```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
# Load the cleaned dataset
df = pd.read_csv('/content/german_credit_data.csv')

# Display the first few rows of the dataset
df.head()
```

	Unnamed: 0	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose
0	0	67	male	2	own	NaN	little	1169	6	radio/TV
1	1	22	female	2	own	little	moderate	5951	48	radio/TV
2	2	49	male	1	own	little	NaN	2096	12	education
3	3	45	male	2	free	little	little	7882	42	furniture/equipment
4	4	53	male	2	free	little	little	4870	24	car

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

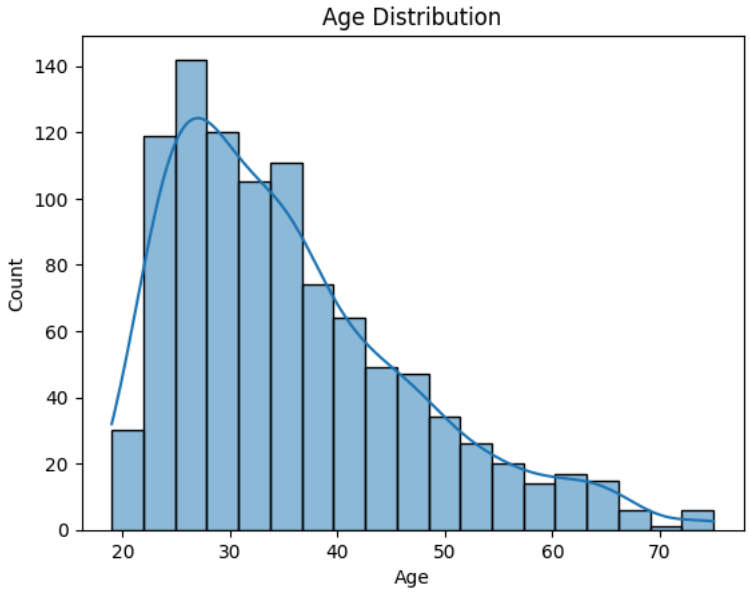
```
# Handle missing values by filling with a placeholder or a statistical value
df = df.copy()  # Ensure df is a copy and avoid chained assignment issues
df['Saving accounts'] = df['Saving accounts'].fillna('unknown')
df['Checking account'] = df['Checking account'].fillna('unknown')

# Encode categorical variables
label_encoders = {}
for column in ['Sex', 'Housing', 'Saving accounts', 'Checking account', 'Purpose']:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

# Display the cleaned and encoded dataset
df.head()
```

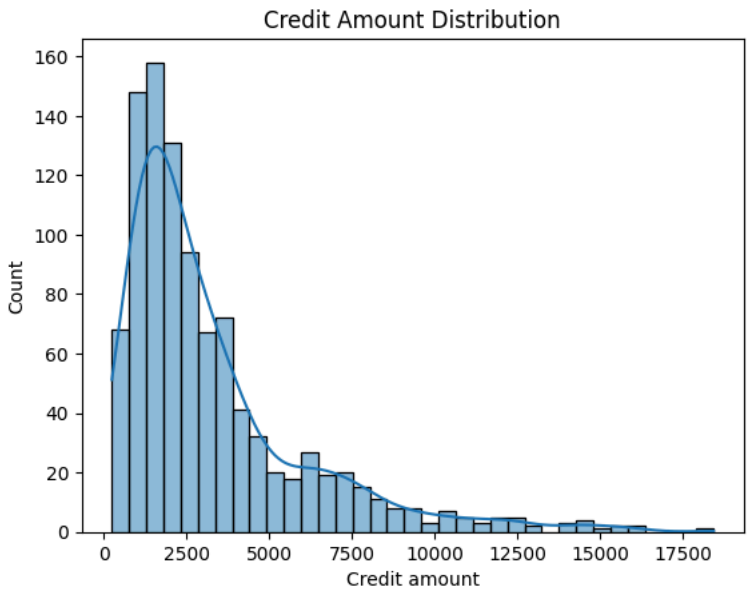
	Unnamed: 0	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose
0	0	67	1	2	1	4	0	1169	6	5
1	1	22	0	2	1	0	1	5951	48	5
2	2	49	1	1	1	0	3	2096	12	3
3	3	45	1	2	0	0	0	7882	42	4
4	4	53	1	2	0	0	0	4870	24	1

