# Documentación del Mini Proyecto

Semana: 6


**Integrantes:**

| | |
|---|---|
| Andrea Gabriela Zelaya Flores | 12241001 |
| Andrea Jimena Ortez Ramirez | 12241140 |
| Dariel Emilio Sevilla Bueso | 12241006 |
| Hector Daniel Sabillón Cerna | 12241089 |
| Tatiana Zuseth Garcia Ferrufino | 12241079 |


**Asignatura:**

Lenguajes de Programación

**Presentado a:**

Ing. Claudia Cortés

**Sección:**

519

**Fecha de entrega:**

01 de marzo de 2025

# Contenido

# 1. Introducción

El Análisis de Componentes Principales (ACP) es un procedimiento estadístico que se utiliza para reducir la cantidad de variables sin perder información esencial al trabajar con grandes volúmenes de datos. Nuestro mini proyecto aplica esta técnica en Python y C++ con el objetivo de comparar su implementación en dos lenguajes con paradigmas totalmente diferentes (lenguajes de tipado fuerte y dinámico con lenguajes de tipado débil y estático respectivamente, en este caso). A lo largo de la documentación, se detallará la estructura del código, sus funciones principales y la forma en que se abordan los cálculos del ACP en ambos lenguajes para demostrar la diferencia entre estos dos paradigmas.

## 2. **Descripción del Problema**

En este proyecto, se realiza el análisis de componentes principales en 2 lenguajes de programación. El primero, Python, es un lenguaje fuertemente tipado, y dinámico. C++ es su completo opuesto. Es un lenguaje débilmente tipado y estático. El análisis de componentes principales es usado frecuentemente en estadística para comprimir los datos y simplificarlos. Con estos datos simplificados se pueden encontrar relaciones y patrones importantes, con una pérdida de información mínima.

## 3. **J**ustificación y Explicación

El primer lenguaje utilizado fue Python. Elegimos este lenguaje debido a que, sintácticamente, es muy diferente a los lenguajes vistos en la universidad, los cuales son C++ y java. Esto hace que sea un nuevo lenguaje útil que aprender a usar para aquellos miembros del equipo que no lo han usado, y es buena práctica para aquellos que sí. Adicionalmente, Python es un lenguaje muy práctico y cómodo de usar, que es particularmente útil para temas de algebra lineal. También nos ayudó a tener algo de practica adicional con un lenguaje que hoy en día es de los más relevantes, al ser el más popular para el desarrollo de redes neuronales.

El segundo lenguaje utilizado fue C++. Este lenguaje es el opuesto a Python en el sentido de que, mientras Python es dinámico y fuertemente tipado, C++ es débilmente tipado y estático. También es su opuesto en el sentido de que consiste en una sintaxis menos fluida y muchas menos conveniencias que Python. C++ fue un lenguaje que todos habíamos visto en programación 3, por lo que es uno en el que todos teníamos experiencia. Es interesante de práctica, ya que le da más libertad al programador de lo que proveen la mayoría de los lenguajes. Es un lenguaje útil de practicar, ya que hace que el programador este más consciente de cómo funcionan las estructuras de datos que utiliza seguido.

# 4. **Documentación del Código**

**Python**

Para el desarrollo del ACP en Python, se creó una clase llamada FileReader. En su constructor, recibe de parámetro una string, la cual es la dirección del .csv que se utiliza para el algoritmo, y la almacena como un atributo de la clase. En esta clase se realizaron todos los pasos del algoritmo, con los siguientes métodos:

1) **def readData(self)**

Este método utiliza la string del csv, y usa la librería pandas para leer el .csv y extraer la matriz de información. La información leída de pandas se almacena en el atributo **self.data**

2) **def centerReduce(self)**

Este método no recibe nada. Usa la **self.data.shape[0]** para acceder a la matriz, y centra y reduce cada valor, restando a cada elemento la media de su respectiva columna y dividiéndolo por su desviación estándar, ignorando la primera columna, ya que es la de los nombres. Se almacena el resultado en el atributo **self.dataProcessed.** Como esta matriz igual incluye la primera columna de nombres para acceder a únicamente los valores por lo general se usa el siguiente código:

rows = self.dataProcessed.values

rows[:,1:] (esta es la matriz de valores)

3) **def createRelationsMatrix2(self)**

Este método primero ejecuta el método anterior. Luego, usando la librería **numpy**, calcula la matriz de correlaciones de la matriz almacenada en **self.dataProcessed** y la almacena en **self.correlationMatrix**. También retorna esta matriz.

**4) def eigen(self)**

Este método, llamado así por **eigenvalues** y **eigenvectors** (valores y vectores propios en inglés), calcula los valores y vectores propios de la matriz de correlación en **self.correlationMatrix**, y los ordena de mayor a menor. **Numpy** ya devuelve los vectores propios en la forma matricial pedida en el paso 4, por lo que solo se necesitó ordenarlo. Los valores y vectores propios se almacenan en **self.properValues** y **self.properVector** respectivamente.

**5) def principalComponents(self)**

Se calcula la matriz de componentes principal con las matrices en **self.dataProcessed** y **self.properVector**, y se almacena en **self.pComponents**.

6) **def individualsQualities(self)**

Este método calcula la matriz de calidad de individuos con la formula dada, usando las matrices en **self.pComponents** y **self.dataProcessed**, y la almacena el resultado en **self.iQualities.**

7) **def variablesCoordinates(self)**

Multiplica una matriz diagonal formada por la raíz de cada valor propio en **self.properValues** y **self.dataProcessed.values** para conseguir la matriz de coordenadas.

