1. Upload the Dataset

```
from google.colab import files
import pandas as pd

# Upload file manually
uploaded = files.upload()

# Load dataset (assuming one CSV is uploaded)
for filename in uploaded.keys():
    df = pd.read_csv(filename)

# Display first few rows
df.head()
```

```
Choose files students_dataset.csv
• students_dataset.csv(text/csv) - 313 bytes, last modified: 24/04/2025 - 100% done
Saving students_dataset.csv to students_dataset.csv
```

	Name	Age	Marks	Attendance	Passed
0	Alice	20.0	85.0	90.0	Yes
1	Bob	21.0	NaN	80.0	No
2	Charlie	NaN	78.0	NaN	Yes
3	David	22.0	90.0	85.0	Yes
4	Eve	20.0	88.0	95.0	No

Next steps: Generate code with df | View recommended plots | New interactive sheet

2.Load the Dataset

```
import pandas as pd
import os

# Get the current working directory
current_directory = os.getcwd()
print(f"Current working directory: {current_directory}")

# Construct the full file path
file_path = os.path.join(current_directory, "creditcard.csv")

# Check if the file exists at the specified path
if os.path.exists(file_path):
    # Load the dataset if the file exists
    df = pd.read_csv(file_path)
    # Preview the dataset
    print("Dataset loaded successfully!")
    print("Shape of the dataset:", df.shape)
    print("\nFirst 5 rows:")
    print(df.head())
else:
    # Print an error message if the file does not exist
    print(f"Error: File not found at {file_path}")
```

```
Current working directory: /content
Error: File not found at /content/creditcard.csv
```

3. Data Exploration

```
# Importing libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import io # Import the io module

# Load the dataset
# Assuming the file is uploaded using the previous cell (ipython-input-6-ed9414f01325)
# df = pd.read_csv(io.BytesIO(uploaded['creditcard.csv'])) # Read from uploaded data
# Updated line to use the actual filename from the uploaded dictionary
```

```
for filename in uploaded.keys():
    df = pd.read_csv
```

4.Check for Missing Values and Duplicates

```
import pandas as pd
import os

# Get the current working directory
current_directory = os.getcwd()
print(f"Current working directory: {current_directory}")

# Construct the full file path - Modify this with your actual filename if it's different
file_path = os.path.join(current_directory, "creditcard.csv")

# Load the dataset if the file exists
try:
    df = pd.read_csv(file_path)
    # Preview the dataset
    print("Dataset loaded successfully!")
    print("Shape of the dataset:", df.shape)
    print("\nFirst 5 rows:")
    print(df.head())
except FileNotFoundError:
    # Print an error message if the file does not exist
    print(f"Error: File not found at {file_path}. Please check the file name and path.")
```

```
Current working directory: /content
Error: File not found at /content/creditcard.csv. Please check the file name and path.
```

5. Visualize a Few Features

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import io
from google.colab import files

# Upload file manually if not already uploaded
try:
    # Try accessing the uploaded file using the correct filename
    # (Assuming 'students_dataset.csv' is the actual uploaded file)
    df = pd.read_csv(io.BytesIO(uploaded['students_dataset.csv']))
except NameError:  # Handle the case where 'uploaded' is not defined
    uploaded = files.upload()
    df = pd.read_csv(io.BytesIO(uploaded['students_dataset.csv'])) # Indented this line to be part of the except block
```

6. Identify Target and Features

```
from google.colab import files

# Upload the file
uploaded = files.upload()

# Check if the file was uploaded successfully
if 'creditcard.csv' in uploaded:
    # Load the dataset
    df = pd.read_csv(io.BytesIO(uploaded['creditcard.csv']))  # io.BytesIO is used to read the in-memory file

    # Now you can continue with your data processing
    print("✅ Dataset loaded successfully!")
else:
    print("❌ 'creditcard.csv' was not found in the uploaded files.")
```

```
Choose files students_dataset.csv
• students_dataset.csv(text/csv) - 313 bytes, last modified: 24/04/2025 - 100% done
Saving students_dataset.csv to students_dataset (1).csv
❌ 'creditcard.csv' was not found in the uploaded files.
```

7.Convert Categorical Columns to Numerical

```
from google.colab import files
import io
import pandas as pd

# Upload the file
uploaded = files.upload()

# Load dataset
if 'creditcard.csv' in uploaded:
    df = pd.read_csv(io.BytesIO(uploaded['creditcard.csv']))
    print("✅ Dataset loaded successfully!")
else:
    print("❌ 'creditcard.csv' was not found in the uploaded files.")

