```
# This Python 3 environment comes with many helpful analytics
libraries installed
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns# data processing, CSV file I/O (e.g.
pd.read_csv)

# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/)
that gets preserved as output when you create a version using "Save &
Run All"
# You can also write temporary files to /kaggle/temp/, but they won't
be saved outside of the current session
```

```
/kaggle/input/taxi-fare-guru-total-amount-prediction-challenge/
sample.csv.csv
/kaggle/input/taxi-fare-guru-total-amount-prediction-challenge/train.c
sv
/kaggle/input/taxi-fare-guru-total-amount-prediction-challenge/test.cs
v
```

1. Data Loading

```
data=pd.read_csv('/kaggle/input/taxi-fare-guru-total-amount-
prediction-challenge/train.csv')
data_t=pd.read_csv('/kaggle/input/taxi-fare-guru-total-amount-
prediction-challenge/test.csv')

data.head()
```

```
   VendorID tpep_pickup_datetime tpep_dropoff_datetime
passenger_count  \
0         1  2023-06-28 17:20:21   2023-06-28 16:34:45
1.0
1         0  2023-06-29 23:05:01   2023-06-29 22:01:35
1.0
```

```
2         1  2023-06-30 10:19:31   2023-06-30 11:13:10
1.0
3         0  2023-06-29 13:23:09   2023-06-29 14:20:01
1.0
4         1  2023-06-29 22:03:32   2023-06-29 22:22:22
3.0

   trip_distance  RatecodeID store_and_fwd_flag  PULocationID
DOLocationID  \
0           2.14         1.0                  N           120
9
1           2.70         1.0                  N            15
215
2           1.15         1.0                  N           167
223
3           0.40         1.0                  N           128
239
4           1.10         1.0                  N           203
52

  payment_type  extra  tip_amount  tolls_amount  improvement_surcharge
\
0  Credit Card    2.5    7.165589           0.0                    1.0

1  Credit Card    3.5    6.067401           0.0                    1.0

2  Credit Card    0.0    4.111547           0.0                    1.0

3  Credit Card    2.5    6.411079           0.0                    1.0

4  Credit Card    1.0    4.769377           0.0                    1.0


   total_amount  congestion_surcharge  Airport_fee
0         20.64                   2.5          0.0
1         25.55                   2.5          0.0
2         17.64                   2.5          0.0
3         12.80                   2.5          0.0
4         18.00                   2.5          0.0

feature_list = data.columns[:-1].values
label = data.columns[-3]
print("Feature List:", feature_list)
print("Label:", label)

Feature List: ['VendorID' 'tpep_pickup_datetime'
'tpep_dropoff_datetime'
 'passenger_count' 'trip_distance' 'RatecodeID' 'store_and_fwd_flag'
 'PULocationID' 'DOLocationID' 'payment_type' 'extra' 'tip_amount'
 'tolls_amount' 'improvement_surcharge' 'total_amount'
```

```
 'congestion_surcharge']
Label: total_amount
```

2. Exploratory Data Analysis

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

# Let us summarize our data
data.describe()
```

```
            VendorID  passenger_count  trip_distance     RatecodeID  \
count  175000.000000    168923.000000  175000.000000  168923.000000
mean        0.728377         1.357678       5.145930       1.518307
std         0.445606         0.891283     394.971052       6.514678
min         0.000000         0.000000       0.000000       1.000000
25%         0.000000         1.000000       1.080000       1.000000
50%         1.000000         1.000000       1.840000       1.000000
75%         1.000000         1.000000       3.610000       1.000000
max         2.000000         9.000000  135182.060000      99.000000

        PULocationID   DOLocationID          extra     tip_amount  \
count  175000.000000  175000.000000  175000.000000  175000.000000
mean      132.710349     132.701429       1.932143       6.127497
std        76.148799      76.192493       1.948497       4.610834
min         1.000000       1.000000      -7.500000       0.000079
25%        67.000000      67.000000       0.000000       3.473321
50%       133.000000     133.000000       1.000000       5.286217
75%       199.000000     199.000000       2.500000       7.502746
max       264.000000     264.000000      11.750000     484.876151

        tolls_amount  improvement_surcharge   total_amount  \
count  175000.000000          175000.000000  175000.000000
mean        0.646816               0.979689      29.633901
std         2.328274               0.198775      25.425206
min       -29.300000              -1.000000    -576.750000
25%         0.000000               1.000000      16.300000
50%         0.000000               1.000000      21.450000
75%         0.000000               1.000000      31.800000
max        80.000000               1.000000     587.250000

       congestion_surcharge    Airport_fee
count         168923.000000  168923.000000
mean               2.246971       0.158825
std                0.819216       0.511968
min               -2.500000      -1.750000
25%                2.500000       0.000000
50%                2.500000       0.000000
```