8) **def variablesQualities(self)**

Calcula la matriz de calidad de variables con la matriz de coordenadas en **self.vCoordinates.**

9) **def inertiaVector(self)**

Calcula el vector de inercia con los valores propios en **self.properValues.**

10) **def graphCorrelationMatrix(self, dim1, dim2)**

Utilizando la librería matplotlib.pyplot, crea las gráficas del círculo de correlación de las dimensiones dadas usando la matriz de coordenadas en **self.vCoordinates.**

**11) graphPrincipalPlane(self, dim1, dim2):**

Utilizando la librería matplotlib.pyplot, crea las gráficas de planos de puntos de las dimensiones dadas usando la matriz de coordenadas en **self.pComponents.**

**C++**

Para el desarrollo del ACP en Python, se creó un .h llamado matplotlibcpp que es más que todo, la librería que usamos para hacer los gráficos. En archivo .cpp llamado PrincipalComponentAnalysis se realizaron todos los pasos del algoritmo, con los siguientes métodos:

1) **void imprimirMatriz(const vector<vector<double>>& matriz**

Método para imprimir cada una de las matrices que vamos formando en el ACP.

2) **vector<vector<double>> leerCSV(const string& nombreArchivo)**

Método para leer el archivo de entrada de formato .csv y guardarlo en un vector de vectores.

3) **vector<vector<double>> estandarizar(const vector<vector<double>>& matriz)**

Centra y reduce la matriz, sacando la media y desviación estándar de cada fila. Por cada elemento, se resta la media y se divide por la desviación estándar correspondiente.

4) **vector<vector<double>> correlaciones(const vector<vector<double>>& matriz)**

Recibe la matriz estandarizada y retorna la matriz de correlaciones. Para construir la matriz, se hace la multiplicación matricial de la matriz estandarizada por su transpuesta. Esto nos da como resultado una matriz cuadrada.

5) **pair<vector<double>, vector<vector<double>>> calcularValoresVectoresPropios(const vector<vector<double>>& matriz)**

Recibe la matriz de correlaciones/covarianzas. Calcula los valores y vectores propios de matriz con ayuda de la librería Eigen para C++. Además, ordena los valores y vectores en orden de mayor a menor y retorna ambos vectores como un par se elementos.

6) **vector<vector<double>> componentes(const vector<vector<double>>& matrizX, const vector<vector<double>>& matrizV)**

Calcula la matriz de componentes principales C, utilizando la formula $C = XV$. Donde X es la matriz estandarizada y V es la matriz de los vectores propios.

7) **vector<vector<double>> calidades(const vector<vector<double>>& matrizX, const vector<vector<double>>& matrizC)**

Calcula la matriz de calidades de individuos Q, utilizando la formula:

$$Q_{ir} = \frac{(C_{i,r})^2}{\sum_{j=1}^{m}(X_{ij})^2} \quad \text{para } i = 1, 2, \ldots, n; \quad r = 1, 2, \ldots, m.$$

Donde X es la matriz estandarizada y C es la matriz de componentes principales. Donde por cada fila se hace la sumatoria de los elementos en cada fila al cuadrado y estos son divididos por cada elemento de la fila de componentes principales al cuadrado, para conseguir cada elemento de la matriz de calidades Q.

8) **vector<vector<double>> coordenadasVariables (const vector<double>& valoresPropios, const vector<vector<double>>& vectoresPropios)**

Calcula la matriz de coordenadas de las variables *T*, utilizando la fórmula **T = V * √(λ),** donde V es la matriz de vectores propios de la matriz de correlación y λ es la matriz diagonal de los autovalores. Se reciben valoresPropios y vectoresPropios como parámetros y se retorna la matriz T.

9) **vector<vector<double>> calidadesVariables (const vector<vector<double>>& matrizT, const vector<double>& valoresPropios)**

Calcula la matriz de calidades de las variables *S*, utilizando la fórmula $S_{ij} = \frac{{T_{ij}}^2}{\lambda_j}$ . $T_{ij}$ corresponde a la matriz de coordenadas y $\lambda_j$ corresponde al autovalor correspondiente al componente principal j. Recibe matrizT y valoresPropios como parámetros y retorna la matriz S.

10) **vector<double> vectorInercias(const vector<double>& valoresPropios)**

Calcula el vector de inercias *I*, recibiendo como parámetros valoresPropios y retorna el vector I.

11) **void PlanoPrincipal(vector<string>& etiquetas, vector<vector<double>> componentesP, int columna1, int columna2)**

Recibe las etiquetas (ya sea nombres, materias, lo que el usuario quiera) de los puntos a graficar, la matriz de componentes principales C y las dos columnas a graficar de la misma, para imprimir el Plano Principal.

**12) void Circulo_Correlacion(vector<string>& etiquetas, vector<vector<double>> matrizT, int columna1, int columna2, vector<double>vectorI)**

Recibe las etiquetas (ya sea nombres, materias, lo que el usuario quiera) de los elementos a graficar, la matriz de coordenadas T y las dos columnas a graficar de la misma, para imprimir el Circulo de Correlación, así como también el vector de Inercias que pertenece a cada columna de esta.

Finalmente hacemos la impresión de todos los pasos realizados por los métodos anteriormente explicados en el main.

