Importing Useful Libraries and Reading Data from CSV

In [33]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from scipy.linalg import eigh
from sklearn import decomposition
```

Data Loading and Understanding

In [7]:
```
df=pd.read_csv("Data/secom.data",delimiter=" ",header=None)
```

In [8]:
```
df.head()
```

Out[8]:

	0	1	2	3	4	5	6	7	8	9	...	
0	3030.93	2564.00	2187.7333	1411.1265	1.3602	100.0	97.6133	0.1242	1.5005	0.0162	...	N
1	3095.78	2465.14	2230.4222	1463.6606	0.8294	100.0	102.3433	0.1247	1.4966	-0.0005	...	0.0
2	2932.61	2559.94	2186.4111	1698.0172	1.5102	100.0	95.4878	0.1241	1.4436	0.0041	...	0.0
3	2988.72	2479.90	2199.0333	909.7926	1.3204	100.0	104.2367	0.1217	1.4882	-0.0124	...	0.0
4	3032.24	2502.87	2233.3667	1326.5200	1.5334	100.0	100.3967	0.1235	1.5031	-0.0031	...	N

5 rows × 590 columns

In [9]:
```
df.describe()
```

Out[9]:

	0	1	2	3	4	5	6	
count	1561.000000	1560.000000	1553.000000	1553.000000	1553.000000	1553.0	1553.000000	155
mean	3014.452896	2495.850231	2200.547318	1396.376627	4.197013	100.0	101.112908	(
std	73.621787	80.407705	29.513152	441.691640	56.355540	0.0	6.237214	(
min	2743.240000	2158.750000	2060.660000	0.000000	0.681500	100.0	82.131100	(
25%	2966.260000	2452.247500	2181.044400	1081.875800	1.017700	100.0	97.920000	
50%	3011.490000	2499.405000	2201.066700	1285.214400	1.316800	100.0	101.512200	(
75%	3056.650000	2538.822500	2218.055500	1591.223500	1.525700	100.0	104.586700	(
max	3356.350000	2846.440000	2315.266700	3715.041700	1114.536600	100.0	129.252200	(

8 rows × 590 columns

Data Insights:

- We can see there are NULL values in the data. So, we can use respective imputation methods to replace them
- If we feel the respective columns are not useful to complete task, we can remove them

Data Preprocessing

Imputation : NULL values are replaced with median imputation method

In [11]:
```
impt_df=df.fillna(df.median())
```

In [12]:
```
impt_df.describe()
```

Out[12]:

	0	1	2	3	4	5	6	
count	1567.000000	1567.000000	1567.000000	1567.000000	1567.000000	1567.0	1567.000000	156
mean	3014.441551	2495.866110	2200.551958	1395.383474	4.171281	100.0	101.116476	(
std	73.480841	80.228143	29.380973	439.837330	56.103721	0.0	6.209385	(
min	2743.240000	2158.750000	2060.660000	0.000000	0.681500	100.0	82.131100	(
25%	2966.665000	2452.885000	2181.099950	1083.885800	1.017700	100.0	97.937800	
50%	3011.490000	2499.405000	2201.066700	1285.214400	1.316800	100.0	101.512200	(
75%	3056.540000	2538.745000	2218.055500	1590.169900	1.518800	100.0	104.530000	(
max	3356.350000	2846.440000	2315.266700	3715.041700	1114.536600	100.0	129.252200	(

8 rows × 590 columns

Columns with only one unique value after imputation mean they do not contribute anything to our signal, we can safely ignore them

