working with a subset of the dataset containing 100,000 randomly sampled rows. Use the last 4 digits(6708) of your student ID as the random seed to ensure consistency across submissions. Loaded the dataset and performed the sampling:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from geopy.distance import geodesic
import time
from IPython.display import clear_output
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('train.csv')

# Set the random seed
np.random.seed(6708)

# Sample 100,000 rows
df = df.sample(n=100000)
```

This command is used to check random 5 rows to check the data.

```
df.head(5)
```

	id	ve	picku	drop	pass	picku	pick	dropo	drop	store_	trip
548	id2 11	1	2016- 01-15	2016- 01-15	1	-73,99	40,74	-73,99	40,73	N	494
243	id1 95	2	2016- 05-10	2016- 05-10	2	-73,98	40,75	-73,99	40,76	N	445
111	id3 65	1	2016- 01-01	2016- 01-01	1	-74,00	40,75	-73,98	40,78	N	355
950	id0 84	2	2016- 03-06	2016- 03-06	1	-73,97	40,80	-73,94	40,80	N	932
678	id2 81	2	2016- 04-30	2016- 04-30	3	-74,00	40,67	-73,99	40,67	N	276

- **->Provided summary statistics for the dataset.**

```
# Generated summary statistics for the existing DataFrame "df"
df.describe()
```

	vend	passenge	pickup_lo	pickup_l	dropoff_lo	dropoff_l	trip_du
co	100000	100000,00	100000,00	100000,00	100000,00	100000,00	100000,0
me	1,54	1,66	-73,97	40,75	-73,97	40,75	951,60
std	0,50	1,31	0,16	0,03	0,16	0,03	3197,40
mi	1,00	0,00	-121,93	37,39	-121,93	37,39	1,00
25 %	1,00	1,00	-73,99	40,74	-73,99	40,74	394,00
50 %	2,00	1,00	-73,98	40,75	-73,98	40,75	662,00
75 %	2,00	2,00	-73,97	40,77	-73,96	40,77	1071,00
ma	2,00	7,00	-65,90	41,12	-65,90	41,42	86362,00

- Check the data types of the features.

```
df.dtypes
# Check the data types of the features in the existing DataFrame "df"
df.dtypes
```

```
id                    object
vendor_id              int64
pickup_datetime       object
dropoff_datetime      object
passenger_count        int64
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
store_and_fwd_flag    object
trip_duration          int64
dtype: object
```

```
# Checking for missing data and dropping the null values
df = df.dropna()
```

Checking for null values if they are removed and calculating the count:

```
missing_data = df.isnull().sum()
print("Missing data counts:")
print(missing_data)
```

```
Missing data counts:
id                    0
vendor_id             0
pickup_datetime       0
```

```
dropoff_datetime      0
passenger_count       0
pickup_longitude      0
pickup_latitude       0
dropoff_longitude     0
dropoff_latitude      0
store_and_fwd_flag    0
trip_duration         0
dtype: int64

# Filter out rows with negative trip durations
df = df[df['trip_duration'] >= 0]
```

Filtering trip duration columns to greater than zero or non negative.

```
# Defining acceptable bounds for latitude and longitude
latitude_range = (40.0, 45.0)
longitude_range = (-75, -70.0)

df = df[df['pickup_latitude'].between(*latitude_range) &
          df['pickup_longitude'].between(*longitude_range) &
          df['dropoff_latitude'].between(*latitude_range) &
          df['dropoff_longitude'].between(*longitude_range)]
```

This code is useful for cleaning and preparing the data when dealing with geographical coordinates, ensuring that only data within the desired geographic region is retained in the DataFrame.

```
import datetime as dt
df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'])
df['dropoff_datetime'] = pd.to_datetime(df['dropoff_datetime'])
#assigning new columns to the dataframe such as weekday,
#month and pickup_hour which will help us to gain more insights from the data.

df['weekday'] = df['pickup_datetime'].dt.strftime('%A')
df['month'] = df.pickup_datetime.dt.month
df['weekday_num'] = df.pickup_datetime.dt.weekday
df['pickup_hour'] = df.pickup_datetime.dt.hour
```

data preprocessing and feature engineering for a DataFrame containing taxi trip records. It begins by converting the 'pickup_datetime' and 'dropoff_datetime' columns into datetime objects, which is crucial for time-based analysis. Subsequently, it creates new columns within the DataFrame to extract additional insights from the date and time data. These new columns include 'weekday,' which identifies the day of the week, 'month' to represent the month of the year, and 'pickup_hour' for the hour of the day when the trips occur. These derived features are instrumental in unraveling patterns and trends within the data, such as understanding which days of the week or months experience higher or lower trip volumes, as well as the distribution of trips across different hours of the day. Feature engineering of this kind enhances the dataset's richness, facilitating more in-depth analysis and modeling.

df.head()
Checking if the columns has been added for further analysis.

pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration	weekday	month	weekday_num	pickup_hour
40.74	-73.99	40.73	N	494	Friday	1	4	15
40.75	-73.99	40.76	N	445	Tuesday	5	1	16
40.75	-73.98	40.78	N	355	Friday	1	4	11
40.80	-73.94	40.80	N	932	Sunday	3	6	16
40.67	-73.99	40.67	N	276	Saturday	4	5	23

Checking for the datatypes if we need to process any categorical or string variables to float for further analysis.

```
df.dtypes
```

```
id                          object
vendor_id                    int64
pickup_datetime      datetime64[ns]
dropoff_datetime     datetime64[ns]
passenger_count              int64
pickup_longitude           float64
pickup_latitude            float64
dropoff_longitude          float64
dropoff_latitude           float64
store_and_fwd_flag          object
trip_duration                int64
weekday                     object
month                        int32
weekday_num                  int32
pickup_hour                  int32
dtype: object
```

