Import packages

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import sqlite3
import datetime
import geopandas as gpd
from shapely.geometry import Point, Polygon
import dask.dataframe as dd
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

EDA

Import data for one month of NYC Taxi and Limousine Commison (TLC)

January 2025 TLC file

```
eda_ddf = dd.read_parquet(['/content/drive/MyDrive/PHASE_5_PROJECT/U2025.parquet'])
eda_df = eda_ddf.compute()
eda_df
```

	hvfhs_license_num	dispatching_base_num	originating_base_num	request_datetime	on_scene_datetime	pickup_datetime
0	HV0003	B03404	B03404	2025-01-01 00:28:07	2025-01-01 00:31:17	2025-01-01 00:33:25
1	HV0005	B03406	<NA>	2025-01-01 00:18:33	NaT	2025-01-01 00:29:49
2	HV0003	B03404	B03404	2025-01-01 00:28:22	2025-01-01 00:31:52	2025-01-01 00:32:39
3	HV0003	B03404	B03404	2025-01-01 00:27:13	2025-01-01 00:33:58	2025-01-01 00:34:55
4	HV0003	B03404	B03404	2025-01-01 00:33:29	2025-01-01 00:45:46	2025-01-01 00:46:19
...	...	...	...	...	...	...
3628445	HV0003	B03404	B03404	2025-01-31 23:45:39	2025-01-31 23:51:13	2025-01-31 23:51:52
3628446	HV0003	B03404	B03404	2025-01-31 23:06:19	2025-01-31 23:08:27	2025-01-31 23:10:28
3628447	HV0003	B03404	B03404	2025-01-31 23:25:48	2025-01-31 23:30:47	2025-01-31 23:31:24
3628448	HV0003	B03404	B03404	2025-01-31 23:48:59	2025-01-31 23:55:45	2025-01-31 23:57:06
3628449	HV0003	B03404	B03404	2025-01-31 23:16:25	2025-01-31 23:22:00	2025-01-31 23:22:03

20405666 rows × 24 columns

Reduce the row count from 20 million to 1 million

```
mini_eda_df= eda_df.sample(frac=.05, random_state=42)
mini_eda_df
```

	hvfhs_license_num	dispatching_base_num	originating_base_num	request_datetime	on_scene_datetime	pickup_datetime
2707637	HV0003	B03404	B03404	2025-01-05 11:33:26	2025-01-05 11:33:39	2025-01-05 11:34:58
1762921	HV0003	B03404	B03404	2025-01-16 22:52:08	2025-01-16 22:55:45	2025-01-16 22:55:59
3943151	HV0003	B03404	B03404	2025-01-19 23:09:59	2025-01-19 23:27:05	2025-01-19 23:28:09
6097672	HV0005	B03406	<NA>	2025-01-23 10:50:08	NaT	2025-01-23 10:53:50
3332466	HV0005	B03406	<NA>	2025-01-19 01:35:56	NaT	2025-01-19 01:44:22
...	...	...	...	...	...	...
5036411	HV0003	B03404	B03404	2025-01-09 08:00:19	2025-01-09 08:05:19	2025-01-09 08:05:37
4331130	HV0005	B03406	<NA>	2025-01-08 07:14:40	NaT	2025-01-08 07:16:25
1816595	HV0003	B03404	B03404	2025-01-04 00:45:25	2025-01-04 00:49:12	2025-01-04 00:51:14
3230322	HV0003	B03404	B03404	2025-01-18 22:41:55	2025-01-18 22:44:22	2025-01-18 22:44:42
4686214	HV0003	B03404	B03404	2025-01-21 09:17:40	2025-01-21 09:19:33	2025-01-21 09:20:46

1020283 rows × 24 columns

Import table containg taxi zones corresponding to LocationID and Borough

```
zone_lookup = pd.read_csv('/content/drive/MyDrive/PHASE_5_PROJECT/taxi_zone_lookup.csv')
zone_lookup
```

	LocationID	Borough	Zone	service_zone
0	1	EWR	Newark Airport	EWR
1	2	Queens	Jamaica Bay	Boro Zone
2	3	Bronx	Allerton/Pelham Gardens	Boro Zone
3	4	Manhattan	Alphabet City	Yellow Zone
4	5	Staten Island	Arden Heights	Boro Zone
...	...	...	...	...
260	261	Manhattan	World Trade Center	Yellow Zone
261	262	Manhattan	Yorkville East	Yellow Zone
262	263	Manhattan	Yorkville West	Yellow Zone
263	264	Unknown	NaN	NaN
264	265	NaN	Outside of NYC	NaN

265 rows × 4 columns

Next steps: Generate code with zone_lookup | View recommended plots | New interactive sheet

Merge TLC table with taxi zone table to include borough

```
pickup = zone_lookup[['Borough', 'LocationID']].copy()
pickup.columns= ['pickup_borough','PULocationID',]
dropoff = zone_lookup[['Borough', 'LocationID']].copy()
dropoff.columns =['dropoff_borough', 'DOLocationID',]
df_merged = pd.merge(
            mini_eda_df,
            pickup,
            on = 'PULocationID',
            how = 'left'
```

```
)

df_merged = pd.merge(
            df_merged,
            dropoff,
            on = 'DOLocationID',
            how = 'left'
)
df_merged
```

