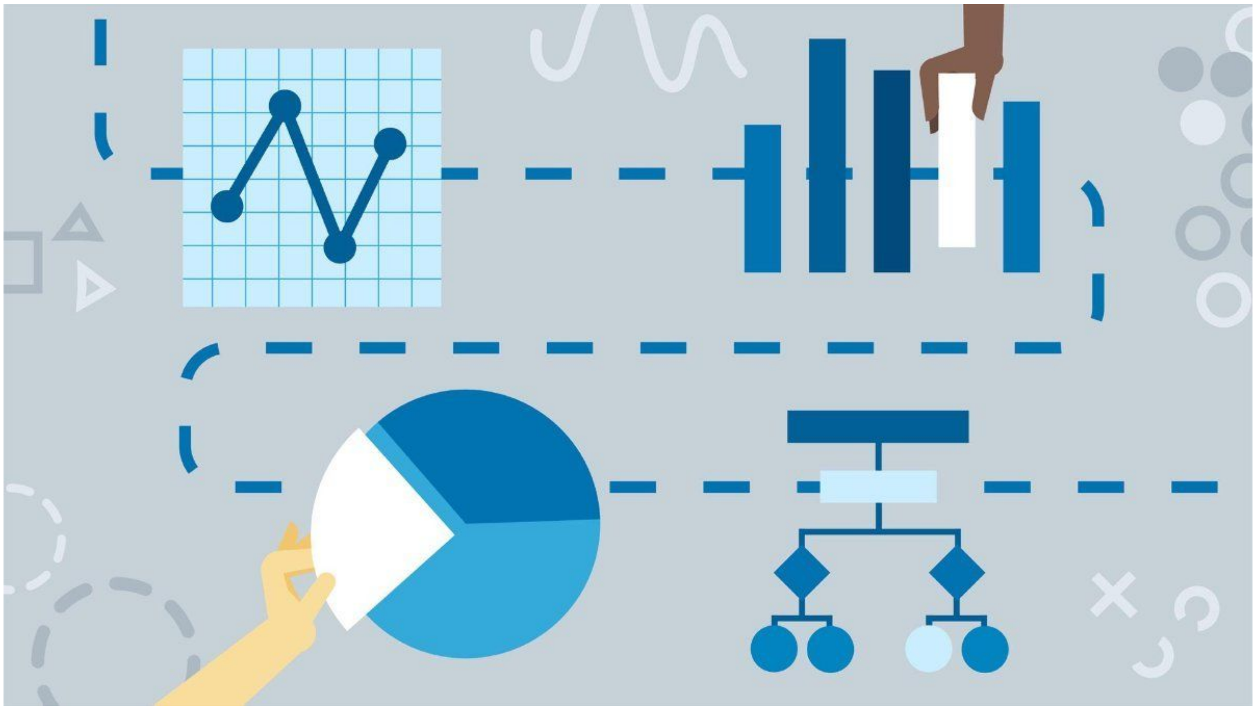


Ride Rating Prediction

26/07/19

SOUMYOJIT SINGHO ROY
HENRY HARVIN
C-102 SECTOR 2
NOIDA 201301

Overview

A Company wants us to read their data and to help them to understand their hypothesis and also help them to predict their customer's ride rating so that they can get feedback which could help them to improve their business model.

Goals

To check whether customer loves the speedy ride, the factors which can give them a good rating and to help them to predict customer ratings if the customer didn't rate their service.

Dataset We have imported the train(consist all variable) and test(consist all variable from train dataset except Loan_status) dataset from the given repository and it shows as below:

The dataset:

customer_id	Unique customer identifier. If a customer took more than one trip within 30 days, there will be more than one entry with the customer's id
driver_id	Unique driver identificator. If a driver made more than one trip within 30 days, there MAY be more than one entry with the driver's id. But be careful, we filtered the data by the customers, not drivers, so a drive may have a ride that's not shown in the dataset
creation_date	Date and time when the customer booked the ride
booking_source	The application type via which the customer booked the trip. It can be Android/IOS App, web/mobile web, etc.
car_type	Type of the car used in the trip. There are different prices and service provided by the different car type. It can be economical, luxury, minivan, etc.
estimated_distance	Estimated distance between pick-up and drop-off location according to our algorithms. Can be empty, if the customer didn't put the drop-off location in the app. Measured in kilometers
distance_travelled	Real trip distance calculated after the trip finished. Measured in kilometers
distance_travelled_while_moving	Distance driven when the car was running fast enough (eg. not stuck in a traffic)
estimated_duration	The number of minutes that we predict the trip will take. Can be empty, if the customer didn't put the drop-off location in the app
duration_time	The number of minutes that the trip really took
wait_time_initial	The number of minutes between the driver arrives to the pick-up location and customer gets into the car
wait_time_in_journey	The number of minutes during the trip when the car's speed was extremely slow (eg stuck in a traffic)
estimated_price	Price that our algorithms predict. Can be empty, if the customer didn't put the drop-off location in the app

