MANUAL TÉCNICO Y DE USO DEL MODELO

MODELO: ÁRBOL DE DECISIÓN

1. Instalar las librerías requeridas.

- python -m pip install --upgrade pip
- python -m pip install Pillow
- python -m pip install -U matplotlib
- python -m pip install scipy
- python -m pip install pandas
- python -m pip install scikit-learn
- python -m pip install openpyxl

2. Cargamos las librerías que necesitará el programa.

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
```

3. Se establece la condición de estado denominada "Aprobado" para un estudiante, la cual se verifica mediante dos criterios específicos: la obtención de un promedio igual o superior a 7.0 y una asistencia que alcance o supere el 70%. Este proceso se implementa mediante el empleo de una función designada con el nombre de "Condición Aprobado". Dicha función se encarga de examinar las columnas pertinentes, evaluando si se cumplen las condiciones establecidas para determinar el estado "Aprobado" o su contraparte.

```
def CondicionAprobado(fila): # Definición de la función que asigna la condición del estado del estudiante x materia
    if fila["ASISTENCIA"]>=70 and fila["PROM FINAL"]>=7.0:
        cap="Aprobado"
    else:
        cap="Reprobado"
    return cap
```

4. Se procede a preparar los datos para su carga en el sistema de gestión de base de datos NoSQL Cassandra.

```
def convertird(dato,tipo):
    convertido=""
    if tipo == "text":
        convertido=str("'" + str(dato) + "'")
    elif tipo == "int":
        convertido=str(dato)
    elif tipo == "date":
        convertido=str("'" + str(datetime.strftime(datetime.strptime(str(dato), '%d/%m/%Y %H:%M:%S'),'%Y-%m-%d')) + "'")
    elif tipo == "bigint":
        convertido=str(dato)
    elif tipo == "float":
        convertido=str(dato)
    else:
        convertido=str("'" + str(dato) + "'")
    return convertido
```

5. Se especifica la ruta de acceso que alberga los datos correspondientes a ciclos anteriores destinados al proceso de modelado, los cuales están contenidos en una lista de archivos ubicada en la carpeta denominada "ArchiDirec". A continuación, se establece la conexión con el clúster de Cassandra, y se realiza la preparación de la tabla denominada "calificaciones".

```
ArchiDirec=os.getcwd()+"\\data\\calif\\"
# Obtener la lista de archivos en la carpeta
archivos = os.listdir(ArchiDirec)

# Conecta con el cluster de Cassandra
cluster = Cluster(['localhost'])
session = cluster.connect('dataug')
# Preparar la tabla en Cassandra
session.execute('''
    CREATE TABLE IF NOT EXISTS calificaciones (
    periodo TEXT, identificacion TEXT, nivel INT, cod_materia BIGINT, cod_tcalifica INT, docente TEXT, nota DOUBLE,
    PRIMARY KEY (periodo, identificacion, nivel, cod_materia, cod_tcalifica, docente))''')
```

6. Se implementa una condición que implica la lectura del archivo Excel contenido en la carpeta denominada "ArchiDrec". A través de este proceso, se procede a la inserción de los datos correspondientes en un formato compatible con los campos de Cassandra, específicamente con respecto a la clave primaria "COD_TCALIFICA".

```
for archivo in archivos:
    print(ArchiDirec + archivo)
    # Lee el archivo Excel
    df = pd.read_excel( ArchiDirec + archivo)
    # Insertar datos en Cassandra
    for index, row in df.iterrows():
        # Obtener los datos con el formato acorde a los campo en cassandra
        tipoCal=row['COD_TCALIFICA']
        if (tipoCal==151615  or tipoCal== 151635 or tipoCal== 151625 or tipoCal== 151633):
            addrow =convertird(row['PERIODO'],"text")  + "," + convertird(row['IDENTIFICACION'],"text") + "," + convertird(re.search(r'\d+',row['GRUPO'][9:11]).group(),"int") \
                + "," + convertird(row['COD_MATERIA'],"bigint") + "," + convertird(row['COD_TCALIFICA'],"int") + "," + convertird(row['DOCENTE'],"text")[:10] + "'," + convertird(row['NOTA'],"flo
            #print(addrow)
            session.execute('INSERT INTO calificaciones (periodo,identificacion, nivel, cod_materia, cod_tcalifica,docente,nota) VALUES (' + addrow +')')
```

7. Se procede a la carga de los datos correspondientes al ciclo actual en el sistema de gestión de base de datos Cassandra, con el propósito de realizar predicciones. Simultáneamente, se lleva a cabo la preparación de la tabla denominada "calificacionesT" en Cassandra, configurando sus respectivos campos de manera acorde a los requisitos del contexto.

```
# Ruta de datos=carpeta actual\\data\\califTest : Aquí están los datos del ciclo actual para la predicci{on
ArchiDirec=os.getcwd()+"\\data\\califTest\\"
# Obtener la lista de archivos en la carpeta
archivos = os.listdir(ArchiDirec)

# Preparar la tabla en Cassandra
session.execute('''
    CREATE TABLE IF NOT EXISTS calificacionesT (
    periodo TEXT, identificacion TEXT, nivel INT, cod_materia BIGINT, cod_tcalifica INT, docente TEXT, nota DOUBLE,
    PRIMARY KEY (periodo, identificacion, nivel, cod_materia, cod_tcalifica, docente))''')
```

8. Se ejecuta el procedimiento análogo al anterior, empleando en esta ocasión los datos correspondientes al ciclo actual.

```
for archivo in archivos:
    print(ArchiDirec + archivo)
    # Lee el archivo Excel
    df = pd.read_excel( ArchiDirec + archivo)
    # Insertar datos en Cassandra
    for index, row in df.iterrows():
        # Obtener los datos con el formato acorde a los campoe en cassandra
        tipoCal=row['COD_TCALIFICA']
        if (tipoCal==151615  or tipoCal== 151635):
            addrow =convertird(row['PERIODO'],"text")  + "," + convertird(row['IDENTIFICACION'],"text") + "," + convertird(re.search(r'\d+',row['GRUPO'][9:11]).group(),"int") \
                + "," + convertird(row['COD_MATERIA'],"bigint") + "," + convertird(row['COD_TCALIFICA'],"int") + "," + convertird(row['DOCENTE'],"text")[:10] + "'," + convertird(row['NOTA'],"flo
            #print(addrow)
            session.execute('INSERT INTO calificacionesT (periodo,identificacion, nivel, cod_materia, cod_tcalifica,docente,nota) VALUES (' + addrow +')')
```

9. Se configura el DataFrame para el proceso de modelado, estableciendo una conexión con el localhost y accediendo a la carpeta "dataug" para transformar los resultados de la consulta en un DataFrame de Pandas. Posteriormente, se cierra la conexión y se procede a la transformación de los datos.

