```
# Import libraries
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, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
import joblib

# Load the dataset
data = pd.read_csv("churn_data.csv")

# Display the first few rows
data.head()

# Check for missing values
print("Missing values per column:\n", data.isnull().sum())

# Fill or drop missing values (example: filling with mean)
data.fillna(data.mean(), inplace=True)

# Exploratory Data Analysis (EDA)
sns.countplot(x='Churn', data=data)
plt.title("Churn Distribution")
plt.show()
```

```
# Visualize correlations
corr_matrix = data.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Correlation Matrix")
plt.show()

# Feature engineering (convert categorical variables to dummy variables)
data = pd.get_dummies(data, drop_first=True)

# Define features and target
X = data.drop("Churn", axis=1)
y = data["Churn"]

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

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

# Model training
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
```

```
# Model evaluation
y_pred = rf.predict(X_test)
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

# ROC curve
y_pred_proba = rf.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f"ROC curve (area = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristic")
plt.legend(loc="lower right")
plt.show()

# Hyperparameter tuning using GridSearchCV
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

```
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=3, n_jobs=-1,
verbose=2)
grid_search.fit(X_train, y_train)

# Best parameters and evaluation
print("Best Parameters:\n", grid_search.best_params_)
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)
print("Optimized Classification Report:\n", classification_report(y_test, y_pred_best))

# Save the model and scaler
joblib.dump(best_model, "churn_prediction_model.pkl")
joblib.dump(scaler, "scaler.pkl")

# Example: Loading the model and making predictions
loaded_model = joblib.load("churn_prediction_model.pkl")
loaded_scaler = joblib.load("scaler.pkl")
new_data = pd.DataFrame([[25, 50, 3]], columns=X.columns)  # Example new customer data
new_data_scaled = loaded_scaler.transform(new_data)
prediction = loaded_model.predict(new_data_scaled)
print("Prediction for new customer:", prediction)
```