PROYECTO IA:
PREDICCIÓN DE APROBACIÓN DE PRÉSTAMOS BANCARIOS

Santiago Camargo Ardila - 2211873
Luis Carlos Benavides Torres - 2211925

Inteligencia Artificial I - F1

OBJETIVO

El presente proyecto tiene como objetivo predecir de forma eficiente y precisa la aprobación de un préstamo bancario, considerando el perfil del solicitante y diversos aspectos de su historial crediticio mediante diversos modelos de aprendizaje automático y aprendizaje profundo.

DEFINICIÓN DEL PROBLEMA

Se trata de un problema de clasificación binaria, ya que la variable objetivo solo puede tomar dos valores: préstamo aprobado o préstamo no aprobado.

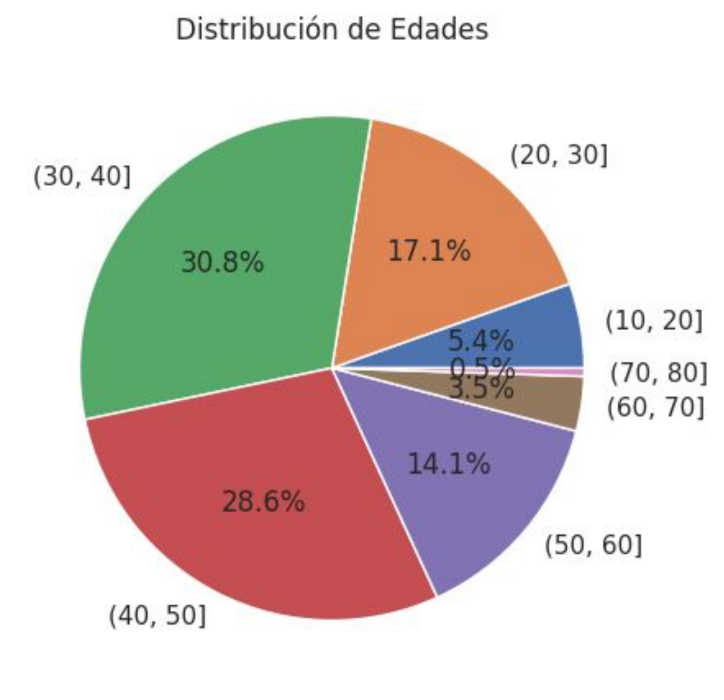
Este enfoque permite automatizar y optimizar la toma de decisiones en procesos crediticios.

ACERCA DEL DATASET

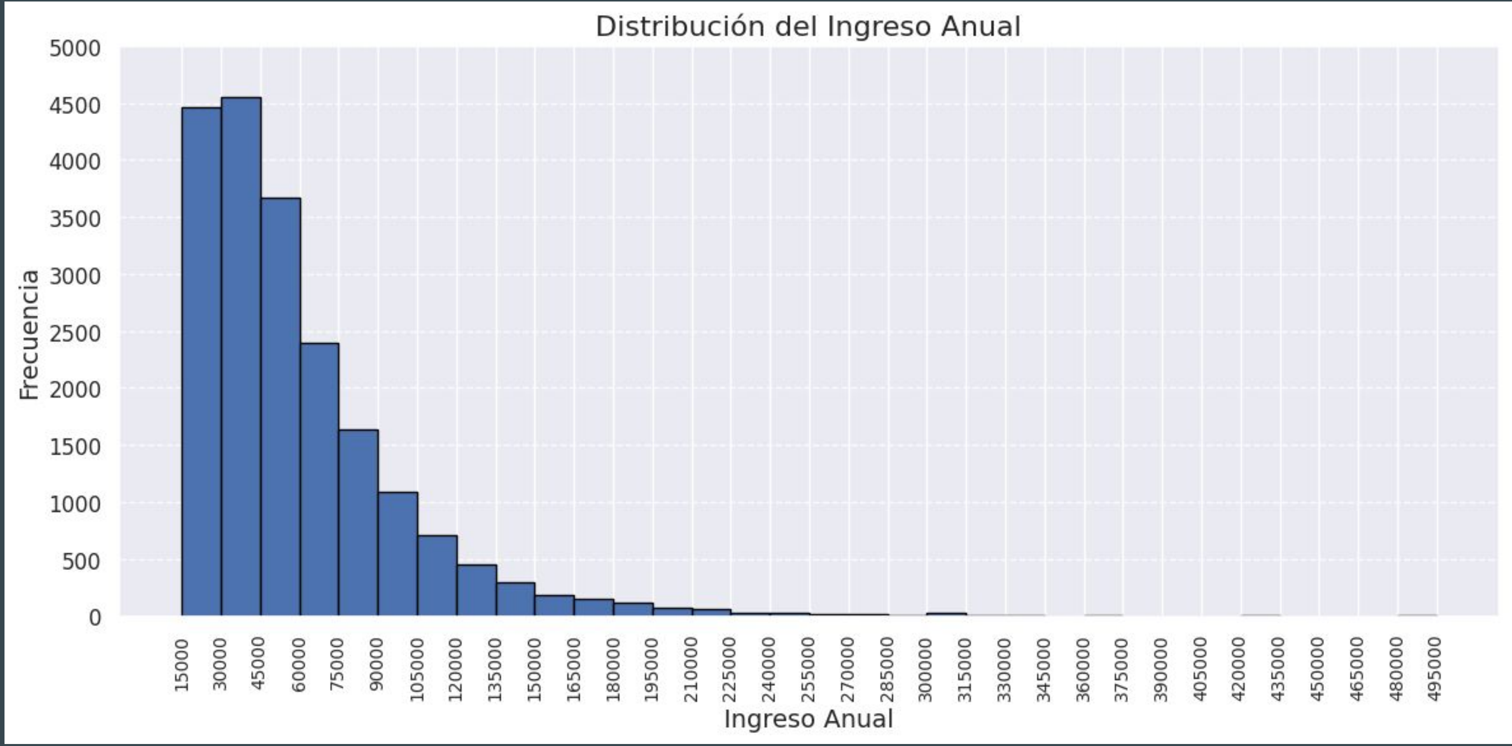
El dataset fue obtenido en Kaggle a través del siguiente enlace https://www.kaggle.com/datasets/lorenzozoppelletto/financial-risk-for-loan-approval?select=Loan.csv

El conjunto de datos está conformado por 20.000 registros y 36 columnas que incluyen información sobre la edad, nivel de ingresos, historial crediticio, situación laboral, entre otros, lo que proporciona una base sólida para análisis predictivos y toma de decisiones en el ámbito crediticio.

