Taxi Trips Analysis Project

The purpose of this project is to demonstrate various techniques for exploring and analyzing data, including descriptive and correlation analysis, creating regression models, identifying key predictor variables, and manipulating existing variables through transformations or additional analysis.

Introduction:

Cab Taxi is a thriving transportation business in many parts of the world. It facilitates easy movement of people, goods, and services from one location to another. The aim of this data analysis project is to analyze the Yellow Medallion Taxi cabs dataset: the famous New York City (NYC) yellow taxis that provide transportation exclusively through street hails (i.e., the pickups are not prearranged). Passengers stand by the street and hail on an available taxi with their hand.

Data:

The dataset was sourced from the NYC Taxi & Limousine Commission (TLC) official website. The dataset contains several explanatory variables used to assess a completed trip such as pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts. A subsample of the original data is provided to use for the tasks itemized in the following sections. The trip records are divided into two files Main Sample and New Sample, respectively. In this analysis, I will answer some business questions and perform regression analysis to predict the total amount paid by the passengers after a given trip.

Description of Tasks:

Task A:
Knowing some statistics about the daily cab activities can help to improve the transportation business. Thus, analyzing the dataset provided in the Main Sample file I answered the following business questions:

1. What is the average demand for the taxis in the days of the week (i.e., daily trend). Which of the days has the highest and which lowest demand?
2. Which time of the day (morning, afternoon, evening, and night) is likely be a peak period for the taxi's operation from the data?
3. On average, how much revenue was generated in the weekdays and weekends for the business for the period covered in the dataset?

Task B:
Creating a regression model to predict the total amount paid for taxi ride, given the trip information in the dataset:

- Sequentially split the data in the Main Sample file into two sets, such that the first 80% of the records in the file is used for fitting the regression model. While the last 20% is used for testing the model and reporting the prediction errors (e.g., RMSE) and R2 scores respectively.
- Provide the equation for the finalized model in the report.

- Once the model is finalized, predict the total amount paid on a trip for the trip records shown in New Sample file and tabulate the predicated values in the report, in the order the records are arranged in the file.

The report will have the following Chapters:

1. **Introduction:** This includes the goals of the analysis and a brief description of the data. It will also include EDA and transformation I had to perform on the data.
2. **Data Analysis:** This section covers the discussion and answers to the question from Task A.
3. **Regression Analysis:** This section includes discussion about the activities in Task B including the modelling process.
4. **Discussion:** This includes relevant predictions and/or conclusions drawn from the model.
5. **Conclusion:** In this section, I summarized my analysis and highlighted final points from the analysis.
6. **Reference:** In this section, I provided the references to the source of data.

In [34]:
```
import numpy as np
import pandas as pd
import datetime as dt
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(style="whitegrid")
```

1: Introduction

EDA

In [2]:
```
# load the dataset
df = pd.read_parquet('taxi_dataset/main.parquet')
```

In [6]:
```
# show the first 5 rows of the dataframe
df.head()
```

Out[6]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitude	pick
0	2	2016-01-01 00:00:00	2016-01-01 00:00:00	2	1.10	-73.990372	
1	2	2016-01-01 00:00:00	2016-01-01 00:00:00	5	4.90	-73.980782	
2	2	2016-01-01 00:00:00	2016-01-01 00:00:00	1	10.54	-73.984550	
3	2	2016-01-01 00:00:00	2016-01-01 00:00:00	1	4.75	-73.993469	
4	2	2016-01-01 00:00:00	2016-01-01 00:00:00	3	1.76	-73.960625	

In [7]:
```
# show the shape of the dataframe - (rows, columns)
df.shape
```

Out[7]:
```
(10906858, 19)
```

In [8]:
```
# show the column names in the dataset
df.columns
```

Out[8]:
```
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'pickup_longitude',
       'pickup_latitude', 'RatecodeID', 'store_and_fwd_flag',
       'dropoff_longitude', 'dropoff_latitude', 'payment_type', 'fare_amount',
       'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
```

```
       'improvement_surcharge', 'total_amount'],
      dtype='object')
```

In [9]:
```
# show the data types of values in the columns
df.dtypes
```

Out[9]:
```
VendorID                   int64
tpep_pickup_datetime      object
tpep_dropoff_datetime     object
passenger_count            int64
trip_distance            float64
pickup_longitude         float64
pickup_latitude          float64
RatecodeID                 int64
store_and_fwd_flag        object
dropoff_longitude        float64
dropoff_latitude         float64
payment_type               int64
fare_amount              float64
extra                    float64
mta_tax                  float64
tip_amount               float64
tolls_amount             float64
improvement_surcharge    float64
total_amount             float64
dtype: object
```

In [10]:
```
# is there any missing values? - no
df.isna().sum()
```

Out[10]:
```
VendorID                 0
tpep_pickup_datetime     0
tpep_dropoff_datetime    0
passenger_count          0
trip_distance            0
pickup_longitude         0
pickup_latitude          0
RatecodeID               0
store_and_fwd_flag       0
dropoff_longitude        0
dropoff_latitude         0
payment_type             0
fare_amount              0
extra                    0
mta_tax                  0
tip_amount               0
tolls_amount             0
improvement_surcharge    0
total_amount             0
dtype: int64
```

In [11]:
```
# show some descriptive statistics of the numerical columns
df.describe().T
```

Out[11]:

	count	mean	std	min	25%	50%	75%	
VendorID	10906858.0	1.535024	0.498772	1.000000	1.000000	2.000000	2.000000	2.00
passenger_count	10906858.0	1.670847	1.324891	0.000000	1.000000	1.000000	2.000000	9.00
trip_distance	10906858.0	4.648197	2981.095329	0.000000	1.000000	1.670000	3.080000	8.00
pickup_longitude	10906858.0	-72.818695	9.168964	-121.934288	-73.991508	-73.981377	-73.966103	0.00
pickup_latitude	10906858.0	40.114943	5.051022	0.000000	40.736301	40.753689	40.768082	6.09
RatecodeID	10906858.0	1.039350	0.518631	1.000000	1.000000	1.000000	1.000000	9.90

dropoff_longitude	10906858.0	-72.886591	8.900841	-121.933487	-73.991074	-73.979424	-73.961960	0.00
dropoff_latitude	10906858.0	40.153152	4.903456	0.000000	40.734806	40.754131	40.769619	6.09
payment_type	10906858.0	1.347536	0.491080	1.000000	1.000000	1.000000	2.000000	5.00
fare_amount	10906858.0	12.486929	35.564004	-957.600000	6.500000	9.000000	14.000000	1.11
extra	10906858.0	0.313076	0.415679	-42.610000	0.000000	0.000000	0.500000	6.48
mta_tax	10906858.0	0.497670	0.050467	-0.500000	0.500000	0.500000	0.500000	8.97
tip_amount	10906858.0	1.750663	2.623546	-220.800000	0.000000	1.260000	2.320000	9.98
tolls_amount	10906858.0	0.293345	1.694572	-17.400000	0.000000	0.000000	0.000000	9.80
improvement_surcharge	10906858.0	0.299724	0.012326	-0.300000	0.300000	0.300000	0.300000	3.0
total_amount	10906858.0	15.641395	36.412802	-958.400000	8.300000	11.620000	17.160000	1.11

Some numerical values are less than 0, which doesn't make sense. Let's find them

In [12]:
```
df[(df['tip_amount'] < 0) | (df['fare_amount'] < 0) | (df['extra'] < 0) | (df['mta_tax']
```

Out[12]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitu
1774	2	2016-01-02 00:50:32	2016-01-02 00:51:16	1	0.26	-73.8256
3492	2	2016-01-02 01:00:59	2016-01-02 01:01:26	1	0.05	-73.9385
5105	2	2016-01-02 01:11:25	2016-01-02 01:14:30	1	0.53	-73.9558
5611	2	2016-01-02 01:14:29	2016-01-02 01:19:01	1	0.56	-73.9917
5666	2	2016-01-02 01:14:51	2016-01-02 01:16:23	2	0.04	-74.0061
...	...	...	...	...	...	
10895616	2	2016-01-29 08:57:35	2016-01-29 08:59:31	1	0.58	-73.9588
10898624	2	2016-01-29 09:06:34	2016-01-29 09:06:53	1	0.00	-73.9924
10905279	2	2016-01-29 09:30:29	2016-01-29 09:36:22	1	0.64	-73.9958
10906542	2	2016-01-22 23:02:07	2016-01-22 23:02:44	1	0.19	-73.9740
10906550	2	2016-01-23 01:30:18	2016-01-23 01:35:59	1	0.79	-73.9618

