Big Data Analytics Homework 04

Complete this assignment in Google Colab. Prior to submitting a copy of this notebook (.ipynb format), run every cell and ensure you have corrected all runtime errors. Be sure to fill in your Name and SUID in the following cell. As always, you must do your own work. This means you may not use answers to the following questions generated by any other person or a generative AI tool such as ChatGPT. You may, however, discuss this assignment with others in a general way and seek help when you need it, but, again, you must do your own work.

Name:

SUID:

Setup

```
! pip install pyspark -q
```

```
from pyspark.sql import SparkSession
from pyspark import SparkContext

sc = SparkContext.getOrCreate()

spark = SparkSession\
    .builder\
    .appName('Homework 04')\
    .getOrCreate()
```

This assignment uses a data set containing information about data science programs at universities worldwide.

The dataset contains many columns that we can use to understand how these data science programs differ from one another.

```
# download the data scince programs data set
%%bash
if [[ ! -f colleges-data-science-programs.csv ]]; then
 wget https://syr-bda.s3.us-east-2.amazonaws.com/colleges-data-science-programs.csv -q
fi
```

Q1

Read `colleges-data-science-programs.csv` into a Spark data frame named `raw_ds_programs_text`.

```
# your code here
raw_ds_programs_text = spark.read.csv("colleges-data-science-programs.csv", header=True, inferSchema=True)

```

```
# do not modify
print('rows: ', raw_ds_programs_text.count(),
      ', cols:', len(raw_ds_programs_text.columns))

raw_ds_programs_text\
  .show(5)
```

```
rows:  222 , cols: 28
+---+--------------------+--------------------+--------------------+-------+-------+-----+------+--------+--------------------+---------
| id|                name|                 url|             program| degree|country|state|online|oncampus|          department|
+---+--------------------+--------------------+--------------------+-------+-------+-----+------+--------+--------------------+---------
|  1|South Dakota Stat...|http://www.sdstat...|        Data Science|Masters|     US|   SD| false|    true|Mathematics and S...|2015-01-1
|  2|Dakota State Univ...|http://www.dsu.ed...|           Analytics|Masters|     US|   SD|  true|    true|Business and Info...|2015-01-1
|  3|    Lewis University|http://www.lewisu...|        Data Science|Masters|     US|   IL|  true|    true|    Computer Science|2015-01-1
|  4|Saint Joseph's Un...|http://online.sju...|Business Intellig...|Masters|     US|   PA|  true|    true|            Business|2015-01-1
|  5| University Of Leeds|http://www.engine...|Advanced Computer...|Masters|     GB| NULL| false|    true|    Computer Science|2015-01-1
+---+--------------------+--------------------+--------------------+-------+-------+-----+------+--------+--------------------+---------
only showing top 5 rows
```

Q2

Starting with `raw_ds_programs_text`, create a new data frame named `ds_programs_text` which simply adds a column named `text` to the original data frame.

The `text` column will be a concatenation of the following columns, separated by a space: `program`, `degree`, and `department`. You eill find the appropriate function in `pyspark.sql.functions`

An example of the `ds_programs_text_df` should give you:

```
ds_programs_text.orderBy('id').first().text
```

```
'Data Science Masters Mathematics and Statistics'
```

```
# your code here
from pyspark.sql.functions import concat_ws, col
ds_programs_text = raw_ds_programs_text.withColumn(
    "text",
    concat_ws(" ",
              col("program"),
              col("degree"),
              col("department")
             )
)
```

```
# do not modify
ds_programs_text.select('text')\
  .show(5, truncate = False)
```

```
+----------------------------------------------------------------+
|text                                                            |
+----------------------------------------------------------------+
|Data Science Masters Mathematics and Statistics                 |
|Analytics Masters Business and Information Systems              |
|Data Science Masters Computer Science                           |
|Business Intelligence & Analytics Masters Business              |
|Advanced Computer Science(Data Analytics) Masters Computer Science|
+----------------------------------------------------------------+
only showing top 5 rows
```

Q3

Create a pipeline named `pipe_features` that creates a new dataframe `ds_features`. The `pipe_features` pipeline should add a column, `features` to `ds_programs_text` that contains the `tfidf` of the `text` column.