	hvfhs_license_num	dispatching_base_num	originating_base_num	request_datetime	on_scene_datetime	pickup_datetime
0	HV0003	B03404	B03404	2025-01-05 11:33:26	2025-01-05 11:33:39	2025-01-05 11:34:58
1	HV0003	B03404	B03404	2025-01-16 22:52:08	2025-01-16 22:55:45	2025-01-16 22:55:59
2	HV0003	B03404	B03404	2025-01-19 23:09:59	2025-01-19 23:27:05	2025-01-19 23:28:09
3	HV0005	B03406	<NA>	2025-01-23 10:50:08	NaT	2025-01-23 10:53:50
4	HV0005	B03406	<NA>	2025-01-19 01:35:56	NaT	2025-01-19 01:44:22
...	...	...	...	...	...	...
1020278	HV0003	B03404	B03404	2025-01-09 08:00:19	2025-01-09 08:05:19	2025-01-09 08:05:37
1020279	HV0005	B03406	<NA>	2025-01-08 07:14:40	NaT	2025-01-08 07:16:25
1020280	HV0003	B03404	B03404	2025-01-04 00:45:25	2025-01-04 00:49:12	2025-01-04 00:51:14
1020281	HV0003	B03404	B03404	2025-01-18 22:41:55	2025-01-18 22:44:22	2025-01-18 22:44:42
1020282	HV0003	B03404	B03404	2025-01-21 09:17:40	2025-01-21 09:19:33	2025-01-21 09:20:46

1020283 rows × 26 columns

Filter for only Manhattan pickup and dropoff locations

```
manhattan_df = df_merged[(df_merged['pickup_borough'] == 'Manhattan') & (df_merged['dropoff_borough'] == 'Manhattan')].drop(colu
manhattan_df
```

n_scene_datetime	pickup_datetime
2025-01-05 11:33:39	2025-01-05 11:34:58
2025-01-23 16:31:13	2025-01-23 16:31:43
2025-01-11 05:27:49	2025-01-11 05:29:50
2025-01-03 17:56:01	2025-01-03 17:58:02
NaT	2025-01-08 14:31:28
...	...
NaT	2025-01-09 11:58:25
2025-01-18 23:15:36	2025-01-18 23:15:44
2025-01-08 07:42:05	2025-01-08 07:42:25
NaT	2025-01-08 07:16:25
2025-01-18 22:44:22	2025-01-18 22:44:42

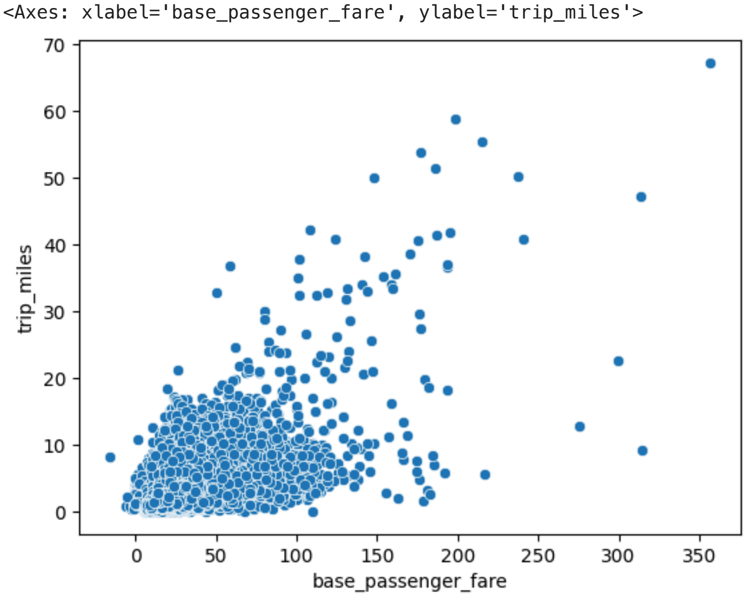
Remove base fares equal to zero

```
manhattan_df_2 = manhattan_df[manhattan_df['base_passenger_fare'] != 0]
```

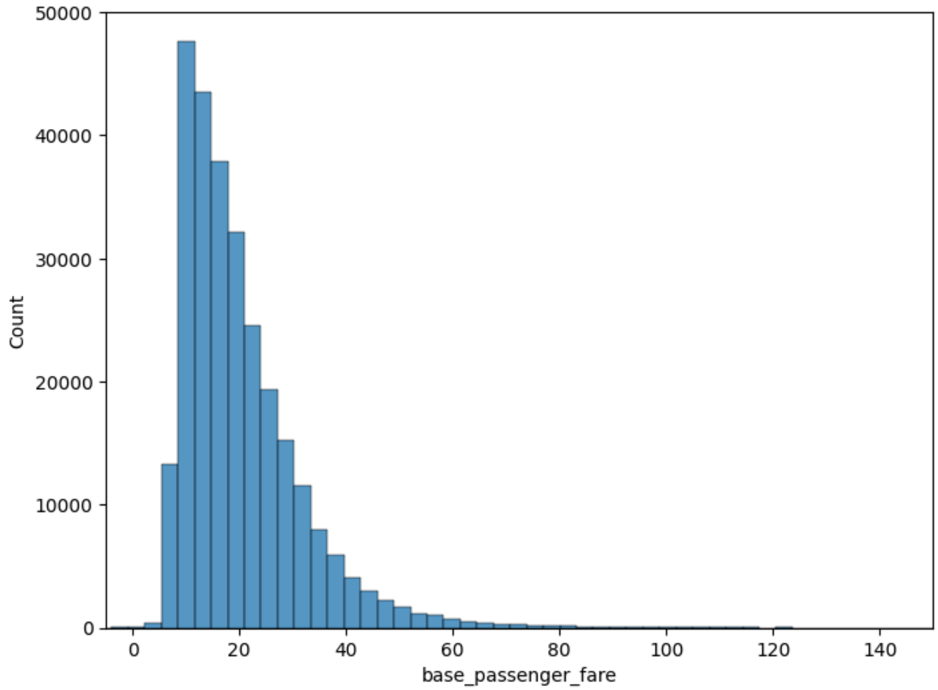
```
pricing_df_all =manhattan_df_2['base_passenger_fare'].sort_values(ascending=False)
```

```
sns.scatterplot(data=manhattan_df_2, x = 'base_passenger_fare', y= 'trip_miles')
```

```
<Axes: xlabel='base_passenger_fare', ylabel='trip_miles'>
```



