Index

Importing the Libraries

In [28]:
```
import pandas as pd
from scipy.stats import skew
import numpy as np
import scipy.stats
import pylab
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
%matplotlib inline
sns.set()
pd.set_option('display.float_format', lambda x: '%.3f' % x)
```

Data Preparation/Extraction/Cleaning

In [29]:
```
df=pd.read_csv('Task1_Data.csv')
df.head()
```

Out[29]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PU
0	C10001	40.901	0.818	95.400	0.000	95.400	0.000	
1	C10002	3202.467	0.909	0.000	0.000	0.000	6442.945	
2	C10003	2495.149	1.000	773.170	773.170	0.000	0.000	
3	C10004	1666.671	0.636	1499.000	1499.000	0.000	205.788	
4	C10005	817.714	1.000	16.000	16.000	0.000	0.000	

In [30]:
```
df.shape
```

Out[30]: (8950, 18)

In [31]:
```
df.isnull().sum()
```

Out[31]:
```
CUST_ID                              0
BALANCE                              0
BALANCE_FREQUENCY                    0
PURCHASES                            0
ONEOFF_PURCHASES                     0
INSTALLMENTS_PURCHASES               0
CASH_ADVANCE                         0
PURCHASES_FREQUENCY                  0
ONEOFF_PURCHASES_FREQUENCY           0
PURCHASES_INSTALLMENTS_FREQUENCY     0
CASH_ADVANCE_FREQUENCY               0
CASH_ADVANCE_TRX                     0
PURCHASES_TRX                        0
CREDIT_LIMIT                         1
PAYMENTS                             0
MINIMUM_PAYMENTS                   313
PRC_FULL_PAYMENT                     0
TENURE                               0
dtype: int64
```

In [32]:
```
df['MINIMUM_PAYMENTS'].fillna(0,inplace=True)
df['CREDIT_LIMIT'].fillna(0,inplace=True)
```

In [33]:
```
df.columns
```

Out[33]:
```
Index(['CUST_ID', 'BALANCE', 'BALANCE_FREQUENCY', 'PURCHASES',
       'ONEOFF_PURCHASES', 'INSTALLMENTS_PURCHASES', 'CASH_ADVANCE',
       'PURCHASES_FREQUENCY', 'ONEOFF_PURCHASES_FREQUENCY',
       'PURCHASES_INSTALLMENTS_FREQUENCY', 'CASH_ADVANCE_FREQUENCY',
       'CASH_ADVANCE_TRX', 'PURCHASES_TRX', 'CREDIT_LIMIT', 'PAYMENTS',
       'MINIMUM_PAYMENTS', 'PRC_FULL_PAYMENT', 'TENURE'],
      dtype='object')
```

EDA

Overall Data Statistics

In [34]:
```
df.describe()
```

Out[34]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHA
count	8950.000	8950.000	8950.000	8950.000	8950.000	8950.000	
mean	1564.475	0.877	1003.205	592.437	411.068	978.871	
std	2081.532	0.237	2136.635	1659.888	904.338	2097.164	
min	0.000	0.000	0.000	0.000	0.000	0.000	
25%	128.282	0.889	39.635	0.000	0.000	0.000	
50%	873.385	1.000	361.280	38.000	89.000	0.000	
75%	2054.140	1.000	1110.130	577.405	468.637	1113.821	
max	19043.139	1.000	49039.570	40761.250	22500.000	47137.212	

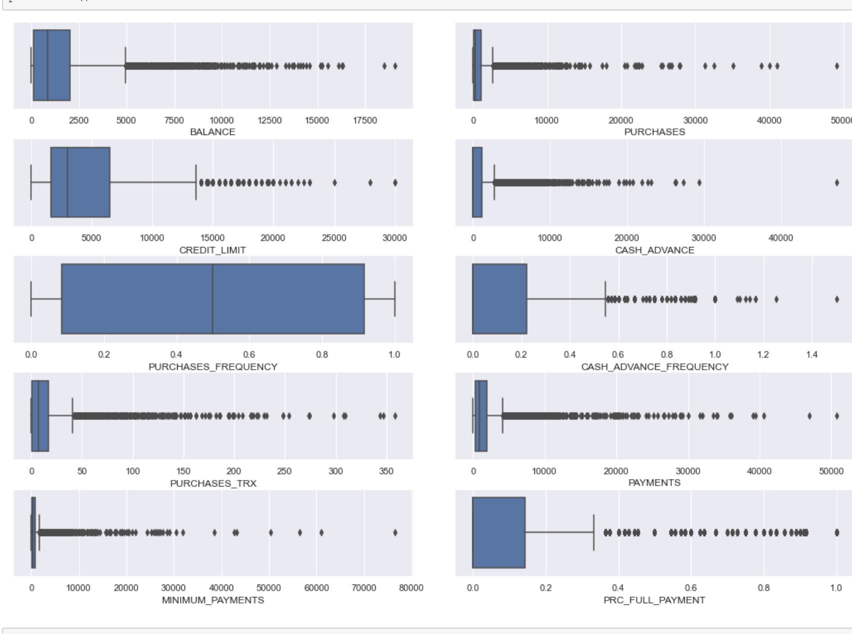
In [35]:
```
data=pd.DataFrame(columns=['Column Name','Normal Mean','Trimmed Mean','Normal Median','Outliers or Extreme Values Effect'])
for col in list(df.columns)[1:]:
    val1=float(df[col].mean())
    val2=float(stats.trim_mean(df[col], 0.1))
    val3=float(df[col].median())
    val4=(abs(val2-val1)/val1)*100
    data.loc[len(data)]=[col,val1,val2,val3,val4]

data
```

Out[35]:

	Column Name	Normal Mean	Trimmed Mean	Normal Median	Outliers or Extreme Values Effect
0	BALANCE	1564.475	1128.165	873.385	27.889
1	BALANCE_FREQUENCY	0.877	0.938	1.000	6.935
2	PURCHASES	1003.205	583.100	361.280	41.876
3	ONEOFF_PURCHASES	592.437	263.816	38.000	55.469
4	INSTALLMENTS_PURCHASES	411.068	223.032	89.000	45.743
5	CASH_ADVANCE	978.871	494.902	0.000	49.442
6	PURCHASES_FREQUENCY	0.490	0.488	0.500	0.492
7	ONEOFF_PURCHASES_FREQUENCY	0.202	0.137	0.083	32.225
8	PURCHASES_INSTALLMENTS_FREQUENCY	0.364	0.331	0.167	9.299
9	CASH_ADVANCE_FREQUENCY	0.135	0.092	0.000	31.584
10	CASH_ADVANCE_TRX	3.249	1.729	0.000	46.784
11	PURCHASES_TRX	14.710	9.433	7.000	35.871
12	CREDIT_LIMIT	4493.947	3926.747	3000.000	12.621
13	PAYMENTS	1733.144	1152.895	856.902	33.480
14	MINIMUM_PAYMENTS	833.983	463.582	289.628	44.414
15	PRC_FULL_PAYMENT	0.154	0.077	0.000	49.825
16	TENURE	11.517	11.919	12.000	3.491