price	Real trip price calculated after the trip completed
is_cancelled	Shows if the trip was cancelled
rating	1-5 stars the customer rated the trip. 0 is when there is no data, 1 is a minimal rating 5 is a maximal one
was_rated	Shows if the customer rated the trip

summary of the data 51083 rows and 18 columns with no null value

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 51083 entries, 0 to 51082

Data columns (total 18 columns):

Unnamed: 0 51083 non-null int64

customer_id 51083 non-null int64

driver_id 51083 non-null int64

creation_date 51083 non-null object

booking_source 51083 non-null int64

car_type 51083 non-null int64

estimated_distance 51083 non-null float64

distance_travelled 51083 non-null float64

distance_travelled_while_moving 51083 non-null float64

estimated_duration 51083 non-null int64

duration_time 51083 non-null int64

wait_time_initial 51083 non-null int64

wait_time_in_journey 51083 non-null int64

estimated_price 51083 non-null float64

price 51083 non-null float64

is_cancelled 51083 non-null int64

rating 51083 non-null int64
was_rated 51083 non-null int64

Understanding the data dictionary

1. getting a count of number of rating in eaach category

   Ride_Data['rating'].astype('category').value_counts()

5 30298

0 16206

4 3086

1 706

3 576

2 211
Name: rating, dtype: int64

2. getting a count of number of rides rated

Ride_Data['was_rated'].astype('category').value_counts()

1 34877

0 16206
Name: was_rated, dtype: int64

3.getting a count of number of rides being cancelled

Ride_Data['is_cancelled'].astype('category').value_counts()

0 48898

1 2185
Name: is_cancelled, dtype: int64

Data Visualization

plotting different variable

sns.distplot(Ride_Data['rating'])

plt.show()

sns.distplot(Ride_Data['car_type'])

plt.show()

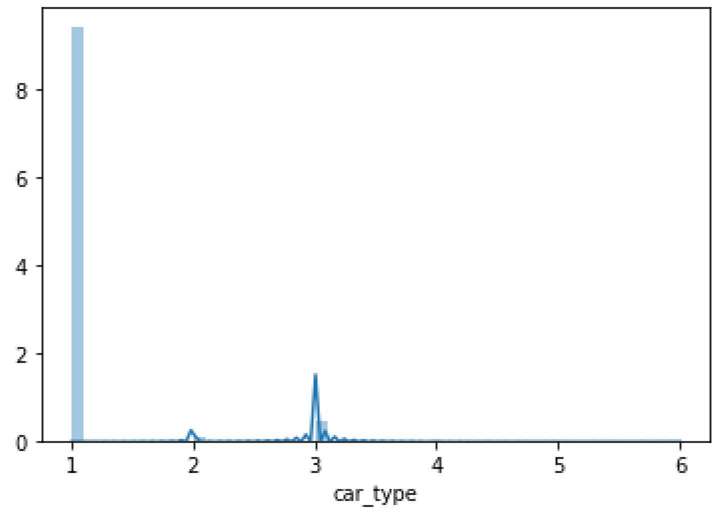
sns.distplot(Ride_Data['price'])

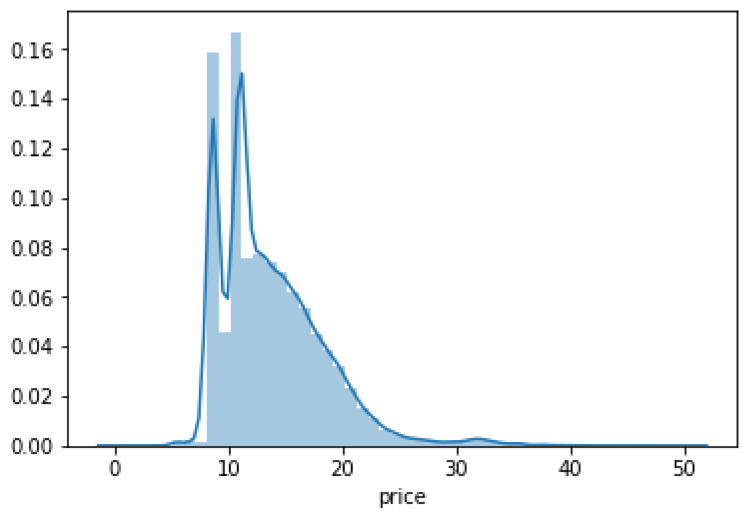
plt.show()

