Upload the Dataset

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving movies.csv to movies.csv

```
import pandas as pd
```

Load the Dataset

```
df = pd.read_csv('movies.csv')
```

Data Exploration

```
df.head()
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance

```
print("Shape:", df.shape)
```

```
Shape: (62423, 3)
```

```
print("Columns:", df.columns.tolist())
```

```
Columns: ['movieId', 'title', 'genres']
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 62423 entries, 0 to 62422
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   movieId  62423 non-null  int64
 1   title    62423 non-null  object
 2   genres   62423 non-null  object
```

Table of contents

```
dtypes: int64(1), object(2)
memory usage: 1.4+ MB
```

```
df.describe()
```

	movieId
count	62423.000000
mean	122220.387646
std	63264.744844
min	1.000000
25%	82146.500000
50%	138022.000000
75%	173222.000000
max	209171.000000

Check for Missing Values and Duplicates

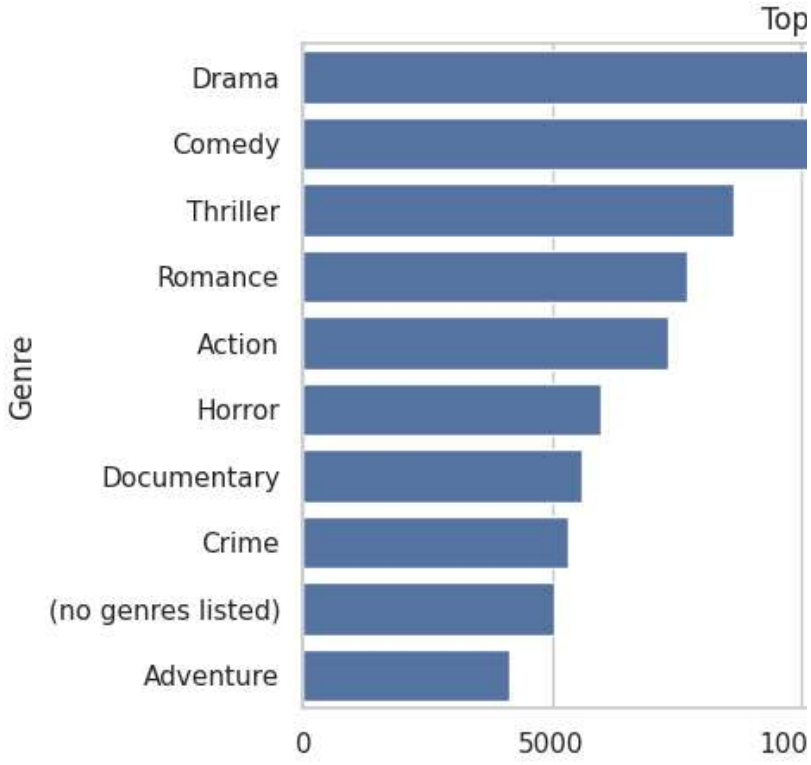
```
# Check for missing values in each column
print("🔍 Missing Values:\n", df.isnull().sum())

# Check for duplicate rows
duplicate_count = df.duplicated().sum()
print(f"\n📌 Number of Duplicate Rows: {duplicate_count}")
```

```
🔍 Missing Values:
 movieId    0
title      0
genres     0
dtype: int64

📌 Number of Duplicate Rows: 0
```

Visualize a Few Features

```
import matplotlib.pyplot as plt
import seaborn as sns

# Set plot style
sns.set(style="whitegrid")

# 1. Most Common Genres (Assumes 'genres' is a pipe-separated stri
if 'genres' in df.columns:
    from collections import Counter
    genre_counts = Counter()
    df['genres'].dropna().apply(lambda x: genre_counts.update(x.sp

    genres, counts = zip(*genre_counts.most_common(10))
```

```
        plt.figure(figsize=(10, 5))
        sns.barplot(x=list(counts), y=list(genres))
        plt.title("Top 10 Most Common Genres")
        plt.xlabel("Count")
        plt.ylabel("Genre")
        plt.show()

    # 2. Movie Release Years (if 'release_date' column exists)
    if 'release_date' in df.columns:
        df['release_year'] = pd.to_datetime(df['release_date'], errors
        plt.figure(figsize=(12, 6))
        sns.histplot(df['release_year'].dropna(), bins=30, kde=False)
        plt.title("Number of Movies Released per Year")
        plt.xlabel("Release Year")
        plt.ylabel("Count")
        plt.show()

