D4

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df=pd.read_csv(r"C:\Users\user\Downloads\6_Salesworkload1.csv")
df
```

Out[2]:

	MonthYear	Time index	Country	StoreID	City	Dept_ID	Dept. Name	HoursOwn	HoursLe
0	10.2016	1.0	United Kingdom	88253.0	London (I)	1.0	Dry	3184.764	
1	10.2016	1.0	United Kingdom	88253.0	London (I)	2.0	Frozen	1582.941	
2	10.2016	1.0	United Kingdom	88253.0	London (I)	3.0	other	47.205	
3	10.2016	1.0	United Kingdom	88253.0	London (I)	4.0	Fish	1623.852	
4	10.2016	1.0	United Kingdom	88253.0	London (I)	5.0	Fruits & Vegetables	1759.173	
...	...	...	...	...	...	...	...	...	
7653	06.2017	9.0	Sweden	29650.0	Gothenburg	12.0	Checkout	6322.323	
7654	06.2017	9.0	Sweden	29650.0	Gothenburg	16.0	Customer Services	4270.479	
7655	06.2017	9.0	Sweden	29650.0	Gothenburg	11.0	Delivery	0	
7656	06.2017	9.0	Sweden	29650.0	Gothenburg	17.0	others	2224.929	
7657	06.2017	9.0	Sweden	29650.0	Gothenburg	18.0	all	39652.2	

7658 rows × 14 columns

In [3]:
```
df.head(10)
```

Out[3]:

	MonthYear	Time index	Country	StoreID	City	Dept_ID	Dept. Name	HoursOwn	HoursLease	
0	10.2016	1.0	United Kingdom	88253.0	London (I)	1.0	Dry	3184.764	0.0	3
1	10.2016	1.0	United Kingdom	88253.0	London (I)	2.0	Frozen	1582.941	0.0	
2	10.2016	1.0	United Kingdom	88253.0	London (I)	3.0	other	47.205	0.0	4
3	10.2016	1.0	United Kingdom	88253.0	London (I)	4.0	Fish	1623.852	0.0	3
4	10.2016	1.0	United Kingdom	88253.0	London (I)	5.0	Fruits & Vegetables	1759.173	0.0	1
5	10.2016	1.0	United Kingdom	88253.0	London (I)	6.0	Meat	8270.316	0.0	17
6	10.2016	1.0	United Kingdom	88253.0	London (I)	13.0	Food	16468.251	0.0	31
7	10.2016	1.0	United Kingdom	88253.0	London (I)	7.0	Clothing	4698.471	0.0	2
8	10.2016	1.0	United Kingdom	88253.0	London (I)	8.0	Household	1183.272	0.0	
9	10.2016	1.0	United Kingdom	88253.0	London (I)	9.0	Hardware	2029.815	0.0	

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7658 entries, 0 to 7657
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   MonthYear      7658 non-null   object 
 1   Time index     7650 non-null   float64
 2   Country        7650 non-null   object 
 3   StoreID        7650 non-null   float64
 4   City           7650 non-null   object 
 5   Dept_ID        7650 non-null   float64
 6   Dept. Name     7650 non-null   object 
 7   HoursOwn       7650 non-null   object 
 8   HoursLease     7650 non-null   float64
 9   Sales units    7650 non-null   float64
 10  Turnover       7650 non-null   float64
 11  Customer       0 non-null      float64
 12  Area (m2)      7650 non-null   object 
 13  Opening hours  7650 non-null   object 
dtypes: float64(7), object(7)
memory usage: 837.7+ KB
```

In [5]:
```
dff=df.drop("Customer",axis=1)
```

In [6]:
```
dft=dff.dropna()
```

In [7]:
```
dft.isnull().sum()
```

Out[7]:
```
MonthYear        0
Time index       0
Country          0
StoreID          0
City             0
Dept_ID          0
Dept. Name       0
HoursOwn         0
HoursLease       0
Sales units      0
Turnover         0
Area (m2)        0
Opening hours    0
dtype: int64
```

In [8]:
```
dft.describe()
```

Out[8]:

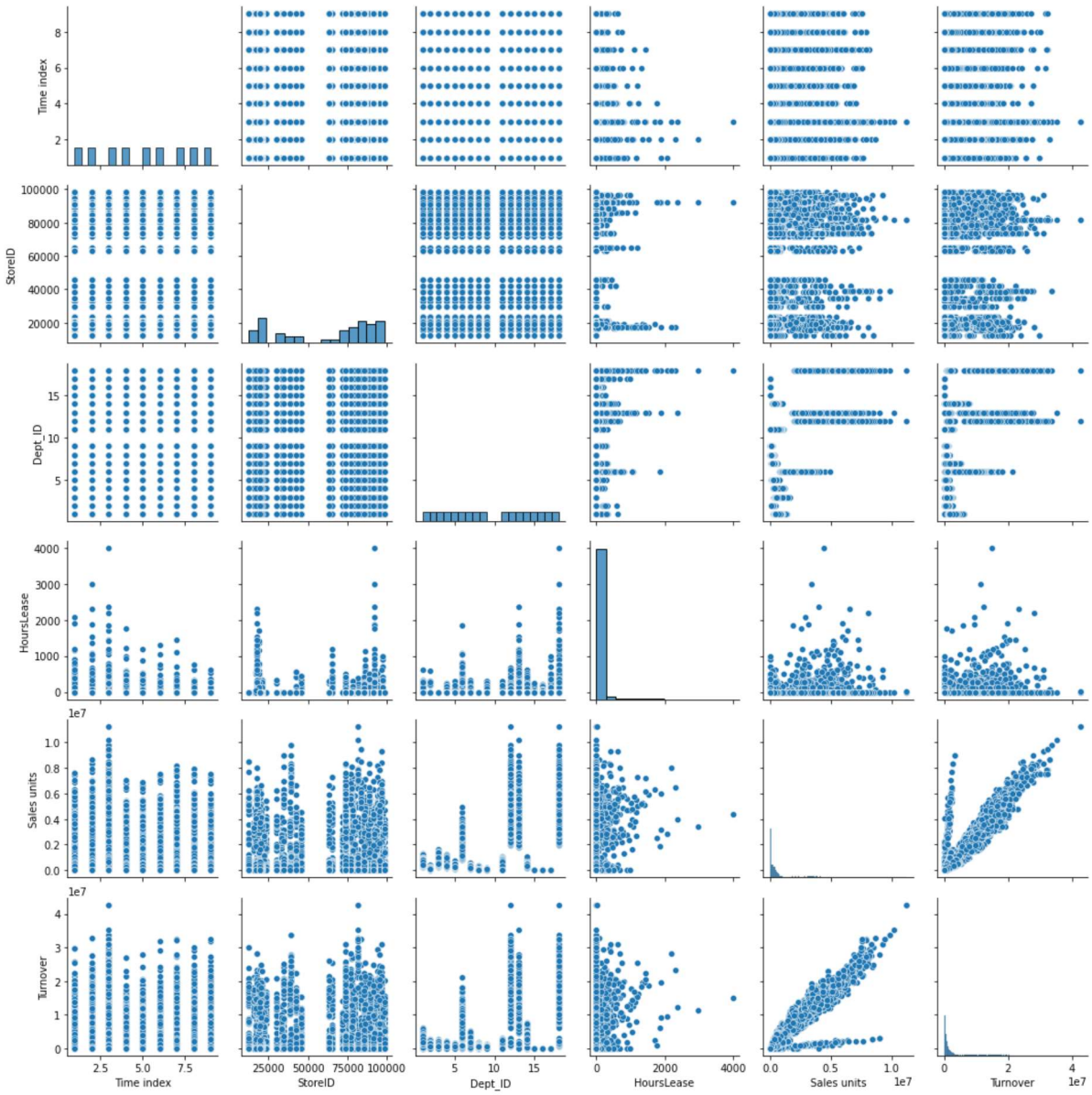
	Time index	StoreID	Dept_ID	HoursLease	Sales units	Turnover
count	7650.000000	7650.000000	7650.000000	7650.000000	7.650000e+03	7.650000e+03
mean	5.000000	61995.220000	9.470588	22.036078	1.076471e+06	3.721393e+06
std	2.582158	29924.581631	5.337429	133.299513	1.728113e+06	6.003380e+06
min	1.000000	12227.000000	1.000000	0.000000	0.000000e+00	0.000000e+00
25%	3.000000	29650.000000	5.000000	0.000000	5.457125e+04	2.726798e+05
50%	5.000000	75400.500000	9.000000	0.000000	2.932300e+05	9.319575e+05
75%	7.000000	87703.000000	14.000000	0.000000	9.175075e+05	3.264432e+06
max	9.000000	98422.000000	18.000000	3984.000000	1.124296e+07	4.271739e+07

