PUMP SENSOR PROJECT

FINAL REPORT

Bokka Sai Pradeep

(17QE30001)

DESCRIPTIVE ANALYSIS :

Descriptive analysis or statistics does exactly what the name implies: they "describe", or summarize, raw data and make it something that is interpretable by humans. They are analytics that describe the past. The past refers to any point of time that an event has occurred, whether it is one minute ago, or one year ago. Descriptive analytics are useful because they allow us to learn from past behaviors, and understand how they might influence future outcomes.

The vast majority of the statistics we use fall into this category. (Think basic arithmetic like sums, averages, percent changes.) Usually, the underlying data is a count, or aggregate of a filtered column of data to which basic math is applied. For all practical purposes, there are an infinite number of these statistics. Descriptive statistics are useful to show things like total stock in inventory, average dollars spent per customer and year-over-year change in sales. Common examples of descriptive analytics are reports that provide historical insights regarding the company's production, financials, operations, sales, finance, inventory and customers.

PROCEDURE:

For descriptive Analysis ,we have to remove first unwanted columns which has no values(i.e. Nan or 0). The column sensor_15 has only Nan in every row. So I have removed it using data.drop command(I named my data frame as data).

Then we get descriptive analysis by using data.describe command.

RESULTS:

	sensor_00	sensor_01	sensor_02	sensor_03	sensor_04	sensor_05	sensor_06	sensor_07	sensor_08	sensor_0
count	210112.000000	219951.000000	220301.000000	220301.000000	220301.000000	220301.000000	215522.000000	214869.000000	215213.000000	215725.00000
mean	2.372221	47.591611	50.867392	43.752481	590.673936	73.396414	13.501537	15.843152	15.200721	14.79921
std	0.412227	3.296666	3.666820	2.418887	144.023912	17.298247	2.163736	2.201155	2.037390	2.09196
min	0.000000	0.000000	33.159720	31.640620	2.798032	0.000000	0.014468	0.000000	0.028935	0.00000
25%	2.438831	46.310760	50.390620	42.838539	626.620400	69.976260	13.346350	15.907120	15.183740	15.05353
50%	2.456539	48.133678	51.649300	44.227428	632.638916	75.576790	13.642940	16.167530	15.494790	15.08247
75%	2.499826	49.479160	52.777770	45.312500	637.615723	80.912150	14.539930	16.427950	15.697340	15.11863
max	2.549016	56.727430	56.032990	48.220490	800.000000	99.999880	22.251160	23.596640	24.348960	25.00000

PRINCIPLE COMPONENT ANALYSIS(PCA):

This is an unsupervised technique used for dimensionality reduction. This is similar to the above discussed techniques in a way that PCA also performs transformations of the data to obtain a set of axes in lower dimension. But, the drawback of this is that it does not guarantee the separation of classes as is the case with FLD. The idea of dimensionality reduction here is that useless dimensions can be eliminated after finding the new set of coordinate axes in the lower dimension. In PCA, it is always the best representation of the data without information loss which is the main aim. The principal component is the direction in which the variability of the data is maximum. The first step is to centre the data by subtracting the mean from all the data points corresponding a specific attribute. The next step is to plot this processed data and find an axis along which the variability is maximum. Then, it the algorithm tries to achieve an axis orthogonal to the first principal component covering as much as the remaining variability available. This process goes on as long as the desired amount of variation is obtained. In the process, as the entire variation is along the axes of the coordinate frame, the covariance matrix becomes diagonal. This implies that each of the attributes is correlated to itself but not to any other attribute. Seeing this in mathematical terms, in PCA, we consider a data matrix X and try to rotate it in a way that the data gets transformed in a direction of maximum variance.

Multiplication of the data matrix with a rotation matrix helps use achieve this. This can be written as Y = $P\ T\ .X$ where P has to be selected in a way that the covariance matrix of Y is diagonal. The eigen vectors of the covariance matrix gives the directions of maximum variance. Also, the eigen values corresponding to these eigen vectors gives an estimate of the direction with maximum variance among the principal components obtained. As the eigen vectors of a square symmetric matrix are orthogonal to each other, these are our desired new set of orthogonal axes in lower dimension.

