Background on the Automatidata scenario

Automatidata works with its clients to transform their unused and stored data into useful solutions, such as performance dashboards, customer-facing tools, strategic business insights, and more. They specialize in identifying a client's business needs and utilizing their data to meet those business needs.

Automatidata is consulting for the New York City Taxi and Limousine Commission (TLC). New York City TLC is an agency responsible for licensing and regulating New York City's taxi cabs and for-hire vehicles. The agency has partnered with Automatidata to develop a regression model that predicts taxi and limousine ride durations based on location and time of day data that TLC has gathered.

The TLC data comes from over 200,000 taxi and limousine licensees, making approximately one million combined trips per day.

Project background

Automatidata is near the end of the TLC project. The following tasks are needed at this stage of the project:

1. Determine the correct modeling approach
2. Build a regression model
3. Finish checking model assumptions
4. Evaluate the model
5. Interpret model results and summarize findings for stakeholders within TLC

Automatidata Teammates

Udo Bankole, Director of Data Analysis

Deshawn Washington, Data Analysis Manager

Luana Rodriquez, Senior Data Analyst

Uli King, Senior Project Manager

Stakeholders

Juliana Soto, Finance and Administration Department Head

Titus Nelson, Operations Manager

Project goal

1. Build multiple linear regression model
2. Evaluate the model

In [1]:
```
# Imports

# Packages for numerics + dataframes
import pandas as pd
import numpy as np

# Packages for visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Packages for date conversions for calculating trip durations
from datetime import datetime
from datetime import date
from datetime import timedelta

# Packages for OLS, MLR, confusion matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics # For confusion matrix
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,r2_score,mean_squared_error
lr=LinearRegression()
```

Analyze pphase/EDA

In [3]:
```

# import data
df0=pd.read_csv("archive (3).zip")
```

In [5]:
```
df0.head(10) #   first ten rows
```

Out[5]:

	Unnamed: 0	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	Ratecoı
0	24870114	2	3/25/2017 8:55	3/25/2017 9:09	6	3.34	
1	35634249	1	4/11/2017 14:53	4/11/2017 15:19	1	1.80	
2	106203690	1	12/15/2017 7:26	12/15/2017 7:34	1	1.00	
3	38942136	2	5/7/2017 13:17	5/7/2017 13:48	1	3.70	
4	30841670	2	4/15/2017 23:32	4/15/2017 23:49	1	4.37	
5	23345809	2	3/25/2017 20:34	3/25/2017 20:42	6	2.30	
6	37660487	2	5/3/2017 19:04	5/3/2017 20:03	1	12.83	
7	69059411	2	8/15/2017 17:41	8/15/2017 18:03	1	2.98	
8	8433159	2	2/4/2017 16:17	2/4/2017 16:29	1	1.20	
9	95294817	1	11/10/2017 15:20	11/10/2017 15:40	1	1.60	

In [6]:

```
df0.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 18 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Unnamed: 0             22699 non-null  int64  
 1   VendorID               22699 non-null  int64  
 2   tpep_pickup_datetime   22699 non-null  object 
 3   tpep_dropoff_datetime  22699 non-null  object 
 4   passenger_count        22699 non-null  int64  
 5   trip_distance          22699 non-null  float64
 6   RatecodeID             22699 non-null  int64  
 7   store_and_fwd_flag     22699 non-null  object 
 8   PULocationID           22699 non-null  int64  
 9   DOLocationID           22699 non-null  int64  
 10  payment_type           22699 non-null  int64  
 11  fare_amount            22699 non-null  float64
 12  extra                  22699 non-null  float64
 13  mta_tax                22699 non-null  float64
 14  tip_amount             22699 non-null  float64
 15  tolls_amount           22699 non-null  float64
 16  improvement_surcharge  22699 non-null  float64
 17  total_amount           22699 non-null  float64
dtypes: float64(8), int64(7), object(3)
memory usage: 3.1+ MB
```

In [7]:

```python

df = df0.copy()

# Display the dataset's shape
print(df.shape)

# Display basic info about the dataset
df.info()
```

```
(22699, 18)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 18 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Unnamed: 0             22699 non-null  int64  
 1   VendorID               22699 non-null  int64  
 2   tpep_pickup_datetime   22699 non-null  object 
 3   tpep_dropoff_datetime  22699 non-null  object 
 4   passenger_count        22699 non-null  int64  
 5   trip_distance          22699 non-null  float64
 6   RatecodeID             22699 non-null  int64  
 7   store_and_fwd_flag     22699 non-null  object 
 8   PULocationID           22699 non-null  int64  
 9   DOLocationID           22699 non-null  int64  
 10  payment_type           22699 non-null  int64  
 11  fare_amount            22699 non-null  float64
 12  extra                  22699 non-null  float64
 13  mta_tax                22699 non-null  float64
 14  tip_amount             22699 non-null  float64
 15  tolls_amount           22699 non-null  float64
 16  improvement_surcharge  22699 non-null  float64
 17  total_amount           22699 non-null  float64
dtypes: float64(8), int64(7), object(3)
memory usage: 3.1+ MB
```

