PREDICCIÓN DEL DESEMPEÑO ACADÉMICO ESTUDIANTIL

Mariangelica De Sousa
Data Science II
Proyecto Final

TEMAS

- Introducción
- Contexto analítico e hipótesis
- Glosario de variables
- Limpieza y transformación de datos
- Análisis exploratorio
- Modelado
- Evaluación y comparación de resultados
- Conclusión

INTRODUCCIÓN

- Este proyecto investiga cómo factores como los hábitos de estudio, asistencia, apoyo familiar, acceso a recursos y estilo de vida afectan el desempeño académico. Se realiza un análisis exploratorio mediante visualizaciones y transformaciones de datos para identificar patrones y correlaciones. Posteriormente, se desarrolla un modelo de regresión para predecir el puntaje final de exámenes, que busca proporcionar información clave para optimizar estrategias educativas y mejorar el rendimiento.

CONTEXTO ANALÍTICO E HIPÓTESIS

Se cree que el rendimiento académico depende no solo de horas de estudio sino de otros factores como los hábitos de estudio, asistencia, apoyo familiar y acceso a recursos. Usaremos modelos de regresión para evaluar el impacto de estas variables en las calificaciones y optimizar estrategias educativas.

Se espera que un mayor tiempo de estudio y apoyo familiar mejoren el desempeño, mientras que la falta de sueño y la ansiedad lo perjudiquen, permitiéndonos desarrollar modelos predictivos sólidos

GLOSARIO DE VARIABLES

- **Hours_Studied:** Horas de estudio semanales (numérico)
- **Attendance:** Porcentaje de asistencia (numérico)
- **Parental_Involvement:** Participación parental (Bajo/Medio/Alto, categórico)
- **Access_to_Resources:** Disponibilidad de recursos educativos (Bajo/Medio/Alto, categórico)
- **Extracurricular_Activities:** Participación en actividades extracurriculares (Sí/No, booleano)
- **Sleep_Hours:** Horas de sueño promedio (numérico)
- **Previous_Scores:** Calificaciones anteriores (numérico)
- **Motivation_Level:** Nivel de motivación (Bajo/Medio/Alto, categórico)
- **Internet_Access:** Acceso a Internet (Sí/No, booleano)
- **Tutoring_Sessions:** Sesiones de tutoría mensuales (numérico)
- **Family_Income:** Ingresos familiares (Bajo/Medio/Alto, categórico)
- **Teacher_Quality:** Calidad docente (Baja/Media/Alta, categórico)
- **School_Type:** Tipo de escuela (Pública/Privada, categórico)
- **Peer_Influence:** Influencia de compañeros (Positiva/Neutral/Negativa, categórico)
- **Physical_Activity:** Horas de actividad física semanales (numérico)
- **Learning_Disabilities:** Discapacidades de aprendizaje (Sí/No, booleano)
- **Parental_Education_Level:** Nivel educativo de los padres (Secundaria/Universidad/Postgrado, categórico)
- **Distance_from_Home:** Distancia a la escuela (Cerca/Moderada/Lejos, categórico)
- **Gender:** Género (Masculino/Femenino, categórico)
- **Exam_Score:** Puntaje final del examen (numérico)

LIMPIEZA Y TRANSFORMACIÓN DE DATOS

- **Dataset:** 6607 filas, 20 columnas

- **Valores nulos identificados:**
  - Parental_Education_Level: 90
  - Distance_from_Home: 67
  - Teacher_Quality: 78

```
# Reemplazar valores nulos con la moda
missing_col = ['Parental_Education_Level', 'Distance_from_Home', 'Teacher_Quality']

for col in missing_col:
    mode_value = df[col].mode()[0]
    df[col] = df[col].fillna(mode_value)

    #Comprobamos que nos hayamos deshecho de los valores
df.isnull().sum()
```

- **Estrategia:** Para evitar borrar esas filas, reemplazamos los nulos con la moda, teniendo en cuenta que al ser un muy bajo porcentaje de nulos en relación a la cantidad de filas no debería afectar negativamente al modelo.