Make sure to create your pipeline using methodology similar to what was demonstrated in class. Aside from removing stop words and setting a minumum token length of 2, no further restrictions should be imposed on the resulting vocabulary.

```
# your code here
from pyspark.ml import Pipeline
from pyspark.ml.feature import Tokenizer, StopWordsRemover, HashingTF, IDF

tokenizer = Tokenizer(inputCol="text", outputCol="words")
stop_words_remover = StopWordsRemover(inputCol="words", outputCol="filtered_words")
hashing_tf = HashingTF(inputCol="filtered_words", outputCol="raw_features", numFeatures=20)
idf = IDF(inputCol="raw_features", outputCol="features")

pipe_features = Pipeline(stages=[tokenizer, stop_words_remover, hashing_tf, idf])
ds_features = pipe_features.fit(ds_programs_text).transform(ds_programs_text)

ds_features_fitted = pipe_features.fit(ds_programs_text)
ds_features= ds_features_fitted.transform(ds_programs_text)
```

```
# do not modify
ds_features.select('features')\
  .show(5,
        truncate = False)
```

```
+------------------------------------------------------------------------------------------------------------------------------------------+
|features                                                                                                                                  |
+------------------------------------------------------------------------------------------------------------------------------------------+
```

```
|(20,[3,6,13,15,16],[1.1305056524440635,0.2481164722455897,2.76811444184486,0.5868902058550819,2.188295946591918])                |
|(20,[6,8,9,10,14],[0.2481164722455897,0.7158238892309751,2.188295946591918,1.5785303749710238,0.853294879859578])                |
|(20,[3,6,11,15],[2.261011304888127,0.2481164722455897,1.6229821375418576,0.5868902058550819])                                      |
|(20,[6,8,12,14],[0.2481164722455897,1.4316477784619501,1.5359707605522277,1.706589759719156])                                      |
|(20,[3,6,11,14,18,19],[1.1305056524440635,0.2481164722455897,3.245964275083715,0.853294879859578,4.020877410340228,2.699121570357909])|
+----------------------------------------------------------------------------------------------------------------------------------+
only showing top 5 rows
```

Q4

Create a pipeline model called `pipe_pca` that computes the first two principle components of the `features` column created by `pipe_features`, and creates a new column named `scores`.

Use `pipe_pca` to create a data frame, `ds_features_1` with the columns `id`, `name`, `url`, and `scores`.

Note: Prior to computing PCA scores, you will want to scale the TF-IDF outputs. Refer to lecture notes regarding the appropriate parameters to use during this step.

```
# your code here
from pyspark.ml import Pipeline
from pyspark.ml.feature import StandardScaler, PCA

scaler = StandardScaler(inputCol="features", outputCol="scaled_features", withMean=True, withStd=True)
pca = PCA(k=2, inputCol="scaled_features", outputCol="scores")

pipe_pca = Pipeline(stages=[scaler, pca])
pipe_pca_model = pipe_pca.fit(ds_features)
ds_features_1 = pipe_pca_model.transform(ds_features).select("id", "name", "url", "scores")
```

```
# do not modify
ds_features_1\
  .select('scores')\
  .show(5,
        truncate = False)
```

```
+-----------------------------------------+
|scores                                   |
+-----------------------------------------+
|[-0.8273713018426965,-0.09636389661089356]|
|[1.541109169337008,1.0772516193312067]   |
|[-1.6352404416200834,-1.316722397188407] |
|[2.3459831485125564,-0.7393473954578522] |
|[-1.4596210033256862,1.4993072818402193] |
+-----------------------------------------+
only showing top 5 rows
```

Q5

In this question you will write code that makes recommendations for programs closest to a program of interest.

Create a function named `get_nearest_programs` that returns the 3 closest programs to a program of interest.

The `get_nearest_programs` function should take 1 argument: `program_of_interest`. Write the function so that it returns the 3 programs (as defined by the `name` column) closest to the program argument as defined by Euclidian (L2) distance. Do not return the program of interest as one of the names.

Your function should **not** consider **Bachelors** programs.

```
# your code here
from pyspark.sql.functions import *
from pyspark.ml.linalg import Vectors
from pyspark.sql.functions import col
from pyspark.sql import DataFrame


def euclidean_distance(v1, v2):
    return float(Vectors.squared_distance(v1, v2))**0.5

def get_nearest_programs(name: str, df=ds_features_1) -> list:
    scores = df.filter(col('name') == name).select('scores').first()['scores']
```

```
    return (df
            .filter((col('degree') != 'Bachelors') & (col('name') != name))
            .select('name', 'scores')
            .rdd.map(lambda r: (r['name'], euclidean_distance(r['scores'], scores)))
            .toDF(['name', 'distance'])
            .orderBy('distance')
            .limit(3)
            .select('name')
            .toPandas()['name'].tolist())
```

```
# do not modify
get_nearest_programs('Syracuse University')
```

```
['Columbia University',
 'New Jersey Institute of Technology',
 'Coventry University']
```