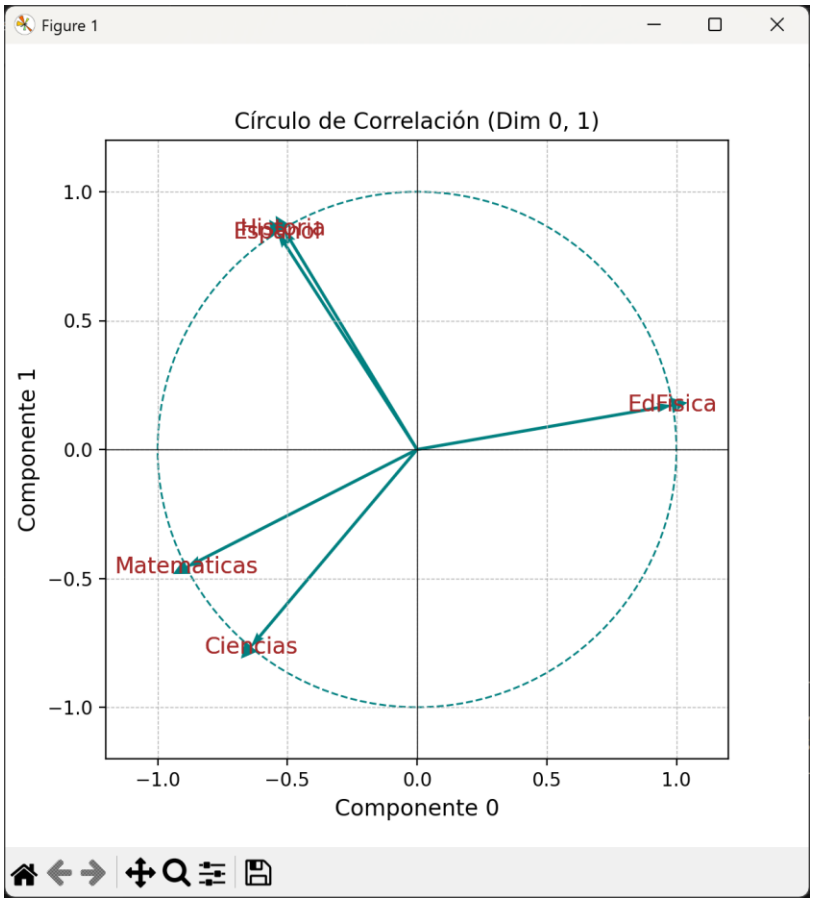
# 5. Ejemplos y demostraciones

## Python

```
Paso #1 - Matriz Centrada y Reducida
  Unnamed: 0  Matematicas  Ciencias   Espanol  Historia  EdFisica
0      Lucia     0.232631 -0.752986  1.788485  0.657923  0.658581
1      Pedro     0.786514  1.145849 -0.538996 -0.845901 -0.476903
2       Ines     0.897290  1.014894  0.318497  0.093989  0.090839
3       Luis    -1.982900 -0.752986 -1.518987 -0.845901  1.794065
4     Andres    -0.875135 -1.080371  0.073499  0.939889 -0.136258
5        Ana     1.118843  1.276803 -0.049000  0.093989 -1.044645
6     Carlos    -0.542805 -0.818463  0.563495  1.033878 -0.249807
7       Jose     1.229620  1.342280 -0.293998  0.093989 -1.612388
8      Sonia    -0.875135 -1.080371 -1.518987 -2.255735  1.453420
9      Maria     0.011078 -0.294647  1.175990  1.033878 -0.476903

Paso #2 - Matriz de Correlaciones
[ 1.0000000000000002 0.854078777925743 0.3845742420832209 0.2071942525126377 -0.7871626873142727 ]
[ 0.854078777925743 1.0000000000000002 -0.020052184332105795 -0.02153942000025033 -0.6877205571207659 ]
[ 0.3845742420832209 -0.020052184332105795 1.0 0.820916192604564 -0.36554341640973126 ]
[ 0.2071942525126377 -0.02153942000025033 0.820916192604564 1.0000000000000004 -0.5080013180428953 ]
[ -0.7871626873142727 -0.6877205571207659 -0.36554341640973126 -0.5080013180428953 1.0 ]
```

```
Paso #3 - Vectores y Valores Propios
Vectores Propios
[[-0.52664397 -0.2704963  -0.43820071 -0.62387762 -0.26121779]
 [-0.42493622 -0.50807221 -0.04049491  0.32538951  0.67362724]
 [-0.35914704  0.56208159 -0.56227583  0.48374732 -0.07008647]
 [-0.35269747  0.58648985  0.39418032 -0.42043348  0.44664495]
 [ 0.53730181  0.09374599 -0.57862603 -0.30679407  0.52305619]]

Valores propios
[2.89324967 1.62865042 0.34659605 0.00889139 0.12261246]

Paso #4 - Matriz de Vectores Propios
[[-0.52664397 -0.2704963  -0.43820071 -0.26121779 -0.62387762]
 [-0.42493622 -0.50807221 -0.04049491  0.67362724  0.32538951]
 [-0.35914704  0.56208159 -0.56227583 -0.07008647  0.48374732]
 [-0.35269747  0.58648985  0.39418032  0.44664495 -0.42043348]
 [ 0.53730181  0.09374599 -0.57862603  0.52305619 -0.30679407]]

Paso #5 - Matriz de Componentes Principales
[ -0.3230626266016874 1.772524503141889 -1.1988007368012144 -0.05501531987702446 -0.0036333843126489795 ]
[ -0.6654405682498179 -1.6387021469393976 -0.14547627676678138 -0.02306467937995385 0.12337729563355185 ]
[ -1.0025470460795078 -0.5156924670549482 -0.6288876353863664 0.5164435083015092 -0.14287582397517526 ]
[ 3.1720948098127866 -0.26278200513647587 0.38196026603420985 0.6777769099837783 0.06250355441488142 ]
[ 0.4888679696638494 1.3654020988140008 0.8352356974443502 -0.15579197457514377 -0.12336725505033837 ]
[ -1.7086332242740485 -1.0217004400454872 0.12707706692367948 0.06683294716135213 -0.025291502978896707 ]
[ -0.06758577087845002 1.4623364186094374 0.5062404415211305 -0.11792846804266305 -0.0131239803792252 ]
[ -2.0118551642638414 -1.2758645650079772 0.5421500158355587 -0.1977867043985425 -0.017434169918509768 ]
[ 3.042030294188611 -1.2548806928818648 -0.44882860640209854 -0.6399987577488282 -0.03788484049582669 ]
[ -0.9238686733178801 1.3693592965008263 0.029329767597538747 -0.0714674614244877 0.17773010706219053 ]
```

```
Paso #6 - Matriz de Calidades de Individuos
[ 0.022270826773549437 0.6704206697345347 0.3066598386989438 0.0006458478012290272 2.8169917431955025e-06 ]
[ 0.1399055021814009 0.848430538804085 0.006686526987827044 0.00016807808478011058 0.00480935394190659 ]
[ 0.5144688990915026 0.13612289513875295 0.20243971373773556 0.13651967562636635 0.010448816405642557 ]
[ 0.936851989868683 0.006429392054854758 0.013583604844661652 0.04277127571685122 0.0003637375149493321 ]
[ 0.08413951074449778 0.65635371475864 0.24560370300002515 0.008544899889207333 0.005358171607629396 6 ]
[ 0.7326861103846867 0.26197957039412545 0.004052794931721142 0.001120989359852062 0.0001605349296143251 3 ]
[ 0.0018927333897678902 0.8860811386684871 0.10619218887119843 0.005762569999312297 7.13690712343655e-05 ]
[ 0.6736121076980296 0.27091035912296857 0.04891650388397538 0.006510444614762414 5.058468026371102e-05 ]
[ 0.8088299294375152 0.13763694322051473 0.01760723668480743 0.035800443428897225 0.0001254472282658715 8 ]
[ 0.30855427105697325 0.6778692116194983 0.0003109770385120139 0.001846408504143455 0.011419131780873292 ]

Paso #7 - Matriz de Coordenadas de la Variables
[[-0.89579797 -0.46010218 -0.74535991 -0.44431985 -1.06118808]
 [-0.54229761 -0.64839459 -0.05167903  0.85967358  0.4152575 ]
 [-0.21143851  0.33091097 -0.33102532 -0.04126159  0.2847937 ]
 [-0.12350076  0.20536565  0.13802642  0.15639747 -0.14721925]
 [ 0.05066444  0.0088397  -0.05456107  0.04932116 -0.0289289 ]]

Paso #8 - Matriz de Calidades de las Variables
[ 0.27735387523218524 0.12998124765113686 1.6029074631949347 1.610114718449155 126.65282808670653 ]
[ 0.10164580614339591 0.2581373751319586 0.007705576229358333 6.027435342968965 19.39389919438466 ]
[ 0.015451913580719567 0.06723485261292621 0.31615410343633177 0.013885367232057496 9.122017980077475 ]
[ 0.00527173269362695 0.025895704534124587 0.054966851813658235 0.1994917124713552 2.4375827424997825 ]
[ 0.0008871979370267315 4.797858003563991e-05 0.008588991096748076 0.019839554318180866 0.0941226018361381 ]

Paso #9 - Vector de Inercias de los Ejes
[57.86499347 32.5730085   6.93192097  2.45224919  0.17782787]
```