- **Duplicados:** No se encontraron

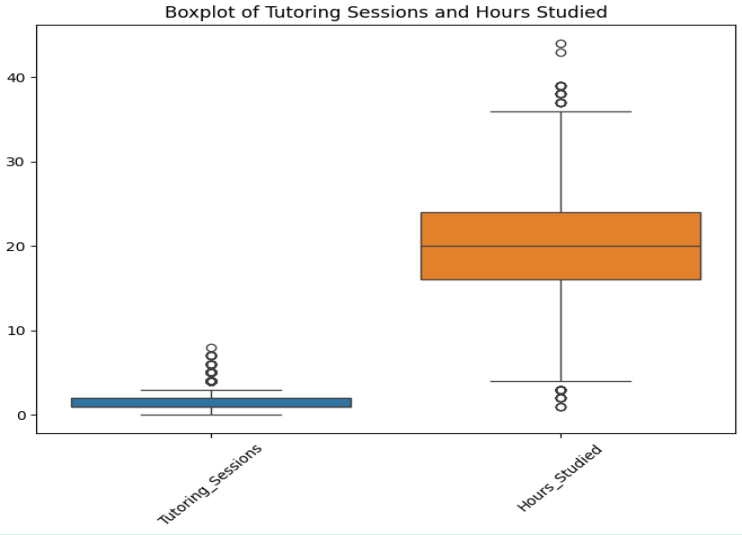
```
#Revisamos si hay duplicados
df.duplicated().sum()
```

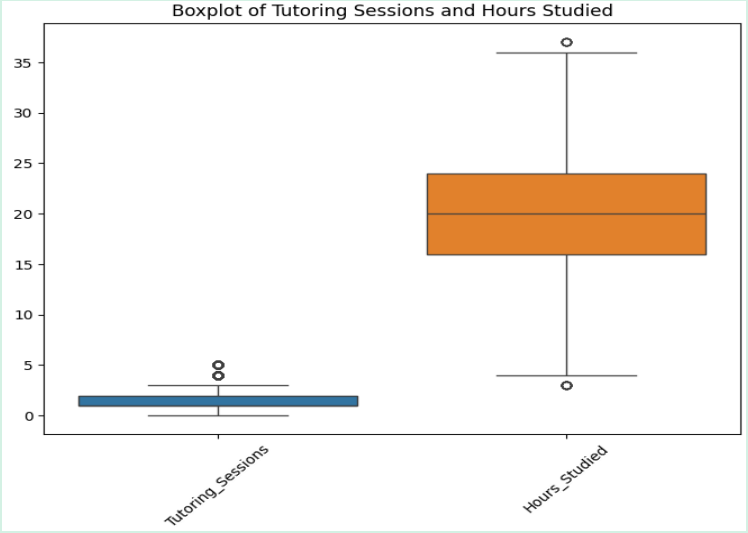
```
np.int64(0)
```

LIMPIEZA Y TRANSFORMACIÓN DE DATOS

- **Detección de Outliers:** Mediante boxplot se identificaron outliers en tres variables. Uno de estos casos correspondía a 'Exam_Score', el cual fue conservado, ya que sus valores oscilan entre 0 y 100. Aunque se observaron algunas calificaciones muy altas o muy bajas, estos datos son reales y se encuentran dentro del rango estipulado
- **Estrategia:** Se aplicó Z-score a las variables numéricas con outliers, excluyendo Exam_Score (variable objetivo).