In [9]:

```python
# Create `trip_duration

# Display data types of `tpep_dropoff_datetime`, `tpep_pickup_datetime`
print("Data type of tpep_dropoff_datetime:", df["tpep_dropoff_datetime"].dtype)
print("Data type of tpep_pickup_datetime:", df["tpep_pickup_datetime"].dtype)

# Convert `tpep_dropoff_datetime` to datetime format
df["drop_off_converted"] = pd.to_datetime(df["tpep_dropoff_datetime"])

# Convert `tpep_pickup_datetime` to datetime format
df["pick_up_converted"] = pd.to_datetime(df["tpep_pickup_datetime"])

# Display data types of `drop_off_converted`, `pick_up_converted`
print("Data type of drop_off_converted:", df["drop_off_converted"].dtype)
print("Data type of pick_up_converted:", df["pick_up_converted"].dtype)

# Compute `trip_duration`
df["trip_duration"] = (df["drop_off_converted"] - df["pick_up_converted"])/np.timede

# Display first ten rows of dataframe after adding the new columns
df.head(10)
```

```
Data type of tpep_dropoff_datetime: object
Data type of tpep_pickup_datetime: object
Data type of drop_off_converted: datetime64[ns]
Data type of pick_up_converted: datetime64[ns]
```

Out[9]:

	Unnamed: 0	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	Rateco
0	24870114	2	3/25/2017 8:55	3/25/2017 9:09	6	3.34	
1	35634249	1	4/11/2017 14:53	4/11/2017 15:19	1	1.80	
2	106203690	1	12/15/2017 7:26	12/15/2017 7:34	1	1.00	
3	38942136	2	5/7/2017 13:17	5/7/2017 13:48	1	3.70	
4	30841670	2	4/15/2017 23:32	4/15/2017 23:49	1	4.37	
5	23345809	2	3/25/2017 20:34	3/25/2017 20:42	6	2.30	
6	37660487	2	5/3/2017 19:04	5/3/2017 20:03	1	12.83	
7	69059411	2	8/15/2017 17:41	8/15/2017 18:03	1	2.98	
8	8433159	2	2/4/2017 16:17	2/4/2017 16:29	1	1.20	
9	95294817	1	11/10/2017 15:20	11/10/2017 15:40	1	1.60	

10 rows × 21 columns

In [11]:

```python
# Check for missing data and duplicates

# Check for duplicates
print("Shape of dataframe:", df.shape)
print("Shape of dataframe with duplicates dropped:", df.drop_duplicates().shape)

# Check for missing values in dataframe
print("Total count of missing values:", df.isna().sum().sum())

# Display missing values per column in dataframe
print("Missing values per column:")
df.isna().sum()
```

```
Shape of dataframe: (22699, 21)
Shape of dataframe with duplicates dropped: (22699, 21)
Total count of missing values: 0
Missing values per column:
```

Out[11]:

```
Unnamed: 0               0
VendorID                 0
tpep_pickup_datetime     0
tpep_dropoff_datetime    0
passenger_count          0
trip_distance            0
RatecodeID               0
store_and_fwd_flag       0
PULocationID             0
DOLocationID             0
payment_type             0
fare_amount              0
extra                    0
mta_tax                  0
tip_amount               0
tolls_amount             0
improvement_surcharge    0
total_amount             0
drop_off_converted       0
pick_up_converted        0
trip_duration            0
dtype: int64
```

In [12]:

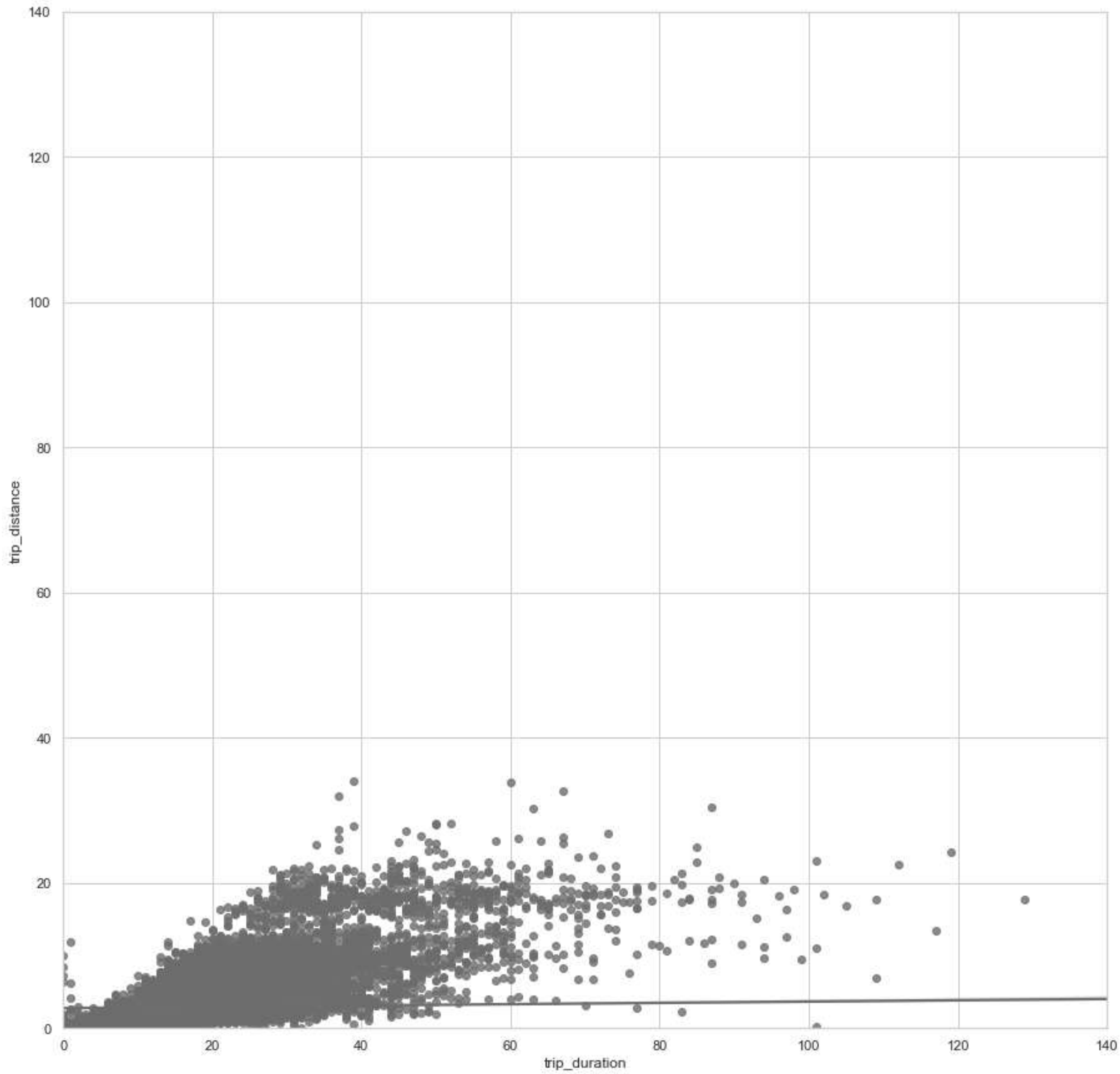
```python
# Display descriptive stats about the data
df.describe()
```

