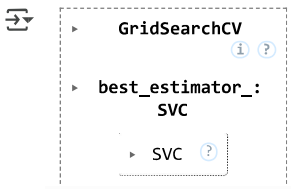
```
import numpy as np
import pandas as pd
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score
from scipy import stats


# Load data
df = pd.read_csv("diabetes.csv")

# Add feature engineering
df['Glucose_BMI'] = df['Glucose'] * df['BMI']
df['Age_Glucose'] = df['Age'] * df['Glucose']


# Remove outliers
z_scores = np.abs(stats.zscore(df))
df_clean = df[(z_scores < 3).all(axis=1)]


# Prepare data
X = df_clean.drop(columns=['Outcome'])
y = df_clean['Outcome']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)


# Grid search for best parameters
param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['rbf', 'linear'],
    'gamma': ['scale', 'auto', 0.1]
}
svm = SVC(random_state=42)
grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
```




```
# Evaluate best model
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)


print("Best parameters:", grid_search.best_params_)
print(f"Improved Model Accuracy: {accuracy * 100:.2f}%")
```

```
Best parameters: {'C': 1, 'gamma': 'scale', 'kernel': 'linear'}
Improved Model Accuracy: 74.26%
```

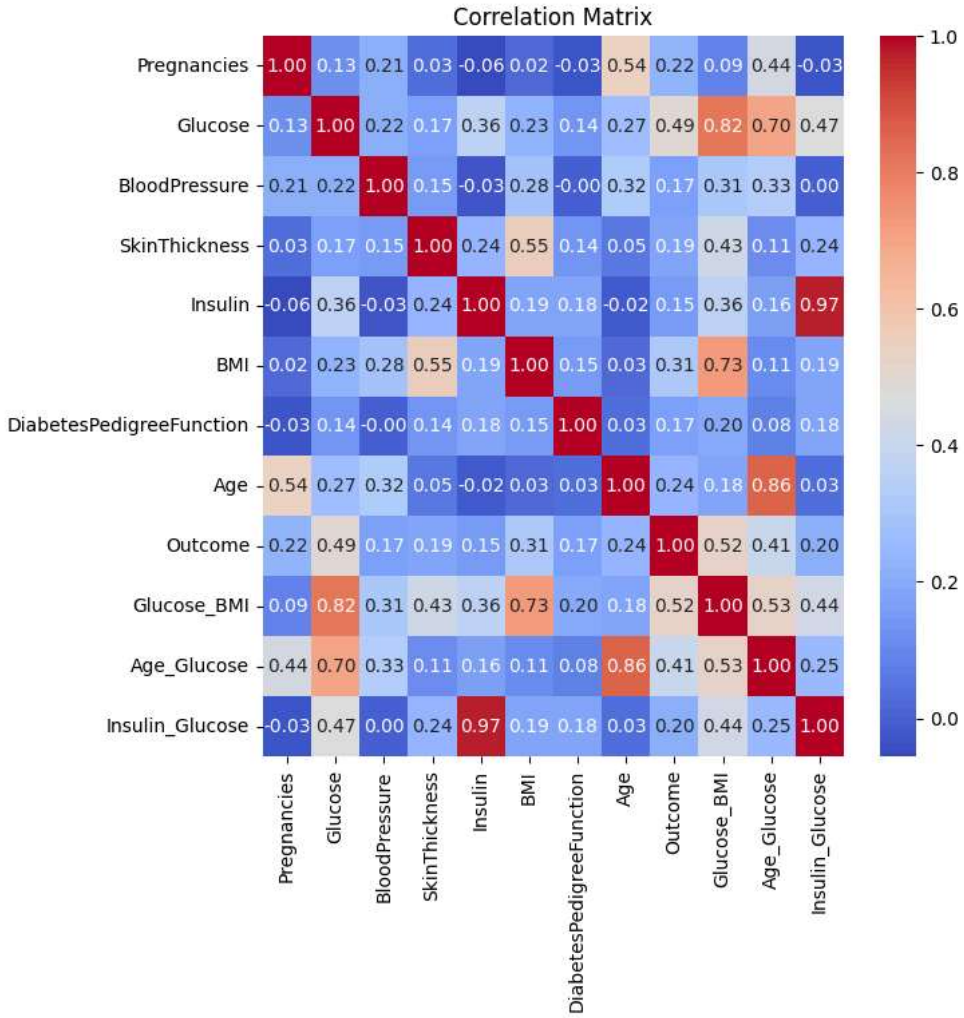
```
# Test case
test_case = np.array([[6, 148, 72, 35, 0, 33.6, 0.627, 50, 148*33.6, 50*148]])
test_scaled = scaler.transform(test_case)
prediction = best_model.predict(test_scaled)[0]
print("\nTest Case Prediction:", "Diabetic" if prediction == 1 else "Non-Diabetic")
```

```
Test Case Prediction: Diabetic
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but StandardScaler was
  warnings.warn(
```

USING XGBOOST

```
import numpy as np
import pandas as pd
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.feature_selection import RFE
from imblearn.over_sampling import SMOTE
from scipy import stats
from sklearn.ensemble import VotingClassifier
from xgboost import XGBClassifier
```

```
# Load data
df = pd.read_csv("diabetes.csv")


for column in ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']:
    median = df[column].median()
    df[column] = df[column].replace(0, median)

# Correlation matrix
correlation_matrix = df.corr()

# Visualize with a heatmap
plt.figure(figsize=(7, 7))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Matrix")
plt.show()

# Correlation with target
print("Correlation with Outcome:")
print(correlation_matrix['Outcome'].sort_values(ascending=False))
```




```
Correlation with Outcome:
Outcome                     1.000000
Glucose_BMI                 0.519938
Glucose                     0.492782
Age_Glucose                 0.410235
BMI                         0.312249
Age                         0.238356
Pregnancies                 0.221898
Insulin_Glucose             0.204349
SkinThickness               0.189065
DiabetesPedigreeFunction    0.173844
BloodPressure               0.165723
Insulin                     0.148457
Name: Outcome, dtype: float64
```

```
# Feature engineering
df['Glucose_BMI'] = df['Glucose'] * df['BMI']
df['Age_Glucose'] = df['Age'] * df['Glucose']
df['Insulin_Glucose'] = df['Insulin'] * df['Glucose']

# Remove outliers
z_scores = np.abs(stats.zscore(df))
df_clean = df[(z_scores < 3).all(axis=1)]