```
df.value_counts()
```

```
id          vendor_id  pickup_datetime      dropoff_datetime     passenger_count  pickup_longitude  pickup_latitude  d
ropoff_longitude  dropoff_latitude  store_and_fwd_flag  trip_duration  weekday   month  weekday_num  pickup_hour
id0000001  2          2016-06-14 10:43:10  2016-06-14 11:01:35  1                -74.01            40.71            -
73.97             40.75              N                   1105           Tuesday   6      1            10           1
id2661279  1          2016-03-03 22:22:14  2016-03-03 22:49:38  1                -73.87            40.77            -
74.00             40.72              N                   1644           Thursday  3      3            22           1
id2662355  2          2016-01-29 21:22:22  2016-01-29 21:35:47  1                -74.00            40.74            -
73.97             40.75              N                   805            Friday    1      4            21           1
id2662326  2          2016-04-05 18:04:46  2016-04-05 18:41:15  1                -73.98            40.76            -
73.87             40.77              N                   2189           Tuesday   4      1            18           1
id2662313  1          2016-03-01 12:02:38  2016-03-01 12:08:30  1                -73.99            40.72            -
73.99             40.73              N                   352            Tuesday   3      1            12           1
```

#
Creating dummy variables for fuel_type
dummy_store_and_fwd_flag = pd.get_dummies(df['store_and_fwd_flag'], prefix='store_and_fwd_flag')

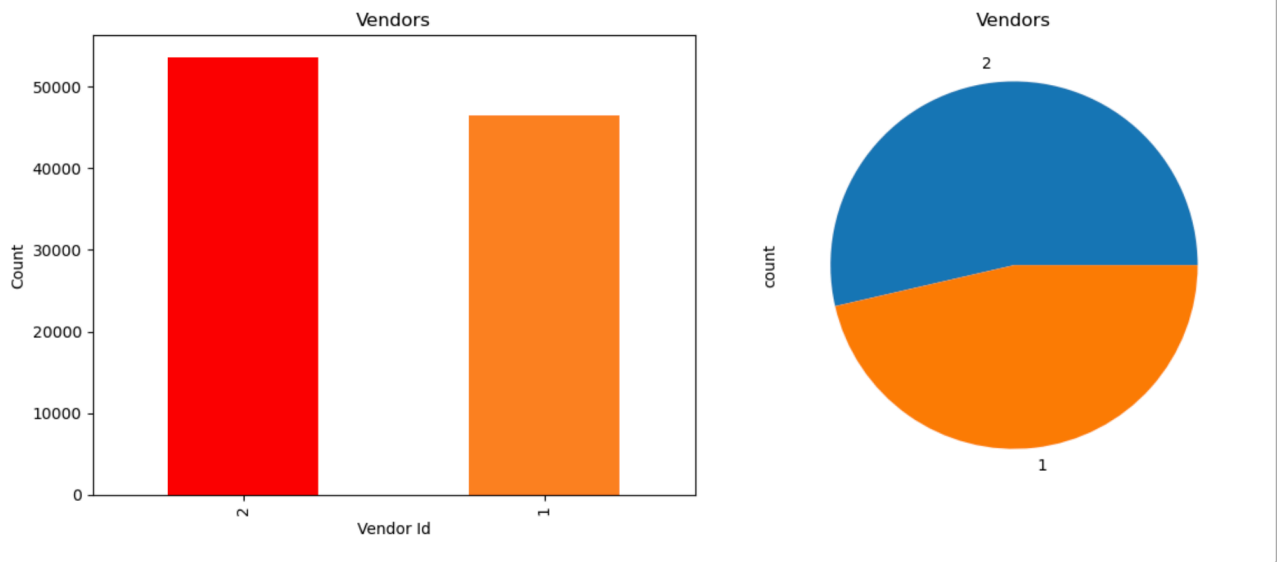
# Droping the first level of dummy variable
dummy_store_and_fwd_flag = dummy_store_and_fwd_flag.iloc[:, 1:]

# Replacing the original 'fuel_type' column with the dummy variables
df = pd.concat([df, dummy_store_and_fwd_flag], axis=1)

# Droping the original 'fuel_type' column
df = df.drop(columns=‘store_and_fwd_flag’)

process of transforming a categorical column, 'store_and_fwd_flag,' into a set of binary dummy variables in a DataFrame. This technique is integral in machine learning and regression analysis, as it allows for the representation of categorical data in a format suitable for modeling. The code starts by creating these dummy variables through the `pd.get_dummies()` function. It then drops one of the binary columns to prevent multicollinearity, a situation where one dummy variable can be perfectly predicted from the others. Subsequently, the original 'store_and_fwd_flag' column is replaced with the newly created binary variables, enhancing the dataset for analysis and modeling. Finally, the original column is discarded, as it's no longer required.

```
fig, axes = plt.subplots(nrows=1, ncols=2,figsize=(12,5))
ax = df['vendor_id'].value_counts().plot(kind='bar',title="Vendors",ax=axes[0],color = ('red',(1, 0.5, 0.13)))
df['vendor_id'].value_counts().plot(kind='pie',title="Vendors")
ax.set_ylabel("Count")
ax.set_xlabel("Vendor Id")
fig.tight_layout()
```



It does this by employing two different types of plots. On the left side, a bar plot is generated, showcasing the counts of each 'vendor_id.' The bars in this plot provide a clear comparison of the frequency of each vendor. On the right side, a pie chart is presented, offering a different perspective on the same data. The data indicates that there are only two vendors in this dataset, and they share a nearly equal number of trips, as evident from the similar heights of the bars in the bar plot and the nearly equal-sized slices in the pie chart. Interestingly, while the number of trips is distributed evenly between the two vendors, Vendor 2 appears to be more prominent among the population. This observation is drawn from the fact that Vendor 2 occupies a larger portion of the pie chart, reflecting a higher market share or popularity compared to Vendor 1. It's a valuable insight for understanding the competitive landscape and market dynamics in the context of these taxi trips.

```
# Seting the display format for floating-point numbers
pd.options.display.float_format = '{:.2f}'.format

# Counting the values in the 'passenger_count' column
passenger_count_counts = df['passenger_count'].value_counts()

# Displaying the counts
print(passenger_count_counts)
```

```
passenger_count
1    70891
2    14477
5     5285
3     4113
6     3307
4     1923
0        1
7        1
Name: count, dtype: int64
```

```
# Filtering the DataFrame to exclude rows with 0, 6, and 7 passengers
df = df[(df['passenger_count'] > 0) & (df['passenger_count'] < 6)]
```

To check the passenger count in each taxi and we observe that there are some outliers in 6,0,7 because in each car in new york taxi there will be max of 5 passenger if the baby is seated on top of passenger lap and there should not be 0 passengers because there trip might be canceled after booked or there must be some errors in the data .

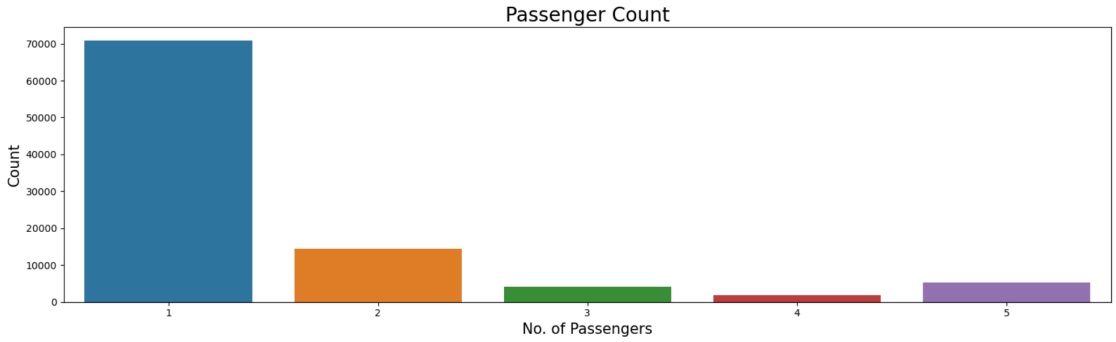
Here I have removed those outliers and and displayed the data so the data would be more useful and insightful

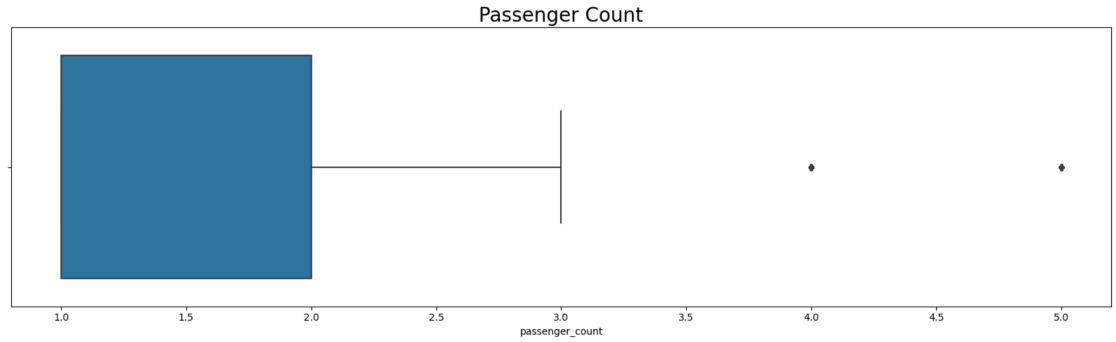
```
# Seting the display format for floating-point numbers
pd.options.display.float_format = '{:.2f}'.format