Out[12]:

ionID	DOLocationID	payment_type	fare_amount	extra	mta_tax	tip_amount	tolls_amount	im
00000	22699.000000	22699.000000	22699.000000	22699.000000	22699.000000	22699.000000	22699.000000	
12353	161.527997	1.336887	13.026629	0.333275	0.497445	1.835781	0.312542	
33373	70.139691	0.496211	13.243791	0.463097	0.039465	2.800626	1.399212	
00000	1.000000	1.000000	-120.000000	-1.000000	-0.500000	0.000000	0.000000	
00000	112.000000	1.000000	6.500000	0.000000	0.500000	0.000000	0.000000	
00000	162.000000	1.000000	9.500000	0.000000	0.500000	1.350000	0.000000	
00000	233.000000	2.000000	14.500000	0.500000	0.500000	2.450000	0.000000	
00000	265.000000	4.000000	999.990000	4.500000	0.500000	200.000000	19.100000	

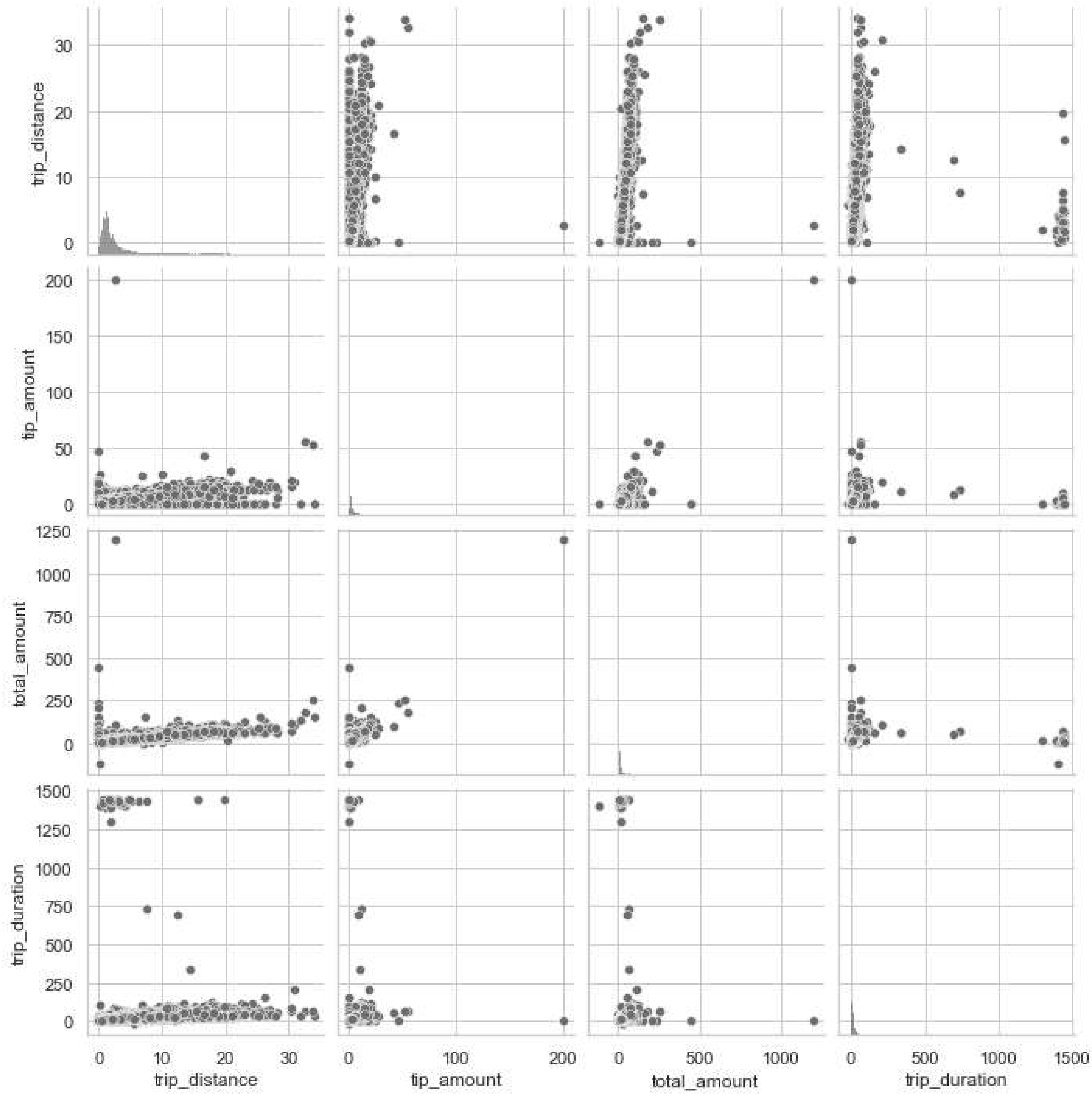
In [13]:

```python

# Create a scatter plot of trip_duration and trip_distance, with a line of best fit
sns.set(style='whitegrid')
f = plt.figure()
f.set_figwidth(15)
f.set_figheight(15)
sns.regplot(x=df["trip_duration"], y=df["trip_distance"])
plt.ylim(0, 140)
plt.xlim(0,140)
plt.show()
```



