Algoritmo para analisis de componentes principales

```
from cmath import sqrt
import numpy as np
import numpy.linalg as alg

from sklearn.decomposition import PCA

import seaborn as sns
from matplotlib import pyplot as plt

import pandas as pd

np.set_printoptions(precision=6, suppress=True)
```

La siguiente clase implementa el algoritmo de analisis de componentes principales

Para la implementación del algoritmo se siguen los siguientes pasos:

- Centrar y reducir los datos
- Calcular la matriz de correlación
- Calcular los valores y vectores propios de la matriz de correlación
- Ordenar los vectores propios basado en la magnitud de sus valores propios asociados
- Selección de las n columnas utilizadas como componente principal

A Este punto se pueden calcular metricas para la eficiencia del algoritmo, como el vector de inercia que representa el porcentaje de importancia para cada columna de la transformación obtenida.

```
class My_PCA:
    def __init__(self, n_components = -1):
        self.n_components = n_components
        self.matrix = []
        self.inertia = []
        self.points = []
        self.eigenvalues = []
        self.eigenvectors = []
    pass

    def fit(self, X):
        redux = self.center_and_scale(X)
```

```
        corr = self.corr_matrix(redux)
        self.eigenvalues, self.eigenvectors = self.eigensomething(corr)
        self.matrix = self.pca_matrix(self.eigenvectors)
        self.remove_components()
        self.inertia = self.calculate_inertia(self.eigenvalues)
        self.points = self.calculate_points(self.eigenvalues)
        pass


    def transform(self, X):
        return np.matmul(X, self.matrix)


    def center_and_scale(self, X):
        _X = X

        mmean = np.mean(_X, axis=0)
        mstd = np.std(_X, axis=0)

        _X = (_X - mmean) / mstd
        return _X


    def corr_matrix(self, X):
        n, m = X.shape
        return (1/n) * np.matmul(X.transpose(), X)


    def eigensomething(self, R):
        w, v = alg.eigh(R)
        sort_index = np.argsort(abs(w))[::-1]
        sorted_eigenvals = w[sort_index]
        sorted_eigenvecs = v[:, sort_index]

        return (sorted_eigenvals, sorted_eigenvecs)


    def pca_matrix(self, vectors):
        return np.array(vectors)


    def remove_components(self):
        self.matrix = np.delete(self.matrix, np.s_[self.n_components - 1 : -1], axis = 1)
        self.eigenvalues = self.eigenvalues[0:self.n_components]
        pass

    def calculate_inertia(self, eigenvalues):
        n, m = self.matrix.shape
        return eigenvalues / m


```

```
    def calculate_points(self, eigenvalues):
        w0 = eigenvalues[0]
        w1 = eigenvalues[1]
        v0 = self.matrix[:,0]
        v1 = self.matrix[:,1]

        return ((v0 * sqrt(w0)).real, (v1 * sqrt(w1)).real)
```

Poniendo a prueba la imlementación

Se importará el dataset del Titanic para analizar cuales de sus caracteristicas pueden ser de interes

```
# Importación del dataset
base_dataset = pd.read_csv("/content/titanic.csv")
print(base_dataset.head())

# Busqueda de valores ausentes
percent_missing = base_dataset.isnull().sum() * 100 / len(base_dataset)
missing_value_df = pd.DataFrame({'column_name': base_dataset.columns,
                                 'percent_missing': percent_missing})

print(missing_value_df)
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
             column_name  percent_missing
PassengerId  PassengerId         0.000000
Survived        Survived         0.000000
Pclass            Pclass         0.000000
Name                Name         0.000000
```

```
Sex                Sex         0.000000
Age                Age        20.091673
SibSp              SibSp       0.000000
Parch              Parch       0.000000
Ticket           Ticket        0.000000
Fare                Fare       0.076394
Cabin              Cabin       77.463713
Embarked        Embarked       0.152788
```

Información basica

Podemos observar que hay datos utiles para ubicar a cada uno de los pasajeros, sin embargo son datos que no serán utiles para la clasificación de si este pasajero sobrevive o no, por lo que las columnas "PassengerId", "Name", "Ticket" ser completamente eliminadas, tambien podemos notar que la columna de Cabina tiene un 77% de información faltante por lo que podemos eliminarla completamente.

Codificación de información nominal

Las clases "Sex", "Embarked" y "Survived" se pueden codificar con la tecnica de One Hot Encoding para facilitar el proceso de clasificación a futuro, la variable "Pclass" puede ser más compleja de tratar ya que es una variable ordinal, sin embargo no es una variable continua por lo que será mejor evitar la complejidad de esta variable y codificarla igualmente.

```
# Eliminación de variables innecesarias
titanic_data = base_dataset.drop(["PassengerId", "Name", "Cabin", "Ticket"], axis=1)
titanic_data.dropna(axis=0, how="any", inplace=True)

# One hot encoding
cnames = ["Survived", "Pclass", "Sex", "Embarked"]
for cname in cnames:
    dummies = pd.get_dummies(titanic_data[cname], prefix=cname)
    titanic_data = titanic_data.drop(cname, axis=1)
    titanic_data = titanic_data.join(dummies)
```

Preparación de los datos para utilizar PCA

Para poder procesar los datos con el algoritmo de PCA es necesario transformar los datos de un Pandas dataset a una matriz numerica de NumPy.