```
75%                   2.500000       0.000000
max                   2.500000       1.750000
```

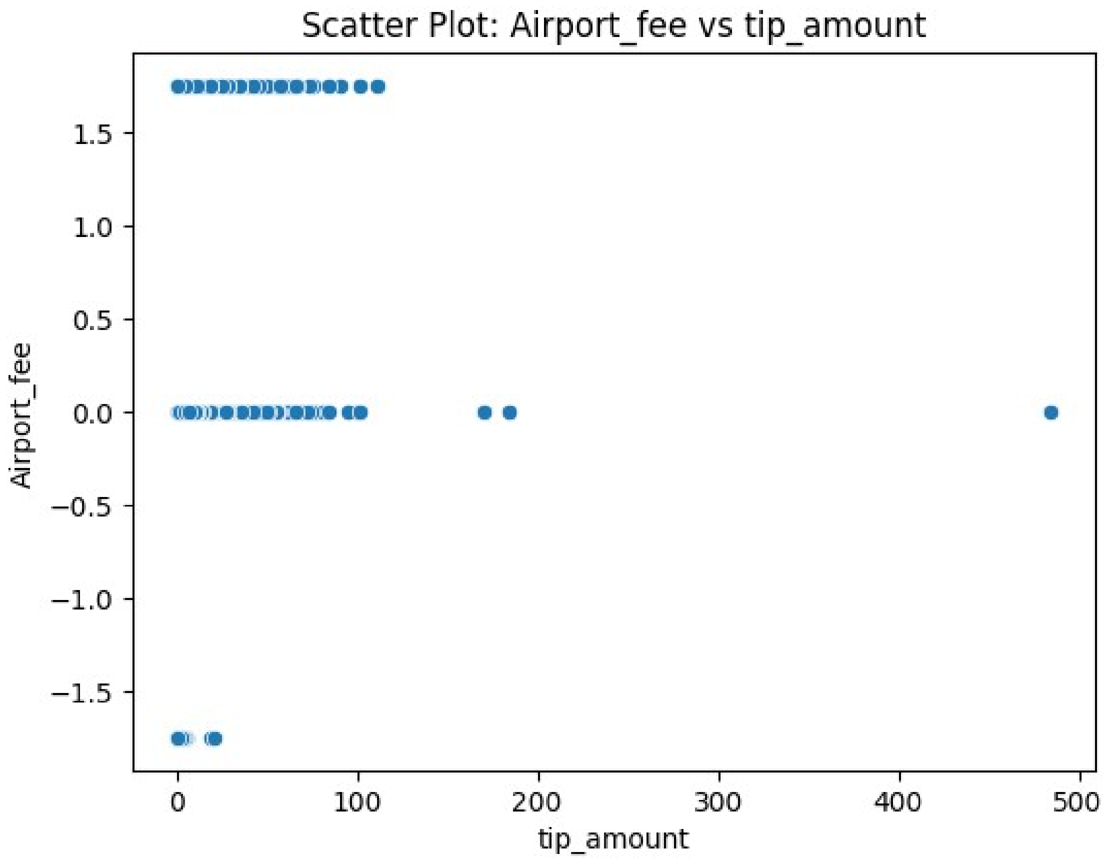
```python
# Finding out the missing values in our data
missing_value_counts = data.isnull().sum()
missing_value_counts
```

```
VendorID                     0
tpep_pickup_datetime         0
tpep_dropoff_datetime        0
passenger_count           6077
trip_distance                0
RatecodeID                6077
store_and_fwd_flag        6077
PULocationID                 0
DOLocationID                 0
payment_type                 0
extra                        0
tip_amount                   0
tolls_amount                 0
improvement_surcharge        0
total_amount                 0
congestion_surcharge      6077
Airport_fee               6077
dtype: int64
```

```python
# counts the occurrences of unique values in the Airport_fee column.
unique_values = data['Airport_fee'].value_counts()
unique_values
```

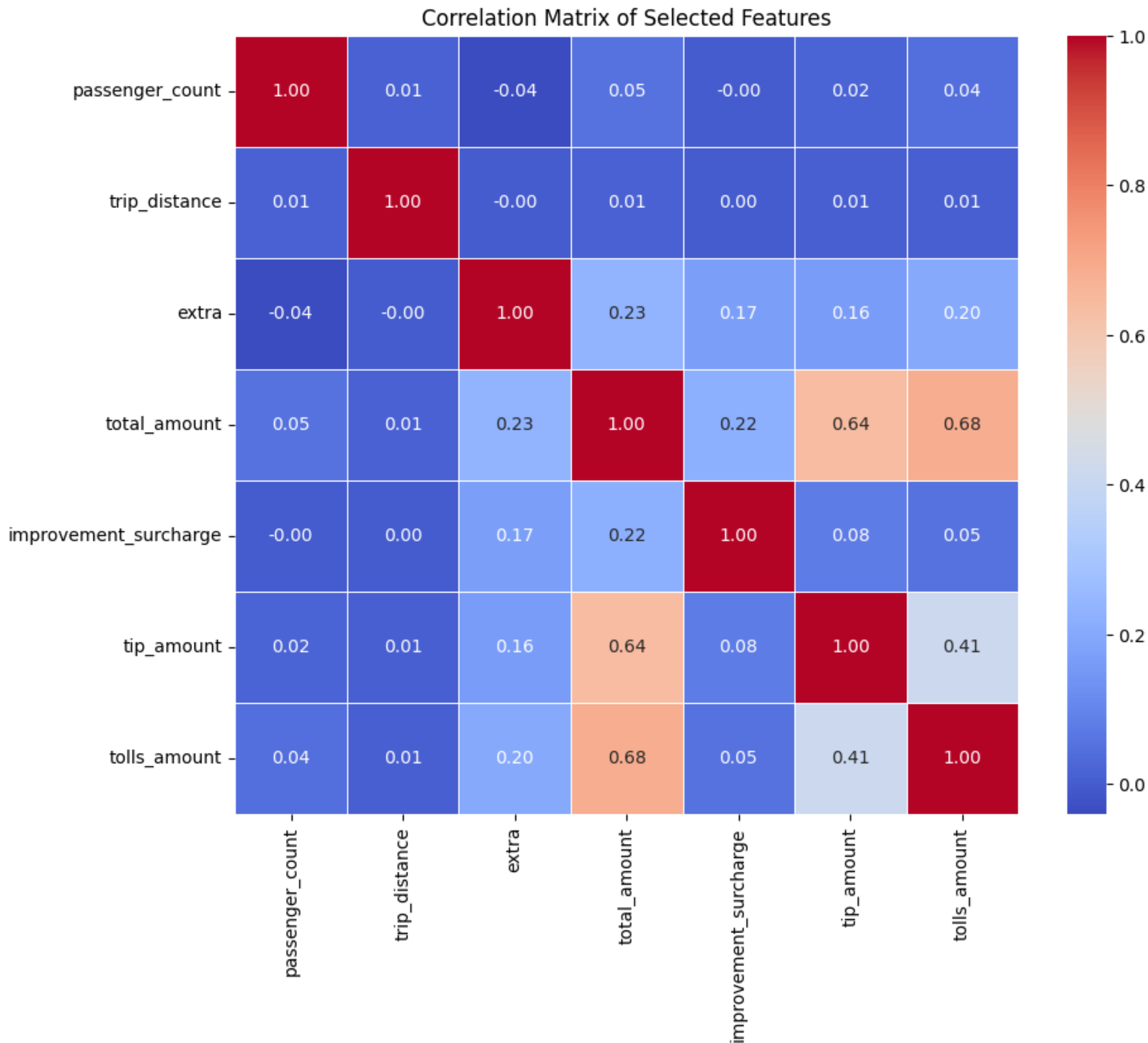
```
Airport_fee
 0.00    153074
 1.75     15590
-1.75       259
Name: count, dtype: int64
```

