CS 5433: Bigdata Management
Programming Assignment 3
Task5 – Identifying the Principal Components

Group 4

Task 5: Identify the principal components are the features which contribute most to the prediction

Description of Dataset:

In this task we are using the dataset which is the output of the Task 1 i.e., the dataset which we have obtained after performing the data correction. The dataset consists of 10 columns and 500 records without any null values and out of range values. The columns of the dataset are,

- Institute ID which is a "Double" column
- Name – Name of the university/institute of type "Double"
- City – Name of the city where university is located, which is of type "Double"
- State – Name of the State where university is located, which is of type "Double"
- PR Score – PR Score of the university which is of type "Double"
- PR Rank – PR Rank of the university which is of type "Double"
- PR Score – PR Score of the university which is of type "Double"
- Score – Score of the university which is of type "Double"
- Year –Year (contains values 2017,2018,2019,2020 & 2021) is of type "Double"
- Rank – Rank of the university which is of type "Double"

  We are using 3 of these columns for identifying principal component analysis and we included columns which are filled after performing first task.

  - PR Score of the university
  - Score of the university
  - PR Rank of the university

Principal Component Analysis: Principal component analysis is a popular dimensional reduction technique for large datasets. We know that processing will be slow for large datasets it will increase the data loss. So, this dimensionality reduction technique increases the interpretability, and it will decrease the information loss. This technique is used to find principal components, contribution rate. It is also used to solve difficult problems like eigen vector problem. Finally, this technique is used to simplify the complexity nature of large datasets and without loss of data.

Approach:

Below are the steps we have followed to complete Task5 of this assignment,

1. At first, we have created a python file("Assign3_Group4_Task5.py") in the Hadoop cluster as shown below. This file contains the below code,

2. **Code Explanation:**

   a. Imported the required libraries

- from pyspark.sql import SparkSession → This library is imported to create a sparksession. This sparksession can be used to create a dataframe.
- from pyspark.ml.feature import StringIndexer,VectorAssembler → StringIndexer library is imported for converting the String columns into Double type and VectorAssembler is imported for merging multiple columns into a vector column.
- from pyspark.ml.feature import MinMaxScaler → By Using column summary statistics, MinMaxScaler rescales each feature to a common range [min, max] linearly.
- from pyspark.sql.functions import udf – It is imported for creating a user defined function (UDF).
- from pyspark.sql.types import DoubleType – This library is imported for representing the double precision floats.
- from pyspark.sql.types import * - It is imported for using the pyspark sql datatypes.
- from pyspark.ml import Pipeline – Pipeline is imported to run the stages in sequence.
- from pyspark.ml.functions import vector_to_array - It is imported for Converting a column of MLlib sparse/dense vectors into a column of dense arrays.
- from pyspark.sql.functions import concat_ws,col – It I imported for concatenating multiple string columns into a single column with a given delimiter.
- from pyspark.ml.regression import LinearRegression – It is imported to perform Linear Regression on the training data.

   b. Created Spark Session

spark = SparkSession.builder.appName("Assign3_Group4_Task5").getOrCreate() → Here, we provided the name to our application by setting a string "Assign3_Group4_Task5" to.appName() as a parameter. Next, used .getOrCreate() to create and instantiate SparkSession into our object "spark".

c. Function used to read csv file into PySpark DataFrame

```
def getDataFram(spark, path_to_input):
  return spark.read.csv(path_to_input,header=True,inferSchema=True,
mode="DROPMALFORMED", encoding='UTF-8')

path_to_input='hdfs://hadoop-
nn001.cs.okstate.edu:9000/user/sdarapu/Assign3_Group4_Task1_Output_inpfor_
Task2-4/part-00000-571e77d2-85ae-4579-92f8-dd4dc788ab7f-c000.csv'

df = getDataFram(spark, path_to_input)
```

It will take the path and gives to getDataFram() method and then return the spark dataframe. This spark dataframe will be used for next process.

