```python
import pandas as pd
import numpy as np

df = pd.read_csv('/content/sample_data/dataset.csv')

df.head()

df.describe()

df.isnull().sum()

df = df.drop_duplicates()

checking_status_mapping = {'<0': 0, '0<=X<200': 1, 'no checking': 2, '>=200':4}

# Replace the values in the 'checking_status' column with numbers using the defined mapping
df['checking_status'] = df['checking_status'].replace(checking_status_mapping)

credit_history_mapping = {'critical/other existing credit': 0, 'existing paid':1, 'delayed previously':2, 'no credits/all paid':3,'all paid':4}

# Replace the values in the 'credit_history' column with numbers using the defined mapping
df['credit_history'] = df['credit_history'].replace(credit_history_mapping)


savings_status_mapping = {'no known savings': 0, '<100': 1, '500<=X<1000': 2, '>=1000': 3, '100<=X<500': 4}

# Replace the values in the 'savings_status' column with numbers using the defined mapping
df['savings_status'] = df['savings_status'].replace(savings_status_mapping)


other_payment_plans_mapping = {'none': 0, 'bank': 1, 'stores':2}

# Replace the values in the 'other_payment_plans' column with numbers using the defined mapping
df['other_payment_plans'] = df['other_payment_plans'].replace(other_payment_plans_mapping)

property_magnitude_mapping = {'real estate': 1, 'life insurance': 2, 'no known property': 0, 'car':3}

# Replace the values in the 'property_magnitude' column with numbers using the defined mapping
df['property_magnitude'] = df['property_magnitude'].replace(property_magnitude_mapping)


other_parties_mapping = {'none': 0, 'guarantor': 1, 'co-applicant': 2, 'co applicant':2}

# Replace the values in the 'other_parties' column with numbers using the defined mapping
df['other_parties'] = df['other_parties'].replace(other_parties_mapping)

personal_status_mapping = {'male single': 0, 'female div/dep/mar': 1, 'male div/sep': 2, 'male mar/wid':3}

# Replace the values in the 'personal_status' column with numbers using the defined mapping
df['personal_status'] = df['personal_status'].replace(personal_status_mapping)

employment_mapping = {'unemployed': 0, '>=7': 1, '1<=X<4': 2, '4<=X<7':3, '<1':4}

# Replace the values in the 'employment' column with numbers using the defined mapping
df['employment'] = df['employment'].replace(employment_mapping)

# Define the mapping of categories to numbers
job_mapping = {'skilled': 1, 'unskilled resident': 0, 'high qualif/self emp/mgmt': 3, 'unemp/unskilled non res': 1}

# Replace the values in the 'job' column with numbers using the defined mapping
df['job'] = df['job'].replace(job_mapping)

house_mapping = {'own': 1, 'rent': 0, 'for free':2}

# Replace the values in the 'housing' column with numbers using the defined mapping
df['housing'] = df['housing'].replace(house_mapping)

# df['own_telephone', 'foreign_worker', 'class'].unique()
print(df['own_telephone'].unique())
print(df['foreign_worker'].unique())
print(df['class'].unique())

own_telephone_mapping = {'yes': 1, 'none': 0}
foreign_worker_mapping = {'yes': 1, 'no': 0}
class_mapping = {'good': 1, 'bad': 0}

df['own_telephone'] = df['own_telephone'].replace(own_telephone_mapping)
df['foreign_worker'] = df['foreign_worker'].replace(foreign_worker_mapping)
df['class'] = df['class'].replace(class_mapping)

import seaborn as sns
import matplotlib.pyplot as plt

# Assuming df is your DataFrame
# Compute the correlation matrix
corr_matrix = df.corr()

# Set up the matplotlib figure
plt.figure(figsize=(12, 10))

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True, square=True, linewidths=.5)

# Set the title
plt.title('Correlation Matrix')
```

```python
# Show the plot
plt.show()

class_corr = corr_matrix['class']

# Sort the correlations with the target variable 'class'
class_corr_sorted = class_corr.sort_values(ascending=False)

# Print the highly correlated features with 'class'
print("Features highly correlated with 'class':")
print(class_corr_sorted)


# Selecting the features (X) and target (y)
X = df.drop(columns=['class'])
y = df['class']
from sklearn.model_selection import train_test_split

# Splitting the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

# Logistic Regression
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
y_pred_log_reg = log_reg.predict(X_test)

# Evaluation
print("Logistic Regression")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_log_reg))
print("Classification Report:\n", classification_report(y_test, y_pred_log_reg))
print("Accuracy: ", accuracy_score(y_test, y_pred_log_reg))
from sklearn.neighbors import KNeighborsClassifier

# KNN Classifier
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)

# Evaluation
print("K-Nearest Neighbors")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_knn))
print("Classification Report:\n", classification_report(y_test, y_pred_knn))
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
from sklearn.svm import SVC

# SVM with linear kernel
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train, y_train)
y_pred_svm_linear = svm_linear.predict(X_test)

# Evaluation
print("SVM with Linear Kernel")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svm_linear))
print("Classification Report:\n", classification_report(y_test, y_pred_svm_linear))
print("Accuracy:", accuracy_score(y_test, y_pred_svm_linear))

# SVM with RBF kernel
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_train, y_train)
y_pred_svm_rbf = svm_rbf.predict(X_test)

# Evaluation
print("SVM with RBF Kernel")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svm_rbf))
print("Classification Report:\n", classification_report(y_test, y_pred_svm_rbf))
print("Accuracy:", accuracy_score(y_test, y_pred_svm_rbf))
# Collecting accuracies
accuracies = {
    'Logistic Regression': accuracy_score(y_test, y_pred_log_reg),
    'K-Nearest Neighbors': accuracy_score(y_test, y_pred_knn),
    'SVM with Linear Kernel': accuracy_score(y_test, y_pred_svm_linear),
    'SVM with RBF Kernel': accuracy_score(y_test, y_pred_svm_rbf)
}

# Finding the best model
best_model = max(accuracies, key=accuracies.get)
print(f"The model with the best accuracy is {best_model} with an accuracy of {accuracies[best_model]:.2f}")

#proceed
```