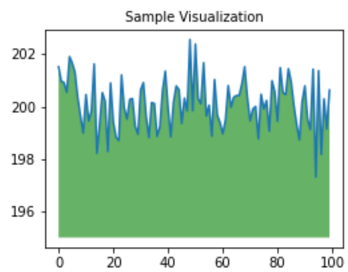
Start coding or generate with AI.

```
import numpy as np
import IPython.display as display
from matplotlib import pyplot as plt
import io
import base64

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

fig = plt.figure(figsize=(4, 3), facecolor='w')
plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)
plt.title("Sample Visualization", fontsize=10)

data = io.BytesIO()
plt.savefig(data)
image = F"data:image/png;base64,{base64.b64encode(data.getvalue()).decode()}"
alt = "Sample Visualization"
display.display(display.Markdown(F"""![{alt}]({image})"""))
plt.close(fig)
```



To learn more about accelerating pandas on Colab, see the 10 minute guide or US stock market data analysis demo.

```
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.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from imblearn.over_sampling import SMOTE



# Load dataset
file_path = '/content/creditcard.csv'  # Use your file path in Colab
df = pd.read_csv(file_path)


# Explore dataset
print(df.head())
print(df.info())
print(df['Class'].value_counts())  # Check class distribution
```

```
         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120901 entries, 0 to 120900
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    120901 non-null  int64
 1   V1      120901 non-null  float64
 2   V2      120901 non-null  float64
 3   V3      120901 non-null  float64
 4   V4      120901 non-null  float64
 5   V5      120901 non-null  float64
 6   V6      120901 non-null  float64
 7   V7      120901 non-null  float64
 8   V8      120901 non-null  float64
 9   V9      120901 non-null  float64
 10  V10     120901 non-null  float64
 11  V11     120901 non-null  float64
 12  V12     120901 non-null  float64
 13  V13     120901 non-null  float64
 14  V14     120901 non-null  float64
 15  V15     120901 non-null  float64
 16  V16     120901 non-null  float64
 17  V17     120901 non-null  float64
 18  V18     120901 non-null  float64
 19  V19     120901 non-null  float64
 20  V20     120901 non-null  float64
 21  V21     120901 non-null  float64
 22  V22     120901 non-null  float64
 23  V23     120901 non-null  float64
 24  V24     120901 non-null  float64
 25  V25     120900 non-null  float64
 26  V26     120900 non-null  float64
 27  V27     120900 non-null  float64
 28  V28     120900 non-null  float64
 29  Amount  120900 non-null  float64
 30  Class   120900 non-null  float64
dtypes: float64(30), int64(1)
memory usage: 28.6 MB
None
Class
0.0    120651
1.0       249
Name: count, dtype: int64
```

```
# Splitting features and target
X = df.drop(columns=['Class'])
y = df['Class']


# Handling imbalanced data using SMOTE
smote = SMOTE(random_state=42)

# Drop rows with NaN values in the target variable 'Class'
df = df.dropna(subset=['Class'])

# Update X and y after removing NaN values
X = df.drop(columns=['Class'])
y = df['Class']

X_resampled, y_resampled = smote.fit_resample(X, y)



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


# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)



# Subset the data for faster training
X_train_subset = X_train[:10000]  # Use the first 10000 samples for training
y_train_subset = y_train[:10000]

model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train_subset, y_train_subset)
```

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

```
# Predictions
y_pred = model.predict(X_test)


# Evaluation
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy Score: 0.9951513644557718
Classification Report:
               precision    recall  f1-score   support

         0.0       0.99      1.00      1.00     23990
         1.0       1.00      0.99      1.00     24271

    accuracy                           1.00     48261
   macro avg       1.00      1.00      1.00     48261
weighted avg       1.00      1.00      1.00     48261

Confusion Matrix:
 [[23962    28]
 [  206 24065]]
```

```
# Plot confusion matrix
plt.figure(figsize=(6,4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```