In [14]:

```
# Create a pairplot to visualize pairwise relationships between variables in the data

sns.pairplot(df[['trip_distance', 'tip_amount', 'total_amount', 'trip_duration']]);
```

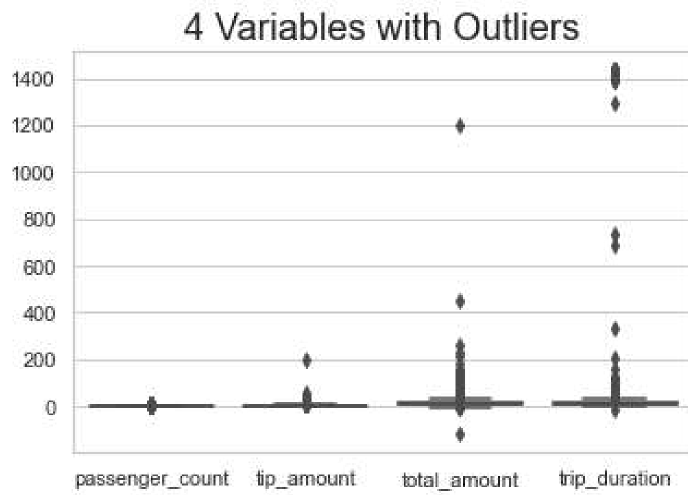


Address any outliers

In [15]:

```
# Create boxplot to visualize the outliers

g = sns.boxplot(data=df[["passenger_count","tip_amount","total_amount", "trip_durati
g.set_title("4 Variables with Outliers",fontsize=20)
```

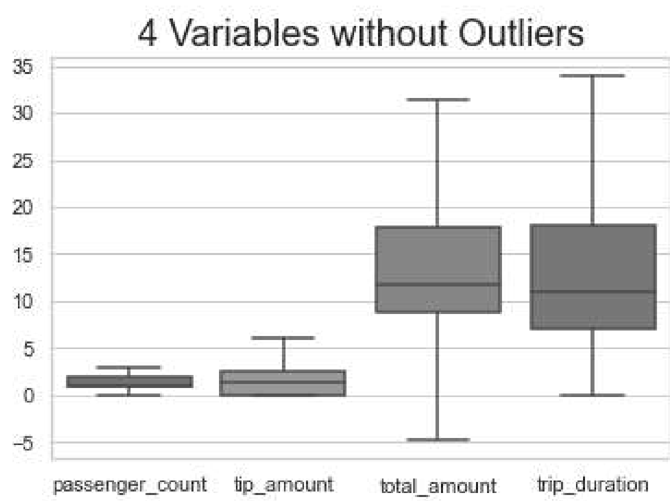
Out[15]: Text(0.5, 1.0, '4 Variables with Outliers')



In [16]:

```
# Create boxplot to visualize distribution of data without outliers

g = sns.boxplot(data=df[["passenger_count","tip_amount","total_amount", "trip_durati
g.set_title("4 Variables without Outliers",fontsize=20)
```

Out[16]: Text(0.5, 1.0, '4 Variables without Outliers')



In [17]:

```
# Compute the 25th and 75th percentile values in `trip_duration`
percentile25 = df["trip_duration"].quantile(0.25)
percentile75 = df["trip_duration"].quantile(0.75)

# Compute the interquartile range for `trip_duration`
iqr = percentile75 - percentile25

# Compute the upper limit for `trip_duration`
upper_limit = percentile75 + 1.5 * iqr
upper_limit
```