```python
# Scatter plot between airport fee and tip_amount
sns.scatterplot(x='tip_amount', y='Airport_fee', data=data)
plt.title('Scatter Plot: Airport_fee vs tip_amount')
plt.xlabel('tip_amount')
plt.ylabel('Airport_fee')
plt.show()
```



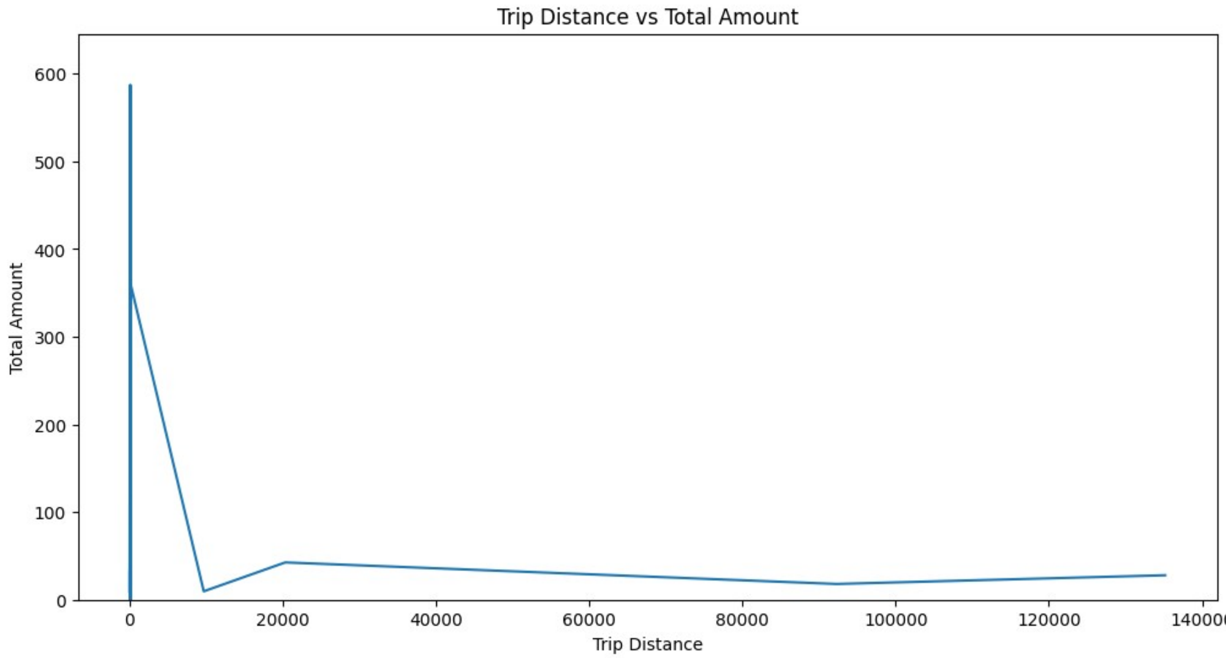
The scatter plot effectively illustrates the relationship between 'tip_amount' and 'Airport_fee'.It allows us to identify outliers within the dataset, as certain data points deviate significantly from the overall pattern. Besides, it can also be concluded that airport_fee is a categorical data.

```
# Using heatmap to find correlation among the specified features
selected_features = data[['passenger_count', 'trip_distance', 'extra',
'total_amount','improvement_surcharge','tip_amount','tolls_amount']]

correlation_matrix = selected_features.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f", linewidths=.5)
plt.title('Correlation Matrix of Selected Features')
plt.show()
```



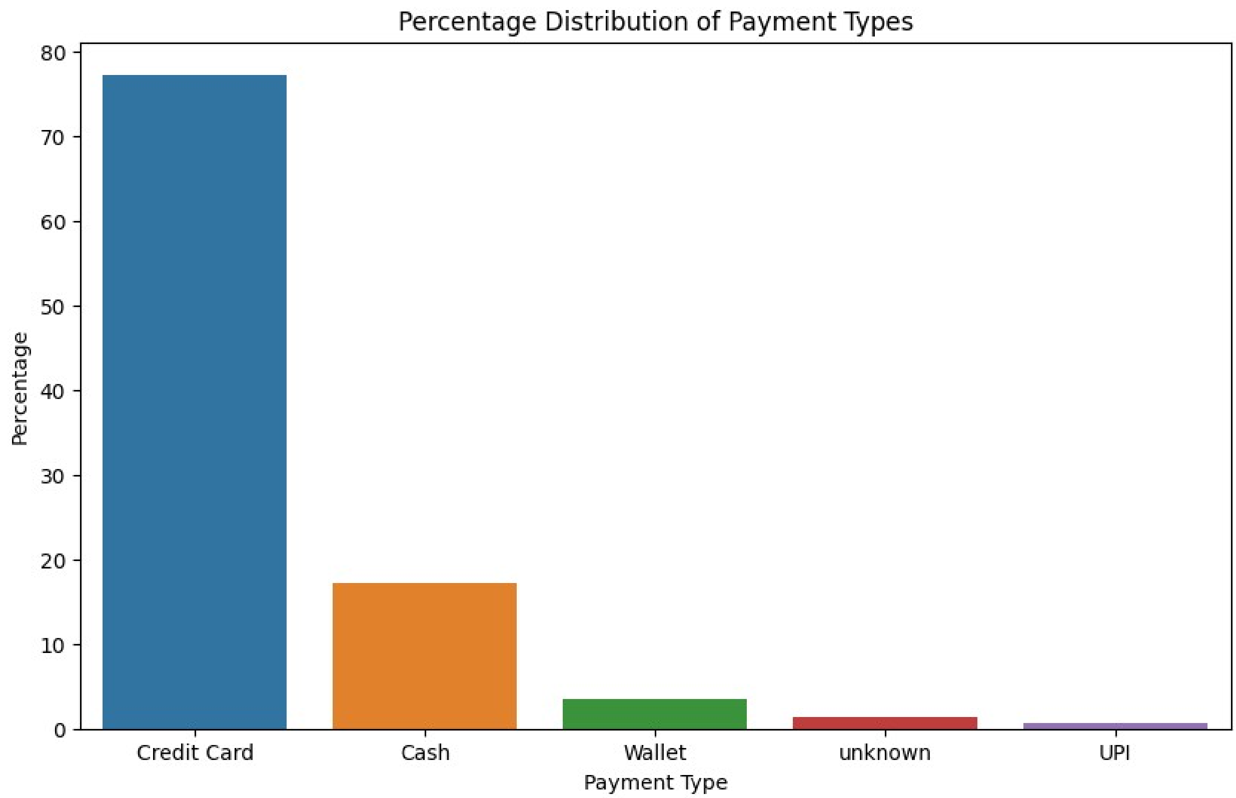
In our analysis, it is evident that tolls amount and total amount exhibit a high degree of correlation, suggesting a strong relationship between these two variables. On the other hand, features such as passenger count and extras appear to be the least correlated, indicating a weaker association between these aspects.

