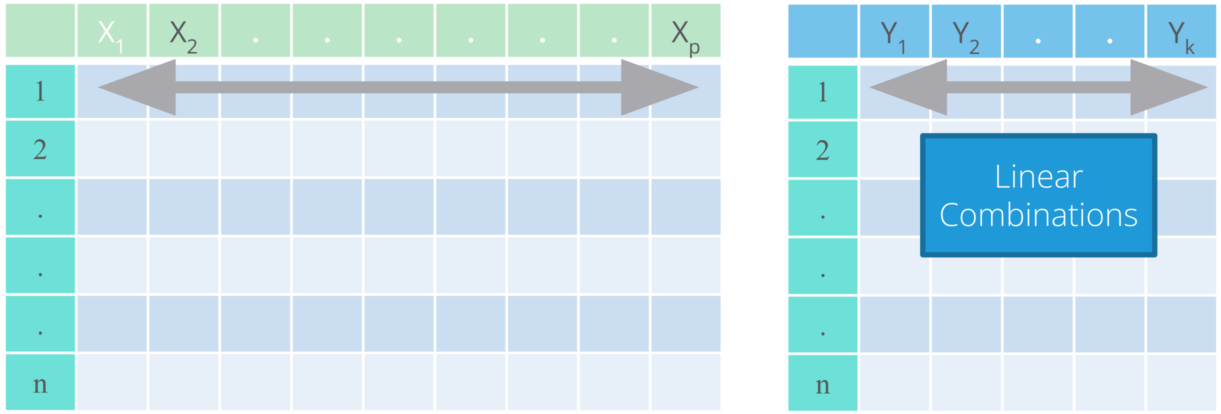
Principal Component Analysis

Learn How to Manage Data Dimensionality Without Losing Information

Contents

Data Reduction

- Summarization of data with p variables by a smaller set of (k) derived variables.
- These k derived variables are **linear combinations of original p variables.**



- In short, **n * p** matrix is **reduced to n * k** matrix.

Case Study – Athletics Records

Background

- Data on national athletics records for various countries is available.

Objective

- To achieve data reduction and obtain score for each country which can be used to rank countries based on athletics records.

Available Information

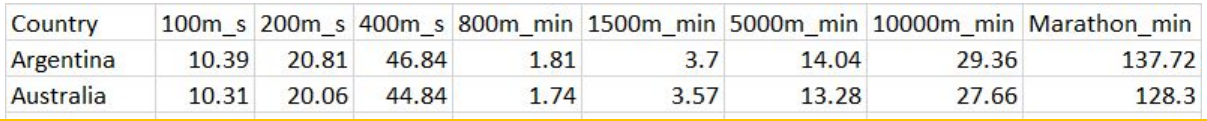
- Data Source: Applied Multivariate Statistical Analysis by Richard A. Johnson , Dean W. Wichern
- Sample size is 55 countries athletics.
- Records for 8 different athletics events – 100 meters to Marathon

Data Snapshot

Athleticsdata

Variables

Observations



Country	100m_s	200m_s	400m_s	800m_min	1500m_min	5000m_min	10000m_min	Marathon_min
Argentina	10.39	20.81	46.84	1.81	3.7	14.04	29.36	137.72
Australia	10.31	20.06	44.84	1.74	3.57	13.28	27.66	128.3

Column	Description	Type	Measurement	Possible Values
Country	Country Name	Categorical	-	-
100m_s	Time for 100 meter running	Continuous	Seconds	Positive Values
200m_s	Time for 200 meter running	Continuous	Seconds	Positive Values
400m_s	Time for 400 meter running	Continuous	Seconds	Positive Values
800m_min	Time for 800 meter running	Continuous	Minutes	Positive Values
1500m_min	Time for 1500 meter running	Continuous	Minutes	Positive Values
5000m_min	Time for 5000 meter running	Continuous	Minutes	Positive Values
10000m_min	Time for 10000 meter running	Continuous	Minutes	Positive Values
Marathon_min	Time for Marathon running	Continuous	Minutes	Positive Values

PCA in Python

```
#Importing data
import pandas as pd
import numpy as np

athletics=pd.read_csv("Athleticsdata.csv")
```

- **read_csv()** is used to import csv file. Our data is stored as an object named **athletics.**

```
# standardize all variables
athletics2=athletics.drop(['Country'], axis=1)


from sklearn.decomposition import PCA
from sklearn.preprocessing import scale

standardisedX = scale(athletics2)
X = pd.DataFrame(standardisedX, index=athletics2.index,
columns=athletics2.columns)
```

- **drop()** is used to remove the column named "Country" from the data.

- The data is standardised using the function **scale()** and stored as a dataframe object X

PCA in Python

```
#Running PCA and creating summary table
pca = PCA().fit(X)
names = ["PC"+str(i) for i in range(1,9)]

SD = list(np.std(pca.transform(X), axis=0))
VarProp = list(pca.explained_variance_ratio_)
CumProp = [np.sum(VarProp[:i]) for i in range(1,9)]

summary = pd.DataFrame(list(zip(SD, VarProp, CumProp)), index=names,
columns=['Standard Deviation','Proportion of Variance','Cumulative
Proportion'])
summary
```

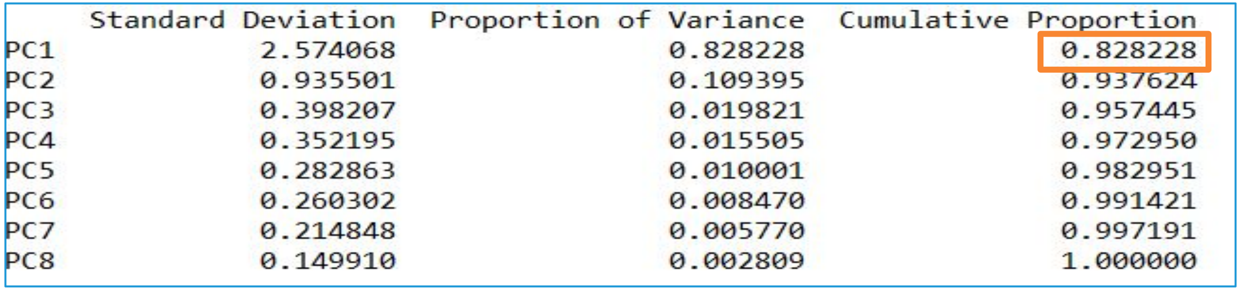
- ❑ Create a vector of names as PC1,PC2 …. PC8.