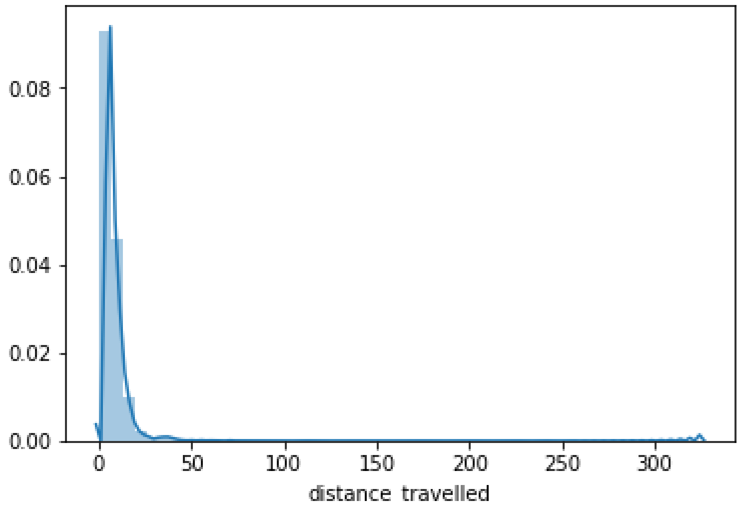
sns.distplot(Ride_Data['distance_travelled'])

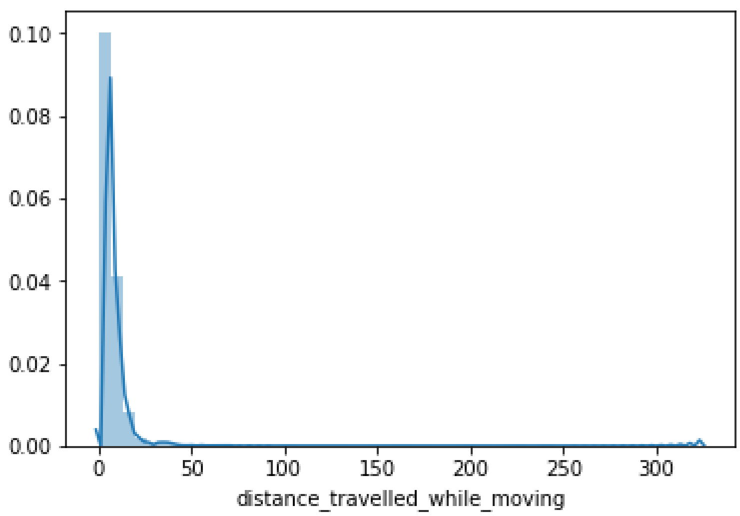
plt.show()

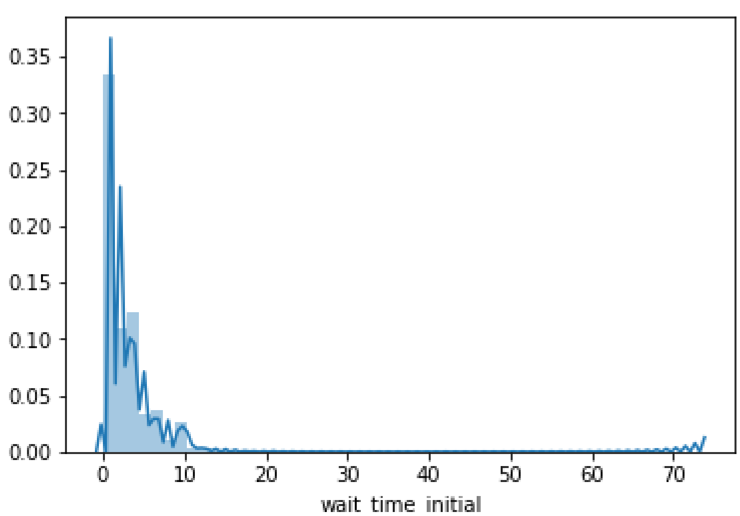
```
sns.distplot(Ride_Data['distance_travelled_while_moving'])
plt.show()
sns.distplot(Ride_Data['wait_time_initial'])
plt.show()
sns.distplot(Ride_Data['wait_time_in_journey'])
plt.show()
sns.distplot(Ride_Data['estimated_duration'])
plt.show()
```

