```
# Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Load the dataset
df=pd.read_csv('/content/breast-cancer.csv')
```

```
# Display first few rows
df.head()
```

| | id | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothn |
|---|---|---|---|---|---|---|---|
| **0** | 842302 | M | 17.99 | 10.38 | 122.80 | 1001.0 | |
| **1** | 842517 | M | 20.57 | 17.77 | 132.90 | 1326.0 | |
| **2** | 84300903 | M | 19.69 | 21.25 | 130.00 | 1203.0 | |
| **3** | 84348301 | M | 11.42 | 20.38 | 77.58 | 386.1 | |
| **4** | 84358402 | M | 20.29 | 14.34 | 135.10 | 1297.0 | |

5 rows × 32 columns

```
# Display basic information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
```

```
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

1.Load and prepare a dataset for binary classification

```
# Drop the 'id' column
df.drop('id',axis=1,inplace=True)
```

```
# Encode 'diagnosis' as binary
df['diagonsis']=df['diagnosis'].map({'M':1,'B':0})
```

```
# Split data into features and target
X=df.drop(['diagnosis'],axis=1)
y=df['diagonsis']
```

```
# Normalize the features
from sklearn.preprocessing import StandardScaler
```

```
ss=StandardScaler()
```

```
X=ss.fit_transform(X)
```

```
# Train-test split
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test= train_test_split(X,y,test_size=0.2,random_state=42)
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((455, 31), (114, 31), (455,), (114,))
```

2.Train an SVM with linear and RBF kernel.

```
from sklearn.svm import SVC
```

```
# Train SVM with linear kernel
svm_linear = SVC(kernel='linear', random_state=42)
svm_linear.fit(X_train, y_train)
y_pred_linear = svm_linear.predict(X_test)
```

```
# Train SVM with RBF kernel
svm_rbf = SVC(kernel='rbf', random_state=42)
svm_rbf.fit(X_train, y_train)
y_pred_rbf = svm_rbf.predict(X_test)
```

```
# Evaluate both models
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
linear_results = {
    "Accuracy": accuracy_score(y_test, y_pred_linear),
    "Classification Report": classification_report(y_test, y_pred_linear, output_dict=Tru
    "Confusion Matrix": confusion_matrix(y_test, y_pred_linear)
}
```

```
linear_results
```

```
{'Accuracy': 1.0,
 'Classification Report': {'0': {'precision': 1.0,
   'recall': 1.0,
   'f1-score': 1.0,
   'support': 71.0},
  '1': {'precision': 1.0, 'recall': 1.0, 'f1-score': 1.0, 'support': 43.0},
  'accuracy': 1.0,
  'macro avg': {'precision': 1.0,
   'recall': 1.0,
   'f1-score': 1.0,
   'support': 114.0},
  'weighted avg': {'precision': 1.0,
   'recall': 1.0,
   'f1-score': 1.0,
   'support': 114.0}},
 'Confusion Matrix': array([[71,  0],
        [ 0, 43]])}
```

```
rbf_results = {
    "Accuracy": accuracy_score(y_test, y_pred_rbf),
    "Classification Report": classification_report(y_test, y_pred_rbf, output_dict=True),
    "Confusion Matrix": confusion_matrix(y_test, y_pred_rbf)
}
```

```
rbf_results
```

```
{'Accuracy': 1.0,
 'Classification Report': {'0': {'precision': 1.0,
   'recall': 1.0,
   'f1-score': 1.0,
```

```
  'support': 71.0},
 '1': {'precision': 1.0, 'recall': 1.0, 'f1-score': 1.0, 'support': 43.0},
 'accuracy': 1.0,
 'macro avg': {'precision': 1.0,
  'recall': 1.0,
  'f1-score': 1.0,
  'support': 114.0},
 'weighted avg': {'precision': 1.0,
  'recall': 1.0,
  'f1-score': 1.0,
  'support': 114.0}},
'Confusion Matrix': array([[71,  0],
       [ 0, 43]])}
```

3.Visualize decision boundary using 2D data.

```
# Select two features for 2D visualization
feature1 = 'radius_mean'
feature2 = 'texture_mean'
```

```
# Extract corresponding columns and scale them
X_2d = df[[feature1, feature2]]
X_2d_scaled = ss.fit_transform(X_2d)
```

```
# Re-split for 2D visualization
X_train_2d, X_test_2d, y_train_2d, y_test_2d = train_test_split(X_2d_scaled, y, test_size
```

```
# Fit both SVMs on 2D data
svm_linear_2d = SVC(kernel='linear')
svm_linear_2d.fit(X_train_2d, y_train_2d)
```