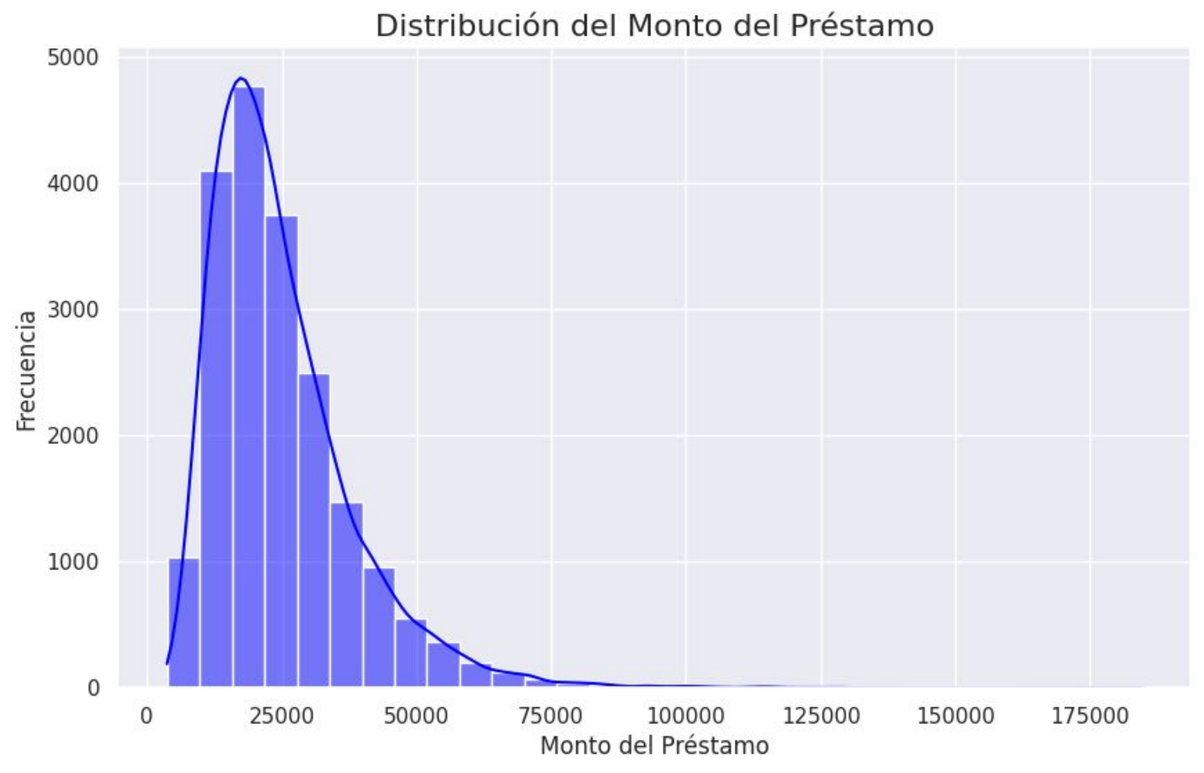
ACERCA DEL DATASET



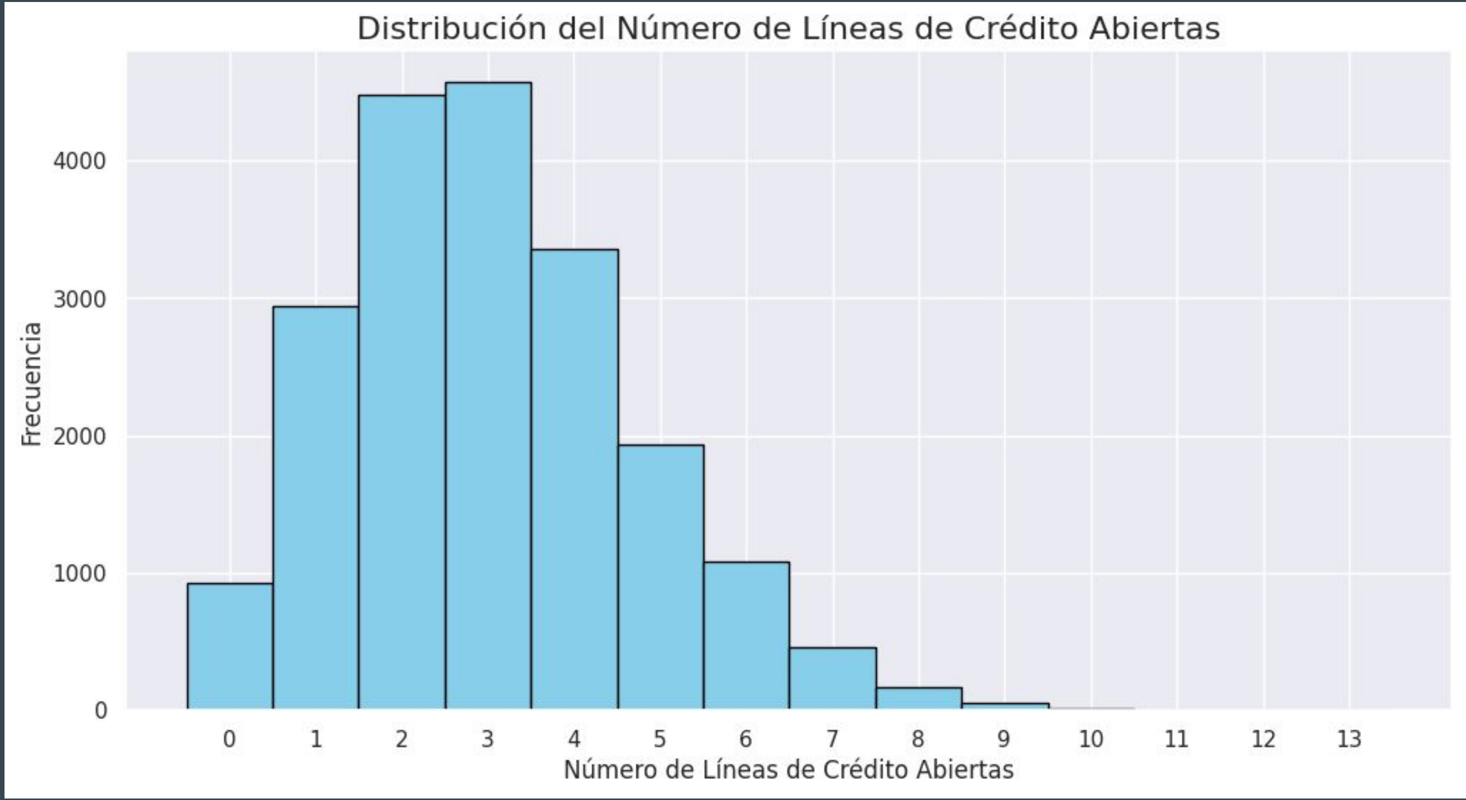
ACERCA DEL DATASET



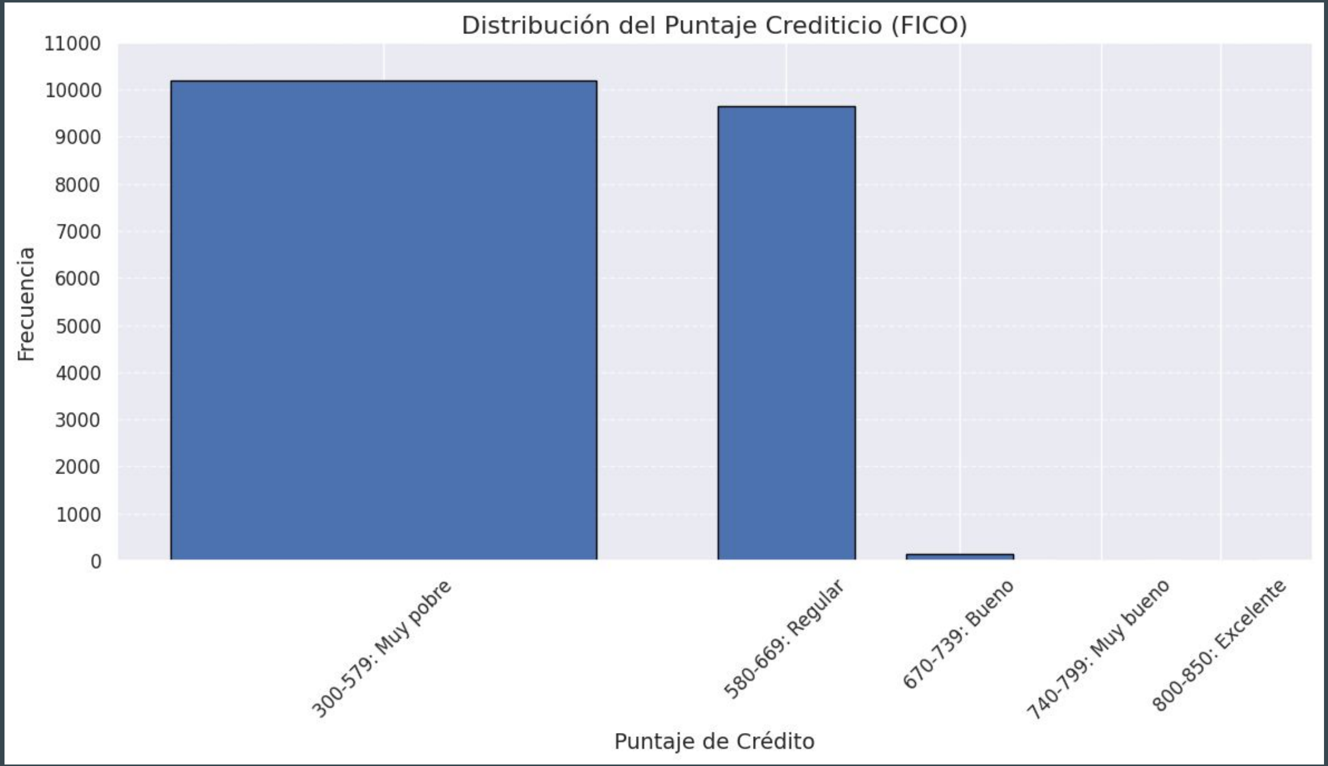
ACERCA DEL DATASET



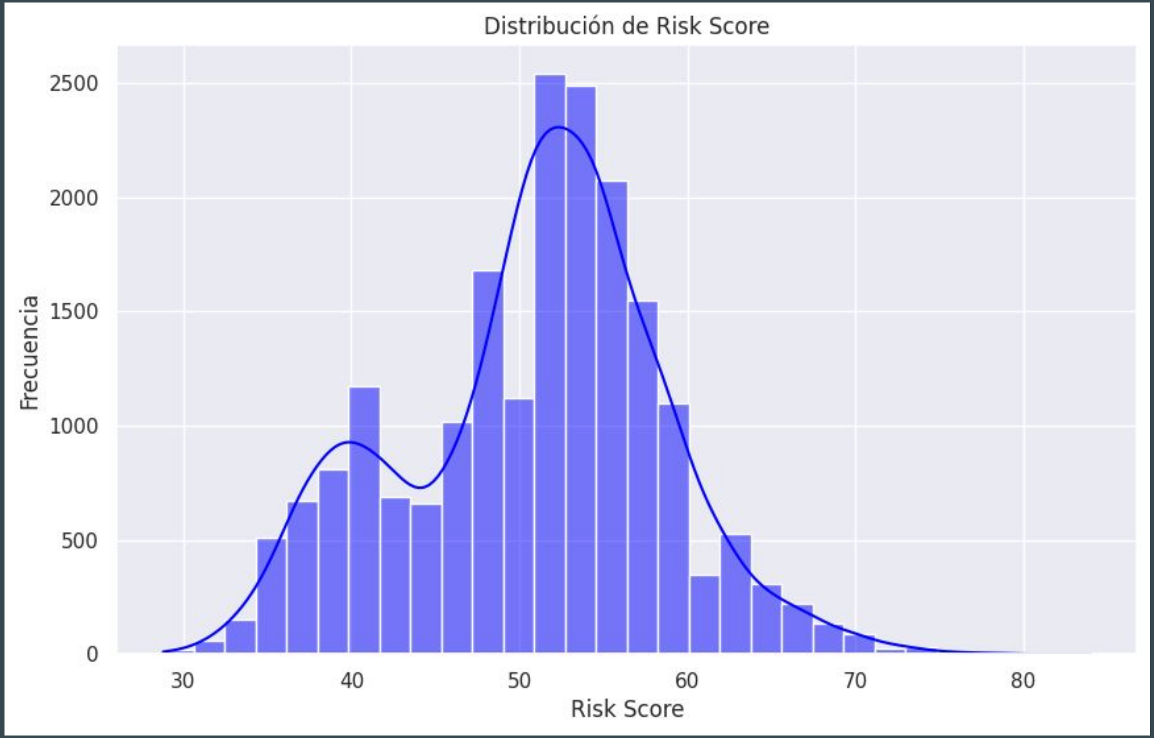
ACERCA DEL DATASET



ACERCA DEL DATASET



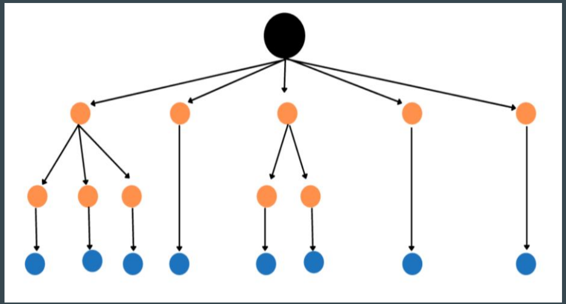
ACERCA DEL DATASET



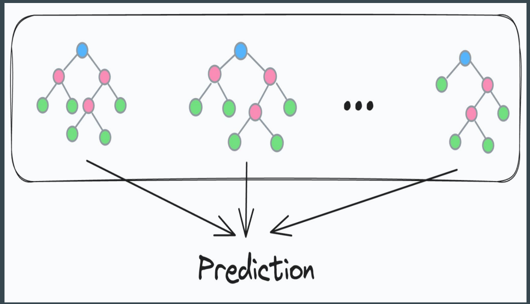
ACERCA DEL DATASET