- ❑ Extract Standard Deviation and Proportion of variance explained. Define Cumulative proportion for the summary table
- ❑ pca.transform(X) computes scores

- ❑ Create a dataframe of summary output

PCA in Python

```
# Output:
```



```
     Standard Deviation  Proportion of Variance  Cumulative Proportion
PC1            2.574068                0.828228               0.828228
PC2            0.935501                0.109395               0.937624
PC3            0.398207                0.019821               0.957445
PC4            0.352195                0.015505               0.972950
PC5            0.282863                0.010001               0.982951
PC6            0.260302                0.008470               0.991421
PC7            0.214848                0.005770               0.997191
PC8            0.149910                0.002809               1.000000
```

Interpretation:

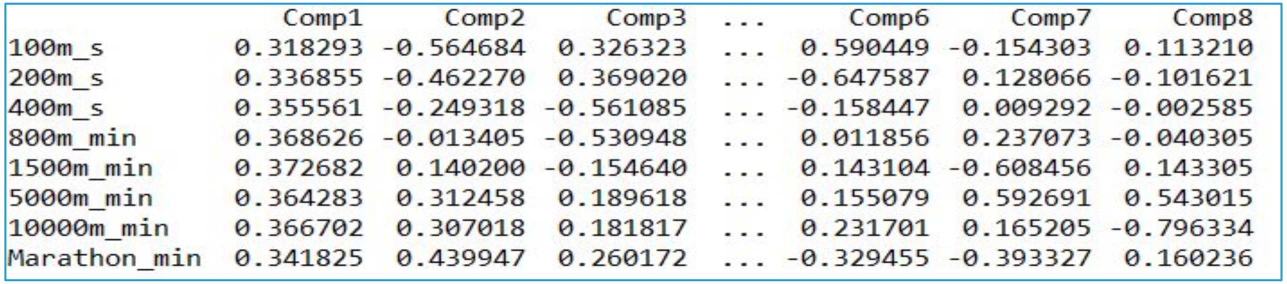
- summary gives **std. deviation (sd), proportion of variance and cumulative proportion**. Variance is nothing but the Eigenvalue of correlation matrix.
- First Principal Component explains 83% of the variation. Note that 8 PC's are derived using 8 variables but first PC explains most of the variation.

PCA in Python - Loadings and Scores

```
#Component Loadings
```

```
rows = X.columns
col = ["Comp"+str(i) for i in range(1, len(X.columns)+1)]
L = pd.DataFrame(list(zip(pca.components_[0],pca.components_[1],
pca.components_[2],pca.components_[3],pca.components_[4],
pca.components_[5],pca.components_[6],pca.components_[7])),index=rows,
columns = col)
L
```

```
# Output:
```



```
                 Comp1     Comp2     Comp3  ...     Comp6     Comp7     Comp8
100m_s        0.318293 -0.564684  0.326323  ...  0.590449 -0.154303  0.113210
200m_s        0.336855 -0.462270  0.369020  ... -0.647587  0.128066 -0.101621
400m_s        0.355561 -0.249318 -0.561085  ... -0.158447  0.009292 -0.002585
800m_min      0.368626 -0.013405 -0.530948  ...  0.011856  0.237073 -0.040305
1500m_min     0.372682  0.140200 -0.154640  ...  0.143104 -0.608456  0.143305
5000m_min     0.364283  0.312458  0.189618  ...  0.155079  0.592691  0.543015
10000m_min    0.366702  0.307018  0.181817  ...  0.231701  0.165205 -0.796334
Marathon_min  0.341825  0.439947  0.260172  ... -0.329455 -0.393327  0.160236
```

Interpretation:

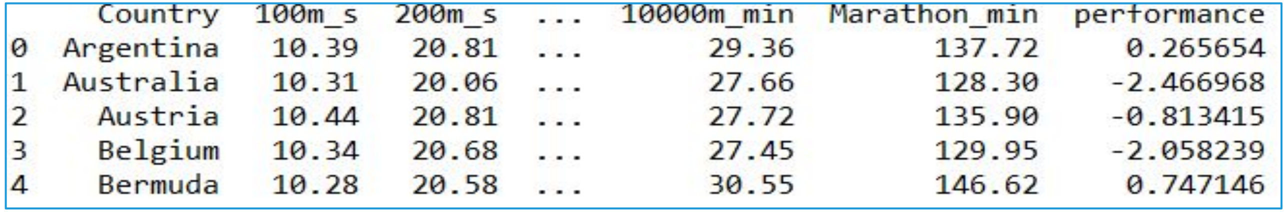
- First Principal Component can be interpreted as ‘general athletics skill’ since all variables have similar loadings.

PCA in Python - Loadings and Scores

```
#Scores Based on PCA
```

```
Score= PCA().fit_transform(X)
Score_df = pd.DataFrame(Score, index = athletics.index, columns = col)
athletics = athletics.assign(performance=Score_df.Comp1)
athletics.head()
```

```
# Output:
```



```
     Country  100m_s  200m_s  ...  10000m_min  Marathon_min  performance
0  Argentina   10.39   20.81  ...       29.36        137.72     0.265654
1  Australia   10.31   20.06  ...       27.66        128.30    -2.466968
2    Austria   10.44   20.81  ...       27.72        135.90    -0.813415
3    Belgium   10.34   20.68  ...       27.45        129.95    -2.058239
4    Bermuda   10.28   20.58  ...       30.55        146.62     0.747146
```

Interpretation:

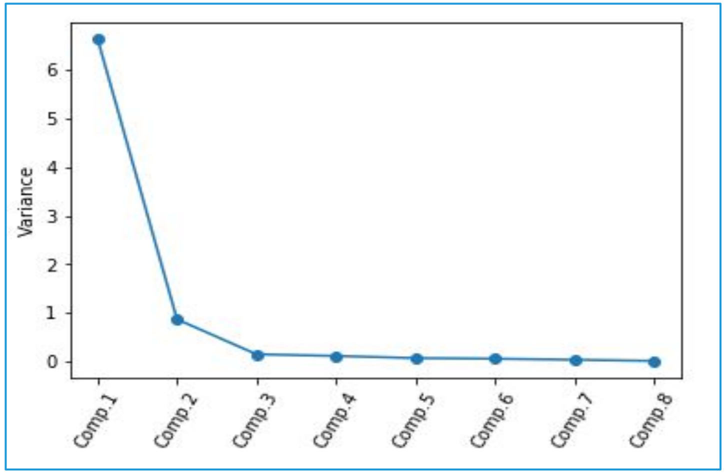
- New column ‘performance’ enumerates scores returned by the PCA summary.
- Lower score implies lesser time and hence better athletics performance.

