upload Dataset

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

```
Choose files credit_card…_dataset.csv
```

- **credit_card_fraud_dataset.csv**(text/csv) - 6405190 bytes, last modified: 13/05/2025 - 100% done

```
Saving credit_card_fraud_dataset.csv to credit_card_fraud_dataset.csv
```

Load the Dataset

```
import pandas as pd
df = pd.read_csv('credit_card_fraud_dataset.csv')
df.head()
```

	TransactionID	TransactionDate	Amount	MerchantID	TransactionType	Location	IsFraud
0	1	2024-04-03 14:15:35.462794	4189.27	688	refund	San Antonio	0
1	2	2024-03-19 13:20:35.462824	2659.71	109	refund	Dallas	0
2	3	2024-01-08 10:08:35.462834	784.00	394	purchase	New York	0
3	4	2024-04-13 23:50:35.462850	3514.40	944	purchase	Philadelphia	0
4	5	2024-07-12 18:51:35.462858	369.07	475	purchase	Phoenix	0

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

Data Exploration

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

```
Shape: (100000, 7)
```

```
# Info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 7 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   TransactionID    100000 non-null  int64
 1   TransactionDate  100000 non-null  object
 2   Amount           100000 non-null  float64
 3   MerchantID       100000 non-null  int64
 4   TransactionType  100000 non-null  object
 5   Location         100000 non-null  object
 6   IsFraud          100000 non-null  int64
dtypes: float64(1), int64(3), object(3)
memory usage: 5.3+ MB
```

```
# Summary statistics
df.describe()
```

	TransactionID	Amount	MerchantID	IsFraud
count	100000.000000	100000.000000	100000.000000	100000.000000
mean	50000.500000	2497.092666	501.676070	0.010000
std	28867.657797	1442.415999	288.715868	0.099499
min	1.000000	1.050000	1.000000	0.000000
25%	25000.750000	1247.955000	252.000000	0.000000
50%	50000.500000	2496.500000	503.000000	0.000000
75%	75000.250000	3743.592500	753.000000	0.000000
max	100000.000000	4999.770000	1000.000000	1.000000

Check for Missing Values and Duplicates

```
# Missing values
print("Missing Values:\n", df.isnull().sum())
```

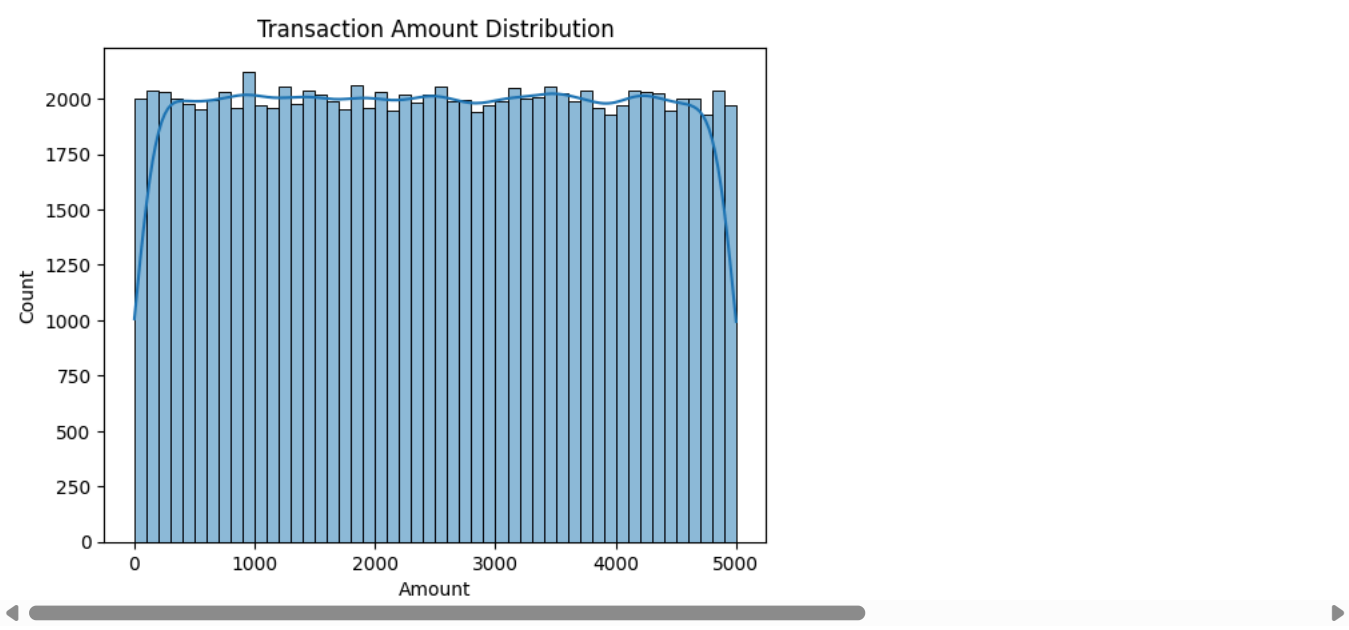
```
Missing Values:
 TransactionID      0
TransactionDate    0
Amount             0
MerchantID         0
TransactionType    0
Location           0
IsFraud            0
dtype: int64
```

```
# Duplicates
print("Duplicate Rows:", df.duplicated().sum())
```

```
Duplicate Rows: 0
```