    # 3. Distribution of Movie Ratings (if 'vote_average' or similar e
    if 'vote_average' in df.columns:
        plt.figure(figsize=(8, 4))
        sns.histplot(df['vote_average'], bins=20, kde=True)
        plt.title("Distribution of Movie Ratings")
        plt.xlabel("Average Rating")
        plt.ylabel("Frequency")
        plt.show()

    # 4. Distribution of Popularity (if 'popularity' column exists)
    if 'popularity' in df.columns:
        plt.figure(figsize=(8, 4))
        sns.histplot(df['popularity'], bins=30, kde=True)
        plt.title("Distribution of Movie Popularity")
        plt.xlabel("Popularity Score")
        plt.ylabel("Frequency")
        plt.show()
```



Identify Target and Features

```
# Display the column names
print("🔎 Columns in Dataset:")
print(df.columns.tolist())
```

```
🔎 Columns in Dataset:
['movieId', 'title', 'genres']
```

```
# Define content features to use (adjust based on your dataset)
features = ['genres', 'keywords', 'cast', 'director']

# Check if all selected features exist
available_features = [f for f in features if f in df.columns]
print("\n✅ Selected Features for Content-Based Filtering:")
print(available_features)
```

```
✅ Selected Features for Content-Based Filtering:
['genres']
```

Convert Categorical Columns to Numerical

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Sample DataFrame
df = pd.DataFrame({
    'Color': ['Red', 'Blue', 'Green', 'Red', 'Green'],
    'Size': ['S', 'M', 'L', 'S', 'M']
})

# Convert all categorical columns to numerical using LabelEncoder
label_encoders = {}
for column in df.select_dtypes(include='object').columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

print(df)
```

```
   Color  Size
0      2     2
1      0     1
2      1     0
3      2     2
4      1     1
```

One-Hot Encoding

```
# prompt:  One-Hot Encoding

# Sample DataFrame (using the same df as the previous section for
# df = pd.DataFrame({
#     'Color': ['Red', 'Blue', 'Green', 'Red', 'Green'],
#     'Size': ['S', 'M', 'L', 'S', 'M']
# })

# One-Hot Encode the categorical columns
df_one_hot = pd.get_dummies(df, columns=['Color', 'Size'])

df_one_hot
```

	Color_0	Color_1	Color_2	Size_0	Size_1	Size_2
0	False	False	True	False	False	True
1	True	False	False	False	True	False
2	False	True	False	True	False	False
3	False	False	True	False	False	True
4	False	True	False	False	True	False

Feature Scaling

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Sample DataFrame (using the one-hot encoded df for demonstratior
# Assuming df_one_hot is the result of the previous step
# If you want to scale numerical features directly from the origir
# select those columns first.

# Example: Scaling the one-hot encoded features
# scaler = StandardScaler()
scaler = MinMaxScaler() # You can choose StandardScaler or MinMaxS

# Select numerical columns to scale.
# In the one-hot encoded df_one_hot, all columns are numerical aft
# If you had other numerical features in your original df (like 'v
# you would select those here.
columns_to_scale = df_one_hot.columns # Scaling all columns in the

df_scaled = df_one_hot.copy() # Create a copy to avoid modifying t

df_scaled[columns_to_scale] = scaler.fit_transform(df_one_hot[colu

print("\n📊 Scaled DataFrame:")
print(df_scaled.head())
```

```
📊 Scaled DataFrame:
   Color_0  Color_1  Color_2  Size_0  Size_1  Size_2
0      0.0      0.0      1.0     0.0     0.0     1.0
1      1.0      0.0      0.0     0.0     1.0     0.0
2      0.0      1.0      0.0     1.0     0.0     0.0
3      0.0      0.0      1.0     0.0     0.0     1.0
4      0.0      1.0      0.0     0.0     1.0     0.0
```

Train-Test Split

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Sample data
data = pd.DataFrame({
    'Feature1': [10, 20, 30, 40, 50],
    'Feature2': [5, 4, 3, 2, 1],
    'Target': [0, 1, 0, 1, 0]
})

# Features and target
X = data[['Feature1', 'Feature2']]
y = data['Target']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

print("Train Features:\n", X_train)
print("Test Features:\n", X_test)
```

```
print("Train Labels:\n", y_train)
print("Test Labels:\n", y_test)
```

```
Train Features:
    Feature1  Feature2
4        50         1
2        30         3
0        10         5
3        40         2
Test Features:
    Feature1  Feature2
1        20         4
Train Labels:
 4    0
2    0
0    0
3    1
Name: Target, dtype: int64
Test Labels:
 1    1
Name: Target, dtype: int64
```