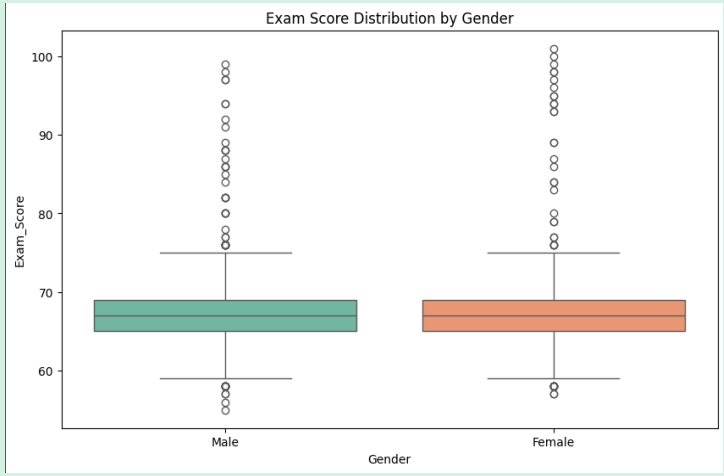
```python
#Voy a usar Z-score para detectar y borrar outliers excluyendo 'Exam_Score'
numerical_features = df.select_dtypes(include=['int64', 'float64']).drop(columns=['Exam_Score'])

#Outliers
z = np.abs((numerical_features - numerical_features.mean()) / numerical_features.std())
threshold = 3
df = df[(z < threshold).all(axis=1)]
```





LIMPIEZA Y TRANSFORMACIÓN DE DATOS

- **Eliminación de columnas:** Se descartó la columna *Gender* tras confirmar (mediante análisis visual y estadístico) que no presentaba diferencias significativas en el desempeño académico.



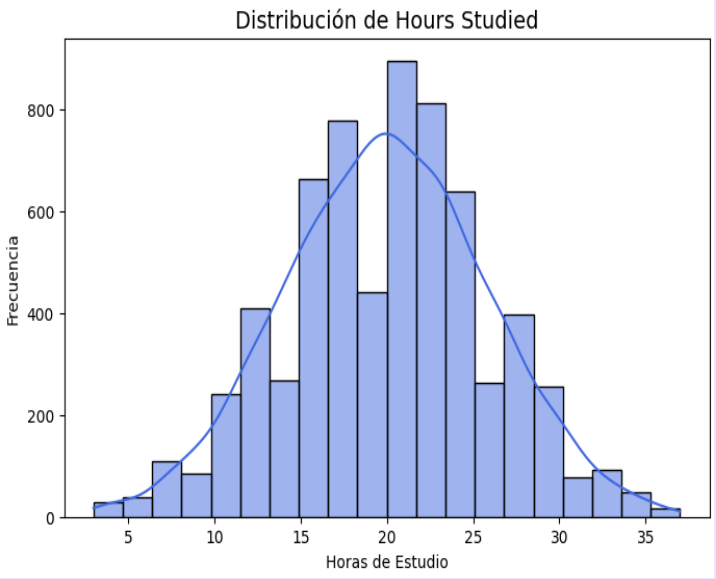

```python
# 1. Eliminar la columna "Gender"
df = df.drop(columns=['Gender_Male'])
```

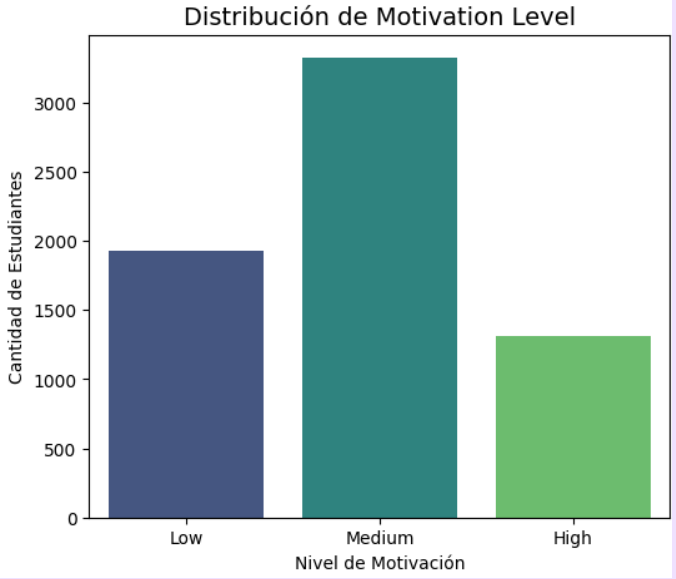
- **Conversión de variables categóricas:** Se aplicó One-Hot Encoding para transformar todas las variables categóricas en datos numéricos, facilitando el entrenamiento y la interpretación de los modelos.