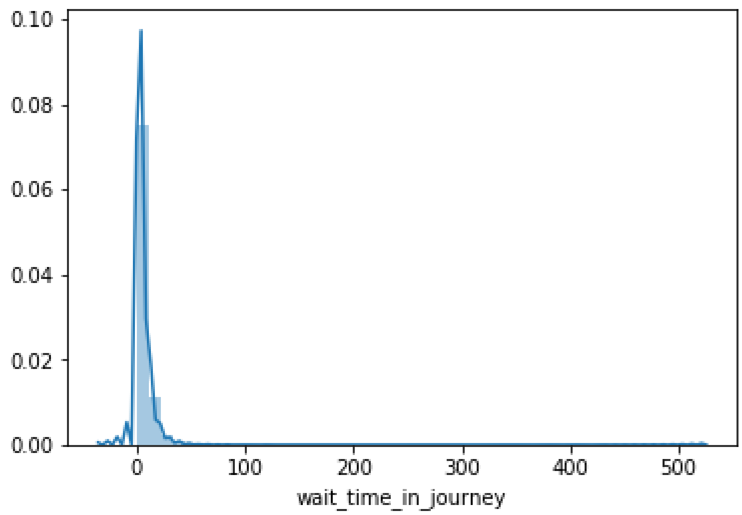


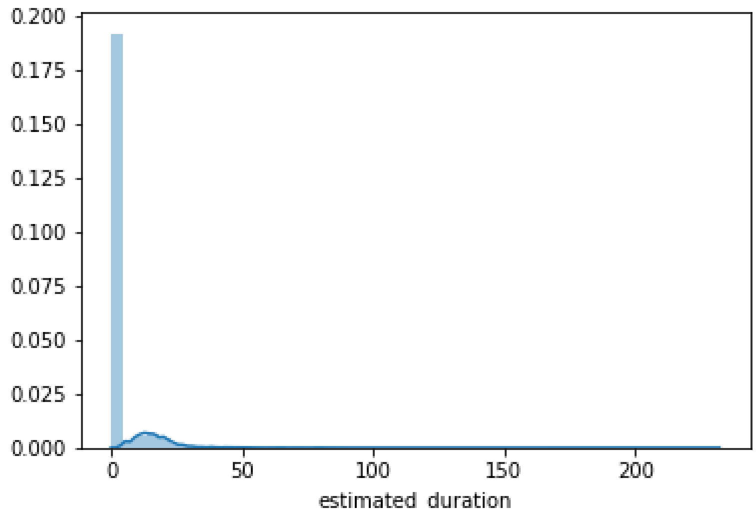



0.08
0.06
0.04
0.02
0.00
0
50
100
150
200
250
300
distance_travelled


0.10
0.08
0.06
0.04
0.02
0.00
0
50
100
150
200
250
300
distance_travelled_while_moving

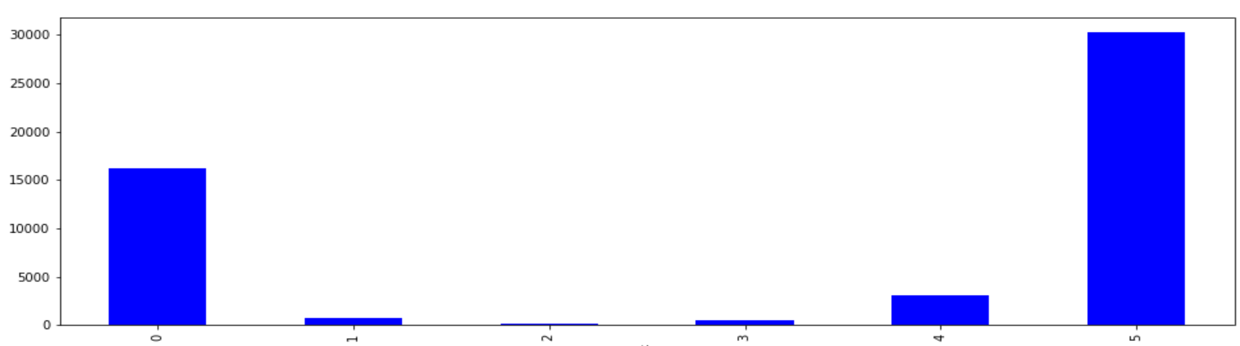

0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
10
20
30
40
50
60
70
wait_time_initial