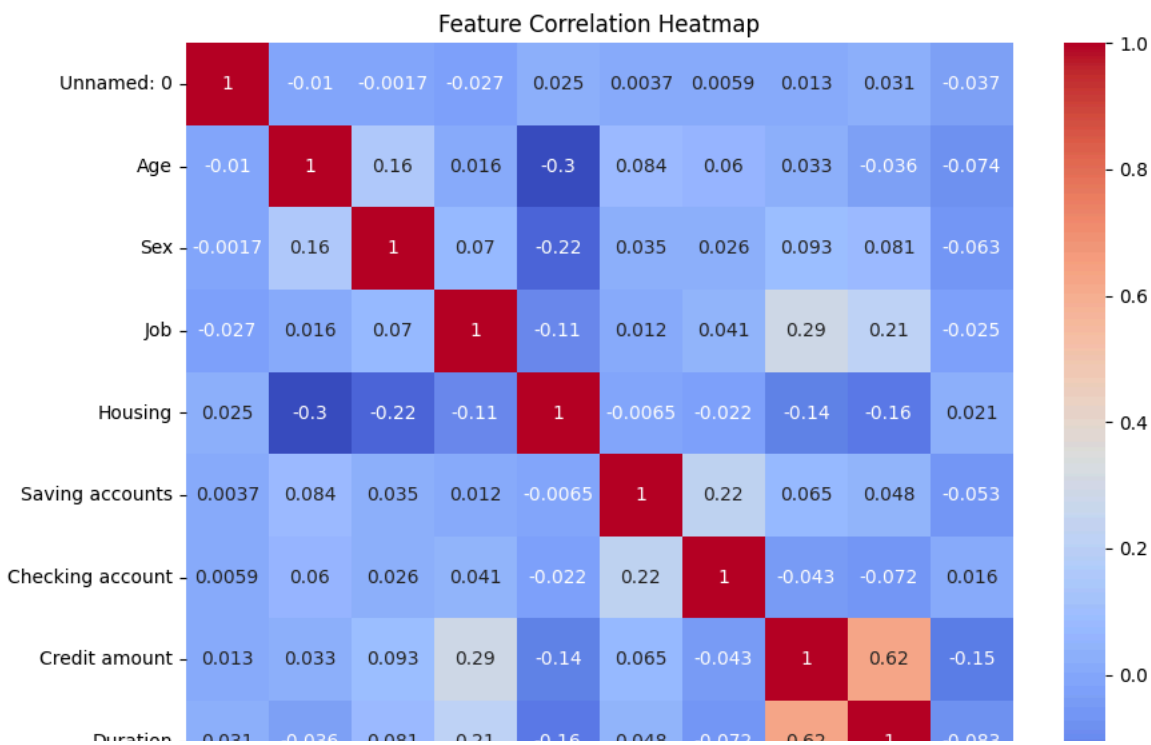
Next steps: Generate code with df | View recommended plots | New interactive sheet

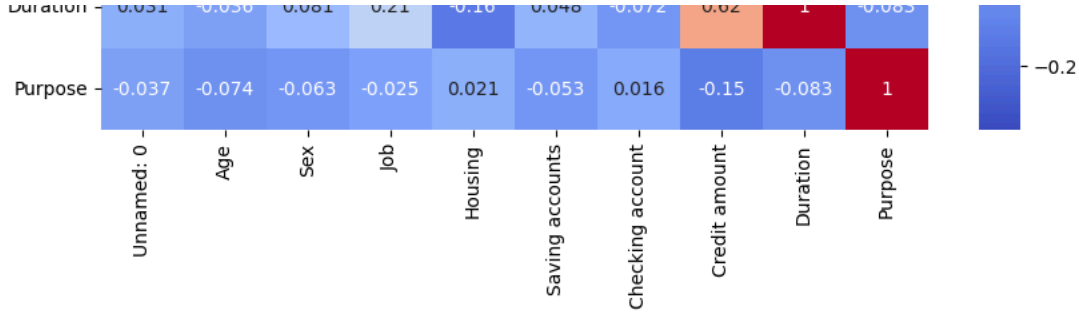
```
# Ensure that any infinite values are converted to NaN before analysis
df = df.replace([np.inf, -np.inf], np.nan)

# Age distribution
sns.histplot(df['Age'], kde=True)
plt.title('Age Distribution')
plt.show()

# Credit amount distribution
sns.histplot(df['Credit amount'], kde=True)
plt.title('Credit Amount Distribution')
plt.show()

# Correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
```

```
plt.title('Feature Correlation Heatmap')
plt.show()
```


Age Distribution
Count
Age
0
20
40
60
80
100
120
140
20
30
40
50
60
70


Credit Amount Distribution
Count
Credit amount
0
20
40
60
80
100
120
140
160
0
2500
5000
7500
10000
12500
15000
17500


Feature Correlation Heatmap
Unnamed: 0 1 -0.01 -0.0017 -0.027 0.025 0.0037 0.0059 0.013 0.031 -0.037
Age -0.01 1 0.16 0.016 -0.3 0.084 0.06 0.033 -0.036 -0.074
Sex -0.0017 0.16 1 0.07 -0.22 0.035 0.026 0.093 0.081 -0.063
Job -0.027 0.016 0.07 1 -0.11 0.012 0.041 0.29 0.21 -0.025
Housing 0.025 -0.3 -0.22 -0.11 1 -0.0065 -0.022 -0.14 -0.16 0.021
Saving accounts 0.0037 0.084 0.035 0.012 -0.0065 1 0.22 0.065 0.048 -0.053
Checking account 0.0059 0.06 0.026 0.041 -0.022 0.22 1 -0.043 -0.072 0.016
Credit amount 0.013 0.033 0.093 0.29 -0.14 0.065 -0.043 1 0.62 -0.15
1.0
0.8
0.6
0.4
0.2
0.0




```
# Define features and target
X = df.drop(columns=['Credit amount'])  # Assuming 'Credit amount' as target
y = df['Credit amount'] > df['Credit amount'].median()  # Binary classification (High/Low Credit Amount)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

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


# Train a Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)


# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
Accuracy: 0.74
Confusion Matrix:
[[124  31]
 [ 47  98]]
Classification Report:
              precision    recall  f1-score   support

       False       0.73      0.80      0.76       155
        True       0.76      0.68      0.72       145

    accuracy                           0.74       300
   macro avg       0.74      0.74      0.74       300
weighted avg       0.74      0.74      0.74       300
```

```
# Add predictions to the DataFrame
df_test = df.iloc[y_test.index].copy()  # Create a copy of the test dataset rows from the original DataFrame
df_test['Prediction'] = y_pred
df_test['Actual'] = y_test.values

# True Negatives: Actual = 0 (low risk), Prediction = 0 (low risk)
true_negatives = df_test[(df_test['Actual'] == 0) & (df_test['Prediction'] == 0)]

# True Positives: Actual = 1 (high risk), Prediction = 1 (high risk)
true_positives = df_test[(df_test['Actual'] == 1) & (df_test['Prediction'] == 1)]

# Display True Negatives
print("True Negatives (Low risk correctly predicted as low risk):")
display(true_negatives)
```