```
cluster = Cluster(['localhost'])
session = cluster.connect('dataug')

query = "SELECT * FROM calificaciones"
result = session.execute(query)

# Convertir los resultados de la consulta a un DataFrame de pandas
df = pd.DataFrame(result)
# Cerrar la conexión
cluster.shutdown()
# Hacer la transformación de datos
DataCalif = df.pivot_table(index=['periodo' , 'nivel', 'cod_materia', 'docente' ,'identificacion'],
                           columns='cod_tcalifica',
                           values='nota',
                           aggfunc='first').reset_index()

# Cambiar nombres de columnas por valores mas amigables
DataCalif.rename(columns={'periodo': 'PERIODO'}, inplace=True)
DataCalif.rename(columns={'nivel': 'NIVEL'}, inplace=True)
DataCalif.rename(columns={'cod_materia': 'COD_MATERIA'}, inplace=True)
DataCalif.rename(columns={'docente': 'DOCENTE'}, inplace=True)
DataCalif.rename(columns={'identificacion': 'IDENTIFICACION'}, inplace=True)
DataCalif.rename(columns={151615: 'ASISTENCIA 1P'}, inplace=True)
DataCalif.rename(columns={151635: 'PROMEDIO P1'}, inplace=True)
DataCalif.rename(columns={151625: 'PROMEDIOS ASISTENCIA'}, inplace=True)
DataCalif.rename(columns={151633: 'PROMEDIO FINAL'}, inplace=True)
```

10. Incorporamos nuevas columnas ("EfiDocen", "EfiEstud", "EfiMater") al conjunto de datos, asignándoles inicialmente el valor None (vacío). Posteriormente, aplicamos la condición de aprobación de materia, llevando a cabo la eliminación de las columnas superfluas.

```
DataCalif['EfiDocen']  = None
DataCalif['EfiEstud']  = None
DataCalif['EfiMater']  = None

# Aplicar la condición de aprobación de materia
DataCalif['ESTADOAP']  = DataCalif.apply(CondicionAprobado, axis = 1)

# Eliminar columnas no necesarias
DataCalif = DataCalif.drop('PROMEDIOS ASISTENCIA', axis=1)
DataCalif = DataCalif.drop('PROMEDIO FINAL', axis=1)
```

11. Preparamos los datos para el proceso de modelado, seleccionando el último periodo registrado. Se procede a realizar el cálculo de la eficiencia de estudiantes, la tasa de aprobación de materias y la evaluación de desempeño docente. Los resultados obtenidos estarán listos para ser exportados y utilizados en el análisis y modelado subsiguientes.

```
# Encontrar el ultimo periodo ingresado
periodos = DataCalif.groupby('PERIODO')['ESTADOAP'].count()
periodos.sort_index(inplace=True)
uperiodo = periodos.index[-1]
#print(uperiodo)

# Calcular eficiencia de aprobación de estudiantes
EfiMatri = DataCalif.groupby('IDENTIFICACION')['ESTADOAP'].count()
EfiAprob = DataCalif.groupby('IDENTIFICACION')['ESTADOAP'].sum()
Efiestud = EfiAprob / EfiMatri
DataCalif['EfiEstud']=(DataCalif.apply(lambda x: Efiestud[x['IDENTIFICACION']], axis=1))
# Calcular eficiencia de aprobación de materia
EfiMatri2 = DataCalif.groupby('COD_MATERIA')['ESTADOAP'].count()
EfiAprob2 = DataCalif.groupby('COD_MATERIA')['ESTADOAP'].sum()
EfiMater = EfiAprob2 / EfiMatri2
DataCalif['EfiMater']=(DataCalif.apply(lambda x: EfiMater[x['COD_MATERIA']], axis=1))
# Calcular eficiencia de aprobación de docentes
EfiMatri3 = DataCalif.groupby('DOCENTE')['ESTADOAP'].count()
EfiAprob3 = DataCalif.groupby('DOCENTE')['ESTADOAP'].sum()
EfiDocen = EfiAprob3 / EfiMatri3
DataCalif['EfiDocen']=(DataCalif.apply(lambda x: EfiDocen[x['DOCENTE']], axis=1))
# exportar los datos listos para analisis y modelado
ArchivoPX = os.getcwd()+"\\data\\datafin.xlsx"
DataCalif.to_excel(ArchivoPX, index=False)
print("Estudiante", Efiestud.head())
print("Materias", EfiMater.head())
print("Docentes",EfiDocen.head())
```

```
Estudiante IDENTIFICACION
1004941322    0.000000
107111510     1.000000
107348385     1.000000
1104419831    1.000000
1104643489    0.978723
Name: ESTADOAP, dtype: float64
Materias COD_MATERIA
1516001    0.738255
1516002    0.772643
1516003    0.518333
1516004    0.891258
1516005    0.918159
Name: ESTADOAP, dtype: float64
Docentes DOCENTE
060337539    0.707941
090688189    0.840491
090792195    0.921569
090833926    0.926829
090956599    0.936270
Name: ESTADOAP, dtype: float64
```

12. Optamos por extraer únicamente los datos correspondientes al último periodo académico, así como los datos relativos al nivel 1. Posteriormente, procedemos a calcular el promedio de eficiencia de los estudiantes, el cual se utilizará como un valor de relleno para aquellos estudiantes recién ingresados que aún no disponen de un promedio de eficiencia asignado.

```
# Promedio de eficiencia para estudiante
DataCalifUP = DataCalif[(DataCalif['PERIODO'] == uperiodo) & (DataCalif['NIVEL'] == 1)]
EfiMatriU = DataCalifUP.groupby(['IDENTIFICACION'])['ESTADOAP'].count()
EfiAprobU = DataCalifUP.groupby(['IDENTIFICACION'])['ESTADOAP'].sum()
EfiestudU = EfiAprobU / EfiMatriU
EfiPPEstN1 = EfiestudU.mean()
EfiMatriU.drop(EfiMatriU.index, inplace=True)
EfiAprobU.drop(EfiAprobU.index, inplace=True)
EfiestudU.drop(EfiestudU.index, inplace=True)
DataCalifUP.drop(DataCalifUP.index, inplace=True)

# Promedio de eficiencia para materias
DataCalifUP = DataCalif[(DataCalif['PERIODO'] == uperiodo)]
EfiMatriU = DataCalifUP.groupby(['COD_MATERIA'])['ESTADOAP'].count()
EfiAprobU = DataCalifUP.groupby(['COD_MATERIA'])['ESTADOAP'].sum()
EfiMatU = EfiAprobU / EfiMatriU
EfiPPMat = EfiMatU.mean()
EfiMatriU.drop(EfiMatriU.index, inplace=True)
EfiAprobU.drop(EfiAprobU.index, inplace=True)
DataCalifUP.drop(DataCalifUP.index, inplace=True)