Data Distribution

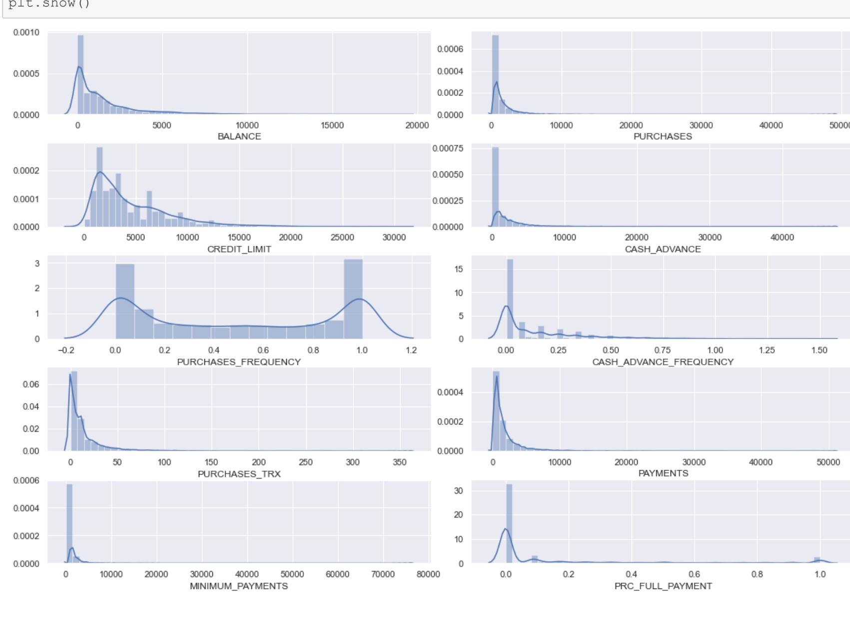
In [36]:
```
fig, ax = plt.subplots(5, 2)
# Setting height and width of subplots
fig.set_figheight(15)
fig.set_figwidth(15)
# Adding spacing between boxes
fig.tight_layout(h_pad=True, w_pad=True)
sns.boxplot(df["BALANCE"], ax=ax[0, 0])
sns.boxplot(df["PURCHASES"], ax=ax[0, 1])
sns.boxplot(df["CREDIT_LIMIT"], ax=ax[1, 0])
sns.boxplot(df["CASH_ADVANCE"], ax=ax[1, 1])
sns.boxplot(df["PURCHASES_FREQUENCY"], ax=ax[2, 0])
sns.boxplot(df["CASH_ADVANCE_FREQUENCY"], ax=ax[2, 1])
sns.boxplot(df["PURCHASES_TRX"], ax=ax[3, 0])
sns.boxplot(df["PAYMENTS"], ax=ax[3, 1])
sns.boxplot(df["MINIMUM_PAYMENTS"], ax=ax[4, 0])
sns.boxplot(df["PRC_FULL_PAYMENT"], ax=ax[4, 1])
plt.show()
```



In [37]:
```
fig, ax = plt.subplots(5, 2)
# Setting height and width of subplots
fig.set_figheight(10)
fig.set_figwidth(15)
# Adding spacing between boxes
fig.tight_layout(h_pad=True, w_pad=True)
sns.scatterplot(data=df["BALANCE"], ax=ax[0, 0])
sns.scatterplot(data=df["PURCHASES"], ax=ax[0, 1])
sns.scatterplot(data=df["CREDIT_LIMIT"], ax=ax[1, 0])
sns.scatterplot(data=df["CASH_ADVANCE"], ax=ax[1, 1])
sns.scatterplot(data=df["PURCHASES_FREQUENCY"], ax=ax[2, 0])
sns.scatterplot(data=df["CASH_ADVANCE_FREQUENCY"], ax=ax[2, 1])
sns.scatterplot(data=df["PURCHASES_TRX"], ax=ax[3, 0])
sns.scatterplot(data=df["PAYMENTS"], ax=ax[3, 1])
sns.scatterplot(data=df["MINIMUM_PAYMENTS"], ax=ax[4, 0])
sns.scatterplot(data=df["PRC_FULL_PAYMENT"], ax=ax[4, 1])
plt.show()
```

Skewness

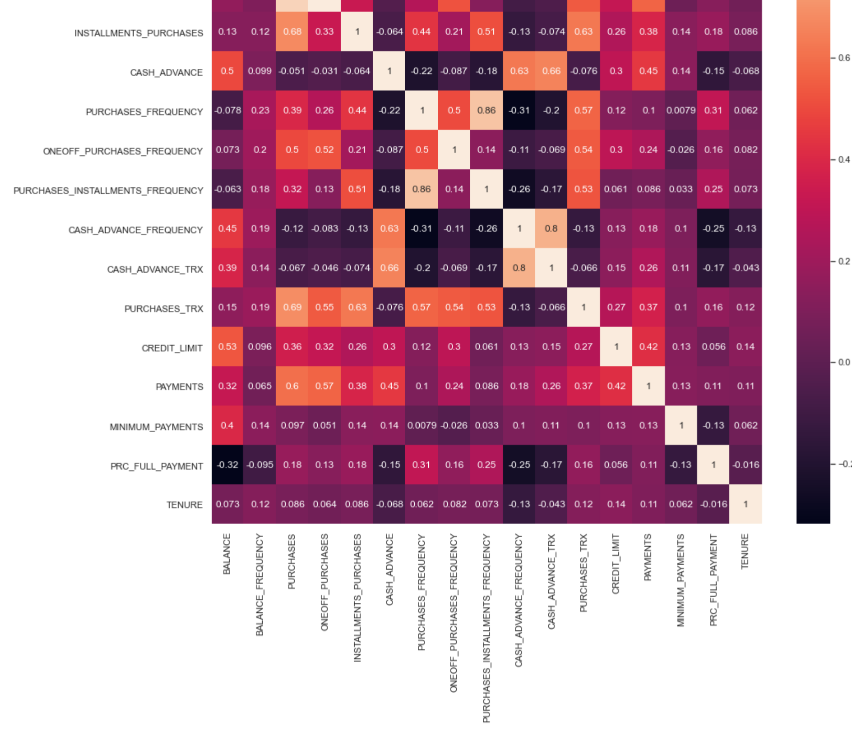
In [38]:
```
fig, ax = plt.subplots(5, 2)
# Setting height and width of subplots
fig.set_figheight(10)
fig.set_figwidth(15)
# Adding spacing between boxes
fig.tight_layout(h_pad=True, w_pad=True)
sns.distplot(df["BALANCE"], ax=ax[0, 0])
sns.distplot(df["PURCHASES"], ax=ax[0, 1])
sns.distplot(df["CREDIT_LIMIT"], ax=ax[1, 0])
sns.distplot(df["CASH_ADVANCE"], ax=ax[1, 1])
sns.distplot(df["PURCHASES_FREQUENCY"], ax=ax[2, 0])
sns.distplot(df["CASH_ADVANCE_FREQUENCY"], ax=ax[2, 1])
sns.distplot(df["PURCHASES_TRX"], ax=ax[3, 0])
sns.distplot(df["PAYMENTS"], ax=ax[3, 1])
sns.distplot(df["MINIMUM_PAYMENTS"], ax=ax[4, 0])
sns.distplot(df["PRC_FULL_PAYMENT"], ax=ax[4, 1])
plt.show()
```