d. Print the PySpark data frame

df.show(truncate=False) → Prints the schema of the dataframe “df”.

e. Use UDF rounding

un_wraper_li = udf(lambda x: round(float(list(x)[0]),3), DoubleType())→ This is used to convert values having double type into float values and round them up to 3 decimals,

f. Create a column list and scale those values in the mentioned columns

```
col_list = ["PR Score","PR Rank","Score"]

indexx = 0

while indexx < len(col_list):

  assembler=
  VectorAssembler(inputCols=[col_list[indexx]],outputCol=col_list[indexx]+"_V
  ect")

  scaler=MinMaxScaler(inputCol=col_list[indexx]+"_Vect",
  outputCol=col_list[indexx]+"_Scaled")

  pipeline = Pipeline(stages=[assembler, scaler])

  df = pipeline.fit(df).transform(df).withColumn(col_list[indexx]+"_Scaled",
un_wraper_li(col_list[indexx]+"_Scaled")).drop(col_list[indexx]+"_Vect")
```

indexx = indexx +1

→This loop is used to iterate all the values in the columns mentioned in the col_list and use VectorAssembler which is a transformer that combines a given list of columns into a single vector column. Scaler is used to scale all the values to the min values. These assembler and scaler are passed through pipeline and use transform and fit methods and at last we can get the scaled columns attached to dataframe.

g. Function used to calculate the variable from scaled values

def get_featureVec(df, assembler):

  return assembler.transform(df)

assembler = VectorAssembler(inputCols=df.columns[9:], outputCol="variable")

vec_fetrd = get_featureVec(df, assembler)

vec_fetrd.show()

→This function is used to transform assembler and assemble the columns which are scaled, and this calculate the "variable" vector. This vector contains the scaled values of "PR Rank", "PR Score" and "Score" columns.

h. Function used to calculate Standardized variate

def getModelFit(scaler, vec_fetrd):

  return scaler.fit(vec_fetrd)

scaler = StandardScaler(inputCol="variable", outputCol="Standardized variate", withStd=True, withMean=True)

s_mdl = getModelFit(scaler, vec_fetrd)

def getTrans(s_mdl, vec_fetrd):

  return s_mdl.transform(vec_fetrd)

VECTOR_STD = getTrans(s_mdl, vec_fetrd)

→This function takes "variable" vector as input and calculate Standardized variate after passing through Standard Scaler function and calculates the desired variable

i. Displaying the standardized variate

```
VECTOR_STD.select("Standardized variate").show(truncate=False)
```

j. Shows descriptive statistics of training and test data

```
def getPCA(i_k, i_col, o_col ):
        return PCA(k=i_k, inputCol=i_col, outputCol=o_col)
i_k=3
i_col="Standardized variate"
o_col="Main component score"
pca = getPCA(i_k, i_col, o_col)
def getModelFit_2(pca, VECTOR_STD ):
        return pca.fit(VECTOR_STD)
MDL_PCA = getModelFit_2(pca, VECTOR_STD )
```

→The above function is used to calculate "Main Component Score" by taking Standardized variate as input. K=3 mention takes three columns as input.

k. Printing Eigen vector values

```
print(MDL_PCA.pc)
```

```
 ############ Eigenvector ####
DenseMatrix([[-0.60496295,  0.22328741,  0.76430528],
             [ 0.54284721,  0.81788602,  0.19073374],
             [-0.58252616,  0.53028783, -0.61600169]])
```

l. Printing Contribution rate

```
print("################ Contribution rate #################")
print(MDL_PCA.explainedVariance)
```

```
################# Contribution rate ##################
[0.7016699649623026,0.18292388625092215,0.11540614878677524]
```

Note: If we add all the contribution rates then all should be equal to "1".