PCA in Python – Scree Plot

```
#Scree Plot : plot the principal components vs. variances.
```

```
y = np.std(pca.transform(X), axis=0)**2
x = np.arange(len(y)) + 1
plt.plot(x, y, "o-")
plt.xticks(x, ["Comp."+str(i) for i in x], rotation=60)
plt.ylabel("Variance")
plt.show()
```

```
# Output:
```



Interpretation:

- First Principal Component is sufficient in explaining the maximum variation.

Exploratory Data Analysis Based on PCA

```
#Using Scores to Check Data Features
```

```
athletics.sort_values(by = 'performance').head(3)
athletics.sort_values(by = 'performance').tail(3)
```

- **sort_values()** sorts the data frame based on performance. Python takes **ascending = True** as default.
- **head()** and **tail()** extracts top n and bottom n countries in terms of performance measured by performance variable. Here n=3.

```
# Output :
```

```
                                Country  100m_s  ...  Marathon_min  performance
52                                  USA    9.93  ...        128.22    -3.460450
20  Great Britain and Northern Ireland   10.11  ...        129.13    -3.050287
28                                Italy   10.01  ...        131.08    -2.750446
```

```
           Country  100m_s  200m_s  ...  10000m_min  Marathon_min  performance
35       Mauritius   11.19   22.45  ...       31.77        152.23     4.299192
54  Western Samoa    10.82   21.86  ...       34.71        161.83     7.297965
11     Cook Isands   12.18   23.20  ...       35.38        164.70    10.653867
```

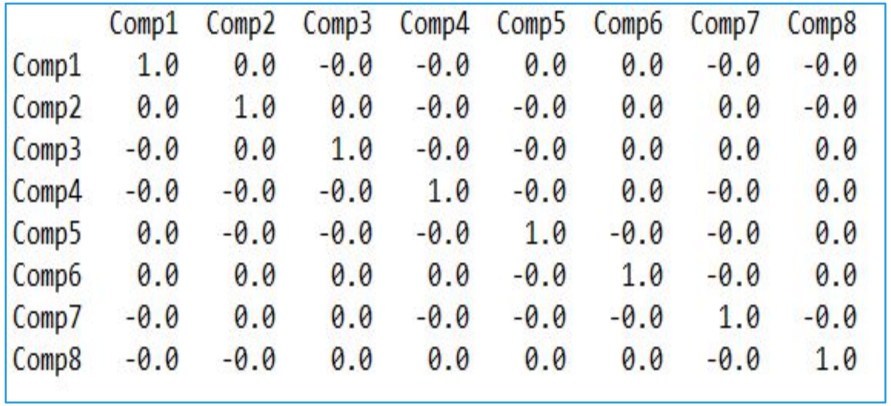
Interpretation:

- USA, Britain and Italy are the top three performing countries.
- Cook Islands, Western Samoa and Mauritius are the bottom three countries.

PCA in Python – Verification of PCA

```
#Verification that PC's are Uncorrelated
print(Score_df.corr().round())
```

- **corr()** and **round()** to calculate the rounded correlations for the principal components.



```
       Comp1  Comp2  Comp3  Comp4  Comp5  Comp6  Comp7  Comp8
Comp1    1.0    0.0   -0.0   -0.0    0.0    0.0   -0.0   -0.0
Comp2    0.0    1.0    0.0   -0.0   -0.0    0.0    0.0   -0.0
Comp3   -0.0    0.0    1.0   -0.0   -0.0    0.0    0.0    0.0
Comp4   -0.0   -0.0   -0.0    1.0   -0.0    0.0   -0.0    0.0
Comp5    0.0   -0.0   -0.0   -0.0    1.0   -0.0   -0.0    0.0
Comp6    0.0    0.0    0.0    0.0   -0.0    1.0   -0.0    0.0
Comp7   -0.0    0.0    0.0   -0.0   -0.0   -0.0    1.0   -0.0
Comp8   -0.0   -0.0    0.0    0.0    0.0    0.0   -0.0    1.0
```

Interpretation:

- The principal components are uncorrelated

Quick Recap

In this session we learnt about, Principal Component Regression :

Data Reduction and PCA	• Principal Component Analysis is a key data reduction method • Data Reduction is necessary while analyzing high dimensional data
PCA in Python	• **`PCA()`** function in Scikit Learn performs PCA • Scree Plot is used to decide number of components to be retained

Principal Component Regression (PCR)

Contents

Multiple Linear Regression: Statistical Model

$$\boxed{Y = b_0 + b_1X_1 + b_2X_2 + \ldots + b_pX_p + e}$$

Where,

Y	: Dependent Variable
$X_1, X_2, \ldots, X_p$	: Independent Variables
$b_0, b_1, \ldots, b_p$	: Parameters of Model
e	: Random Error Component

Independent variables can either be **Continuous or Categorical**

Problem of Multicolinearity

Multicolinearity exists ▶ if there is a **strong linear relationship among independent variables.**

Consequences

- Highly Unstable Model Parameters
  As standard errors of their estimates are inflated
- Model Fails to predict accurately out of sample Data

Multicolinearity is detected using Variance Inflation Factor, VIF

Tolerance = $1 - R_i^2$

VIF = 1/Tolerance

where R_i^2 (R Squared) is obtained using regression of Xi on other independent variables

Any VIF > 5, indicates presence of multicollinearity

Multicollinearity – Remedial Measures

The problem of Multicollinearity can be solved by different approaches:

- Drop one of the independent variables, which is explained by others
- Use Principal Component Regression in case of severe Multicollinearity
- Use Ridge Regression

Principal Component Regression

In Principal Component Regression,

First k principal components are used as independent variables instead of original X variables

- Each PC is a linear combination of all X variables
- Final model is expressed in terms of original independent variables for ease of interpretation

Principal Component Regression

Transformation into PCs

The original p variables are transformed into a new set of orthogonal or uncorrelated variables called “Principal Components “

Regression Analysis

In the second step, after elimination of the least important principal components, a multiple regression analysis of the response variable against the reduced set of principal components is performed using the OLS estimation

Back Transformation

In the third step, model equation is back transformed in terms of original variables.