# Promedio de eficiencia para docentes
DataCalifUP = DataCalif[(DataCalif['PERIODO'] == uperiodo)]
EfiMatriU = DataCalifUP.groupby(['DOCENTE'])['ESTADOAP'].count()
EfiAprobU = DataCalifUP.groupby(['DOCENTE'])['ESTADOAP'].sum()
EfiDocU = EfiAprobU / EfiMatriU
EfiPPDoc = EfiDocU.mean()
EfiMatriU.drop(EfiMatriU.index, inplace=True)
EfiAprobU.drop(EfiAprobU.index, inplace=True)
DataCalifUP.drop(DataCalifUP.index, inplace=True)
```

13. Configuramos los datos para la fase de proyección mediante la conexión correspondiente al localhost con el repositorio denominado "dataug". Acto seguido, transformamos los resultados en un DataFrame de Pandas y llevamos a cabo la necesaria transformación de los datos para su adecuación al proceso de proyección.

```
cluster = Cluster(['localhost'])
session = cluster.connect('dataug')

query = "SELECT * FROM calificacionest"
result = session.execute(query)

# Convertir los resultados de la consulta a un DataFrame de pandas
df = pd.DataFrame(result)

# Cerrar la conexión
cluster.shutdown()

# Hacer la transformación de datos
DataCalifT = df.pivot_table(index=['periodo' , 'nivel', 'cod_materia', 'docente' ,'identificacion'],
                            columns='cod_tcalifica',
                            values='nota',
                            aggfunc='first').reset_index()
```

14. Iniciamos la modificación de los nombres de las columnas con el objetivo de mejorar la claridad y comprensión de los resultados. Posteriormente, realizamos el cálculo de la eficiencia en la aprobación de estudiantes.

```
# Cambiar nombres de columnas por valores mas amigables
DataCalifT.rename(columns={'periodo': 'PERIODO'}, inplace=True)
DataCalifT.rename(columns={'nivel': 'NIVEL'}, inplace=True)
DataCalifT.rename(columns={'cod_materia': 'COD_MATERIA'}, inplace=True)
DataCalifT.rename(columns={'docente': 'DOCENTE'}, inplace=True)
DataCalifT.rename(columns={'identificacion': 'IDENTIFICACION'}, inplace=True)
DataCalifT.rename(columns={151615: 'ASISTENCIA 1P'}, inplace=True)
DataCalifT.rename(columns={151635: 'PROMEDIO P1'}, inplace=True)

# Calcular eficiencia de aprobación de estudiante

#df_actualizado = pd.merge(df, serie_identificacion, left_on='columna1', right_index=True, how='left')
EfiDocen.name = "EfiDocen"
DataCalifT=pd.merge(DataCalifT,EfiDocen, left_on='DOCENTE', right_index=True, how='left')
Efiestud.name = "EfiEstud"
DataCalifT=pd.merge(DataCalifT,Efiestud, left_on='IDENTIFICACION', right_index=True, how='left')
EfiMater.name = "EfiMater"
DataCalifT=pd.merge(DataCalifT,EfiMater, left_on='COD_MATERIA', right_index=True, how='left')
DataCalifT['ESTADOAP']  = None

DataCalifT = DataCalifT.fillna({'EfiDocen': EfiPPDoc, 'EfiEstud': EfiPPEstN1, 'EfiMater': EfiPPMat})
ArchivoPX=os.getcwd()+"\\data\\dataTest.xlsx"
DataCalifT.to_excel(ArchivoPX, index=False)
#print(type(Efiestud))
```

15. En este contexto, se especifica la ubicación de los datos mediante la definición de la ruta de acceso. Es imperativo que dentro del directorio que aloje el programa, exista una estructura organizativa que incluya una carpeta designada como "data". En esta carpeta, se deben alojar dos archivos fundamentales: "datafin.xlsx" para llevar a cabo la construcción del modelo, y "datatest" para la ejecución de predicciones.

```
ArchiDirec=os.getcwd()+"\\data\\" # Ruta de datos
Archivo=ArchiDirec + "datafin.xlsx"
DataCalif=pd.read_excel(Archivo)
```

16. El DataFrame utilizado presenta dimensiones lo suficientemente considerables y abarca un conjunto sustancial de opciones que comprenden tanto aprobaciones como reprobaciones. En una fase inicial, se procede a la identificación y localización de duplicados con base en campos específicos que deben poseer datos únicos. Se opta por seleccionar la primera aparición en caso de duplicidades.

DataCalif[DataDuplicada].count() Se realiza una verificación mediante el recuento de duplicados utilizando el método count(). En caso de que no se detecten duplicados, se omitirá el proceso de eliminación.

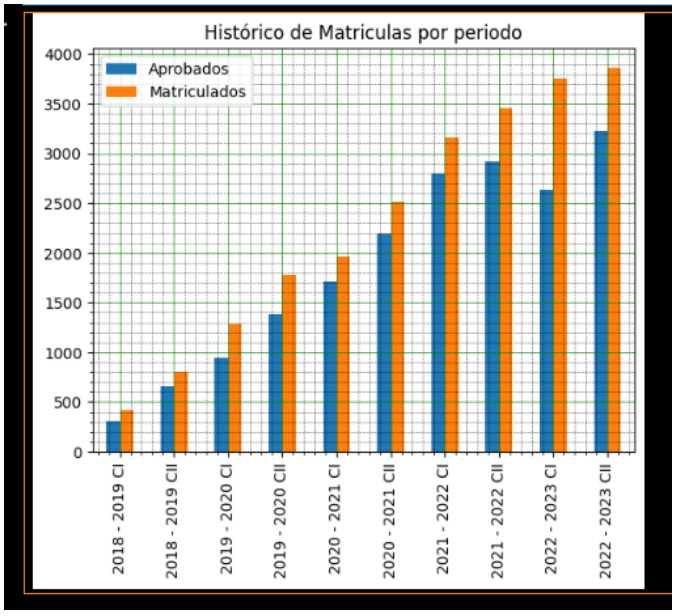
DataCalif = DataCalif.drop_duplicates(subset=["PERIODO", "NUM_EST", "NIVEL", "COD_MAT"], keep="first") En esta etapa, se procede a la eliminación de duplicados identificados previamente, utilizando la función drop_duplicates.

Además, se lleva a cabo una exploración y eliminación de aquellas filas que presentan valores nulos en los campos **"ASISTENCIA 1P" y "PROM P1"**, ya que la integridad de la información es esencial en estos contextos.