```
plt.figure(figsize=(12, 6))
sns.lineplot(x='trip_distance', y='total_amount', data=data)
plt.title('Trip Distance vs Total Amount')
plt.xlabel('Trip Distance')
plt.ylabel('Total Amount')
plt.ylim(0, None)
plt.show()
```



As the distance increases, a discernible trend emerges, indicating an unchanging pattern in the total trip amount. This suggests that, beyond a certain distance, the total amount does not exhibit significant variation, implying a potential saturation or consistency in pricing with increasing trip distance.

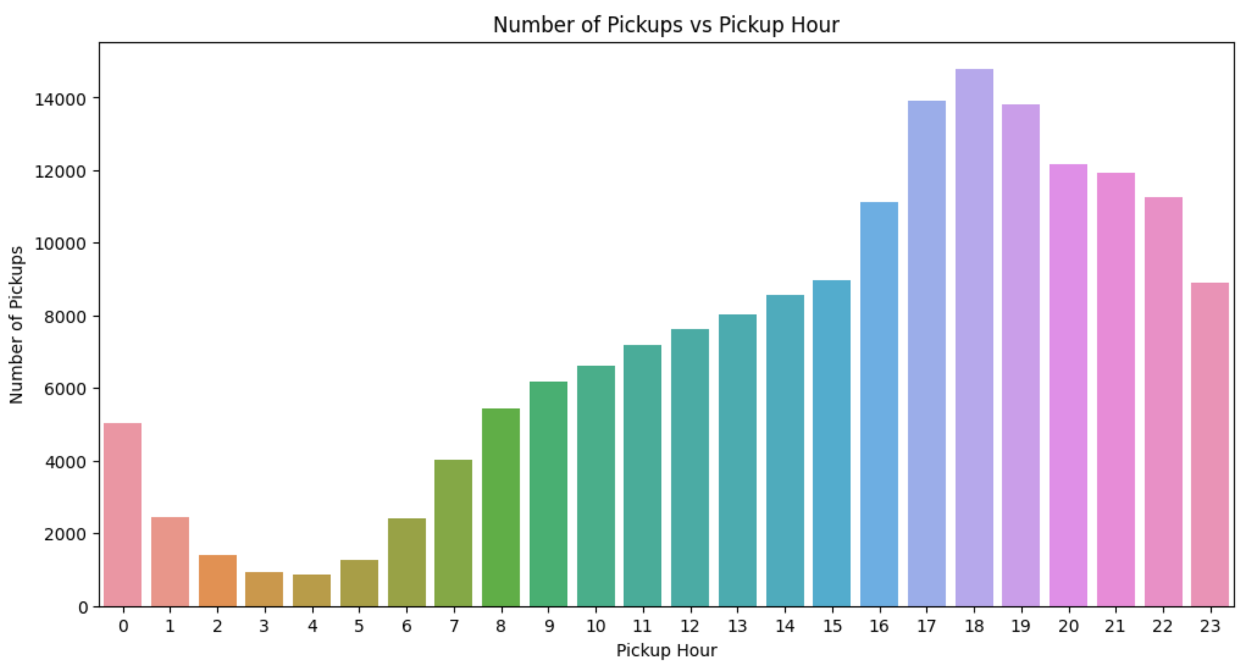
```
# Let us calculate the percentage each unique payment mode present
over there in the dataset.
payment_method_c = data['payment_type'].value_counts()
payment_method_p = (payment_method_c / len(data)) * 100
payment_method_p_rounded = payment_method_p.round(2)
print(payment_method_p_rounded)

payment_type
Credit Card    77.29
Cash           17.22
Wallet          3.47
unknown         1.33
UPI             0.68
Name: count, dtype: float64