Out[17]: 34.5

In [18]:
```python
# Remove outliers in `trip_duration`:
# Set values greater than the upper limit to the upper limit (approximately 36)
# Set values less than 0 to 0
df[df["trip_duration"] > 36] = 36
df[df["trip_duration"] < 0] = 0

df["trip_duration"].describe()
```

Out[18]:
```
count    22699.000000
mean        13.663686
std          9.254188
min          0.000000
25%          7.000000
50%         11.000000
75%         18.000000
max         36.000000
Name: trip_duration, dtype: float64
```

In [19]:
```python
# Compute the 25th and 75th percentile values in `total_amount`
percentile25 = df["total_amount"].quantile(0.25)
percentile75 = df["total_amount"].quantile(0.75)

# Compute the interquartile range for `total_amount`
iqr = percentile75 - percentile25

# Compute the upper limit for `total_amount`
upper_limit = percentile75 + 1.5 * iqr
upper_limit
```

Out[19]:
```
31.375
```

In [20]:
```python
# Remove outliers in `total_amount`:
# Set values greater than the upper limit to the upper limit (approximately 32)
# Set values less than 0 to 0
df[df["total_amount"] > 32] = 32
df[df["total_amount"] < 0] = 0

# Display descriptive stats after removing outliers in `total_amount`
df["total_amount"].describe()
```

Out[20]:
```
count    22699.000000
mean        14.478729
std          7.981224
min          0.000000
25%          8.750000
50%         11.800000
75%         17.800000
max         32.000000
Name: total_amount, dtype: float64
```

Identify correlations

In [21]:

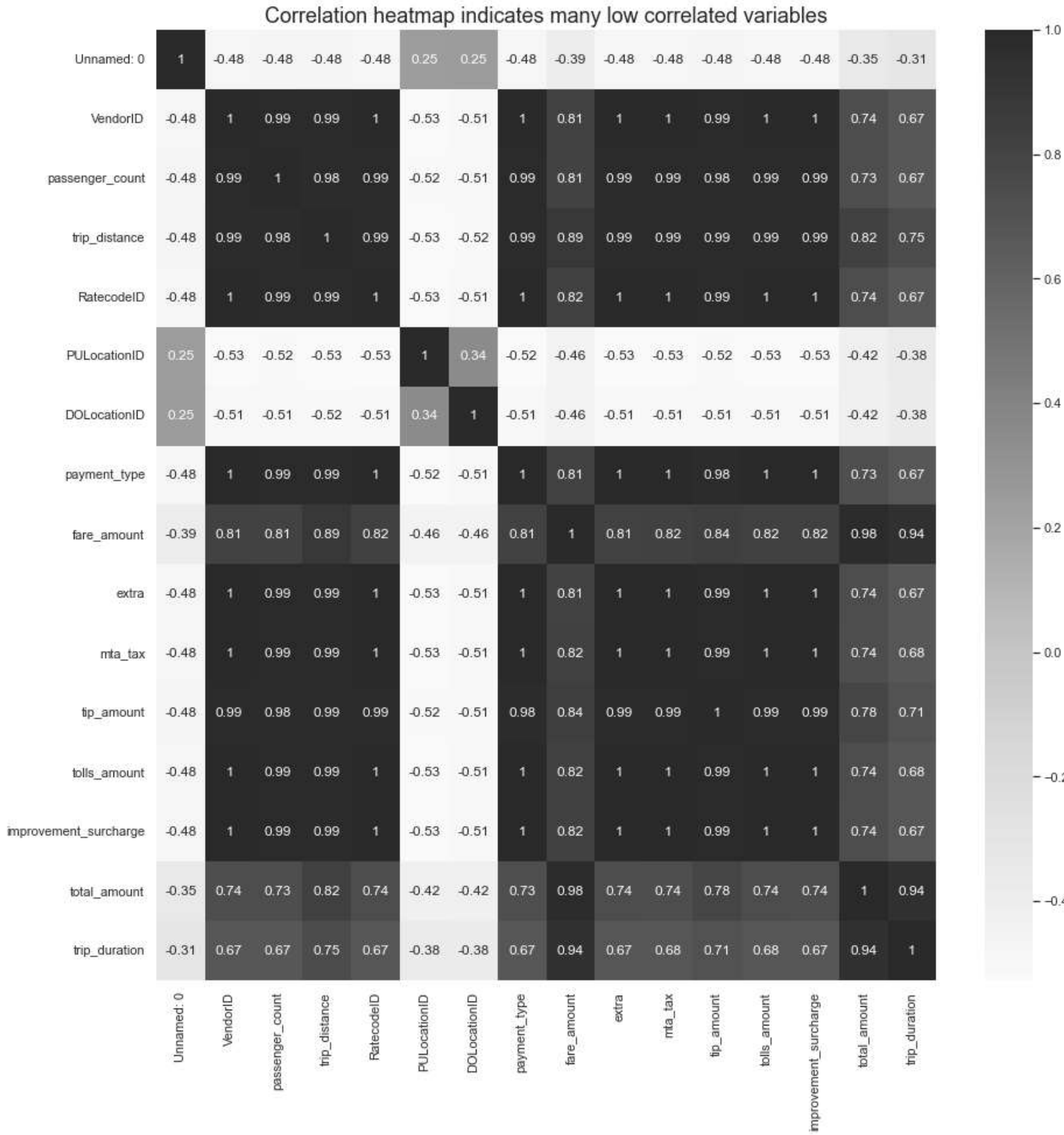
```
# Create correlation matrix containing pairwise correlation of columns, using pearso
df.corr(method="pearson")
```