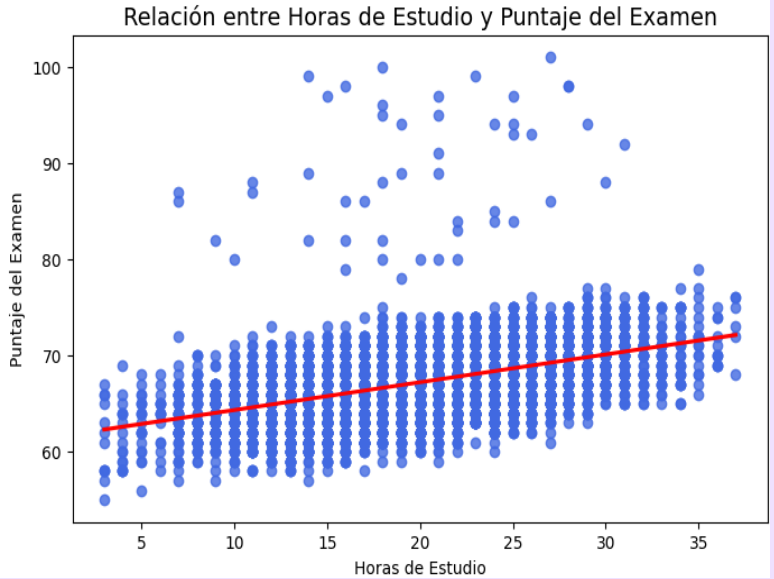
```python
# Lista de columnas categóricas
categorical_columns = [
    'Parental_Involvement', 'Access_to_Resources', 'Motivation_Level',
    'Family_Income', 'Teacher_Quality', 'Peer_Influence',
    'Parental_Education_Level', 'Distance_from_Home',
    'Extracurricular_Activities', 'Internet_Access',
    'School_Type', 'Learning_Disabilities', 'Gender'
]

# Aplicar One-Hot Encoding
df = pd.get_dummies(df, columns=categorical_columns, drop_first=True)
```

ANÁLISIS EXPLORATORIO (EDA)
SIMPLE

- **Horas de Estudio:** La mayoría de los alumnos estudia entre 20 y 23 horas semanales, mostrando un comportamiento consistente.
- **Nivel de Motivación:** La mayoría presenta un nivel de motivación "medio" antes de los exámenes, siendo los estudiantes con alta motivación una minoría.
- **Relación Estudio vs. Calificaciones:** Aunque algunos estudiantes obtienen buenas calificaciones con menos de 15 horas de estudio, se observa una tendencia positiva: a mayor cantidad de estudio, mejores resultados académicos.