plotting bar graph of ratings

Ride_Data.groupby('rating').size().plot(kind='bar' , color='b', figsize=(16,5))



adding a average speed column

Ride_Data['avg_speed'] = (Ride_Data['distance_travelled']/Ride_Data['duration_time'])

Ride_Data.head()

_while_moving	estimated_duration	duration_time	wait_time_initial	wait_time_in_journey	estimated_price	price	is_cancelled	rating	was_rated	avg_speed
7.17712	0	11	18	2	0.0	17.375184	0	4	1	0.662424
1.05664	0	4	1	1	0.0	8.507064	0	4	1	0.314590
14.44060	0	16	3	2	0.0	19.745205	0	5	1	0.911138
1.18684	0	8	0	5	0.0	10.968003	0	0	0	0.189889
10.67240	0	19	1	4	0.0	17.595398	0	5	1	0.583853

DATA EXLORATION..

To perform linear regression, the (numeric) target variable should be linearly related to at least one another numeric variable. Let's see whether that's true in this case.

We'll first subset the list of all (independent) numeric variables, and then make a pairwise plot.

Ride_numeric = Ride_Data.select_dtypes(include=['float64', 'int64'])

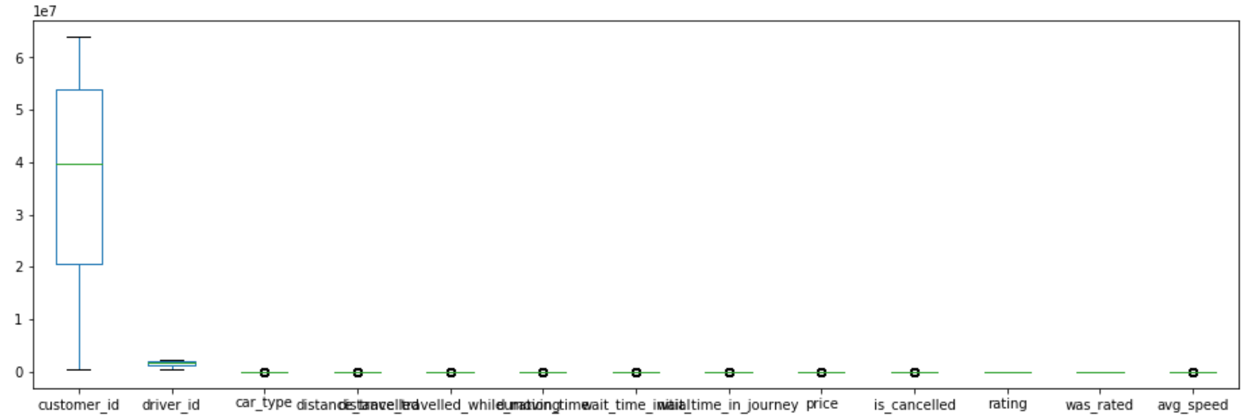
Ride_numeric.head()

dropping booking_source, estimated distance , estimated duration , estimated price

Ride_numeric = Ride_numeric.drop(['Unnamed: 0','booking_source', 'estimated_distance','estimated_duration','estimated_price'], axis=1)

Ride_numeric.head()