# Check for categorical columns
categorical_cols = df.select_dtypes(include=['object', 'category']).columns.tolist()

print(f"📄 Categorical columns found: {categorical_cols if categorical_cols else 'None'}")

# Convert categorical columns to numerical (if any)
if categorical_cols:
    df_encoded = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
    print("✅ Categorical columns converted using one-hot encoding.")
else:
    df_encoded = df.copy()
    print("✅ No categorical columns to convert.")

# Optional: Show new dataframe shape
print(f"New dataframe shape: {df_encoded.shape}")
```

```
Choose files students_dataset.csv
• students_dataset.csv(text/csv) - 313 bytes, last modified: 24/04/2025 - 100% done
Saving students_dataset.csv to students_dataset (2).csv
❌ 'creditcard.csv' was not found in the uploaded files.
📄 Categorical columns found: ['Name', 'Passed']
✅ Categorical columns converted using one-hot encoding.
New dataframe shape: (16, 10)
```

8.One-Hot Encoding

```
from google.colab import files
import io
import pandas as pd

# Upload the file manually
uploaded = files.upload()

# Load the dataset (assuming 'creditcard.csv' is uploaded)
if 'creditcard.csv' in uploaded:
    df = pd.read_csv(io.BytesIO(uploaded['creditcard.csv']))  # Use io.BytesIO for in-memory file
    print("✅ Dataset loaded successfully!")
else:
    print("❌ 'creditcard.csv' was not found in the uploaded files.")

# Continue with your data processing
# ... (rest of your code) ...
```

```
Choose files students_dataset.csv
• students_dataset.csv(text/csv) - 313 bytes, last modified: 24/04/2025 - 100% done
Saving students_dataset.csv to students_dataset (3).csv
❌ 'creditcard.csv' was not found in the uploaded files.
```

9. Feature Scaling

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
import io
from google.colab import files

# Upload file manually if not already uploaded
try:
    df  # Check if df is already defined
except NameError:
    uploaded = files.upload()
    df = pd.read_csv(io.BytesIO(uploaded['creditcard.csv']))  # Make sure 'creditcard.csv' is the correct filename
```

```
# Check if 'Class' column exists before dropping
if 'Class' in df.columns:
    # Separate features and target
    X = df.drop(columns=['Class'])  # Features
    y = df['Class']  # Target

    # Scale features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Optional: convert scaled data back to DataFrame for readability
    X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)

    # Show the scaled feature sample
    print("✅ Feature scaling applied using StandardScaler.")
    print(X_scaled_df.head())
else:
    print("❌ 'Class' column not found in the DataFrame.")
```

```
❌ 'Class' column not found in the DataFrame.
```

10.Train-Test Split

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import io
from google.colab import files

# Try to load the DataFrame from memory (if it exists)
try:
    df
except NameError:
    # If df is not in memory, upload the file manually
    #uploaded = files.upload() # Commented out to avoid re-uploading if file is already uploaded.
    # Assume the user has already uploaded the file in previous cells
    df = pd.read_csv(io.BytesIO(uploaded['students_dataset.csv']))  # Changed filename to 'students_dataset.csv'

# Replace 'actual_target_column' with the actual name of your target column
# For example, if your target column is named 'Grade', use 'Grade' instead of 'actual_target_column'
actual_target_column = 'Marks'

# Separate features and target
# Select only numerical features for scaling
numerical_features = df.select_dtypes(include=['number']).columns.tolist()
numerical_features.remove(actual_target_column)  # Remove target column from features

X = df[numerical_features]
y = df[actual_target_column]

# ---->  Drop rows with NaN in the target variable ('Marks') <----
X = X[y.notna()] # Filter X to keep rows where y is not NaN
y = y.dropna()   # Remove NaN values from y