# Here is our plot for percentage distribution of the various modes of
payment
plt.figure(figsize=(10, 6))
sns.barplot(x=payment_method_p_rounded.index,
y=payment_method_p_rounded.values)
plt.title('Percentage Distribution of Payment Types')
plt.xlabel('Payment Type')
plt.ylabel('Percentage')
plt.show()
```



Clearly , the number of people using credit card is more when compared to the other modes like upi and wallet

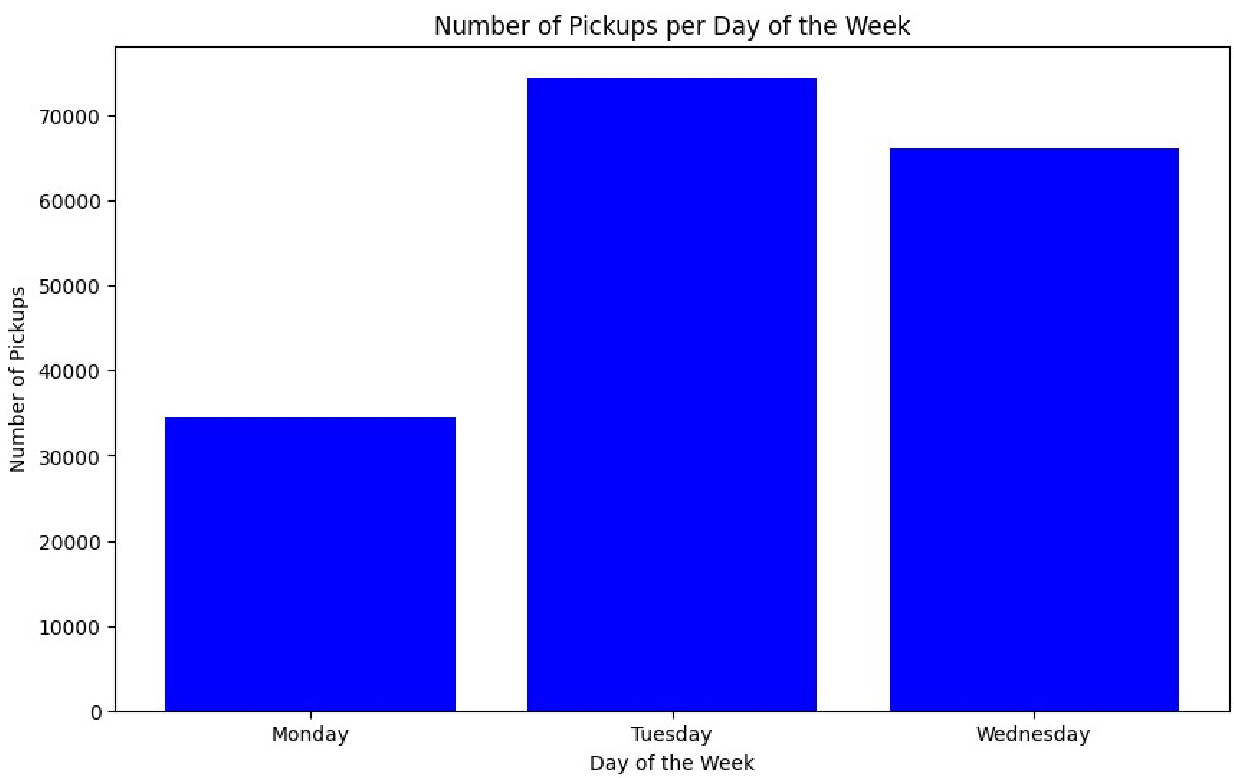
```
''' Converting the 'tpep_pickup_datetime' and 'tpep_dropoff_datetime'
columns to datetime objects
 and then extracting day, month, and hour information for both pickup
and dropoff times. '''

data['tpep_pickup_datetime'] =
pd.to_datetime(data['tpep_pickup_datetime'])
data['tpep_dropoff_datetime'] =
pd.to_datetime(data['tpep_dropoff_datetime'])


data['pickup_day'] = data['tpep_pickup_datetime'].dt.day
data['pickup_month'] = data['tpep_pickup_datetime'].dt.month
data['pickup_hour'] = data['tpep_pickup_datetime'].dt.hour

data['dropoff_day'] = data['tpep_dropoff_datetime'].dt.day
data['dropoff_month'] = data['tpep_dropoff_datetime'].dt.month
data['dropoff_hour'] = data['tpep_dropoff_datetime'].dt.hour


data.drop(columns=['tpep_pickup_datetime', 'tpep_dropoff_datetime'],
inplace=True)
```

```python
data_t['tpep_pickup_datetime'] =
pd.to_datetime(data_t['tpep_pickup_datetime'])
data_t['tpep_dropoff_datetime'] =
pd.to_datetime(data_t['tpep_dropoff_datetime'])


data_t['pickup_day'] = data_t['tpep_pickup_datetime'].dt.day
data_t['pickup_month'] = data_t['tpep_pickup_datetime'].dt.month
data_t['pickup_hour'] = data_t['tpep_pickup_datetime'].dt.hour

data_t['dropoff_day'] = data_t['tpep_dropoff_datetime'].dt.day
data_t['dropoff_month'] = data_t['tpep_dropoff_datetime'].dt.month
data_t['dropoff_hour'] = data_t['tpep_dropoff_datetime'].dt.hour


data_t.drop(columns=['tpep_pickup_datetime', 'tpep_dropoff_datetime'],
inplace=True)