PROCEDURE:

Step 1:

For performing PCA analysis , all columns should have either float or int values. Therefore, I have removed columns(timestamp and machine_status) which are of string data type and stored at different place creating new dataframes.

Step 2:

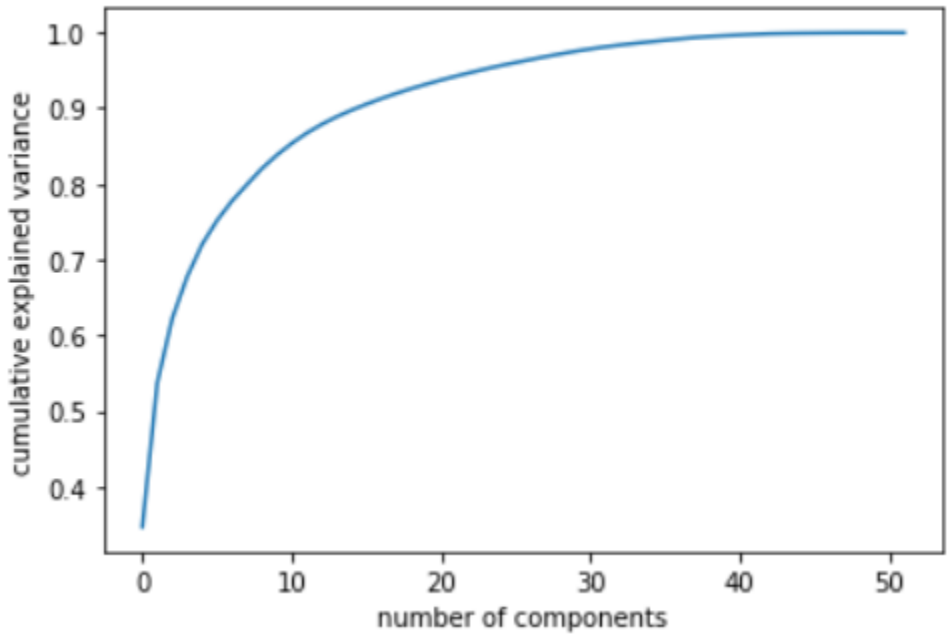
Standardising the data(i.e. scaling) : This means that we need to center and scale the data. In this way the average value of each record would be 0 and the variance for each record would be 1.

To scale our data, we would use StandardScalar which is available in sklearn.

Step 3:

We should now choose the number of components to which our data should be minimized. So to do this, we have to plot explained variance ratio and choose a number of components that "capture" at least 95% of the variance. From the above data the number of components that capture around 95% of the variance is around 20.

```
Text(0, 0.5, 'cumulative explained variance')
```



Step 4:

Perform PCA:

To then perform PCA we would use PCA module from sklearn which we have already imported using the command pca.fit and then we get the reduced array by using the command pca.transform.

RESULTS:

```
x_pca
```

```
array([[-0.02397053,  0.67333103, -0.51043314, ..., -0.22723493,
        -0.59078698,  0.46967172],
       [-0.02397053,  0.67333103, -0.51043314, ..., -0.22723493,
        -0.59078698,  0.46967172],
       [-0.16451561,  0.68154806, -0.48273584, ..., -0.16940969,
        -0.51737993,  0.35550179],
       ...,
       [-2.0344543 ,  2.42398516, -1.05590696, ...,  0.84823454,
         1.21427959, -0.51402917],
       [-1.92042831,  2.43314934, -1.01543919, ...,  0.80121889,
         1.20938945, -0.57342309],
       [-1.96586198,  2.40111081, -1.06135927, ...,  0.78680705,
         1.11367134, -0.64029607]])
```