m. Print Main Component Score

```
FINAL_PCA_SCORE = MDL_PCA.transform(VECTOR_STD).select("Main component score")
print("################ FINAL PCA SCOREs ################")
FINAL_PCA_SCORE.show(truncate=False)
```

```
+--------------------------------------------------------------+
|Main component score                                          |
+--------------------------------------------------------------+
|[-2.8475860138937548,0.6555509410225909,-0.35683979506912467] |
|[-2.5099772855353377,0.435501074409122,-0.2740779549223128]   |
|[-1.7984913915960998,0.2059718424800403,-0.7896178318926204]  |
|[-2.297227041952679,0.2750076908267295,-0.066332483062998]    |
|[-1.633631899586792,0.13803161617743287,-0.7674230806244504]  |
|[-1.7297355452212397,0.05558721220075158,-0.5405054979407649] |
|[-1.4584904739496252,0.028751738701833673,-0.6722479873372307]|
|[-0.5393196384054464,0.06762446283122203,-1.0791996575067304] |
|[-0.32705457063862275,0.08418349046202545,-1.0290927457584742]|
|[-2.134938711810865,-0.06915479280594028,0.62247500112131]    |
|[-1.4931101342136794,-0.226352315009137,-0.10501235838986739] |
|[-0.42748552233609305,-0.06487357201970728,-0.9293586660842779]|
|[-0.5871387551295573,-0.3881610288025444,-0.5391620782140139] |
|[0.6016502807444919,0.501002841853035,-0.8831213707334981]    |
|[0.4290420817523568,0.1911494100642601,-0.8387683001904124]   |
|[-0.3286922645747341,-0.5200349129081956,-0.4091378195806554] |
|[0.19195250966108768,-0.23799128419143845,-0.5995450667319702] |
|[0.7589025536880172,0.2958675372014221,-0.6761104351857921]   |
|[0.05999236589841709,-0.4529103247197494,-0.4527609322058134] |
|[0.8652749272124152,0.17400055304060855,-0.5563118272650989]  |
+--------------------------------------------------------------+
only showing top 20 rows
```

3. Steps to execute the code:

i. To run the code, we have executed below command as shown below.

```
sdarapu@hadoop-nn001:~$ spark-submit /home/sdarapu/Assign3_Group4_Task5.py
```

Fig 5,1: Command to execute

ii. The above command executes as follows.

```
sdarapu@hadoop-nn001:~$ spark-submit /home/sdarapu/Assign3_Group4_Task5.py
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/usr/local/spark-3.0.1-bin-hadoop3.2/jars/spark-unsafe_2.12-3.0.1.jar
DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
2022-04-29 23:46:06,383 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where app
2022-04-29 23:46:07,441 INFO spark.SparkContext: Running Spark version 3.0.1
2022-04-29 23:46:07,488 INFO resource.ResourceUtils: ==============================================================
2022-04-29 23:46:07,491 INFO resource.ResourceUtils: Resources for spark.driver:

2022-04-29 23:46:07,492 INFO resource.ResourceUtils: ==============================================================
2022-04-29 23:46:07,492 INFO spark.SparkContext: Submitted application: Assign3_Group4_Task5
2022-04-29 23:46:07,549 INFO spark.SecurityManager: Changing view acls to: sdarapu
2022-04-29 23:46:07,549 INFO spark.SecurityManager: Changing modify acls to: sdarapu
2022-04-29 23:46:07,549 INFO spark.SecurityManager: Changing view acls groups to:
2022-04-29 23:46:07,550 INFO spark.SecurityManager: Changing modify acls groups to:
2022-04-29 23:46:07,550 INFO spark.SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users  with view permissions: Set(
 permissions: Set(); users  with modify permissions: Set(sdarapu); groups with modify permissions: Set()
2022-04-29 23:46:07,825 INFO util.Utils: Successfully started service 'sparkDriver' on port 45873.
2022-04-29 23:46:07,856 INFO spark.SparkEnv: Registering MapOutputTracker
2022-04-29 23:46:07,888 INFO spark.SparkEnv: Registering BlockManagerMaster
2022-04-29 23:46:07,909 INFO storage.BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology informa
2022-04-29 23:46:07,909 INFO storage.BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
2022-04-29 23:46:07,954 INFO spark.SparkEnv: Registering BlockManagerMasterHeartbeat
2022-04-29 23:46:07,967 INFO storage.DiskBlockManager: Created local directory at /tmp/blockmgr-a3653f5e-0efe-45b9-bdbd-9cc3a77536ee
2022-04-29 23:46:07,991 INFO memory.MemoryStore: MemoryStore started with capacity 434.4 MiB
2022-04-29 23:46:08,033 INFO spark.SparkEnv: Registering OutputCommitCoordinator
2022-04-29 23:46:08,135 INFO util.log: Logging initialized @3726ms to org.sparkproject.jetty.util.log.Slf4jLog
2022-04-29 23:46:08,219 INFO server.Server: jetty-9.4.z-SNAPSHOT; built: 2019-04-29T20:42:08.989Z; git: e1bc35120a6617ee3df052294e433f3a25ce7097;
ntu0.20.04.1
2022-04-29 23:46:08,238 INFO server.Server: Started @3832ms
```