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split into train and test sets (stratify ensures class balance)
# Note: Stratify might not be appropriate for regression tasks. Consider removing it if 'Marks' is continuous.
# ---->  Remove stratify parameter <----
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42)

# Confirm shapes
print(f"✅ Training set shape: {X_train.shape}, {y_train.shape}")
print(f"✅ Test set shape: {X_test.shape}, {y_test.shape}")
```

```
✅ Training set shape: (8, 2), (8,)
✅ Test set shape: (4, 2), (4,)
```

11.Model Building

```
pip install pandas scikit-learn matplotlib seaborn
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.15.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
from google.colab import files
import io

# ... (rest of your code) ...


# Step 4: Split dataset
X = df.drop(target_column, axis=1)
y = df[target_column]

# ----> Drop rows with NaN in the target variable ('Marks') <----
X = X[y.notna()] # Filter X to keep rows where y is not NaN
y = y.dropna()   # Remove NaN values from y


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  # ----> Remove stratify parameter <----

# ... (rest of your code) ...
```

12. Evaluation

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier  # Assuming RandomForestClassifier is your desired model
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
from google.colab import files
import io


# ... (Load and prepare your data as before) ...

# Assuming 'target_column' is defined in your code to indicate the target variable
target_column = 'Marks'  # Replace 'Marks' with your actual target column name

# Step 4: Split dataset
X = df.drop(target_column, axis=1)
y = df[target_column]

# Drop rows with NaN in the target variable
X = X[y.notna()]
y = y.dropna()

# ----> Select only numerical features for training <----
```

```
X = X.select_dtypes(include=['number']) # This line keeps only numerical features

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# ----> Initialize and train the model <----
model = RandomForestClassifier(random_state=42) # Create a RandomForestClassifier instance
model.fit(X_train, y_train) # Train the model on the training data


# Now you can continue with the evaluation part of your code
from sklearn.metrics import (
    confusion_matrix,
    classification_report,
    roc_auc_score,
    roc_curve,
    precision_recall_curve
)

# ... (rest of the evaluation code) ...
```

13.Make Predictions from New Input

```
import pandas as pd
import numpy as np

# Example: New transaction data (replace with real values or inputs)
# ----> Update new_transaction with the actual column names from your training data and their respective values <----
new_transaction = {
    'Age': 25,  # Example value, replace with actual value
    'Attendance': 90,  # Example value, replace with actual value
    # ... Add other relevant columns and their values...
}

# Convert to DataFrame
new_df = pd.DataFrame([new_transaction])

# ----> Ensure the columns in new_df match the order of columns used during training <----
training_columns = X_train.columns  # Get the column names from your training data
new_df = new_df[training_columns]  # Reorder columns in new_df to match training data

# Predict using trained model
prediction = model.predict(new_df)[0]
# probability = model.predict_proba(new_df)[0][1]  # Assuming binary classification - comment out or adjust if not needed

# Output the result
# Adjust output based on your specific prediction task
print(f"Prediction for new data: {prediction}")
```

```
Prediction for new data: 92.0
```

14.Make Convert to DataFrame and Encode

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

```
Choose files students_dataset.csv
• students_dataset.csv(text/csv) - 313 bytes, last modified: 24/04/2025 - 100% done
Saving students_dataset.csv to students_dataset (3).csv
```

15. Predict the Final Grade

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

```
Choose files students_dataset.csv
• students_dataset.csv(text/csv) - 313 bytes, last modified: 24/04/2025 - 100% done
Saving students_dataset.csv to students_dataset (4).csv
```