In [13]:
```
cols = impt_df.nunique()==1
```

In [18]:
```
prun_data=impt_df.drop(list(impt_df.columns[cols]),axis=1)
```

In [19]:
```
prun_data.shape
```

Out[19]:
```
(1567, 474)
```

Principle Component Analysis

Step1 : Standardization

In [22]:
```
std_data=StandardScaler().fit_transform(prun_data)
std_data.shape
```

Out[22]: (1567, 474)

Step2 : Covariance Matrix

X.X^T

In [23]:
```
covar_matrix = np.matmul(std_data.T,std_data)
```

In [25]:
```
covar_matrix.shape
```

Out[25]: (474, 474)

Step3 : Finding Eigen Values and Eigen Vectors

In [30]:
```
ei_values,ei_vectors = eigh(covar_matrix)
```

In [31]:
```
ei_vectors.shape
```

Out[31]: (474, 474)

In [32]:
```
ei_values
```

Out[32]:
```
array([-3.19699635e-13, -1.23832267e-13, -5.32527121e-14, -3.06980482e-16,
        4.70483139e-13,  2.94765182e-07,  1.03397281e-05,  1.28284434e-04,
        4.14830248e-04,  4.90169224e-04,  6.18019575e-04,  1.08672745e-03,
        2.55230422e-03,  3.71502474e-03,  3.97943660e-03,  5.47744555e-03,
        5.67069581e-03,  6.18012341e-03,  6.75068619e-03,  7.13477913e-03,
        8.08436908e-03,  1.12135379e-02,  1.33919741e-02,  1.42546723e-02,
        1.52665284e-02,  1.64840364e-02,  2.06802053e-02,  2.43945360e-02,
        2.64216193e-02,  3.00232648e-02,  3.28949102e-02,  3.54097527e-02,
        4.28791454e-02,  4.80209785e-02,  5.14296888e-02,  7.40364404e-02,
        7.78820006e-02,  8.72572776e-02,  9.35095547e-02,  1.04211154e-01,
        1.06873660e-01,  1.27039159e-01,  1.31054631e-01,  1.35328503e-01,
        1.38382097e-01,  1.48672855e-01,  1.63268517e-01,  1.71126072e-01,
        1.79155996e-01,  1.81481406e-01,  1.96681819e-01,  2.14337823e-01,
        2.24625493e-01,  2.58788682e-01,  2.71803040e-01,  2.86571094e-01,
        2.94291554e-01,  3.39726182e-01,  3.42768567e-01,  3.61098902e-01,
        3.76792672e-01,  3.96494031e-01,  4.41745317e-01,  4.77610946e-01,
        4.87615370e-01,  5.52788358e-01,  5.76609249e-01,  6.08846267e-01,
        6.61908092e-01,  6.75621582e-01,  7.58663112e-01,  8.14248679e-01,
        8.29838535e-01,  8.47849747e-01,  9.77476530e-01,  9.96601945e-01,
```

In [34]:
```
pca = decomposition.PCA(std_data)
```

In [35]:
```
pca.n_components = 474
pca_data = pca.fit_transform(std_data)
```

In [37]:
```
pca.explained_variance_
```

Out[37]:
```
array([2.63633688e+01, 1.72743886e+01, 1.33923210e+01, 1.20672691e+01,
       1.03899213e+01, 9.85089401e+00, 9.36521585e+00, 8.70194890e+00,
       8.53612401e+00, 7.69171544e+00, 6.92858488e+00, 6.31641050e+00,
       6.22095270e+00, 6.06098346e+00, 5.99357961e+00, 5.66289344e+00,
       5.46800377e+00, 5.40084730e+00, 5.31395436e+00, 5.03717250e+00,
       4.89732003e+00, 4.78918927e+00, 4.71192980e+00, 4.58288439e+00,