PCR-Statistical Model

Model in terms of original X variables:

$$Y = b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_p x_p + e$$

Model in terms of Principal Components:

$$Y = a_0 + a_1 PC_1 + a_2 PC_2 + \cdots + a_k PC_k + e'$$

Case Study

Background

- A company periodically records data for sales and expenses. The company wishes to model the relationship between its sales and sales related expenses and obtain predictions

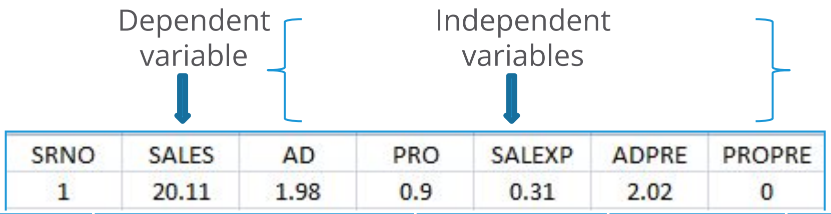
Objective

- To predict incremental sales based on planned sales related expenses

Available Information

- **Data available for 143 micro business zones**
- **Sales** is the Dependent Variable
- **Expenditure towards advertisements and promotions in the current and previous months** are Predictors

Data Snapshot


Dependent variable
Independent variables

SRNO	SALES	AD	PRO	SALEXP	ADPRE	PROPRE
1	20.11	1.98	0.9	0.31	2.02	0

Columns	Description	Type	Measurement	Possible values
SRNO	Serial Number	-	-	Intergers
SALES	Incremental Sales	Numerical	INR Million	positive value
AD	Current Advertising Expenses	Numerical	INR Million	positive value
PRO	Current Promotional Expenses	Numerical	INR Million	positive value
SALEXP	Misc. Sales Expenses	Numerical	INR Million	positive value
ADPRE	Previous Period's Advertising Expenses	Numerical	INR Million	positive values
PROPRE	Previous Period's Promotional Expenses	Numerical	INR Million	Positive value

PCR in Python

```
# Importing csv file “pcrdata”
```

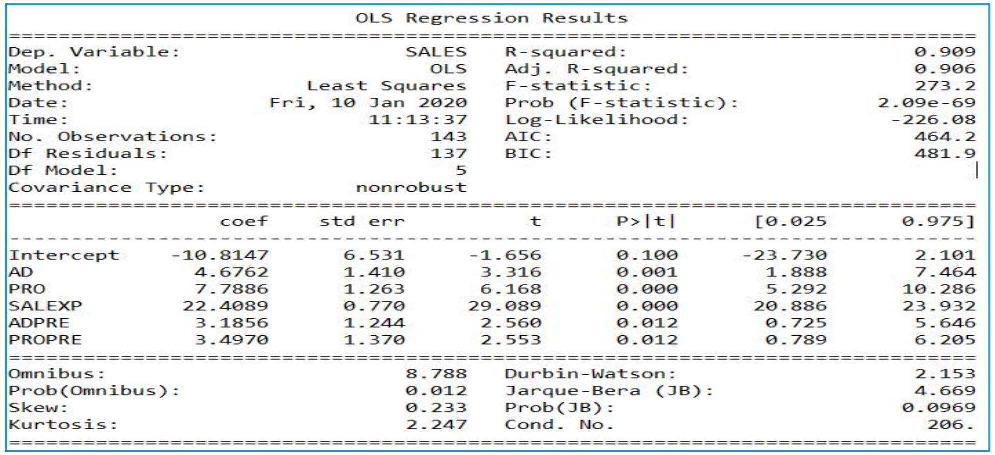
```python
import pandas as pd
salesdata = pd.read_csv('pcrdata.csv')
```

```
# Fit a Linear Model :
```

```python
import statsmodels.formula.api as smf
predsales=smf.ols('SALES~AD+PRO+SALEXP+ADPRE+PROPRE',
data=salesdata).fit()
predsales.summary()
```

- **smf.ols()** fits a linear regression model.
- **summary()** generates model summary.

```
# Output:
```



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  SALES   R-squared:                       0.909
Model:                            OLS   Adj. R-squared:                  0.906
Method:                 Least Squares   F-statistic:                     273.2
Date:                Fri, 10 Jan 2020   Prob (F-statistic):           2.09e-69
Time:                        11:13:37   Log-Likelihood:                -226.08
No. Observations:                 143   AIC:                             464.2
Df Residuals:                     137   BIC:                             481.9
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    -10.8147      6.531     -1.656      0.100     -23.730       2.101
AD             4.6762      1.410      3.316      0.001       1.888       7.464
PRO            7.7886      1.263      6.168      0.000       5.292      10.286
SALEXP        22.4089      0.770     29.089      0.000      20.886      23.932
ADPRE          3.1856      1.244      2.560      0.012       0.725       5.646
PROPRE         3.4970      1.370      2.553      0.012       0.789       6.205
==============================================================================
Omnibus:                        8.788   Durbin-Watson:                   2.153
Prob(Omnibus):                  0.012   Jarque-Bera (JB):                4.669
Skew:                           0.233   Prob(JB):                       0.0969
Kurtosis:                       2.247   Cond. No.                         206.
==============================================================================
```

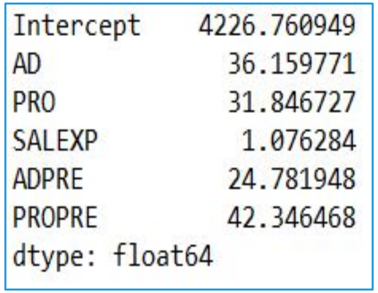
Interpretation :

- Multiple R-Squared is 0.909, showing model to be a good fit.

PCR in Python