Fig 5, 2: Execution Results

```
2022-04-29 23:46:08,643 INFO client.RMProxy: Connecting to ResourceManager at hadoop-nn001.cs.okstate.edu/192.168.122.2:8032
2022-04-29 23:46:08,917 INFO yarn.Client: Requesting a new application from cluster with 12 NodeManagers
2022-04-29 23:46:09,363 INFO conf.Configuration: resource-types.xml not found
2022-04-29 23:46:09,363 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2022-04-29 23:46:09,379 INFO yarn.Client: Verifying our application has not requested more than the maximum memory capability of t
2022-04-29 23:46:09,380 INFO yarn.Client: Will allocate AM container, with 896 MB memory including 384 MB overhead
2022-04-29 23:46:09,381 INFO yarn.Client: Setting up container launch context for our AM
2022-04-29 23:46:09,383 INFO yarn.Client: Setting up the launch environment for our AM container
2022-04-29 23:46:09,389 INFO yarn.Client: Preparing resources for our AM container
2022-04-29 23:46:09,428 WARN yarn.Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading librari
2022-04-29 23:46:12,547 INFO yarn.Client: Uploading resource file:/tmp/spark-0eb440e6-c634-42cf-b0c8-62e2d318ba1f/__spark_libs__18
001.cs.okstate.edu:9000/user/sdarapu/.sparkStaging/application_1647031195237_1528/__spark_libs__18036825927996035996.zip
2022-04-29 23:46:15,582 INFO yarn.Client: Uploading resource file:/usr/local/spark/python/lib/pyspark.zip -> hdfs://hadoop-nn001.c
ing/application_1647031195237_1528/pyspark.zip
2022-04-29 23:46:15,642 INFO yarn.Client: Uploading resource file:/usr/local/spark/python/lib/py4j-0.10.9-src.zip -> hdfs://hadoop
parkStaging/application_1647031195237_1528/py4j-0.10.9-src.zip
2022-04-29 23:46:15,901 INFO yarn.Client: Uploading resource file:/tmp/spark-0eb440e6-c634-42cf-b0c8-62e2d318ba1f/__spark_conf__17
001.cs.okstate.edu:9000/user/sdarapu/.sparkStaging/application_1647031195237_1528/__spark_conf__.zip
2022-04-29 23:46:15,963 INFO spark.SecurityManager: Changing view acls to: sdarapu
2022-04-29 23:46:15,964 INFO spark.SecurityManager: Changing modify acls to: sdarapu
2022-04-29 23:46:15,964 INFO spark.SecurityManager: Changing view acls groups to:
2022-04-29 23:46:15,964 INFO spark.SecurityManager: Changing modify acls groups to:
2022-04-29 23:46:15,964 INFO spark.SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users  with view p
 permissions: Set(); users  with modify permissions: Set(sdarapu); groups with modify permissions: Set()
2022-04-29 23:46:15,988 INFO yarn.Client: Submitting application application_1647031195237_1528 to ResourceManager
2022-04-29 23:46:16,227 INFO impl.YarnClientImpl: Submitted application application_1647031195237_1528
2022-04-29 23:46:17,233 INFO yarn.Client: Application report for application_1647031195237_1528 (state: ACCEPTED)
2022-04-29 23:46:17,238 INFO yarn.Client:
         client token: N/A
         diagnostics: AM container is launched, waiting for AM container to Register with RM
         ApplicationMaster host: N/A
         ApplicationMaster RPC port: -1
         queue: default
         start time: 1651293976000
```