```
DataDuplicada = DataCalif.duplicated(subset = ["PERIODO","IDENTIFICACION","NIVEL_MAT","COD_MATERIA","CCDOCEN"], keep = 'first')
DataCalif.drop(DataCalif[(DataCalif["ASISTENCIA 1P"].isnull() )].index, inplace=True)
DataCalif.drop(DataCalif[(DataCalif["PROMEDIO P1"].isnull() )].index, inplace=True)
```

17. Se presenta, desglosado por nivel, la cantidad de estudiantes que han logrado la aprobación y el total de estudiantes matriculados. Estos datos se plasman de manera visual en un gráfico, otorgando así una perspectiva más clara y esclarecedora de la información recopilada. La representación gráfica nos proporciona una proyección visual de los periodos académicos, destacando tanto el número total de estudiantes matriculados como aquellos que han alcanzado el estado de aprobación. Este enfoque visual facilita la interpretación y análisis de los resultados obtenidos.

```
DataPeriodos=np.array(DataCalif["PERIODO"].unique()) # se busca las filas unicas en el campo "PERIODO"
c2=[] # Almacena la cantidad de estudiantes que tienen la condición de Aprobado
c3=[] # Almacena la cantidad total de estudiantes que hay en el PERIODO
DataPeriodos.sort() # ordena los periodos

for DataPeriodo in DataPeriodos: # Se preparan los datos que se mostrarán en la gráfica
    c2.append(DataCalif.apply(lambda x: x['ESTADOAP'] == 1 and x["PERIODO"] == DataPeriodo, axis=1).sum() )
    c3.append(DataCalif.apply(lambda x: x["PERIODO"] == DataPeriodo, axis=1).sum())

DataGrafi=pd.DataFrame({'Aprobados':c2, 'Matriculados':c3}, index=DataPeriodos)
DataGrafi.plot(kind='bar', title='Histórico de Matriculas por periodo')
plt.minorticks_on() # Para activar las grillas en el gráfico
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



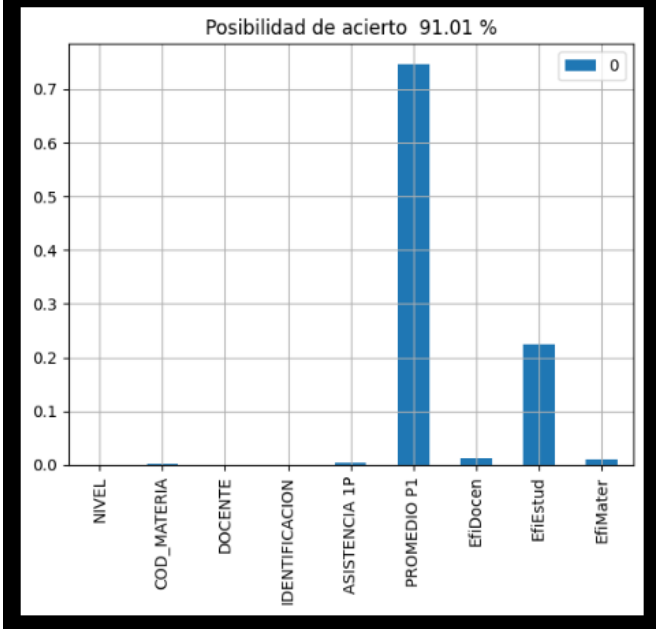
18. Posteriormente, procedemos a la definición de la variable "X", la cual encapsula los promedios obtenidos por los estudiantes en el primer parcial. Simultáneamente, establecemos la variable "Y", que alberga la información relativa al estado final de aprobación de los estudiantes en cada materia. Es crucial llevar a cabo una verificación de la naturaleza numérica de los datos contenidos en nuestro DataFrame para asegurar la coherencia y validez de la información.

```
# Se escoje las columnas que se van a utilizar para el analisis predictivo
# (variables independientes) y se asignan a la variable "x"
x= DataCalif.copy().iloc[:,1:10]

# Se escoje la columna de resultados (variable dependiente)
# se asignan a la variable "y"
y= DataCalif["ESTADOAP"]

# Se divide la muestra en datos de entrenamiento y de prueba#, random_state=42)
# el atributo  random_state= es para que siempre se escoja los mismos datos
X_train, X_test, y_train, y_test = train_test_split(x, y, train_size = 0.75)

# Verificamos que los datos de nuestro DataFrame son numericos
# x = x.astype(float)
# x.info()
```

19. Se realizó a la construcción del árbol de decisiones, configurándolo con una profundidad de 5 niveles para garantizar una modelización precisa y detallada del conjunto de datos. Posteriormente, llevamos a cabo la fase de predicción, la cual nos proporciona resultados expresados en forma de porcentajes, representativos de la probabilidad de acierto en las predicciones realizadas. Vale destacar que la tasa de exactitud alcanzada en este proceso fue notable, alcanzando un impresionante 91,01%. Este indicador subraya la eficacia del modelo en la tarea de realizar predicciones acertadas en relación con el estado de aprobación de los estudiantes en el contexto analizado.