4225 rows × 19 columns

There're **4225** records with negative numerical values, which somehow should be fixed

---

Let's check if each of the numerical variable is within the dataset description limits

---

`Improvement_surcharge: 0.30 improvement surcharge assessed trips at the flag drop.`

So it should be either 0.30 or 0. Other values to be considered as mistake

In [13]:
```
df['improvement_surcharge'].value_counts()
```

Out[13]:
```
 0.30    10901039
-0.30        4202
 0.00        1609
 0.10           5
 0.12           1
 0.16           1
 0.25           1
Name: improvement_surcharge, dtype: int64
```

`MTA_tax: 0.50 MTA tax that is automatically triggered based on the metered rate in use.`

Should be either 0 or 0.5

In [14]:
```
df['mta_tax'].value_counts()
```

Out[14]:
```
 0.50     10859581
 0.00        43201
-0.50         4062
 0.35            2
 0.89            1
 2.22            1
 2.45            1
 20.50           1
 36.44           1
 10.35           1
 3.00            1
 33.49           1
 17.45           1
 89.70           1
 43.41           1
 0.93            1
Name: mta_tax, dtype: int64
```

`Extra: Miscellaneous extras and surcharges. Currently, this only includes the $0.50 and $1 rush hour and overnight charges.`

Should be either 0.5, 1 or 0

In [15]:
```
df['extra'].value_counts()
```

Out[15]:
```
 0.00      5710200
 0.50      3558725
 1.00      1635787
-0.50         1486
-1.00          513
 1.50           44
 0.02           25
 4.50           20
 2.00           12
 0.04           10
 0.20            6
 0.30            3
 2.50            3
 3.50            2
 0.70            2
 0.45            1
-32.69           1
-42.61           1
 0.80            1
```

```
 8.50            1
-16.65           1
 4.71            1
-0.45            1
 7.00            1
-4.50            1
 648.87          1
 4.10            1
-35.64           1
 5.00            1
 1.30            1
 0.10            1
 1.45            1
 31.80           1
-1.65            1
-0.20            1
Name: extra, dtype: int64
```

`Fare_amount: The time-and-distance fare calculated by the meter.`

The fare can be different, but it can't be less than 0 or astronomicaly high

In [16]:
```
df['fare_amount'].sort_values()
```

Out[16]:
```
4269251      -957.60
4334878      -434.00
2193262      -405.00
193432       -300.00
923034       -280.00
               ...
4751459      3039.00
3856423      4001.15
3838692      5000.00
8499603      8008.00
7461456    111270.85
Name: fare_amount, Length: 10906858, dtype: float64
```

`Total_amount The total amount charged to passengers`

The same situation as with `fare_amount`

In [17]:
```
df['total_amount'].sort_values()
```

Out[17]:
```
4269251      -958.40
4334878      -440.34
2193262      -405.30
193432       -300.80
923034       -280.30
               ...
4751459      3045.34
3856423      4002.05
3838692      5000.80
8499603      8008.80
7461456    111271.65
Name: total_amount, Length: 10906858, dtype: float64
```

`Tolls_amount Total amount of all tolls paid in trip.`

Can't be negative

In [18]:
```
df['tolls_amount'].sort_values()
```

Out[18]:
```
110255      -17.40
884722      -17.12
7726868     -12.50
10459645    -12.50
639043      -12.50
             ...
598602      882.22
5966724     885.59
3630019     900.10
992315      923.58
8548190     980.15
Name: tolls_amount, Length: 10906858, dtype: float64
```

`Tip_amount This field is automatically populated for credit card tips.`

Can't be negative

In [19]:
```
df['tip_amount'].sort_values()
```

Out[19]:
```
10724554   -220.80
3614225     -70.00
7766456     -65.00
9649637     -62.00
10269801    -34.64
             ...
8719432     550.00
5274108     800.00
180698      800.00
1617570     900.00
67897       998.14
Name: tip_amount, Length: 10906858, dtype: float64
```

`Payment_type: A numeric code signifying how the passenger paid for the trip.`

Should be:

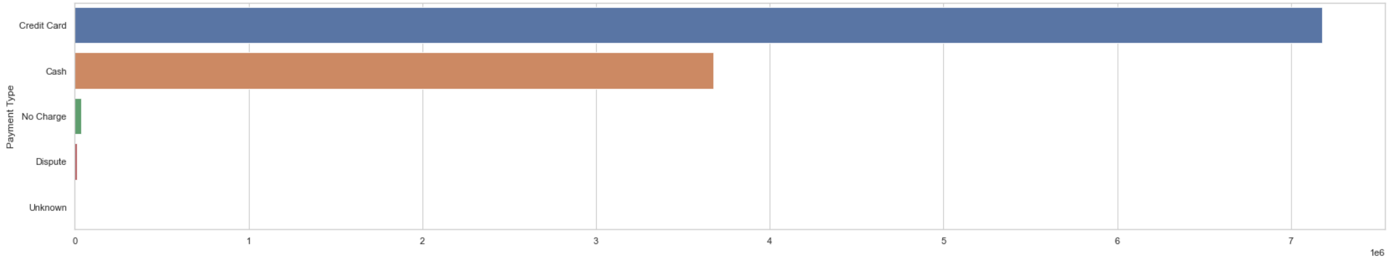
1. Credit card
2. Cash
3. No charge
4. Dispute
5. Unknown
6. Voided trip

In [20]:
```
df.payment_type.value_counts()
```

Out[20]:
```
1    7181476
2    3673651
3      38319
4      13411
5          1
Name: payment_type, dtype: int64
```

In [35]:
```
plt.figure(figsize=(28, 5))
sns.countplot(data=df, y='payment_type', orient='h')
plt.yticks(np.arange(5), ['Credit Card', 'Cash', 'No Charge', 'Dispute', 'Unknown'])
plt.ylabel('Payment Type')
```

```
plt.xlabel(None)
plt.show()
```



`Pickup_longitude` Longitude where the meter was engaged.

`Pickup_latitude` Latitude where the meter was engaged.

`Dropoff_longitude` Longitude where the meter was disengaged.

`Dropoff_ latitude` Latitude where the meter was disengaged.

Seems that there are 0 values in these columns, which means actually missing values or inproper geo location

In [31]:
```
# showing rows with any coordinate data equals to 0
df[(df['dropoff_longitude'] == 0) | (df['dropoff_latitude'] == 0) | (df['pickup_longitud
```

Out[31]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitu
38	1	2016-01-01 00:00:19	2016-01-01 00:19:33	1	3.40	0.0000
67	2	2016-01-01 00:00:41	2016-01-01 00:00:46	5	0.00	0.0000
150	1	2016-01-01 00:01:34	2016-01-01 00:15:38	1	2.90	0.0000
156	2	2016-01-01 00:01:36	2016-01-01 00:20:36	1	3.55	0.0000
158	2	2016-01-01 00:01:37	2016-01-01 00:25:25	2	1.53	0.0000
...	...	...	...	...	...	
10906034	1	2016-01-29 09:32:48	2016-01-29 09:47:49	1	2.50	0.0000
10906370	1	2016-01-29 09:33:58	2016-01-29 09:43:28	1	1.10	0.0000
10906404	1	2016-01-29 09:34:04	2016-01-29 09:44:03	1	0.60	0.0000
10906760	2	2016-01-29 14:52:29	2016-01-29 14:53:29	2	0.00	-73.9527
10906764	2	2016-01-29 15:55:10	2016-01-29 15:55:45	5	0.00	0.0000

185991 rows × 19 columns

There're **185991** records with missing geo data

`Store_and_fwd_flag` Y = store and forward trip, N = not a store and forward trip