Fig 5, 3: Execution Results

```
2022-04-29 23:46:31,609 INFO scheduler.DAGScheduler: Got job 0 (csv at NativeMethodAccessorImpl.java:0) with 1 output partitions
2022-04-29 23:46:31,610 INFO scheduler.DAGScheduler: Final stage: ResultStage 0 (csv at NativeMethodAccessorImpl.java:0)
2022-04-29 23:46:31,611 INFO scheduler.DAGScheduler: Parents of final stage: List()
2022-04-29 23:46:31,613 INFO scheduler.DAGScheduler: Missing parents: List()
2022-04-29 23:46:31,620 INFO scheduler.DAGScheduler: Submitting ResultStage 0 (MapPartitionsRDD[3] at csv at NativeMethodAccessorImpl.java:0),
2022-04-29 23:46:31,694 INFO memory.MemoryStore: Block broadcast_1 stored as values in memory (estimated size 10.7 KiB, free 434.2 MiB)
2022-04-29 23:46:31,707 INFO memory.MemoryStore: Block broadcast_1_piece0 stored as bytes in memory (estimated size 5.3 KiB, free 434.2 MiB)
2022-04-29 23:46:31,708 INFO storage.BlockManagerInfo: Added broadcast_1_piece0 in memory on hadoop-nn001:45659 (size: 5.3 KiB, free: 434.4 Mi
2022-04-29 23:46:31,709 INFO spark.SparkContext: Created broadcast 1 from broadcast at DAGScheduler.scala:1223
2022-04-29 23:46:31,727 INFO scheduler.DAGScheduler: Submitting 1 missing tasks from ResultStage 0 (MapPartitionsRDD[3] at csv at NativeMetho
 tasks are for partitions Vector(0))
2022-04-29 23:46:31,728 INFO cluster.YarnScheduler: Adding task set 0.0 with 1 tasks
2022-04-29 23:46:31,778 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 0.0 (TID 0, hadoop-dn003.cs.okstate.edu, executor 1, partiti
2022-04-29 23:46:32,097 INFO storage.BlockManagerInfo: Added broadcast_1_piece0 in memory on hadoop-dn003.cs.okstate.edu:34381 (size: 5.3 KiB,
2022-04-29 23:46:32,807 INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in memory on hadoop-dn003.cs.okstate.edu:34381 (size: 28.7 KiB
2022-04-29 23:46:33,431 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 1664 ms on hadoop-dn003.cs.okstate.edu (execu
2022-04-29 23:46:33,436 INFO cluster.YarnScheduler: Removed TaskSet 0.0, whose tasks have all completed, from pool
2022-04-29 23:46:33,444 INFO scheduler.DAGScheduler: ResultStage 0 (csv at NativeMethodAccessorImpl.java:0) finished in 1.804 s
2022-04-29 23:46:33,451 INFO scheduler.DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
2022-04-29 23:46:33,452 INFO cluster.YarnScheduler: Killing all running tasks in stage 0: Stage finished
2022-04-29 23:46:33,455 INFO scheduler.DAGScheduler: Job 0 finished: csv at NativeMethodAccessorImpl.java:0, took 1.866442 s
2022-04-29 23:46:33,493 INFO codegen.CodeGenerator: Code generated in 17.8174 ms
2022-04-29 23:46:33,563 INFO datasources.FileSourceStrategy: Pruning directories with:
2022-04-29 23:46:33,564 INFO datasources.FileSourceStrategy: Pushed Filters:
2022-04-29 23:46:33,564 INFO datasources.FileSourceStrategy: Post-Scan Filters:
2022-04-29 23:46:33,565 INFO datasources.FileSourceStrategy: Output Data Schema: struct<value: string>
2022-04-29 23:46:33,580 INFO memory.MemoryStore: Block broadcast_2 stored as values in memory (estimated size 179.1 KiB, free 434.0 MiB)
2022-04-29 23:46:33,607 INFO memory.MemoryStore: Block broadcast_2_piece0 stored as bytes in memory (estimated size 28.7 KiB, free 434.0 MiB)
2022-04-29 23:46:33,608 INFO storage.BlockManagerInfo: Added broadcast_2_piece0 in memory on hadoop-nn001:45659 (size: 28.7 KiB, free: 434.3 M
2022-04-29 23:46:33,610 INFO spark.SparkContext: Created broadcast 2 from csv at NativeMethodAccessorImpl.java:0
```