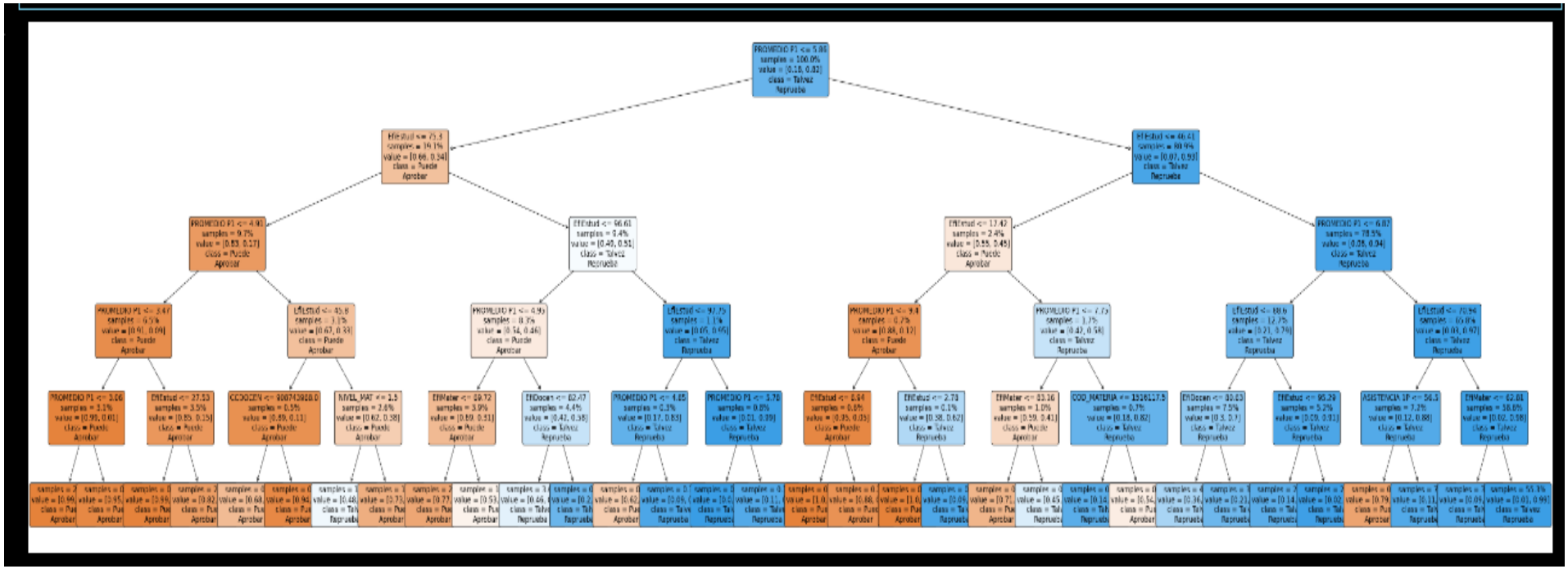
```
# Generamos el arbol de desiciones con una profundidad de 5 niveles
DataArbolDecis = tree.DecisionTreeClassifier(max_depth=5)
DataArbolDecis = DataArbolDecis.fit(X_train, y_train)
# Ejecutamos la prediccion de prueba
yPredicha = DataArbolDecis.predict(X_test)
# accuray_score nos permite conocer el valor porcentual de aciertos en la prediccion
DataPreciPredi=accuracy_score(y_test, yPredicha)
# La matriz de confusion nos enseña cuantas veces una clase fue asiganada correctamente
# precision_score muestra el porcentaje de posibilidad que la prediccion sea correcta
DataMatriPredi=confusion_matrix(y_test, yPredicha,labels = (1, 0))
precision_score(y_test, yPredicha, pos_label = 1)
precision_score(y_test, yPredicha, pos_label = 0)
# Obtenemos aquellas variables que son más importantes en nuestro arbol de decision
DataPrediPosib = pd.DataFrame(DataArbolDecis.feature_importances_, index = x.columns)
# Grafica de los indices que son relevantes para la prediccion
DataPrediPosib.plot(kind="bar",title=f"Posibilidad de acierto  {round(DataPreciPredi*100,2)} %",grid=True)
```

```
<Axes: title={'center': 'Posibilidad de acierto  91.01 %'}>
```



20. Procedemos a la representación visual del árbol de decisiones, configurándolo con una profundidad de 5 niveles para lograr una visualización detallada y estructurada del modelo. La visualización gráfica de este árbol proporciona una perspectiva más profunda y comprensible de las decisiones tomadas por el modelo en cada nivel, permitiendo una interpretación más precisa de su lógica interna.

```
DataGrafoArbol = plt.figure(figsize=(60,15))
_ = tree.plot_tree(DataArbolDecis,max_depth=5,impurity = False, rounded = True, proportion=True,
                   feature_names = x.columns, precision=2,
                   class_names = {0:"Puede\nAprobar",1:"Talvez\nReprueba"},
                   filled = True,
                   fontsize = 14)
```

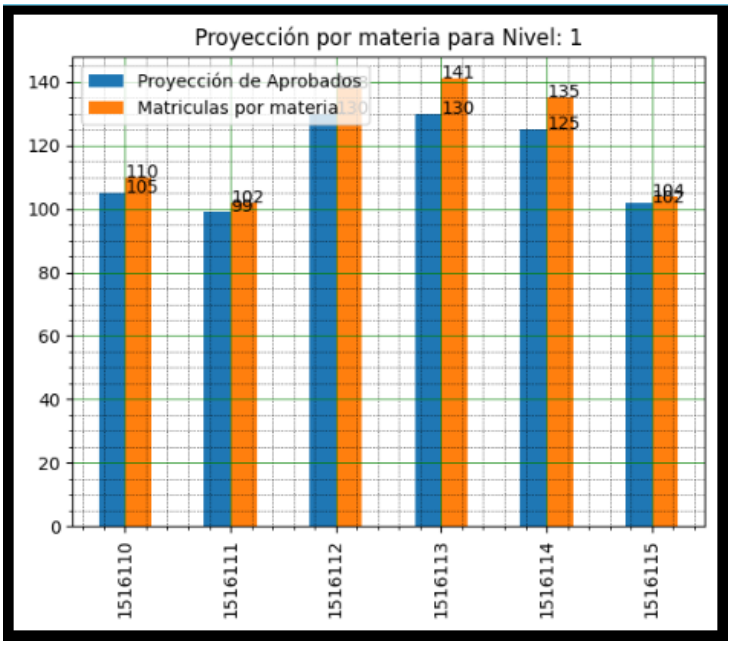


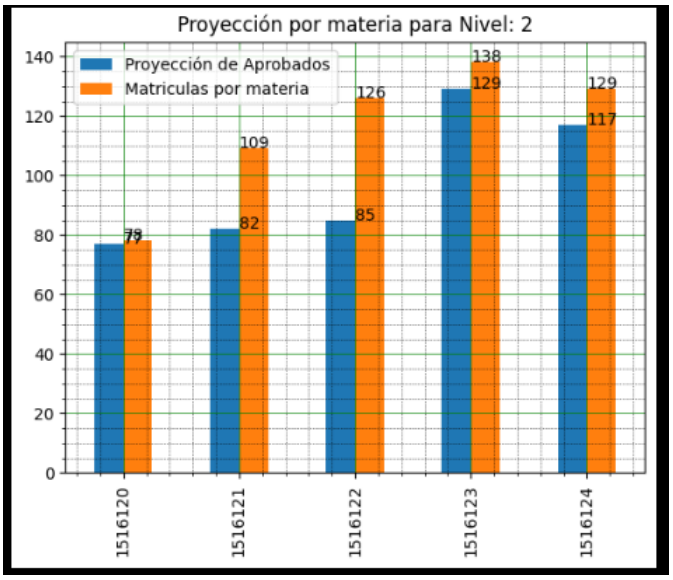
21. En la fase de ejecución de la predicción del modelo, inicializamos el proceso cargando los datos pertinentes mediante la especificación de la ruta correspondiente. Los resultados de la predicción se canalizan y almacenan en dos archivos distintos, cada uno con formatos específicos: "proyección.csv" y "proyección.xlsx". Este enfoque multifacético en la generación de archivos asegura una flexibilidad en la utilización de los resultados, permitiendo su integración eficiente en diversos entornos y plataformas según las necesidades específicas del análisis subsiguiente.