```
# Checking for Multicollinearity
from patsy import dmatrices
from statsmodels.stats.outliers_influence import
variance_inflation_factor
y, X = dmatrices('SALES~AD+PRO+SALEXP+ADPRE+PROPRE', data=salesdata,
return_type="dataframe")
vif = pd.Series([variance_inflation_factor(X.values, i)for i in
range(X.shape[1])],index=X.columns)
vif
```

```
# Output of VIF
```



```
Intercept    4226.760949
AD             36.159771
PRO            31.846727
SALEXP          1.076284
ADPRE          24.781948
PROPRE         42.346468
dtype: float64
```

- **patsy** is a library that helps convert data frames into design matrices.
- **dmatrices** Construct two design matrices given a formula_like and data. By convention, the first matrix is the "y" data, and the second is the "x" data.
- **variance_inflation_factor** requires a design matrix as input to calculate vif.
- **variance_iinflation_factor()** calculates VIFs.

Interpretation:
VIF values are very high (>5, except for SALEXP) indicating severe multicollinearity problem.

PCR in Python

```
# PCA in Python
# Subsetting data for PCA and using PCA function
```

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale

salesdata2=salesdata.drop(['SRNO','SALES'], axis=1)
standardisedX = scale(salesdata2)
```

- **drop()** is used to remove the columns from the data.
- The data should be standardised using the function **scale()**

```
X = pd.DataFrame(standardisedX, index=salesdata2.index,
columns=salesdata2.columns)

pca = PCA().fit(X)
```

PCR in Python

```
# Summary of PCA
```

```
import numpy as np
names = ["PC"+str(i) for i in range(1,6)]

SD = list(np.std(pca.transform(X), axis=0))
VarProp = list(pca.explained_variance_ratio_)
CumProp = [np.sum(pca.explained_variance_ratio_[:i]) for i in range(1,6)]
```

- Define names of columns as PC1,PC2 and so on.

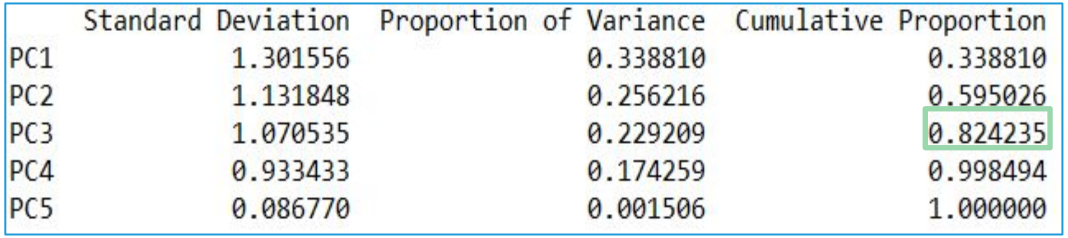
- Extract Standard Deviation and Proportion of variance explained. Define Cumulative proportion for the summary table
- pca.transform(X) computes scores

```
summary = pd.DataFrame(list(zip(SD, VarProp, CumProp)), index=names,
columns=['Standard Deviation','Proportion of Variance','Cumulative
Proportion'])
summary
```

- Creates a dataframe of summary output

PCR in Python

```
# Output
```



```
     Standard Deviation  Proportion of Variance  Cumulative Proportion
PC1            1.301556                0.338810               0.338810
PC2            1.131848                0.256216               0.595026
PC3            1.070535                0.229209               0.824235
PC4            0.933433                0.174259               0.998494
PC5            0.086770                0.001506               1.000000
```

Interpretation:
The first three principal components explain 82% of the variation in the data. Therefore, we will use 3 components in PCR

PCR in Python

```
# Import library hoggorm for PCR in Python
import hoggorm

pcmodel = hoggorm.pcr.nipalsPCR(standardisedX, y)
```

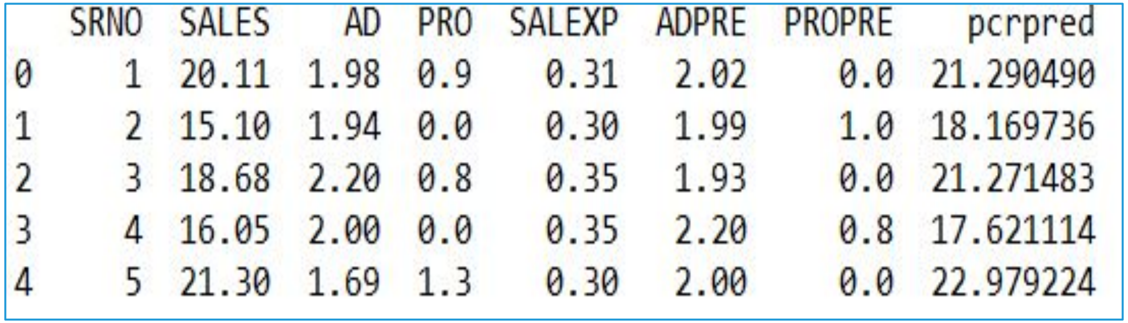
- ❑ **pcr.nipalsPCR()** in library **hoggorm** performs Principal Component
- ❑ standardisedX is the X in the PCR model and y is object of dependent variable
- ❑ **ncomp=3** is the number of components to be included in the model,

```
salesdata['pcrpred'] = pcmodel.Y_predCal()[3]

salesdata.head()
```

.Y_predCal() is used to get predictions using PCR.

PCR in Python

```
# Output
```



```
   SRNO  SALES    AD  PRO  SALEXP  ADPRE  PROPRE    pcrpred
0     1  20.11  1.98  0.9    0.31   2.02     0.0  21.290490
1     2  15.10  1.94  0.0    0.30   1.99     1.0  18.169736
2     3  18.68  2.20  0.8    0.35   1.93     0.0  21.271483
3     4  16.05  2.00  0.0    0.35   2.20     0.8  17.621114
4     5  21.30  1.69  1.3    0.30   2.00     0.0  22.979224
```

pcrpred column gives the predicted values of SALES using PCR.