1)IMPUTATION: I replaced all the null values in the dataframe with the mean of the respective columns.

```
data.drop(['sensor_15'],axis=1,inplace=True)
columns = [col for col in data.columns if not col.find('sensor')]
for col in columns:
    data[col] = data[col].fillna(data[col].mean())
```

Confirmation that there are no null values :

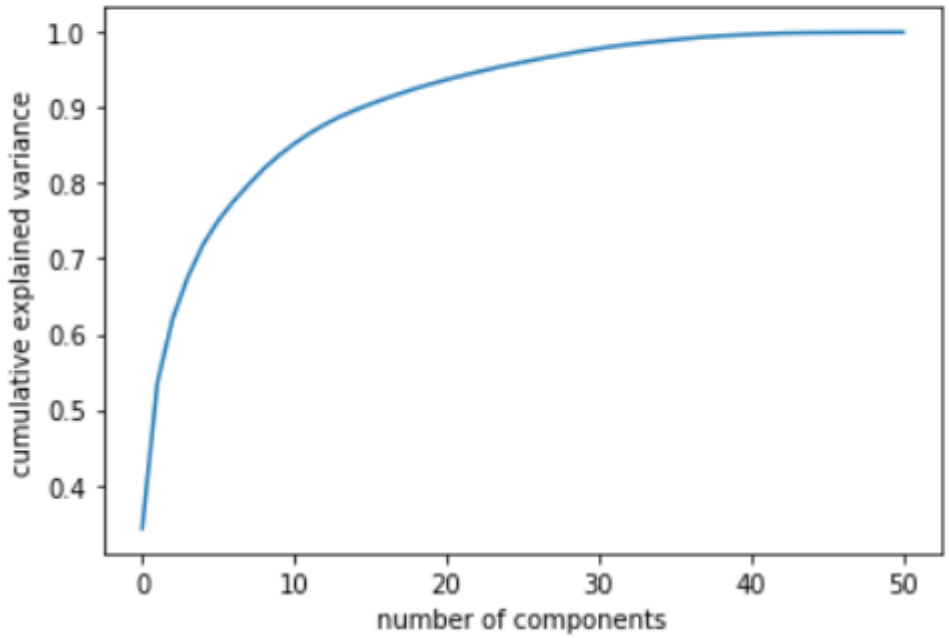
```
data.index=data['timestamp']
data.isna().sum()
```

```
timestamp         0
sensor_00         0
sensor_01         0
sensor_02         0
sensor_03         0
sensor_04         0
sensor_05         0
sensor_06         0
sensor_07         0
sensor_08         0
sensor_09         0
sensor_10         0
sensor_11         0
sensor_12         0
sensor_13         0
sensor_14         0
sensor_16         0
```

2)Finding eigen values and eigen vectors: Since I have a huge amount of data, I couldn't form the covariance matrix from which I can find correlation between columns and also eigen values. I am getting memory error(program runs out of memory)

So I considered the no of components as 8 for PCA analysis which has total variance of 75-80%

```
Text(0, 0.5, 'cumulative explained variance')
```



3)Clustering using K-Means algorithm: I have applied k-Means to reduced dataframe after performing PCA analysis.