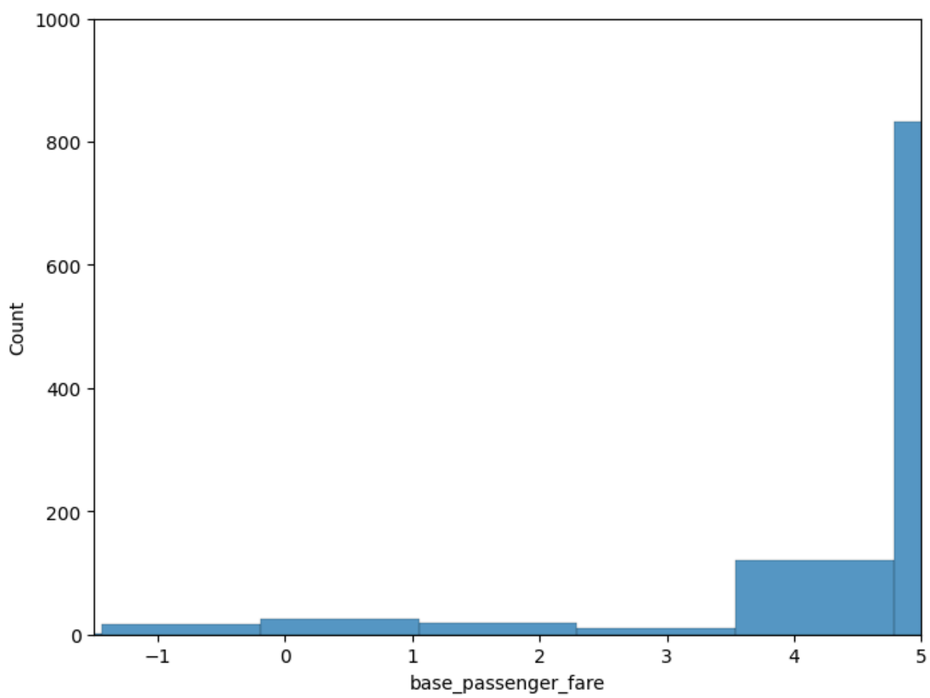

```
plt.figure(figsize=(8,6))
sns.histplot(pricing_df_all, bins=120)

plt.xlim(-5,150)
plt.show()
```




```
#Drop all rows with passenger fares below 5
```

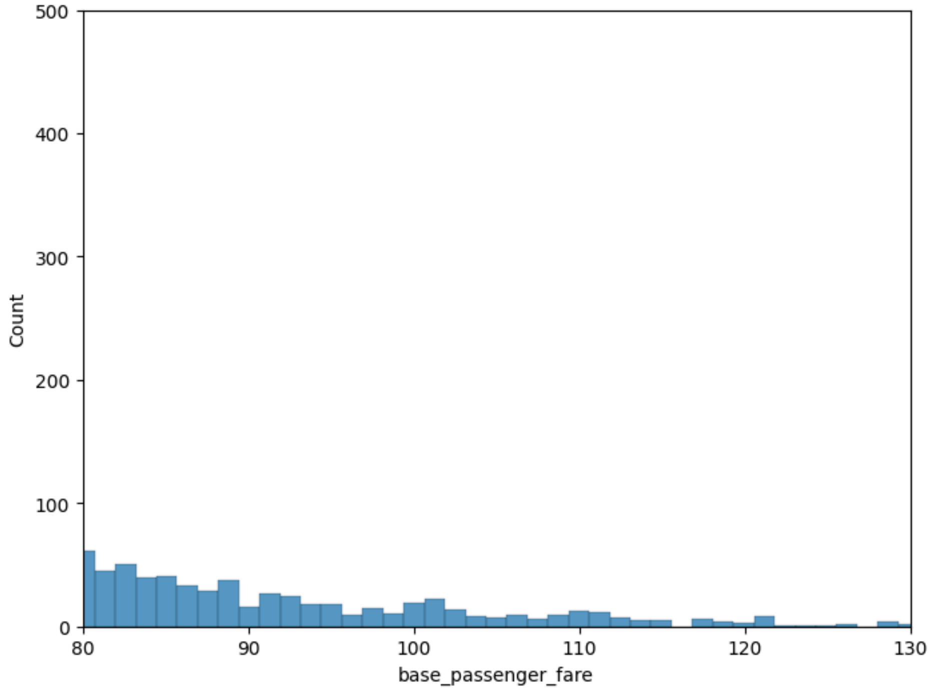
Start coding or generate with AI.

```
plt.figure(figsize=(8,6))
sns.histplot(pricing_df_all, bins=300)

plt.xlim(-1.5,5)
plt.ylim(0,1000)
plt.show()
```




```
plt.figure(figsize=(8,6))
sns.histplot(pricing_df_all, bins=300)

plt.xlim(80,130)
```

```
plt.ylim(0,500)
plt.show()
```