Out[21]:

	Unnamed: 0	VendorID	passenger_count	trip_distance	RatecodeID	PULocationID	DOI
Unnamed: 0	1.000000	-0.480352	-0.476459	-0.476197	-0.481048	0.245243	
VendorID	-0.480352	1.000000	0.991809	0.986387	0.998622	-0.525293	
passenger_count	-0.476459	0.991809	1.000000	0.979355	0.991223	-0.521025	
trip_distance	-0.476197	0.986387	0.979355	1.000000	0.987438	-0.527592	
RatecodeID	-0.481048	0.998622	0.991223	0.987438	1.000000	-0.525036	
PULocationID	0.245243	-0.525293	-0.521025	-0.527592	-0.525036	1.000000	
DOLocationID	0.250318	-0.512225	-0.507229	-0.518649	-0.512051	0.343807	
payment_type	-0.480212	0.997359	0.990115	0.986073	0.998661	-0.524588	
fare_amount	-0.386649	0.814052	0.808990	0.887596	0.815310	-0.455797	
extra	-0.481011	0.998006	0.990576	0.987081	0.999235	-0.525037	
mta_tax	-0.481147	0.998684	0.991297	0.987574	0.999911	-0.525102	
tip_amount	-0.476850	0.988978	0.981527	0.985181	0.990265	-0.520802	
tolls_amount	-0.481163	0.998259	0.990888	0.987646	0.999529	-0.525127	
improvement_surcharge	-0.481169	0.998683	0.991297	0.987569	0.999923	-0.525134	
total_amount	-0.349756	0.736321	0.731639	0.821739	0.737534	-0.416848	
trip_duration	-0.309033	0.673751	0.670005	0.747596	0.674577	-0.377790	

In [22]:

```
# Create correlation heatmap

plt.figure(figsize=(15,15))
sns.heatmap(df.corr(method="pearson"), annot=True, cmap="Blues")  #cmap="crest")
plt.title("Correlation heatmap indicates many low correlated variables",
          fontsize=18)
plt.show()
```



Construct phase

In [23]:

```python
# Set your Y and X variables

# Set Y variable
Y = df[["trip_duration"]]

# Remove the target column from the features
X = df.drop(columns="trip_duration")

# Display first few rows
X.head()
```

Out[23]:

	Unnamed: 0	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	Rateco
0	24870114	2	3/25/2017 8:55	3/25/2017 9:09	6	3.34	
1	35634249	1	4/11/2017 14:53	4/11/2017 15:19	1	1.80	
2	106203690	1	12/15/2017 7:26	12/15/2017 7:34	1	1.00	
3	38942136	2	5/7/2017 13:17	5/7/2017 13:48	1	3.70	
4	30841670	2	4/15/2017 23:32	4/15/2017 23:49	1	4.37	

In [ ]:

```python

```

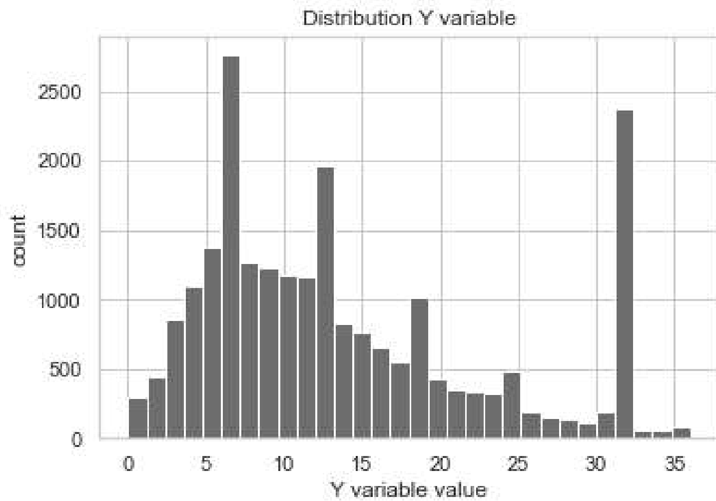
In [24]:

```python
# columns to drop

columns_to_drop = ['tpep_pickup_datetime', 'tpep_dropoff_datetime',
                   'store_and_fwd_flag', 'passenger_count', 'VendorID',
                   'fare_amount', 'PULocationID', 'DOLocationID', 'total_amount',
                   'drop_off_converted', 'pick_up_converted']
X = X.drop(columns_to_drop, axis=1)
X = X.loc[:, ~X.columns.str.contains("Unnamed")]
X.head()
```

Out[24]:

	trip_distance	RatecodeID	payment_type	extra	mta_tax	tip_amount	tolls_amount	improvement_surcharge
0	3.34	1	1	0.0	0.5	2.76	0.0	0.3
1	1.80	1	1	0.0	0.5	4.00	0.0	0.3
2	1.00	1	1	0.0	0.5	1.45	0.0	0.3
3	3.70	1	1	0.0	0.5	6.39	0.0	0.3
4	4.37	1	2	0.5	0.5	0.00	0.0	0.3