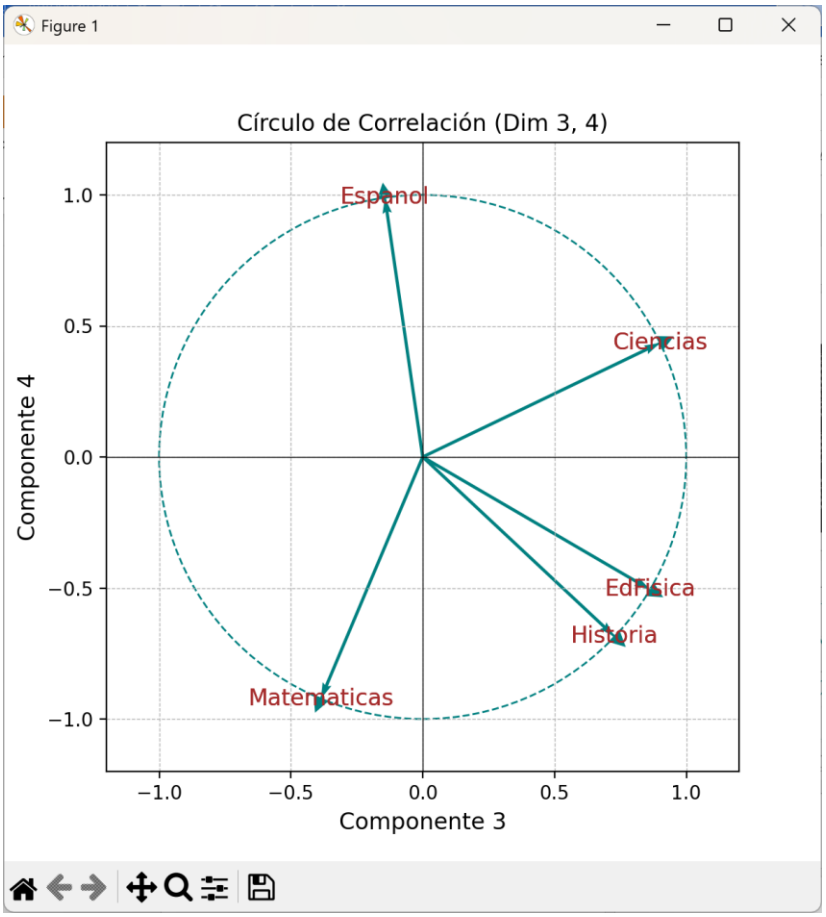


Figure 1
Círculo de Correlación (Dim 3, 4)
Español
Ciencias
EdFísica
Historia
Matemáticas
1.0
0.5
0.0
−0.5
−1.0
Componente 4
−1.0
−0.5
0.0
0.5
1.0
Componente 3