16.Deployment-Building an Interactive App

```
pip install streamlit pandas scikit-learn
```

```
Collecting streamlit
  Downloading streamlit-1.45.0-py3-none-any.whl.metadata (8.9 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: altair<6,>=4.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (5.5.0)
Requirement already satisfied: blinker<2,>=1.5.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (1.9.0)
Requirement already satisfied: cachetools<6,>=4.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (5.5.2)
Requirement already satisfied: click<9,>=7.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (8.1.8)
Requirement already satisfied: numpy<3,>=1.23 in /usr/local/lib/python3.11/dist-packages (from streamlit) (2.0.2)
Requirement already satisfied: packaging<25,>=20 in /usr/local/lib/python3.11/dist-packages (from streamlit) (24.2)
Requirement already satisfied: pillow<12,>=7.1.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (11.2.1)
Requirement already satisfied: protobuf<7,>=3.20 in /usr/local/lib/python3.11/dist-packages (from streamlit) (5.29.4)
Requirement already satisfied: pyarrow>=7.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (18.1.0)
Requirement already satisfied: requests<3,>=2.27 in /usr/local/lib/python3.11/dist-packages (from streamlit) (2.32.3)
Requirement already satisfied: tenacity<10,>=8.1.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (9.1.2)
Requirement already satisfied: toml<2,>=0.10.1 in /usr/local/lib/python3.11/dist-packages (from streamlit) (0.10.2)
Requirement already satisfied: typing-extensions<5,>=4.4.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (4.13.2)
Collecting watchdog<7,>=2.1.5 (from streamlit)
  Downloading watchdog-6.0.0-py3-none-manylinux2014_x86_64.whl.metadata (44 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.3/44.3 kB 1.5 MB/s eta 0:00:00
Requirement already satisfied: gitpython!=3.1.19,<4,>=3.0.7 in /usr/local/lib/python3.11/dist-packages (from streamlit) (3.1.44)
Collecting pydeck<1,>=0.8.0b4 (from streamlit)
  Downloading pydeck-0.9.1-py2.py3-none-any.whl.metadata (4.1 kB)
Requirement already satisfied: tornado<7,>=6.0.3 in /usr/local/lib/python3.11/dist-packages (from streamlit) (6.4.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.15.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from altair<6,>=4.0->streamlit) (3.1.6)
Requirement already satisfied: jsonschema>=3.0 in /usr/local/lib/python3.11/dist-packages (from altair<6,>=4.0->streamlit) (4.23.0)
Requirement already satisfied: narwhals>=1.14.2 in /usr/local/lib/python3.11/dist-packages (from altair<6,>=4.0->streamlit) (1.37.1)
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3.11/dist-packages (from gitpython!=3.1.19,<4,>=3.0.7->streamlit)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->streamlit) (
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->streamlit) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->streamlit) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->streamlit) (2025.4
Requirement already satisfied: smmap<6,>=3.0.1 in /usr/local/lib/python3.11/dist-packages (from gitdb<5,>=4.0.1->gitpython!=3.1.19,<4,>=
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->altair<6,>=4.0->streamlit) (3.0.
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=3.0->al
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=3.0->altair<6,>=4.0->str
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=3.0->altair<6,>=4.0->streamli
Downloading streamlit-1.45.0-py3-none-any.whl (9.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.9/9.9 MB 57.6 MB/s eta 0:00:00
Downloading pydeck-0.9.1-py2.py3-none-any.whl (6.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.9/6.9 MB 66.8 MB/s eta 0:00:00
Downloading watchdog-6.0.0-py3-none-manylinux2014_x86_64.whl (79 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79.1/79.1 kB 5.9 MB/s eta 0:00:00
Installing collected packages: watchdog, pydeck, streamlit
Successfully installed pydeck-0.9.1 streamlit-1.45.0 watchdog-6.0.0
```

```
# train_model.py
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import joblib
from google.colab import files
import io  # Make sure to import the io module

# Upload the file manually
uploaded = files.upload()

# Load dataset - Assuming the file is now uploaded
# ----> Get the actual filename from uploaded.keys() <----
filename = list(uploaded.keys())[0]
df = pd.read_csv(io.BytesIO(uploaded[filename]))  # Use io.BytesIO for in-memory file

# Encode categorical variables
# ... (rest of your code)
```

```
Choose files students_dataset.csv
```

- **students_dataset.csv**(text/csv) - 313 bytes, last modified: 24/04/2025 - 100% done

```
Saving students_dataset.csv to students_dataset (6).csv
```