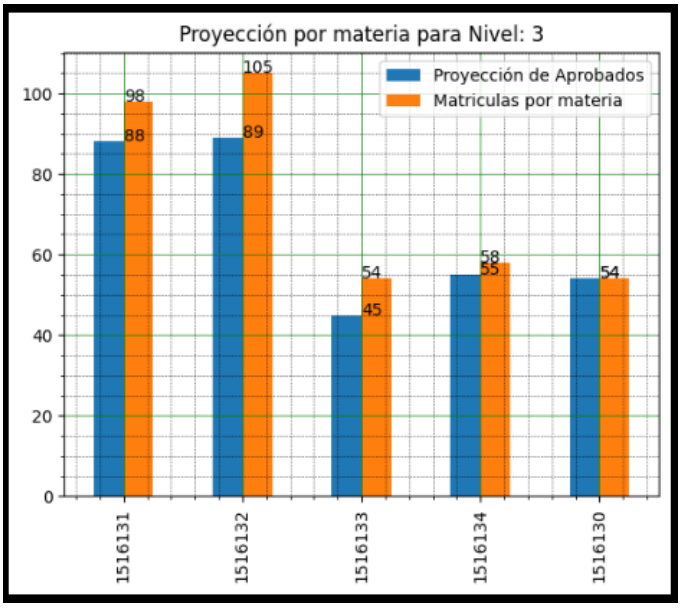
```
def add_value_label(x_list,y_list):
    for i in range(1, len(x_list)+1):
        #print((i,y_list[i-1],y_list[i-1]))
        plt.text(i-1.0,y_list[i-1],y_list[i-1])

# Se carga los datos para hacer las predicciones
ArchiDirec=os.getcwd()+"\\data\\" # Ruta de datos
Archivo=ArchiDirec + "datafin.xlsx"
ArchivoTest=ArchiDirec + "datatest.xlsx"
DataTest=pd.read_excel(ArchivoTest)
#DataTest.drop(DataTest[(DataTest["ASISTENCIA 1P"].isnull() )].index, inplace=True)
#DataTest.drop(DataTest[(DataTest["PROMEDIO P1"].isnull() )].index, inplace=True)
xTest= DataTest.copy().iloc[:,1:]
DataTest["ESTADOAP"] = DataArbolDecis.predict(xTest)
DataTest.to_csv(ArchiDirec + "proyeccion.csv",index=False)
DataTest.to_excel(ArchiDirec + "proyeccion.xlsx",index=False)
```

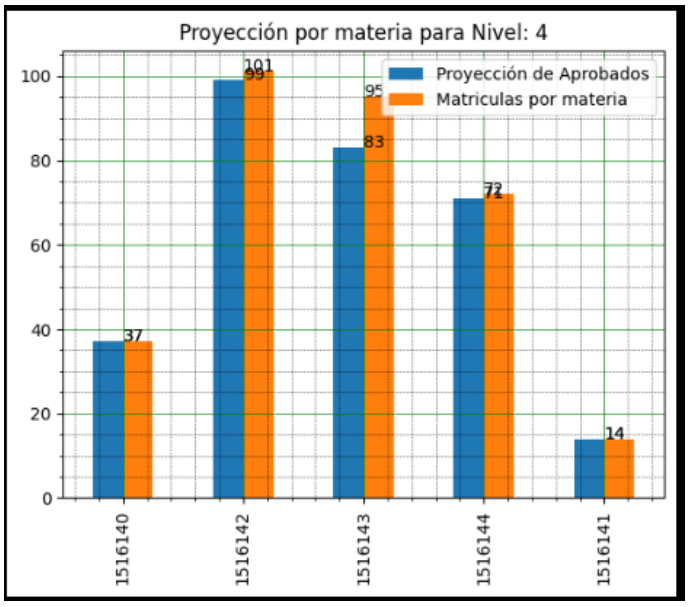
22. Capturamos y preservamos la cantidad de estudiantes que han obtenido una predicción de aprobación, así como la cantidad de estudiantes matriculados, categorizándolos por materias y niveles académicos. Estos datos, meticulosamente archivados, se encuentran listos para ser visualizados en las próximas representaciones gráficas. La utilización de gráficos permitirá una exposición visual precisa y detallada de la distribución de estudiantes aprobados y matriculados, proporcionando una comprensión más profunda de los patrones y tendencias presentes en los diferentes niveles educativos y materias específicas.

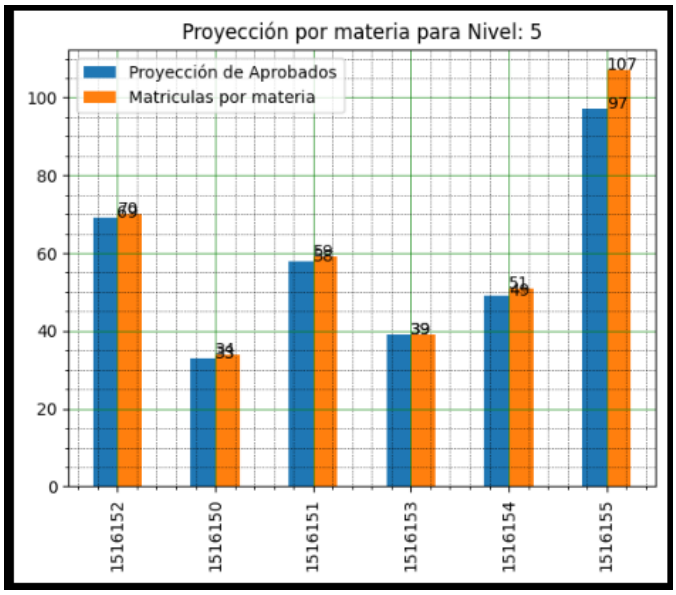
```
for i in range(10):
    DataPrediNiv=DataTest.loc[DataTest["NIVEL_MAT"] == (i+1)]
    DataMaterNiv=np.array(DataPrediNiv["COD_MATERIA"].unique()) # se busca las filas unicas en el campo "COD_MAT"
    c2=[] # Almacena la cantidad de estudiantes que tienen la prediccion de Aprobado
    c3=[] # Almacena la cantidad de estudiantes registrados en la materia
    for DataGrafi01 in DataMaterNiv: # Se preparan los datos que se mostraran en la grafica
        c2.append(DataPrediNiv.apply(lambda x: x['ESTADOAP'] == 1 and x["COD_MATERIA"] == DataGrafi01, axis=1).sum() )
        c3.append(DataPrediNiv.apply(lambda x: x["COD_MATERIA"] == DataGrafi01, axis=1).sum() )

    DataGrafiTest=pd.DataFrame({'Proyección de Aprobados':c2, "Matriculas por materia":c3}, index=DataMaterNiv)
    DataGrafiTest.plot(kind='bar', title=f'Proyección por materia para Nivel: {i+1}')

    add_value_label(DataMaterNiv,c2)
    add_value_label(DataMaterNiv,c3)

    plt.minorticks_on() # Para activar las grillas en el gráfico
    plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
    plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
    plt.show()
```