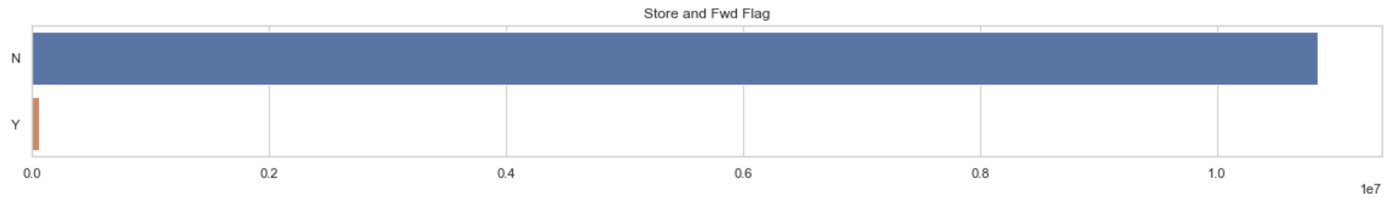
Should be either **Y** or **N**

In [32]:
```
df.store_and_fwd_flag.value_counts()
```

Out[32]:
```
N    10843625
Y       63233
```

```
Name: store_and_fwd_flag, dtype: int64
```

In [39]:
```
plt.figure(figsize=(20, 2))
sns.countplot(data=df, y='store_and_fwd_flag', orient='h')
plt.xlabel(None)
plt.ylabel(None)
plt.title('Store and Fwd Flag')
plt.show()
```



`RateCodeID` `The final rate code in effect at the end of the trip.`

1. Standard rate
2. JFK
3. Newark
4. Nassau or Westchester
5. Negotiated fare
6. Group ride

In [40]:
```
# What is 99 here?
df.RatecodeID.value_counts()
```

Out[40]:
```
1     10626315
2       225019
5        33688
3        16822
4         4696
99         216
6          102
Name: RatecodeID, dtype: int64
```

`Trip_distance` `The elapsed trip distance in miles reported by the taximeter.`

If measured correctly it shouldn't be equal to 0 or astronomicaly high

It can also be strange comparing to starting-ending time of the trip (1 hour long trip and 200 miles)

In [41]:
```
df.trip_distance.sort_values()
```

Out[41]:
```
10906857          0.0
2047967           0.0
6059957           0.0
6059997           0.0
2047928           0.0
                 ...
1256517     1403240.5
2768776     1653402.0
8551614     2441418.8
2708971     4667468.7
1027151     8000010.0
Name: trip_distance, Length: 10906858, dtype: float64
```

In [42]:
```
df[df['trip_distance']==0]
```

Out[42]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitu
67	2	2016-01-01 00:00:41	2016-01-01 00:00:46	5	0.0	0.0000
232	1	2016-01-29 09:18:28	2016-01-29 09:18:34	1	0.0	-73.9499
336	2	2016-01-29 09:18:45	2016-01-29 09:18:47	1	0.0	0.0000
425	2	2016-01-29 09:19:02	2016-01-29 09:19:06	1	0.0	-74.0051
448	1	2016-01-29 09:19:05	2016-01-29 09:19:05	1	0.0	-74.0099
...	...	...	...	...	...	
10906760	2	2016-01-29 14:52:29	2016-01-29 14:53:29	2	0.0	-73.9527
10906764	2	2016-01-29 15:55:10	2016-01-29 15:55:45	5	0.0	0.0000
10906779	2	2016-01-29 22:48:38	2016-01-29 22:48:47	1	0.0	-73.9871
10906854	1	2016-01-05 00:15:55	2016-01-05 00:16:06	1	0.0	-73.9454
10906857	1	2016-01-05 06:15:21	2016-01-05 06:15:36	3	0.0	-73.9609

64065 rows × 19 columns

In [43]:
```
df[df['trip_distance'] > 1000]
```

Out[43]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitud
1027151	1	2016-01-04 14:56:45	2016-01-04 15:27:47	1	8000010.0	-73.99897
1046137	1	2016-01-04 17:12:01	2016-01-04 17:27:36	1	633008.3	-73.99115
1256517	1	2016-01-05 22:51:22	2016-01-05 22:51:35	1	1403240.5	-73.94500
1256744	1	2016-01-05 22:53:29	2016-01-05 22:53:49	1	281060.3	-73.94502
2708971	1	2016-01-28 10:41:15	2016-01-28 11:03:53	1	4667468.7	-74.00715
2768776	1	2016-01-28 11:39:07	2016-01-28 11:41:29	2	1653402.0	-73.94566
8551614	1	2016-01-21 14:18:40	2016-01-21 14:27:53	4	2441418.8	-73.95209

`**Passenger_count** The number of passengers in the vehicle. This is a driver-entered value.`

0 passengers considered as mistake

In [44]:
```
df['passenger_count'].value_counts()
```

Out[44]:
```
1    7726984
2    1561977
5     601079
3     436431
6     369155
4     210641
0        520
8         26
9         23
7         22
Name: passenger_count, dtype: int64
```

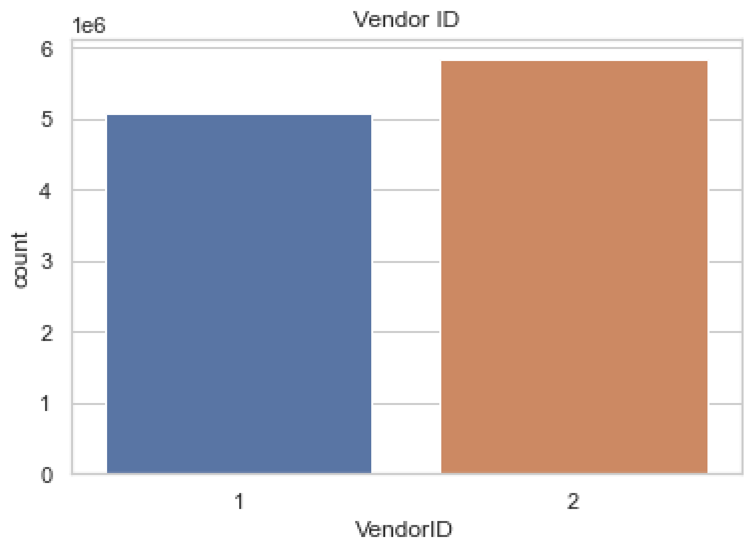
`VendorID` `A code indicating the TPEP provider that provided the record.

1. Creative Mobile Technologies
2. VeriFone Inc.

In [45]:
```
df['VendorID'].value_counts()
```

Out[45]:
```
2    5835429
1    5071429
Name: VendorID, dtype: int64
```

In [46]:
```
sns.countplot(data=df, x='VendorID');
plt.xticks([0,1], ['1', '2'])
plt.title('Vendor ID')
plt.show()
```



2: Data Analysis

Data Preprocessing

It looks that there are some incorrect values in the dataset, so it needs to be cleaned and preprocessed for further analysis.

There are several ways to deal with incorrect or missing values. One way, for example, is to impute or replace bad values by the most closely to truth. However, in this case I decided to drop trips where we have totally wrong numberic values, because one improper value led to the final `total_amount` mistake and also because amog more that 10 mio trips we have just few rows with wrong data, so deleting them won't change the overall picture, but will definetely increase the accuracy of our analysis.

