Topics Covered in DataFarame PreProcessing with PySpark

- **1.Converting String in to Float**
- **2.Checking Missing Values**
- **3.Treating Missing Values**
- **4.Statistics**
- **5.checking correlation using pearson method**
- **6.VectorAssembler**
- **7.Standard Scaling**
- **8.PCA**

Import modules and Libraries

In [64]:

```
import pyspark
import numpy as np
import pandas as pd

from pyspark.sql import SparkSession

from pyspark.mllib.feature import StandardScaler,PCA
from pyspark.mllib.stat import Statistics

from pyspark.ml.feature import VectorAssembler
```

In [65]:

```
spark = SparkSession.builder.appName("DataFrame_Preprocessing").getOrCreate()
```

In [66]:

```
dataset = spark.read.csv("Admission_Prediction.csv",header=True,inferSchema=True)
```

In [69]:

```
dataset.show(5,truncate=False)
```

```
+---------+-----------+-----------------+---+----+----+--------+----------------+
|GRE Score|TOEFL Score|University Rating|SOP|LOR |CGPA|Research|Chance of Admit |
+---------+-----------+-----------------+---+----+----+--------+----------------+
|337      |118        |4                |4.5|4.5 |9.65|1       |0.92            |
|324      |107        |4                |4.0|4.5 |8.87|1       |0.76            |
|316      |104        |3                |3.0|3.5 |8.0 |1       |0.72            |
|322      |110        |3                |3.5|2.5 |8.67|1       |0.8             |
|null     |103        |2                |2.0|3.0 |8.21|0       |0.65            |
+---------+-----------+-----------------+---+----+----+--------+----------------+
only showing top 5 rows
```

In [70]:

```
dataset.printSchema()
```

```
root
 |-- GRE Score: string (nullable = true)
 |-- TOEFL Score: string (nullable = true)
 |-- University Rating: string (nullable = true)
 |-- SOP: double (nullable = true)
 |-- LOR : double (nullable = true)
 |-- CGPA: double (nullable = true)
 |-- Research: integer (nullable = true)
```

```
 |-- Chance of Admit : double (nullable = true)
```

Converting String in to Float

In [71]:

```
from pyspark.sql.functions import col
new_data = dataset.select(*(col(c).cast("float") for c in dataset.columns))
```

In [72]:

```
new_data.show(5,truncate=False)
```

```
+---------+-----------+-----------------+---+----+----+--------+----------------+
|GRE Score|TOEFL Score|University Rating|SOP|LOR |CGPA|Research|Chance of Admit |
+---------+-----------+-----------------+---+----+----+--------+----------------+
|337.0    |118.0      |4.0              |4.5|4.5 |9.65|1.0     |0.92            |
|324.0    |107.0      |4.0              |4.0|4.5 |8.87|1.0     |0.76            |
|316.0    |104.0      |3.0              |3.0|3.5 |8.0 |1.0     |0.72            |
|322.0    |110.0      |3.0              |3.5|2.5 |8.67|1.0     |0.8             |
|null     |103.0      |2.0              |2.0|3.0 |8.21|0.0     |0.65            |
+---------+-----------+-----------------+---+----+----+--------+----------------+
only showing top 5 rows
```

In [73]:

```
new_data.printSchema()
```

```
root
 |-- GRE Score: float (nullable = true)
 |-- TOEFL Score: float (nullable = true)
 |-- University Rating: float (nullable = true)
 |-- SOP: float (nullable = true)
 |-- LOR : float (nullable = true)
 |-- CGPA: float (nullable = true)
 |-- Research: float (nullable = true)
 |-- Chance of Admit : float (nullable = true)
```

Checking Missing Values

In [74]:

```
#### when we drop the columns is where null values are there inside the dataframe
# data_without_missing = dataset.dropna(how='any')
# data_without_missing = dataset.dropna(how='all')
```

In [75]:

```
from pyspark.sql.functions import col, count, isnan, when
#checking for null ir nan type values in our columns
new_data.select([count(when(col(c).isNull(), c)).alias(c) for c in new_data.columns]).sh
ow()
```

```
+---------+-----------+-----------------+---+----+----+--------+----------------+
|GRE Score|TOEFL Score|University Rating|SOP|LOR |CGPA|Research|Chance of Admit |
+---------+-----------+-----------------+---+----+----+--------+----------------+
|        5|          3|                1|  0|   0|   0|       0|               0|
+---------+-----------+-----------------+---+----+----+--------+----------------+
```