Correlation

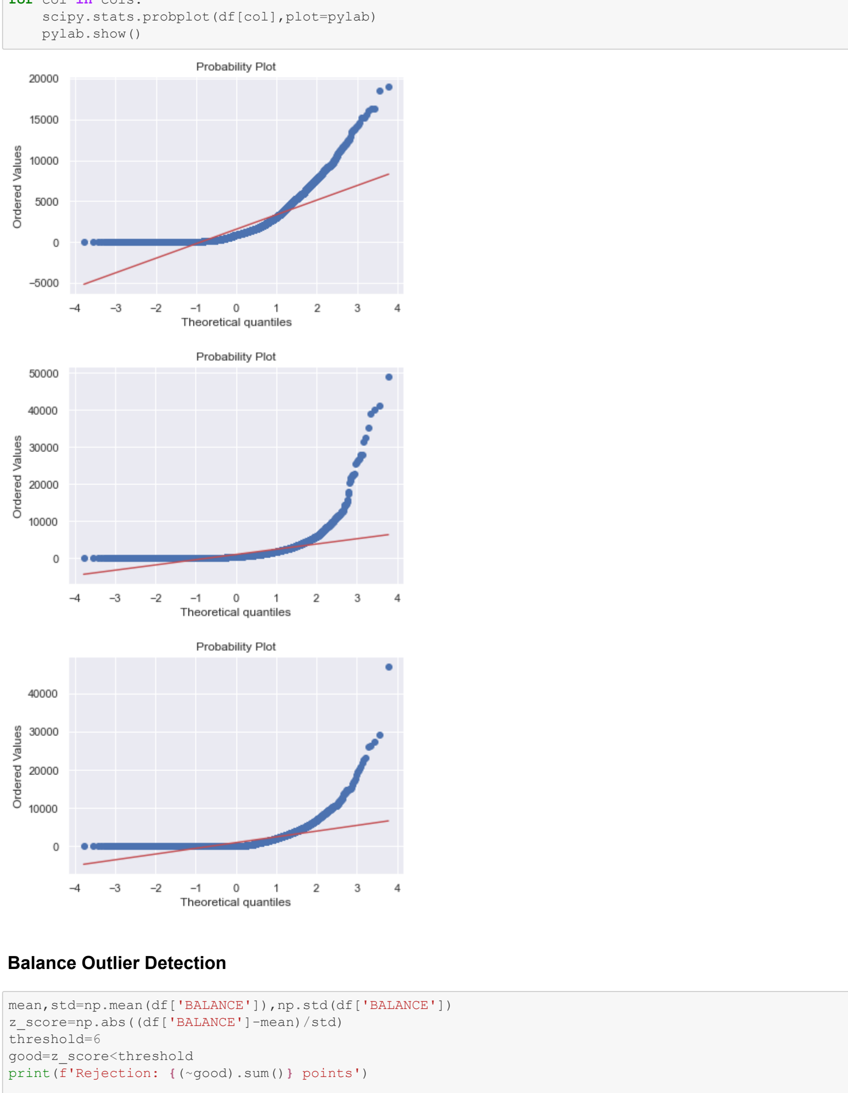
In [39]:
```
# Looking at mulitcollinearity
plt.figure(figsize=(15,15))
sns.heatmap(df.corr(),annot=True)
```

Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0x20f9d66160>



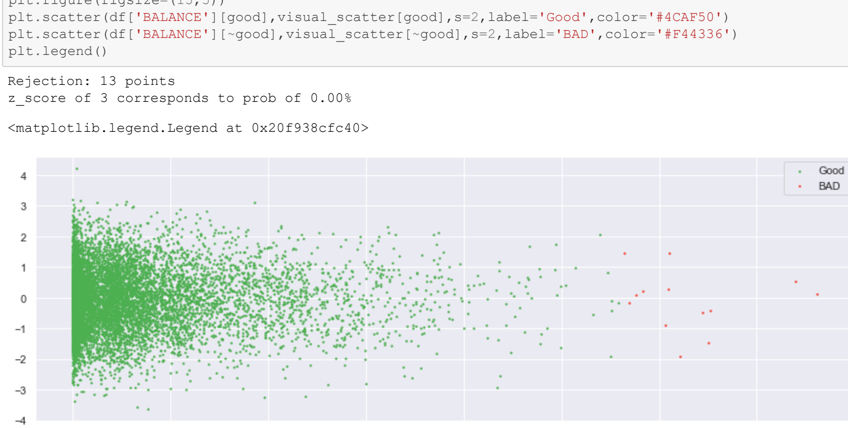
Q-Q Plot

[illegible]



Balance Outlier Detection

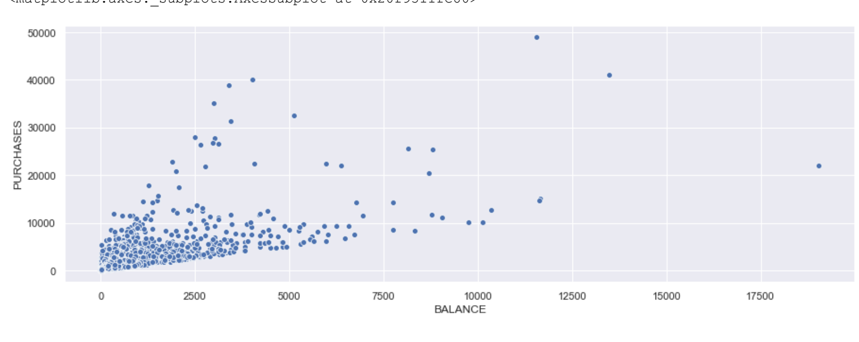
[illegible]



Customers with more purchase than balance

[illegible]

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x20f9311fe80>



Customer with less purchase than balance

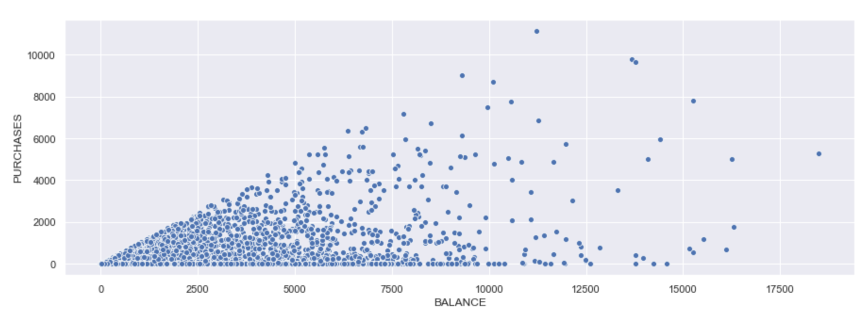
In [44]:
```
data=df[df.apply(lambda x:x['BALANCE']-x['PURCHASES']>0,axis=1)]
data.head()
```