Steps to perform:

1. Change all money-related negative values to non-negative by taking their absolute values
2. Remove trips where `improvement_surcharge` is not equal to 0.3 or 0
3. Remove trips where `mta_tax` is not equal to 0 or .5
4. Remove trips where `extra` is not equal to .5, 1 or 0
5. Remove extremely high `Fare_amount` and `total_amount`
6. Replace '99' in `RateCodeID` with the most frequently observed `1`

7. Remove rows with extremely high values of `Tip_amount`, `Tolls_amount`, `fare_amount` and `Total_amount`
8. Convert date columns to datetime format

Negative to non-negative

In [47]:
```
data = df.copy()
data.columns
```

Out[47]:
```
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'pickup_longitude',
       'pickup_latitude', 'RatecodeID', 'store_and_fwd_flag',
       'dropoff_longitude', 'dropoff_latitude', 'payment_type', 'fare_amount',
       'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
       'improvement_surcharge', 'total_amount'],
      dtype='object')
```

In [48]:
```
# take columns with money-related numerical values
data.columns[-7:]
```

Out[48]:
```
Index(['fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
       'improvement_surcharge', 'total_amount'],
      dtype='object')
```

In [49]:
```
# show how much negative values we have there
(data.loc[:, data.columns[-7:]] < 0).sum()
```

Out[49]:
```
fare_amount              4216
extra                    2007
mta_tax                  4062
tip_amount                128
tolls_amount               24
improvement_surcharge    4202
total_amount             4217
dtype: int64
```

In [50]:
```
# take the absolute value of all our money-related values
data.loc[:, data.columns[-7:]] = data.loc[:, data.columns[-7:]].abs()
```

In [51]:
```
# now we don't have negatives
(data.loc[:, data.columns[-7:]] < 0).sum()
```

Out[51]:
```
fare_amount              0
extra                    0
mta_tax                  0
tip_amount               0
tolls_amount             0
improvement_surcharge    0
total_amount             0
dtype: int64
```

Remove trips with wrong `extra`, `mta_tax` and `improvement_surcharge` data

In [53]:
```
# there are only 159 rows with incorrect money values, so delete them
data[(~data['improvement_surcharge'].isin([.3, 0])) |
     (~data['mta_tax'].isin([.5, 0])) |
     (~data['extra'].isin([.5, 1, 0]))].shape[0]
```

Out[53]:
```
159
```

In [54]:
```
# drop the rows with bad values
data.drop(data[(~data['improvement_surcharge'].isin([.3, 0])) |
               (~data['mta_tax'].isin([.5, 0])) |
               (~data['extra'].isin([.5, 1, 0]))].index, inplace=True)
```

In [55]:
```
# confirm that we deleted only 159 rows
df.shape[0] - data.shape[0]
```

Out[55]:
```
159
```

Replace '99' in `RateCodeID` to '1'

In [56]:
```
# number of rows where 'RatecodeID' is 99 before
data[data['RatecodeID'] == 99].shape[0]
```

Out[56]:
```
214
```

In [57]:
```
# replacing '99' with '1', keeping other untouched
data['RatecodeID'] = np.where(data['RatecodeID'] == 99, 1, data['RatecodeID'])
```

In [58]:
```
# number of rows where 'RatecodeID' is 99 after
data[data['RatecodeID'] == 99].shape[0]
```

Out[58]:
```
0
```

Dealing with `Tip_amount`, `Tolls_amount`, `fare_amount` and `Total_amount`

We have some outlier values in these columns. However, we don't know if all of them wrong or not.

I would propose to keep all of the high values, except top 2 of `total_amount` and `fare_amount`, for **Talk A**.

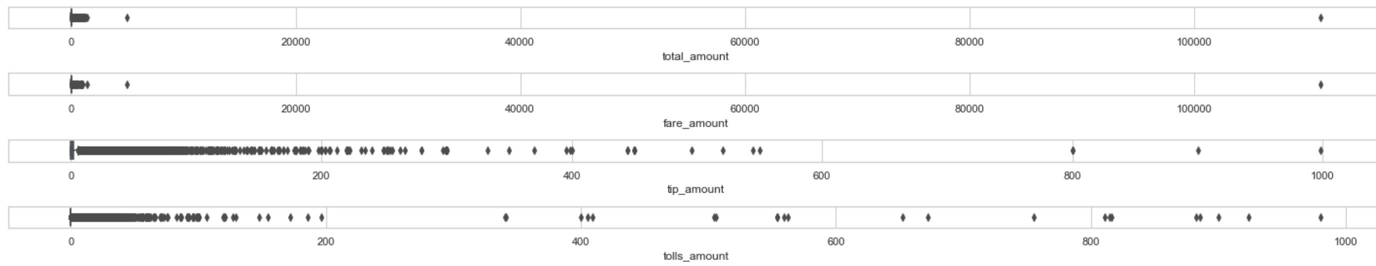
For **Task B** we'll probably remove the outliers in order to increase the model accuracy.

In [59]:
```
data[['tip_amount', 'tolls_amount', 'fare_amount', 'total_amount']].describe(percentiles
```

Out[59]:

	count	mean	std	min	50%	75%	99.99%	max
tip_amount	10906699.0	1.750782	2.623411	0.0	1.26	2.32	50.000000	998.14
tolls_amount	10906699.0	0.293374	1.694555	0.0	0.00	0.00	21.800000	980.15
fare_amount	10906699.0	12.493146	35.446353	0.0	9.00	14.00	190.000000	111270.85
total_amount	10906699.0	15.648501	36.296233	0.0	11.62	17.16	215.109812	111271.65

In [60]:
```
fig, axs = plt.subplots(4, 1, figsize=(20, 4))
sns.boxplot(data=data, x='total_amount', ax=axs[0])
sns.boxplot(data=data, x='fare_amount', ax=axs[1])
sns.boxplot(data=data, x='tip_amount', ax=axs[2])
sns.boxplot(data=data, x='tolls_amount', ax=axs[3])
plt.tight_layout()
plt.show()
```



In [61]:
```
# what are those 2 highest values of fare_amount?
top2fare = data['fare_amount'].sort_values(ascending=False).head(2)
top2fare
```

Out[61]:
```
7461456    111270.85
3838692      5000.00
Name: fare_amount, dtype: float64
```

In [62]:
```
# what are those 2 highest values of total_amount?
top2total = data['total_amount'].sort_values(ascending=False).head(2)
top2total
```

Out[62]:
```
7461456    111271.65
3838692      5000.80
Name: total_amount, dtype: float64
```

In [63]:
```
data[data['fare_amount'].isin(top2fare.tolist())]
```

Out[63]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitud
3838692	1	2016-01-25 16:32:07	2016-01-25 16:32:12	1	0.0	-73.98144
7461456	1	2016-01-30 14:41:23	2016-01-30 14:48:55	1	0.9	-73.99163

In [64]:
```
data[data['total_amount'].isin(top2total.tolist())]
```

Out[64]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitud
3838692	1	2016-01-25 16:32:07	2016-01-25 16:32:12	1	0.0	-73.98144
7461456	1	2016-01-30 14:41:23	2016-01-30 14:48:55	1	0.9	-73.99163

Let's remove from our dataframe these two trips

In [65]:
```
data.shape
```

Out[65]:
```
(10906699, 19)
```

In [66]:
```
# removing top 2 values of total_amount and fare_amount
data.drop(data[data['fare_amount'].isin(top2fare.tolist())].index, inplace=True)
```

In [67]:
```
data.shape
```

Out[67]:
```
(10906697, 19)
```

Convert date columns to datetime format

In [68]:
```
data['tpep_pickup_datetime'] = pd.to_datetime(data['tpep_pickup_datetime'])
data['tpep_dropoff_datetime'] = pd.to_datetime(data['tpep_dropoff_datetime'])
```

In [69]:
```
data.dtypes
```

Out[69]:
```
VendorID                          int64
tpep_pickup_datetime     datetime64[ns]
tpep_dropoff_datetime    datetime64[ns]
passenger_count                   int64
trip_distance                   float64
pickup_longitude                float64
pickup_latitude                 float64
RatecodeID                        int64
store_and_fwd_flag               object
dropoff_longitude               float64
dropoff_latitude                float64
payment_type                      int64
fare_amount                     float64
extra                           float64
mta_tax                         float64
tip_amount                      float64
tolls_amount                    float64
improvement_surcharge           float64
total_amount                    float64
dtype: object
```

Task A

For this task we'll need to work with datetime and total amount data from the preprocessed dataset.

A target datetime column will be `tpep_pickup_datetime`. I will calculate daily and hourly trends based on the time of the taxi trip starts.

Also, I will create temporary additional columns to answer the questions.