```
SVC
SVC(kernel='linear')
```

```
svm_rbf_2d = SVC(kernel='rbf')
svm_rbf_2d.fit(X_train_2d, y_train_2d)
```

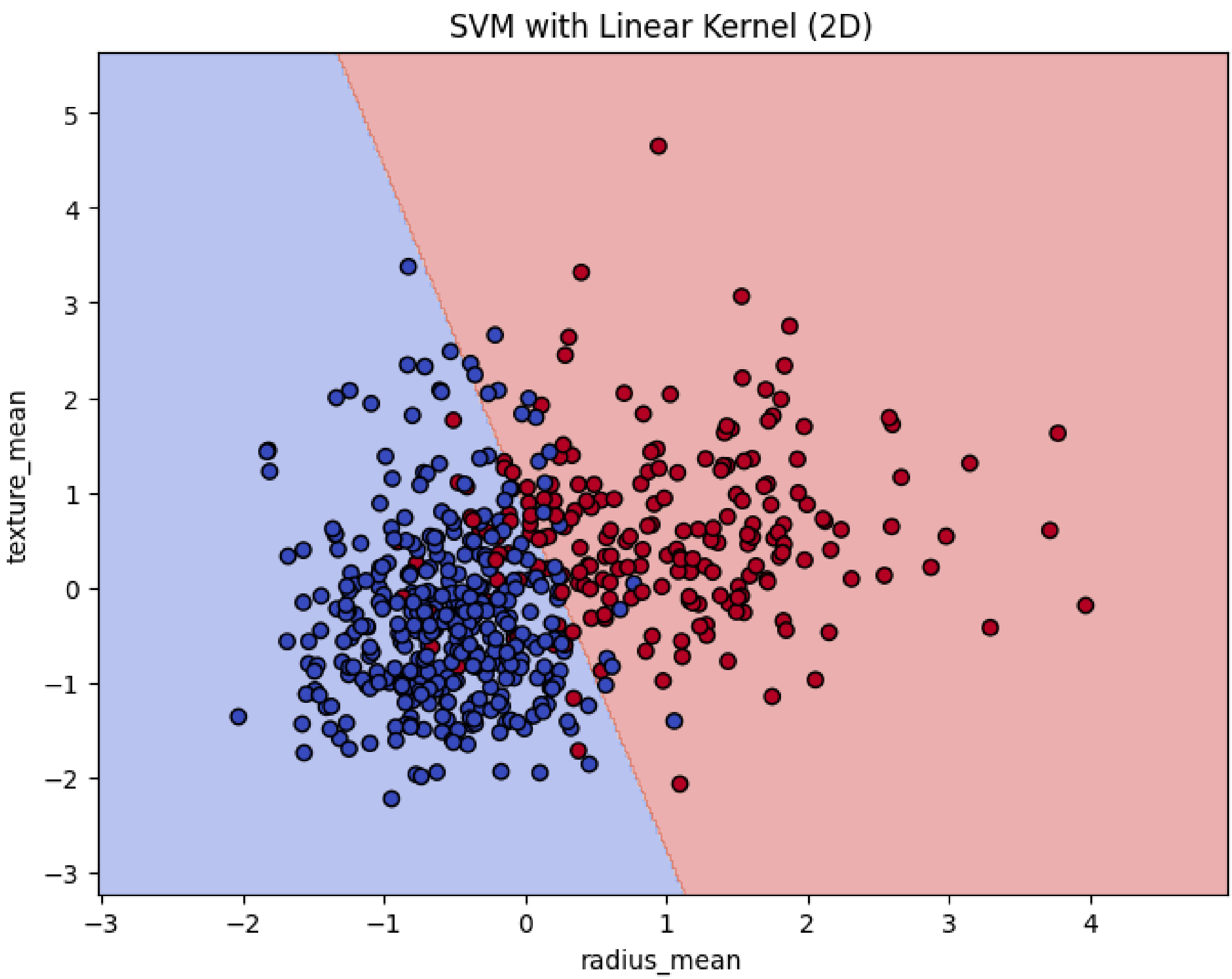
```
SVC
SVC()
```