Out[44]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PU
1	C10002	3202.467	0.909	0.000	0.000	0.000	6442.945	
2	C10003	2495.149	1.000	773.170	773.170	0.000	0.000	
3	C10004	1666.671	0.636	1499.000	1499.000	0.000	205.788	
4	C10005	817.714	1.000	16.000	16.000	0.000	0.000	
5	C10006	1809.829	1.000	1333.280	0.000	1333.280	0.000	

In [45]:
```
plt.figure(figsize=(15,5))
sns.scatterplot(data['BALANCE'],data['PURCHASES'])
```

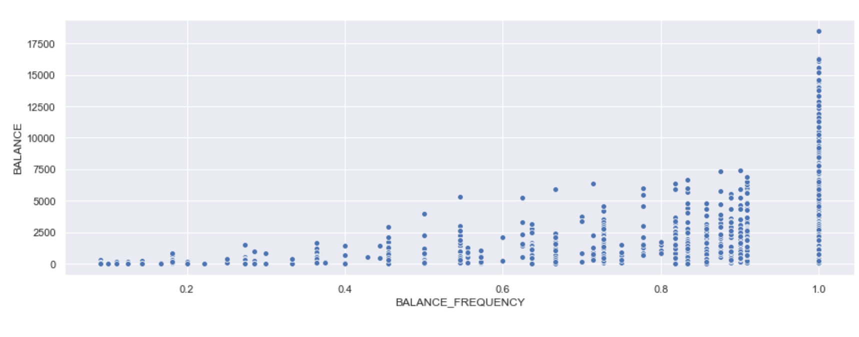
Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x20f9287b190>



Balance Vs Balance Frequency

In [46]:
```
plt.figure(figsize=(15,5))
sns.scatterplot(data['BALANCE_FREQUENCY'],data['BALANCE'])
```

Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x20f93250760>

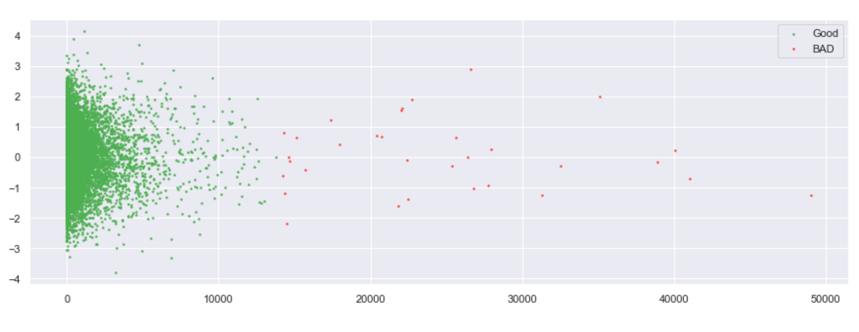


Purchase Outliers

In [47]:
```
mean,std=np.mean(df['PURCHASES']),np.std(df['PURCHASES'])
z_score=np.abs((df['PURCHASES']-mean)/std)
threshold=6
good=z_score<threshold
print(f'Rejection: {(~good).sum()} points')

from scipy.stats import norm
print(f"z_score of 3 corresponds to prob of {100*2*norm.sf(threshold):0.2f}%")
visual_scatter=np.random.normal(size=df['PURCHASES'].size)

plt.figure(figsize=(15,5))
plt.scatter(df['PURCHASES'][good],visual_scatter[good],s=2,label='Good',color='#4CAF50')
plt.scatter(df['PURCHASES'][~good],visual_scatter[~good],s=2,label='BAD',color='#F44336')
plt.legend()
```

```
Rejection: 32 points
z_score of 3 corresponds to prob of 0.00%
```

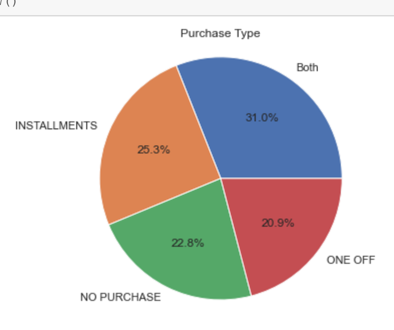
Out[47]: <matplotlib.legend.Legend at 0x20f94ca0670>



Purchase Types

In [48]:
```
def f(x):
    if((x['ONEOFF_PURCHASES']!=0) & (x['INSTALLMENTS_PURCHASES']!=0)):
        val='Both'
    elif(x['ONEOFF_PURCHASES']!=0):
        val='ONE OFF'
    elif(x['INSTALLMENTS_PURCHASES']!=0):
        val = 'INSTALLMENTS'
    else:
        val='NO PURCHASE'
    return val

df['PURCHASE_TYPE'] = df.apply(f,axis=1)
```

In [49]:
```
vals=list(df['PURCHASE_TYPE'].value_counts().values)
labels=list(df['PURCHASE_TYPE'].value_counts().index)
plt.figure(figsize=(10,5))
plt.pie(vals,labels=labels,autopct='%1.1f%%')
plt.title('Purchase Type')
plt.axis('equal')
plt.show()
```



In [50]:
```
print("Average Amount Payment Through ONEOFF: ",df['ONEOFF_PURCHASES'].mean())
print("Average Amount Payment Through INSTALLMENTS: ",df['INSTALLMENTS_PURCHASES'].mean())
```

```
Average Amount Payment Through ONEOFF:   592.437370949722
Average Amount Payment Through INSTALLMENTS:   411.06764469273713
```

Customer Opting Both ONEOFF and INSTALLMENTS

In [51]:
```
data=df[df['PURCHASE_TYPE']=='Both']
data['ONEOFF_PURCHASES_Ratio']=data['ONEOFF_PURCHASES']/data['PURCHASES']
data['INSTALLMENTS_PURCHASES_Ratio']=data['INSTALLMENTS_PURCHASES']/data['PURCHASES']
```

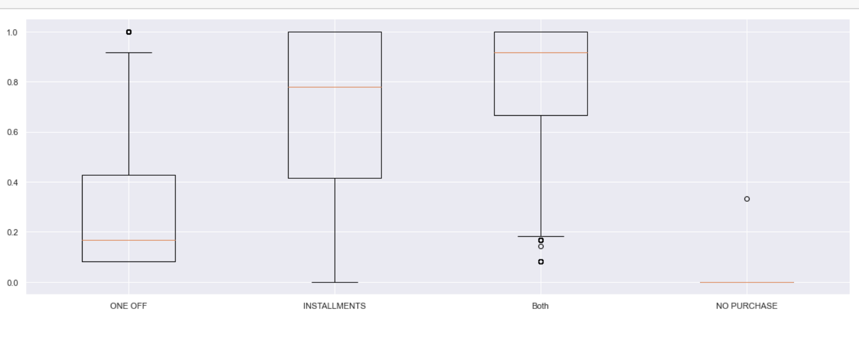
```
<ipython-input-51-89324664e7db>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
g.html#returning-a-view-versus-a-copy
  data['ONEOFF_PURCHASES_Ratio']=data['ONEOFF_PURCHASES']/data['PURCHASES']
<ipython-input-51-89324664e7db>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
g.html#returning-a-view-versus-a-copy
  data['INSTALLMENTS_PURCHASES_Ratio']=data['INSTALLMENTS_PURCHASES']/data['PURCHASES']
```