In [70]:
```
# take only 2 needed columns from the cleaned dataset
data_1 = data[['tpep_pickup_datetime', 'total_amount']].copy()
```

In [71]:
```
# create additional columns for Task A.2 and Task A.3
# morning: 06:00 - 11:59
# afternoon: 12:00-17:59
# evening: 18:00-23:59
# night: 00:00-05:59
data_1['day_time'] = 'NA'
data_1['day_time'] = np.where(data_1['tpep_pickup_datetime'].dt.hour.isin(np.arange(6, 1
data_1['day_time'] = np.where(data_1['tpep_pickup_datetime'].dt.hour.isin(np.arange(12,
data_1['day_time'] = np.where(data_1['tpep_pickup_datetime'].dt.hour.isin(np.arange(18,
data_1['day_time'] = np.where(data_1['tpep_pickup_datetime'].dt.hour.isin(np.arange(0, 6

# day names
data_1['day_name'] = data_1['tpep_pickup_datetime'].dt.day_name()

data_1
```

Out[71]:

	tpep_pickup_datetime	total_amount	day_time	day_name
0	2016-01-01 00:00:00	8.80	night	Friday
1	2016-01-01 00:00:00	19.30	night	Friday
2	2016-01-01 00:00:00	34.30	night	Friday
3	2016-01-01 00:00:00	17.30	night	Friday
4	2016-01-01 00:00:00	8.80	night	Friday
...	...	...	...	...

10906853	2016-01-31 23:30:32	9.80	evening	Sunday
10906854	2016-01-05 00:15:55	3.80	night	Tuesday
10906855	2016-01-05 06:12:46	8.80	morning	Tuesday
10906856	2016-01-05 06:21:44	14.75	morning	Tuesday
10906857	2016-01-05 06:15:21	58.34	morning	Tuesday

10906697 rows × 4 columns

Task A.1

What is the average demand for the taxis in the days of the week (i.e., daily trend). Which of the days has the highest and which lowest demand?

In [72]:

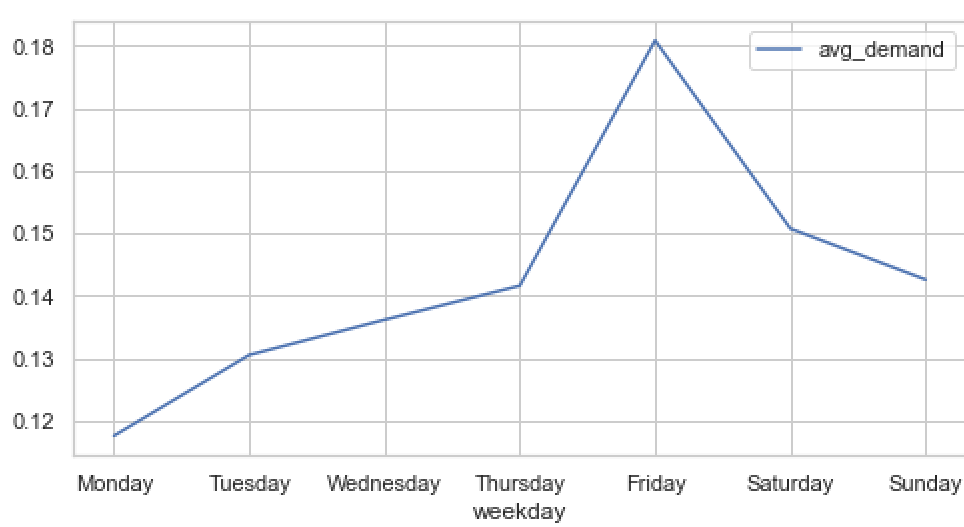
```
daily_demand = data_1[['tpep_pickup_datetime']].groupby(data_1['tpep_pickup_datetime'].d
daily_demand.index.rename('weekday', inplace=True)
daily_demand['avg_demand'] = daily_demand['avg_demand']/data.shape[0]
daily_demand
```

Out[72]:

weekday	avg_demand
0	0.117576
1	0.130512
2	0.136143
3	0.141594
4	0.180859
5	0.150737
6	0.142580

In [75]:

```
plt.figure(figsize=(8, 4))
sns.lineplot(data=daily_demand)
plt.xticks(np.arange(7), ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Satur
plt.show()
```



Top demand day is **Friday**

Task A.2

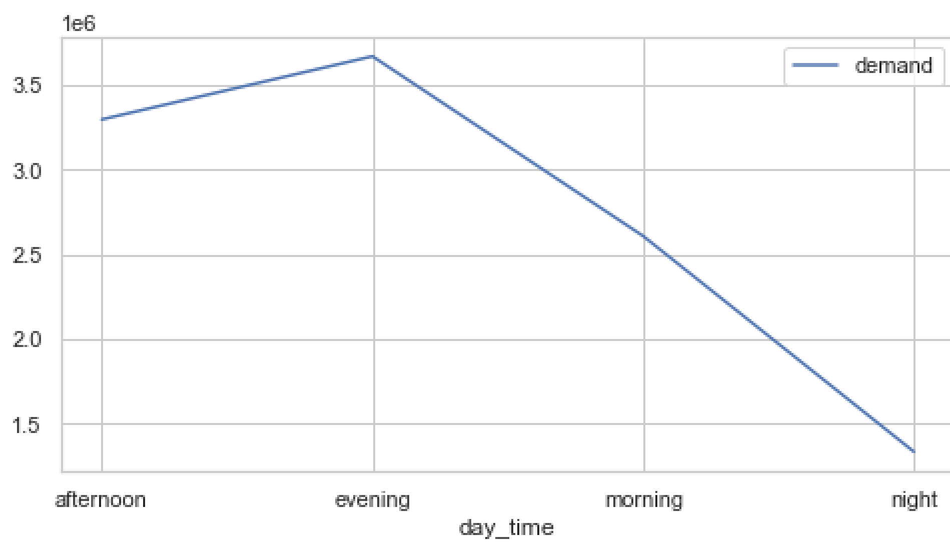
Which time of the day (morning, afternoon, evening, and night) is likely be a peak period for the taxi's operation from the data?

In [76]:
```
hourly_demand = data_1.groupby(data_1['day_time']).tpep_pickup_datetime.count().to_frame
hourly_demand = hourly_demand.rename(columns={'tpep_pickup_datetime':'demand'})
hourly_demand
```

Out[76]:

day_time	demand
afternoon	3296961
evening	3670543
morning	2607360
night	1331833

In [77]:
```
plt.figure(figsize=(8, 4))
sns.lineplot(data=hourly_demand)
plt.show()
```



Evening is the peak part of the day in terms of taxi demand

Task A.3

On average, how much revenue was generated in the weekdays and weekends for the business for the period covered in the dataset?

In [78]:
```
data_1['working_weekend'] = np.where(data_1.tpep_pickup_datetime.dt.weekday.isin(np.aran
```

In [79]:
```
data_1.working_weekend.value_counts()
```

Out[79]:
```
weekday    7707582
weekend    3199115
```

```
Name: working_weekend, dtype: int64
```

In [82]:
```python
# group by weekday-weekend
data_1.groupby('working_weekend').mean().rename(columns={'total_amount':'average_revenue
```

Out[82]:

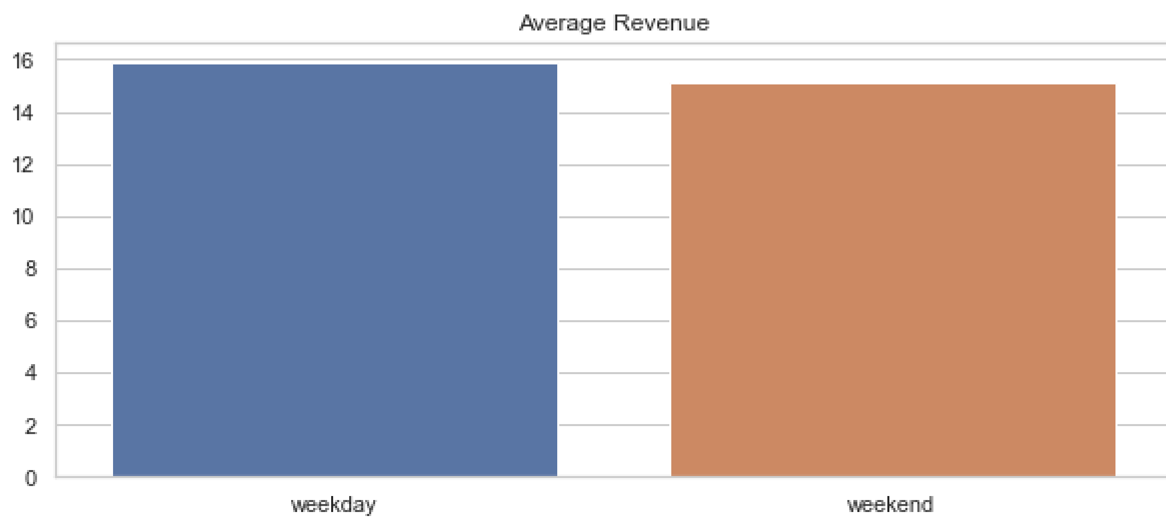
	average_revenue
working_weekend	
weekday	15.866139
weekend	15.087812