Drop rows contianing base fares under 5 and over 120

```
manhattan_df_3 = manhattan_df_2[(manhattan_df_2['base_passenger_fare'] > 5) & (manhattan_df_2['base_passenger_fare'] < 120)]
```

```
manhattan_df_3['base_passenger_fare'].describe()
```

	base_passenger_fare
count	275339.000000
mean	20.248496
std	11.241745
min	5.010000
25%	12.180000
50%	17.380000
75%	25.230000
max	119.680000

dtype: float64

Double-click (or enter) to edit

Remove rows that where the bae fare is over 100 and the pickup and dropoff location is the same

```
manhattan_df_4 = manhattan_df_3[(manhattan_df_3['base_passenger_fare'] <100) & (manhattan_df_3['PULocationID'] != manhattan_df_3
```

```
manhattan_df_4.columns
```

```
Index(['hvfhs_license_num', 'dispatching_base_num', 'originating_base_num',
       'request_datetime', 'on_scene_datetime', 'pickup_datetime',
       'dropoff_datetime', 'PULocationID', 'DOLocationID', 'trip_miles',
       'trip_time', 'base_passenger_fare', 'tolls', 'bcf', 'sales_tax',
       'congestion_surcharge', 'airport_fee', 'tips', 'driver_pay',
       'shared_request_flag', 'shared_match_flag', 'access_a_ride_flag',
       'wav_request_flag', 'wav_match_flag'],
      dtype='object')
```

```
manhattan_df_4.isna().sum()
```

	0
hvfhs_license_num	0
dispatching_base_num	0
originating_base_num	72518
request_datetime	0
on_scene_datetime	72518
pickup_datetime	0
dropoff_datetime	0
PULocationID	0
DOLocationID	0
trip_miles	0
trip_time	0
base_passenger_fare	0
tolls	0
bcf	0
sales_tax	0
congestion_surcharge	0
airport_fee	0
tips	0
driver_pay	0
shared_request_flag	0
shared_match_flag	0
access_a_ride_flag	0
wav_request_flag	0
wav_match_flag	0

dtype: int64

```
manhattan_df_4.dropna()
```

	hvfhs_license_num	dispatching_base_num	originating_base_num	request_datetime	on_scene_datetime	pickup_datetime
0	HV0003	B03404	B03404	2025-01-05 11:33:26	2025-01-05 11:33:39	2025-01-05 11:34:58
12	HV0003	B03404	B03404	2025-01-23 16:25:22	2025-01-23 16:31:13	2025-01-23 16:31:43
13	HV0003	B03404	B03404	2025-01-11 05:25:57	2025-01-11 05:27:49	2025-01-11 05:29:50
21	HV0003	B03404	B03404	2025-01-03 17:55:28	2025-01-03 17:56:01	2025-01-03 17:58:02
43	HV0003	B03404	B03404	2025-01-28 20:29:16	2025-01-28 20:29:59	2025-01-28 20:30:21
...	...	...	...	...	...	...
1020267	HV0003	B03404	B03404	2025-01-15 03:00:03	2025-01-15 03:02:37	2025-01-15 03:02:52
1020268	HV0003	B03404	B03404	2025-01-17 14:03:49	2025-01-17 14:09:38	2025-01-17 14:09:57
1020272	HV0003	B03404	B03404	2025-01-18 23:12:27	2025-01-18 23:15:36	2025-01-18 23:15:44
1020277	HV0003	B03404	B03404	2025-01-08 07:38:12	2025-01-08 07:42:05	2025-01-08 07:42:25
1020281	HV0003	B03404	B03404	2025-01-18 22:41:55	2025-01-18 22:44:22	2025-01-18 22:44:42

191199 rows × 24 columns

Filter for 'HV0003' in hvfhs_license_num column as this is the desingated number for Ubers

```
manhattan_df_4['hvfhs_license_num'].value_counts()
```

	count
hvfhs_license_num	
HV0003	191029
HV0005	72688

dtype: int64[pyarrow]

```
temp_df_1 = manhattan_df_4.loc[manhattan_df_4['hvfhs_license_num']=='HV0003']
```

```
#Filter out shared rides (Both those where the request was fufilled as the price is the same for both)
```

Filter out shared rides (Both those where the request was requested and fufilled as the price is the same for both)

```
ddf_temp_2 = temp_df_1[(temp_df_1['shared_match_flag'] == 'N')& (temp_df_1['shared_request_flag'] == 'N')]
ddf_temp_2.head()
```

	hvfhs_license_num	dispatching_base_num	originating_base_num	request_datetime	on_scene_datetime	pickup_datetime	dropo
0	HV0003	B03404	B03404	2025-01-05 11:33:26	2025-01-05 11:33:39	2025-01-05 11:34:58	2025-
12	HV0003	B03404	B03404	2025-01-23 16:25:22	2025-01-23 16:31:13	2025-01-23 16:31:43	2025-
13	HV0003	B03404	B03404	2025-01-11 05:25:57	2025-01-11 05:27:49	2025-01-11 05:29:50	2025-
21	HV0003	B03404	B03404	2025-01-03 17:55:28	2025-01-03 17:56:01	2025-01-03 17:58:02	2025-
43	HV0003	B03404	B03404	2025-01-28 20:29:16	2025-01-28 20:29:59	2025-01-28 20:30:21	2025-