Q6

Create two Pandas dataframes `pc1` and `pc2` with the columns `word` and `absolute_loading` that contain the top 5 absolute values (descending order) of loadings.

```
# your code here
import pandas as pd
import numpy as np

def extract_principal_components(pipe_pca_model):
    pca_model = pipe_pca_model.stages[-1]
    pc_loadings = pca_model.pc.toArray()

    result_df = pd.DataFrame({
        'feature': range(len(pc_loadings)),
        'load_pc1': pc_loadings[:, 0],
        'load_pc2': pc_loadings[:, 1]
    })
    result_df['abs_pc1'] = np.abs(result_df['load_pc1'])
    result_df['abs_pc2'] = np.abs(result_df['load_pc2'])
    return result_df

def get_top_components(df, n=5):
    pc1 = (df.sort_values('abs_pc1', ascending=False)
          .head(n)[['feature', 'abs_pc1']]
          .rename(columns={'abs_pc1': 'loading'}))
    pc2 = (df.sort_values('abs_pc2', ascending=False)
          .head(n)[['feature', 'abs_pc2']]
          .rename(columns={'abs_pc2': 'loading'}))
    return pc1, pc2

full_df = extract_principal_components(pipe_pca_model)
top_pc1, top_pc2 = get_top_components(full_df)

```

```
# do not modify
display(pc1.head())
display(pc2.head())
```

	word	absolute_loading
0	test	0.408248
1	another	0.408248
2	important.	0.408248
5	words	0.408248
3	sentence	0.408248

	word	absolute_loading
6	sentence.	0.632456
1	another	0.316228
0	test	0.316228
2	important.	0.316228
3	sentence	0.316228

Q7

Create a new pipeline called `pipe_pca_1` where you fit the maximum possible number of principal components for this dataset.

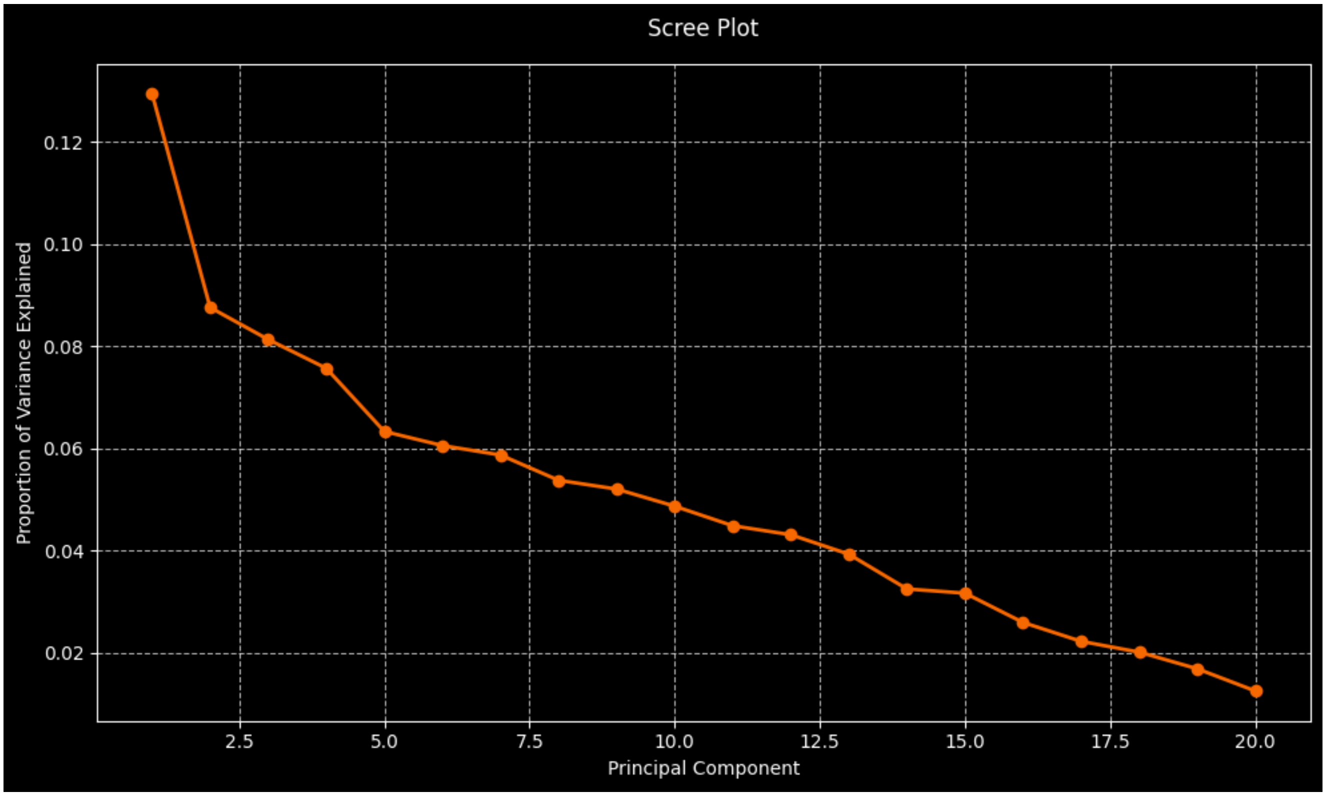
Create a scree plot and a plot of cumulative variance explained (exactly 2 plots).