# Here is a plot for number of pickups vs pickup hour
plt.figure(figsize=(12, 6))
sns.countplot(x='pickup_hour', data=data)
plt.title('Number of Pickups vs Pickup Hour')
plt.xlabel('Pickup Hour')
plt.ylabel('Number of Pickups')
plt.show()
```



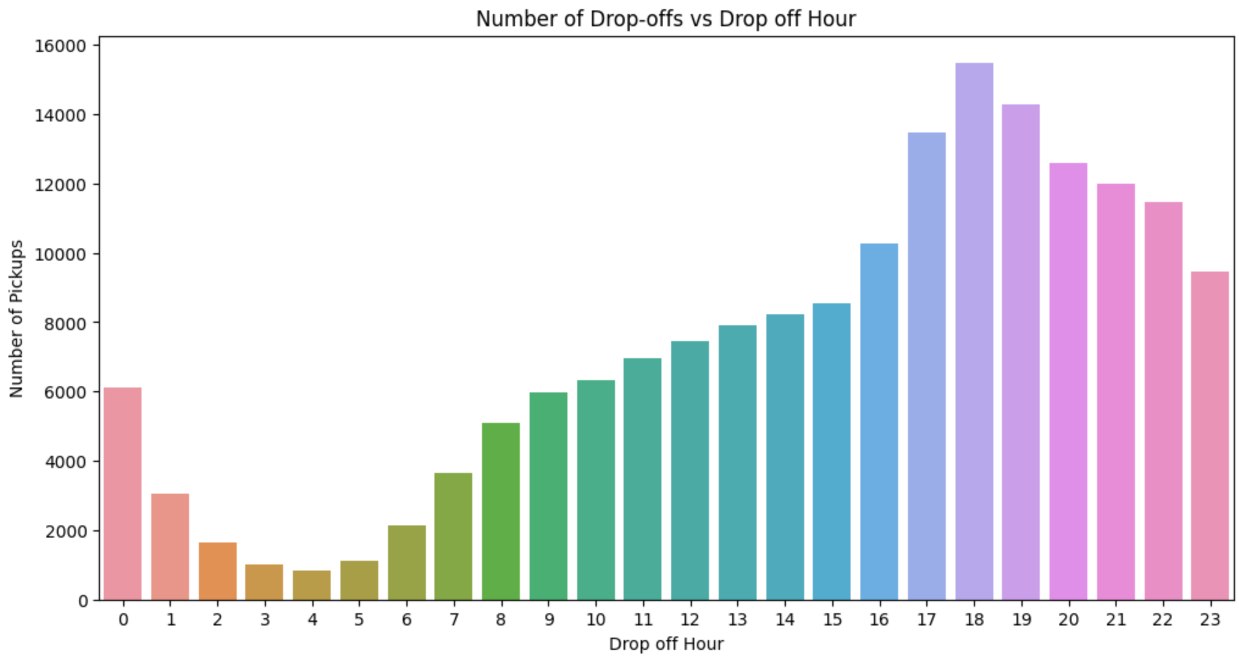
When it comes to pickup hour , 2 am to 6 am in the morning is quiet idle where 5 pm to 10 pm in the evening seems to be the busiest

```
# Here is a bar chart created for getting the most busiest weekday
present in our dataset.

weekday_names = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday', 'Sunday']

# Grouping by pickup day and counting the number of pickups for each
day
pickup_counts =
data.groupby('pickup_day').size().reset_index(name='pickup_count')

pickup_counts['weekday_name'] = pickup_counts['pickup_day'].map(lambda
day: weekday_names[day % 7])

pickup_counts_by_weekday = pickup_counts.groupby('weekday_name')
['pickup_count'].sum().reset_index()

plt.figure(figsize=(10, 6))
plt.bar(pickup_counts_by_weekday['weekday_name'],
pickup_counts_by_weekday['pickup_count'], color='blue')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Pickups')
plt.title('Number of Pickups per Day of the Week')
plt.show()
```



On our radar, Tuesday stands out as the busiest pickup day of the week, with Monday being the least active. A clear pattern emerges, highlighting the varying intensity of pickups throughout the week

```
# Here is a plot vs number of drop-offs vs drop off hour
plt.figure(figsize=(12, 6))
sns.countplot(x='dropoff_hour', data=data)
plt.title('Number of Drop-offs vs Drop off Hour')
plt.xlabel('Drop off Hour')
plt.ylabel('Number of Pickups')
plt.show()
```



When it comes to drop off hour , 2 am to 6 am in the morning is quiet idle where 5 pm to 11 pm in the evening seems to be the busiest

Data Cleaning

```
data

        VendorID  passenger_count  trip_distance  RatecodeID  \
0              1              1.0           2.14         1.0
1              0              1.0           2.70         1.0
2              1              1.0           1.15         1.0
3              0              1.0           0.40         1.0
4              1              3.0           1.10         1.0
...          ...              ...            ...         ...
174995         1              3.0           3.45         1.0
174996         1              1.0           9.44         1.0
174997         0              1.0           2.40         1.0
```

```
174998       1            1.0           4.71         1.0
174999       1            1.0           1.01         1.0

       store_and_fwd_flag  PULocationID  DOLocationID payment_type
extra  \
0                       N           120             9  Credit Card
2.5
1                       N            15           215  Credit Card
3.5
2                       N           167           223  Credit Card
0.0
3                       N           128           239  Credit Card
2.5
4                       N           203            52  Credit Card
1.0
...                   ...           ...           ...          ...
...
174995                  N           147           167  Credit Card
1.0
174996                  N           154           191         Cash
5.0
174997                  N           168           106  Credit Card
2.5
174998                  N           240           100  Credit Card
2.5
174999                  N           153            72  Credit Card
1.0

        tip_amount  ...  improvement_surcharge  total_amount  \
0         7.165589  ...                    1.0         20.64
1         6.067401  ...                    1.0         25.55
2         4.111547  ...                    1.0         17.64
3         6.411079  ...                    1.0         12.80
4         4.769377  ...                    1.0         18.00
...            ...  ...                    ...           ...
174995    8.732495  ...                    1.0         28.08
174996    0.283275  ...                    1.0         59.95
174997    4.245354  ...                    1.0         33.50
174998   10.479776  ...                    1.0         40.80
174999    6.541699  ...                    1.0         16.32

        congestion_surcharge  Airport_fee  pickup_day  pickup_month  \
0                        2.5         0.00          28             6
1                        2.5         0.00          29             6
2                        2.5         0.00          30             6
3                        2.5         0.00          29             6
4                        2.5         0.00          29             6
...                      ...          ...         ...           ...
174995                   2.5         0.00          30             6
174996                   2.5         1.75          30             6
```