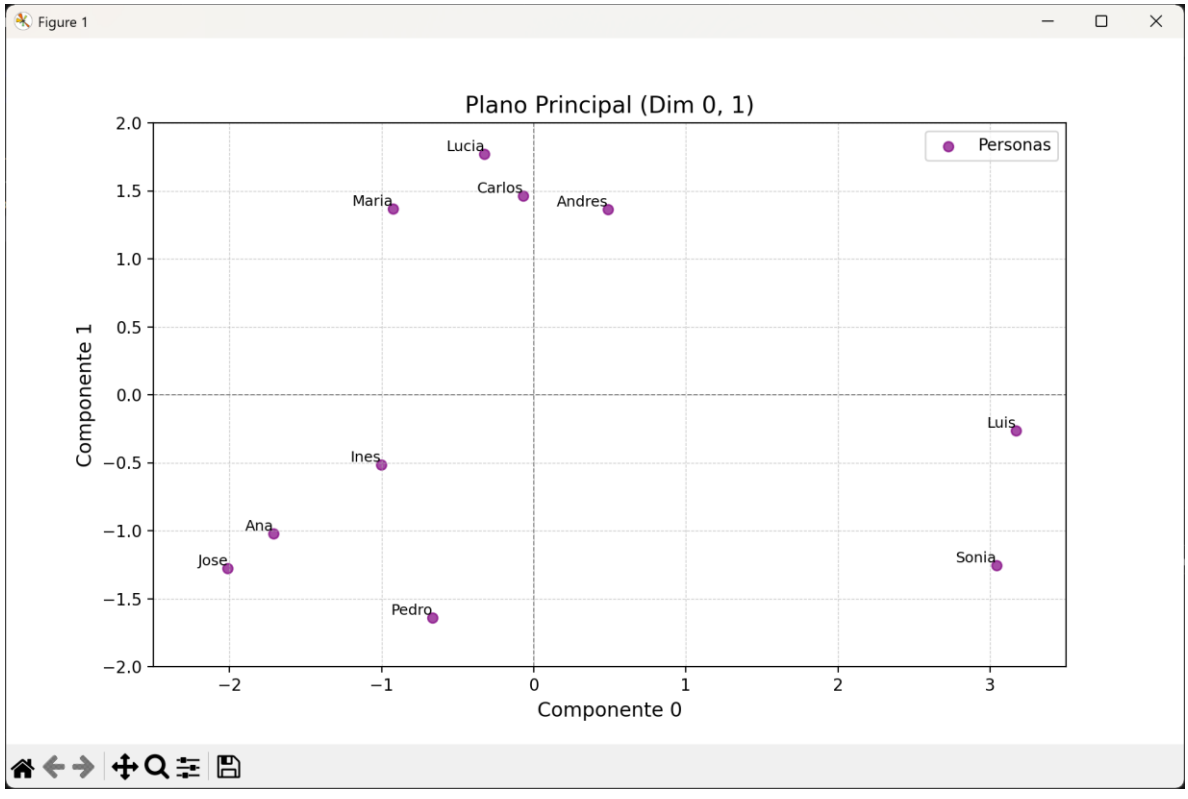
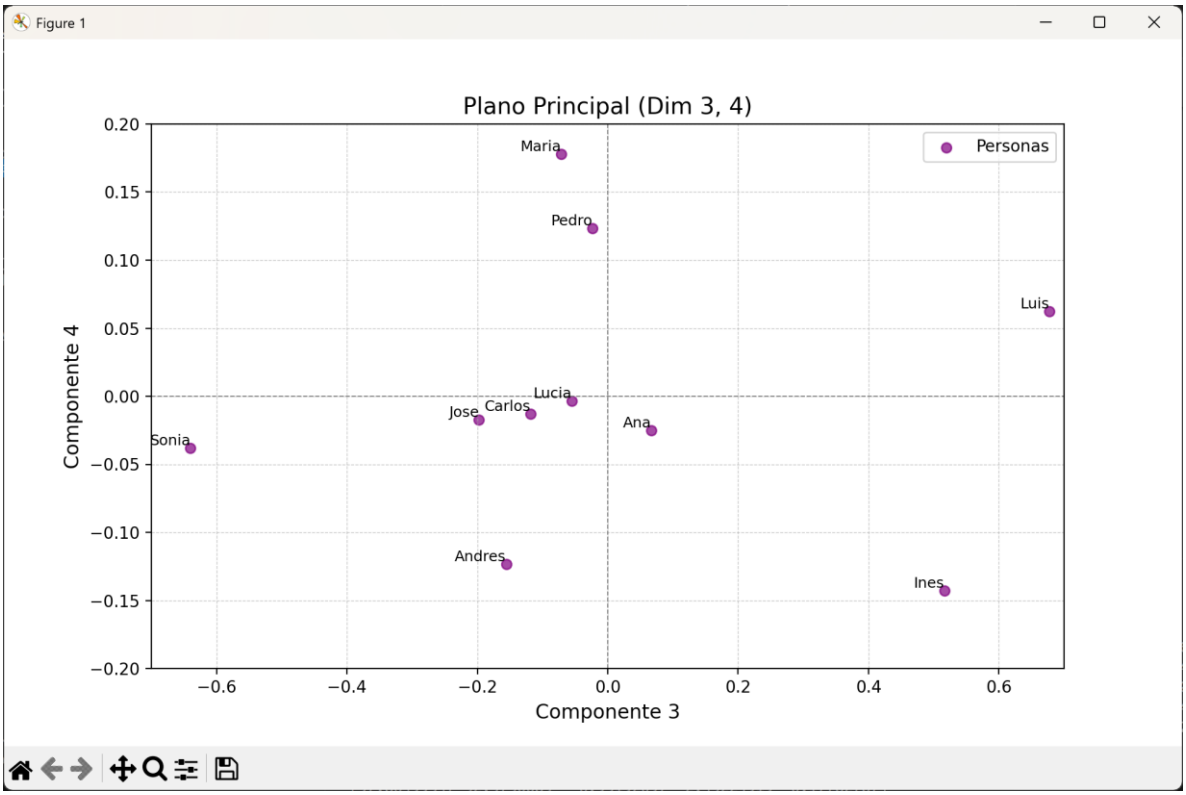


Figure 1
Plano Principal (Dim 0, 1)
Personas
Lucia
Carlos
Maria
Andres
Luis
Ines
Ana
Jose
Sonia
Pedro
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
Componente 1
−2
−1
0
1
2
3
Componente 0