       4.53726627e+00, 4.47509457e+00, 4.38684780e+00, 4.21697918e+00,
       4.12895524e+00, 4.01320156e+00, 3.95271790e+00, 3.88455715e+00,
       3.83404888e+00, 3.77433459e+00, 3.66948307e+00, 3.64494209e+00,
       3.57887822e+00, 3.54935338e+00, 3.45550698e+00, 3.41105623e+00,
       3.37765422e+00, 3.29946613e+00, 3.20866391e+00, 3.17397543e+00,
       3.15647304e+00, 3.11795646e+00, 3.03698198e+00, 2.97186294e+00,
       2.95049394e+00, 2.87837110e+00, 2.85416945e+00, 2.84066647e+00,
       2.78090436e+00, 2.69371574e+00, 2.65751997e+00, 2.65199254e+00,
       2.62462680e+00, 2.56152890e+00, 2.53605147e+00, 2.48901568e+00,
       2.43283315e+00, 2.40712231e+00, 2.34736284e+00, 2.31689465e+00,
       2.27430364e+00, 2.23117593e+00, 2.19717603e+00, 2.17545333e+00,
       2.15407410e+00, 2.10707687e+00, 2.08639056e+00, 2.04370240e+00,
       2.02470114e+00, 2.01160573e+00, 1.99226342e+00, 1.94513561e+00,
```

Preserved variance ratio

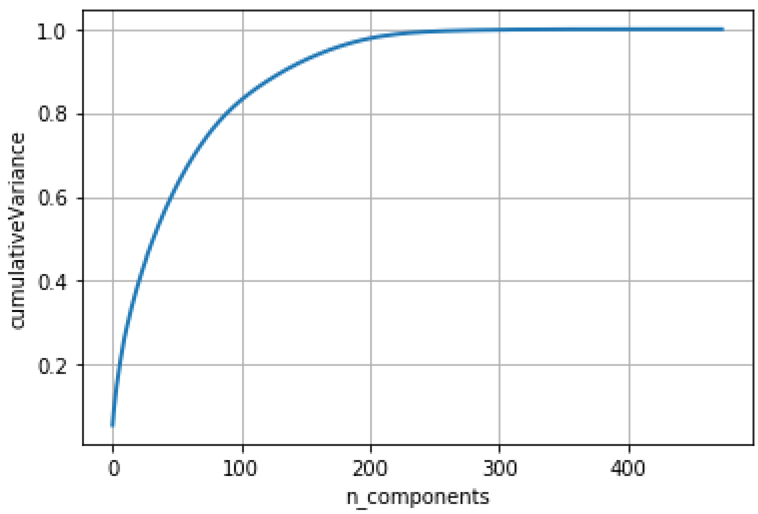
In [38]:
```
each_pc_explained_variance_ratio=pca.explained_variance_ratio_
each_pc_explained_variance_ratio
```

Out[38]:
```
array([5.55834276e-02, 3.64206008e-02, 2.82358113e-02, 2.54421271e-02,
       2.19056770e-02, 2.07692142e-02, 1.97452306e-02, 1.83468263e-02,
       1.79972080e-02, 1.62168921e-02, 1.46079395e-02, 1.33172566e-02,
       1.31159973e-02, 1.27787248e-02, 1.26366134e-02, 1.19394084e-02,
       1.15285112e-02, 1.13869213e-02, 1.12037198e-02, 1.06201645e-02,
       1.03253054e-02, 1.00973270e-02, 9.93443635e-03, 9.66236238e-03,
       9.56618303e-03, 9.43510281e-03, 9.24904700e-03, 8.89090309e-03,
       8.70531709e-03, 8.46126684e-03, 8.33374563e-03, 8.19003833e-03,
       8.08354882e-03, 7.95764968e-03, 7.73658511e-03, 7.68484394e-03,
       7.54555762e-03, 7.48330868e-03, 7.28544685e-03, 7.19172874e-03,
       7.12130533e-03, 6.95645683e-03, 6.76501322e-03, 6.69187746e-03,
       6.65497616e-03, 6.57376940e-03, 6.40304619e-03, 6.26575191e-03,
       6.22069840e-03, 6.06863762e-03, 6.01761187e-03, 5.98914275e-03,
       5.86314282e-03, 5.67931795e-03, 5.60300430e-03, 5.59135050e-03,
       5.53365371e-03, 5.40062073e-03, 5.34690518e-03, 5.24773689e-03,
       5.12928398e-03, 5.07507633e-03, 4.94908194e-03, 4.88484407e-03,
       4.79504697e-03, 4.70411831e-03, 4.63243433e-03, 4.58663509e-03,
       4.54156003e-03, 4.44247302e-03, 4.39885887e-03, 4.30885694e-03,
       4.26879548e-03, 4.24118565e-03, 4.20040514e-03, 4.10104282e-03,
```

In [40]:
```
cumulative_Variance = pca.explained_variance_ratio_.cumsum()
```

We can see that We should consider 200 - 230, approx optimal number of principal components Principal Components to explain more than 98% of variance using below graph