Modelos implementados

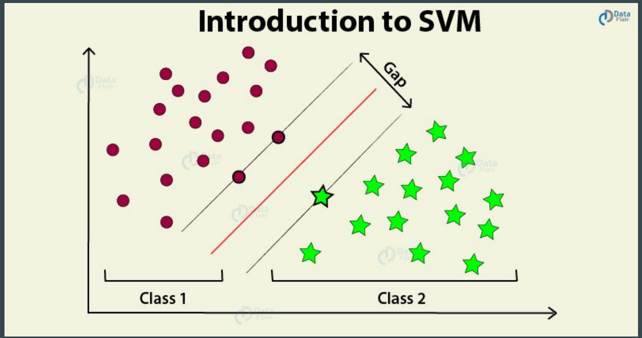
Decision Tree



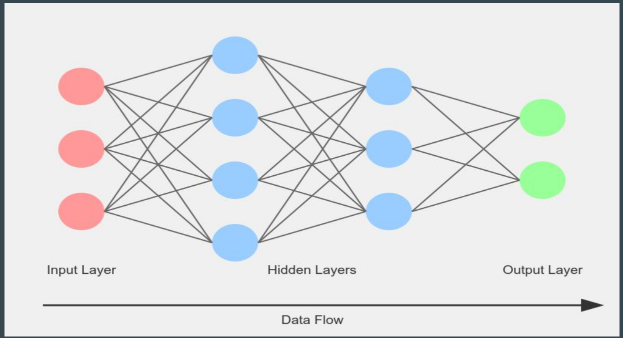
Random Forest



SVM



DeepLearning



CONVERSIÓN DE DATOS CATEGÓRICOS A VALORES NUMÉRICOS

```python
# Identificar columnas categóricas
categorical_columns = df.select_dtypes(include=['object', 'category']).columns

# Mostrar las columnas categóricas y sus valores únicos
for col in categorical_columns:
    print(f"Columna: {col}")
    print(f"Valores únicos: {df[col].unique()}")
    print()

# Convertir las columnas categóricas a números
for col in categorical_columns:
    df[col] = df[col].astype('category').cat.codes

# Verificar los cambios
print(df.head())
```

Ejemplo:

```
Columna: EmploymentStatus
Valores únicos: ['Employed' 'Self-Employed' 'Unemployed']
```