Visualize a Few Features

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

# Distribution of transaction amount
sns.histplot(df['Amount'], bins=50, kde=True)
plt.title('Transaction Amount Distribution')
plt.show()
```




```
# Fraud count
sns.countplot(data=df, x='IsFraud')
plt.title('Fraud vs Non-Fraud')
plt.show()
```


```
# Boxplot of amount by fraud
sns.boxplot(x='IsFraud', y='Amount', data=df)
plt.title('Amount by Fraud Status')
plt.show()
```

Identify Target and Features

```
# Target variable
target = 'IsFraud'


# Drop unneeded columns
features = df.drop(['TransactionID', 'TransactionDate', target], axis=1).columns.tolist()

print("Target:", target)
print("Features:", features)
```

```
Target: IsFraud
Features: ['Amount', 'MerchantID', 'TransactionType', 'Location']
```

Convert Categorical Columns to Numerical

```
# Identify categorical columns
categorical_cols = df[features].select_dtypes(include='object').columns
print("Categorical Columns:", categorical_cols.tolist())
```

```
Categorical Columns: ['TransactionType', 'Location']
```

One-Hot Encoding

```
df_encoded = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X = df_encoded.drop(['TransactionID', 'TransactionDate', target], axis=1)
y = df_encoded[target]

X_scaled = scaler.fit_transform(X)
```

Train-Test Split

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42, stratify=y)
```

Model Building

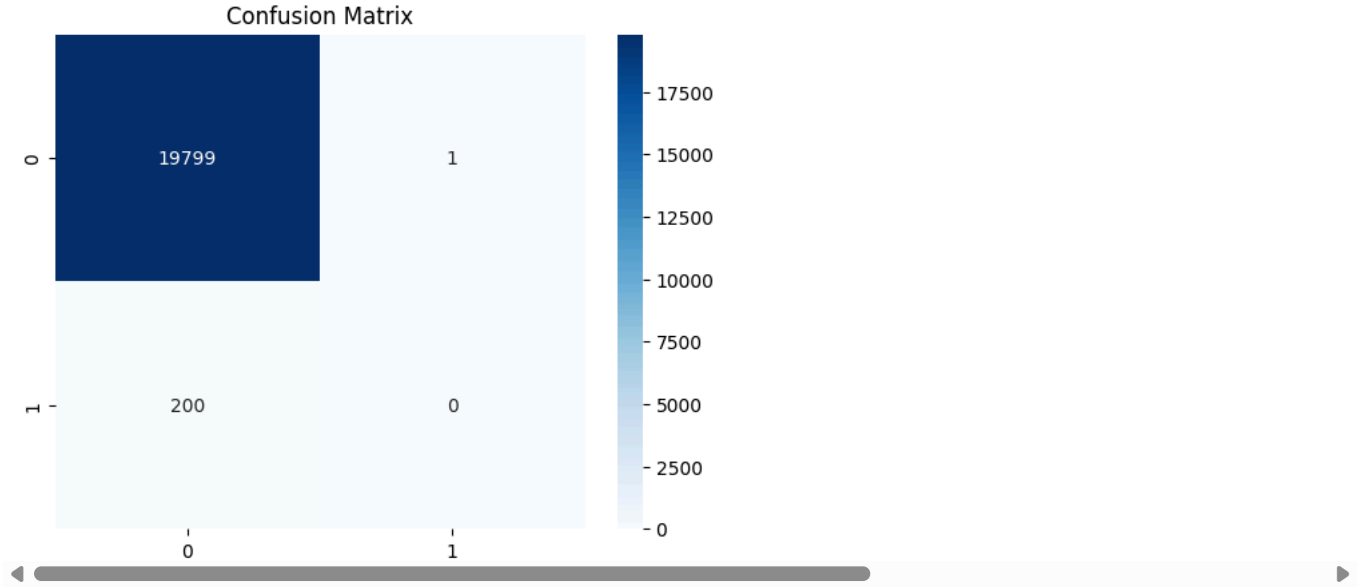
```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Evaluation

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy: 0.98995

Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     19800
           1       0.00      0.00      0.00       200

    accuracy                           0.99     20000
   macro avg       0.49      0.50      0.50     20000
weighted avg       0.98      0.99      0.99     20000
```