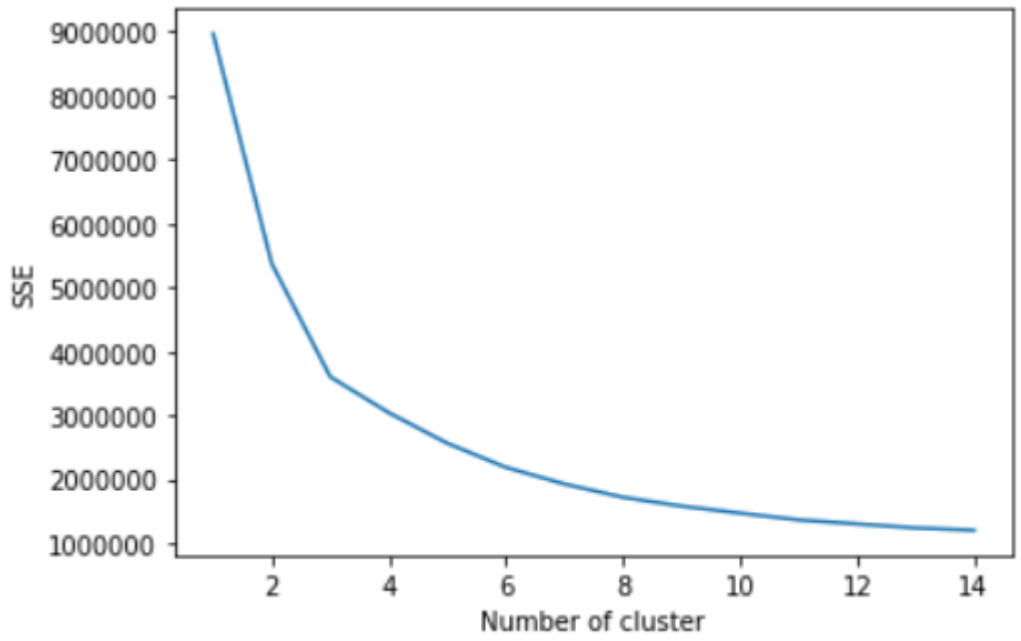
For finding the optimal value of K , I used elbow method(plotting total variance vs no of clusters)

Code:

```
from sklearn.cluster import KMeans
from sklearn import metrics
from scipy.spatial.distance import cdist
sse = {}
k=1
for k in range(1, 15):
    kmeans = KMeans(n_clusters=k, max_iter=1000).fit(x_pca)
    #print(data["clusters"])
    sse[k] = kmeans.inertia_
```

```
plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of cluster")
plt.ylabel("SSE")
plt.show()
```

Result:



From the above plot , I found that the elbow point occurs at 3 clusters.

So, k=3.

Then I applied k-Means for k=3 and divided them into three clusters.

Results:

Centroids:

```
xp["clusters"] = kmeans.labels_
centroids = kmeans.cluster_centers_
print(centroids)
```

```
[[-1.66085662e+00 -9.97062652e+00  2.80978433e-01  1.26875408e+00
   3.15428726e-01  7.08883389e-01  1.16181489e-01  3.72133119e-01]
 [-1.35819929e+00  8.79776948e-01 -1.89406754e-02 -1.55543614e-01
  -5.36727188e-02 -5.16164610e-02 -9.41313749e-03 -3.76523929e-02]
 [ 1.17626783e+01  2.01725993e-02 -4.63006110e-02  3.38101779e-01
   2.01298607e-01 -8.68750431e-02 -6.78663721e-03  3.68873475e-02]]
```

Clusters:

```
xp[(xp['clusters'] == 0)]
```

	0	1	2	3	4	5	6	7	clusters	target
17173	-1.417229	-4.615044	3.168258	3.784061	-7.997853	2.277559	2.625181	-4.967666	0	RECOVERING
17174	-1.365582	-4.768405	3.048815	3.793874	-7.940467	2.130906	2.548821	-4.825132	0	RECOVERING
17175	-1.421200	-4.930435	2.957102	3.720244	-7.962476	2.068623	2.489605	-4.652739	0	RECOVERING
17176	-1.230774	-5.106716	2.855154	3.600279	-7.875190	1.962862	2.381413	-4.616960	0	RECOVERING
17177	-1.293225	-5.214818	2.748897	3.714349	-7.695694	1.772099	2.277457	-4.524251	0	RECOVERING
...	...	...	...	...	...	...	...	...	...	...
195162	-1.675589	-5.271157	3.925632	4.909667	-7.485543	-4.820456	-0.751628	-5.139905	0	NORMAL
195163	-1.688120	-5.430181	3.652068	4.804314	-7.456865	-4.935877	-0.672910	-5.076259	0	NORMAL
195164	-1.727757	-5.616560	3.444859	4.644426	-7.532679	-4.978043	-0.551578	-5.003691	0	NORMAL
195165	-1.644102	-8.796579	4.668454	5.307412	-7.850737	-4.317618	0.601178	-3.165329	0	NORMAL
195166	-1.508901	-8.377784	4.545850	5.421621	-8.352590	-4.863261	0.518444	-3.059728	0	NORMAL

```
xp[(xp['clusters'] == 1)]
```

