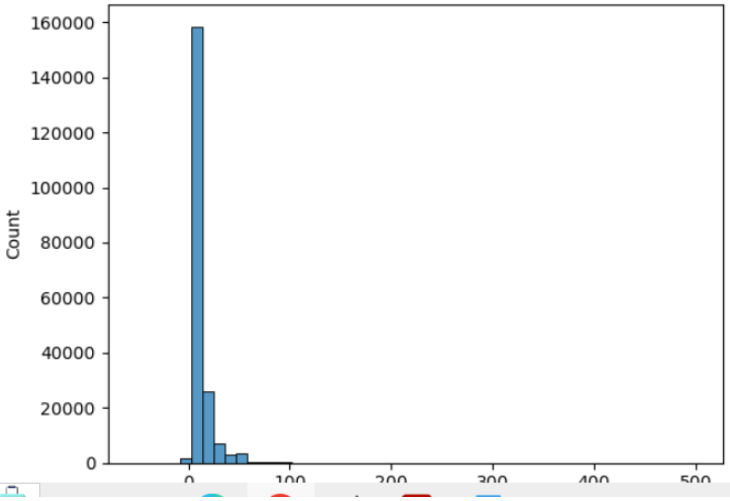
[20]:
```
df_clean.to_csv("uber_cleaned.csv", index=False)
```

[22]:
```
df_clean.describe()
```

[22]:

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	1.999990e+05	199999.000000	199999.000000	199999.000000	199999.000000	199999.000000	199999.000000
mean	2.771248e+07	11.359892	-72.527631	39.935881	-72.525292	39.923890	1.684543
std	1.601386e+07	9.901760	11.437815	7.720558	13.117408	6.794829	1.385995
min	1.000000e+00	-52.000000	-1340.648410	-74.015515	-3356.666300	-881.985513	0.000000
25%	1.382534e+07	6.000000	-73.992065	40.734796	-73.991407	40.733823	1.000000
50%	2.774524e+07	8.500000	-73.981823	40.752592	-73.980093	40.753042	1.000000
75%	4.155535e+07	12.500000	-73.967154	40.767158	-73.963659	40.768001	2.000000
max	5.542357e+07	499.000000	57.418457	1644.421482	1153.572603	872.697628	208.000000

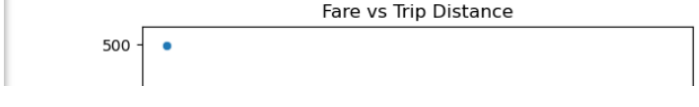
[28]:
```
import matplotlib.pyplot as plt

sns.histplot(df_clean['fare_amount'], bins=50)
plt.show()
```



[40]:
```
from math import radians, cos, sin, asin, sqrt

def haversine(lon1, lat1, lon2, lat2):
    # Convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])

    # Haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    r = 6371  # Radius of earth in kilometers
    return c * r
```

[42]:
```
df_clean.loc[:, 'trip_distance'] = df_clean.apply(lambda row:
    haversine(row['pickup_longitude'], row['pickup_latitude'],
              row['dropoff_longitude'], row['dropoff_latitude']), axis=1)
```

[44]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.scatterplot(data=df_clean, x="trip_distance", y="fare_amount")
plt.xlabel("Trip Distance (km)")
plt.ylabel("Fare Amount ($)")
plt.title("Fare vs Trip Distance")
plt.show()
```



```
import pandas as pd

# Try converting datetime safely
df_clean['pickup_datetime'] = pd.to_datetime(df_clean['pickup_datetime'], errors='coerce')

# Drop rows where datetime couldn't be parsed
df_clean = df_clean.dropna(subset=['pickup_datetime'])
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_11788\1949210122.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_clean['pickup_datetime'] = pd.to_datetime(df_clean['pickup_datetime'], errors='coerce')
```

```
# SAFEST WAY
df_clean = df.dropna().copy()

# Then safely convert datetime
df_clean['pickup_datetime'] = pd.to_datetime(df_clean['pickup_datetime'], errors='coerce')
```

```
# Extracting useful datetime parts
df_clean['hour'] = df_clean['pickup_datetime'].dt.hour
df_clean['day'] = df_clean['pickup_datetime'].dt.day
df_clean['month'] = df_clean['pickup_datetime'].dt.month
df_clean['day_of_week'] = df_clean['pickup_datetime'].dt.day_name()
```

```
def classify_peak(hour):
    if 7 <= hour <= 9 or 16 <= hour <= 19:
        return 'Peak'
    else:
```

[62]:
```
# Extracting useful datetime parts
df_clean['hour'] = df_clean['pickup_datetime'].dt.hour
df_clean['day'] = df_clean['pickup_datetime'].dt.day
df_clean['month'] = df_clean['pickup_datetime'].dt.month
df_clean['day_of_week'] = df_clean['pickup_datetime'].dt.day_name()
```

[64]:
```
def classify_peak(hour):
    if 7 <= hour <= 9 or 16 <= hour <= 19:
        return 'Peak'
    else:
        return 'Off-Peak'

df_clean['peak_period'] = df_clean['hour'].apply(classify_peak)
```

[66]:
```
def fare_bucket(fare):
    if fare < 5:
        return 'Very Low'
    elif fare < 15:
        return 'Low'
    elif fare < 30:
        return 'Medium'
    elif fare < 50:
        return 'High'
    else:
        return 'Very High'