0 1 2

PARTICIONADO [train_test_split]

```
X = df.drop(columns=['LoanApproved'])
y = df['LoanApproved']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

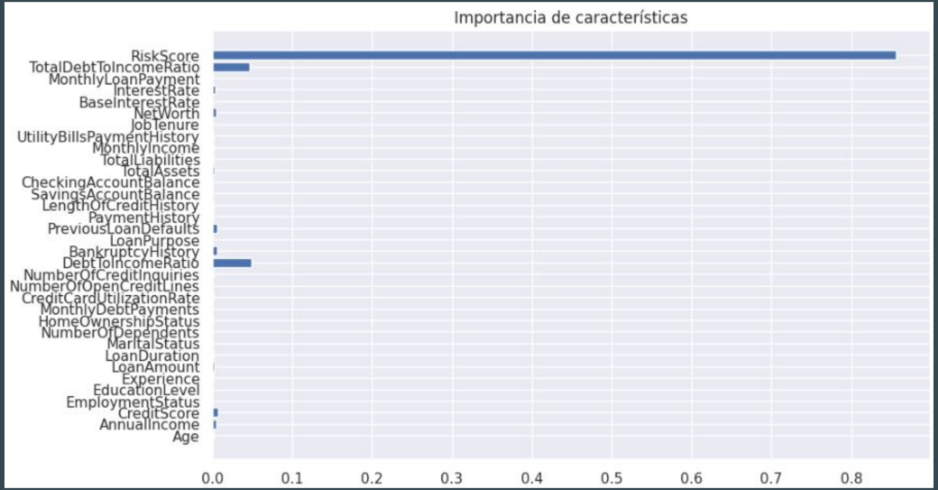
- *Ground truth (y): 'LoanApproved'*
- *Training size: 80%*
- *Test size: 20%*

ESTIMADORES - Accuracy: Decision Tree

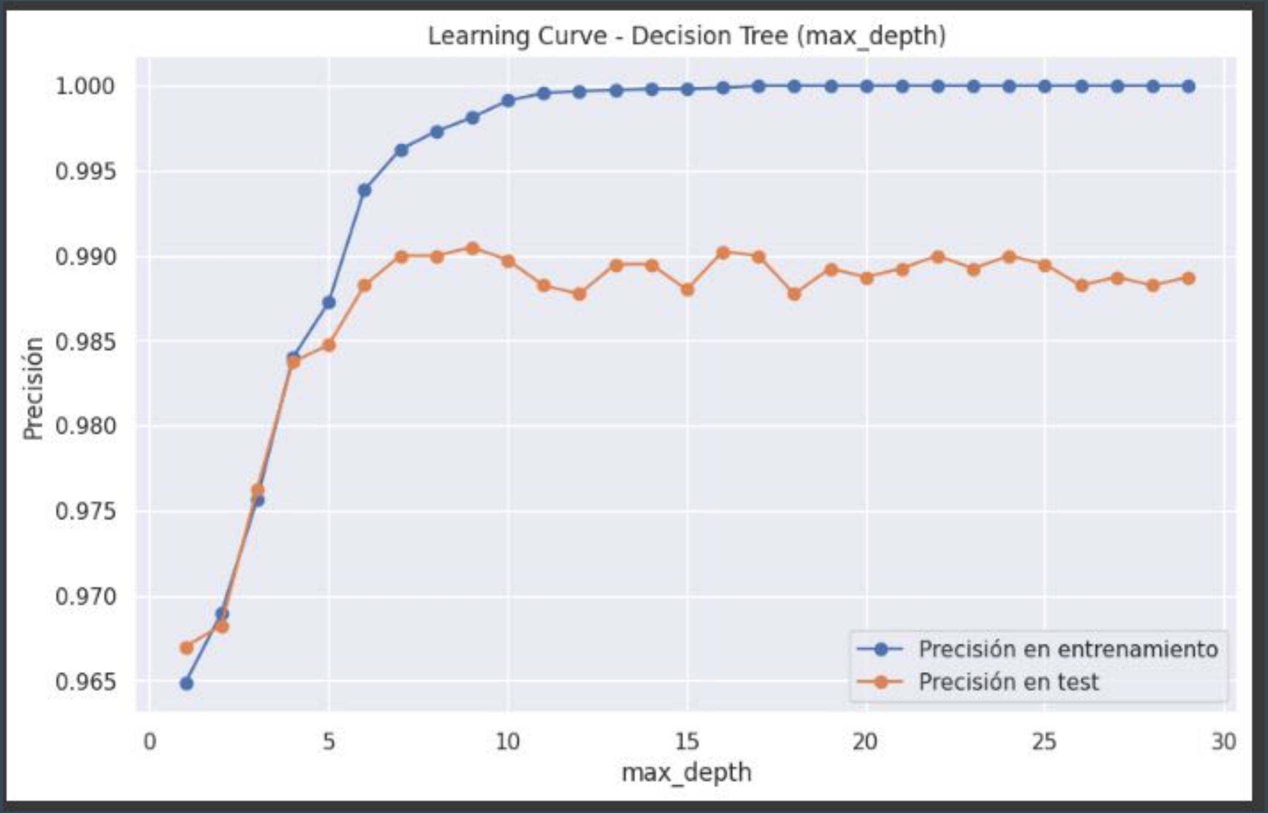
```
est = DecisionTreeClassifier()
est.fit(X_train,y_train)
print(accuracy_score(est.predict(X_test), y_test))
```

```
0.98975
```

*est.feature_importances_*



LEARNING CURVE [Decision Tree - max depth]



ESTIMADORES - Accuracy: Random Forest y SVM

Random Forest