Comparing Linear Regression Model and PCR model on Test data

```
# Importing Test Data
```

```
salesdata_test = pd.read_csv('pcrdata_test.csv')
```

```
# Getting RMSE of PCR model
```

```
salesdata_test2=salesdata_test.drop(['SRNO','SALES'], axis=1)
standardisedX2 = scale(salesdata_test2)
salesdata_test['pcrpredict'] =  pcmodel.Y_predict(standardisedX2,
numComp=3)
salesdata_test['pcrres'] = salesdata_test['SALES'] -
salesdata_test['pcrpredict’]

import math
import statistics
RMSE_pcr = math.sqrt(statistics.mean(salesdata_test['pcrres'])**2)
```

❑ RMSE_pcr stores RMSE value using PCR Model.

```
# Getting RMSE of linear regression model
```

```
salesdata_test['lmpred'] = predsales.predict(salesdata_test)
salesdata_test['lmres'] = salesdata_test['SALES'] -
salesdata_test['lmpred’]

RMSE_lm= math.sqrt(statistics.mean(salesdata_test['lmres'])**2)
```

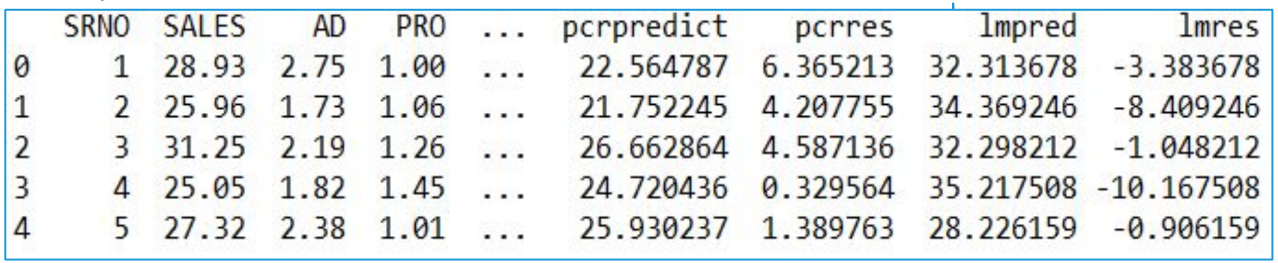
❑ RMSE_lm stores RMSE using normal regression

Comparing Linear Regression Model and PCR model on Test data

```
# Viewing data after adding predicted & residual variables
```

```
salesdata_test.head()
```

```
# Output
```



```
   SRNO  SALES    AD   PRO  ...  pcrpredict    pcrres     lmpred      lmres
0     1  28.93  2.75  1.00  ...   22.564787  6.365213  32.313678  -3.383678
1     2  25.96  1.73  1.06  ...   21.752245  4.207755  34.369246  -8.409246
2     3  31.25  2.19  1.26  ...   26.662864  4.587136  32.298212  -1.048212
3     4  25.05  1.82  1.45  ...   24.720436  0.329564  35.217508 -10.167508
4     5  27.32  2.38  1.01  ...   25.930237  1.389763  28.226159  -0.906159
```

```
RMSE_lm
[1] 7.949631207886278
RMSE_pcr
[1] 0.773461538461537
```

Interpretation:

- **RMSE using PCR is less than RMSE using linear regression**, we may conclude that PCR model predicts SALES better than linear regression model when multicollinearity exists.

Quick Recap

Multiple Linear Regression and Multicollinearity	• Fundamental assumption for building a good multiple linear regression model is to have non-correlated predictor variables. • However, highly correlated predictor variables is a very frequent phenomenon in real world analytics.
Principal Component Regression	• Such severe multicolinearity can be effectively handled by combining the regression with Principal Component Analysis. • PCR is a three way process where the variables are first transformed to principal components, regression is run by considering these components as regressors and finally, they are transformed back to their original forms.
PCR in Python	• **`pcr.nipalsPCR()`** function in library **`hoggorm`** performs PCR.

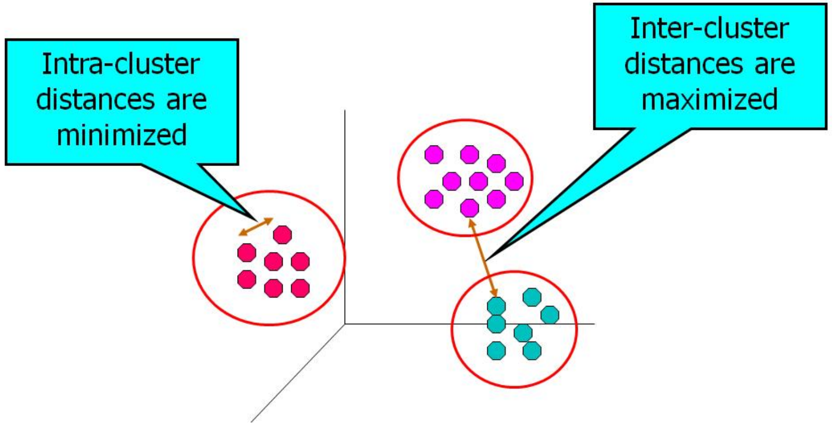
Non-Hierarchical Clustering K-Means Method

Contents

Cluster Analysis

Cluster analysis is a class of statistical techniques that can be used to classify objects or cases into groups called **Clusters.**

- A cluster is a group of relatively homogeneous cases or observations.
- The observations are dissimilar to objects outside the cluster, particularly objects in other clusters.


Intra-cluster distances are minimized
Inter-cluster distances are maximized

- Cluster Analysis is one of the unsupervised learning method.