Treating Missing Values with Imputer

In [76]:

```python
from pyspark.ml.feature import Imputer
imputer = Imputer(inputCols=["GRE Score", "TOEFL Score","University Rating"],
                  outputCols=["GRE Score", "TOEFL Score","University Rating"])
model = imputer.fit(new_data)

imputed_data = model.transform(new_data)
```

In [77]:

```python
imputed_data.show(5,truncate=False)
```

```
+---------+-----------+-----------------+---+----+----+--------+----------------+
|GRE Score|TOEFL Score|University Rating|SOP|LOR |CGPA|Research|Chance of Admit |
+---------+-----------+-----------------+---+----+----+--------+----------------+
|337.0    |118.0      |4.0              |4.5|4.5 |9.65|1.0     |0.92            |
|324.0    |107.0      |4.0              |4.0|4.5 |8.87|1.0     |0.76            |
|316.0    |104.0      |3.0              |3.0|3.5 |8.0 |1.0     |0.72            |
|322.0    |110.0      |3.0              |3.5|2.5 |8.67|1.0     |0.8             |
|316.4909 |103.0      |2.0              |2.0|3.0 |8.21|0.0     |0.65            |
+---------+-----------+-----------------+---+----+----+--------+----------------+
only showing top 5 rows
```

In [78]:

```python
from pyspark.sql.functions import col, count, isnan, when
#checking for null or nan type values in our columns
imputed_data.select([count(when(col(c).isNull(), c)).alias(c) for c in imputed_data.colu
mns]).show()
```

```
+---------+-----------+-----------------+---+----+----+--------+----------------+
|GRE Score|TOEFL Score|University Rating|SOP|LOR |CGPA|Research|Chance of Admit |
+---------+-----------+-----------------+---+----+----+--------+----------------+
|        0|          0|                0|  0|   0|   0|       0|               0|
+---------+-----------+-----------------+---+----+----+--------+----------------+
```

In [79]:

```python
imputed_data.count()
```

Out[79]:

```
500
```

In [80]:

```python
imputed_data.corr('SOP','Research')
```

Out[80]:

```
0.4064577967296779
```

In [81]:

```python
features = imputed_data.drop('Chance of Admit')
```

we need to convert dataframe into a RDD to check for correlation

In [82]:

```python
col_names = features.columns
features_rdd = features.rdd
```

In [93]:

```python
## show the top features in rdd
features_rdd.top(5)
```

Out[93]:

```
[(340.0, 120.0, 5.0, 4.5, 4.5, 9.90999984741211, 1.0, 0.9700000286102295),
 (340.0, 120.0, 5.0, 4.5, 4.5, 9.600000381469727, 1.0, 0.9399999976158142),
 (340.0, 120.0, 4.0, 5.0, 5.0, 9.5, 1.0, 0.9599999785423279),
 (340.0, 120.0, 4.0, 4.5, 4.0, 9.920000076293945, 1.0, 0.9700000286102295),
 (340.0, 115.0, 5.0, 5.0, 4.5, 9.0600004196167, 1.0, 0.949999988079071)]
```

In [94]:

```
features_rdd = features.rdd.map(lambda row: row[0:])
```

In [95]:

```
# features_rdd.collect()
features_rdd.top(5)
```

Out[95]:

```
[(340.0, 120.0, 5.0, 4.5, 4.5, 9.90999984741211, 1.0, 0.9700000286102295),
 (340.0, 120.0, 5.0, 4.5, 4.5, 9.600000381469727, 1.0, 0.9399999976158142),
 (340.0, 120.0, 4.0, 5.0, 5.0, 9.5, 1.0, 0.9599999785423279),
 (340.0, 120.0, 4.0, 4.5, 4.0, 9.920000076293945, 1.0, 0.9700000286102295),
 (340.0, 115.0, 5.0, 5.0, 4.5, 9.0600004196167, 1.0, 0.949999988079071)]
```