# Counting the values in the 'passenger_count' column
passenger_count_counts = df['passenger_count'].value_counts()

# Displaying the counts after removing outliers
print("After removing outliers from -", passenger_count_counts)
```

```
After removing outliers from - passenger_count
1    70891
2    14477
5     5285
3     4113
4     1923
Name: count, dtype: int64
```

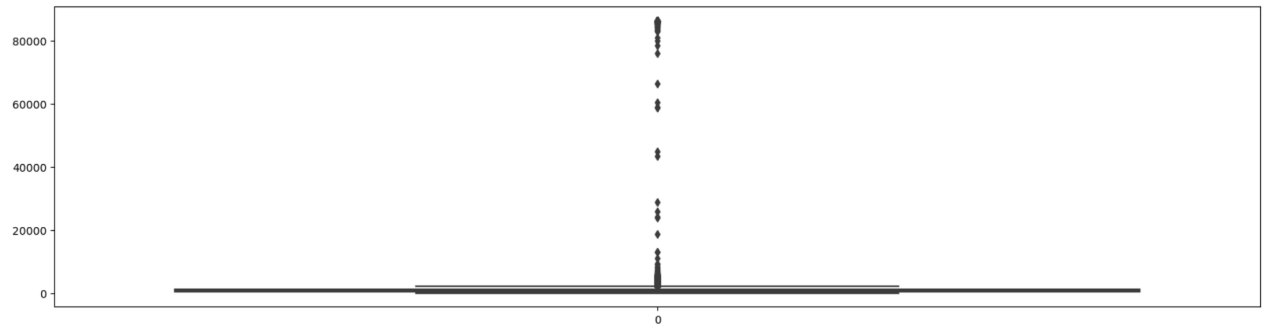
```
import matplotlib.pyplot as plt
import seaborn as sns

# Bar plot for passenger count
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(16, 5))
bar = sns.countplot(data=df, x='passenger_count')
bar.set_ylabel("Count", fontsize=15)
bar.set_xlabel("No. of Passengers", fontsize=15)
bar.set_title('Passenger Count', fontsize=20)
fig.tight_layout()

# Box plot for passenger count
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(16, 5))
box = sns.boxplot(data=df, x='passenger_count', orient='h')
box.set_title('Passenger Count', fontsize=20)
fig.tight_layout()

# Show the plots
plt.show()
```

You can see that we removed the outliers and Most of the passengers are 1.
The first plot takes the form of a bar chart, illustrating the frequency of trips with varying passenger counts. Each bar in the plot corresponds to a specific count of passengers, and the height of the bars represents the number of trips falling into each category. This visualization offers a clear perspective on how often different passenger counts are encountered in the dataset, helping us understand the most common scenarios.
The second plot, a box plot, offers a complementary viewpoint. By presenting key statistical measures like the median, quartiles, and potential outliers, the box plot provides insights into the distribution of passenger counts. It allows us to identify whether there are any unusual patterns or extreme values in the data. Both visualizations serve as valuable tools for gaining a comprehensive understanding of the dataset's passenger count distribution






```
plt.figure(figsize = (20,5))
sns.boxplot(df.trip_duration)
plt.show()
```



We can observe that no trip will be more than 20000 secs so we can consider them as outliers.

Here I have removed all the outliers more than 20000 secs.

# Filter out rows with negative trip durations
df = df[(df['trip_duration'] >= 0) & (df[‘trip_duration’]<=20000)]

Converted the data into bins and checked the observations,

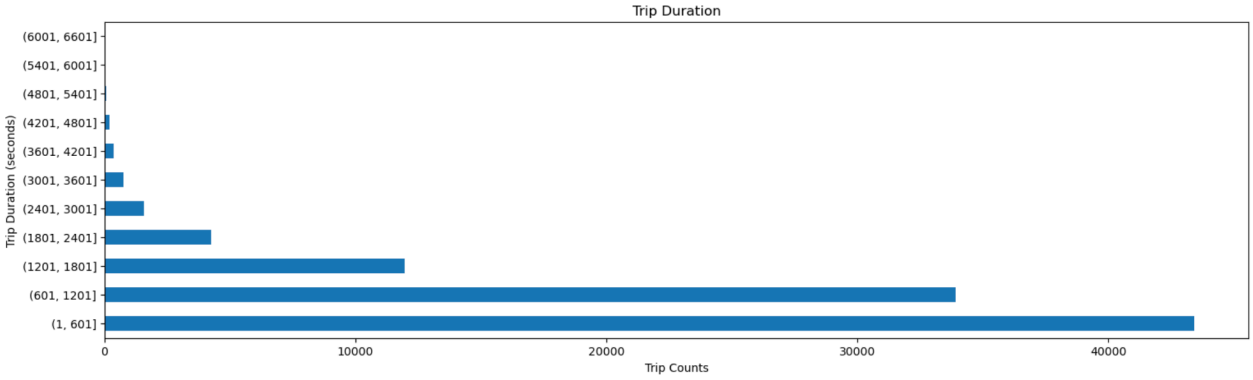
- All these trips are either taken on Tuesday's in 1st month or Saturday's in 2nd month. There might be some relation with the weekday, pickup location, month and the passenger.
- But they fail our purpose of correct prediction and bring inconsistencies in the algorithm calculation.