In [30]:
```
plt.hist(Y, bins=30)
plt.title("Distribution Y variable")
plt.xlabel("Y variable value")
plt.ylabel("count")
plt.show()
```



In [31]:
```
# Standardize the X variables
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
print("X scaled:", X_scaled)
```

```
X scaled: [[-0.18035575 -0.33618447 -0.37233967 ... -0.18408591 -0.33725472
  -0.33583702]
 [-0.34804254 -0.33618447 -0.37233967 ... -0.05123384 -0.33725472
  -0.33583702]
 [-0.43515255 -0.33618447 -0.37233967 ... -0.32443769 -0.33725472
  -0.33583702]
 ...
 [-0.49830731 -0.33618447 -0.26440748 ... -0.4797889  -0.33725472
  -0.33583702]
 [-0.28706552 -0.33618447 -0.37233967 ... -0.297653   -0.33725472
  -0.33583702]
 [-0.31537628 -0.33618447 -0.37233967 ... -0.2280128  -0.33725472
  -0.33583702]]
```

Build model

In [32]:
```
# Create training and testing sets

X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, Y,  test_size=0.2, ran
```

In [33]:
```
# Build and fit your model to the training data
lr=LinearRegression()
lr.fit(X_train,Y_train)
```

Out[33]: LinearRegression()

Evaluate model

In [34]:

```
# Evaluate the model performance on the training data
r_sq = lr.score(X_train, Y_train)
print("Coefficient of determination:", r_sq)
Y_pred = lr.predict(X_train)
print("R^2:", r2_score(Y_train, Y_pred))
print("MAE:", mean_absolute_error(Y_train,Y_pred))
print("RMSE:",np.sqrt(mean_squared_error(Y_train, Y_pred)))
```

```
Coefficient of determination: 0.7429168111442532
R^2: 0.7429168111442532
MAE: 3.2405134739474537
RMSE: 4.61484392596783
```

In [35]:

```
# Evaluate the model performance on the testing data
r_sq_test = lr.score(X_test, Y_test)
print("Coefficient of determination:", r_sq_test)
Y_pred_test = lr.predict(X_test)
print("R^2:", r2_score(Y_test, Y_pred_test))
print("MAE:", mean_absolute_error(Y_test,Y_pred_test))
print("RMSE:",np.sqrt(mean_squared_error(Y_test, Y_pred_test)))
```

```
Coefficient of determination: 0.736833952588452
R^2: 0.736833952588452
MAE: 3.253815127917235
RMSE: 4.632230695076403
```

Execute phase

In [36]:

```
# Create a `results` dataframe

results = pd.DataFrame(data={"actual": Y_test["trip_duration"],
                             "predicted": Y_pred_test.ravel()})
results["residual"] = results["actual"] - results["predicted"]
results.head()
```

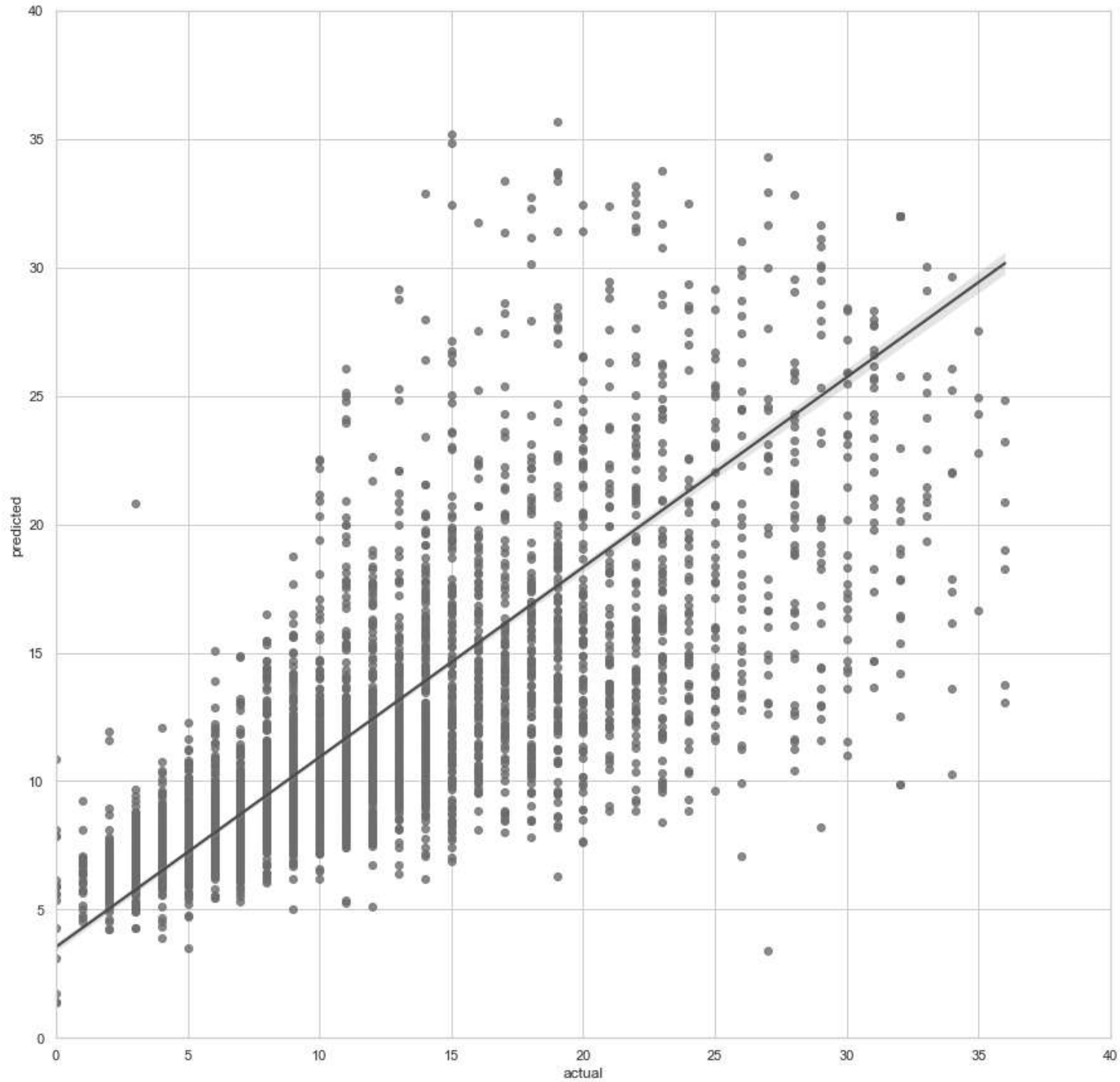
Out[36]:

	actual	predicted	residual
5818	18.0	15.675123	2.324877
18134	32.0	32.000118	-0.000118
4655	6.0	7.328350	-1.328350
7378	16.0	18.358000	-2.358000
13914	11.0	11.550253	-0.550253

Visualize model results

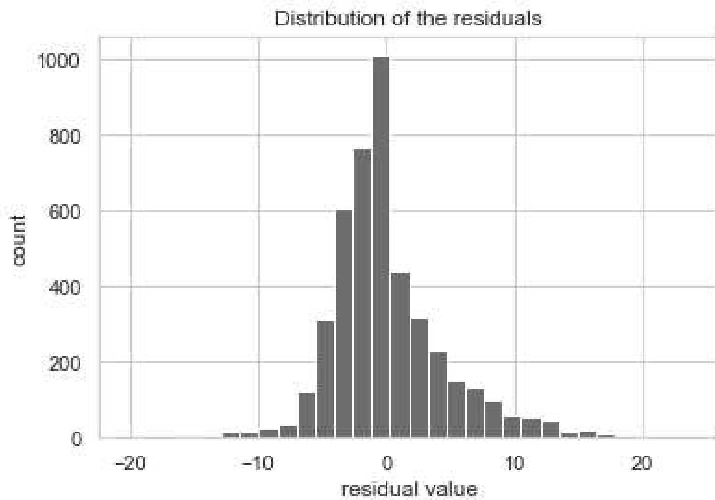
In [37]:

```
# Create a scatterplot to visualize `predicted` over `actual`

sns.set(style='whitegrid')
f = plt.figure()
f.set_figwidth(15)
f.set_figheight(15)
sns.regplot(x="actual",
           y="predicted",
           data=results, line_kws={"color": "red"})
plt.ylim(0, 40)
plt.xlim(0,40)
plt.show()
```



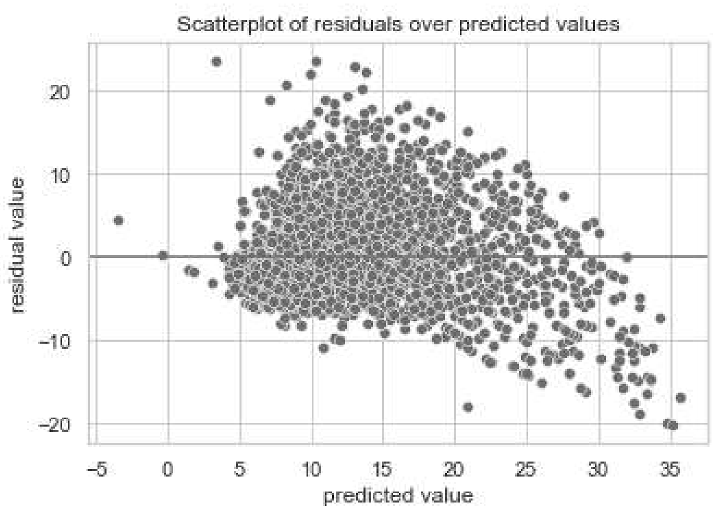
In [38]:

```python
# Visualize the distribution of the `residuals`

plt.hist(results["residual"], bins=30)
plt.title("Distribution of the residuals")
plt.xlabel("residual value")
plt.ylabel("count")
plt.show()
```



In [39]:

```python
# Create a scatterplot of `residuals` over `predicted`

sns.scatterplot(x="predicted", y="residual", data=results)
plt.axhline(0)
plt.title("Scatterplot of residuals over predicted values")
plt.xlabel("predicted value")
plt.ylabel("residual value")
plt.show()
```



Conclusion

Multiple linear regression is a powerful tool to estimate a dependent continous variable from several independent variables.

Exploratory data analysis is useful for selecting both numeric and categorical features for multiple linear regression.

Fitting multiple linear regression models may require trial and error to select variables that fit an accurate model while maintaining model assumptions.

In [ ]:
```

```