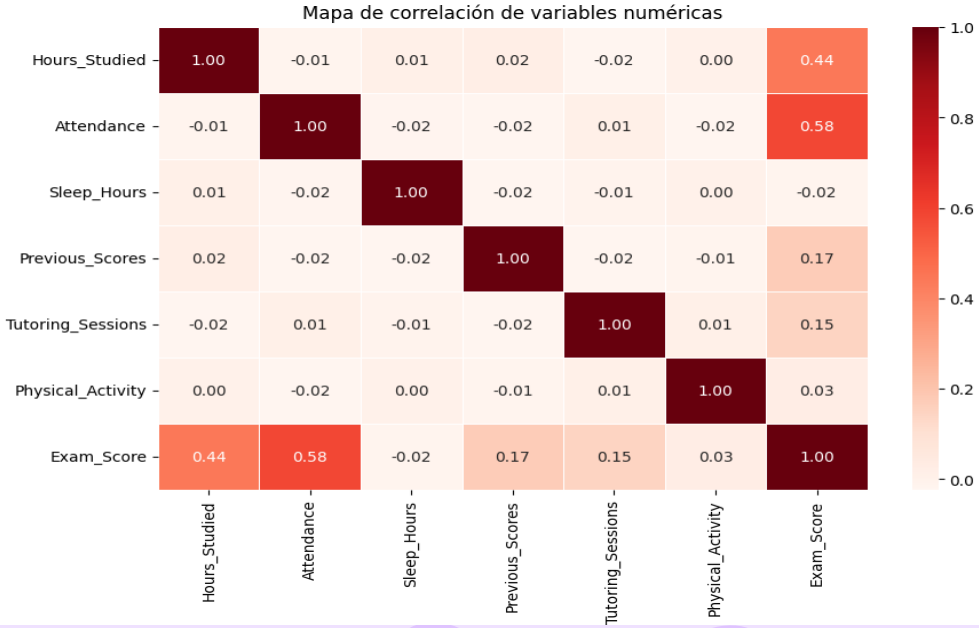




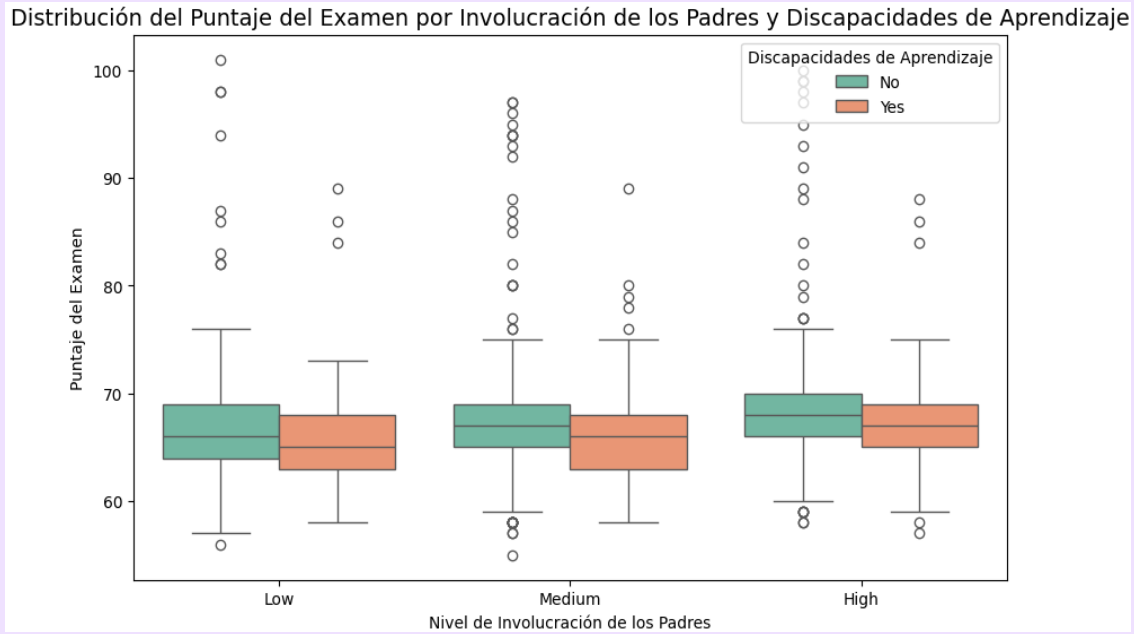


ANÁLISIS EXPLORATORIO (EDA)
COMPLEJO

- **Correlación entre Variables:** El mapa de correlación indica que, si bien ninguna variable tiene una correlación perfecta, las horas de estudio y la asistencia a clases son las que muestran una relación positiva más fuerte con las calificaciones. Por otro lado, la actividad física y las horas de sueño no parecen tener impacto significativo.



- **Influencia del Involucramiento Parental:** El gráfico de boxplot revela que, aunque los estudiantes con discapacidades de aprendizaje tienden a tener calificaciones ligeramente inferiores, en ambos grupos se evidencia que un mayor involucramiento de los padres se asocia a promedios más altos y a una mayor cantidad de estudiantes con calificaciones sobresalientes.



RESULTADOS DEL ANÁLISIS EXPLORATORIO

- El análisis reveló que las horas de estudio y la asistencia a clases se correlacionan positivamente con las calificaciones, lo que confirma que un mayor esfuerzo se traduce en mejores resultados. Además, se observó que la motivación, el sueño, la actividad física y el género no influyen de forma determinante en el rendimiento.