En la implementación de PCA propia se considera que los datos serán centrados y reducidos dentro de la función de ´fit´ sin embargo al utilizar PCA implementado por sklearn los datos deben ser centrados y reducidos antes de ser introducidos en el algoritmo de PCA.

```
#   Conversion de datos a numpy matrix
titanic_as_np = titanic_data.to_numpy()

#   Uso del metodo PCA implementado
mypca = My_PCA(n_components=4)
mypca.fit(titanic_as_np)

#   Uso del metodo PCA de la biblioteca sklearn
pca = PCA(n_components=4)

# Centrado y reducido previo al uso de sklearn
titanic_prep = mypca.center_and_scale(titanic_as_np)
pca.fit(titanic_prep)
```

```
PCA(n_components=4)
```

Uso del algoritmo de PCA sobre los datos originales

Para modificar la matriz de datos originales solo es necesario centrar y reducir los datos, luego multiplicar la matríz resultante por la matríz obtenida con el algoritmo de PCA

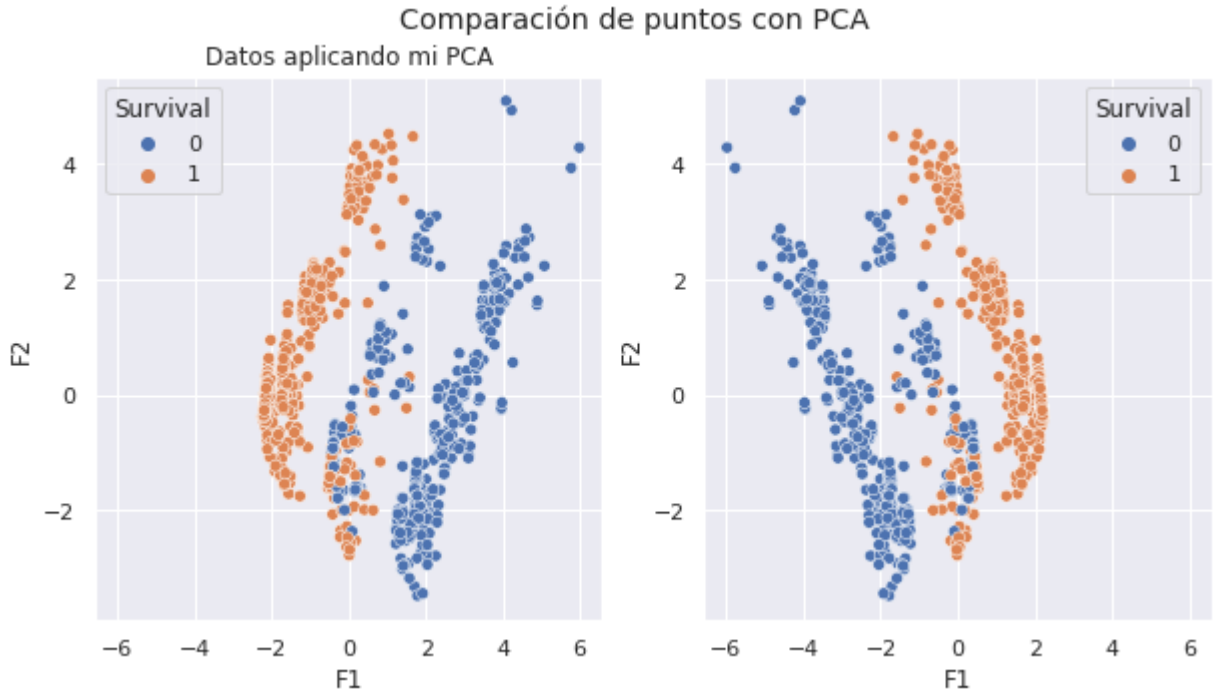
```
# titanic_prep se encuentra centrada y reducida del bloque de codigo anterior

# Transformación de los datos a partir del PCA implementado
my_matrix = mypca.transform(titanic_prep)

# Transformacion de los daatos a partir del PCA sklearn
sk_matrix = pca.transform(titanic_prep)
```

Graficación de los resultados

Podemos ver como los datos se agrupan en 2 categorías princiales, sin embargo existe un tercer grupo donde los datos se encuentran mezclados, aún así este grupo es menor a los 2 grupos principales, esto se logra al transformar las coordenadas de cada punto aumentando al maximo la varianza entre las clases.

```
transformed_sk_data = pd.DataFrame({'F1' : sk_matrix[:, 0], 'F2': sk_matrix[:, 1], 'Surviv
transformed_my_data = pd.DataFrame({'F1' : my_matrix[:, 0], 'F2': my_matrix[:, 1], 'Surviv
sns.set()

fig, axes = plt.subplots(1, 2, sharex=True, figsize=(10,5))
fig.suptitle('Comparación de puntos con PCA')

sns.scatterplot(ax=axes[0], x="F1", y="F2", hue='Survival', data=transformed_sk_data)
```

```
axes[0].set_title('Datos aplicando PCA de sklearn')

sns.scatterplot(ax=axes[1], x="F1", y="F2", hue='Survival', data=transformed_my_data)
axes[0].set_title('Datos aplicando mi PCA')

plt.show()
```



Inercia e importancia de caracteristicas

A partir de la inercia podemos saber que tan importante será cada columna de datos final, pero tambien podemos obtener esta información de manera grafica dentro de un circulo de correlación, esto nos dirá cuanta correlación o "importancia" tendrá cada columna para obtener un resultado.

Para este caso podemos ver que las variables con una correlación más fuerte son:

- Supervivencia
- Sexo
- Clase
- Donde ha embarcado
- Tarifa
- Edad

```
x, y = mypca.points

(fig, ax) = plt.subplots(figsize=(10, 10))
for i in range(0, len(titanic_data.columns)):
    ax.arrow(0, 0,
```

```
            x[i],y[i],
            head_width=0.1,head_length=0.1)
    plt.text(x[i],y[i],titanic_data.columns[i])

an = np.linspace(0, 2 * np.pi, 100)
plt.plot(np.cos(an), np.sin(an))
plt.axis('equal')
ax.set_title('Circulo de correlación')
plt.show()
```