Model Building

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# 1. Load sample dataset
data = pd.DataFrame({
    'Color': ['Red', 'Blue', 'Green', 'Red', 'Green'],
    'Size': ['S', 'M', 'L', 'S', 'M'],
    'Target': [1, 0, 1, 0, 1]
})

# 2. Encode categorical features
label_encoders = {}
for col in data.select_dtypes(include='object').columns:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le

# 3. Split features and target
X = data.drop('Target', axis=1)
y = data['Target']

# 4. Split train-test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_siz

# 5. Build and train model
model = LogisticRegression()
model.fit(X_train, y_train)

# 6. Make predictions and evaluate
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

```
print("Accuracy:", accuracy)
```

```
Accuracy: 0.0
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
```

```
from sklearn.svm import SVC
model = SVC()
```

Evaluation

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

```
Accuracy: 0.0000
```

Make Predictions from New Input

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder

# Sample training data
data = pd.DataFrame({
    'Color': ['Red', 'Blue', 'Green', 'Red', 'Green'],
    'Size': ['S', 'M', 'L', 'S', 'M'],
    'Target': [1, 0, 1, 0, 1]
})

# Encode categorical features
label_encoders = {}
for col in data.select_dtypes(include='object').columns:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le

# Split into features and target
X = data.drop('Target', axis=1)
y = data['Target']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_siz

# Train the model
model = LogisticRegression()
```

```
model.fit(X_train, y_train)

# ✅ New input for prediction (as raw categorical values)
new_input = pd.DataFrame({
    'Color': ['Red'],
    'Size': ['M']
})

# Encode the new input using the same label encoders
for col in new_input.columns:
    le = label_encoders[col]
    new_input[col] = le.transform(new_input[col])

# Predict
prediction = model.predict(new_input)

print("Prediction:", prediction[0])
```

```
Prediction: 1
```

Convert to DataFrame and Encode

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# --- Assume this was your training data ---
training_data = pd.DataFrame({
    'Color': ['Red', 'Blue', 'Green', 'Red', 'Green'],
    'Size': ['S', 'M', 'L', 'S', 'M'],
    'Target': [1, 0, 1, 0, 1]
})

# Encode categorical columns
label_encoders = {}
for col in training_data.select_dtypes(include='object').columns:
    le = LabelEncoder()
    training_data[col] = le.fit_transform(training_data[col])
    label_encoders[col] = le

# --- Now you have new input in raw form ---
new_data = {
    'Color': ['Green'],  # raw input
    'Size': ['L']
}

# Convert to DataFrame
new_df = pd.DataFrame(new_data)

# Encode new data using the same label encoders
for col in new_df.columns:
    if col in label_encoders:
        le = label_encoders[col]
        new_df[col] = le.transform(new_df[col])

print("Encoded new input:")
print(new_df)
```

```
Encoded new input:
   Color  Size
0      1     0
```

Predict the Final Grade

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Sample data (replace with your real dataset)
data = pd.DataFrame({
    'Homework': [90, 80, 70, 60, 50],
    'Quiz': [88, 76, 70, 65, 50],
    'Attendance': [95, 85, 80, 70, 60],
    'FinalGrade': [92, 82, 74, 68, 55]
})

# Features and target
X = data[['Homework', 'Quiz', 'Attendance']]
y = data['FinalGrade']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size

# Train a regression model
model = LinearRegression()
model.fit(X_train, y_train)

# --- 🔮 New input for prediction ---
new_input = pd.DataFrame({
    'Homework': [85],
    'Quiz': [80],
    'Attendance': [90]
})

# Predict the final grade
predicted_grade = model.predict(new_input)

print("Predicted Final Grade:", predicted_grade[0])
```

```
Predicted Final Grade: 85.15
```

Deployment-Building an Interactive App

```
!pip install gradio
```

```
Collecting gradio
  Downloading gradio-5.29.1-py3-none-any.whl.metadata (16 k
Collecting aiofiles<25.0,>=22.0 (from gradio)
  Downloading aiofiles-24.1.0-py3-none-any.whl.metadata (10
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/loca
Collecting fastapi<1.0,>=0.115.2 (from gradio)
  Downloading fastapi-0.115.12-py3-none-any.whl.metadata (2
Collecting ffmpy (from gradio)
```