In [52]:
```
print("Average % Payment Through ONEOFF: ",round(data['ONEOFF_PURCHASES_Ratio'].mean()*100,1),'%')
print("Average % Payment Through INSTALLMENTS: ",round(data['INSTALLMENTS_PURCHASES_Ratio'].mean()*100,
1),'%')
```

```
Average % Payment Through ONEOFF:   54.9 %
Average % Payment Through INSTALLMENTS:   45.1 %
```

Purchase Frequency

In [53]:
```
data = [df[df['PURCHASE_TYPE']=='ONE OFF']["PURCHASES_FREQUENCY"],
        df[df['PURCHASE_TYPE']=='INSTALLMENTS']["PURCHASES_FREQUENCY"],
        df[df['PURCHASE_TYPE']=='Both']["PURCHASES_FREQUENCY"],
        df[df['PURCHASE_TYPE']=='NO PURCHASE']["PURCHASES_FREQUENCY"]]

fig = plt.figure(figsize =(15, 5))
ax = fig.add_axes([0, 0, 1, 1])
ax.set_xticklabels(['ONE OFF', 'INSTALLMENTS',
                    'Both', 'NO PURCHASE'])
bp = ax.boxplot(data)

plt.show()
```

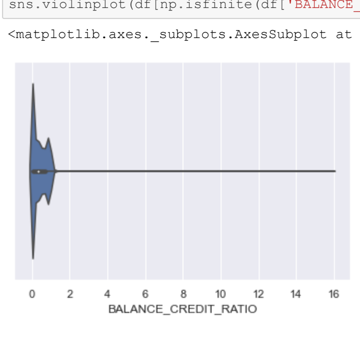


In [69]:
```
sns.scatterplot(df['CREDIT_LIMIT'],df['BALANCE'])
```
Out[69]: <matplotlib.axes._subplots.AxesSubplot at 0x20f94997370>

In [70]:
```
df[['CREDIT_LIMIT','BALANCE','PURCHASES','PURCHASES_TRX','PURCHASES_FREQUENCY','CASH_ADVANCE']].corr()
```
Out[70]:

	CREDIT_LIMIT	BALANCE	PURCHASES	PURCHASES_TRX	PURCHASES_FREQUENCY	CASH_ADVANCE
CREDIT_LIMIT	1.000	0.531	0.357	0.273	0.120	0.304
BALANCE	0.531	1.000	0.181	0.154	-0.078	0.497
PURCHASES	0.357	0.181	1.000	0.690	0.393	-0.051
PURCHASES_TRX	0.273	0.154	0.690	1.000	0.568	-0.076
PURCHASES_FREQUENCY	0.120	-0.078	0.393	0.568	1.000	-0.216
CASH_ADVANCE	0.304	0.497	-0.051	-0.076	-0.216	1.000

In [71]:
```
df['BALANCE_CREDIT_RATIO']=df['BALANCE']/df['CREDIT_LIMIT']
sns.violinplot(df[np.isfinite(df['BALANCE_CREDIT_RATIO'])]['BALANCE_CREDIT_RATIO'])
```
Out[71]: <matplotlib.axes._subplots.AxesSubplot at 0x20f949f31f0>



Customers with very limit usage

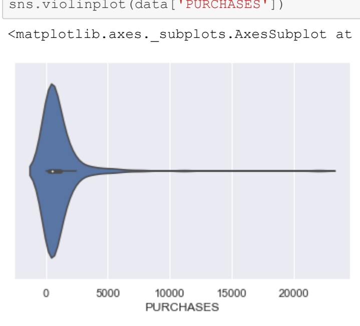
In [72]:
```
data=df[df['BALANCE_CREDIT_RATIO']>1]
data.head()
```
Out[72]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE
5	C10006	1809.829	1.000	1333.280	0.000	1333.280	0.000
10	C10011	1293.125	1.000	920.120	0.000	920.120	0.000
20	C10021	2016.685	1.000	176.680	0.000	176.680	0.000
64	C10067	1923.887	1.000	1887.640	0.000	1887.640	1605.949
78	C10082	1205.717	1.000	0.000	0.000	0.000	38.326

5 rows × 21 columns

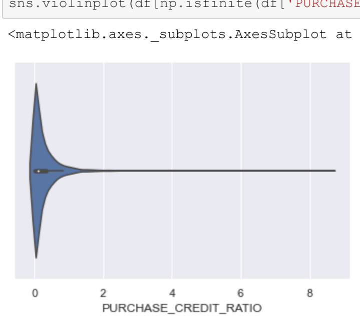
In [73]:
```
data['PURCHASE_TYPE'].value_counts()
```
Out[73]:
```
INSTALLMENTS    124
Both             58
NO PURCHASE      23
ONE OFF          23
Name: PURCHASE_TYPE, dtype: int64
```

In [74]:
```
sns.violinplot(data['PURCHASES'])
```
Out[74]: <matplotlib.axes._subplots.AxesSubplot at 0x20f94bf8880>



Spending vs Limit

In [75]:
```
df['PURCHASE_CREDIT_RATIO']=df['PURCHASES']/df['CREDIT_LIMIT']
sns.violinplot(df[np.isfinite(df['PURCHASE_CREDIT_RATIO'])]['PURCHASE_CREDIT_RATIO'])
```
Out[75]: <matplotlib.axes._subplots.AxesSubplot at 0x20f94cc7880>



Customer spending more than their limit

In [76]:
```
data=df[df['PURCHASE_CREDIT_RATIO']>1]
data.head()
```
Out[76]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE
12	C10013	1516.929	1.000	3217.990	2500.230	717.760	0.000
51	C10053	1078.889	1.000	3107.850	2221.740	886.110	0.000
64	C10067	1923.887	1.000	1887.640	0.000	1887.640	1605.949
125	C10131	2392.918	1.000	12462.440	12462.440	0.000	0.000
138	C10144	19043.139	1.000	22009.920	9449.070	12560.850	0.000

5 rows × 22 columns

In [77]:
```
data['PURCHASE_TYPE'].value_counts()
```
Out[77]:
```
Both            309
INSTALLMENTS     66
ONE OFF          66
Name: PURCHASE_TYPE, dtype: int64
```

Customers with High Credit Limit

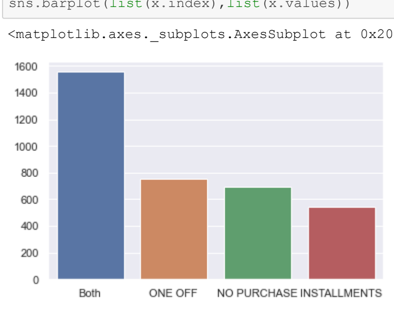
In [78]:
```
data1=df[df['CREDIT_LIMIT']>=df['CREDIT_LIMIT'].mean()]
data1.head()
```
Out[78]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE
1	C10002	3202.467	0.909	0.000	0.000	0.000	6442.945
2	C10003	2495.149	1.000	773.170	773.170	0.000	0.000
3	C10004	1666.671	0.636	1499.000	1499.000	0.000	205.788
6	C10007	627.261	1.000	7091.010	6402.630	688.380	0.000
8	C10009	1014.926	1.000	861.490	661.490	200.000	0.000