- En cuanto a recursos, el acceso a internet y un nivel medio de recursos se asocian a mejores calificaciones, y el involucramiento parental resulta clave, especialmente en estudiantes con discapacidades de aprendizaje.

- Estos hallazgos consolidan una base sólida para el desarrollo de un modelo de regresión que optimice estrategias educativas.

(En esta presentación se muestran ejemplos representativos, aunque no se incluyen todos los gráficos generados durante el EDA.)

MODELADO

- **Modelos Utilizados:**
  - Regresión Lineal
  - Lasso
  - Random Forest
  - XGBoost
  - SVR

- **Enfoque:**
  - Cada modelo se implementó mediante un pipeline que incluye la estandarización de datos con *StandardScaler* y el entrenamiento del modelo.
  - Se empleó validación cruzada (K-Fold, 5 pliegues) para evaluar el rendimiento usando las métricas R^2, MAE y RMSE.

- **Ajuste de Hiperparámetros**
  - Para algunos modelos, como SVR y XGBoost, se empleó **GridSearchCV** para evaluar exhaustivamente todas las combinaciones posibles de un conjunto predefinido de valores para cada hiperparámetro.

MODELADO

- **Regresión Lineal**

```
X = df.drop("Exam_Score", axis=1)
y = df["Exam_Score"]

# Definir K-Fold (5 pliegues)
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# Creamos el modelo de Regresión Lineal, que incluye el escalado de variables.
modelo_RL = Pipeline([
    ("scaler", StandardScaler()),
    ("lr", LinearRegression())
])

# Definir las métricas de evaluación:
scoring = {
    "R2": "r2",
    "MAE": "neg_mean_absolute_error",
    "RMSE": "neg_root_mean_squared_error"
}

# Realizar la validación cruzada
cv_results = cross_validate(modelo_RL, X, y, cv=kfold, scoring=scoring)

# Calcular los promedios de las métricas
mean_r2 = np.mean(cv_results["test_R2"])
mean_mae = -np.mean(cv_results["test_MAE"])
mean_rmse = -np.mean(cv_results["test_RMSE"])

print("Evaluación de Regresión Lineal:")
print(f"Mean R²: {mean_r2:.3f}")
print(f"Mean MAE: {mean_mae:.3f}")
print(f"Mean RMSE: {mean_rmse:.3f}")
```

```
Evaluación de Regresión Lineal:
Mean R²: 0.729
Mean MAE: 0.483
Mean RMSE: 1.993
```

- **Lasso**

```
from sklearn.linear_model import Lasso, LassoCV
X = df.drop("Exam_Score", axis=1)
y = df["Exam_Score"]

# K-Fold con 5 pliegues
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# Primero, encontraremos el mejor valor de alpha usando LassoCV
# Escalamos los datos temporalmente para LassoCV
scaler_temp = StandardScaler()
X_scaled = scaler_temp.fit_transform(X)

lasso_cv = LassoCV(cv=kfold, random_state=42)
lasso_cv.fit(X_scaled, y)
best_alpha = lasso_cv.alpha_
print("Mejor valor de alpha encontrado por LassoCV:", best_alpha)

# Crear el pipeline para el modelo Lasso con el alpha óptimo encontrado
modelo_lasso = Pipeline([
    ("scaler", StandardScaler()),
    ("lasso", Lasso(alpha=best_alpha))
])

# Métricas de evaluación:
scoring = {
    "R2": "r2",
    "MAE": "neg_mean_absolute_error",
    "RMSE": "neg_root_mean_squared_error"
}

# Validación cruzada
cv_results = cross_validate(modelo_lasso, X, y, cv=kfold, scoring=scoring)

mean_r2 = np.mean(cv_results["test_R2"])
mean_mae = -np.mean(cv_results["test_MAE"])
mean_rmse = -np.mean(cv_results["test_RMSE"])
```

```
Mejor valor de alpha encontrado por LassoCV: 0.0022488119939167595
Evaluación de Lasso:
Mean R²: 0.729
Mean MAE: 0.483
Mean RMSE: 1.993
```