17.Create a Prediction Function

```
# train_model.py
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import joblib
from google.colab import files
import io  # Make sure to import the io module

# ... (your code for loading and preprocessing data) ...

# Assuming X_train, X_test, y_train, y_test are defined
# Initialize and train the model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Save the trained model
joblib.dump(model, "fraud_model.pkl")

# ... (rest of your code) ...
```

```
['fraud_model.pkl']
```

18.Create the Gradio Interface

```
pip install gradio
```

```
Collecting gradio
  Downloading gradio-5.29.0-py3-none-any.whl.metadata (16 kB)
Collecting aiofiles<25.0,>=22.0 (from gradio)
  Downloading aiofiles-24.1.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.9.0)
Collecting fastapi<1.0,>=0.115.2 (from gradio)
  Downloading fastapi-0.115.12-py3-none-any.whl.metadata (27 kB)
Collecting ffmpy (from gradio)
  Downloading ffmpy-0.5.0-py3-none-any.whl.metadata (3.0 kB)
Collecting gradio-client==1.10.0 (from gradio)
  Downloading gradio_client-1.10.0-py3-none-any.whl.metadata (7.1 kB)
Collecting groovy~=0.1 (from gradio)
  Downloading groovy-0.1.2-py3-none-any.whl.metadata (6.1 kB)
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: huggingface-hub>=0.28.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.30.2)
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.1.6)
Requirement already satisfied: markupsafe<4.0,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.0.2)
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.0.2)
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.10.18)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from gradio) (24.2)
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.2.2)
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (11.2.1)
Requirement already satisfied: pydantic<2.12,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.11.4)
Collecting pydub (from gradio)
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting python-multipart>=0.0.18 (from gradio)
  Downloading python_multipart-0.0.20-py3-none-any.whl.metadata (1.8 kB)
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (6.0.2)
Collecting ruff>=0.9.3 (from gradio)
  Downloading ruff-0.11.9-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (25 kB)
Collecting safehttpx<0.2.0,>=0.1.6 (from gradio)
  Downloading safehttpx-0.1.6-py3-none-any.whl.metadata (4.2 kB)
Collecting semantic-version~=2.0 (from gradio)
  Downloading semantic_version-2.10.0-py2.py3-none-any.whl.metadata (9.7 kB)
Collecting starlette<1.0,>=0.40.0 (from gradio)
  Downloading starlette-0.46.2-py3-none-any.whl.metadata (6.2 kB)
Collecting tomlkit<0.14.0,>=0.12.0 (from gradio)
  Downloading tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.15.3)
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.13.2)
Collecting uvicorn>=0.14.0 (from gradio)
  Downloading uvicorn-0.34.2-py3-none-any.whl.metadata (6.5 kB)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.0->gradio) (2025.3.2)
Requirement already satisfied: websockets<16.0,>=10.0 in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.0->gradio)
```

```
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (3.10)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (1.3.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (2025.4.26)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (1.0.9)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.11/dist-packages (from httpcore==1.*->httpx>=0.24.1->gradio) (0.16
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (3.18.0)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (2.32.3)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (4.67.1
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2.9
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (2
```

```python
# train_model.py
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import joblib
from google.colab import files
import io  # Make sure to import the io module
import os # Import the os module

# ... (your code for loading and preprocessing data) ...

# Assuming X_train, X_test, y_train, y_test are defined
# Initialize and train the model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Save the trained model
joblib.dump(model, "fraud_model.pkl")

# ----> Create and save encoders if categorical features exist <----
# Initialize an empty dictionary to store encoders
encoders = {}
# Get a list of categorical features (replace with your actual categorical columns)
categorical_features = ['location', 'merchant'] # Replace with your actual categorical columns
# Loop through categorical features and create encoders
for feature in categorical_features:
    if feature in X_train.columns: # Check if the feature exists in your data
        le = LabelEncoder()
        le.fit(X_train[feature]) # Fit the encoder on training data
        encoders[feature] = le # Store the encoder in the dictionary
        # Optionally, transform the feature in your training and test data:
        # X_train[feature] = le.transform(X_train[feature])
        # X_test[feature] = le.transform(X_test[feature])

# Save the encoders using joblib
joblib.dump(encoders, "encoders.pkl")

# ... (rest of your code) ...
```

```
['encoders.pkl']
```