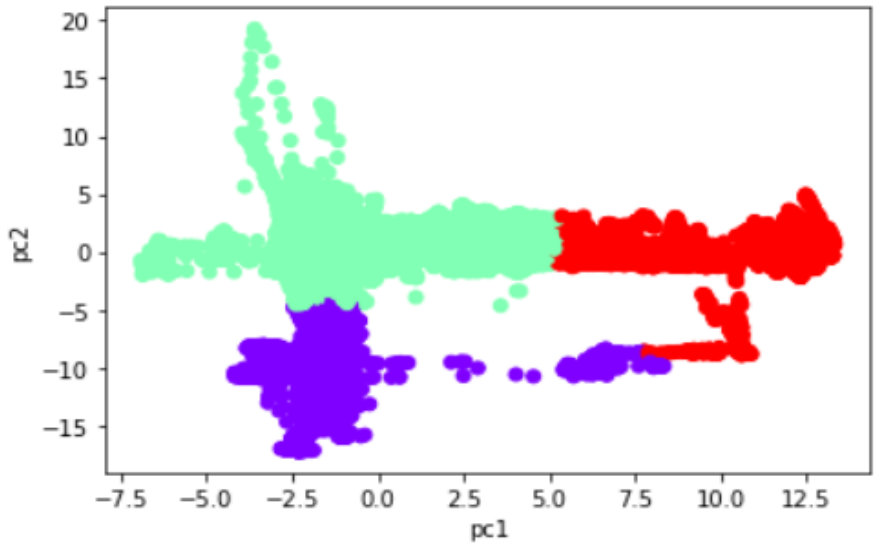
	0	1	2	3	4	5	6	7	clusters	target
0	-0.010524	0.776836	-0.522603	-0.782964	-1.943576	1.535780	-1.131822	-0.176975	1	NORMAL
1	-0.010524	0.776836	-0.522603	-0.782964	-1.943576	1.535780	-1.131822	-0.176975	1	NORMAL
2	-0.151425	0.782444	-0.493742	-0.843024	-2.105139	1.502787	-1.127567	-0.032656	1	NORMAL
3	-0.151886	0.816479	-0.541571	-0.971666	-2.012947	1.586102	-1.169140	-0.009812	1	NORMAL
4	-0.106250	0.929110	-0.407706	-1.025510	-1.989514	1.510769	-1.170467	-0.145796	1	NORMAL
...	...	...	...	...	...	...	...	...	...	...
220315	-2.014969	2.388225	-0.941653	3.506744	-0.979843	1.217796	0.750745	0.072589	1	NORMAL
220316	-2.014980	2.404067	-1.000591	3.411709	-1.028934	1.205897	0.654814	0.075514	1	NORMAL
220317	-2.028597	2.399141	-1.010339	3.368683	-0.962789	1.132215	0.482721	0.104420	1	NORMAL
220318	-1.914576	2.407206	-0.969698	3.409101	-0.933151	1.002620	0.348732	0.128536	1	NORMAL
220319	-1.960072	2.376199	-1.016026	3.325608	-0.919378	0.885308	0.240808	0.120280	1	NORMAL

```
xp[(xp['clusters'] == 2)]
```