K-Means Clustering

- K-Means Clustering is one of the most popular non-hierarchical clustering methods
- **K –Means method is suitable for large data sets** and widely used for customer segmentation in BFSI or retail domains
- **The number of clusters (k) must be known a priori**
  (Though in reality this may not be the case)
- **Alternatively, cluster solutions can be observed for different k and evaluated to get the best possible cluster solution**

K-Means Clustering – Steps

Define K

Define the Distance Measure

Select Initial Seeds

Execute Algorithm

Check the Output for

- Cluster Contribution
- R-squared (Overall and for all the variables)
- RMSSTD for each of the Cluster

Repeat the procedure for different K if criterion not satisfied

Case Study

Background

- A FMCG company has recorded information of customers based on their buying behaviour for a period of 1 year and would like to implement strategies by segmenting these customers into tiers.

Objective

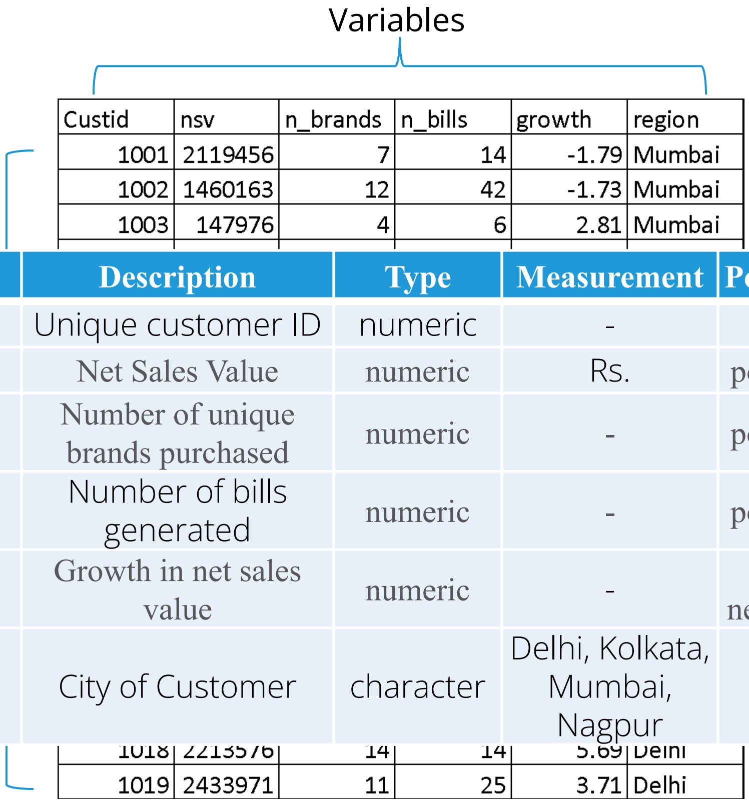
- To create segment of customers.

Available Information

- Sample size is 1158.
- Variables : Custid, nsv, n_brands , n_bills, growth, region

Data Snapshot

RETAILERS DATA



Custid	nsv	n_brands	n_bills	growth	region
1001	2119456	7	14	-1.79	Mumbai
1002	1460163	12	42	-1.73	Mumbai
1003	147976	4	6	2.81	Mumbai
1018	2213576	14	14	5.69	Delhi
1019	2433971	11	25	3.71	Delhi

Columns	Description	Type	Measurement	Possible values
Custid	Unique customer ID	numeric	-	-
nsv	Net Sales Value	numeric	Rs.	positive values
n_brands	Number of unique brands purchased	numeric	-	positive values
n_bills	Number of bills generated	numeric	-	positive values
growth	Growth in net sales value	numeric	-	Positive & negative values
region	City of Customer	character	Delhi, Kolkata, Mumbai, Nagpur	4

K-Means Method in Python

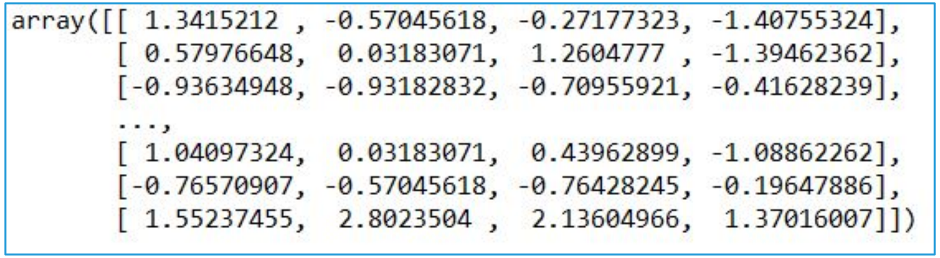
```
# Importing Data
import pandas as pd
custsales = pd.read_csv("RETAILERS DATA.csv")
custsales_cl = custsales
custsales_cl = custsales_cl.drop(["Custid", "region"], axis = 1)
```

- **read_csv()** is used to import csv file. Our data is saved as an object named custsales. The subset of numeric variables is created.

```
# Scale (standardize) all variables.(subtract mean and divide by
# standard deviation)
```

```
import sklearn.preprocessing
custsales_cl2 = sklearn.preprocessing.scale(custsales_cl)
custsales_cl2
```

```
# Output
```



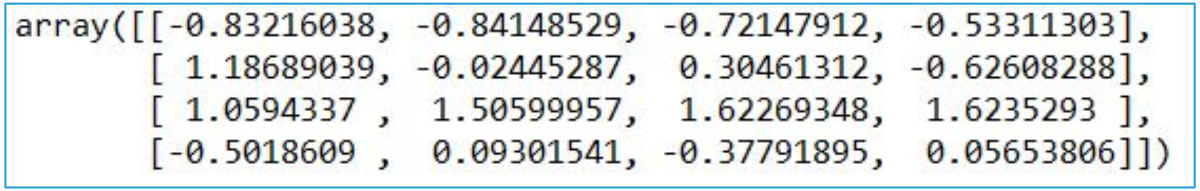
```
array([[ 1.3415212 , -0.57045618, -0.27177323, -1.40755324],
       [ 0.57976648,  0.03183071,  1.2604777 , -1.39462362],
       [-0.93634948, -0.93182832, -0.70955921, -0.41628239],
       ...,
       [ 1.04097324,  0.03183071,  0.43962899, -1.08862262],
       [-0.76570907, -0.57045618, -0.76428245, -0.19647886],
       [ 1.55237455,  2.8023504 ,  2.13604966,  1.37016007]])
```