```
[51] est = RandomForestClassifier()
     est.fit(X_train,y_train)
     print(accuracy_score(est.predict(X_test), y_test))
```

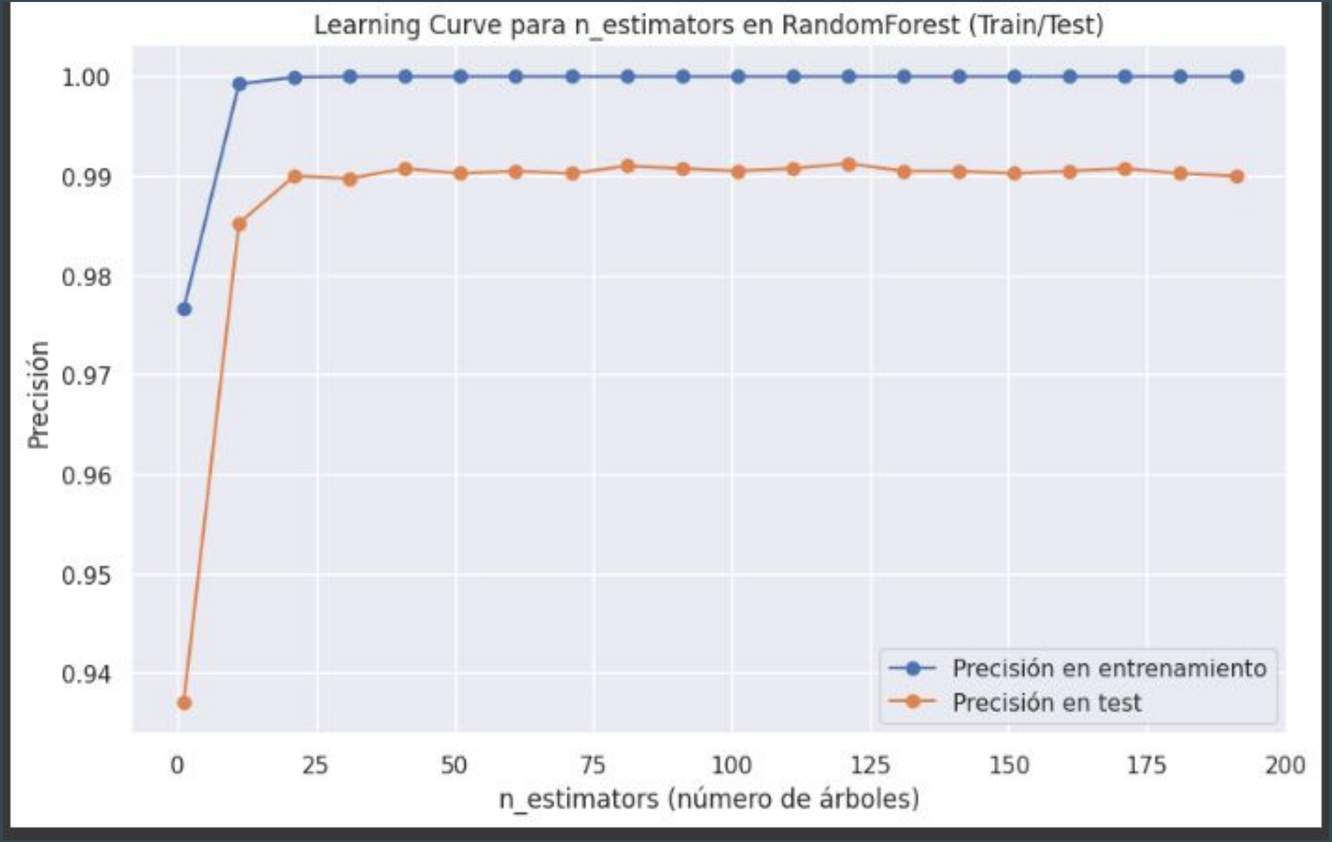
```
0.992
```

SVM

```
from sklearn.svm import SVC
est = SVC()
est.fit(X_train,y_train)
print(accuracy_score(est.predict(X_test), y_test))
```

```
0.88125
```

LEARNING CURVE [Random Forest - n_estimators]



SVM - Kernel

rbf

```
[29] from sklearn.svm import SVC
     est = SVC(kernel="rbf",random_state=1)
     est.fit(X_train,y_train)
     print(accuracy_score(est.predict(X_test), y_test))
```

```
0.88125
```

poly

```
[52] from sklearn.svm import SVC
     est = SVC(kernel="poly",random_state=1)
     est.fit(X_train,y_train)
     print(accuracy_score(est.predict(X_test), y_test))
```

```
0.87025
```

linear

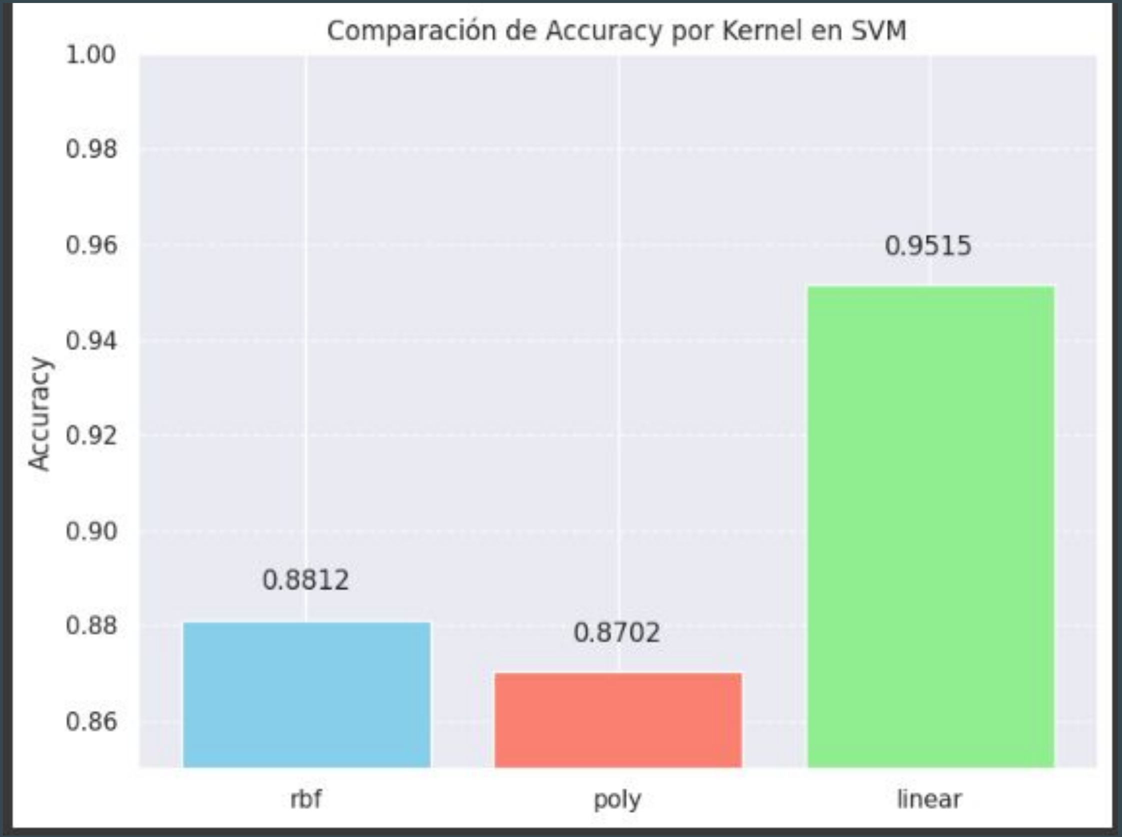
```
from sklearn.svm import SVC