Make Predictions from New Input

```
# Example new input
new_input = {
    'Amount': 100.0,
    'MerchantID': 123,
    'TransactionType_purchase': 1,
    'TransactionType_refund': 0,
    'Location_Dallas': 0,
    'Location_New York': 1,
    # Add remaining dummy variables set to 0 or 1 as needed
}
```

```
#Create a template row
template_input = pd.DataFrame([np.zeros(len(X.columns))], columns=X.columns)
```

```
#  Update with real values
for key, value in new_input.items():
    if key in template_input.columns:
        template_input[key] = value
    else:
        print(f"⚠️ Warning: '{key}' not in training features, skipping.")
```

```
⚠️ Warning: 'TransactionType_purchase' not in training features, skipping.
```

```
# Scale input using the fitted scaler
new_df_scaled = scaler.transform(template_input)
```

```
#  Predict
prediction = model.predict(new_df_scaled)
print("Prediction:", "Fraud" if prediction[0] == 1 else "Not Fraud")
```

```
Prediction: Not Fraud
```

Convert to DataFrame and Encode

```
#  Get the training columns from X
training_columns = X.columns


# Step 2: Create a zero-filled DataFrame with correct structure
input_aligned = pd.DataFrame([np.zeros(len(training_columns))], columns=training_columns)


#  Update only the matching keys in new_input
for key, value in new_input.items():
    if key in input_aligned.columns:
        input_aligned[key] = value
    else:
        print(f"⚠️ Skipping '{key}' - not in training features.")
```

```
⚠️ Skipping 'TransactionType_purchase' - not in training features.
```

```
#  Scale and predict
input_scaled = scaler.transform(input_aligned)
prediction = model.predict(input_scaled)

print("Prediction:", "Fraud" if prediction[0] == 1 else "Not Fraud")
```

```
Prediction: Not Fraud
```

Install Gradio

```
!pip install gradio
```

```
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==1.10.1->gradio) (2025.3.2)
Requirement already satisfied: websockets<16.0,>=10.0 in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.1->gra
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) (
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.
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio)
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)
```

```
Installing collected packages: pydub, uvicorn, tomlkit, semantic-version, ruff, python-multipart, groovy, ffmpy, aiofiles, starle
Successfully installed aiofiles-24.1.0 fastapi-0.115.12 ffmpy-0.5.0 gradio-5.29.1 gradio-client-1.10.1 groovy-0.1.2 pydub-0.25.1
```

Deployment – Interactive App (Gradio)

```
import gradio as gr

def predict_fraud(Amount, MerchantID, TransactionType, Location):
    # Rebuild input
    input_data = {col: 0 for col in X.columns}
    input_data['Amount'] = Amount
    input_data['MerchantID'] = MerchantID
    if f'TransactionType_{TransactionType}' in input_data:
        input_data[f'TransactionType_{TransactionType}'] = 1
    if f'Location_{Location}' in input_data:
        input_data[f'Location_{Location}'] = 1

    df_input = pd.DataFrame([input_data])
    scaled_input = scaler.transform(df_input)
    pred = model.predict(scaled_input)
    return "Fraud" if pred[0] == 1 else "Not Fraud"

gr.Interface(
    fn=predict_fraud,
    inputs=[
        gr.Number(label="Transaction Amount"),
        gr.Number(label="Merchant ID"),
        gr.Dropdown(df['TransactionType'].unique().tolist(), label="Transaction Type"),
        gr.Dropdown(df['Location'].unique().tolist(), label="Location")
    ],
    outputs="text",
    title="💳 Credit Card Fraud Predictor"
).launch()
```

```
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled. Automatica

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://bc3180a2ccab25d7c0.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working
```