K-Means Method in Python

```
# K means clustering
from sklearn.cluster import KMeans
CL = KMeans(n_clusters=4)
CL.fit(custsales_cl2)

# Compute centroids
centroids = CL.cluster_centers_
centroids
```

- **KMeans()** performs kmeans clustering on data matrix. The function requires data object and number of clusters to be formed.

```
# Output
```



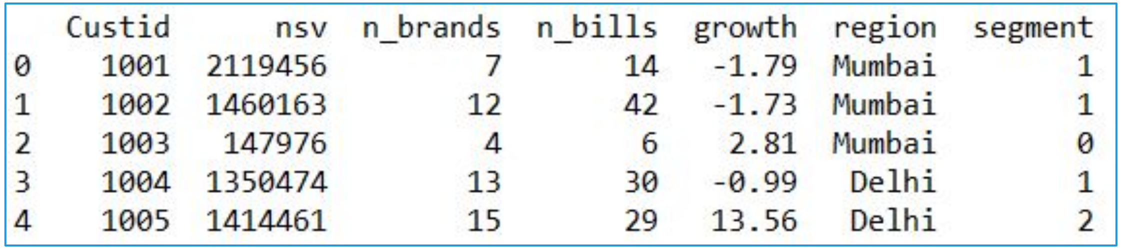
```
array([[-0.83216038, -0.84148529, -0.72147912, -0.53311303],
       [ 1.18689039, -0.02445287,  0.30461312, -0.62608288],
       [ 1.0594337 ,  1.50599957,  1.62269348,  1.6235293 ],
       [-0.5018609 ,  0.09301541, -0.37791895,  0.05653806]])
```

K-Means Method in Python

```
# Create Segments
```

```
segment = pd.DataFrame(CL.labels_)
custsales = custsales.assign(segment = segment)
custsales.head()
```

```
# Output
```



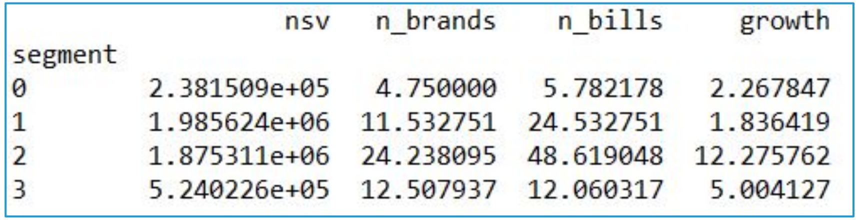
```
   Custid      nsv  n_brands  n_bills  growth  region  segment
0    1001  2119456         7       14   -1.79  Mumbai        1
1    1002  1460163        12       42   -1.73  Mumbai        1
2    1003   147976         4        6    2.81  Mumbai        0
3    1004  1350474        13       30   -0.99   Delhi        1
4    1005  1414461        15       29   13.56   Delhi        2
```

K-Means Method in Python : Summarize Clusters Using Original Variables

```
# Aggregating data based on segments
```

```
nsv = custsales.groupby('segment')['nsv'].mean()
n_brands = custsales.groupby('segment')['n_brands'].mean()
n_bills = custsales.groupby('segment')['n_bills'].mean()
growth = custsales.groupby('segment')['growth'].mean()
pd.concat([nsv,n_brands,n_bills,growth],axis=1)
```

```
# Output
```



```
                  nsv   n_brands    n_bills     growth
segment
0        2.381509e+05   4.750000   5.782178   2.267847
1        1.985624e+06  11.532751  24.532751   1.836419
2        1.875311e+06  24.238095  48.619048  12.275762
3        5.240226e+05  12.507937  12.060317   5.004127
```

Interpretation :

- Cluster 3 is group of ‘Platinum’ clusters.
- Cluster 1 is a group of ‘non-performers’

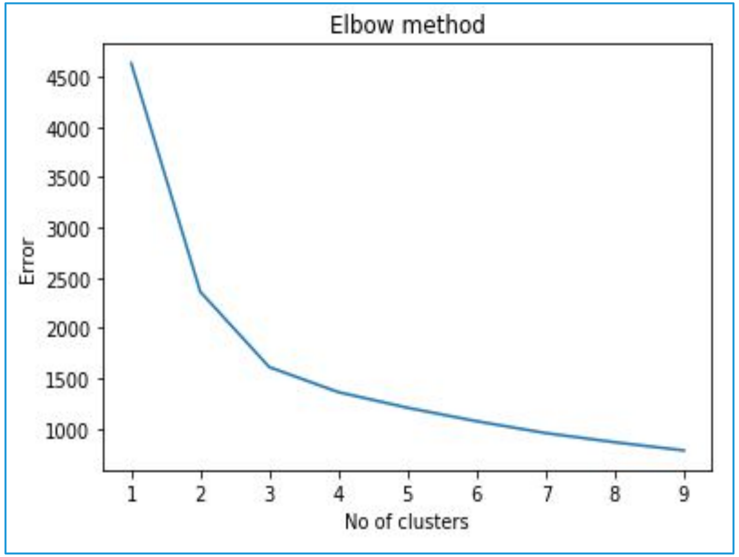
K-Means Method in Python
Elbow Method

```
# Optimum value of K by Elbow method
Error =[]
import matplotlib.pyplot as plt

for i in range(1, 10):
    CL = KMeans(n_clusters = i).fit(custsales_cl2)
    Error.append(CL.inertia_)

plt.plot(range(1, 10), Error)
plt.title('Elbow method')
plt.xlabel('No of clusters')
plt.ylabel('Error')
plt.show()
```

- We use Elbow method to find optimum value of K in which the sum of square distances from each point to its assigned center is calculated. (Within Sum of Squares)
- **CL.inertia_** gives Within Sum of Squares

K-Means Method in Python
Elbow Method

```
# Output
```



Interpretation :

- The location of a bend in the plot is generally considered as an indicator of the appropriate number of clusters.
- Here K= 3 or 4 is a good solution.
- The method is termed as Elbow Method.

Quick Recap

K-Means Clustering in R

- **`kMeans()`** function in Python performs K-Means Clustering
- The KMeans function requires data object and value of K

Elbow Method

- Elbow method is used to determine optimum value of K