X_small = X_train[:1000]
y_small = y_train[:1000]

est = SVC(kernel="linear",random_state=1)
est.fit(X_small, y_small)
print(accuracy_score(est.predict(X_test), y_test))
```

```
0.93475
```

```
[58] from sklearn.svm import SVC
     est = SVC(kernel="linear",random_state=1)
     est.fit(X_train,y_train)
     print(accuracy_score(est.predict(X_test), y_test))
```

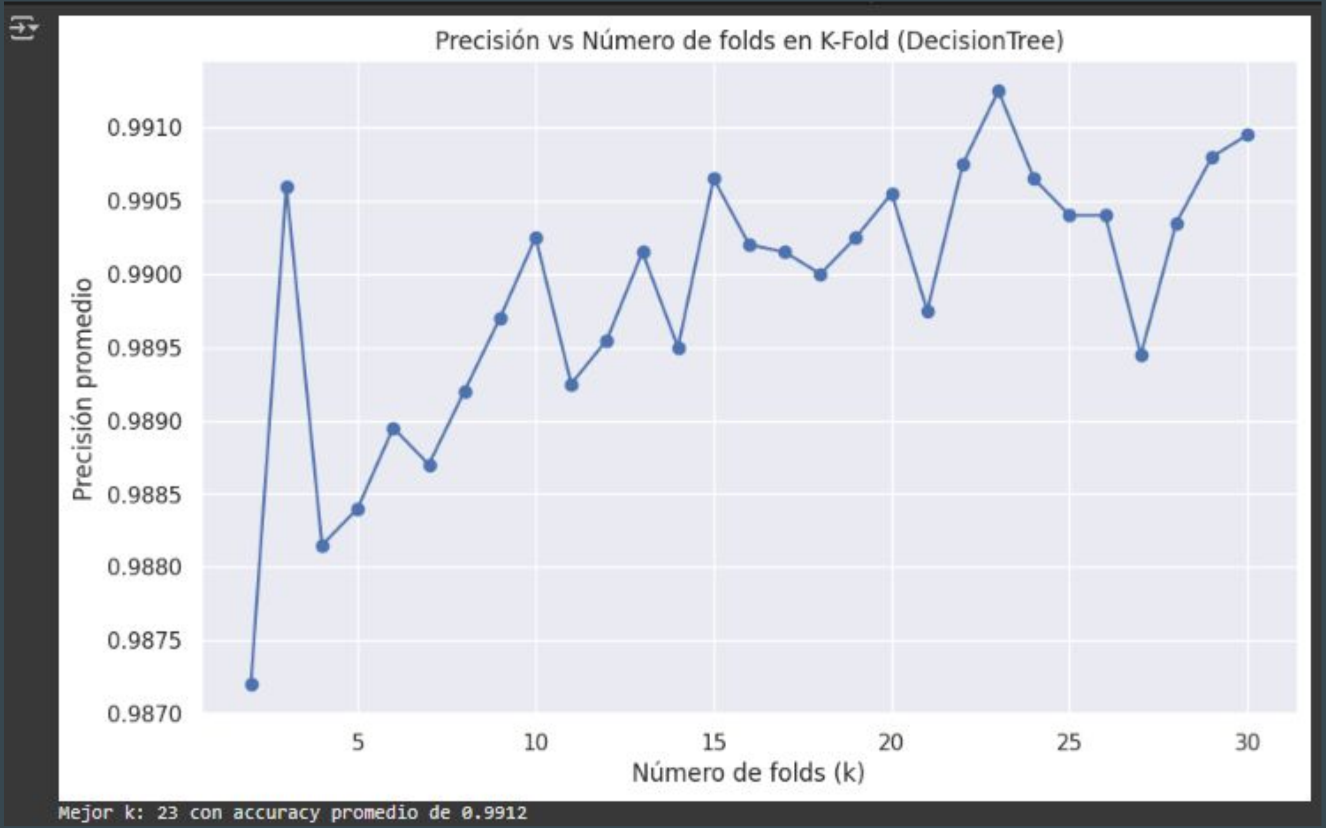
```
0.9515
```

SVM - Kernel



Comparación de Accuracy por Kernel en SVM
Accuracy
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0.8812
0.8702
0.9515
rbf
poly
linear

LEARNING CURVE [Cross Validation - Decision Tree]



RED NEURONAL - Perceptrón de 3 capas

```python
X = df.drop(columns=['LoanApproved'])
y = df['LoanApproved']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Normalización de datos
mean = X_train.mean(axis=0)
std = X_train.std(axis=0)
X_train = (X_train - mean) / std
X_test = (X_test - mean) / std

model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(X_train.shape[1],)),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(2, activation=tf.nn.softmax)
])

model.compile(optimizer='adam', loss='crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/flatten.py:37:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequentia

500/500 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9489 - loss: 0.1180
Epoch 2/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9970 - loss: 0.0079
Epoch 3/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9985 - loss: 0.0092
Epoch 4/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9979 - loss: 0.0039
Epoch 5/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 9.4836e-05
Epoch 6/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 3.6514e-05
Epoch 7/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 9.5012e-06
Epoch 8/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 4.2292e-06
Epoch 9/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 1.0000 - loss: 2.9318e-06
Epoch 10/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 1.0000 - loss: 1.9669e-06
<keras.src.callbacks.history.History at 0x7d87dc1d65d0>
```

RED NEURONAL - Perceptrón de 6 capas

```
X = df.drop(columns=['LoanApproved'])
y = df['LoanApproved']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Normalización de datos
mean = X_train.mean(axis=0)
std = X_train.std(axis=0)
X_train = (X_train - mean) / std
X_test = (X_test - mean) / std

model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(X_train.shape[1],)),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(2, activation=tf.nn.softmax)
])

model.compile(optimizer='adam', loss='crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/flatten.py:37: User