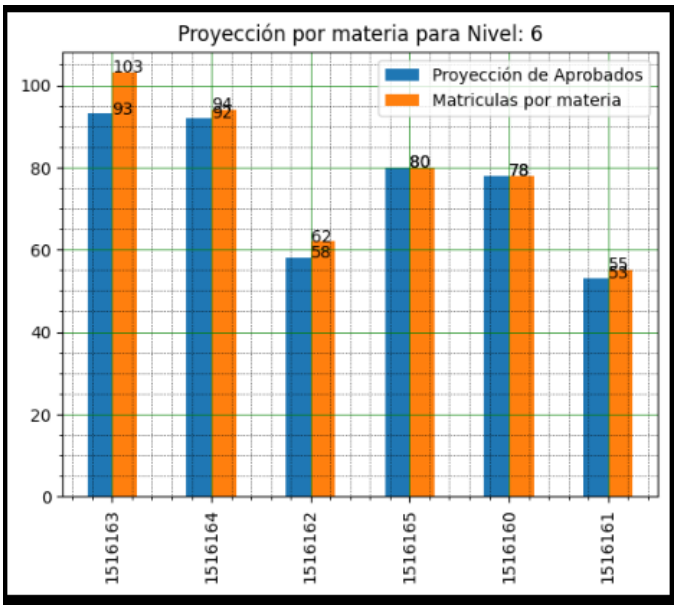

Proyección por materia para Nivel: 1
Proyección de Aprobados
Matriculas por materia
140
120
100
80
60
40
20
0
110
105
102
99
130
141
130
135
125
104
102
1516110
1516111
1516112
1516113
1516114
1516115

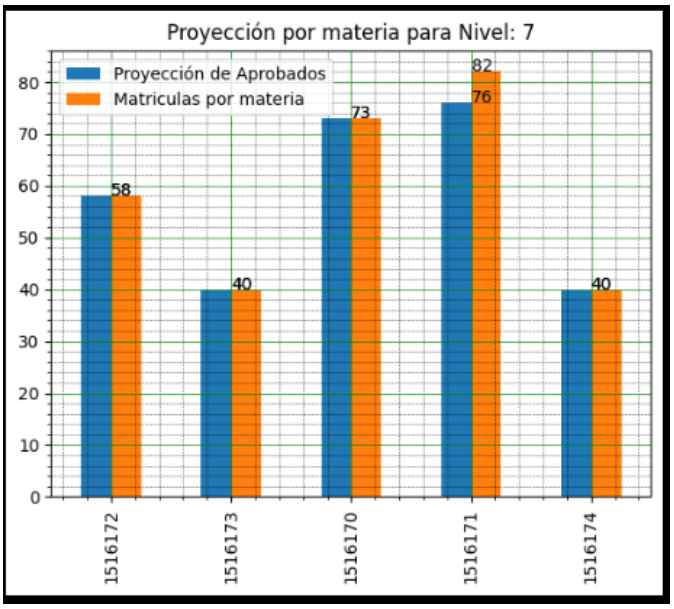

Proyección por materia para Nivel: 2
Proyección de Aprobados
Matriculas por materia
140
120
100
80
60
40
20
0
78
77
109
82
126
85
138
129
129
117
1516120
1516121
1516122
1516123
1516124

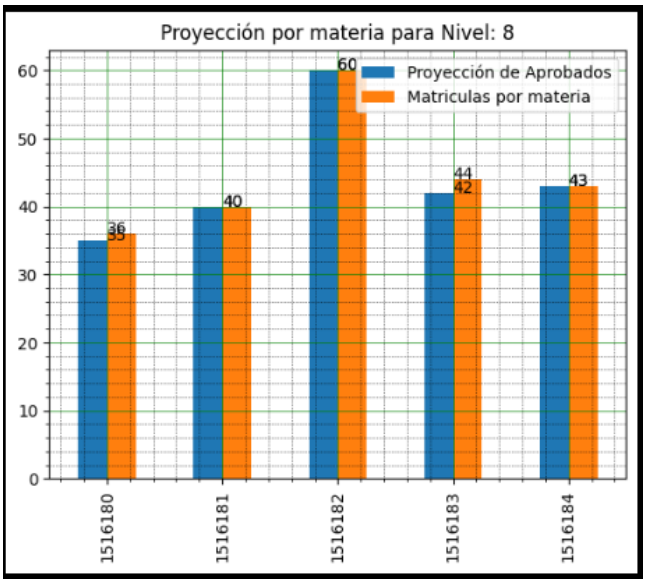

Proyección por materia para Nivel: 3
Proyección de Aprobados
Matriculas por materia
100
80
60
40
20
0
98
88
105
89
54
45
58
55
54
1516131
1516132
1516133
1516134
1516130

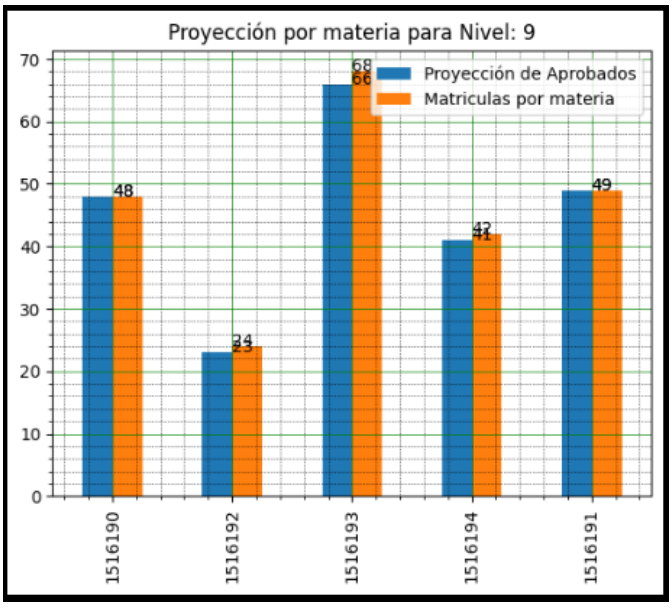

Proyección por materia para Nivel: 4
Proyección de Aprobados
Matriculas por materia
100
80
60
40
20
0
37
101
99
95
83
72
71
14
1516140
1516142
1516143
1516144
1516141