**C++**

```
C:\Users\tatig\Documents\Un

 == Matriz original ==
7       6.5     9.2     8.6     8
7.5     9.4     7.3     7       7
7.6     9.2     8       8       7.5
5       6.5     6.5     7       9
6       6       7.8     8.9     7.3
7.8     9.6     7.7     8       6.5
6.3     6.4     8.2     9       7.2
7.9     9.7     7.5     8       6
6       6       6.5     5.5     8.7
6.8     7.2     8.7     9       7

 == Matriz estandarizada ==
0.232631        -0.752986       1.78849 0.657923        0.658581
0.786514        1.14585 -0.538996       -0.845901       -0.476903
0.89729 1.01489 0.318497        0.0939889       0.0908387
-1.9829 -0.752986       -1.51899        -0.845901       1.79407
-0.875135       -1.08037        0.0734994       0.939889        -0.136258
1.11884 1.2768  -0.0489996      0.0939889       -1.04465
-0.542805       -0.818463       0.563495        1.03388 -0.249807
1.22962 1.34228 -0.293998       0.0939889       -1.61239
-0.875135       -1.08037        -1.51899        -2.25573        1.45342
0.0110777       -0.294647       1.17599 1.03388 -0.476903
```

```
 == Matriz de correlaciones/covarianzas ==
1       0.854079        0.384574        0.207194        -0.787163
0.854079        1       -0.0200522      -0.0215394      -0.687721
0.384574        -0.0200522      1       0.820916        -0.365543
0.207194        -0.0215394      0.820916        1       -0.508001
-0.787163       -0.687721       -0.365543       -0.508001       1


 == Valores propios ==
2.89325 1.62865 0.346596 0.122612 0.00889139

 == Vectores propios ==
0.526644        -0.270496       0.438201        0.261218        0.623878
0.424936        -0.508072       0.0404949       -0.673627       -0.32539
0.359147        0.562082        0.562276        0.0700865       -0.483747
0.352697        0.58649 -0.39418        -0.446645       0.420433
-0.537302       0.093746        0.578626        -0.523056       0.306794

 == Matriz de componentes principales ==
0.323063        1.77252 1.1988  0.0550153       0.00363338
0.665441        -1.6387 0.145476        0.0230647       -0.123377
1.00255 -0.515692       0.628888        -0.516444       0.142876
-3.17209        -0.262782       -0.38196        -0.677777       -0.0625036
-0.488868       1.3654  -0.835236       0.155792        0.123367
1.70863 -1.0217 -0.127077       -0.0668329      0.0252915
0.0675858       1.46234 -0.50624        0.117928        0.013124
2.01186 -1.27586        -0.54215        0.197787        0.0174342
-3.04203        -1.25488        0.448829        0.639999        0.0378848
0.923869        1.36936 -0.0293298      0.0714675       -0.17773
```

```
 == Matriz de calidades de individuos ==
0.0222708       0.670421        0.30666 0.000645848     2.81699e-06
0.139906        0.848431        0.00668653      0.000168078     0.00480935
0.514469        0.136123        0.20244 0.13652 0.0104488
0.936852        0.00642939      0.0135836       0.0427713       0.000363738
0.0841395       0.656354        0.245604        0.0085449       0.00535817
0.732686        0.26198 0.00405279      0.00112099      0.000160535
0.00189273      0.886081        0.106192        0.00576257      7.13691e-05
0.673612        0.27091 0.0489165       0.00651044      5.05847e-05
0.80883 0.137637        0.0176072       0.0358004       0.000125447
0.308554        0.677869        0.000310977     0.00184641      0.0114191

 == Matriz de coordenadas de variables ==
0.895798        -0.460102       0.74536 0.44432 1.06119
0.542298        -0.648395       0.051679        -0.859674       -0.415257
0.211439        0.330911        0.331025        0.0412616       -0.284794
0.123501        0.205366        -0.138026       -0.156397       0.147219
-0.0506644      0.0088397       0.0545611       -0.0493212      0.0289289

 == Matriz de calidades de variables ==
0.277354        0.129981        1.60291 1.61011 126.653
0.101646        0.258137        0.00770558      6.02744 19.3939
0.0154519       0.0672349       0.316154        0.0138854       9.12202
0.00527173      0.0258957       0.0549669       0.199492        2.43758
0.000887198     4.79786e-05     0.00858899      0.0198396       0.0941226

 == Vector de inercias ==
57.865 32.573 6.93192 2.45225 0.177828
```

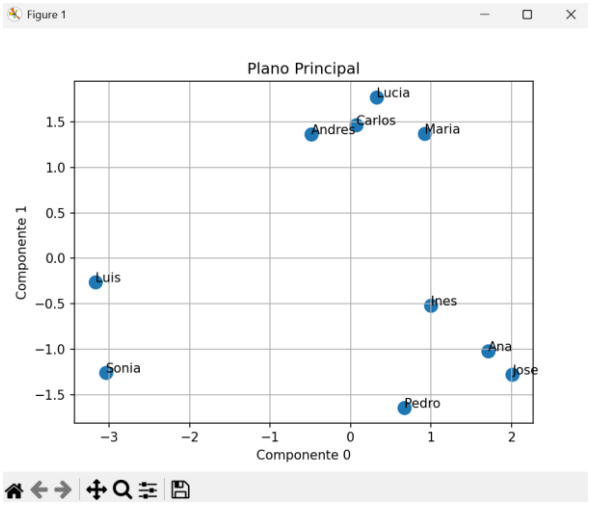


Figure 1
Plano Principal
Lucia
Carlos
Andres
Maria
Luis
Ines
Ana
Sonia
Jose
Pedro
Componente 1
Componente 0

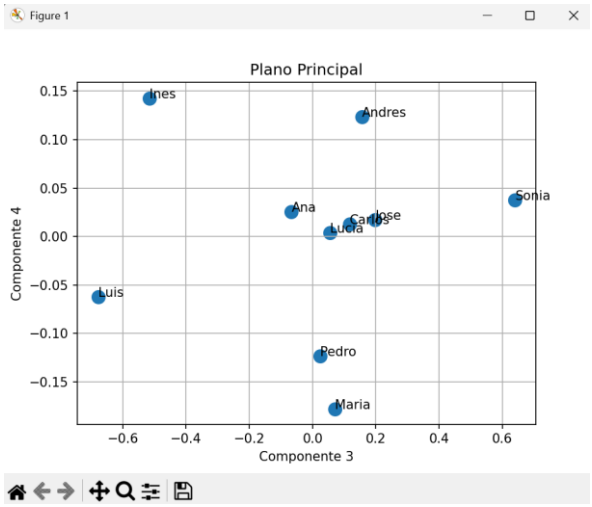


Figure 1
Plano Principal
Ines
Andres
Sonia
Ana
Carlos
Jose
Lucia
Luis
Pedro
Maria
Componente 4
Componente 3