In [83]:
```python
# group by day name
data_1.groupby('day_name').mean().rename(columns={'total_amount':'average_revenue'})
```

Out[83]:

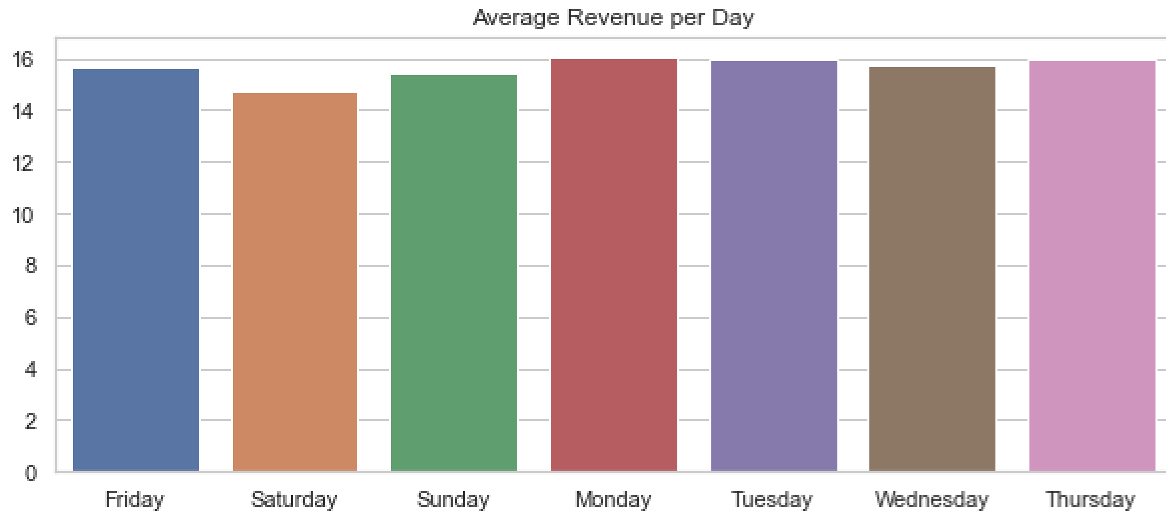
	average_revenue
day_name	
Friday	15.671595
Monday	16.037919
Saturday	14.748544
Sunday	15.446491
Thursday	15.953231
Tuesday	15.997315
Wednesday	15.759897

In [87]:
```python
# plot the average revenue for two groups of days - weekdays and weekends
plt.figure(figsize=(10, 4))
sns.barplot(data=data_1, x='working_weekend', y='total_amount', ci=None)
plt.xlabel(None)
plt.ylabel(None)
plt.title('Average Revenue')
plt.show()
```



In [89]:
```python
# plot the average revenue for day name
plt.figure(figsize=(10, 4))
sns.barplot(data=data_1, x='day_name', y='total_amount', ci=None)
plt.xlabel(None)
plt.ylabel(None)
```

```python
plt.title('Average Revenue per Day')
plt.show()
```



3: Regression Analysis

Task B

First of all, in order to have max accurate prediction model I will remove all outliers from the data. Those values are not necessarily the mistakes. That's why I didn't remove them before analyzing weekly/daily patterns. But those outliers can badly influence on the prediction model performance, so I remove them now.

---

1. `passenger_count` - Number of passengers. 7,8,9 are rare enough to not count them. 0 passengers is meaningless.

In [90]:
```python
data.passenger_count.value_counts()
```

Out[90]:
```
1    7726841
2    1561964
5     601079
3     436428
6     369155
4     210640
0        519
8         26
9         23
7         22
Name: passenger_count, dtype: int64
```

In [91]:
```python
# keep only the trips with 1 t o 6 passengers
data_b = data[data['passenger_count'].isin([1, 2, 3, 4, 5, 6])]
```

---

1. `Fare_amount` - seems to be the most influencial and the highest contributor to the target `total_amount` variable. There are some very large unreal values. So, I will keep only those that are inside the 99.95% percentile.

In [92]:
```
# keep only first 99.95% of fare_amounts
data_b = data_b[data_b['fare_amount'] <= data_b['fare_amount'].quantile(0.9995)]
```

---

1. `trip_distance` - Unfortunately, some of the records are incorrect. I'll keep the first 99.95% of these values as well.

In [93]:
```
# keep only the records with trip distance less than the 99.95% percentile
data_b = data_b[data_b['trip_distance'] <= data_b['trip_distance'].quantile(0.9995)]
```

---

1. `tip_amount` - Although the tips can be huge, they're rarely greater than certain amount. I'll keep the first 99.95% of these values as well.

In [94]:
```
# keep only the records with tip amount less than the 99.95% percentile
data_b = data_b[data_b['tip_amount'] <= data_b['tip_amount'].quantile(0.9995)]
```

---

1. `tolls_amount` - The same approach here - I'll keep the first 99.95% of these values.

In [95]:
```
# keep only the records with tolls amount less than the 99.95% percentile
data_b = data_b[data_b['tolls_amount'] <= data_b['tolls_amount'].quantile(0.9995)]
```

---

1. `payment_type` - seems that only payments made by credit card and in cash are meaningfull, so I'll keep only types 1 and 2.

In [96]:
```
# keep only the records with payment type 1 and 2
data_b = data_b[data_b['payment_type'].isin([1,2])]
```

In [97]:
```
# now the numerical values are close to reality
data_b.describe(percentiles=[.25, .5, .75, .9995]).T
```

Out[97]:

	count	mean	std	min	25%	50%	75%	99.!
VendorID	10833472.0	1.537194	0.498615	1.000000	1.000000	2.000000	2.000000	2.000
passenger_count	10833472.0	1.672939	1.326974	1.000000	1.000000	1.000000	2.000000	6.000
trip_distance	10833472.0	2.864503	3.477488	0.000000	1.000000	1.670000	3.060000	23.400
pickup_longitude	10833472.0	-72.829468	9.127288	-121.934288	-73.991508	-73.981392	-73.966164	0.000
pickup_latitude	10833472.0	40.120878	5.028075	0.000000	40.736408	40.753738	40.768108	40.851
RatecodeID	10833472.0	1.032457	0.251059	1.000000	1.000000	1.000000	1.000000	5.000
dropoff_longitude	10833472.0	-72.894300	8.869680	-121.933487	-73.991058	-73.979424	-73.962021	0.000
dropoff_latitude	10833472.0	40.157452	4.886301	0.000000	40.734928	40.754189	40.769634	40.896
payment_type	10833472.0	1.338691	0.473265	1.000000	1.000000	1.000000	2.000000	2.000
fare_amount	10833472.0	12.339944	10.014654	0.000000	6.500000	9.000000	14.000000	75.000
extra	10833472.0	0.313497	0.365483	0.000000	0.000000	0.000000	0.500000	1.000
mta_tax	10833472.0	0.498611	0.026318	0.000000	0.500000	0.500000	0.500000	0.500
tip_amount	10833472.0	1.728290	2.235994	0.000000	0.000000	1.260000	2.320000	17.500

tolls_amount	10833472.0	0.274108	1.252355	0.000000	0.000000	0.000000	0.000000	12.500
improvement_surcharge	10833472.0	0.299960	0.003468	0.000000	0.300000	0.300000	0.300000	0.300
total_amount	10833472.0	15.454398	12.301355	0.000000	8.300000	11.620000	17.160000	95.150

In [98]:
```
data_b.shape
```

Out[98]:
```
(10833472, 19)
```

Regression models work with numerical variables, but we have a timestamp columns in the dataset - `tpep_pickup_datetime` and `tpep_dropoff_datetime`. I can't just remove them, because I lose important information then. So I extract `day`, `weekday` and `hour` from the pick-up datetime column to keep it for the regression model.