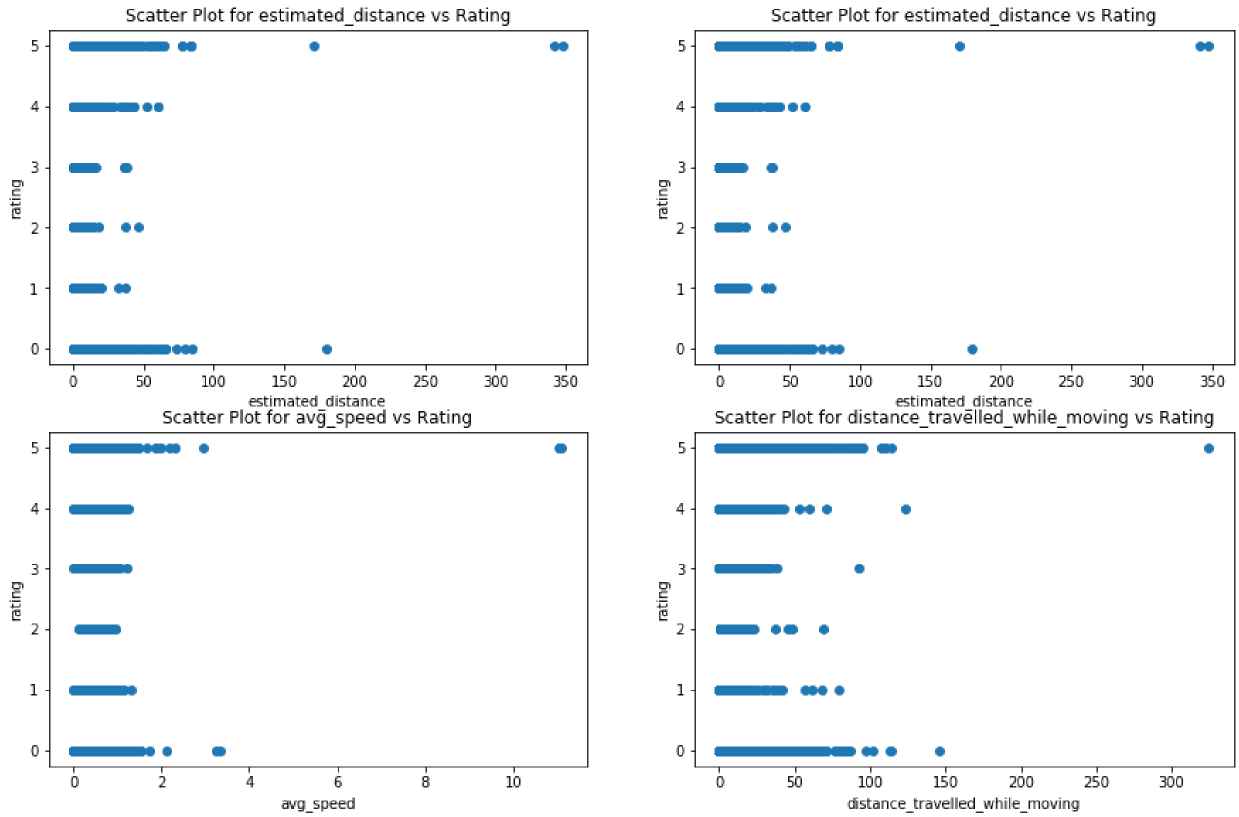
Ride_numeric.plot(kind='box', figsize=(16,5))



Spotting scatter plot against rating

```
def scatterplot(df,var):

    plt.scatter(df[var],Ride_Data['rating'])
    plt.xlabel(var); plt.ylabel('rating')
    plt.title('Scatter Plot for '+var+' vs Rating')
plt.figure(figsize=(15,20))
plt.subplot(4,2,1)
scatterplot(Ride_Data,'estimated_distance')
plt.subplot(4,2,2)
scatterplot(Ride_Data,'estimated_distance')
plt.subplot(4,2,3)
scatterplot(Ride_Data,'avg_speed')
plt.subplot(4,2,4)
scatterplot(Ride_Data,'distance_travelled_while_moving')
plt.show()
plt.tight_layout()
```