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential mo

500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9426 - loss: 0.1305
Epoch 2/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9982 - loss: 0.0059
Epoch 3/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9987 - loss: 0.0045
Epoch 4/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 1.0000 - loss: 4.6508e-04
Epoch 5/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.9994 - loss: 0.0022
Epoch 6/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9994 - loss: 0.0022
Epoch 7/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 1.0000 - loss: 1.6682e-04
Epoch 8/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9994 - loss: 0.0032
Epoch 9/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 1.0000 - loss: 3.6108e-06
Epoch 10/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 1.0000 - loss: 4.1828e-07
<keras.src.callbacks.history.History at 0x7d87ed11b1d0>
```

RED NEURONAL - Perceptrón de 10 capas

```python
X = df.drop(columns=['LoanApproved'])
y = df['LoanApproved']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Normalización de datos
mean = X_train.mean(axis=0)
std = X_train.std(axis=0)
X_train = (X_train - mean) / std
X_test = (X_test - mean) / std

model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(X_train.shape[1],)),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(128, activation=tf.nn.relu),
  tf.keras.layers.Dense(2, activation=tf.nn.softmax)
])

model.compile(optimizer='adam', loss='crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10)
```

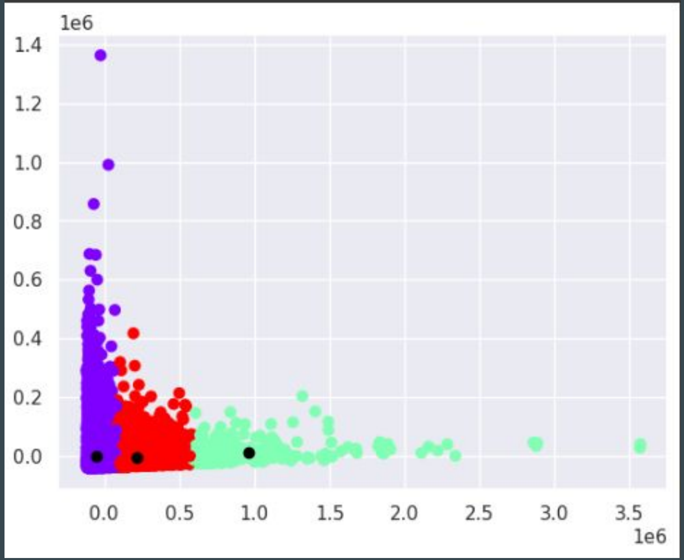
```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefe

500/500 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9218 - loss: 0.1673
Epoch 2/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9971 - loss: 0.0111
Epoch 3/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9989 - loss: 0.0047
Epoch 4/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9991 - loss: 0.0022
Epoch 5/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9996 - loss: 4.7294e-04
Epoch 6/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 1.0000 - loss: 1.3048e-06
Epoch 7/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 1.0000 - loss: 9.3266e-07
Epoch 8/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 1.0000 - loss: 6.1626e-07
Epoch 9/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 1.0000 - loss: 6.4575e-07
Epoch 10/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 1.0000 - loss: 1.3756e-07
<keras.src.callbacks.history.History at 0x7d87d9e59490>
```

REDUCCIÓN DE DIMENSIONALIDAD - PCA

```
from sklearn.decomposition import PCA

X = df.drop(columns=['LoanApproved'])
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
```

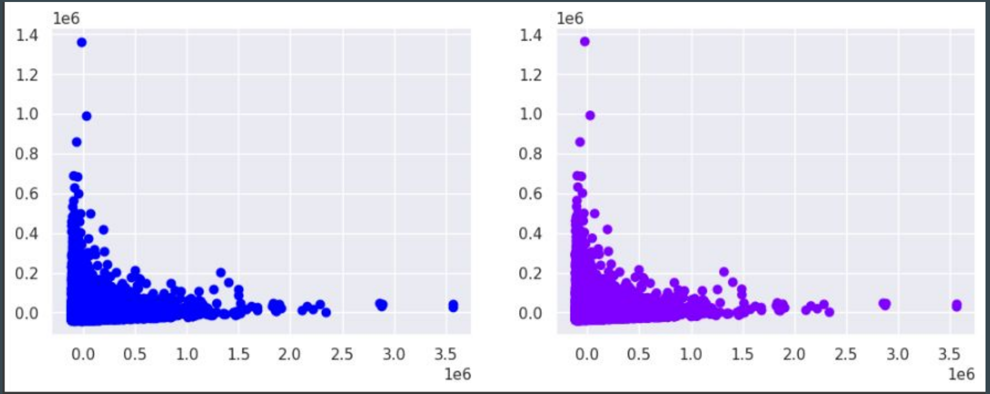
APRENDIZAJE NO SUPERVISADO - K-means

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3)
kmeans.fit(X_pca)
plt.scatter(X_pca[:,0],X_pca[:,1], c=kmeans.labels_, cmap='rainbow')
plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1], color='black')
```



APRENDIZAJE NO SUPERVISADO - DBSCAN

```python
from sklearn.cluster import DBSCAN

DBS = DBSCAN(min_samples=2)
DBS.fit(X_pca)

plt.figure(figsize=(12,4))
plt.subplot(121)
plt.scatter(X_pca[:,0], X_pca[:,1], c="blue", cmap='rainbow');
plt.subplot(122)
plt.scatter(X_pca[:,0],X_pca[:,1], c=DBS.labels_, cmap='rainbow');
```



CONCLUSIONES

Los modelos alcanzaron una precisión casi perfecta, lo que se debe en gran parte a la simplicidad del dataset, con variables limpias, bien estructuradas y altamente correlacionadas con el resultado. Esto facilitó la tarea de predicción, pero también implica que los resultados podrían no generalizar bien a contextos más complejos o reales.

Aunque los resultados son prometedores, se recomienda:

- Probar los modelos en datos más diversos o con [illegible] al.
- Usar validación cruzada y métricas adicionales (precision, rec[illegible]
- Considerar aspectos éticos en la toma de decisiones automatizada.

El sistema tiene potencial para aplicaciones reales, pero requiere pruebas adicionales antes de su implementación práctica.