In [41]:
```
plt.figure(1, figsize=(6, 4))
plt.clf()
plt.plot(cumulative_Variance, linewidth=2)
plt.axis('tight')
plt.grid(True)
plt.xlabel('n_components')
plt.ylabel('cumulativeVariance')
plt.show()
```



We can see around 98% of variance below 200-230 components

In [44]:
```
pca.n_components=7
principal7Components = pca.fit_transform(std_data)
principalDF = pd.DataFrame(data=principal7Components,columns=['pcomp1','pcomp2','
```

Top 7 principal-components based values of their variances

In [45]:
```
principalDF
```

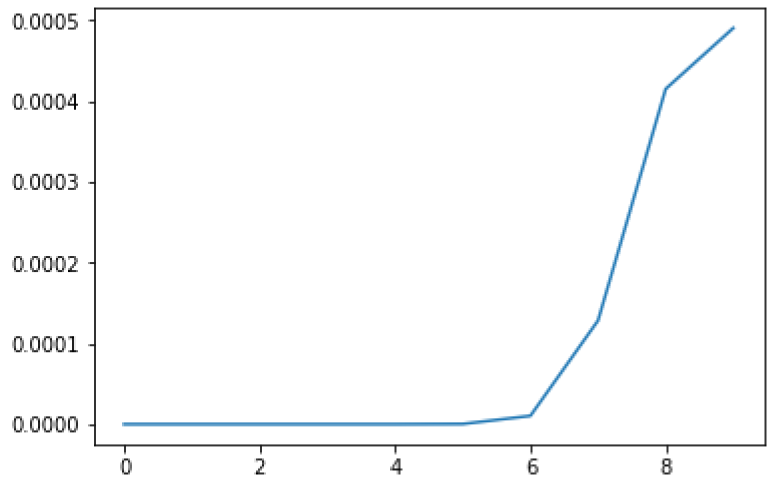
Out[45]:

	pcomp1	pcomp2	pcomp3	pcomp4	pcomp5	pcomp6	pcomp7
0	-1.694695	2.925812	3.890342	-2.545239	-0.070348	-1.184549	-1.716559
1	-2.247194	0.876782	2.878214	-2.034221	-0.328795	-1.527402	-2.236821
2	0.442994	1.231955	1.146580	-0.371850	0.603738	2.164022	-2.154083
3	1.158528	5.131687	4.251766	-3.428451	3.023635	1.925283	-8.182085
4	0.753948	2.379104	2.543256	-0.033414	2.039298	3.268601	-4.900000
5	2.076392	2.875918	3.572881	-2.209836	1.779089	3.565122	-2.458103
6	-1.747077	4.526124	2.858190	-1.451715	0.475425	-1.430457	-2.957467
7	1.292039	2.012678	4.734057	-2.362956	0.374715	1.455374	-2.476023
8	-0.471772	19.143200	-3.488871	3.608310	0.417586	2.585036	-0.662857
9	1.303172	4.006341	5.713491	-2.966116	1.035043	2.495597	-2.476655
10	-2.083069	22.620039	-7.100650	6.414667	-2.238695	-1.268602	2.242799
11	0.735209	3.659814	4.836746	-2.406979	1.306302	2.116889	-2.212877
12	0.561499	1.502583	3.127285	-0.570487	1.310680	2.024431	-3.988153
13	-2.618774	2.687974	4.617412	-1.328211	-0.295889	-1.277196	-1.291162
14	-3.186040	32.201045	-11.994388	7.766983	0.588387	3.348226	2.470326
15	-2.980570	27.061397	-9.958615	5.956671	0.768443	1.754741	0.604205
16	-0.916633	0.769981	1.212576	-1.983483	1.273528	1.628671	-4.723774
17	-1.232625	3.851757	2.888540	-2.577332	1.657917	2.073207	-6.885032
18	-0.749024	0.223831	2.387614	-0.576839	-0.041625	0.556619	-3.552802
19	1.020533	2.615800	2.561091	-0.774736	1.347508	1.960637	-3.280060
20	0.153006	-0.613494	1.383259	-0.801180	0.404798	1.696359	-3.806064
21	-0.282477	0.818745	3.129836	-1.507462	0.758892	0.941241	-4.548035
22	-0.582044	0.337954	1.925557	-0.591190	0.218661	0.566078	-4.681425
23	-2.219011	25.077582	-6.981846	5.048530	0.225625	1.916324	0.185654
24	-3.235047	24.221556	-9.295245	4.397435	-0.342400	0.885021	0.482012
25	-3.543949	23.942219	-7.839054	5.193226	-0.974467	0.387509	0.876493
26	-1.775195	1.526452	1.956624	-1.990451	-0.749068	-3.613862	0.839460
27	-0.587920	2.883265	-0.440802	1.551498	0.051368	2.981275	-4.574252
28	-0.628810	3.647641	3.043362	-2.339473	1.206649	1.325085	-3.950534
29	-0.882877	2.084894	-0.595677	-0.128845	0.695389	1.009549	-2.654831
...	...	...	...	...	...	...	...