# Prepare data
X = df_clean.drop(columns=['Outcome'])
```

```
y = df_clean['Outcome']

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)  # Keep feature names


# Balance classes with SMOTE
smote = SMOTE(random_state=42)
X_balanced, y_balanced = smote.fit_resample(X_scaled_df, y)

# Feature selection with RFE
base_svm = SVC(kernel='linear', random_state=42)
rfe = RFE(estimator=base_svm, n_features_to_select=8)
X_selected = rfe.fit_transform(X_balanced, y_balanced)
selected_features = X_balanced.columns[rfe.support_]

# Split data
X_train, X_test, y_train, y_test = train_test_split(X_selected, y_balanced, test_size=0.2, random_state=42)

# Define models with optimized parameters
svm1 = SVC(kernel='rbf', C=10, gamma=1, probability=True, random_state=42)
svm2 = SVC(kernel='linear', C=50, gamma='scale', probability=True, random_state=42)
svm3 = SVC(kernel='poly', C=10, degree=3, gamma='scale', probability=True, random_state=42)
xgb = XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, random_state=42)


# Create ensemble
ensemble = VotingClassifier(
    estimators=[
        ('rbf_svm', svm1),
        ('linear_svm', svm2),
        ('poly_svm', svm3),
        ('xgboost', xgb)
    ],
    voting='soft',
    weights=[1, 1, 1, 2]  # Give more weight to XGBoost
)

# Train ensemble
ensemble.fit(X_train, y_train)

# Evaluate
y_pred = ensemble.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Ensemble Model Accuracy: {accuracy * 100:.2f}%")
```

```
Ensemble Model Accuracy: 85.19%
```

```
# Feature ranking
feature_ranking = pd.DataFrame({
    'Feature': X.columns,
    'Selected': rfe.support_
})
print("\nSelected Features:")
print(feature_ranking[feature_ranking['Selected']])
```

```
Selected Features:
                      Feature  Selected
0                 Pregnancies      True
1                     Glucose      True
5                         BMI      True
6   DiabetesPedigreeFunction      True
7                         Age      True
8                 Glucose_BMI      True
9                 Age_Glucose      True
10           Insulin_Glucose      True
```

```
# Test case with feature names
test_case = pd.DataFrame(
    [[6, 148, 72, 35, 0, 33.6, 0.627, 50, 148*33.6, 50*148, 8*148]],
    columns=X.columns
)
test_scaled = scaler.transform(test_case)
test_selected = rfe.transform(test_scaled)
prediction = ensemble.predict(test_selected)[0]
print("\nTest Case Prediction:", "Diabetic" if prediction == 1 else "Non-Diabetic")
```

```
Test Case Prediction: Diabetic
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but RFE was fitted wit
  warnings.warn(
```

```
# Test case with feature names
test_case = pd.DataFrame(
```

```
    [[1,85,66,29,0,26.6,0.351,31, 148 33.6, 50 148, 0 148]],
    columns=X.columns
)
test_scaled = scaler.transform(test_case)
test_selected = rfe.transform(test_scaled)
prediction = ensemble.predict(test_selected)[0]
print("\nTest Case Prediction:", "Diabetic" if prediction == 1 else "Non-Diabetic")
```

```
Test Case Prediction: Non-Diabetic
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but RFE was fitted wit
  warnings.warn(
```