Fig 5, 4: Execution Results

Output displayed on the Console:

INSTITUTE ID	NAME	CITY	STATE	PR Score	PR Rank	Score	Year	Rank	PR Score_Scaled	PR Rank_Scaled	Score_Scaled
241.0	22.0	2.0	4.0	47.27	3.0	61.53	2017.0	2.0	0.473	0.008	0.954
306.0	4.0	39.0	3.0	44.01	4.0	58.92	2017.0	3.0	0.44	0.012	0.914
264.0	19.0	7.0	6.0	28.81	9.0	57.32	2017.0	5.0	0.288	0.033	0.89
284.0	3.0	0.0	0.0	43.94	5.0	56.5	2017.0	6.0	0.439	0.017	0.877
235.0	39.0	5.0	10.0	27.06	11.0	56.3	2017.0	7.0	0.271	0.041	0.874
309.0	38.0	21.0	4.0	30.76	7.0	55.37	2017.0	8.0	0.308	0.025	0.86
282.0	2.0	4.0	0.0	26.12	12.0	54.7	2017.0	9.0	0.261	0.046	0.85
254.0	33.0	3.0	1.0	11.2	35.0	52.81	2017.0	10.0	0.112	0.141	0.821
238.0	21.0	2.0	4.0	9.73	42.0	51.75	2017.0	12.0	0.097	0.17	0.805
315.0	85.0	31.0	13.0	49.96	2.0	51.46	2017.0	13.0	0.5	0.004	0.8
312.0	43.0	40.0	0.0	32.95	6.0	51.36	2017.0	14.0	0.33	0.021	0.799
231.0	133.0	2.0	4.0	11.52	34.0	51.2	2017.0	15.0	0.115	0.137	0.796
316.0	10.0	7.0	6.0	17.15	20.0	48.9	2017.0	16.0	0.171	0.079	0.761
269.0	81.0	4.0	0.0	3.53	86.0	48.84	2017.0	17.0	0.035	0.353	0.76
314.0	44.0	7.0	6.0	4.71	69.0	48.19	2017.0	19.0	0.047	0.282	0.75
255.0	166.0	6.0	12.0	15.57	23.0	46.72	2017.0	20.0	0.156	0.091	0.728
278.0	17.0	1.0	1.0	8.7	47.0	46.45	2017.0	21.0	0.087	0.191	0.724
310.0	6.0	0.0	0.0	3.79	84.0	46.45	2017.0	21.0	0.038	0.344	0.724
298.0	27.0	5.0	10.0	11.16	36.0	45.52	2017.0	23.0	0.112	0.145	0.71
226.0	157.0	47.0	8.0	3.83	83.0	44.99	2017.0	24.0	0.038	0.34	0.702

