```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.cluster import KMeans
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.ensemble import VotingRegressor, VotingClassifier
from sklearn.metrics import accuracy_score, mean_squared_error
from sklearn.metrics import silhouette_score
from surprise import Dataset, Reader
from surprise.model_selection import train_test_split as surprise_train_test_split
from surprise import KNNBasic, KNNWithMeans, KNNWithZScore, KNNBaseline

# Load CSV file into a DataFrame
filename = 'example.csv'
data = pd.read_csv(filename)

# Preprocessing
# Fill Null Values
data.fillna(data.mean(), inplace=True)

# Split data into features (X) and target (y)
X = data.drop(columns=['target_column'])
y = data['target_column']

# Standard Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Linear Regression
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
lr_predictions = lr_model.predict(X_test)
lr_rmse = mean_squared_error(y_test, lr_predictions, squared=False)
print("Linear Regression RMSE:", lr_rmse)

# Support Vector Machine (SVM)
svm_model = SVR()
svm_model.fit(X_train, y_train)
svm_predictions = svm_model.predict(X_test)
svm_rmse = mean_squared_error(y_test, svm_predictions, squared=False)
print("SVM RMSE:", svm_rmse)

# Decision Tree
dt_model = DecisionTreeRegressor()
dt_model.fit(X_train, y_train)
dt_predictions = dt_model.predict(X_test)
dt_rmse = mean_squared_error(y_test, dt_predictions, squared=False)
print("Decision Tree RMSE:", dt_rmse)

# K-Means Clustering
kmeans_model = KMeans(n_clusters=3)
kmeans_model.fit(X_train)
silhouette_score_kmeans = silhouette_score(X_train, kmeans_model.labels_)
print("K-Means Silhouette Score:", silhouette_score_kmeans)
```

```
# K-Nearest Neighbors (KNN) for Regression
knn_model = KNeighborsRegressor()
knn_model.fit(X_train, y_train)
knn_predictions = knn_model.predict(X_test)
knn_rmse = mean_squared_error(y_test, knn_predictions, squared=False)
print("KNN Regression RMSE:", knn_rmse)

# K-Nearest Neighbors (KNN) for Classification
knn_classifier_model = KNeighborsClassifier()
knn_classifier_model.fit(X_train, y_train)
knn_classifier_predictions = knn_classifier_model.predict(X_test)
knn_classifier_accuracy = accuracy_score(y_test, knn_classifier_predictions)
print("KNN Classifier Accuracy:", knn_classifier_accuracy)

# Random Forest for Regression
rf_model = RandomForestRegressor()
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
rf_rmse = mean_squared_error(y_test, rf_predictions, squared=False)
print("Random Forest Regression RMSE:", rf_rmse)

# Random Forest for Classification
rf_classifier_model = RandomForestClassifier()
rf_classifier_model.fit(X_train, y_train)
rf_classifier_predictions = rf_classifier_model.predict(X_test)
rf_classifier_accuracy = accuracy_score(y_test, rf_classifier_predictions)
print("Random Forest Classifier Accuracy:", rf_classifier_accuracy)

# Ensemble Models (Voting Regressor)
ensemble_model = VotingRegressor([('lr', lr_model), ('svm', svm_model), ('rf', rf_model)])
ensemble_model.fit(X_train, y_train)
```

```
ensemble_predictions = ensemble_model.predict(X_test)
ensemble_rmse = mean_squared_error(y_test, ensemble_predictions, squared=False)
print("Ensemble Model RMSE:", ensemble_rmse)

# Ensemble Models (Voting Classifier)
ensemble_classifier_model = VotingClassifier([('knn', knn_classifier_model), ('rf', rf_classifier_model)])
ensemble_classifier_model.fit(X_train, y_train)
ensemble_classifier_predictions = ensemble_classifier_model.predict(X_test)
ensemble_classifier_accuracy = accuracy_score(y_test, ensemble_classifier_predictions)
print("Ensemble Classifier Accuracy:", ensemble_classifier_accuracy)

# Collaborative Filtering (Clustering)
# Load data for collaborative filtering
reader = Reader(rating_scale=(1, 5))
surprise_data = Dataset.load_from_df(data[['user_id', 'item_id', 'rating']], reader)

# Split the data into training and testing sets
surprise_trainset, surprise_testset = surprise_train_test_split(surprise_data, test_size=0.2, random_state=42)

# KNN Collaborative Filtering
knn_collab_model = KNNBasic()
knn_collab_model.fit(surprise_trainset)
knn_collab_predictions = knn_collab_model.test(surprise_testset)
knn_collab_rmse = accuracy.rmse(knn_collab_predictions)
print("KNN Collaborative Filtering RMSE:", knn_collab_rmse)

# KNN Collaborative Filtering with Mean Centering
knn_mean_collab_model = KNNWithMeans()
knn_mean_collab_model.fit(surprise_trainset)
knn_mean_collab_predictions = knn_mean_collab_model.test(surprise_testset)
```

```
knn_mean_collab_rmse = accuracy.rmse(knn_mean_collab_predictions)
print("KNN Collaborative Filtering with Mean Centering RMSE:", knn_mean_collab_rmse)

# KNN Collaborative Filtering with Z-score Normalization
knn_zscore_collab_model = KNNWithZScore()
knn_zscore_collab_model.fit(surprise_trainset)
knn_zscore_collab_predictions = knn_zscore_collab_model.test(surprise_testset)
knn_zscore_collab_rmse = accuracy.rmse(knn_zscore_collab_predictions)
print("KNN Collaborative Filtering with Z-score Normalization RMSE:", knn_zscore_collab_rmse)

# KNN Collaborative Filtering with Baseline Prediction
knn_baseline_collab_model = KNNBaseline()
knn_baseline_collab_model.fit(surprise_trainset)
knn_baseline_collab_predictions = knn_baseline_collab_model.test(surprise_testset)
knn_baseline_collab_rmse = accuracy.rmse(knn_baseline_collab_predictions)
print("KNN Collaborative Filtering with Baseline Prediction RMSE:", knn_baseline_collab_rmse)

# Metrics
lr_accuracy = accuracy_score(y_test, lr_predictions)
svm_accuracy = accuracy_score(y_test, svm_predictions)
dt_accuracy = accuracy_score(y_test, dt_predictions)
kmeans_accuracy = silhouette_score_kmeans
knn_accuracy = accuracy_score(y_test, knn_predictions)
rf_accuracy = accuracy_score(y_test, rf_predictions)
ensemble_accuracy = accuracy_score(y_test, ensemble_classifier_predictions)
knn_collab_rmse = accuracy.rmse(knn_collab_predictions)

# Visualization
# Scatter plot
plt.scatter(X_train['feature1'], X_train['feature2'], c=kmeans_model.labels_, cmap='viridis')
plt.scatter(kmeans_model.cluster_centers_[:, 0], kmeans_model.cluster_centers_[:, 1], c='red', marker='x', s=200, label='Cluster Centers')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Scatter Plot for Clustering')
plt.legend()
plt.show()

# Confusion Matrix
cm_viz = ConfusionMatrix(ensemble_classifier_model, labels=np.unique(y))
cm_viz.score(X_test, y_test)
cm_viz.poof()

# ROC Curve
y_prob = ensemble_classifier_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
plt.plot(fpr, tpr, label='ROC Curve')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```