In [9]:
```
dft.columns
```

Out[9]:
```
Index(['MonthYear', 'Time index', 'Country', 'StoreID', 'City', 'Dept_ID',
       'Dept. Name', 'HoursOwn', 'HoursLease', 'Sales units', 'Turnover',
       'Area (m2)', 'Opening hours'],
      dtype='object')
```

In [10]:
```
sns.pairplot(dft)
```

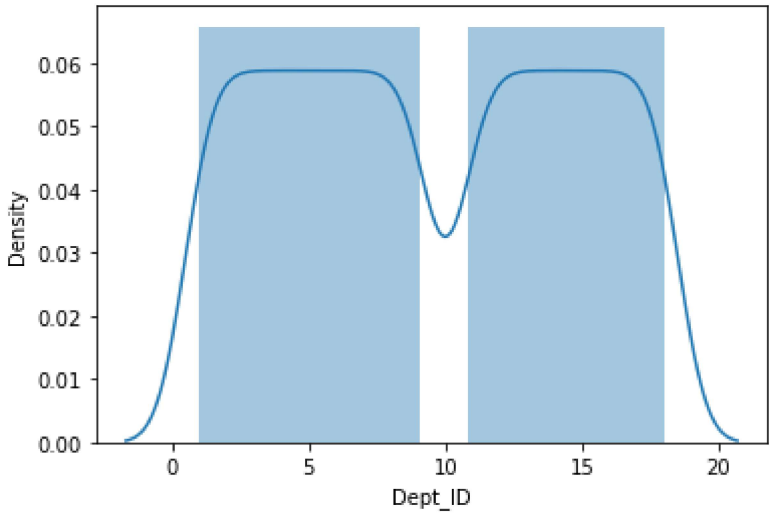
Out[10]:
```
<seaborn.axisgrid.PairGrid at 0x1f8c79d8e50>
```



In [11]:
```
sns.distplot(dft["Dept_ID"])
```

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

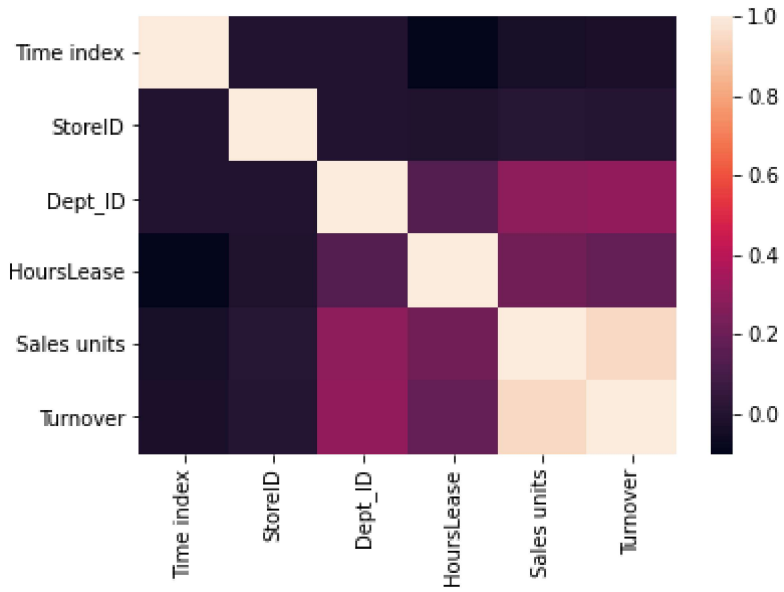
Out[11]: <AxesSubplot:xlabel='Dept_ID', ylabel='Density'>