1537	-0.703425	-1.231339	0.891798	1.979978	-0.129514	0.427263	-2.811470
1538	0.091720	-3.087884	-3.192241	5.471369	0.068783	3.973099	-2.149885
1539	2.564356	-2.983508	-2.726740	2.125188	1.092110	1.886720	1.657666

	pcomp1	pcomp2	pcomp3	pcomp4	pcomp5	pcomp6	pcomp7
1540	-0.470646	-2.763350	-3.081914	2.322429	0.600848	0.821994	-0.973785
1541	-0.339324	-3.486483	-2.681056	1.859345	0.065839	3.177510	-1.637235
1542	-2.601628	-0.698569	-2.068284	1.004057	-2.183479	-3.642114	-1.820422
1543	-2.227351	-0.418138	0.136023	-0.399528	-1.425138	-1.282659	-2.956743
1544	-2.173570	-1.554030	-1.565042	0.977956	-1.454302	-2.460943	-1.048450
1545	1.555604	-1.098203	-1.772633	1.078833	1.690175	-0.074746	-0.500997
1546	-1.787294	-1.470350	-1.929721	0.879589	-0.884571	-1.094185	2.969880
1547	1.938959	-2.775672	-0.625056	2.159403	0.208924	-0.536735	0.434084
1548	1.913413	-0.302717	0.880078	-0.568864	2.056295	0.947159	-1.148983
1549	-0.107957	-4.074654	-0.826749	2.184694	-0.707332	-0.341727	1.203746
1550	-0.860093	-3.616090	-4.467705	3.361321	-0.141656	1.741185	-0.063746
1551	-0.671974	-1.450034	-1.091798	2.733379	-0.184463	-0.013225	-2.577605
1552	1.784591	-0.973232	1.846763	-2.748249	1.679823	1.936444	2.690457
1553	-2.012300	0.061228	0.937146	-1.167804	-0.769876	-1.277587	-2.888866
1554	1.032687	-1.518211	-1.900624	0.879672	0.810838	0.544758	-0.962426
1555	-0.394507	-3.160581	-3.301104	2.694920	-0.416933	3.781954	-2.987121
1556	0.902511	-3.448693	-1.525163	3.925534	1.053600	-0.794520	0.172042
1557	0.875302	-1.185432	-1.124881	0.600505	-0.322499	-0.501485	-0.016299
1558	-0.392312	-1.084178	-0.672872	-0.138581	0.533097	0.530556	-2.472355
1559	1.200491	-3.551510	-3.918440	3.862557	0.430796	4.072492	-1.845865
1560	-1.482319	-1.260658	1.881994	0.224500	0.058204	-1.214285	-2.121000
1561	2.229063	-2.341924	-0.451043	1.357294	1.253069	1.578701	-0.403294
1562	-1.190341	-3.735384	-2.611351	5.708814	-0.557646	1.874632	-1.194507
1563	-0.389805	0.484748	2.564024	-0.089495	1.314357	1.159761	-2.480222
1564	-1.169632	-1.773264	-1.618357	1.098914	-0.768666	-2.221571	0.520276
1565	-1.176656	-3.234535	-3.496680	3.074201	-0.007506	3.095718	-2.471983
1566	1.953314	-3.012189	-2.618094	3.718272	0.471644	3.758435	-2.638329

1567 rows × 7 columns

In [48]:
```
plt.plot(ei_values[:10])
plt.show()
```



Saturation point is 5 from above graph of eigen values

In [ ]:
```

```