```
bin_edges = np.arange(0, df['trip_duration'].max(), 3600)

# Grouping and counting trips based on trip duration bins
trip_counts = df['trip_duration'].groupby(pd.cut(df['trip_duration'], bin_edges)).count()

# Printing the trip counts
print(trip_counts)
```

```
trip_duration
(0, 3600]          95867
(3600, 7200]         675
(7200, 10800]          6
(10800, 14400]         3
(14400, 18000]         0
Name: trip_duration, dtype: int64
```

```
df.trip_duration.groupby(pd.cut(df.trip_duration, np.arange(1,7200,600))).count().plot(kind='barh',figsize = (18,5))
plt.title('Trip Duration')
plt.xlabel('Trip Counts')
plt.ylabel('Trip Duration (seconds)')
plt.show()
```

```
/var/folders/wz/dvpqqs715hs1594hzt3ncl540000gn/T/ipykernel_22882/919114996.py:1: FutureWarning: The default of observ
ed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain curre
nt behavior or observed=True to adopt the future default and silence this warning.
  df.trip_duration.groupby(pd.cut(df.trip_duration, np.arange(1,7200,600))).count().plot(kind='barh',figsize = (18,
5))
```

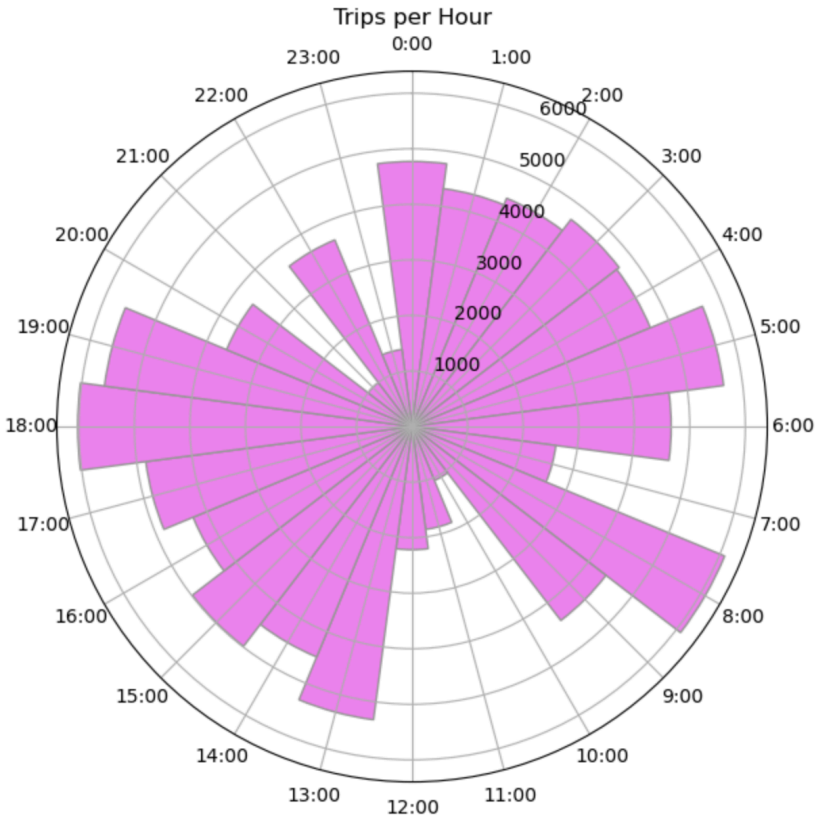


We can observe that most of the trips took 0 - 30 mins to complete.

```
def clock(ax, radii, title, color):
    N = 24
    bottom = 2

    # Create theta for 24 hours
    theta = np.linspace(0.0, 2 * np.pi, N, endpoint=False)

    # Width of each bin on the plot
    width = (2 * np.pi) / N

    bars = ax.bar(theta, radii, width=width, bottom=bottom, color=color, edgecolor="#999999")

    # Set the label to go clockwise and start from the top
    ax.set_theta_zero_location("N")
    # Set the direction to be clockwise
    ax.set_theta_direction(-1)

    # Set the labels
    ax.set_xticks(theta)
    ticks = ["{}:00".format(x) for x in range(24)]
    ax.set_xticklabels(ticks)
    ax.set_title(title)

plt.figure(figsize=(15, 15))
ax = plt.subplot(2, 2, 1, polar=True)

# Calculate the number of trips per hour and convert it to an array
radii = np.array(df['pickup_hour'].value_counts(sort=False).tolist(), dtype="int64")

title = "Trips per Hour"
custom_color = 'violet'

clock(ax, radii, title, custom_color)

plt.show()
```

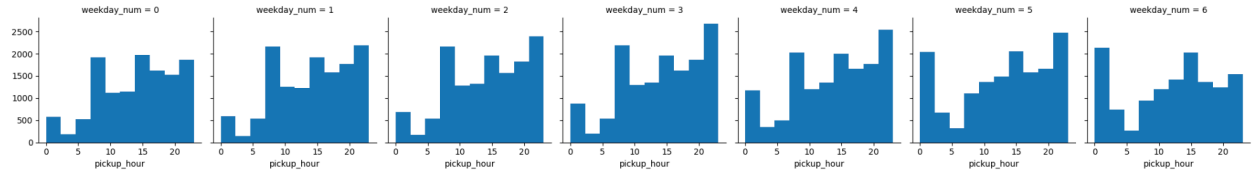
Now It's inline with the general trend of taxi pickups which starts increasing from 8AM in the morning and then declines from around 9 AM. Again starts from 1 pm and till 7 pm it goes good and again decreases around 9pm and again picks up at 12 am till the midnight 4 am. There is no unusual behaviour here.

- The number of pickup is maximum at 5-8 am.




```
n = sns.FacetGrid(df, col='weekday_num')
n.map(plt.hist, 'pickup_hour')
plt.show()
```