only showing top 20 rows

Fig 5,5: Data after performing Scaling

INSTITUTE ID	NAME	CITY	STATE	PR Score	PR Rank	Score	Year	Rank	PR Score_Scaled	PR Rank_Scaled	Score_Scaled	variable
241.0	22.0	2.0	4.0	47.27	3.0	61.53	2017.0	2.0	0.473	0.008	0.954	[0.473,0.008,0.954]
306.0	4.0	39.0	3.0	44.01	4.0	58.92	2017.0	3.0	0.44	0.012	0.914	[0.44,0.012,0.914]
264.0	19.0	7.0	6.0	28.81	9.0	57.32	2017.0	5.0	0.288	0.033	0.89	[0.288,0.033,0.89]
284.0	3.0	0.0	0.0	43.94	5.0	56.5	2017.0	6.0	0.439	0.017	0.877	[0.439,0.017,0.877]
235.0	39.0	5.0	10.0	27.06	11.0	56.3	2017.0	7.0	0.271	0.041	0.874	[0.271,0.041,0.874]
309.0	38.0	21.0	4.0	30.76	7.0	55.37	2017.0	8.0	0.308	0.025	0.86	[0.308,0.025,0.86]
282.0	2.0	4.0	0.0	26.12	12.0	54.7	2017.0	9.0	0.261	0.046	0.85	[0.261,0.046,0.85]
254.0	33.0	3.0	1.0	11.2	35.0	52.81	2017.0	10.0	0.112	0.141	0.821	[0.112,0.141,0.821]
238.0	21.0	2.0	4.0	9.73	42.0	51.75	2017.0	12.0	0.097	0.17	0.805	[0.097,0.17,0.805]
315.0	85.0	31.0	13.0	49.96	2.0	51.46	2017.0	13.0	0.5	0.004	0.8	[0.5,0.004,0.8]
312.0	43.0	40.0	0.0	32.95	6.0	51.36	2017.0	14.0	0.33	0.021	0.799	[0.33,0.021,0.799]
231.0	133.0	2.0	4.0	11.52	34.0	51.2	2017.0	15.0	0.115	0.137	0.796	[0.115,0.137,0.796]
316.0	10.0	7.0	6.0	17.15	20.0	48.9	2017.0	16.0	0.171	0.079	0.761	[0.171,0.079,0.761]
269.0	81.0	4.0	0.0	3.53	86.0	48.84	2017.0	17.0	0.035	0.353	0.76	[0.035,0.353,0.76]
314.0	44.0	7.0	6.0	4.71	69.0	48.19	2017.0	19.0	0.047	0.282	0.75	[0.047,0.282,0.75]
255.0	166.0	6.0	12.0	15.57	23.0	46.72	2017.0	20.0	0.156	0.091	0.728	[0.156,0.091,0.728]
278.0	17.0	1.0	1.0	8.7	47.0	46.45	2017.0	21.0	0.087	0.191	0.724	[0.087,0.191,0.724]
310.0	6.0	0.0	0.0	3.79	84.0	46.45	2017.0	21.0	0.038	0.344	0.724	[0.038,0.344,0.724]
298.0	27.0	5.0	10.0	11.16	36.0	45.52	2017.0	23.0	0.112	0.145	0.71	[0.112,0.145,0.71]
226.0	157.0	47.0	8.0	3.83	83.0	44.99	2017.0	24.0	0.038	0.34	0.702	[0.038,0.34,0.702]