MODELADO

- **Random Forest**

```python
X = df.drop("Exam_Score", axis=1)
y = df["Exam_Score"]

# Definir K-Fold (5 pliegues)
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# Creamos el modelo Random Forest
modelo_RF = Pipeline([
    ("scaler", StandardScaler()),
    ("rf", RandomForestRegressor(random_state=42))
])

# Definir las métricas de evaluación:
scoring = {
    "R2": "r2",
    "MAE": "neg_mean_absolute_error",
    "RMSE": "neg_root_mean_squared_error"
}

# Realizar la validación cruzada
cv_results = cross_validate(modelo_RF, X, y, cv=kfold, scoring=scoring)

# Calcular los promedios de las métricas
mean_r2 = np.mean(cv_results["test_R2"])
mean_mae = -np.mean(cv_results["test_MAE"])
mean_rmse = -np.mean(cv_results["test_RMSE"])
```

- **XGBoost (inicial)**

```python
X = df.drop("Exam_Score", axis=1)
y = df["Exam_Score"]

# Definir K-Fold (5 pliegues)
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# Crear el modelo de XGBRegressor
modelo_XGB = Pipeline([
    ("scaler", StandardScaler()),
    ("xgb", XGBRegressor(random_state=42, verbosity=0))
])

# Definir las métricas de evaluación:
scoring = {
    "R2": "r2",
    "MAE": "neg_mean_absolute_error",
    "RMSE": "neg_root_mean_squared_error"
}

# Realizar la validación cruzada
cv_results = cross_validate(modelo_XGB, X, y, cv=kfold, scoring=scoring)

# Calcular los promedios de las métricas
mean_r2 = np.mean(cv_results["test_R2"])
mean_mae = -np.mean(cv_results["test_MAE"])
mean_rmse = -np.mean(cv_results["test_RMSE"])
```

```
Evaluación de XGBoost:
Mean R²: 0.640
Mean MAE: 0.990
Mean RMSE: 2.305
```

- **XGBoost (mejorado)**

```python
X = df.drop("Exam_Score", axis=1)
y = df["Exam_Score"]
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# Crear el pipelinecon los parámetros óptimos encontrados
modelo_XGB_final = Pipeline([
    ("scaler", StandardScaler()),
    ("xgb", XGBRegressor(
        colsample_bytree=0.8,
        learning_rate=0.1,
        max_depth=3,
        n_estimators=200,
        reg_alpha=1,
        reg_lambda=2,
        subsample=0.8,
        random_state=42,
        verbosity=0
    ))
])

scoring = {
    "R2": "r2",
    "MAE": "neg_mean_absolute_error",
    "RMSE": "neg_root_mean_squared_error"
}

cv_results_final = cross_validate(modelo_XGB_final, X, y, cv=kfold, scoring=scoring)

mean_r2_final = np.mean(cv_results_final["test_R2"])
mean_mae_final = -np.mean(cv_results_final["test_MAE"])
mean_rmse_final = -np.mean(cv_results_final["test_RMSE"])
```

```
Evaluación final de XGBoost con hiperparámetros óptimos:
Mean R²: 0.713
Mean MAE: 0.655
Mean RMSE: 2.055
```