Plotting correlation matrix

```
cor = Ride_numeric.corr()

cor
```

plotting correlations on a heatmap

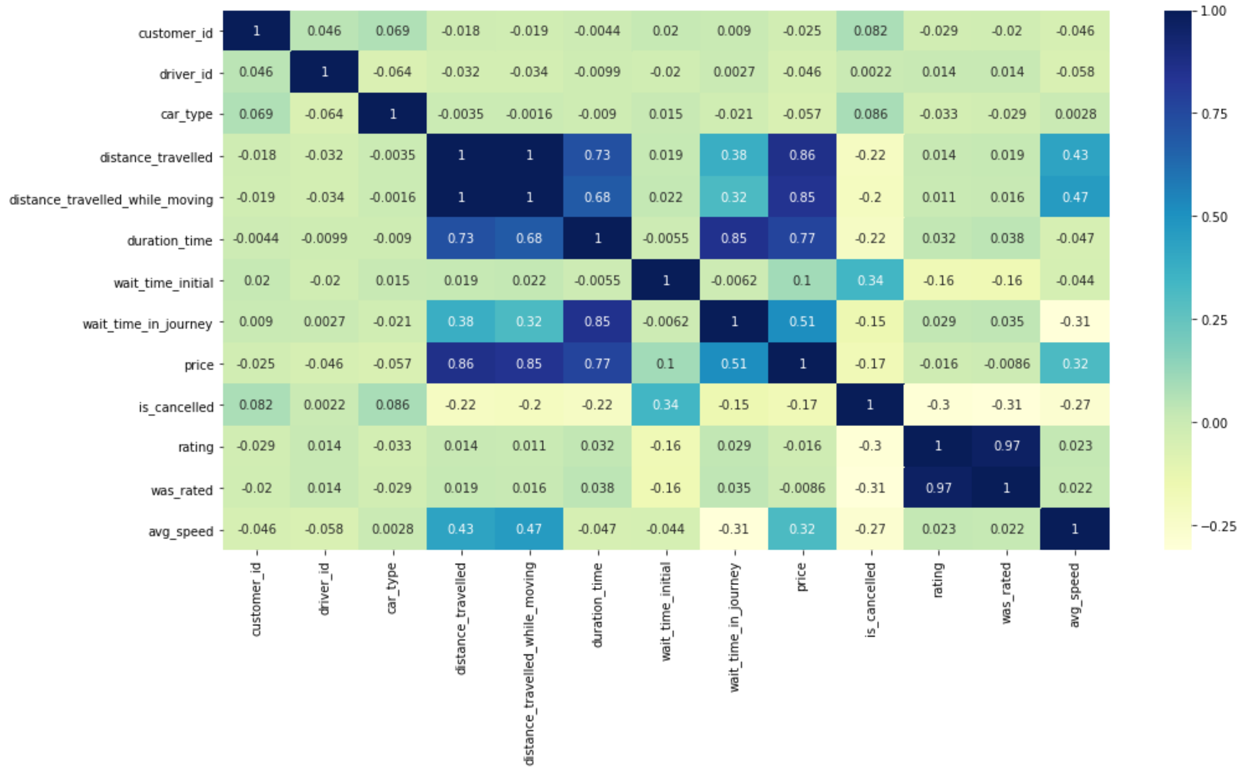
```
# figure size

plt.figure(figsize=(16,8))


# heatmap

sns.heatmap(cor, cmap="YlGnBu", annot=True)

plt.show()
```



The heatmap shows some useful insights:

Correlation of Rating with independent variables:

Rating is highly (positively) correlated with average speed, wait time for the journey, time duration, distance travelled, distance travelled while moving and driver_id

Rating is negatively correlated to price, wait time initial and car type . This suggest that Rides having high price , making the customer to wait longer and the type of the cars provided gives a bad rating

Correlation among independent variables:

Many independent variables are highly correlated : distance traveled ,time taken, price,wait time etc. are all measures of 'time and distance', and are positively correlated Thus, while building the model, we'll have to pay attention to multicollinearity (especially linear models, such as linear and logistic regression, suffer more from multicollinearity).

Ride_numeric.groupby(['car_type','rating']).size().unstack()

:

rating	0	1	2	3	4	5
car_type						
1	15079.0	647.0	193.0	533.0	2887.0	28592.0
2	159.0	11.0	4.0	12.0	29.0	428.0
3	954.0	48.0	14.0	31.0	170.0	1268.0
4	14.0	NaN	NaN	NaN	NaN	8.0
5	NaN	NaN	NaN	NaN	NaN	1.0
6	NaN	NaN	NaN	NaN	NaN	1.0

Splitting the data into train and test to applythe models

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(x_pred, y_pred, test_size = 0.2, random_state = 0)

Linear rigression

from sklearn.linear_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X_train, y_train)

regressor.score(X_train, y_train)

Output 0.938157257770777

Random forest

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(max_depth=9, random_state=None,max_features='auto',max_leaf_nodes=6,n_estimators=105,criterion='gini')

model.fit(x, y)

random forest model score

model.score(x,y)
Output 0.6356517823933598

Logistic regression
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 1)

```
classifier.fit(x, y)
y_pred = classifier.predict(x)
y_pred
classifier.score(x,y)
Output  0.5931131687645597
```