```
174997                       2.5          0.00          29              6
174998                       2.5          0.00          29              6
174999                       2.5          0.00          30              6

        pickup_hour  dropoff_day  dropoff_month  dropoff_hour
0                17           28              6            16
1                23           29              6            22
2                10           30              6            11
3                13           29              6            14
4                22           29              6            22
...             ...          ...            ...           ...
174995           22           30              6            22
174996           13           30              6            14
174997           11           29              6            12
174998           19           29              6            19
174999           21           30              6            22

[175000 rows x 21 columns]
```

```
    #importing the neccesary libraries
    from sklearn.model_selection import train_test_split
    from sklearn.compose import ColumnTransformer
    from sklearn.impute import SimpleImputer
    from sklearn.preprocessing import OneHotEncoder
    from sklearn.preprocessing import MinMaxScaler , StandardScaler
    from sklearn.pipeline import Pipeline,make_pipeline
    from sklearn.feature_selection import SelectKBest,chi2
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.preprocessing import PolynomialFeatures
    from sklearn.tree import DecisionTreeRegressor
    from sklearn.metrics import mean_squared_error, r2_score

numerical_features = ['passenger_count', 'trip_distance' , 'extra',
'tip_amount', 'tolls_amount',
'improvement_surcharge', 'congestion_surcharge', 'Airport_fee']
categorical_features = [ 'store_and_fwd_flag', 'payment_type']
numerical_features_most_frequent = ['RatecodeID','passenger_count']
#numerical_features_most_frequent2 = []

numerical_transformer = Pipeline(steps =[
    ('imputer',SimpleImputer(strategy="mean")),
    ('scaler',StandardScaler())
])

# Pipeline for features with most frequent imputation strategy
numerical_transformer_most_frequent = Pipeline(steps=[
    ('imputer_most_frequent',
SimpleImputer(strategy='most_frequent')),
    ('scaler', StandardScaler())
])
```

```
'''numerical_transformer_most_frequent2 = Pipeline(steps=[
    ('imputer_most_frequent2',
SimpleImputer(strategy='most_frequent')),
    ('scaler', StandardScaler())  # Assuming the same scaling for
these features
])'''

'''So, imputing missing values with the most frequent value and one-
hot encoding with handling of unknown categories.'''

categorical_transformer = Pipeline(steps=[
    ('imputer',SimpleImputer(strategy='most_frequent')),
    ('onehot',OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(transformers=[
    ('num_most_frequent', numerical_transformer_most_frequent,
numerical_features_most_frequent),
    ('num', numerical_transformer, numerical_features),
    ('cat', categorical_transformer, categorical_features)
])
```

SPLITTING THE DATASET INTO TRAIN AND TEST DATA

```
y = data['total_amount']
X_train,X_test, y_train , y_test =
train_test_split(data.drop(columns=['total_amount']),y,test_size
=0.2,random_state =42)

X_train.head()
```

```
        VendorID  passenger_count  trip_distance  RatecodeID  \
143961         1              1.0           7.79         1.0
170292         1              1.0           0.79         1.0
161029         1              1.0           0.29         2.0
84006          0              1.0           0.60         1.0
95628          0              1.0           1.90         1.0

       store_and_fwd_flag  PULocationID  DOLocationID payment_type
extra  \
143961                  N           181           174  Credit Card
1.0
170292                  N           250           226  Credit Card
2.5
161029                  N           236           251      unknown
0.0
84006                   N            83           166  Credit Card
3.5
95628                   N            70            35  Credit Card
5.0
```

```
        tip_amount  tolls_amount  improvement_surcharge
congestion_surcharge  \
143961    7.956385           0.0                    1.0
0.0
170292    2.276785           0.0                    1.0
2.5
161029    1.062698           0.0                   -1.0
-2.5
84006     2.444217           0.0                    1.0
2.5
95628     5.163920           0.0                    1.0
2.5

        Airport_fee  pickup_day  pickup_month  pickup_hour
dropoff_day  \
143961         1.75          29             6           23
30
170292         0.00          28             6           19
28
161029         0.00          30             6           21
30
84006          0.00          29             6           20
29
95628          0.00          28             6           17
28

        dropoff_month  dropoff_hour
143961              6             0
170292              6            18
161029              6            21
84006               6            21
95628               6            17
```

MODEL BUILDING AND EVALUATION

```
#Applying Linear regression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
model_lr = Pipeline(steps=[
    ('preprocessor_lr', preprocessor),
    ('model_lr', LinearRegression())
])

model_lr.fit(X_train, y_train)
y_pred_lr = model_lr.predict(X_test)

r2 = r2_score(y_test, y_pred_lr)
```

```
explained_variance_lr = r2 * 100
print("R2 score for linear regression is" ,explained_variance_lr)
```

```
R2 score for linear regression is 72.19555760853076
```

```
#Applying Polynomial Features
from sklearn.preprocessing import PolynomialFeatures
model_pf = Pipeline(steps=[
    ('preprocessor_pf', preprocessor),
    ('poly_features', PolynomialFeatures(degree=2)),
    ('model_pf', LinearRegression())
])

model_pf.fit(X_train, y_train)
y_pred_pf = model_pf.predict(X_test)
r2 = r2_score(y_test, y_pred_pf)
explained_variance_pf = r2 * 100
print("R2 score for polynomial regression is" ,explained_variance_pf)
```

```
R2 score for polynomial regression is 88.03033643454853
```

Applying Hyperparameter Tuning

```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for PolynomialFeatures
param_grid = {'poly_features__degree': [2,3]}

# Create the grid search
grid_search = GridSearchCV(model_pf, param_grid, cv=3,
scoring='neg_mean_squared_error')

# Fit the grid search to your data
grid_search.fit(X_train, y_train)

# Get the best model from the grid search
best_model = grid_search.best_estimator_

# Print the best degree
best_degree = best_model.named_steps['poly_features'].degree
print("Best Polynomial Degree:", best_degree)
```