MODELADO

- **Support Vector Regression (SVR)**
- **GridSearch**
- **SVR Mejorado**

```python
X = df.drop("Exam_Score", axis=1)
y = df["Exam_Score"]

# Definir K-Fold con 5 pliegues
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# Crear el modelo SVR
modelo_SVR = Pipeline([
    ("scaler", StandardScaler()),
    ("svr", SVR())
])

# Definir las métricas de evaluación:
scoring = {
    "R2": "r2",
    "MAE": "neg_mean_absolute_error",
    "RMSE": "neg_root_mean_squared_error"
}

# Realizar la validación cruzada con los 5 pliegues
cv_results = cross_validate(modelo_SVR, X, y, cv=kfold, scoring=scoring)

# Calcular los promedios de las métricas
mean_r2 = np.mean(cv_results["test_R2"])
mean_mae = -np.mean(cv_results["test_MAE"])
mean_rmse = -np.mean(cv_results["test_RMSE"])
```

```
Evaluación de SVR:
Mean R²: 0.719
Mean MAE: 0.549
Mean RMSE: 2.033
```

```python
from sklearn.model_selection import GridSearchCV

X = df.drop("Exam_Score", axis=1)
y = df["Exam_Score"]

# Crear SVR
pipeline_svr = Pipeline([
    ("scaler", StandardScaler()),
    ("svr", SVR())
])

# Definir grid de parámetros
param_grid = {
    "svr__C": [0.1, 1, 10, 100],
    "svr__epsilon": [0.1, 0.2, 0.5, 1],
    "svr__kernel": ["rbf", "linear"],
    "svr__gamma": ["scale", "auto"]
}

# Definir KFold (5 pliegues)
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# Configurar GridSearchCV
grid_search = GridSearchCV(pipeline_svr, param_grid, cv=kfold, scoring="r2", n_jobs=-1)
grid_search.fit(X, y)
```

```python
X = df.drop("Exam_Score", axis=1)
y = df["Exam_Score"]

# K-Fold (5 pliegues)
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# Entrenar SVR utilizando los parámetros óptimos encontrados
modelo_SVR_final = Pipeline([
    ("scaler", StandardScaler()),
    ("svr", SVR(C=100, epsilon=0.5, gamma="scale", kernel="linear"))
])

# Definir las métricas de evaluación:
scoring = {
    "R2": "r2",
    "MAE": "neg_mean_absolute_error",
    "RMSE": "neg_root_mean_squared_error"
}

# Validación cruzada
cv_results_final = cross_validate(modelo_SVR_final, X, y, cv=kfold, scoring=scoring)

# Calcular los promedios de las métricas
mean_r2_final = np.mean(cv_results_final["test_R2"])
mean_mae_final = -np.mean(cv_results_final["test_MAE"])
mean_rmse_final = -np.mean(cv_results_final["test_RMSE"])
```

```
Evaluación final de SVR con parámetros óptimos:
Mean R²: 0.730
Mean MAE: 0.441
Mean RMSE: 1.991
```

EVALUACIÓN Y COMPARACIÓN DE RESULTADOS

- Los modelos de Regresión Lineal, Lasso y SVR Mejorado ofrecen un desempeño similar, con un R^2 cercano a 0.73.

- **El SVR Mejorado destaca por tener el MAE más bajo (0.441) y un RMSE ligeramente menor, lo que sugiere una mayor precisión en las predicciones.**

- En contraste, Random Forest y XGBoost, aunque útiles, presentan métricas inferiores en comparación con los tres modelos mencionados. Cabe destacar que el ajuste de hiperparámetros mejoró considerablemente el modelo de XGBoost

Modelo	R2	MAE	RMSE
Regresión Lineal	0.729	0.483	1.993
Random Forest	0.613	1.209	2.394
XGBoost	0.640	0.990	2.305
XGBoost Mejorado	0.713	0.655	2.055
SVR	0.719	0.549	2.033
SVR Mejorado	0.730	0.441	1.991
Lasso	0.729	0.483	1.993

CONCLUSIÓN

El producto final de este estudio es un modelo SVR optimizado, capaz de predecir el rendimiento académico con un R^2 del 73%, lo que lo posiciona como una herramienta robusta. Este modelo permitiría identificar de manera temprana a estudiantes en riesgo y facilitar la toma de decisiones en entornos educativos. Su aplicación práctica permitiría a las instituciones ajustar estrategias, asignar recursos y diseñar intervenciones personalizadas para mejorar la calidad educativa y promover el éxito académico.

Además, aunque la capacidad predictiva actual es prometedora, existe la posibilidad de mejora a través de análisis más profundos y ajustes futuros, abriendo la puerta a nuevas adaptaciones según los cambios y hallazgos en el ámbito educativo.