5 rows × 22 columns

In [79]:
```
print("Average Balance: ",data1['BALANCE'].mean())
print("Average Purchases: ",data1['PURCHASES'].mean())
print("Average Purchases Transaction: ",data1['PURCHASES_TRX'].mean())
print("Average Purchases Frequency: ",data1['PURCHASES_FREQUENCY'].mean())
print("Average Tenure: ",int(data1['TENURE'].mean()))
print("Average % Full Payment Month: ",data1['PRC_FULL_PAYMENT'].mean())
```
```
Average Balance:  2667.2907398833754
Average Purchases:  1670.57205633803
Average Purchases Transaction:  21.712394366197184
Average Purchases Frequency:  0.5463446853521137
Average Tenure:  11
Average % Full Payment Month:  0.1676607259154935
```

In [80]:
```
x=data1['PURCHASE_TYPE'].value_counts()
sns.barplot(list(x.index),list(x.values))
```
Out[80]: <matplotlib.axes._subplots.AxesSubplot at 0x20f94e451c0>



Customers with Low Credit Limit

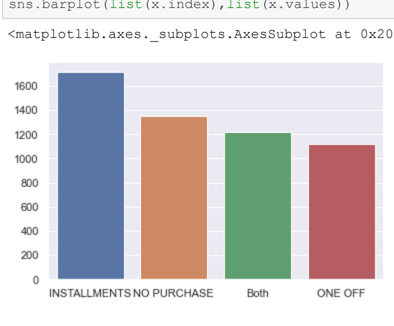
In [81]:
```
data2=df[df['CREDIT_LIMIT']<df['CREDIT_LIMIT'].mean()]
data2.head()
```
Out[81]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE
0	C10001	40.901	0.818	95.400	0.000	95.400	0.000
4	C10005	817.714	1.000	16.000	16.000	0.000	0.000
5	C10006	1809.829	1.000	1333.280	0.000	1333.280	0.000
7	C10008	1823.653	1.000	436.200	0.000	436.200	0.000
10	C10011	1293.125	1.000	920.120	0.000	920.120	0.000

5 rows × 22 columns

In [82]:
```
print("Average Balance: ",data2['BALANCE'].mean())
print("Average Purchases: ",data2['PURCHASES'].mean())
print("Average Purchases Transaction: ",data2['PURCHASES_TRX'].mean())
print("Average Purchases Frequency: ",data2['PURCHASES_FREQUENCY'].mean())
print("Average Tenure: ",int(data2['TENURE'].mean()))
print("Average % Full Payment Month: ",data2['PRC_FULL_PAYMENT'].mean())
```
```
Average Balance:  839.4754779875932
Average Purchases:  564.4726777777768
Average Purchases Transaction:  10.106296296296296
Average Purchases Frequency:  0.4535395879629641
Average Tenure:  11
Average % Full Payment Month:  0.1445463938888894
```

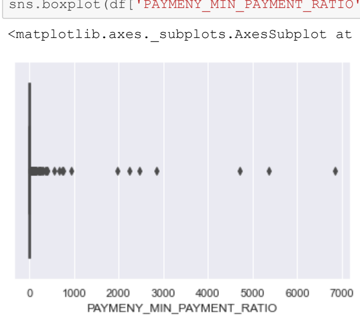
In [83]:
```
x=data2['PURCHASE_TYPE'].value_counts()
sns.barplot(list(x.index),list(x.values))
```
Out[83]: <matplotlib.axes._subplots.AxesSubplot at 0x20f948e4af0>



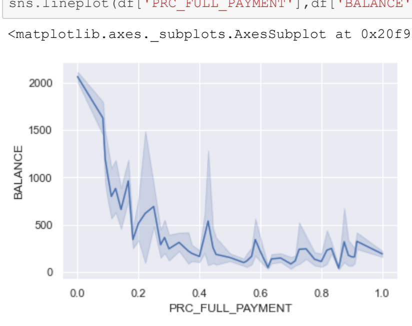
Payments

In [84]:
```
df['PAYMENY_MIN_PAYMENT_RATIO']=0
for i in range(len(df)):
    if(df['MINIMUM_PAYMENTS'][i]!=0):
        df['PAYMENY_MIN_PAYMENT_RATIO'][i]=df['PAYMENTS'][i]/df['MINIMUM_PAYMENTS'][i]

df['PAYMENY_MIN_PAYMENT_RATIO'].head()
```
```
<ipython-input-84-9e1d30da7ef3>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
g.html#returning-a-view-versus-a-copy
  df['PAYMENY_MIN_PAYMENT_RATIO'][i]=df['PAYMENTS'][i]/df['MINIMUM_PAYMENTS'][i]
```
Out[84]:
```
0    1
1    3
2    0
3    0
4    2
Name: PAYMENY_MIN_PAYMENT_RATIO, dtype: int64
```

In [85]:
```
sns.boxplot(df['PAYMENY_MIN_PAYMENT_RATIO'])
```
Out[85]: <matplotlib.axes._subplots.AxesSubplot at 0x20f925474c0>



In [86]:
```
sns.lineplot(df['PRC_FULL_PAYMENT'],df['BALANCE'])
```
Out[86]: <matplotlib.axes._subplots.AxesSubplot at 0x20f924f5310>



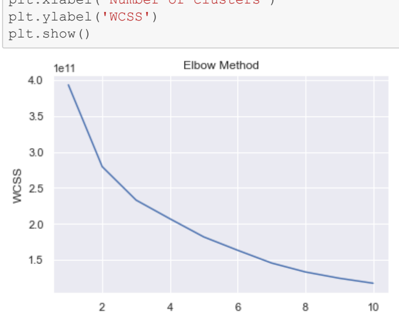
Clustering

In [87]:
```
from sklearn.datasets.samples_generator import make_blobs
from sklearn.cluster import KMeans
```
```
C:\Users\Harsh\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:143: FutureWarning: The sklea
rn.datasets.samples_generator module is  deprecated in version 0.22 and will be removed in version 0.
24. The corresponding classes / functions should instead be imported from sklearn.datasets. Anything
that cannot be imported from sklearn.datasets is now part of the private API.
  warnings.warn(message, FutureWarning)
```

In [88]:
```
data=df.drop('CUST_ID',axis=1)
data.dropna(inplace=True)
data=data.reset_index().drop('index',axis=1)

for i in range(len(data)):
    if(data['PURCHASE_TYPE'][i]=='Both'):
        data['PURCHASE_TYPE'][i]=0
    elif(data['PURCHASE_TYPE'][i]=='NO PURCHASE'):
        data['PURCHASE_TYPE'][i]=1
    if(data['PURCHASE_TYPE'][i]=='ONE OFF'):
        data['PURCHASE_TYPE'][i]=2
    if(data['PURCHASE_TYPE'][i]=='INSTALLMENTS'):
        data['PURCHASE_TYPE'][i]=3
```
```
<ipython-input-88-189ecec41ea7>:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
g.html#returning-a-view-versus-a-copy
  data['PURCHASE_TYPE'][i]=3
<ipython-input-88-189ecec41ea7>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
g.html#returning-a-view-versus-a-copy
  data['PURCHASE_TYPE'][i]=1
<ipython-input-88-189ecec41ea7>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
g.html#returning-a-view-versus-a-copy
  data['PURCHASE_TYPE'][i]=2
<ipython-input-88-189ecec41ea7>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
g.html#returning-a-view-versus-a-copy
  data['PURCHASE_TYPE'][i]=0
```