5 rows × 24 columns

```
ddf_temp_3 = ddf_temp_2.drop(columns=['shared_match_flag','shared_request_flag'])
ddf_temp_3.head()
```

	hvfhs_license_num	dispatching_base_num	originating_base_num	request_datetime	on_scene_datetime	pickup_datetime	dropo
0	HV0003	B03404	B03404	2025-01-05 11:33:26	2025-01-05 11:33:39	2025-01-05 11:34:58	2025-
12	HV0003	B03404	B03404	2025-01-23 16:25:22	2025-01-23 16:31:13	2025-01-23 16:31:43	2025-
13	HV0003	B03404	B03404	2025-01-11 05:25:57	2025-01-11 05:27:49	2025-01-11 05:29:50	2025-
21	HV0003	B03404	B03404	2025-01-03 17:55:28	2025-01-03 17:56:01	2025-01-03 17:58:02	2025-
43	HV0003	B03404	B03404	2025-01-28 20:29:16	2025-01-28 20:29:59	2025-01-28 20:30:21	2025-

5 rows × 22 columns

```
Start coding or generate with AI.
```

Analyzing the access_a_ride_flag column which is based off of people who have thier rides subsidized by the MTA

```
ddf_temp_3['access_a_ride_flag'].value_counts()
```

	count
access_a_ride_flag	
N	184827
Y	113

dtype: int64[pyarrow]

The subsidized fare is about $2.50 so this means that the base fare incldued here is the total price paid to Uber

```
ddf_temp_3[ddf_temp_3['access_a_ride_flag'] == 'Y']['base_passenger_fare'].median()
```

```
25.37
```

```
ddf_temp_3[ddf_temp_3['access_a_ride_flag'] == 'Y']['trip_miles'].median()
```

```
3.25
```

```
ddf_temp_3[ddf_temp_3['access_a_ride_flag'] == 'N']['base_passenger_fare'].median()
```

```
18.82
```

```
ddf_temp_3[ddf_temp_3['access_a_ride_flag'] == 'N']['trip_miles'].median()
```

```
1.91
```

```
ddf_temp_3[(ddf_temp_3['access_a_ride_flag'] == 'Y') & (ddf_temp_3['trip_miles']  < 2)& (ddf_temp_3['trip_miles']  > 1)]['base_p
```

	base_passenger_fare
count	22.000000
mean	15.406818
std	7.978496
min	7.970000
25%	10.795000
50%	12.470000
75%	15.685000
max	36.050000

dtype: float64

```
ddf_temp_3[(ddf_temp_3['access_a_ride_flag'] == 'N') & (ddf_temp_3['trip_miles']  < 2)& (ddf_temp_3['trip_miles']  > 1)]['base_p
```

	base_passenger_fare
count	70120.000000
mean	16.816903
std	7.417686
min	5.010000
25%	11.900000
50%	14.940000
75%	19.182500
max	95.670000

dtype: float64

```
ddf_temp_3[(ddf_temp_3['access_a_ride_flag'] == 'Y')]['PULocationID'].value_counts()[:10]
```

PULocationID	count
74	8
90	8
41	6
68	6
238	5
161	4
244	4
166	4
148	4
140	4

dtype: int64

```
ddf_temp_3[(ddf_temp_3['access_a_ride_flag'] == 'N')]['PULocationID'].value_counts()[:10]
```

	count
PULocationID	
161	6322
234	5915
230	5909
79	5873
231	5752
246	5670
68	5189
249	4868
237	4735
164	4681

dtype: int64

The rides that are subsidized by the MTA may be important to keep as they provide addtional data and have roughlt the same base fare

```
ddf_temp_3
```

	hvfhs_license_num	dispatching_base_num	originating_base_num	request_datetime	on_scene_datetime	pickup_datetime
0	HV0003	B03404	B03404	2025-01-05 11:33:26	2025-01-05 11:33:39	2025-01-05 11:34:58
12	HV0003	B03404	B03404	2025-01-23 16:25:22	2025-01-23 16:31:13	2025-01-23 16:31:43
13	HV0003	B03404	B03404	2025-01-11 05:25:57	2025-01-11 05:27:49	2025-01-11 05:29:50
21	HV0003	B03404	B03404	2025-01-03 17:55:28	2025-01-03 17:56:01	2025-01-03 17:58:02
43	HV0003	B03404	B03404	2025-01-28 20:29:16	2025-01-28 20:29:59	2025-01-28 20:30:21
...	...	...	...	...	...	...
1020267	HV0003	B03404	B03404	2025-01-15 03:00:03	2025-01-15 03:02:37	2025-01-15 03:02:52
1020268	HV0003	B03404	B03404	2025-01-17 14:03:49	2025-01-17 14:09:38	2025-01-17 14:09:57
1020272	HV0003	B03404	B03404	2025-01-18 23:12:27	2025-01-18 23:15:36	2025-01-18 23:15:44
1020277	HV0003	B03404	B03404	2025-01-08 07:38:12	2025-01-08 07:42:05	2025-01-08 07:42:25
1020281	HV0003	B03404	B03404	2025-01-18 22:41:55	2025-01-18 22:44:22	2025-01-18 22:44:42

184940 rows × 22 columns

```
ddf_temp_3['wav_request_flag'].value_counts()
```

	count
wav_request_flag	
N	184451
Y	489

dtype: int64[pyarrow]

```
ddf_temp_3['wav_match_flag'].value_counts()
```

	count
wav_match_flag	
N	162608
Y	22332

dtype: int64[pyarrow]

Keep both wav_request_flag features

```
ddf_temp_3[ddf_temp_3['wav_request_flag'] == 'Y']['base_passenger_fare'].median()
```

```
18.01
```

```
ddf_temp_3[ddf_temp_3['wav_request_flag'] == 'N']['base_passenger_fare'].median()
```

```
18.82
```

```
ddf_temp_3[ddf_temp_3['wav_match_flag'] == 'Y']['base_passenger_fare'].median()
```

```
17.36
```

```
ddf_temp_3[ddf_temp_3['wav_match_flag'] == 'N']['base_passenger_fare'].median()
```

```
19.01
```