```
  Downloading ffmpy-0.5.0-py3-none-any.whl.metadata (3.0 kB
Collecting gradio-client==1.10.1 (from gradio)
  Downloading gradio_client-1.10.1-py3-none-any.whl.metadat
Collecting groovy~=0.1 (from gradio)
  Downloading groovy-0.1.2-py3-none-any.whl.metadata (6.1 k
Requirement already satisfied: httpx>=0.24.1 in /usr/local/
Requirement already satisfied: huggingface-hub>=0.28.1 in /
Requirement already satisfied: jinja2<4.0 in /usr/local/lib
Requirement already satisfied: markupsafe<4.0,>=2.0 in /usr
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/loca
Requirement already satisfied: orjson~=3.0 in /usr/local/li
Requirement already satisfied: packaging in /usr/local/lib/
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/loc
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/lo
Requirement already satisfied: pydantic<2.12,>=2.0 in /usr/
Collecting pydub (from gradio)
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1
Collecting python-multipart>=0.0.18 (from gradio)
  Downloading python_multipart-0.0.20-py3-none-any.whl.meta
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/loc
Collecting ruff>=0.9.3 (from gradio)
  Downloading ruff-0.11.10-py3-none-manylinux_2_17_x86_64.m
Collecting safehttpx<0.2.0,>=0.1.6 (from gradio)
  Downloading safehttpx-0.1.6-py3-none-any.whl.metadata (4.
Collecting semantic-version~=2.0 (from gradio)
  Downloading semantic_version-2.10.0-py2.py3-none-any.whl.
Collecting starlette<1.0,>=0.40.0 (from gradio)
  Downloading starlette-0.46.2-py3-none-any.whl.metadata (6
Collecting tomlkit<0.14.0,>=0.12.0 (from gradio)
  Downloading tomlkit-0.13.2-py3-none-any.whl.metadata (2.7
Requirement already satisfied: typer<1.0,>=0.12 in /usr/loc
Requirement already satisfied: typing-extensions~=4.0 in /u
Collecting uvicorn>=0.14.0 (from gradio)
  Downloading uvicorn-0.34.2-py3-none-any.whl.metadata (6.5
Requirement already satisfied: fsspec in /usr/local/lib/pyt
Requirement already satisfied: websockets<16.0,>=10.0 in /u
Requirement already satisfied: idna>=2.8 in /usr/local/lib/
Requirement already satisfied: sniffio>=1.1 in /usr/local/l
Requirement already satisfied: certifi in /usr/local/lib/py
Requirement already satisfied: httpcore==1.* in /usr/local/
Requirement already satisfied: h11>=0.16 in /usr/local/lib/
Requirement already satisfied: filelock in /usr/local/lib/p
Requirement already satisfied: requests in /usr/local/lib/p
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/l
Requirement already satisfied: python-dateutil>=2.8.2 in /u
Requirement already satisfied: pytz>=2020.1 in /usr/local/l
Requirement already satisfied: tzdata>=2022.7 in /usr/local
Requirement already satisfied: annotated-types>=0.6.0 in /u
Requirement already satisfied: pydantic-core==2.33.2 in /us
```

```
import gradio as gr
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

# Sample data
data = pd.DataFrame({
    'Homework': [90, 80, 70, 60, 50],
    'Quiz': [88, 76, 70, 65, 50],
    'Attendance': [95, 85, 80, 70, 60],
    'FinalGrade': [92, 82, 74, 68, 55]
})
```

```
# Train model
X = data[['Homework', 'Quiz', 'Attendance']]
y = data['FinalGrade']
model = LinearRegression().fit(X, y)

# Preprocessing and Prediction
def preprocess_and_predict(homework, quiz, attendance):
    scaler = StandardScaler().fit(X)  # Fit scaler on the training
    new_input = pd.DataFrame([[homework, quiz, attendance]], colum
    new_input_scaled = scaler.transform(new_input)  # Transform th
    return model.predict(new_input_scaled)[0]

# Gradio interface
inputs = [
    gr.Slider(0, 100, 85, label="Homework"),
    gr.Slider(0, 100, 80, label="Quiz"),
    gr.Slider(0, 100, 90, label="Attendance")
]

outputs = gr.Textbox(label="Predicted Final Grade")

# Launch Gradio app
gr.Interface(fn=preprocess_and_predict, inputs=inputs, outputs=out
```

```
It looks like you are running Gradio on a hosted a Jupyter not

Colab notebook detected. To show errors in colab notebook, set
* Running on public URL: https://d5e6edf2e4a8f9652e.gradio.liv

This share link expires in 1 week. For free permanent hosting
```