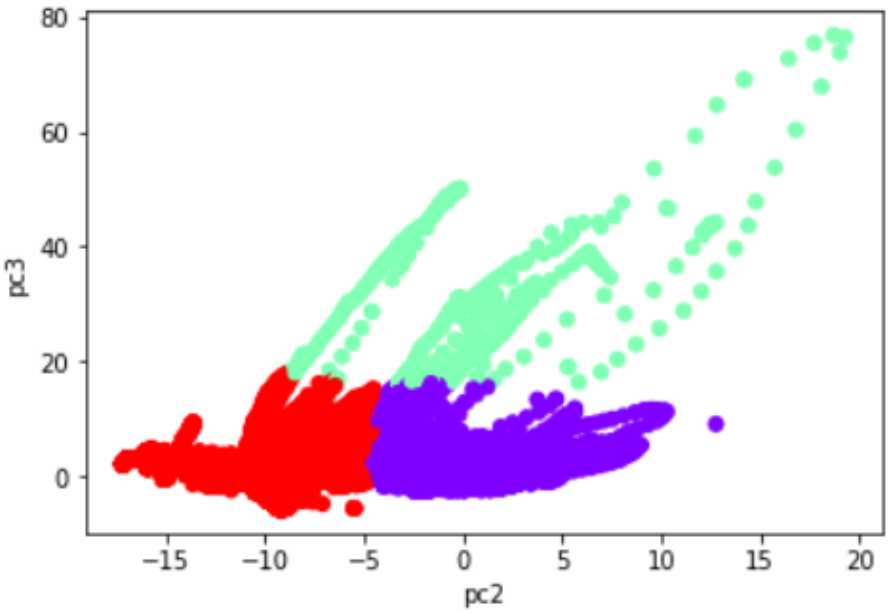
	0	1	2	3	4	5	6	7	clusters	target
39660	8.085840	-1.148561	-1.972194	0.118386	0.815220	-0.636641	-0.230449	-0.196666	2	NORMAL
39661	8.087772	-1.185790	-2.021602	0.114024	0.799860	-0.629380	-0.159363	-0.177415	2	NORMAL
39662	8.074401	-1.270380	-1.983095	0.074233	0.741278	-0.675955	-0.085562	-0.195814	2	NORMAL
39663	8.048415	-1.330921	-1.945650	-0.062065	0.689470	-0.679027	-0.067367	-0.249279	2	NORMAL
39664	8.037099	-1.368592	-1.986239	0.183046	0.794638	-0.833090	-0.034125	-0.216820	2	NORMAL
...	...	...	...	...	...	...	...	...	...	...
184942	5.652461	2.816656	0.551957	2.665838	0.141950	0.037986	1.386006	0.001591	2	NORMAL
184943	5.729103	2.845076	0.659509	2.445690	0.001009	0.089620	1.490149	0.024248	2	NORMAL
184944	5.463188	2.913328	0.663536	2.223170	0.000996	0.138256	1.654859	0.013640	2	NORMAL
207612	6.005808	3.068461	-0.178891	4.895937	-0.265094	0.863667	-0.067764	-0.307972	2	NORMAL
207613	5.374704	3.100638	-0.183182	4.782771	-0.074652	0.747049	0.190755	-1.183274	2	NORMAL

Plotting of Clusters:

Pc1 vs Pc2



Pc2 vs Pc3



4)SVM : I tried for various test sizes, kernels,regularisations(C)and gamma values, but then I found that there is not much in accuracy. Accuracy is always between 0.98-0.99 for every trail.

Accuracy for test_size=0.3, kernel=rbf,regularization (C)=1,gamma=1 :

```
from sklearn import metrics
print("accuracy:",metrics.accuracy_score(y_test,y_pred=pred))
#print("precision:",metrics.precision_score(y_test,y_pred=pred))
```

```
accuracy: 0.9987896393125151
```

Implementing HMM : I couldn't apply HMM using Pomegranate package because it is considering only float values for observation sequence, but our observations are of string datatype(i.e. NORMAL,RECOVERING). I also tried hmms package but it don't have Baum-Welch Algorithm . So, I have used hidden_markov package instead of pomegranate.