Prepare data for model

Upload a years worth of data of TLC paraquet files between Febuarary 2024 and January 2025

```
paths=['/content/drive/MyDrive/PHASE_5_PROJECT/Feb_24.parquet',
                         '/content/drive/MyDrive/PHASE_5_PROJECT/Mar_24.parquet',
                         '/content/drive/MyDrive/PHASE_5_PROJECT/Apr_24.parquet',
                         '/content/drive/MyDrive/PHASE_5_PROJECT/May_24.parquet',
                         '/content/drive/MyDrive/PHASE_5_PROJECT/Jun_24.parquet',
                         '/content/drive/MyDrive/PHASE_5_PROJECT/Jul_24.parquet',
                         '/content/drive/MyDrive/PHASE_5_PROJECT/Aug_24.parquet',
                         '/content/drive/MyDrive/PHASE_5_PROJECT/Sep_24.parquet',
                         '/content/drive/MyDrive/PHASE_5_PROJECT/Oct_24.parquet',
                         '/content/drive/MyDrive/PHASE_5_PROJECT/Nov_24.parquet',
                         '/content/drive/MyDrive/PHASE_5_PROJECT/Dec_24.parquet',
                         '/content/drive/MyDrive/PHASE_5_PROJECT/U2025.parquet']
ddf_24 = dd.read_parquet(paths)


ddf_24['PULocationID'] = ddf_24['PULocationID'].astype('int64')
ddf_24['DOLocationID'] = ddf_24['DOLocationID'].astype('int64')
zone_lookup['LocationID'] = zone_lookup['LocationID'].astype('int64')
pickup = zone_lookup[['Borough', 'LocationID', 'Zone']].copy()
pickup.columns= ['pickup_borough','PULocationID','pickup_zone']
dropoff = zone_lookup[['Borough', 'LocationID', 'Zone']].copy()
dropoff.columns =['dropoff_borough', 'DOLocationID','dropoff_zone']
merged_ddf = ddf_24.merge(
            pickup,
            on = 'PULocationID',
            how = 'left'
)

merged_ddf = merged_ddf.merge(
            dropoff,
            on = 'DOLocationID',
            how = 'left'
)
```

```
print(ddf_24['PULocationID'].dtype)
```

```
int64
```

```
print(pickup['PULocationID'].dtype)
```

```
int64
```

```
print(merged_ddf.columns)
```

```
Index(['hvfhs_license_num', 'dispatching_base_num', 'originating_base_num',
       'request_datetime', 'on_scene_datetime', 'pickup_datetime',
       'dropoff_datetime', 'PULocationID', 'DOLocationID', 'trip_miles',
       'trip_time', 'base_passenger_fare', 'tolls', 'bcf', 'sales_tax',
       'congestion_surcharge', 'airport_fee', 'tips', 'driver_pay',
       'shared_request_flag', 'shared_match_flag', 'access_a_ride_flag',
       'wav_request_flag', 'wav_match_flag', 'pickup_borough', 'pickup_zone',
       'dropoff_borough', 'dropoff_zone'],
      dtype='object')
```

Replicate cleaning done above

```
ddf_manhattan = merged_ddf[(merged_ddf['pickup_borough'] == 'Manhattan') & (merged_ddf['dropoff_borough'] == 'Manhattan')]
```

```
ddf_manhattan_2 = ddf_manhattan.drop(columns=['pickup_borough', 'dropoff_borough'], axis=1)
```

```
ddf_manhattan_3 = ddf_manhattan_2[ddf_manhattan_2['base_passenger_fare'] != 0]
```

```
ddf_manhattan_4 = ddf_manhattan_3[(ddf_manhattan_3['base_passenger_fare'] > 5) & (ddf_manhattan_3['base_passenger_fare'] < 120)]
```

```
ddf_manhattan_5 = ddf_manhattan_4[(ddf_manhattan_4['base_passenger_fare'] <100) & (ddf_manhattan_4['PULocationID'] != ddf_manhat
```

```
ddf_2024 = ddf_manhattan_5.loc[ddf_manhattan_5['hvfhs_license_num']=='HV0003']
```

```
ddf_temp = ddf_2024.loc[ddf_2024['shared_match_flag'] == 'N']
```

```
ddf_temp_2= ddf_temp.loc[ddf_temp['shared_request_flag'] == 'N']
```

```
ddf_temp_3 = ddf_temp_2.loc[(ddf_temp_2['PULocationID'] < 265) & (ddf_temp_2['DOLocationID'] < 265)]
```

Roughly 180 million rows, need to sample

```
ddf_sam = ddf_temp_3.sample(frac=.03, random_state=42)
```

```
df_2024 = ddf_sam.compute()
```

```
df_2024.columns
```

```
Index(['hvfhs_license_num', 'dispatching_base_num', 'originating_base_num',
       'request_datetime', 'on_scene_datetime', 'pickup_datetime',
       'dropoff_datetime', 'PULocationID', 'DOLocationID', 'trip_miles',
       'trip_time', 'base_passenger_fare', 'tolls', 'bcf', 'sales_tax',
       'congestion_surcharge', 'airport_fee', 'tips', 'driver_pay',
       'shared_request_flag', 'shared_match_flag', 'access_a_ride_flag',
       'wav_request_flag', 'wav_match_flag', 'pickup_zone', 'dropoff_zone'],
      dtype='object')
```

```
df_24_Cl = df_2024.drop(columns=['hvfhs_license_num','dispatching_base_num','originating_base_num','shared_request_flag','sharec
```

```
df_24_Cl
```