only showing top 20 rows

Fig 5,6: Found out Variable vector after applying assembler on “Scaled” columns

```
+--------------------------------------------------------------+
|Standardized variate                                          |
+--------------------------------------------------------------+
|[1.5963257792085033,-1.0776995573167338,2.2262379596389934]   |
|[1.4062059524738282,-1.0586198365465274,1.8619008421467076]   |
|[0.5305025081201714,-0.9584513025029443,1.6432985716513355]   |
|[1.4004447456030804,-1.0347701855837697,1.5248890084663425]   |
|[0.4325619913174600 4,-0.9202918609625316,1.497563724654421]  |
|[0.6457266455351264,-0.9966107440433569,1.3700457335321206]   |
|[0.374949922609982 6,-0.896442209999773 6,1.2789614541590488] |
|[-0.48346990113143085,-0.4432988417073735,1.014817043977141]  |
|[-0.569888004192647,-0.3049708661233775,0.8690821969802272]   |
|[1.7518783647186924,-1.09677927808694,0.8235400572936913]     |
|[0.7724731966915768,-1.0156904648135634,0.8144316293563841]   |
|[-0.46618628051918765,-0.4623785624775797,0.7871063455444626] |
|[-0.1435586957573141,-0.7390345136455716,0.4683113677387116]  |
|[-0.927082830179007,0.5679263591135619,0.4592029398014044]    |
|[-0.8579483477300339,0.22926131544239955,0.36811866042833274] |
|[-0.22997679881853028,-0.6817953513349525,0.16773324580757498]|
|[-0.6275000729001244,-0.20480233207979429,0.13129953405834632]|
|[-0.9097992095667636,0.5249969873805976,0.13129953405834632]  |
|[-0.48346990113143085,-0.4242191209371671,0.003781542936045923]|
|[-0.9097992095667636,0.5059172666103916,-0.06908588056241144] |
+--------------------------------------------------------------+
only showing top 20 rows
```

Fig 5,7: Standardized Variate

```
 ############ Eigenvector ####
DenseMatrix([[-0.60496295,  0.22328741,  0.76430528],
              [ 0.54284721,  0.81788602,  0.19073374],
              [-0.58252616,  0.53028783, -0.61600169]])
```

Fig 5,8: Eigen matrix

```
################ Contribution rate #################
[0.7016699649623026,0.18292388625092215,0.11540614878677524]
```

Fig 5,9: Contribution Rate

```
+--------------------------------------------------------------+
|Main component score                                          |
+--------------------------------------------------------------+
|[-2.8475860138937548,0.6555509410225909,-0.35683979506912467]  |
|[-2.5099772855353377,0.435501074409122,-0.2740779549223128]    |
|[-1.7984913915960998,0.2059718424800403,-0.7896178318926204]   |
|[-2.297227041952679,0.2750076908267295,-0.066332483062998]     |
|[-1.633631899586792,0.13803161617743287,-0.7674230806244504]   |
|[-1.7297355452212397,0.05558721220075158,-0.5405054979407649]  |
|[-1.4584904739496252,0.028751738701833673,-0.6722479873372307] |
|[-0.5393196384054464,0.06762446283122203,-1.0791996575067304]  |
|[-0.32705457063862275,0.08418349046202545,-1.0290927457584742] |
|[-2.134938711810865,-0.06915479280594028,0.62247500112131]     |
|[-1.4931101342136794,-0.226352315009137,-0.10501235838986739]  |
|[-0.42748552233609305,-0.06487357201970728,-0.9293586660842779]|
|[-0.5871387551295573,-0.3881610288025444,-0.5391620782140139]  |
|[0.6016502807444919,0.501002841853035,-0.8831213707334981]     |
|[0.4290420817523568,0.1911494100642601,-0.8387683001904124]    |
|[-0.3286922645747341,-0.5200349129081956,-0.4091378195806554]  |
|[0.19195250966108768,-0.23799128419143845,-0.5995450667319702] |
|[0.7589025536880172,0.2958675372014221,-0.6761104351857921]    |
|[0.05999236589841709,-0.4529103247197494,-0.4527609322058134]  |
|[0.865274927212415 2,0.17400055304060855,-0.5563118272650989]   |
+--------------------------------------------------------------+
only showing top 20 rows
```

Fig 5,10: Main Component Score

<u>Discussion of Results</u>

In this Task, Initially we have generated the standardized variate using Standard Scaler and .fit and .transform. And then we have sent this "standardized variate" as an input to the Principal Component Analysis and then we have got the Principal Component Score.

We have seen the Eigenvector and Contribution rate from the model and the Contribution Rate for the 1st One: 76.41 %, 2nd : 15.897%, 3rd: 7.6958% where all the percent’s equal to 100 and this is considered as a validation.

Finally, we have got the Main Component Score from the program.