HMM has four algorithms:

1)Forward Algorithm

2)Backward Algorithm

3)Viterbi Algorithm

4)Forward-Backward Algorithm(Baum-Welch Algorithm)

The first three Algorithms are used when the HMM model(i.e. start probability ,transition and emission probabilities) and observations are already given.

The fourth Algorithm is used only when observations are given and not the HMM model. So, in this algorithm the start ,transition ,emission probabilities are initiated randomly and then this algorithm looks at the observation(sequence) and updates the HMM(i.e the probabilities) after each iteration.

Therefore we can only use Baum-Welch Algorithm for the given dataset since HMM model is not mentioned.

CODE:

Initiation:

I considered the unknown states as 3 and I used target column(i.e. normal,recovering ,broken) as the observation and intiated with random probabilities.

```
pip install hidden_markov
```

```
Requirement already satisfied: hidden_markov in c:\programdata\anaconda3\lib\site-packages (0.3.2)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from hidden_markov) (1.18.1)
Note: you may need to restart the kernel to use updated packages.
```

```
from hidden_markov import hmm
states = ('s', 't','u')
possible_observation = ('NORMAL','RECOVERING','BROKEN' )
sequence=xp['target'].to_list()
quantities_observations = [1]
start_probability = np.matrix( '0.5 0.2 0.3 ')
transition_probability = np.matrix('0.6 0.2 0.2; 0.2 0.1 0.7 ;0.5 0.1 0.4 ')
emission_probability = np.matrix( '0.3 0.5 0.2; 0.8 0.1 0.1;0.2 0.6 0.2' )
test = hmm(states,possible_observation,start_probability,transition_probability,emission_probability)
```

I found out the log probability of the sequence for the randomly intiated HMM model.

```
observation_tuple = []
observation_tuple.extend( [sequence] )
prob = test.log_prob(observation_tuple, quantities_observations)
print ("probability of sequence with original parameters : %f"%(prob))
```

```
probability of sequence with original parameters : -240669.836575
```

Implementation:

This is the code for applying Baum-Welch Algorithm using the hidden_markov package in python and I iterated for 100000 times. The probabilities are not varying that much for more than 100000 times. So, I finalized the emission ,transition ,start probabilities after 100000 iterations.

```
num_iter=10000
emission,transition,start = test.train_hmm(observation_tuple,num_iter,quantities_observations)
```

RESULTS:

The finalized emission, transition, start probabilities are as follows

```
emission
```

```
matrix([[9.17082083e-01, 8.28936917e-02, 2.42251731e-05],
        [9.85514928e-01, 1.44722887e-02, 1.27831758e-05],
        [9.13220068e-01, 8.67150321e-02, 6.49001949e-05]])
```

```
transition
```

```
matrix([[0.56085004, 0.31698909, 0.12216086],
        [0.25692349, 0.25311244, 0.48996407],
        [0.53751167, 0.18512696, 0.27736136]])
```

```
start
```

```
matrix([[0.47139106, 0.37749188, 0.15111706]])
```

Finding the final log probability of the sequence :

```
prob = test.log_prob(observation_tuple, quantities_observations)
prob
```

```
-53786.77717320992
```

We can observe that the log probability of the occurance of the sequence is increased very much as compared to the previous one before applying the algorithm.

I also found that the optimal number of hidden states as 3 by comparing the final log probabilities for states 2,3,4 and found out that the final log probability for state=3 is more than other states.

Summary :

- Descriptive Analysis gave the outlook of the dataset(mean,std,quartiles).
- PCA analysis reduced the no of independent variables(i.e.53 to 8).
- Imputation method replaced null values with the mean of the respective column.
- Clustered the dataset by using K-Means Algorithm and K in this is found by using Elbow method.
- Support Vector Machine (SVM) Algorithm is used on the reduced dataset and found that the accuracy lies between 98-99% after testing with changing all the factors.
- HMM algorithm is applied (i.e. Baum-Welch Algorithm) and found the start ,transition ,emission probabilities.