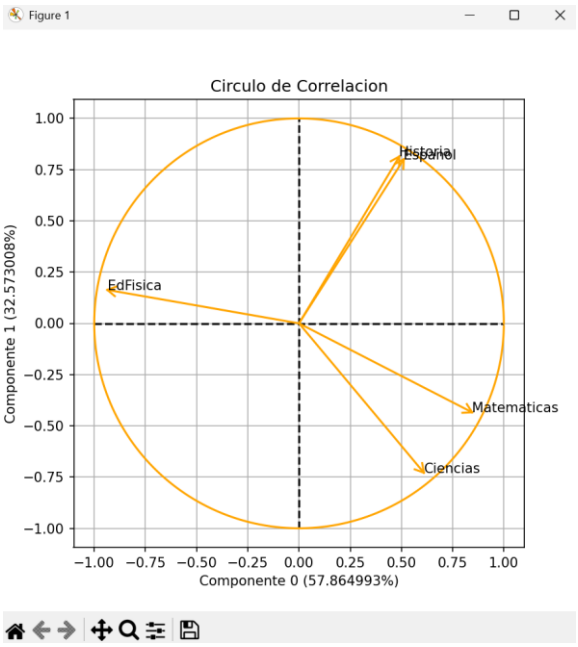


Figure 1
Circulo de Correlacion
Historia
Espanol
EdFisica
Matematicas
Ciencias
Componente 1 (32.573008%)
Componente 0 (57.864993%)

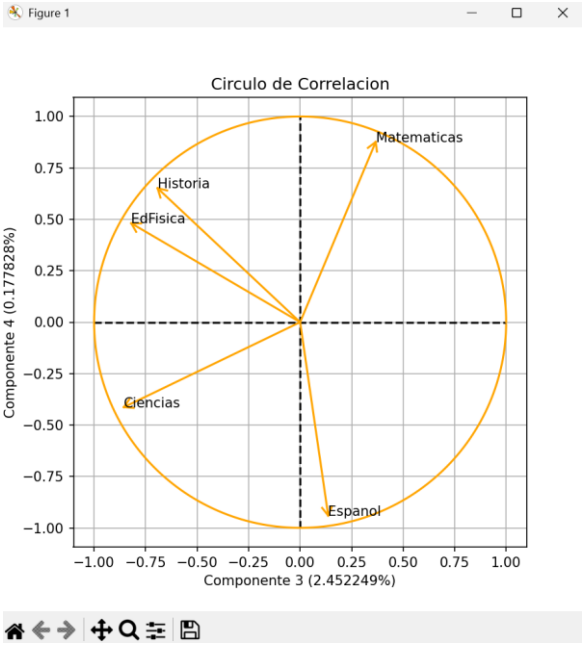


Figure 1
Circulo de Correlacion
Matematicas
Historia
EdFisica
Ciencias
Espanol
Componente 4 (0.177828%)
Componente 3 (2.452249%)

# 6. Análisis

Al construir la solución para el problema en dos lenguajes diferentes con paradigmas distintos, nos dimos cuenta de la gran diferencia que hace la decisión del lenguaje que se va a usar para un proyecto. Consideramos que el tipo de lenguaje influyó extensamente en las soluciones que se crearon.

Las diferencias entre estas soluciones son claras. En Python, fue mucho más fácil implementar conceptos del algebra lineal gracias a la librería NumPy. Asimismo, el manejo de matrices y la creación de gráficos fue mas sencilla. Gracias a que Python es un lenguaje multiparadigma, se pudo utilizar un estilo de programación funcional, donde nos enfocamos en el ¿qué? en lugar del ¿cómo?. Por otro lado, la construcción de la solución en C++ fue más complicada. Las operaciones con matrices tuvieron que ser hechas iterativamente, ya que no podíamos simplemente depender de una librería para hacerlas. Afortunadamente, logramos encontrar y utilizar una librería externa para hacer los cálculos más complicados – los valores propios y vectores propios. Sin embargo, esto también presentó algunos retos. Similarmente, tuvimos que utilizar otra librería externa para la visualización de datos en gráficos. En general, C++ fue mucho más restrictivo que Python.

El mayor aprendizaje que obtuvimos de este proyecto fue como la elección de lenguaje afecta la solución para un problema. Elegir un lenguaje es una cuestión de evaluar el problema y tener prioridades claras. En nuestra opinión, luego de realizar este proyecto, estamos de acuerdo que preferimos usar Python para problemas matemáticos. Sin embargo, si la velocidad de un programa es un factor importante, podríamos considerar C++ u otro lenguaje para esto. Cabe mencionar que es posible crear programar rápidos en Python y

programas lentos en C++ dependiendo de las capacidades del programador, pero, en general, Python es más lento.

Como un comentario adicional, nos gustaría mencionar la diferencia que hacen las librerías en operaciones matemáticas, especialmente en el cálculo de los vectores propios.