```
/Applications/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is
deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Applications/anaconda3/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has chan
ged to tight
  self._figure.tight_layout(*args, **kwargs)
```

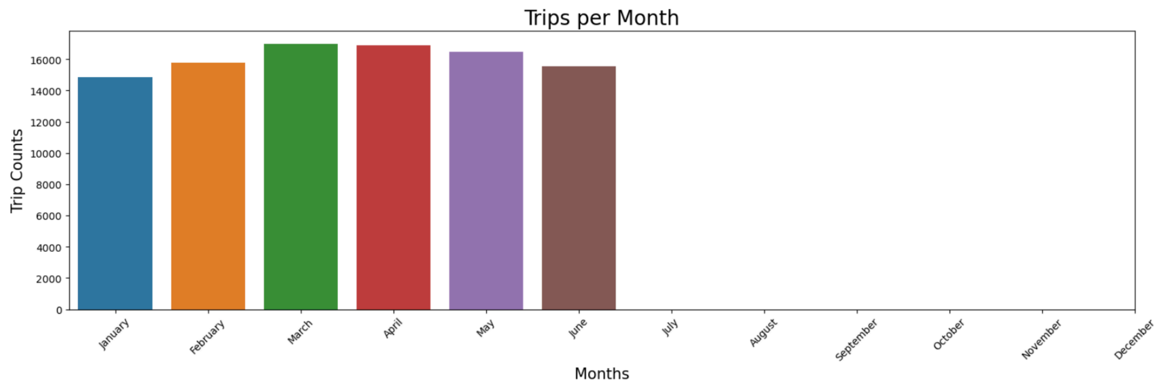


An intriguing pattern emerges when analyzing taxi pickups over the weekend. Late-night pickups exhibit a notable increase, possibly attributed to a surge in outstation rides or late-night leisure activities nearby. In contrast, the early morning hours, particularly before 5 AM, experience an uptick in pickups during the weekend compared to the dwindling demand during typical office hours, which begins after 7 AM, indicating a shift in the travel behavior. Interestingly, taxi pickups maintain a consistent frequency throughout the week, with a distinct peak at 3 PM, suggesting a steady demand for taxi services at this time.

There is consistent trips booking over the months and it is slightly more at march and April mnths.

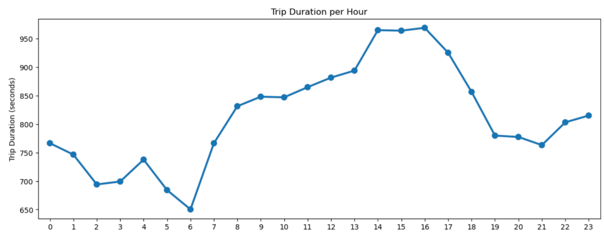
```
plt.figure(figsize=(19, 5))
count_plot = sns.countplot(data=df, x='month')
plt.ylabel('Trip Counts', fontsize=15)
plt.xlabel('Months', fontsize=15)
plt.title('Trips per Month', fontsize=20)

# Specify the positions and labels for the x-axis ticks
month_pos = list(range(12))
monthly_labels = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'No

count_plot.set_xticks(month_pos)
count_plot.set_xticklabels(monthly_labels, rotation=45)

plt.show()
```

```
/Applications/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is
deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Applications/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is
deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Applications/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is
deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```




```
plt.figure(figsize=(14, 5))

# Group by pickup hour and calculate the mean trip duration
gr_1 = df.groupby('pickup_hour')['trip_duration'].mean().reset_index()

# Create a point plot
point_plot = sns.pointplot(x='pickup_hour', y='trip_duration', data=gr_1)
point_plot.set_ylabel('Trip Duration (seconds)')
point_plot.set_xlabel('Pickup Hour')
point_plot.set_title('Trip Duration per Hour')

plt.show()
```

```
/Applications/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is
deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Applications/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is
deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Applications/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is
deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



The average trip duration exhibits interesting patterns throughout the day. At 6 AM, when road traffic is minimal, the average trip duration reaches its lowest point. In contrast, trip durations tend to peak at approximately 3 PM, coinciding with the bustling city streets during the afternoon rush. Notably, average trip durations show remarkable consistency during the early morning hours before 6 AM and the late evening hours after 6 PM, reflecting a comparable travel experience during these time frames.

Similarly I have done for Weekday and monthly, we observed that "The distribution of trip durations spans a scale of 0-900 minutes across the week, with minimal differences in duration times between days. However, it's noteworthy that Thursdays stand out with the longest average trip duration among all weekdays.

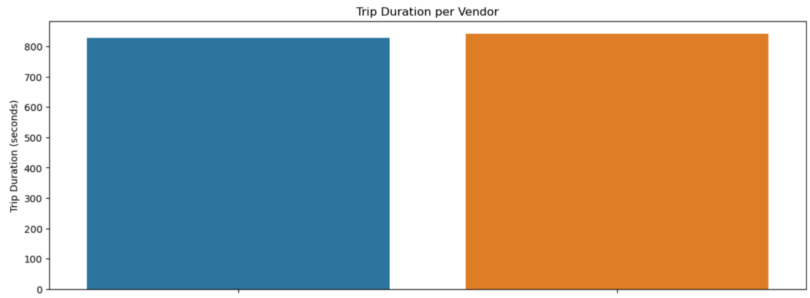
Furthermore, there's a discernible upward trend in average trip duration with each passing month, though the differences between months are relatively modest. This gradual increase extends over a six-month period, with the lowest durations occurring in February as winter conditions start to wane.

These patterns in trip duration may be influenced by seasonal factors like weather, specifically wind and rain. Notably, May is historically one of the wettest months in NYC, and this aligns with our visualizations. Rainy conditions often lead to longer travel times due to increased traffic congestion, making April, May, and June months where trip durations naturally tend to be extende

We can see that there are two vendors almost having the same trip duration.

```
# Group by vendor and calculate the mean trip duration
gr_4 = df.groupby('vendor_id')['trip_duration'].mean().reset_index()

plt.figure(figsize=(14, 5))

# Create a bar plot
bar_plot = sns.barplot(x='vendor_id', y='trip_duration', data=gr_4)
bar_plot.set_ylabel('Trip Duration (seconds)')
bar_plot.set_xlabel('Vendor')
bar_plot.set_title('Trip Duration per Vendor')

plt.show()
```

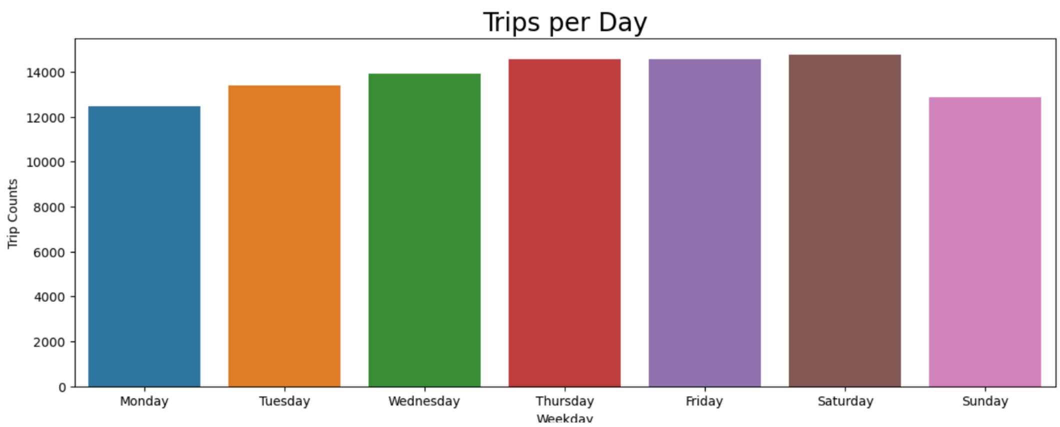
```
/Applications/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is
deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Applications/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is
deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Applications/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is
deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



We can see that Thursday to Saturday it is high demand since it is weekend and many people will go to pubs and restaurants and book cab because they can't drive there own cars because drink and drive is not allowed. It is natural to have high demand in the weekends.

```
plt.figure(figsize=(14, 5))
trip_count_plot = sns.countplot(data=df, x='weekday_num')
plt.xlabel('Weekday')
plt.ylabel('Trip Counts')
plt.title('Trips per Day', fontsize=20)

# Set the x-axis labels to display weekdays
trip_count_plot.set_xticklabels(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])

plt.show()
```

```
/Applications/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is
deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Applications/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is
deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Applications/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is
deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