I will also convert `Y` and `N` values in the column `store_and_fwd_flag` to `1` and `0` respectively.

In [99]:
```
# convert 'Y' and 'N' to '1' and '0'
data_b.store_and_fwd_flag = np.where(data_b.store_and_fwd_flag == 'Y', 1, 0)
```

In [100…
```
# create additional columns by extracting weekday, day and hour from 'tpep_pickup_dateti
data_b['weekday'] = data_b['tpep_pickup_datetime'].dt.weekday
data_b['day'] = data_b['tpep_pickup_datetime'].dt.day
data_b['hour'] = data_b['tpep_pickup_datetime'].dt.hour
```

In [101…
```
# remove timestamp columns
data_b.drop(['tpep_pickup_datetime', 'tpep_dropoff_datetime'], axis=1, inplace=True)
```

In [103…
```
# so, we have only numerical values left here
data_b.dtypes
```

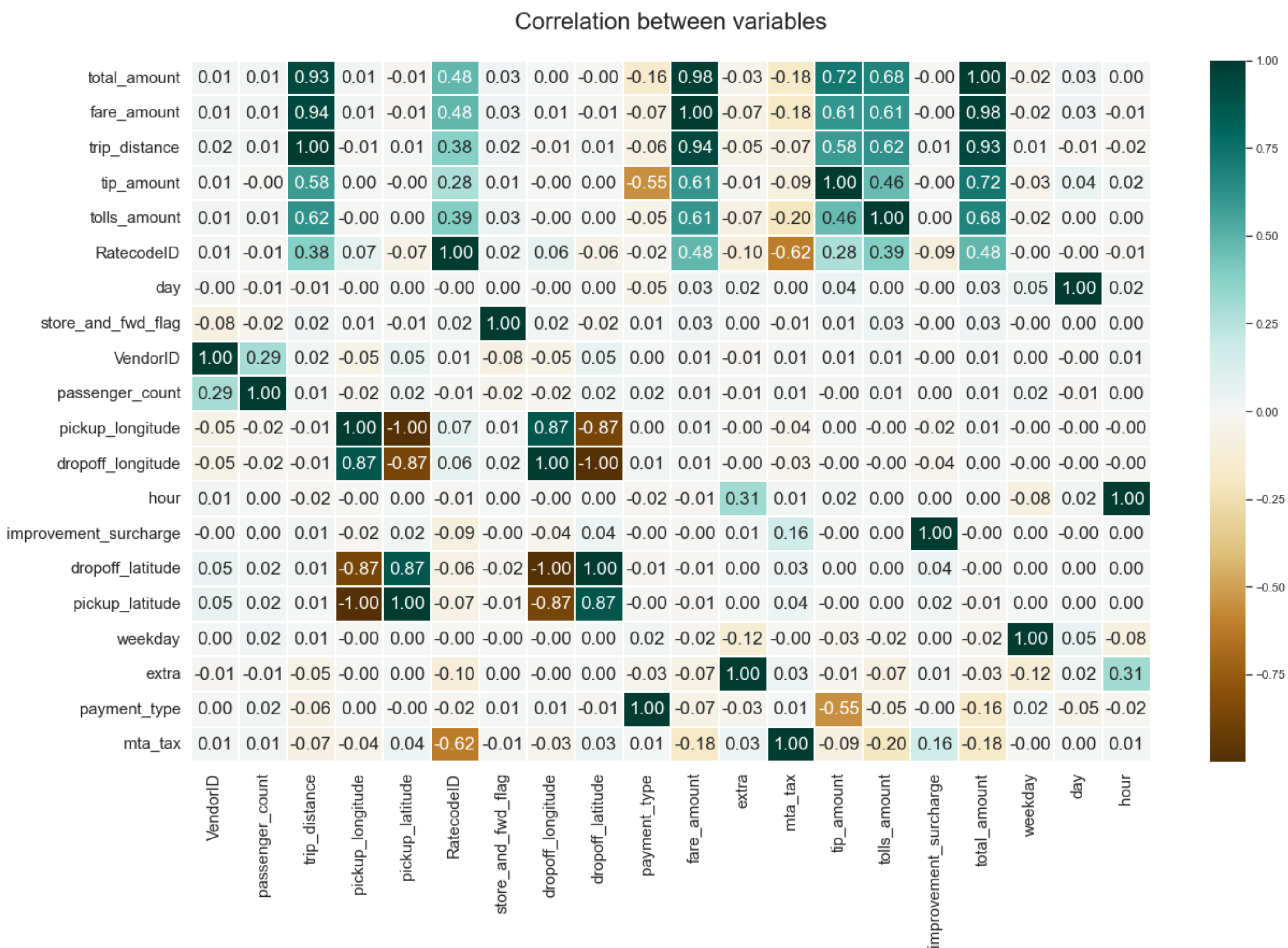
Out[103]:
```
VendorID                   int64
passenger_count            int64
trip_distance            float64
pickup_longitude         float64
pickup_latitude          float64
RatecodeID                 int64
store_and_fwd_flag         int32
dropoff_longitude        float64
dropoff_latitude         float64
payment_type               int64
fare_amount              float64
extra                    float64
mta_tax                  float64
tip_amount               float64
tolls_amount             float64
improvement_surcharge    float64
total_amount             float64
weekday                    int64
day                        int64
hour                       int64
dtype: object
```

Before creating a **regression model**, let's see which **variables** have the strongest and weakest **correlation** with `fare_amount` and also which variables correlate with each other

In [128…
```
# correlation between variables
```

```
corr = data_b.corr().sort_values(by='total_amount', ascending=False)

# plot it
fig, ax = plt.subplots(figsize = (20,12))
sns.heatmap(corr, annot = True, cmap ='BrBG', ax = ax, fmt='.2f', linewidths = 0.05, ann
ax.tick_params(labelsize = 15)
ax.set_title('Correlation between variables\n', fontsize = 22)
plt.savefig('taxi_dataset/corr.png')
plt.show()
```



We see strong positive correlations between `total_amount` and `trip_distance`, `fare_amount`, `tip_amount` and `tolls_amount`. And it looks logical.

Therefore, I will use a **linear regression model** trying to find the most **features** that explain our **target** variable and also their **contribution** to its value.

In [106…

```
# import libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error # metrics
```

Our target variable `y` is a `total_amount` column and our features `X` are all other numerical preprocessed variables from `data_b`.

In [107…

```
X = data_b.drop('total_amount', axis=1)
y = data_b['total_amount']
```

Now, let's split sequentially our dataset to `train` and `test` sets for further training and testing the regression model

In [108…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

Train the **Linear Regression** model with default properties

In [109…
```
lin_reg = LinearRegression()

# training the model
lin_reg.fit(X_train, y_train)
```

Out[109]:
```
▾ LinearRegression
LinearRegression()
```

Now use the trained model to predict `total_amount` based on the features from the `test` part of our dataset.

In [110…
```
# Make prediction
y_pred = lin_reg.predict(X_test)
```

Model Evaluation - checking the model accuracy

Now we have 20% of our dataset with initial values of `total_amount` - **y_test** and predicted - **y_pred**.

We'll compare them using statistical metrics.