```
# Function to plot decision boundary
def plot_decision_boundary(clf, X, y, title):
    h = .02  # step size in the mesh
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
```

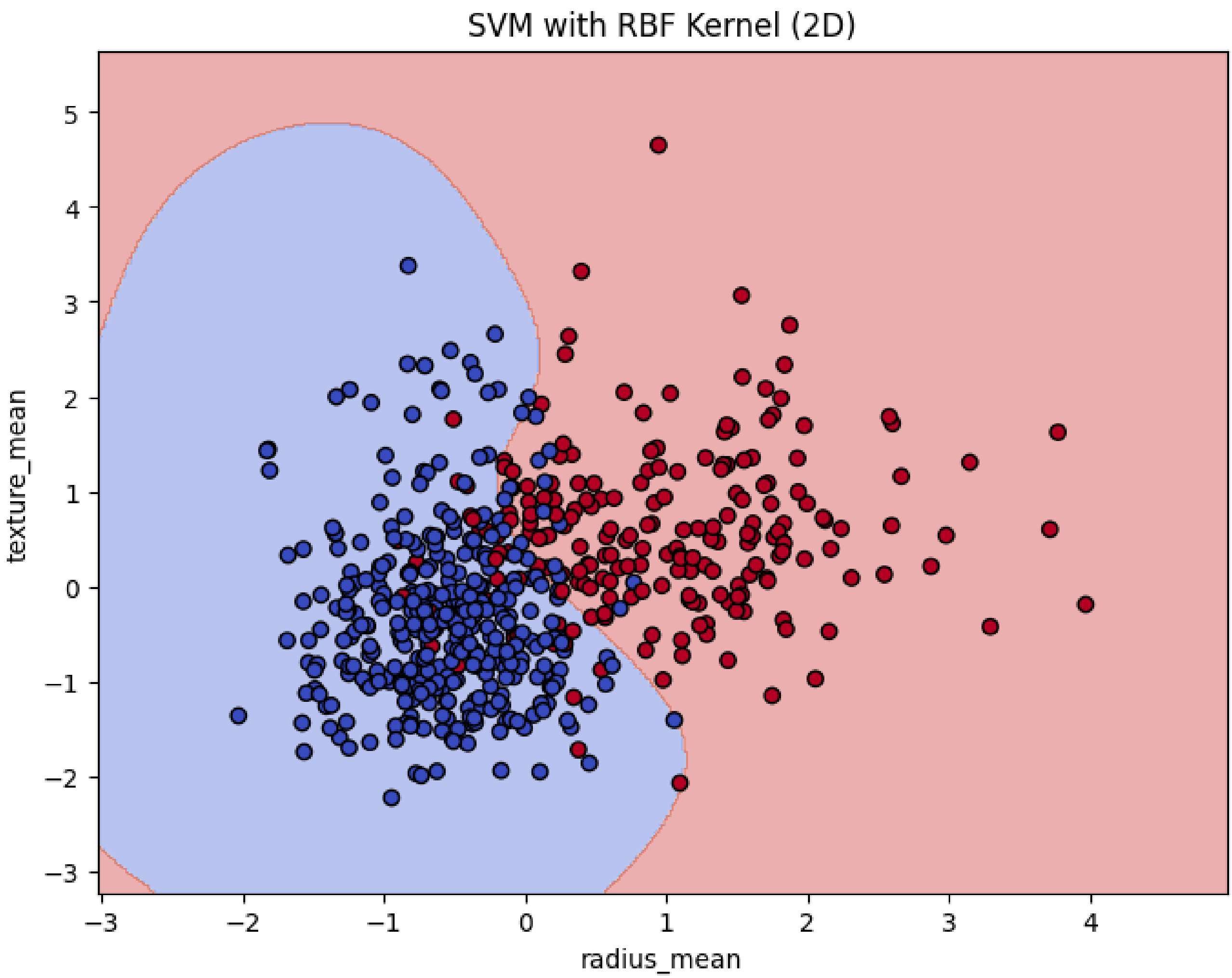
```
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure(figsize=(8, 6))
    plt.contourf(xx, yy, Z, alpha=0.4, cmap=plt.cm.coolwarm)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolors='k')
    plt.xlabel(feature1)
    plt.ylabel(feature2)
    plt.title(title)
    plt.show()
```

```
# Plot decision boundaries
plot_decision_boundary(svm_linear_2d, X_2d_scaled, y, "SVM with Linear Kernel (2D)")
```




```
plot_decision_boundary(svm_rbf_2d, X_2d_scaled, y, "SVM with RBF Kernel (2D)")
```




4.Tune hyperparameters like C and gamma

```
from sklearn.model_selection import GridSearchCV
```

```
# Define parameter grid for SVM with RBF kernel
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [0.01, 0.1, 1, 10],
    'kernel': ['rbf']
}
```

```
# Perform grid search with cross-validation
grid_search = GridSearchCV(SVC(), param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
```

```
GridSearchCV
best_estimator_:
SVC
SVC
```

```
# Print best parameters and accuracy
print("Best Parameters:", grid_search.best_params_)
```

```
Best Parameters: {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
```

```
print("Best CV Accuracy:", grid_search.best_score_)
```

```
Best CV Accuracy: 1.0
```

```
# Test set performance
best_model = grid_search.best_estimator_
test_accuracy = best_model.score(X_test, y_test)
print("Test Accuracy:", test_accuracy)
```

```
Test Accuracy: 1.0
```

## 5.Use cross-validation to evaluate performance

```
from sklearn.model_selection import cross_val_score
```

```
# Linear SVM
svm_linear = SVC(kernel='linear', C=1)
linear_scores = cross_val_score(svm_linear, X, y, cv=5, scoring='accuracy')
```

```
# Print results
print("Linear SVM CV Accuracy: %0.4f ± %0.4f" % (linear_scores.mean(), linear_scores.std(
```

```
Linear SVM CV Accuracy: 1.0000 ± 0.0000
```

```
# RBF SVM
svm_rbf = SVC(kernel='rbf', C=1, gamma='scale')  # 'scale' is default in recent sklearn
rbf_scores = cross_val_score(svm_rbf, X, y, cv=5, scoring='accuracy')
```

```
# Print results
print("RBF SVM CV Accuracy: %0.4f ± %0.4f" % (rbf_scores.mean(), rbf_scores.std()))
```

```
RBF SVM CV Accuracy: 0.9965 ± 0.0070
```

Start coding or generate with AI.