Optimal numbers of Cluster

In [89]:
```
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(data)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

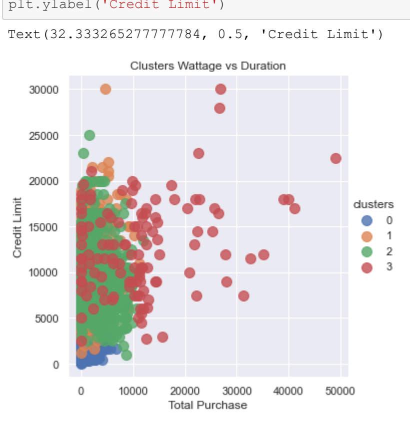


In [90]:
```
kmeans = KMeans(n_clusters=4, random_state=0).fit(data)
labels = kmeans.labels_
data['clusters'] = labels
```

In [91]:
```
data['clusters'].value_counts()
```
Out[91]:
```
0    5721
2    2342
1     766
3     120
Name: clusters, dtype: int64
```

In [92]:
```
for i in range(len(data)):
    if(data['PURCHASE_TYPE'][i]==0):
        data['PURCHASE_TYPE'][i]='Both'
    elif(data['PURCHASE_TYPE'][i]==1):
        data['PURCHASE_TYPE'][i]='NO PURCHASE'
    if(data['PURCHASE_TYPE'][i]==2):
        data['PURCHASE_TYPE'][i]='ONE OFF'
    if(data['PURCHASE_TYPE'][i]==3):
        data['PURCHASE_TYPE'][i]='INSTALLMENTS'
```
```
<ipython-input-92-bdc1372d9fe1>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
g.html#returning-a-view-versus-a-copy
  data['PURCHASE_TYPE'][i]='INSTALLMENTS'
<ipython-input-92-bdc1372d9fe1>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
g.html#returning-a-view-versus-a-copy
  data['PURCHASE_TYPE'][i]='NO PURCHASE'
<ipython-input-92-bdc1372d9fe1>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
g.html#returning-a-view-versus-a-copy
  data['PURCHASE_TYPE'][i]='ONE OFF'
<ipython-input-92-bdc1372d9fe1>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
g.html#returning-a-view-versus-a-copy
  data['PURCHASE_TYPE'][i]='Both'
```

In [93]:
```
sns.lmplot('PURCHASES', 'CREDIT_LIMIT',
           data=data,
           fit_reg=False,
           hue="clusters",
           scatter_kws={"marker": "D",
                        "s": 100})
plt.title('Clusters Wattage vs Duration')
plt.xlabel('Total Purchase')
plt.ylabel('Credit Limit')
```
Out[93]: Text(32.333265277777784, 0.5, 'Credit Limit')



Cluster Analysis

In [94]:
```
final=pd.DataFrame(columns=['Average Balance','Average Purchases','Most Used Purchase Type','Average Ad
vance Cash',
                            'Average Purchase Transaction','Average Credit Limit','Average Payments','Av
erage Tenure',
                             'Average Purchase Credit Ratio'])

for i in range(0,4):
    record=data[data['clusters']==i]

    val1=record['BALANCE'].mean()
    val2=record['PURCHASES'].mean()
    val3=list(record['PURCHASE_TYPE'].value_counts().index)[0]
    val4=record['CASH_ADVANCE'].mean()
    val5=record['PURCHASES_TRX'].mean()
    val6=record['CREDIT_LIMIT'].mean()
    val7=record['PAYMENTS'].mean()
    val8=record['TENURE'].mean()
    val9=record['PURCHASE_CREDIT_RATIO'].mean()

    final.loc[len(final)]=[val1,val2,val3,val4,val5,val6,val7,val8,val9]
```

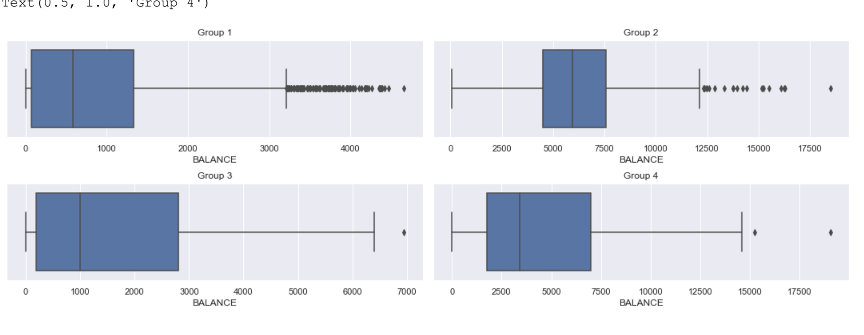
In [95]:
```
final
```
Out[95]:

	Average Balance	Average Purchases	Most Used Purchase Type	Average Advance Cash	Average Purchase Transaction	Average Credit Limit	Average Payments	Average Tenure	Average Purchase Credit Ratio
0	841.506	513.330	INSTALLMENTS	517.605	9.628	2314.043	920.044	11.386	0.273
1	6193.129	1099.748	NO PURCHASE	4625.742	17.127	9769.784	3949.482	11.624	0.118
2	1651.697	1640.224	Both	710.335	22.568	7675.058	2133.131	11.789	0.247
3	4796.388	11317.598	Both	4938.089	88.292	12696.250	18557.987	11.825	1.003

Balance

In [96]:
```python
fig, ax = plt.subplots(2, 2)
# Setting height and width of subplots
fig.set_figheight(5)
fig.set_figwidth(15)
sns.boxplot(data[data['clusters']==0]["BALANCE"], ax=ax[0, 0])
sns.boxplot(data[data['clusters']==1]["BALANCE"], ax=ax[0, 1])
sns.boxplot(data[data['clusters']==2]["BALANCE"], ax=ax[1, 0])
sns.boxplot(data[data['clusters']==3]["BALANCE"], ax=ax[1, 1])
plt.tight_layout(h_pad=2.0)
ax[0,0].set_title('Group 1')
ax[0,1].set_title('Group 2')
ax[1,0].set_title('Group 3')
ax[1,1].set_title('Group 4')
```

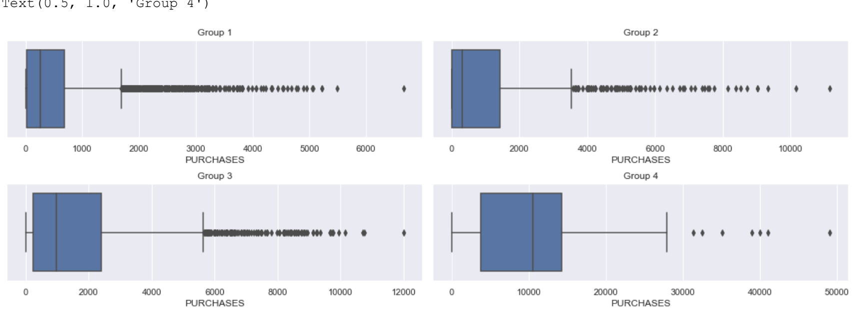
Out[96]: Text(0.5, 1.0, 'Group 4')



Purchases

In [97]:
```python
fig, ax = plt.subplots(2, 2)
# Setting height and width of subplots
fig.set_figheight(5)
fig.set_figwidth(15)
sns.boxplot(data[data['clusters']==0]["PURCHASES"], ax=ax[0, 0])
sns.boxplot(data[data['clusters']==1]["PURCHASES"], ax=ax[0, 1])
sns.boxplot(data[data['clusters']==2]["PURCHASES"], ax=ax[1, 0])
sns.boxplot(data[data['clusters']==3]["PURCHASES"], ax=ax[1, 1])
plt.tight_layout(h_pad=2.0)
ax[0,0].set_title('Group 1')
ax[0,1].set_title('Group 2')
ax[1,0].set_title('Group 3')
ax[1,1].set_title('Group 4')
```

Out[97]: Text(0.5, 1.0, 'Group 4')



Purchases Type

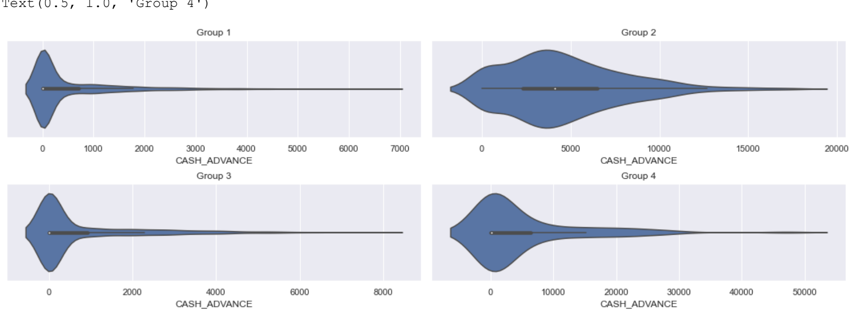
In [98]:
```python
fig, ax = plt.subplots(2, 2)
# Setting height and width of subplots
fig.set_figheight(5)
fig.set_figwidth(15)

