Problem Statement

Linear Regression

Import Libraries

In [1]:

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

In [2]:

```
a=pd.read_csv("data.csv")
a
```

Out[2]:

	row_id	user_id	timestamp	gate_id
0	0	18	2022-07-29 09:08:54	7
1	1	18	2022-07-29 09:09:54	9
2	2	18	2022-07-29 09:09:54	9
3	3	18	2022-07-29 09:10:06	5
4	4	18	2022-07-29 09:10:08	5
...	...	...	...	...
37513	37513	6	2022-12-31 20:38:56	11
37514	37514	6	2022-12-31 20:39:22	6
37515	37515	6	2022-12-31 20:39:23	6
37516	37516	6	2022-12-31 20:39:31	9

To display top 10 rows

In [3]:

```
c=a.head(15)
c
```

Out[3]:

	row_id	user_id	timestamp	gate_id
0	0	18	2022-07-29 09:08:54	7
1	1	18	2022-07-29 09:09:54	9
2	2	18	2022-07-29 09:09:54	9
3	3	18	2022-07-29 09:10:06	5
4	4	18	2022-07-29 09:10:08	5
5	5	18	2022-07-29 09:10:34	10
6	6	18	2022-07-29 09:32:47	11
7	7	18	2022-07-29 09:33:12	4
8	8	18	2022-07-29 09:33:13	4
9	9	1	2022-07-29 09:33:16	7
10	10	18	2022-07-29 09:33:23	9
11	11	18	2022-07-29 09:33:23	9
12	12	18	2022-07-29 09:33:41	5
13	13	18	2022-07-29 09:33:42	5
14	14	18	2022-07-29 09:34:04	10

To find Missing values

In [4]:

```
c.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15 entries, 0 to 14
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   row_id     15 non-null     int64 
 1   user_id    15 non-null     int64 
 2   timestamp  15 non-null     object
 3   gate_id    15 non-null     int64 
dtypes: int64(3), object(1)
memory usage: 608.0+ bytes
```

To display summary of statistics

In [5]:

```
a.describe()
```

Out[5]:

	row_id	user_id	gate_id
count	37518.000000	37518.000000	37518.000000
mean	18758.500000	28.219015	6.819607
std	10830.658036	17.854464	3.197746
min	0.000000	0.000000	-1.000000
25%	9379.250000	12.000000	4.000000
50%	18758.500000	29.000000	6.000000
75%	28137.750000	47.000000	10.000000
max	37517.000000	57.000000	16.000000

To display column heading

In [6]:

```
a.columns
```

Out[6]:

```
Index(['row_id', 'user_id', 'timestamp', 'gate_id'], dtype='object')
```

Pairplot

In [7]:

```
s=a.dropna(axis=1)
s
```

Out[7]:

	row_id	user_id	timestamp	gate_id
0	0	18	2022-07-29 09:08:54	7
1	1	18	2022-07-29 09:09:54	9
2	2	18	2022-07-29 09:09:54	9
3	3	18	2022-07-29 09:10:06	5
4	4	18	2022-07-29 09:10:08	5
...	...	...	...	...
37513	37513	6	2022-12-31 20:38:56	11
37514	37514	6	2022-12-31 20:39:22	6
37515	37515	6	2022-12-31 20:39:23	6
37516	37516	6	2022-12-31 20:39:31	9
37517	37517	6	2022-12-31 20:39:31	9

37518 rows × 4 columns

In [9]:

```
s.columns
```

Out[9]:

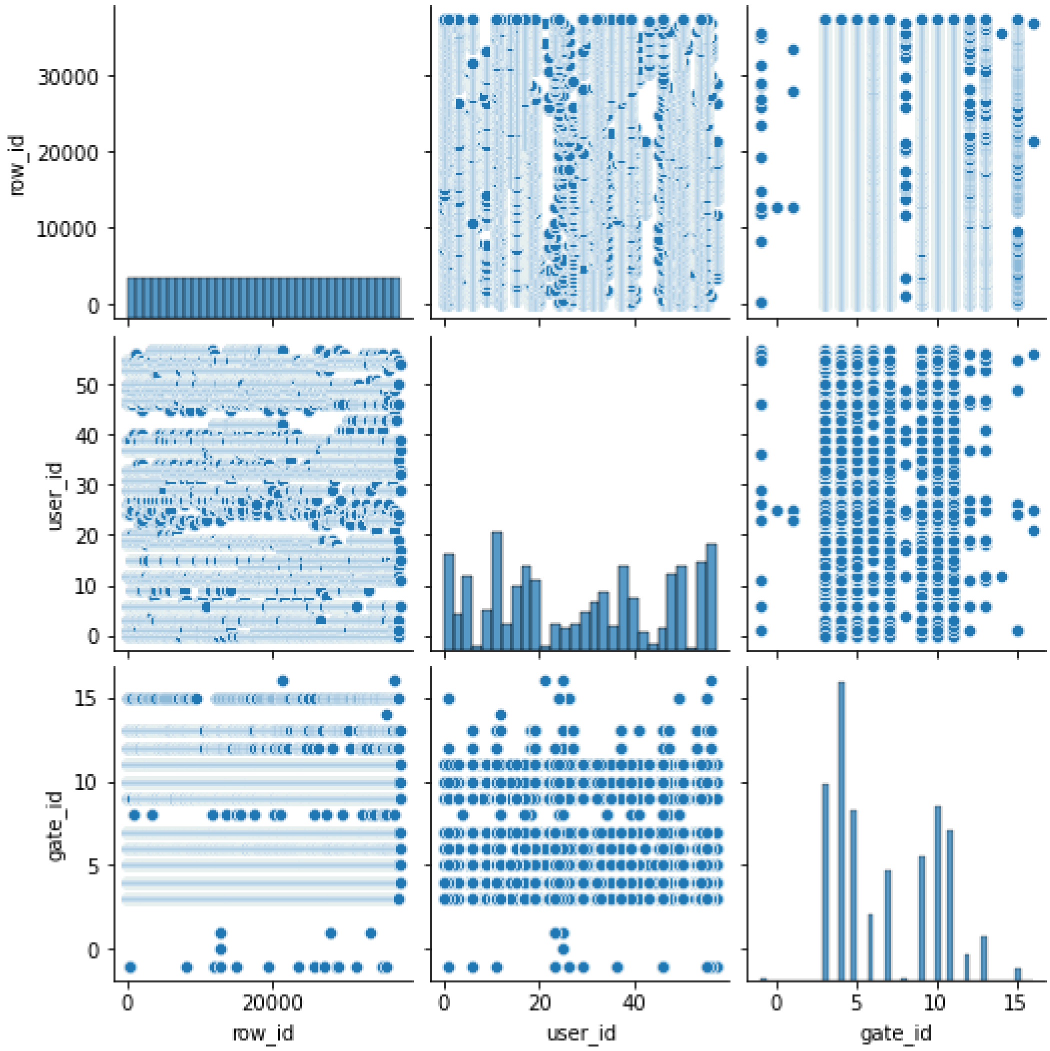
```
Index(['row_id', 'user_id', 'timestamp', 'gate_id'], dtype='object')
```

In [10]:

```
sns.pairplot(a)
```

Out[10]:

```
<seaborn.axisgrid.PairGrid at 0x26d2ec1d7c0>
```



Distribution Plot

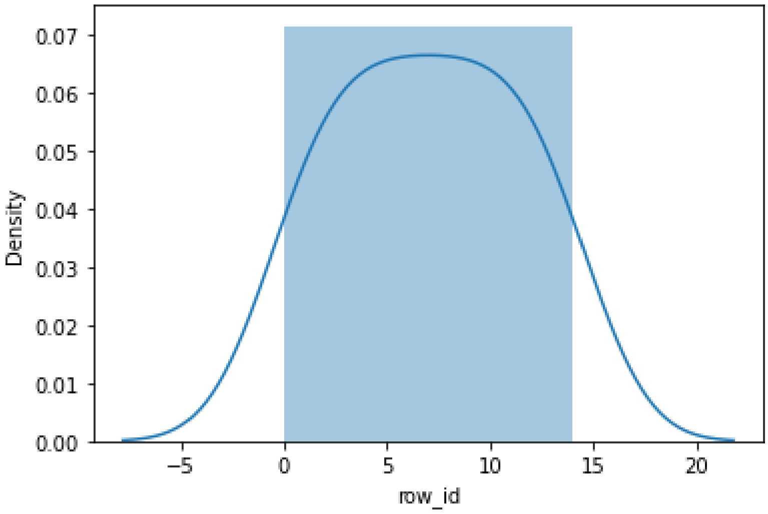
In [12]:

```
sns.distplot(c['row_id'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[12]:

```
<AxesSubplot:xlabel='row_id', ylabel='Density'>
```



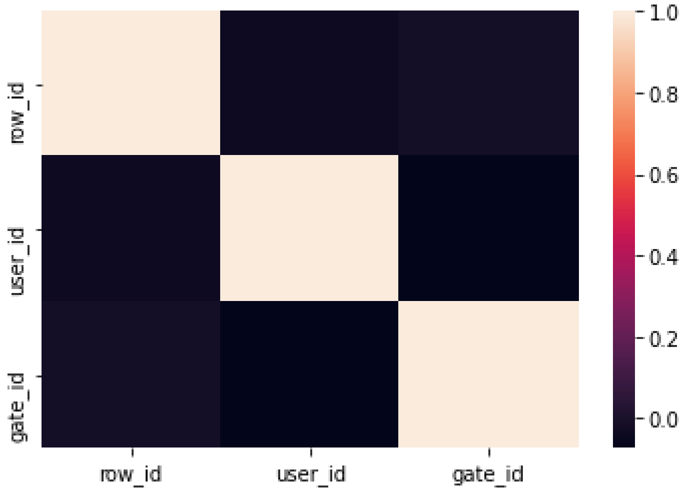
Correlation

In [13]:

```
b=a[['row_id', 'user_id', 'timestamp', 'gate_id']]
sns.heatmap(b.corr())
```

Out[13]:

```
<AxesSubplot:>
```



Train the model - Model Building

In [18]:

```
g=c[['row_id', 'user_id']]
h=c['gate_id']
```

To split dataset into training end test

In [19]:

```
from sklearn.model_selection import train_test_split
g_train,g_test,h_train,h_test=train_test_split(g,h,test_size=0.6)
```

To run the model

In [20]:

```
from sklearn.linear_model import LinearRegression
```

In [21]:

```
lr=LinearRegression()
lr.fit(g_train,h_train)
```

Out[21]:

```
LinearRegression()
```

In [22]:

```
print(lr.intercept_)
```

```
8.75483870967742
```

Coeffecient

In [23]:

```
coeff=pd.DataFrame(lr.coef_,g.columns,columns=['Co-effecient'])
coeff
```

Out[23]:

	Co-effecient
row_id	-0.087097
user_id	0.000000

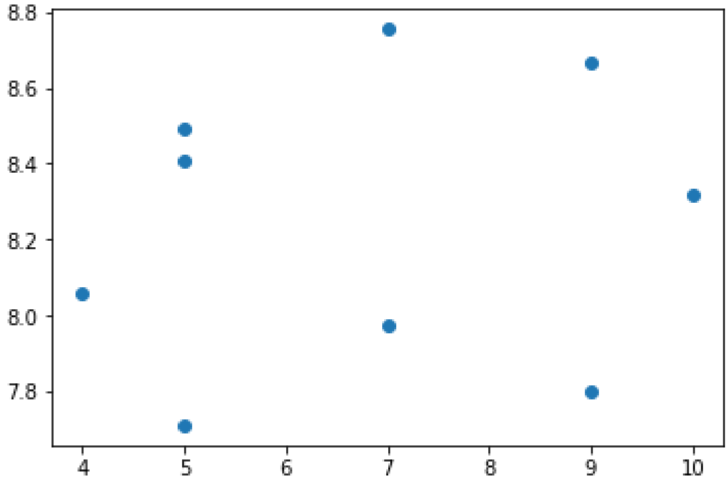
Best Fit line

In [24]:

```
prediction=lr.predict(g_test)
plt.scatter(h_test,prediction)
```

Out[24]:

```
<matplotlib.collections.PathCollection at 0x26d32763ee0>
```



To find score

In [25]:

```
print(lr.score(g_test,h_test))
```

```
-0.49176292569993096
```

Import Lasso and ridge

In [26]:

```
from sklearn.linear_model import Ridge,Lasso
```

Ridge

In [27]:

```
ri=Ridge(alpha=5)
ri.fit(g_train,h_train)
```

Out[27]:

```
Ridge(alpha=5)
```

In [28]:

```
ri.score(g_test,h_test)
```

Out[28]:

```
-0.48364721123210064
```

In [29]:

```
ri.score(g_train,h_train)
```

Out[29]:

```
0.019555029585798578
```

Lasso

In [30]:

```
l=Lasso(alpha=6)
l.fit(g_train,h_train)
```

Out[30]:

```
Lasso(alpha=6)
```

In [31]:

```
l.score(g_test,h_test)
```

Out[31]:

```
-0.3579881656804733
```

In [32]:

```
ri.score(g_train,h_train)
```

Out[32]:

```
0.019555029585798578
```

ElasticNet

In [33]:

```
from sklearn.linear_model import ElasticNet
e=ElasticNet()
e.fit(g_train,h_train)
```

Out[33]:

```
ElasticNet()
```

Coeffecient,intercept

In [34]:

```
print(e.coef_)
```

```
[-0.05642633  0.        ]
```

In [35]:

```
print(e.intercept_)
```

```
8.489028213166144
```

Prediction

In [36]:

```
c=e.predict(g_test)
```

To calculate Score

In [37]:

```
print(e.score(g_test,h_test))
```

```
-0.4345940473469301
```

Evaluation

In [38]:

```
from sklearn import metrics
```

In [39]:

```
print("Mean Absolute Error",metrics.mean_absolute_error(h_test,c))
```

```
Mean Absolute Error 2.1549982584465344
```

In [40]:

```
print("Mean Squared Error",metrics.mean_squared_error(h_test,c))
```

```
Mean Squared Error 5.986330716089658
```

In [41]:

```
print("Root Mean Squared Error",np.sqrt(metrics.mean_squared_error(h_test,c)))
```

```
Root Mean Squared Error 2.4466979208904513
```

import Libraries

In [42]:

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

import Linear Regression

In [43]:

```
from sklearn.linear_model import LogisticRegression
```

In [44]:

```
lgr=LogisticRegression()
```

Select Required data from certain columns

In [45]:

```
a=pd.read_csv("data.csv")
a
```

Out[45]:

	row_id	user_id	timestamp	gate_id
0	0	18	2022-07-29 09:08:54	7
1	1	18	2022-07-29 09:09:54	9
2	2	18	2022-07-29 09:09:54	9
3	3	18	2022-07-29 09:10:06	5
4	4	18	2022-07-29 09:10:08	5
...	...	...	...	...
37513	37513	6	2022-12-31 20:38:56	11
37514	37514	6	2022-12-31 20:39:22	6
37515	37515	6	2022-12-31 20:39:23	6
37516	37516	6	2022-12-31 20:39:31	9
37517	37517	6	2022-12-31 20:39:31	9

37518 rows × 4 columns

In [46]:

```
c=a.dropna()
c
```

Out[46]:

	row_id	user_id	timestamp	gate_id
0	0	18	2022-07-29 09:08:54	7
1	1	18	2022-07-29 09:09:54	9
2	2	18	2022-07-29 09:09:54	9
3	3	18	2022-07-29 09:10:06	5
4	4	18	2022-07-29 09:10:08	5
...	...	...	...	...
37513	37513	6	2022-12-31 20:38:56	11
37514	37514	6	2022-12-31 20:39:22	6
37515	37515	6	2022-12-31 20:39:23	6
37516	37516	6	2022-12-31 20:39:31	9
37517	37517	6	2022-12-31 20:39:31	9

37518 rows × 4 columns

In [47]:

```
c.columns
```

Out[47]:

```
Index(['row_id', 'user_id', 'timestamp', 'gate_id'], dtype='object')
```

In [48]:

```
fm=c[['row_id', 'user_id']]
tv=c[[ 'gate_id']]
```

Shape

In [49]:

```
fm.shape
```

Out[49]:

```
(37518, 2)
```

In [50]:

```
tv.shape
```

Out[50]:

```
(37518, 1)
```

To make the data in order (feature matrix)

In [51]:

```
from sklearn.preprocessing import StandardScaler
```

In [52]:

```
fs=StandardScaler().fit_transform(fm)
```

Imply Logistic Regression

In [53]:

```
lgr.fit(fm,tv)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was ex
pected. Please change the shape of y to (n_samples, ), for example using r
avel().
  return f(*args, **kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.
py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

Out[53]:

```
LogisticRegression()
```

Prediction

In [54]:

```
ab=[[3,90]]
```

In [55]:

```
pre=lgr.predict(ab)
```

In [56]:

```
print(pre)
```

```
[4]
```

To check the output var we have got

In [57]:

```
lgr.classes_
```

Out[57]:

```
array([-1,  0,  1,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 1
6],
      dtype=int64)
```

Prediction in Probablity value

In [58]:

```
lgr.predict_proba(ab)[0][1]
```

Out[58]:

```
0.057990227855722241
```

In [ ]:

```

```