	request_datetime	on_scene_datetime	pickup_datetime	dropoff_datetime	PULocationID	DOLocationID	trip_miles	trip_
1118505	2024-04-05 08:41:35	2024-04-05 08:43:58	2024-04-05 08:45:46	2024-04-05 08:59:16	231	113	1.73	
1075156	2024-04-05 01:15:17	2024-04-05 01:16:22	2024-04-05 01:18:22	2024-04-05 01:25:26	162	236	1.46	
2506900	2024-04-10 17:56:28	2024-04-10 18:00:09	2024-04-10 18:00:50	2024-04-10 18:06:07	41	24	0.44	
2707255	2024-04-11 15:22:38	2024-04-11 15:25:49	2024-04-11 15:26:47	2024-04-11 15:33:54	41	42	0.99	
2380842	2024-04-10 08:25:40	2024-04-10 08:26:15	2024-04-10 08:27:29	2024-04-10 08:50:41	229	151	4.00	
...	...	...	...	...	...	...	...	
1282792	2025-01-31 16:56:48	2025-01-31 17:01:23	2025-01-31 17:01:56	2025-01-31 17:11:43	244	243	1.16	
1252512	2025-01-31 15:19:45	2025-01-31 15:23:22	2025-01-31 15:24:04	2025-01-31 15:28:25	166	116	1.36	
936316	2025-01-30 11:59:34	2025-01-30 12:01:43	2025-01-30 12:03:44	2025-01-30 12:23:35	239	237	2.00	
191627	2025-01-27 08:37:27	2025-01-27 08:39:36	2025-01-27 08:41:36	2025-01-27 09:17:17	74	68	6.56	
1203706	2025-01-31 11:00:54	2025-01-31 11:01:52	2025-01-31 11:03:16	2025-01-31 11:37:45	161	232	4.66	

1332108 rows × 16 columns

```
df_24_Cl['base_passenger_fare'].sort_values()[20:50]
```

	base_passenger_fare
2509388	5.03
2998355	5.04
2117185	5.04
270190	5.04
2043179	5.04
3085981	5.04
101271	5.04
1361766	5.04
2012266	5.04
323850	5.04
236765	5.04
343153	5.05
2639546	5.05
555445	5.05
2860491	5.05
607958	5.05
1590127	5.05
2641328	5.05
1572749	5.06
584355	5.06
942825	5.06
1559750	5.06
2773617	5.06
3249555	5.06
2540391	5.06
2260625	5.06
1555799	5.06
1151108	5.07
636489	5.07
1422828	5.07

dtype: float64

```
df_24_Cl['tolls'].sort_values(ascending=False)[400:425]
```

	tolls
1159443	16.94
197379	16.94
723683	16.94
2192496	16.94
1745258	16.94
1593360	16.94
587499	16.94
671009	16.94
139157	16.94
2525307	16.94
2606526	16.94
1709533	16.94
2146765	16.94
338561	16.94
3078031	16.94
990065	16.94
1502693	16.94
852162	16.94
1378024	16.94
3019469	16.94
2220256	16.94
1957427	16.94
23738	16.94
144937	16.94
1874638	16.94

dtype: float64

```
df_24_Cl.iloc[0]
```

	1118505
request_datetime	2024-04-05 08:41:35
on_scene_datetime	2024-04-05 08:43:58
pickup_datetime	2024-04-05 08:45:46
dropoff_datetime	2024-04-05 08:59:16
PULocationID	231
DOLocationID	113
trip_miles	1.73
trip_time	810
base_passenger_fare	19.35
tolls	0.0
bcf	0.57
sales_tax	1.83
congestion_surcharge	2.75
airport_fee	0.0
pickup_zone	TriBeCa/Civic Center
dropoff_zone	Greenwich Village North

dtype: object

```
# Removing rows for trips that started or ended outside of NYC

#df_24_Cl = df_24_Upd.loc[(df_24_Upd['PULocationID'] < 265) & (df_24_Upd['DOLocationID'] < 265)]

df_24_Cl['sales_tax'].isna().sum()
```

```
np.int64(0)
```

```
df_24_Cl['total_fare'] = (df_24_Cl['base_passenger_fare']) + (df_24_Cl['tolls']) + (df_24_Cl['bcf']) + (df_24_Cl['sales_tax']) +

#Had to take out Datetime becasue keras couldnt use. converted to day of week

df_24_Cl
```

	request_datetime	on_scene_datetime	pickup_datetime	dropoff_datetime	PULocationID	DOLocationID	trip_miles	trip_
1118505	2024-04-05 08:41:35	2024-04-05 08:43:58	2024-04-05 08:45:46	2024-04-05 08:59:16	231	113	1.73	
1075156	2024-04-05 01:15:17	2024-04-05 01:16:22	2024-04-05 01:18:22	2024-04-05 01:25:26	162	236	1.46	
2506900	2024-04-10 17:56:28	2024-04-10 18:00:09	2024-04-10 18:00:50	2024-04-10 18:06:07	41	24	0.44	
2707255	2024-04-11 15:22:38	2024-04-11 15:25:49	2024-04-11 15:26:47	2024-04-11 15:33:54	41	42	0.99	
2380842	2024-04-10 08:25:40	2024-04-10 08:26:15	2024-04-10 08:27:29	2024-04-10 08:50:41	229	151	4.00	
...	...	...	...	...	...	...	...	
1282792	2025-01-31 16:56:48	2025-01-31 17:01:23	2025-01-31 17:01:56	2025-01-31 17:11:43	244	243	1.16	
1252512	2025-01-31 15:19:45	2025-01-31 15:23:22	2025-01-31 15:24:04	2025-01-31 15:28:25	166	116	1.36	
936316	2025-01-30 11:59:34	2025-01-30 12:01:43	2025-01-30 12:03:44	2025-01-30 12:23:35	239	237	2.00	
191627	2025-01-27 08:37:27	2025-01-27 08:39:36	2025-01-27 08:41:36	2025-01-27 09:17:17	74	68	6.56	
1203706	2025-01-31 11:00:54	2025-01-31 11:01:52	2025-01-31 11:03:16	2025-01-31 11:37:45	161	232	4.66	