df_clean['fare_bucket'] = df_clean['fare_amount'].apply(fare_bucket)
```

[70]:
```
df_clean.to_csv("uber_enhanced.csv", index=False)
```

[ ]:

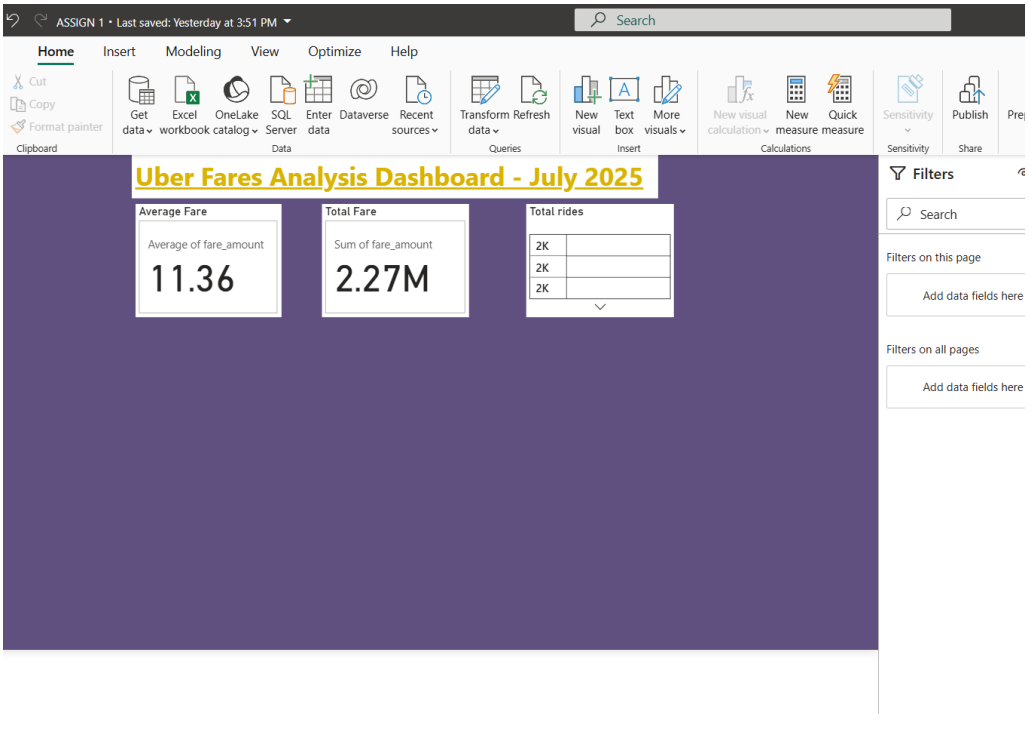
3. **DAX Formulas (if used)**

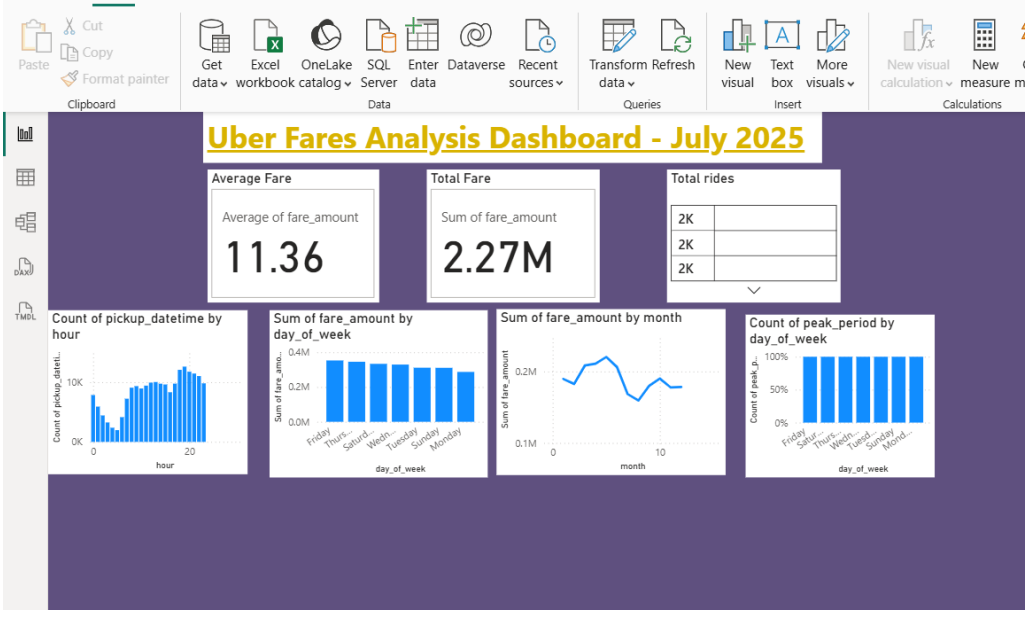
DAX Formulas:

No custom DAX measures were used — all visualizations relied on default field aggregations (count, average, etc.) in Power BI.

Dashboard Development Stages

The Power BI dashboard was built in progressive stages. The design began with summary KPIs at the top, followed by time-based visualizations in the middle and map/scatter visuals at the bottom. Slicers were added for interactivity, and grid alignment tools ensured a clean, professional layout.





Final Dashboard view