En Python obtuvimos los siguientes vectores:

```
Paso #4 - Matriz de Vectores Propios
[[-0.52664397 -0.2704963  -0.43820071 -0.26121779 -0.62387762]
 [-0.42493622 -0.50807221 -0.04049491  0.67362724  0.32538951]
 [-0.35914704  0.56208159 -0.56227583 -0.07008647  0.48374732]
 [-0.35269747  0.58648985  0.39418032  0.44664495 -0.42043348]
 [ 0.53730181  0.09374599 -0.57862603  0.52305619 -0.30679407]]
```

Mientras que en C++ obtuvimos lo siguiente:

```
 == Vectores propios ==
0.526644        -0.270496       0.438201        0.261218        0.623878
0.424936        -0.508072       0.0404949       -0.673627       -0.32539
0.359147        0.562082        0.562276        0.0700865       -0.483747
0.352697        0.58649 -0.39418        -0.446645       0.420433
-0.537302       0.093746        0.578626        -0.523056       0.306794
```

Observando los resultados detenidamente, nos damos cuenta de que los valores son los mismos para los vectores correspondientes, solo difieren por algunos redondeos. Sin embargo, en todos los vectores menos el segundo, los signos están invertidos. Al principio, empezamos a buscar otros problemas con matrices anteriores, pero las matrices de correlaciones eran exactamente las mismas en ambos lenguajes. Luego, buscamos problemas en los cálculos, pero la única diferencia al momento de calcular los vectores propios fue el uso de librerías diferentes. Incluso consideramos multiplicar los vectores de C++ por -1 simplemente para tener los mismos resultados, pero aun haciendo esto, no

quedaban exactamente igual gracias al vector singular que había quedado igual en ambos lenguajes desde el inicio.

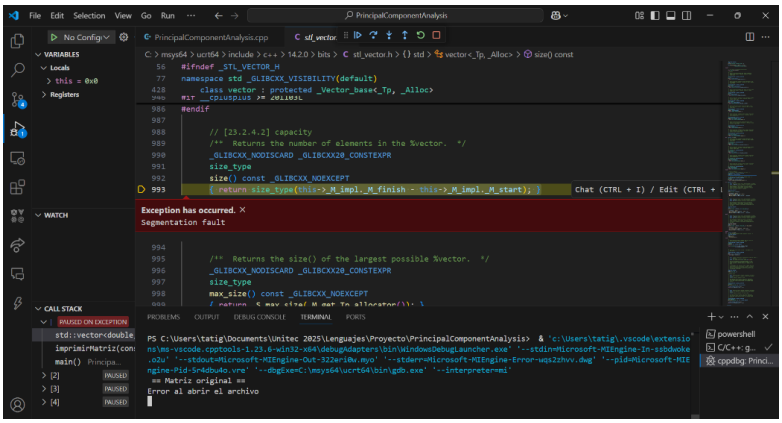
Después de considerar este problema por bastante tiempo, decidimos investigarlo un poco más, y al ver la definición de vectores propios, recordamos algo de álgebra lineal que habíamos olvidado. **Los vectores propios no son únicos**.

Es decir, si A es una matriz cuadrada y X es un vector propio de A, cualquier multiplicación de X por un escalar distinto de 0 también es un vector propio de A. Esto obviamente incluye multiplicar un vector propio por el escalar -1. Las diferencias en resultados de deben, al final, por diferencias en la manera en que las librerías los calculan.

Es extraño al principio ver que los resultados en las matemáticas a veces no son exactamente los mismos, pero que aun así sean válidos. Sin embargo, en el contexto del análisis de componentes principales, esto no afecta el resultado, ya que los vectores propios representan direcciones en el espacio de datos. Un factor de -1 no cambia estas direcciones, solo las orientaciones de los datos. Esto se puede visualizar claramente en los gráficos, donde los gráficos de los diferentes lenguajes son una reflexión del otro, pero los grupos y direcciones son las mismas.

# 7. Dificultades encontradas

Algunas dificultades durante la creación del proyecto ACP surgieron debido a distintos elementos que se habían comentado en clase. Uno de ellos fue el uso de diferentes IDEs para el mismo lenguaje, en este caso, C++. Como se vio en clase, esto causaba muchos problemas de compatibilidad, generando errores en el programa. Por esta razón, finalmente optamos por utilizar el mismo IDE, ya que así resultaba menos complicado adaptar el código a nuestras computadoras y se reducían los ajustes necesarios para su correcto funcionamiento. Otro desafío fue la verificación y corrección de los datos en cada matriz. Para ello, analizamos el código en Python, donde comprendimos a qué se refería la afirmación de que puede ser menos comprensible al ser un lenguaje funcional. Por ejemplo, al encontrarnos con funciones como:




***pComponents = rows[:,1:]@self.properVector***

Nos resultaba difícil comprender su lógica y funcionamiento a simple vista, por lo que tuvimos que investigar al respecto. En comparación, el código en C++ nos resultaba más fácil de analizar, ya que estábamos acostumbrados a su estructura imperativa.

También cabe mencionar que, debido a que C++ es un lenguaje más restrictivo, no permitía el uso de ciertas funciones que sí estaban disponibles en Python.

# 8. Conclusiones

El presente proyecto concluye que el ACP es efectivo para simplificar un conjunto de datos al transformar variables correlacionadas en nuevo conjunto de variables no correlacionadas llamadas componentes principales. Se hizo la implementación de esta técnica en Python y C++, con el propósito de entender más a fondo las diferencias de implementación entre un lenguaje fuertemente tipado y dinámico (Python) y débilmente tipado y estático (C++). Los pasos clave para la realización de esto fueron la normalización de datos, cálculo de matrices y obtención de valores y vectores propios.

Al comparar las implementaciones de la técnica en ambos lenguajes de programación, se observa que Python facilitó enormemente el desarrollo, debido a que tiene más librerías dedicadas al uso matemático de este tipo, como NumPy. Al contrario, C++ requirió un manejo más manual de las matrices, lo que aumentó la complejidad del código, pero permitió un mayor control sobre la eficiencia del procesamiento de los datos. Adicionalmente, en C++ se enfrentó la dificultad del uso de diferentes IDEs para un solo lenguaje, lo cual forzó el uso de uno mismo para evitar errores adicionales.

De este modo, se concluye que Python es más eficiente e ideal para el análisis de datos que requieran el uso del álgebra lineal, mientras que C++ es más adecuado para un entorno donde se debe optimizar el rendimiento.

## 9. Bibliografías o referencias

Repositorio de C++: https://github.com/agzelaya/PrincipalComponentAnalysis

Repositorio de Python: https://github.com/darielsevilla/Python-PCA-Analyst