x1=data[data['clusters']==0]["PURCHASE_TYPE"].value_counts()
x2=data[data['clusters']==1]["PURCHASE_TYPE"].value_counts()
x3=data[data['clusters']==2]["PURCHASE_TYPE"].value_counts()
x4=data[data['clusters']==3]["PURCHASE_TYPE"].value_counts()


sns.barplot(list(x1.index),list(x1.values),ax=ax[0, 0])
sns.barplot(list(x2.index),list(x2.values),ax=ax[0, 1])
sns.barplot(list(x3.index),list(x3.values),ax=ax[1, 0])
sns.barplot(list(x4.index),list(x4.values),ax=ax[1, 1])

plt.tight_layout(h_pad=2.0)
ax[0,0].set_title('Group 1')
ax[0,1].set_title('Group 2')
ax[1,0].set_title('Group 3')
ax[1,1].set_title('Group 4')
```

Out[98]: Text(0.5, 1.0, 'Group 4')

Advance Cash

In [99]:
```python
fig, ax = plt.subplots(2, 2)
# Setting height and width of subplots
fig.set_figheight(5)
fig.set_figwidth(15)
sns.violinplot(data[data['clusters']==0]["CASH_ADVANCE"], ax=ax[0, 0])
sns.violinplot(data[data['clusters']==1]["CASH_ADVANCE"], ax=ax[0, 1])
sns.violinplot(data[data['clusters']==2]["CASH_ADVANCE"], ax=ax[1, 0])
sns.violinplot(data[data['clusters']==3]["CASH_ADVANCE"], ax=ax[1, 1])
plt.tight_layout(h_pad=2.0)
ax[0,0].set_title('Group 1')
ax[0,1].set_title('Group 2')
ax[1,0].set_title('Group 3')
ax[1,1].set_title('Group 4')
```

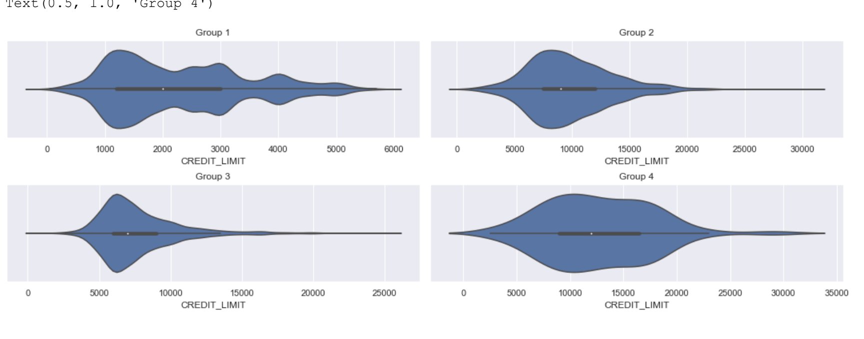
Out[99]: Text(0.5, 1.0, 'Group 4')



Credit Limit

In [100]:
```python
fig, ax = plt.subplots(2, 2)
# Setting height and width of subplots
fig.set_figheight(5)
fig.set_figwidth(15)
sns.violinplot(data[data['clusters']==0]["CREDIT_LIMIT"], ax=ax[0, 0])
sns.violinplot(data[data['clusters']==1]["CREDIT_LIMIT"], ax=ax[0, 1])
sns.violinplot(data[data['clusters']==2]["CREDIT_LIMIT"], ax=ax[1, 0])
sns.violinplot(data[data['clusters']==3]["CREDIT_LIMIT"], ax=ax[1, 1])
plt.tight_layout(h_pad=2.0)
ax[0,0].set_title('Group 1')
ax[0,1].set_title('Group 2')
ax[1,0].set_title('Group 3')
ax[1,1].set_title('Group 4')
```

Out[100]: Text(0.5, 1.0, 'Group 4')



Purchase Credit Ratio

In [101]:
```python
fig, ax = plt.subplots(2, 2)
# Setting height and width of subplots
fig.set_figheight(5)
fig.set_figwidth(15)
sns.boxplot(data[data['clusters']==0]["PURCHASE_CREDIT_RATIO"], ax=ax[0, 0])
sns.boxplot(data[data['clusters']==1]["PURCHASE_CREDIT_RATIO"], ax=ax[0, 1])
sns.boxplot(data[data['clusters']==2]["PURCHASE_CREDIT_RATIO"], ax=ax[1, 0])
sns.boxplot(data[data['clusters']==3]["PURCHASE_CREDIT_RATIO"], ax=ax[1, 1])
plt.tight_layout(h_pad=2.0)
ax[0,0].set_title('Group 1')
ax[0,1].set_title('Group 2')
ax[1,0].set_title('Group 3')
ax[1,1].set_title('Group 4')
```

Out[101]: Text(0.5, 1.0, 'Group 4')