```
from sklearn.preprocessing import LabelEncoder

# Create a LabelEncoder object
label_encoder = LabelEncoder()

# Apply label encoding to the 'vendor_id' column
df['id'] = label_encoder.fit_transform(df['id'])
```

```
df.dtypes
```

```
id                              int64
vendor_id                       int64
pickup_datetime        datetime64[ns]
dropoff_datetime       datetime64[ns]
passenger_count                 int64
pickup_longitude              float64
pickup_latitude               float64
dropoff_longitude             float64
dropoff_latitude              float64
trip_duration                   int64
weekday                        object
month                           int32
weekday_num                     int32
pickup_hour                     int32
store_and_fwd_flag_Y             bool
dtype: object
```

We need to convert string variables to numeric variables and we use One hot encoding (label encoder) to convert because to build the model we need to have converted variables which are very important to be converted to numerical data type.

```
from sklearn.preprocessing import MinMaxScaler

# Create a MinMaxScaler object
scaler = MinMaxScaler()

# Define the columns to be normalized
columns_to_normalize = ['pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude','trip_duration']

# Apply normalization to the selected columns
df[columns_to_normalize] = scaler.fit_transform(df[columns_to_normalize])
```

```
# Select the numerical columns for outlier detection
num_columns = ['pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude', 'trip_duration']

# Calculate the IQR for each numerical column
Q1 = df[num_columns].quantile(0.25)
Q3 = df[num_columns].quantile(0.75)
IQR = Q3 - Q1

# Define a factor to scale the IQR (you can adjust this factor as needed)
IQR_factor = 1.5

# Define a lower and upper bound for outlier detection
lower_bound = Q1 - IQR_factor * IQR
upper_bound = Q3 + IQR_factor * IQR

# Find rows with outliers
outliers = ((df[num_columns] < lower_bound) | (df[num_columns] > upper_bound)).any(axis=1)

# Remove rows with outliers
df_clean = df[~outliers]

df = df_clean

# Display the cleaned DataFrame
df.head()
```

We have to normalise the variables scikit-learn's Min-Max scaling to transform specific columns in the DataFrame. This technique standardizes the column values to the [0, 1] range, facilitating the equitable contribution of features to machine learning models. The code begins by creating a `MinMaxScaler` object, then designates columns for normalization, which include 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude', and 'trip_duration'. These columns are subsequently rescaled using `scaler.fit_transform()`, replacing the original values with the normalized counterparts. This preprocessing ensures features are on a consistent scale, enhancing model performance.

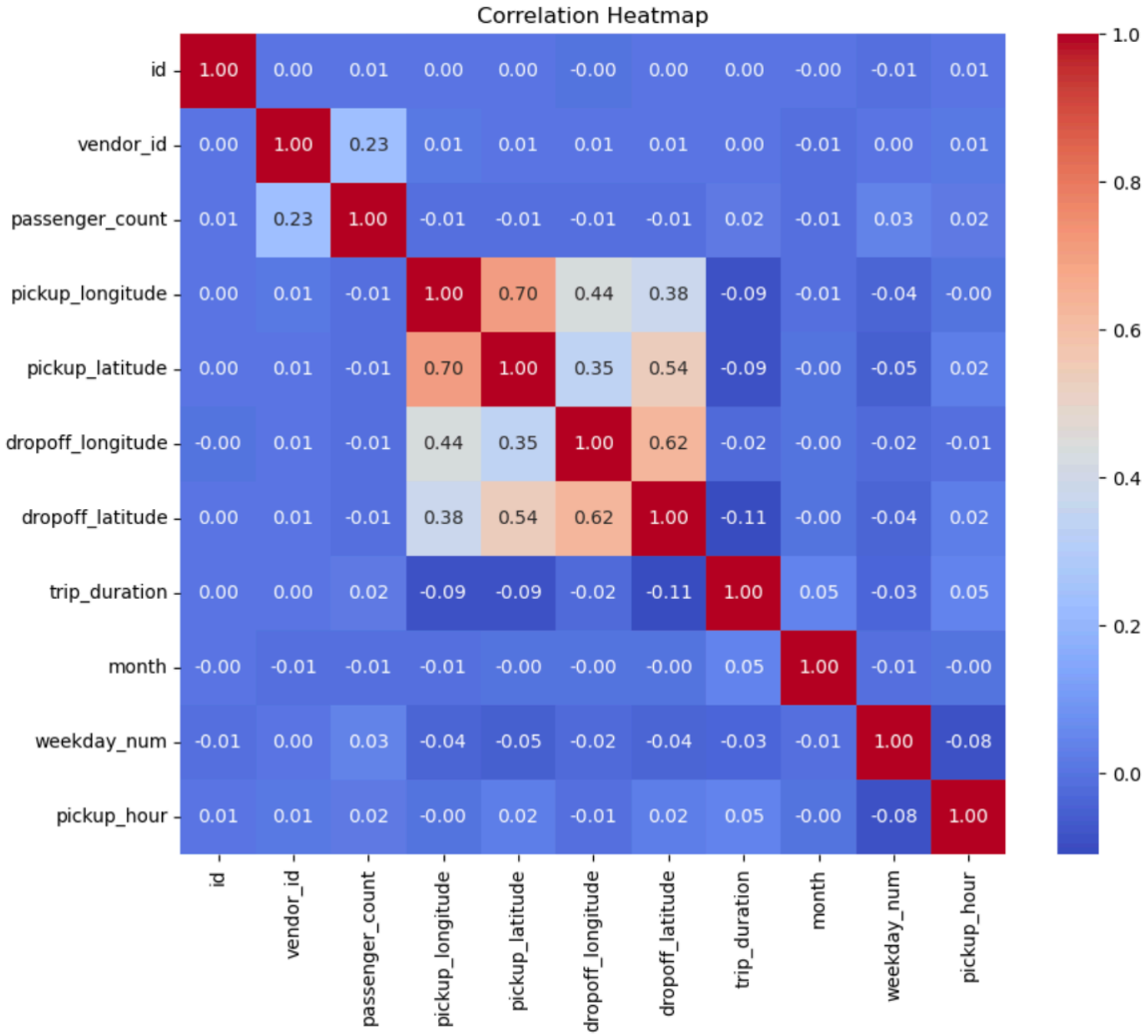
df.shape