In [12]:
```
df1=dft[['MonthYear', 'Time index', 'Country', 'StoreID', 'City', 'Dept_ID',
       'Dept. Name', 'HoursOwn', 'HoursLease', 'Sales units', 'Turnover',
       'Area (m2)', 'Opening hours']]
```

In [13]:
```
sns.heatmap(df1.corr())
```

Out[13]: <AxesSubplot:>



In [14]:
```
x=df1[['Time index', 'StoreID', 'HoursLease', 'Sales units', 'Turnover']]
y=df1['Dept_ID']
```

In [15]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [16]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[16]:
```
LinearRegression()
```

In [17]:
```
print(lr.intercept_)
```

```
8.346262916345273
```

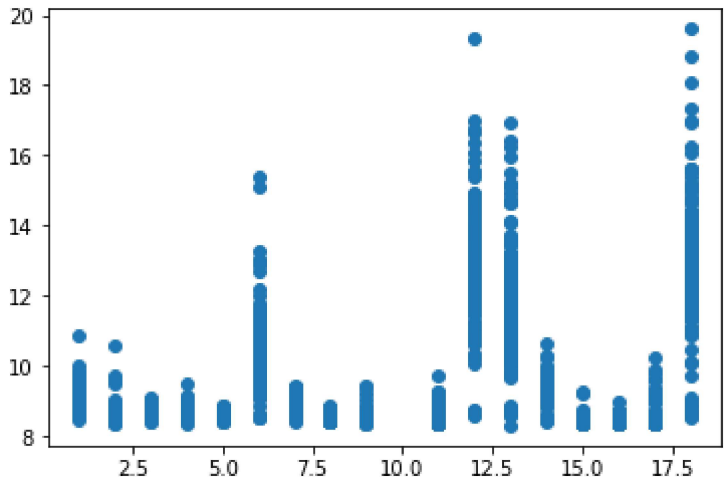
In [18]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[18]:

	Co-efficient
Time index	3.245193e-02
StoreID	-4.843904e-07
HoursLease	3.290617e-03
Sales units	-3.119062e-08
Turnover	2.638055e-07

In [19]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[19]:
```
<matplotlib.collections.PathCollection at 0x1f8cb395d30>
```



In [20]:
```
print(lr.score(x_test,y_test))
```

```
0.09586423055598048
```

In [21]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [22]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[22]:
```
Ridge(alpha=10)
```

In [23]:
```
rr.score(x_test,y_test)
```

Out[23]:
```
0.09586428064648211
```

In [24]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[24]:
```
Lasso(alpha=10)
```

In [25]:
```
la.score(x_test,y_test)
```

Out[25]:
```
0.09552142341867043
```

In [26]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[26]:
```
ElasticNet()
```

In [27]:
```
print(en.coef_)
```

```
[ 0.00000000e+00 -4.76233571e-07  3.20741172e-03 -3.49800204e-08
  2.64969765e-07]
```

In [28]:
```
print(en.intercept_)
```

```
8.509263922567214
```

In [29]:
```
print(en.predict(x_train))
```

```
[ 8.46381312 10.43454115  9.56545058 ... 11.63134792  8.46238542
  8.55956468]
```

In [30]:
```
print(en.score(x_train,y_train))
```

```
0.09708291943663849
```

In [31]:
```
from sklearn import metrics
```

In [32]:
```
print("Mean Absolytre Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolytre Error: 4.3429965832306205
```

In [33]:
```
print("Mean Square Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Square Error: 25.805095802518128
```

In [34]:
```
print("Root Mean Square Error:",np.sqrt(metrics.mean_absolute_error(y_test,pre
```

```
Root Mean Square Error: 2.08398574448834
```

In [ ]:
```

```