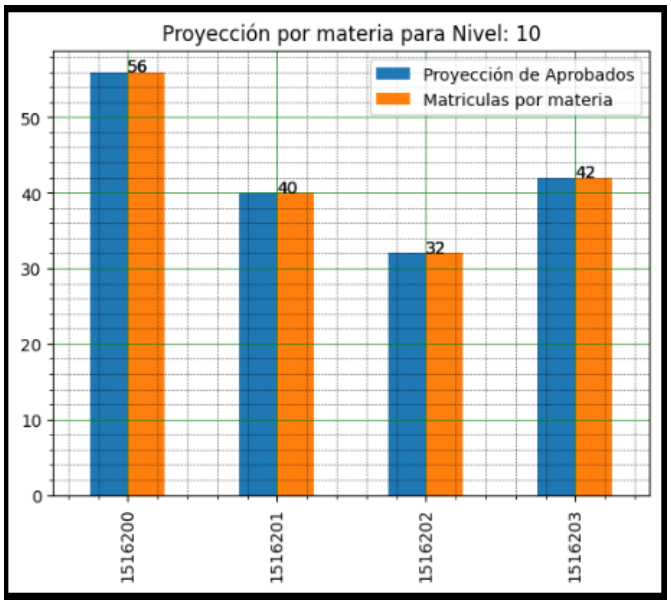

Proyección por materia para Nivel: 5
Proyección de Aprobados
Matriculas por materia
100
80
60
40
20
0
70
69
34
33
58
56
39
51
49
107
97
1516152
1516150
1516151
1516153
1516154
1516155


Proyección por materia para Nivel: 6
Proyección de Aprobados
Matriculas por materia
100
80
60
40
20
0
103
93
94
92
62
58
80
78
55
53
1516163
1516164
1516162
1516165
1516160
1516161


Proyección por materia para Nivel: 7
Proyección de Aprobados
Matriculas por materia
0
10
20
30
40
50
60
70
80
58
40
73
76
82
40
1516172
1516173
1516170
1516171
1516174


Proyección por materia para Nivel: 8
Proyección de Aprobados
Matriculas por materia
0
10
20
30
40
50
60
35
36
40
42
44
60
43
1516180
1516181
1516182
1516183
1516184


Proyección por materia para Nivel: 9
Proyección de Aprobados
Matriculas por materia
0
10
20
30
40
50
60
70
48
48
23
24
66
68
41
42
49
49
1516190
1516192
1516193
1516194
1516191


Proyección por materia para Nivel: 10
Proyección de Aprobados
Matriculas por materia
56
40
32
42
0
10
20
30
40
50
1516200
1516201
1516202
1516203

MODELO DE PRUEBA I: RANDOM FOREST

Se ejecutó el procedimiento similar al del modelo precedente, incluyendo la importación de las bibliotecas esenciales, la lectura de los datos destinados al análisis, la depuración de duplicados con el propósito de mejorar la interpretabilidad de los resultados, el entrenamiento del modelo con el conjunto de datos, la realización de predicciones en el conjunto de prueba y el cálculo de la métrica de precisión del modelo. Los resultados revelaron una precisión del 0,88%, la cual es considerablemente baja para los estándares requeridos por este proyecto de tesis. En consecuencia, se determinó interrumpir el proceso, ya que se busca alcanzar una precisión de aciertos más sustancial para cumplir con los objetivos establecidos.

```
# Importar las bibliotecas necesarias
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

ArchiDirec=os.getcwd()+"\\data\\" # Ruta de datos=carpeta actual\\data\\
Archivo=ArchiDirec + "datafin.xlsx" # Archivo con la información en formato xlsx
DataCalif = pd.read_excel(ArchiDirec + "datafin.xlsx")
# Cargar los datos de prueba desde un archivo CSV
data_test = pd.read_excel(ArchiDirec + "datatest.xlsx")
```

```
# El DataFrame usado es lo suficientemente grande y contiene suficientes opciones de aprobado o reprobado
# Localizamos los duplicados segun los campos que deben dar datos únicos, escogiendo la primera aparición de ser el
DataDuplicada = DataCalif.duplicated(subset = ["PERIODO","IDENTIFICACION","NIVEL","COD_MATERIA","DOCENTE"], keep =

DataCalif.drop(DataCalif[(DataCalif["ASISTENCIA 1P"].isnull() )].index, inplace=True)
DataCalif.drop(DataCalif[(DataCalif["PROMEDIO P1"].isnull() )].index, inplace=True)


# Se escoje las columnas que se van a utilizar para el analisis predictivo
# (variables independientes) y se asignan a la variable "x"
x= DataCalif.copy().iloc[:,1:10]
```

```
# Se escoje la columna de resultados (variable dependiente)
# se asignan a la variable "y"
y= DataCalif["ESTADOAP"]



# Se divide la muestra en datos de entrenamiento y de prueba#, random_state=42)
# el atributo  random_state= es para que siempre se escoja los mismos datos
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.75)
```

```
# Crear un modelo Random Forest
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Entrenar el modelo con los datos de entrenamiento
model.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred = model.predict(X_test)

# Calcular la precisión del modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo Random Forest: {accuracy:.2f}")

# Mostrar un informe de clasificación
report = classification_report(y_test, y_pred)
print("Informe de clasificación:\n", report)
```

```
[12]
...  Precisión del modelo Random Forest: 0.88
     Informe de clasificación:
                   precision    recall  f1-score   support

                0       0.73      0.60      0.66      3178
                1       0.91      0.95      0.93     14064

         accuracy                           0.88     17242
        macro avg       0.82      0.77      0.79     17242
     weighted avg       0.88      0.88      0.88     17242
```

MODELO DE PRUEBA II: REGRESIÓN LOGÍSTICA

En este modelo se llevaron a cabo los procedimientos análogos a los modelos anteriores, obteniendo una precisión del modelo del 0,89%, la cual se considera altamente insuficiente para cumplir con los requisitos establecidos en el marco del proyecto de tesis.

```python
# Importar las bibliotecas necesarias
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report

#x= DataCalif.copy().iloc[:,1:10]
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Escalar las características (opcional pero generalmente recomendado)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Crear un modelo de Regresión Logística
model = LogisticRegression()

# Entrenar el modelo
model.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred = model.predict(X_test)

# Calcular la precisión del modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo de Regresión Logística: {accuracy:.2f}")

# Mostrar un informe de clasificación
report = classification_report(y_test, y_pred)
print("Informe de clasificación:\n", report)
```

```
Precisión del modelo de Regresión Logística: 0.89
Informe de clasificación:
              precision    recall  f1-score   support

           0       0.76      0.55      0.64      3178
           1       0.90      0.96      0.93     14064

    accuracy                           0.89     17242
   macro avg       0.83      0.76      0.79     17242
weighted avg       0.88      0.89      0.88     17242
```