```
(82853, 15)
```

Checking the shape of the dataset after removing outliers

Checking the relationship between dependent and independent variables using correlation heat map.

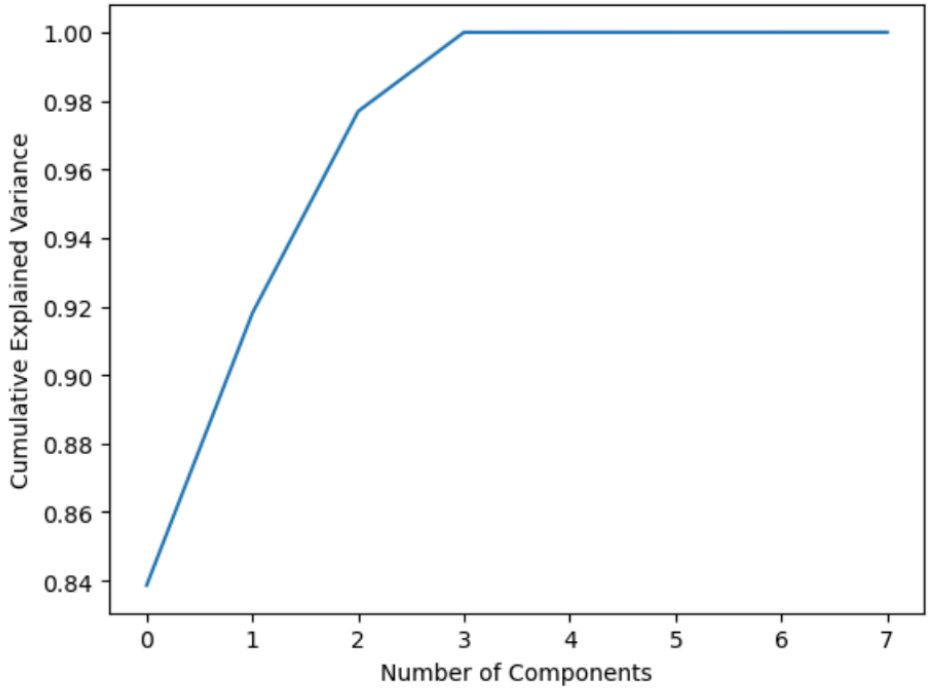
```
# Select only the numeric columns in your DataFrame
num_columns = df.select_dtypes(include=['number'])

# Calculate the correlation matrix for the numeric columns
corr_matrix = num_columns.corr()

# Create a heatmap to visualize the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```



Splitting the data into training and test sets and fit PCA (Principal component analysis), to find and check which independent variables are good and can be used to build the model.

```
# Split the data into training and testing sets
X = df[predictors]
y = df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=8765)
```

```
from sklearn.decomposition import PCA
# Fit PCA to your training data
pca = PCA().fit(X_train)

# Calculate the cumulative explained variance
cumulative_explained_variance = np.cumsum(pca.explained_variance_ratio_)

# Create a plot to visualize the cumulative explained variance
plt.plot(cumulative_explained_variance)
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Explained Variance")
plt.show()
```



Now I built the model and splitting the data and find R2 score. And finding the Model coefficients

Model Coefficients:
{'passenger_count': 0.0003257400243385159, 'month': 0.0005644986175170018,
'weekday_num': -0.0003937991612607301, 'pickup_hour': 0.0001670409664216176,
'pickup_longitude': -0.16684956551708177, 'pickup_latitude': 0.008798167051330123,
'dropoff_longitude': 0.17451075955459278, 'dropoff_latitude': -0.16812825272091725}
Mean Squared Error: 0.0004553457394064483
R-squared (R2) Score: 0.026839641805014747

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


# Select the predictors (independent variables) and the target (dependent variable)
predictors = ['passenger_count', 'month', 'weekday_num', 'pickup_hour', 'pickup_longitude', 'pickup_latitude', 'dropoff
target = 'trip_duration'

# Split the data into training and testing sets
X = df[predictors]
y = df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=8765)

# Create a linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the model coefficients (effect of each predictor)
coefficients = dict(zip(predictors, model.coef_))
print("Model Coefficients:")
print(coefficients)

# Print evaluation metrics
print("Mean Squared Error:", mse)
print("R-squared (R2) Score:", r2)
```

```
Model Coefficients:
{'passenger_count': 0.0003257400243385159, 'month': 0.0005644986175170018, 'weekday_num': -0.0003937991612607301, 'pi
ckup_hour': 0.0001670409664216176, 'pickup_longitude': -0.16684956551708177, 'pickup_latitude': 0.008798167051330123,
'dropoff_longitude': 0.17451075955459278, 'dropoff_latitude': -0.16812825272091725}
Mean Squared Error: 0.0004553457394064483
R-squared (R2) Score: 0.026839641805014747
```

```
arr = np.cumsum(np.round(pca.explained_variance_ratio_,
decimals=4)*100)
list(zip(range(1,len(arr)), arr))
```

```
[(1, 83.87),
 (2, 91.81),
 (3, 97.7),
 (4, 100.0),
 (5, 100.0),
 (6, 100.0),
 (7, 100.0)]
```

Checking the variables Here we can see that 6 variables are sufficient for capturing atleast 92% of the variance in the training dataset. Hence we will use the same set of variables(i.e this model does not require the use of PCA).

To achieve our goal of predicting the duration of NYC taxi trips based on various input features, we require a suitable model. Given that our target variable consists of continuous values, we will employ regression techniques for this task. Regression models will enable us to make predictions about trip durations, leveraging the relationships between the input variables and the continuous outcome variable.
It is used to explain the relationship between one continuous dependent variable and two or more independent variables.

```
start_time = time.time()
lm_regression = LinearRegression()
lm_regression = lm_regression.fit(X_train, y_train)
end_time = time.time()
lm_time = (end_time - start_time)
print(f"Time taken to train linear regression model : {lm_time} seconds")
```

```
Time taken to train linear regression model : 0.04128599166870117 seconds
```

```
trips = lm_regression.predict(X_test)
```

```
lm_score = r2_score(y_test, trips)
print(lm_score)
```

```
0.026839641805014747
```

As the linear model has less r2 score so I tried random forest model. But seeing the score it is also the same

```
start_time = time.time()
rf_regression = RandomForestRegressor()
rf_regression = rf_regression.fit(X_train, y_train)
end_time = time.time()
rf_time = (end_time - start_time)
print(f"Time taken to train Random Forest model : {rf_time} seconds")
```

```
Time taken to train Random Forest model : 31.766072988510132 seconds
```

```
rf_score = r2_score(y_test, trips)
print("random forest r2 score", rf_score)
```

```
random forest r2 score 0.026839641805014747
```

```
from sklearn.tree import DecisionTreeRegressor

# Create and fit the Decision Tree regression model, measuring the time
start_time = time.time()
dt_regression = DecisionTreeRegressor().fit(X_train, y_train)
end_time = time.time()
dt_time = end_time - start_time

print(f"Time to train Decision Tree model: {dt_time:.2f} seconds")

# Predict on the test data
trips = dt_regression.predict(X_test)
```

```
Time to train Decision Tree model: 0.51 seconds
```

```
de_t_score = r2_score(y_test, trips)
print("Decision tree r2 score", de_t_score)
```

```
Decision tree r2 score 0.33673229969968743
```

As the r2 score is less in both linear and random forest regression I have tried Decision tree and this looks positive and Decision tree can be chosen as the best model because it have better R2 score and also executes quicker.