So, in order to assess the accuracy of the model, I'll use the following metrics:

- MSE - mean_squared_error
- RMSE - Root-mean-square deviation
- R2 - coefficient of determination
- MAE - mean_absolute_error

In [111…
```
# for RMSE we need a small function
def rmse(y, y_pred):
    return np.sqrt(mean_squared_error(y, y_pred))
```

In [112…
```
# calculating and printing the metrics
print('The lowest the better:\n')
print(f'MAE: {mean_squared_error(y_test, y_pred)}')
print(f'RMSE: {rmse(y_test, y_pred)}')
print(f'MAE: {mean_absolute_error(y_test, y_pred)}')
print()
print('The closest to 1 the better:\n')
print(f'R2 score: {r2_score(y_test, y_pred)}')
```

```
The lowest the better:

MAE: 0.00011284654974338399
RMSE: 0.010622925667789641
MAE: 7.775121876345914e-05

The closest to 1 the better:

R2 score: 0.9999992264251094
```

Find regression coefficients and make an equation

In [113…
```
#display intercept, regression coefficients and R-squared value of model
print(f'Intercept: {lin_reg.intercept_}', f'Coefficients: {lin_reg.coef_}', f'Score: {li
```

```
Intercept: -0.018414604110684962

Coefficients: [-1.74855084e-05  1.38065078e-06 -8.36300300e-05 -1.20375749e-05
 -2.07685216e-05 -7.00851006e-05  2.19626915e-05  7.58173577e-05
  1.36813565e-04 -2.43817909e-05  1.00003145e+00  1.00003036e+00
  1.00130611e+00  1.00000017e+00  9.99980461e-01  1.05911905e+00
  6.84015799e-06 -1.50443511e-06 -9.63564164e-07]

Score: 0.9999996199789792
```

In [114…
```
print(f'We can see that the R2 value of the model is {lin_reg.score(X, y)}')
```

```
We can see that the R2 value of the model is 0.9999996199789792
```

This means that 99.99% of the variation of the `total_amount` variable can be explained by our variables in the model.

Now, I show the `features` and their `coefficients` to make a final `equation`

In [116…
```
coefs = ['{:f}'.format(item) for item in lin_reg.coef_]
```

In [117…
```
pd.DataFrame(data=coefs, index=X_train.columns)
```

Out[117]:

	0
VendorID	-0.000017
passenger_count	0.000001
trip_distance	-0.000084
pickup_longitude	-0.000012
pickup_latitude	-0.000021
RatecodeID	-0.000070
store_and_fwd_flag	0.000022
dropoff_longitude	0.000076
dropoff_latitude	0.000137
payment_type	-0.000024
fare_amount	1.000031
extra	1.000030
mta_tax	1.001306
tip_amount	1.000000
tolls_amount	0.999980
improvement_surcharge	1.059119
weekday	0.000007

day	-0.000002
hour	-0.000001

Regression Equation

total_amount = -0.018414604110684962 - Vendor * 0.000017 + passenger_count * 0.000001 - trip_distance * 0.000084 - pickup_longitude * 0.000012 - pickup_latitude * 0.000021 - RatecodeID * 0.000070 + store_and_fwd_flag * 0.000022 + dropoff_longitude * 0.000076 + dropoff_latitude * 0.000137 - payment_type * 0.000024 + fare_amount * 1.000031 + extra * 1.000030 + mta_tax * 1.001306 + tip_amount * 1.000000 + tolls_amount * 0.999980 + improvement_surcharge * 1.059119 + weekday * 0.000007 - day * 0.000002 - hour * 0.000001

It looks like the `accuracy` of our `trained model` is good enough, so we can use it to `predict` the `total amount` paid on a trip for the trip records shown in `New Sample` file

In [122…

```
# loading new_sample dataset
new_sample = pd.read_parquet('taxi_dataset/new.parquet')
new_sample.head()
```

Out[122]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitude	pick
0	2	2/25/2016 17:24	2/25/2016 17:27	2	0.70	-73.947250	
1	2	2/25/2016 23:10	2/25/2016 23:31	2	5.52	-73.983017	
2	2	2/1/2016 0:00	2/1/2016 0:10	6	1.99	-73.992340	
3	1	2/1/2016 0:00	2/1/2016 0:05	1	1.50	-73.981453	
4	2	2/1/2016 0:00	2/1/2016 0:20	1	5.60	-74.000603	

We need to `preprocess` the `new` data first in the same way as we did it with the `trained dataset`

In [123…

```
new_sample.tpep_pickup_datetime = pd.to_datetime(new_sample.tpep_pickup_datetime)
new_sample.store_and_fwd_flag = np.where(new_sample.store_and_fwd_flag == 'Y', 1, 0)
new_sample['weekday'] = new_sample['tpep_pickup_datetime'].dt.weekday
new_sample['day'] = new_sample['tpep_pickup_datetime'].dt.day
new_sample['hour'] = new_sample['tpep_pickup_datetime'].dt.hour
new_sample.drop(['tpep_pickup_datetime', 'tpep_dropoff_datetime'], axis=1, inplace=True)
new_sample.head()
```

Out[123]:

	VendorID	passenger_count	trip_distance	pickup_longitude	pickup_latitude	RatecodeID	store_and_fwd_flag
0	2	2	0.70	-73.947250	40.763771	1	0
1	2	2	5.52	-73.983017	40.750992	1	0
2	2	6	1.99	-73.992340	40.758202	1	0
3	1	1	1.50	-73.981453	40.749722	1	0
4	2	1	5.60	-74.000603	40.729755	1	0

Now run the `trained model` on the values to predict the `total_amount`

In [124…

```
new_sample_total_amount = lin_reg.predict(new_sample)
```

In [125…

```
predictions = pd.DataFrame(new_sample_total_amount, columns=['predicted_total_amount'])
```

Tabulating the predicted values in the order the records are arranged in the file

In [126…

```
predictions
```

Out[126]:

	predicted_total_amount
0	5.799895
1	21.299977
2	11.500012
3	7.799944
4	25.300023
5	17.299952
6	9.359952
7	7.799964
8	9.799941
9	17.299981
10	11.759946
11	17.299992
12	8.999970
13	18.000025
14	12.359986
15	6.959957
16	20.159999
17	21.960021
18	36.339773
19	10.789982
20	68.801417
21	53.300021
22	12.739993
23	8.759945
24	28.560024
25	15.359945
26	18.960044
27	10.299996
28	20.160012
29	12.799964
30	8.299934
31	17.160009
32	9.799943
33	6.799933
34	25.560041

35	9.299946
36	4.799931
37	29.749984
38	7.239958
39	8.799992

4: Discussion

We have trained a Linear Regression model with default properties and achieved outstanding results in predicting our target variable using the listed features. Despite the availability of many other regression models, we didn't need to explore them as our simple model produced a remarkable accuracy of 99.99% on the test dataset.

This model serves as a valuable tool for predicting the total amount of a taxi drive and determining which features contribute most significantly to the amount value.

Prior to modeling, we preprocessed the data, making some assumptions and allowances. We observed numerous incorrect values and mistakes in the data. For example, some trip distances were either null or unrealistically large. To rectify this, we could have calculated the distance between the pick-up and drop-off geo locations, but this is also not reliable due to numerous records containing zeros or identical values.

Ideally, we could have verified and corrected values in the datasets using some calculations to obtain more precise data. However, our model produced excellent results even without such refinements.

5: Conclusion

Based on the analysis of the taxi trip dataset, several key findings were uncovered. Firstly, the average demand for taxis varied across different days of the week. Specifically, Friday had the highest demand while Monday had the lowest demand. Secondly, the peak period for taxi operation was observed to be in the evening hours, particularly during rush hour times.

Additionally, it was found that the average revenue generated by the taxi business was slightly higher during weekdays compared to weekends.

After preprocessing and cleaning the data, a linear regression model was created to predict the total amount paid for a taxi trip, given trip information such as time, distance, fees, and fares. The model was trained on 80% of the data and tested on the remaining 20% to ensure its generalization abilities. The final model showed a good performance in predicting the total amount paid for a taxi trip, with a low RMSE (0.0106) and high R2 score (0.9999).

In conclusion, the analysis of the taxi trip dataset provided insights into the daily trends and peak periods for taxi operation, as well as the factors affecting the total amount paid for a trip. The developed regression model could be used to predict the total amount paid for a taxi trip, helping the taxi business to optimize their pricing strategies and increase revenue.

6: Reference

Data: The dataset was sourced from the NYC Taxi & Limousine Commission (TLC) official website.

The dataset contains several explanatory variables used to assess a completed trip such as pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts.

A subsample of the original data is provided to use for the tasks itemised in the task sections. The trip records are divided into two files Main Sample and New Sample, respectively.

In [ ]:

```

```