Answer the following:

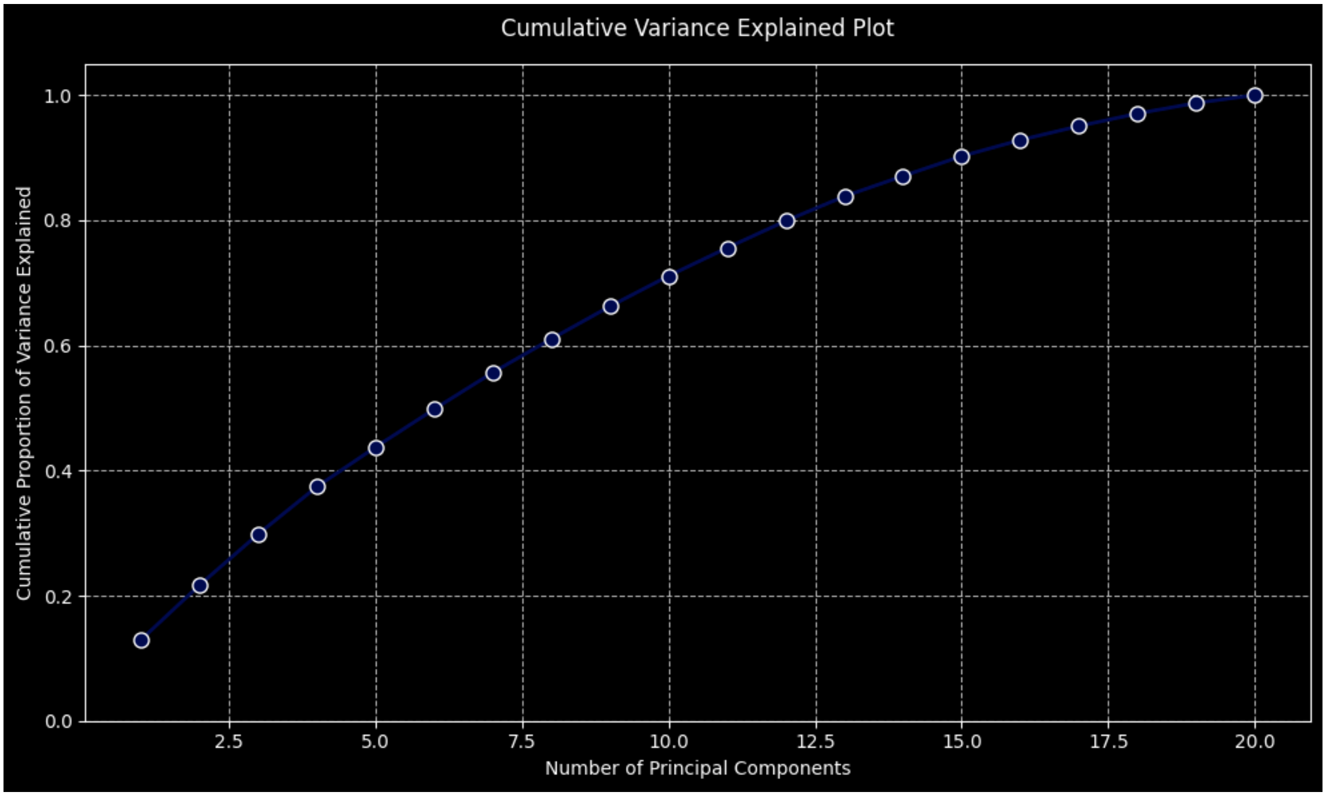
1. How many principal components were able to create (the maximum number)?
2. Based on either the scree or cumulative variance explained plot, how many principal components would you use if you were building a supervised machine learning model, and why?

```
# your code for new pipeline here
import matplotlib.pyplot as plt
import numpy as np
from pyspark.ml.feature import PCA

# Create a new pipeline model with the maximum number of principal components
max_pca = PCA(k=20, inputCol='scaled_features', outputCol='scores')
pipe_pca_1 = Pipeline(stages=[scaler, max_pca])
pipe_pca_model_1 = pipe_pca_1.fit(ds_features)
ds_features_max_pca = pipe_pca_model_1.transform(ds_features)
```

```
# your code for scree plot here

import matplotlib.pyplot as plt
import numpy as np
plt.style.use('dark_background')
SYRACUSE_ORANGE = '#F76900'
pca_model = pipe_pca_model_1.stages[-1]
n_components = len(pca_model.explainedVariance)

plt.figure(figsize=(10, 6))
plt.plot(
    np.arange(1, n_components + 1),
    pca_model.explainedVariance,
    marker='o',
    color=SYRACUSE_ORANGE,
    linewidth=2
)

plt.title('Scree Plot', pad=15)
plt.xlabel('Principal Component')
plt.ylabel('Proportion of Variance Explained')
plt.grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```