```
Best Polynomial Degree: 2
```

```
#Applying DecisionTree Regressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures


model_dt = Pipeline(steps=[
```

```
    ('preprocessor_dt', preprocessor),
    ('model_dt', DecisionTreeRegressor())
])

model_dt.fit(X_train, y_train)
y_pred_dt = model_dt.predict(X_test)
r2 = r2_score(y_test, y_pred_dt)
explained_variance_dt = r2 * 100
print("R2 score for decision tree regression
is" ,explained_variance_dt)
```

```
R2 score for decision tree regression is 91.7292913398568
```

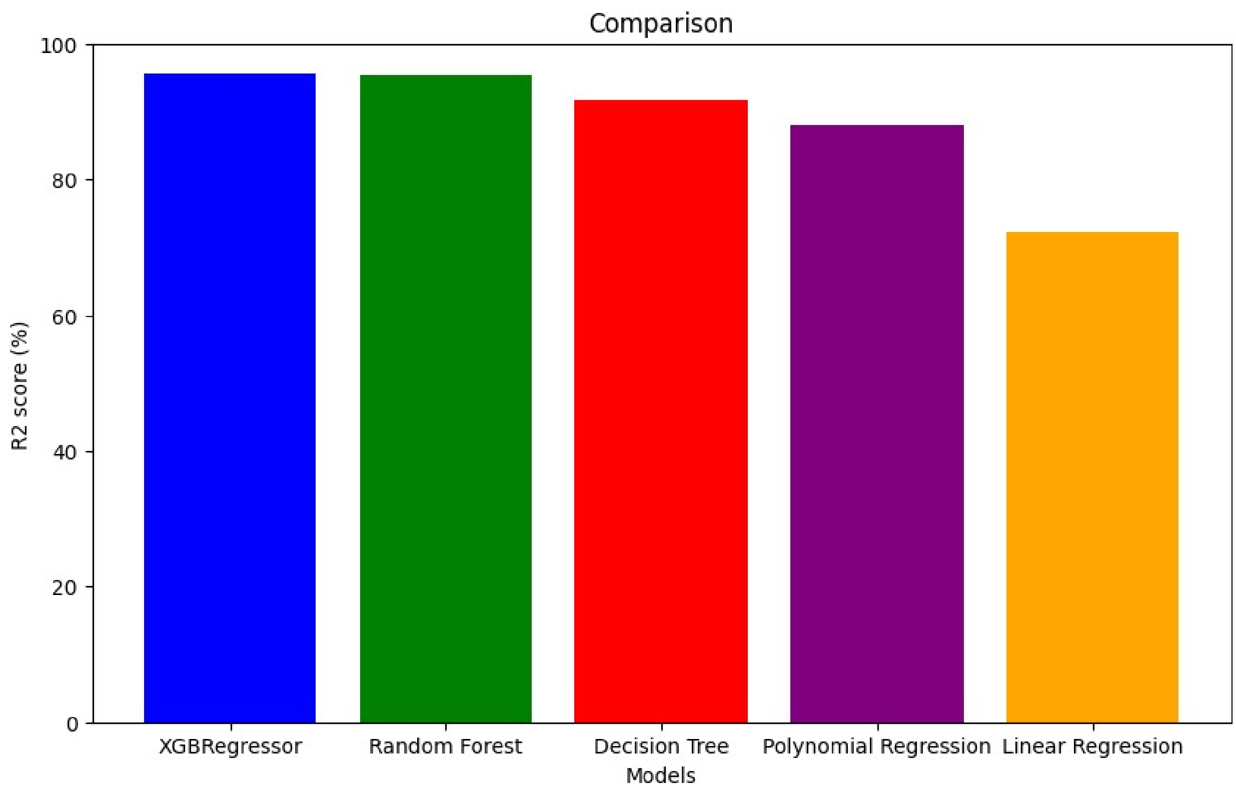
```
#Applying RandomForestRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score


model_rf = Pipeline(steps=[
    ('preprocessor_rf', preprocessor),
    ('model_rf', RandomForestRegressor())
])

model_rf.fit(X_train, y_train)
y_pred_rf = model_rf.predict(X_test)
r2_rf = r2_score(y_test, y_pred_rf)
explained_variance_rf = r2_rf * 100
print("R2 score for Random Forest regression is",
explained_variance_rf)
```

```
R2 score for Random Forest regression is 95.31685928889165
```

```
#Applying XGBRegressor
from xgboost import XGBRegressor
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score

model_xgb = Pipeline(steps=[
    ('preprocessor_xgb', preprocessor),
    ('model_xgb', XGBRegressor())
])

model_xgb.fit(X_train, y_train)
y_pred_xgb = model_xgb.predict(X_test)
r2_xgb = r2_score(y_test, y_pred_xgb)
explained_variance_xgb = r2_xgb * 100
print("R2 score for XGBoost regression is", explained_variance_xgb)
```

```
R2 score for XGBoost regression is 95.64346335641403
```

```
explained_variances = [explained_variance_xgb, explained_variance_rf,
explained_variance_dt, explained_variance_pf, explained_variance_lr]
model_names = ['XGBRegressor', 'Random Forest', 'Decision Tree',
'Polynomial Regression', 'Linear Regression']

plt.figure(figsize=(10, 6))
plt.bar(model_names, explained_variances, color=['blue', 'green',
'red', 'purple', 'orange'])
plt.title('Comparison')
plt.xlabel('Models')
plt.ylabel('R2 score (%)')
plt.ylim(0, 100)
plt.show()
```



So , clearly XGBRegressor is our best model where Linear Regression being the worst one

Prediction

```
y_pred_t=model_xgb.predict(data_t)
y_pred_t

array([34.858337, 24.224398, 14.648788, ..., 20.439754, 37.06559 ,
       16.91554 ], dtype=float32)
```

Submission

```python
# Create a DataFrame with the 'ID' and 'total_amount' columns
submission_df = pd.DataFrame({
    'ID': range(1, len(y_pred_t) + 1),  # Assuming index starts from 1
    'total_amount': y_pred_t
})

# Save the DataFrame to a CSV file
submission_df.to_csv('submission.csv', index=False)

submission_df.shape
```

```
(50000, 2)
```