1332108 rows × 17 columns

```
df_24_Cl['pickup_year'] = df_24_Cl['pickup_datetime'].dt.year.astype(int)
df_24_Cl['pickup_month'] = df_24_Cl['pickup_datetime'].dt.month.astype(int)
df_24_Cl['pickup_day'] = df_24_Cl['pickup_datetime'].dt.day.astype(int)
df_24_Cl['pickup_hour'] = df_24_Cl['pickup_datetime'].dt.hour.astype(int)
df_24_Cl['pickup_minute'] = df_24_Cl['pickup_datetime'].dt.minute.astype(int)
df_24_Cl['pickup_second'] = df_24_Cl['pickup_datetime'].dt.second.astype(int)


df_24_Cl['request_year'] = df_24_Cl['request_datetime'].dt.year.astype(int)
df_24_Cl['request_month'] = df_24_Cl['request_datetime'].dt.month.astype(int)
df_24_Cl['request_day'] = df_24_Cl['request_datetime'].dt.day.astype(int)
df_24_Cl['request_hour'] = df_24_Cl['request_datetime'].dt.hour.astype(int)
df_24_Cl['request_minute'] = df_24_Cl['request_datetime'].dt.minute.astype(int)
df_24_Cl['request_second'] = df_24_Cl['request_datetime'].dt.second.astype(int)


df_24_Cl['on_scene_year'] = df_24_Cl['on_scene_datetime'].dt.year.astype(int)
df_24_Cl['on_scene_month'] = df_24_Cl['on_scene_datetime'].dt.month.astype(int)
df_24_Cl['on_scene_day'] = df_24_Cl['on_scene_datetime'].dt.day.astype(int)
df_24_Cl['on_scene_hour'] = df_24_Cl['on_scene_datetime'].dt.hour.astype(int)
df_24_Cl['on_scene_minute'] = df_24_Cl['on_scene_datetime'].dt.minute.astype(int)
df_24_Cl['on_scene_second'] = df_24_Cl['on_scene_datetime'].dt.second.astype(int)


df_24_Cl['dropoff_year'] = df_24_Cl['dropoff_datetime'].dt.year.astype(int)
df_24_Cl['dropoff_month'] = df_24_Cl['dropoff_datetime'].dt.month.astype(int)
df_24_Cl['dropoff_day'] = df_24_Cl['dropoff_datetime'].dt.day.astype(int)
df_24_Cl['dropoff_hour'] = df_24_Cl['dropoff_datetime'].dt.hour.astype(int)
df_24_Cl['dropoff_minute'] = df_24_Cl['dropoff_datetime'].dt.minute.astype(int)
df_24_Cl['dropoff_second'] = df_24_Cl['dropoff_datetime'].dt.second.astype(int)
```

```
df_base_model = df_24_Cl.drop(columns=['request_datetime',  'on_scene_datetime',    'pickup_datetime',  'dropoff_datetime','base
```

```
df_base_model_1 = df_base_model.drop(columns=['pickup_zone', 'dropoff_zone'])
```

```
df_base_model_1.head()
```

	PULocationID	DOLocationID	trip_miles	trip_time	total_fare	pickup_year	pickup_month	pickup_day	pickup_hour	pic
1118505	231	113	1.73	810	24.50	2024	4	5	8	
1075156	162	236	1.46	424	13.61	2024	4	5	1	
2506900	41	24	0.44	317	8.58	2024	4	10	18	
2707255	41	42	0.99	427	7.25	2024	4	11	15	
2380842	229	151	4.00	1392	25.00	2024	4	10	8	

5 rows × 29 columns

Building Baseline Model

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, root_mean_squared_error, mean_absolute_error, mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.optimizers import Adam
```

```
model = Sequential()
model.add(Dense(40, input_dim=28, activation='relu'))
model.add(Dense(1, activation ='linear'))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
opt = Adam(learning_rate = 0.01, beta_1 = 0.9, beta_2 = 0.999 )
model.compile(loss = 'mean_squared_error', optimizer ='adam', metrics = ['root_mean_squared_error','mean_absolute_error'])
```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 40)	1,160
dense_1 (Dense)	(None, 1)	41

Total params: 1,201 (4.69 KB)
Trainable params: 1,201 (4.69 KB)
Non-trainable params: 0 (0.00 B)

```
X = df_base_model_1.drop(columns=['total_fare'], axis=1)
y = df_base_model_1['total_fare']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=42)
```

```
from tensorflow.keras.callbacks import EarlyStopping
trainCallback = EarlyStopping(monitor='val_loss', min_delta = 1e-4, patience = 10)
```

```
history = model.fit(X_train, y_train, epochs=50, batch_size=512, validation_split=.2, callbacks=[trainCallback])
```

```
Epoch 1/50
1770/1770 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - loss: 83149.1562 - mean_absolute_error: 99.7872 - root_mean_squared_error: 231.
Epoch 2/50
1770/1770 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 101.9481 - mean_absolute_error: 6.9019 - root_mean_squared_error: 10.0969
Epoch 3/50
1770/1770 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - loss: 100.9341 - mean_absolute_error: 6.8496 - root_mean_squared_error: 10.0466
```