```
# your code for cumulative variance explained plot here

plt.style.use('dark_background')
SYRACUSE_BLUE = '#000E54'

pca_model = pipe_pca_model_1.stages[-1]
n_components = len(pca_model.explainedVariance)

cumulative_variance = np.cumsum(pca_model.explainedVariance)

plt.figure(figsize=(10, 6))
plt.plot(
    np.arange(1, n_components + 1),
    cumulative_variance,
    marker='o',
    color=SYRACUSE_BLUE,
    linewidth=2,
    markersize=8,
    markeredgewidth=1,
    markeredgecolor='white'
)

plt.title('Cumulative Variance Explained Plot', pad=15)
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Proportion of Variance Explained')
plt.grid(True, linestyle='--', alpha=0.7)

plt.ylim(bottom=0, top=1.05)

plt.tight_layout()
plt.show()

plt.show()
```



your answers here

Q8

Starting with `pipe_pca_1` from the previous question, transform the pipeline and save the resulting dataframe to a variable named `pca_fun`.

Extract the output from the standard scaler column from the first row of `pca_fun` and store in a variable named `row1_centered`.

Manually compute 5 PCA scores by projecting `row1_centered` onto the first 5 loading vectors which were computed in your PCA object. Save the 5 projected pca scores in a varialbe called `proj_scores`.

Extract the first 5 PCA scores from the first row of the pca_fun scores column and save them in a variable named `pca_fun_scores`.

```
# your answer here
pca_fun = pipe_pca_model_1.transform(ds_features)
row1_centered = pca_fun.select('scaled_features').first()['scaled_features']
loading_vectors = pipe_pca_model_1.stages[1].pc.toArray()[:, :5]
proj_scores = np.dot(row1_centered.toArray(), loading_vectors)
pca_fun_scores = pca_fun.select('scores').first()['scores'][:5]
```

```
# do not modify
print(proj_scores)
print(pca_fun_scores)
```

```
[-0.8273713  -0.0963639   1.94471186 -0.09190493  2.02351546]
[-0.8273713  -0.0963639   1.94471186 -0.09190493  2.02351546]
```

Q9

Perform an **inverse transform** on the `proj_scores` variable and store the result in a variable named `inverse`.

The grading cell below prints `inverse` and the original `row1_centered` data such that they are right next to each other.

If `inverse` is different than `row1_centered`, explain why. How you could modify the forward and reverse transformation process such that the resulting `inverse` data almost exactly matches `row1_centered`.

```
# your code inverse_pca = np.dot(proj_scores, loading_vectors.T)

inverse_pca = np.dot(proj_scores, loading_vectors.T)
```

```
scaler_model = pipe_pca_model_1.stages[0]
mean = scaler_model.mean
std = scaler_model.std
inverse = inverse_pca * std + mean
```

your answer here

```
# do not modify
print(row1_centered[0:5])
print(inverse[0:5])
```

```
[-0.33920093 -0.31538444 -0.11677724  0.80235477 -0.23850672]
[ 0.2392728  -0.39532054 -0.05361332  0.47784174 -0.08726838]
```

Q10

Implement your modification so that `row1_centered` and `inverse` match almost exactly.

```
# your code here
pca_fun = pipe_pca_model_1.transform(ds_features)
row1_centered = pca_fun.select('scaled_features').first()['scaled_features']
loading_vectors = pipe_pca_model_1.stages[1].pc.toArray()  # All loading vectors
proj_scores = np.dot(row1_centered.toArray(), loading_vectors.T)
inverse_pca = np.dot(proj_scores, loading_vectors)

scaler_model = pipe_pca_model_1.stages[0]
```