```
from sortedcontainers import SortedDict

# Initialize a SortedDict to store trip data sorted by trip duration
trip_sorted_data = SortedDict()

# Populate the SortedDict with the data from the DataFrame
for index, row in df.iterrows():
    trip_id = row['id']
    trip_duration = row['trip_duration']
    trip_sorted_data[trip_duration] = (trip_id, row)

# Adding a new trip (as an example)
new_trip_id = 'new_trip_id'
new_trip_duration = 500
new_trip_data = {
    'vendor_id': 'new_vendor',
    'pickup_datetime': '2023-10-28 10:00:00',
    'dropoff_datetime': '2023-10-28 10:30:00',
    'passenger_count': 3,
    'pickup_longitude': -73.9895,
    'pickup_latitude': 40.7523,
    'dropoff_longitude': -73.9876,
    'dropoff_latitude': 40.7612,
    'store_and_fwd_flag': 'N',
}
trip_data_sorted[new_trip_duration] = (new_trip_id, new_trip_data)

# View the data in ascending order of trip duration with formatted durations
for duration, (trip_id, data) in trip_sorted_data.items():
    formatted_duration = f"{duration:.2f}"  # Format the duration with 2 decimal places
    print(f"Trip ID: {trip_id}, Duration: {formatted_duration} seconds")
```

```
Trip ID: 70370, Duration: 0.08 seconds
Trip ID: 37540, Duration: 0.08 seconds
Trip ID: 65071, Duration: 0.08 seconds
Trip ID: 33946, Duration: 0.08 seconds
Trip ID: 20203, Duration: 0.08 seconds
Trip ID: 92617, Duration: 0.08 seconds
Trip ID: 54631, Duration: 0.08 seconds
Trip ID: 80864, Duration: 0.08 seconds
Trip ID: 15193, Duration: 0.08 seconds
Trip ID: 44712, Duration: 0.08 seconds
Trip ID: 24159, Duration: 0.08 seconds
Trip ID: 46187, Duration: 0.08 seconds
```

Justify the choice of data structure (from arrays, stacks, queues, linked lists, and hash tables) to store the data for each of the following scenarios:

- The dataset should be sorted and viewed in ascending order of trip duration. New data is added to the dataset frequently, where these new trips should show up at the end of the sorted list.

- Scenario 1: Sorting and Viewing Data in Ascending Order of Trip Duration
  - Data Structure: Array
  - Justification: Arrays efficiently store and sort numerical data like trip durations. They enable quick access to data once sorted, ideal for viewing it in ascending order.
- Scenario 2: Quickly Filtering Trips by Passenger Phone Numbers
  - Data Structure: Hash Table (Dictionary)
  - Justification: Hash tables excel in fast data retrieval by keys. Using passenger phone numbers as keys and trip data as values allows lightning-fast filtering, making them the ideal choice for this task.

```
trip_durations = df['trip_duration'].values  # Extracting trip durations as a NumPy array

# Sorting the trip durations in ascending order
sorted_trip_durations = sorted(trip_durations)

# Convert the sorted array back to a DataFrame if needed
sorted_df = pd.DataFrame({'trip_duration': sorted_trip_durations})

# Display the sorted DataFrame
print(sorted_df)
```

```
       trip_duration
0               0.00
1               0.00
2               0.00
3               0.00
4               0.00
...              ...
82848           0.11
82849           0.11
82850           0.11
82851           0.11
82852           0.11

[82853 rows x 1 columns]
```

Sorted the trip duration columns and added the column to the data frame. And it is added as NAN since those values are too small it is showing as NAN values.

Installed Faker to create random fake phone numbers and updated the column

```
pip install faker
```

```
Requirement already satisfied: faker in /Applications/anaconda3/lib/python3.11/site-
Requirement already satisfied: python-dateutil>=2.4 in /Applications/anaconda3/lib/p
er) (2.8.2)
Requirement already satisfied: six>=1.5 in /Applications/anaconda3/lib/python3.11/si
>=2.4->faker) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

```
from faker import Faker

# Initialize Faker to generate random data
var_fake = Faker()

# Replace 'df' with your actual DataFrame

# Generate random phone numbers for each row in the DataFrame
phone_numbers_random = [var_fake.phone_number() for _ in range(len(df))]

# Add the 'phone_number' column to the DataFrame
df['passenger_phone_number'] = phone_numbers_random

# Display the updated DataFrame with phone numbers
print(df)
```

Here we have added the last column with random numbers.

```
700780              1            10

        passenger_phone_number
548214   +1-566-312-2176x04832
243720             694-577-2509
1118153    +1-364-998-6956x720
950698        536.846.1319x145
489339   +1-478-631-1010x35130
...                        ...
951906           (954)350-2499
1072937     715.911.2811x96093
1154625  001-379-645-8180x2419
339012        886.997.5124x5541
700780       (881)555-9046x6067

[82853 rows x 17 columns]
```

```python
# Geting the unique phone numbers in the DataFrame
unique_phone_numbers = df['passenger_phone_number'].unique()

chosen_phone_numbers = unique_phone_numbers[:3]

# Filtering trips for each chosen phone number
for chosen_phone_number in chosen_phone_numbers:
    filtered_trips = df[df['passenger_phone_number'] == chosen_phone_number]

    # Checking if any trips were found for the chosen phone number
    if not filtered_trips.empty:
        print(f"Trips made by passenger with phone number {chosen_phone_number}:")
        print(filtered_trips)
    else:
        print(f"No trips found for passenger with phone number {chosen_phone_number}")
```

```
Trips made by passenger with phone number +1-566-312-2176x04832:
           id  vendor_id     pickup_datetime    dropoff_datetime  \
548214  50989          1 2016-01-15 15:17:28 2016-01-15 15:25:42

        passenger_count  pickup_longitude  pickup_latitude  dropoff_longitude  \
548214                1              0.49             0.63               0.47

        dropoff_latitude  trip_duration weekday  month  weekday_num  \
548214              0.46           0.03  Friday      1            4

        pickup_hour  store_and_fwd_flag_Y  sorted_trip_duration  \
548214           15                 False                   NaN

       passenger_phone_number
548214  +1-566-312-2176x04832
Trips made by passenger with phone number 694-577-2509:
           id  vendor_id     pickup_datetime    dropoff_datetime  \
243720  47101          2 2016-05-10 16:19:36 2016-05-10 16:27:01

        passenger_count  pickup_longitude  pickup_latitude  dropoff_longitude  \
243720                2              0.50             0.64               0.47

        dropoff_latitude  trip_duration  weekday  month  weekday_num  \
243720              0.48           0.02  Tuesday      5            1

        pickup_hour  store_and_fwd_flag_Y  sorted_trip_duration  \
243720           16                 False                   NaN
```

I have displayed 3 rows with phone numbers data.