Statistics

In [96]:

```
summary = Statistics.colStats(features_rdd)
print(summary.mean())  # a dense vector containing the mean value for each column
print(summary.variance())  # column-wise variance
print(summary.numNonzeros())  # number of nonzeros in each column
print(summary.normL1())# return a column of normL1 summary
```

```
[316.49090906 107.20724347   3.11222445   3.373        3.482
   8.5775       0.558        0.72174   ]
[1.26143706e+02 3.68289658e+01 1.30604295e+00 9.82335671e-01
 8.52380762e-01 3.64575080e-01 2.47130261e-01 1.99206139e-02]
[500. 500. 500. 500. 500. 500. 279. 500.]
[158245.45452881  53603.62173462   1556.11222434   1686.5
   1741.           4288.75000238    279.            360.86999956]
```

checking correlation using pearson method

In [97]:

```
corr_mat=Statistics.corr(features_rdd, method="pearson")
corr_df = pd.DataFrame(corr_mat)
corr_df.index, corr_df.columns = col_names, col_names
```

In [98]:

```
corr_df.columns
```

Out[98]:

```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

In [99]:

```
corr_df.index
```

Out[99]:

```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research', 'Chance of Admit '],
      dtype='object')
```

In [100]:

```
corr_df
```

Out[100]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
GRE Score	1.000000	0.819736	0.630379	0.610009	0.515327	0.822921	0.562442	0.807365
TOEFL Score	0.819736	1.000000	0.646440	0.641482	0.535526	0.808236	0.463112	0.786247
University Rating	0.630379	0.646440	1.000000	0.726602	0.607916	0.703442	0.425723	0.689575
SOP	0.610009	0.641482	0.726602	1.000000	0.665460	0.712155	0.406458	0.684564
LOR	0.515327	0.535526	0.607916	0.665460	1.000000	0.636094	0.369054	0.645548
CGPA	0.822921	0.808236	0.703442	0.712155	0.636094	1.000000	0.497910	0.880543
Research	0.562442	0.463112	0.425723	0.406458	0.369054	0.497910	1.000000	0.543089
Chance of Admit	0.807365	0.786247	0.689575	0.684564	0.645548	0.880543	0.543089	1.000000

In [ ]:

Vector Assembler

In [101]:

```
imputed_data.show(5)
```

```
+---------+-----------+-----------------+---+----+----+--------+----------------+
|GRE Score|TOEFL Score|University Rating|SOP|LOR |CGPA|Research|Chance of Admit |
+---------+-----------+-----------------+---+----+----+--------+----------------+
|    337.0|      118.0|              4.0|4.5| 4.5|9.65|     1.0|            0.92|
|    324.0|      107.0|              4.0|4.0| 4.5|8.87|     1.0|            0.76|
|    316.0|      104.0|              3.0|3.0| 3.5| 8.0|     1.0|            0.72|
|    322.0|      110.0|              3.0|3.5| 2.5|8.67|     1.0|             0.8|
| 316.4909|      103.0|              2.0|2.0| 3.0|8.21|     0.0|            0.65|
+---------+-----------+-----------------+---+----+----+--------+----------------+
only showing top 5 rows
```

In [102]:

```
features = imputed_data.drop('Chance of Admit ')
```

In [103]:

```
assembler = VectorAssembler(inputCols=features.columns,outputCol="features")
```

In [104]:

```
output = assembler.transform(imputed_data)
```

In [105]:

```
output.select("features", "Chance of Admit ").show(5,truncate=False)
```

```
+------------------------------------------------------------+----------------+
|features                                                    |Chance of Admit |
+------------------------------------------------------------+----------------+
|[337.0,118.0,4.0,4.5,4.5,9.649999618530273,1.0]             |0.92            |
|[324.0,107.0,4.0,4.0,4.5,8.869999885559082,1.0]             |0.76            |
|[316.0,104.0,3.0,3.0,3.5,8.0,1.0]                           |0.72            |
|[322.0,110.0,3.0,3.5,2.5,8.670000076293945,1.0]             |0.8             |
|[316.49090576171875,103.0,2.0,2.0,3.0,8.210000038146973,0.0]|0.65            |
+------------------------------------------------------------+----------------+
only showing top 5 rows
```

Standard Scaling

In [106]:

```
label = imputed_data.select('Chance of Admit ')
```

In [107]:

```
label.show(5)
```

```
+-----------------+
|Chance of Admit |
+-----------------+
|             0.92|
|             0.76|
|             0.72|
|              0.8|
|             0.65|
+-----------------+
only showing top 5 rows
```

In [108]:

```
features = imputed_data.drop('Chance of Admit ')
```

In [109]:

```
col_names = features.columns
features_rdd = features.rdd.map(lambda row: row[0:])
```

In [110]:

```
# features_rdd.collect()
features_rdd.top(5)
```

Out[110]:

```
[(340.0, 120.0, 5.0, 4.5, 4.5, 9.90999984741211, 1.0),
 (340.0, 120.0, 5.0, 4.5, 4.5, 9.600000381469727, 1.0),
 (340.0, 120.0, 4.0, 5.0, 5.0, 9.5, 1.0),
 (340.0, 120.0, 4.0, 4.5, 4.0, 9.920000076293945, 1.0),
 (340.0, 115.0, 5.0, 5.0, 4.5, 9.0600004196167, 1.0)]
```

In [111]:

```
scaler1 = StandardScaler().fit(features_rdd)
```

In [112]:

```
scaled_features=scaler1.transform(features_rdd)
```

In [122]:

```
for data in scaled_features.take(5):
    print(data)
```

```
[30.00524025385319,19.444073123688433,3.5001065242645106,4.540279160253269,4.874114130761
268,15.982098619569024,2.011578737704796]
[28.847768077888528,17.63149003588697,3.5001065242645106,4.035803698002906,4.874114130761
268,14.690281713001964,2.011578737704796]
[28.13547750806412,17.1371491937593,2.625079893198383,3.0268527735021795,3.79097765725876
37,13.249408705782436,2.011578737704796]
[28.669695435432427,18.125830878014643,2.625079893198383,3.5313282357525426,2.70784118375
626,14.359046811247923,2.011578737704796]
[28.179185951157212,16.972368913050072,1.7500532621322553,2.017901849001453,3.24940942050
7512,13.59720574748733,0.0]
```

PCA

In [123]:

```
pca = PCA(k=3)
model = pca.fit(scaled_features)
```

In [124]:

```
result = model.transform(scaled_features)
```

In [127]:

```
result.take(5)
```

Out[127]:

```
[DenseVector([-31.9772, 8.0295, -18.9905]),
 DenseVector([-30.0457, 7.6391, -17.5162]),
 DenseVector([-27.8526, 8.392, -17.4271]),
 DenseVector([-28.7505, 8.9783, -18.7505]),
 DenseVector([-26.4578, 7.7233, -19.1492])]
```

In [128]:

```
type(result)
```

Out[128]:

```
pyspark.rdd.RDD
```

In [61]:

```
#store dense vector in a dataframe
```

In [129]:

```
df =result.map(lambda x: (x, )).toDF(["PCA_Features"])
```

In [130]:

```
df.show(truncate=False)
```

```
+-------------------------------------------------------------+
|PCA_Features                                                 |
+-------------------------------------------------------------+
|[-31.977182487884036,8.029514763493243,-18.990453304678162]|
|[-30.045686880290237,7.639077844922069,-17.51623616623448] |
|[-27.852638239765152,8.392009538098923,-17.427074491217123]|
|[-28.75051513242634,8.978264691544112,-18.750488476990267] |
|[-26.457821418000282,7.723327004504269,-19.149180690578625]|
|[-31.083831050782454,8.278073275989826,-19.080459919848717]|
|[-28.68952685554881,8.384321835447519,-17.797432187149532] |
|[-26.569596508855597,6.63946310855808,-17.944097192101765] |
|[-24.82613160888541,8.256382319020885,-19.349665442263994] |
|[-28.21294041814849,7.258105092116565,-19.562480895131383] |
|[-28.96909729084087,8.260368281698211,-17.693333297572856] |
|[-30.50808885154048,7.796030288412726,-17.966541505750584] |
|[-30.679796347128107,7.843434871719907,-18.113950167927552]|
|[-28.061111225041095,8.148375853385518,-17.60159778694419] |
|[-27.443911746514434,8.806512394929458,-18.006387726361968]|
|[-27.29480963476284 6,7.199613494166247,-19.177076375587625]|
|[-28.1961363700801 3,6.914939863847723,-19.253041669191095] |
|[-28.2933443706931,8.396699131556772,-17.75467621750696]   |
|[-28.499864856244326,7.000838760264483,-19.552122727995897]|
|[-27.029928112884416,6.629862208629025,-18.433295117669324]|
+-------------------------------------------------------------+
only showing top 20 rows
```

END