

- مشروع لتعليم الاله على تصنيف تشخيص بمصاب او غير مصاب -

تم إعداده بواسطة : شذا مبارك القحطاني

classification

Importing the libraries

```
1 import numpy as np
2 import pandas as pd
3 from sklearn.model_selection import train_test_split
4 from sklearn.metrics import accuracy_score, recall_score, precision_score, confusion_matrix, classification_report
5 from sklearn.linear_model import LogisticRegression
6 from sklearn.svm import SVC
7 from sklearn.naive_bayes import GaussianNB
8 from sklearn.tree import DecisionTreeClassifier
9 from sklearn.ensemble import RandomForestClassifier
10 from sklearn.neighbors import KNeighborsClassifier
11 from sklearn.model_selection import GridSearchCV
12 from sklearn.decomposition import PCA
13 import matplotlib.pyplot as plt
14 from matplotlib.colors import ListedColormap
15 from matplotlib.colors import ListedColormap
16 import pandas as pd
17 import io
18 import base64
19 import matplotlib.pyplot as plt
20 from matplotlib.font_manager import FontProperties
21 import scipy
22 from sklearn.preprocessing import LabelEncoder
23 from sklearn.preprocessing import StandardScaler
24 from sklearn.preprocessing import OneHotEncoder
25 from matplotlib import gridspec
26 import seaborn as sns
27
```

Import dataset

```
1 df = pd.read_csv('/content/data.csv')
2 df.head()
3
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	cor points_
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.1
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.0
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.1
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.1
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.1

5 rows × 33 columns

```
1 df.info()
2
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 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
32 Unnamed: 32            0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

Exploratory Data Analysis (EDA)

```
1 df['diagnosis'].value_counts()
```

count	
diagnosis	
B	357
M	212

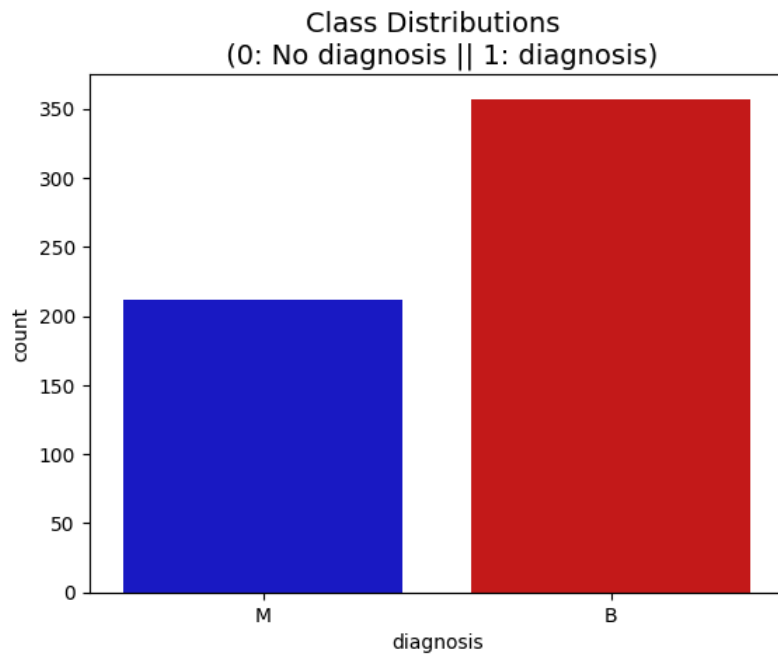
dtype: int64

```
1 colors = ["#0101DF", "#DF0101"]
2 sns.countplot(x='diagnosis', data=df, palette=colors) # تحديد المعامل 'x' بدلاً من 'Class'
3 plt.title('Class Distributions \n (0: No diagnosis || 1: diagnosis)', fontsize=14)
4 plt.show()
5
```

/tmp/ipython-input-3033717015.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend`

```
sns.countplot(x='diagnosis', data=df, palette=colors) # تحديد المعامل 'x' بدلاً من 'Class'
```



```
1 df.isnull().sum()
```

	0
id	0
diagnosis	0
radius_mean	0
texture_mean	0
perimeter_mean	0
area_mean	0
smoothness_mean	0
compactness_mean	0
concavity_mean	0
concave points_mean	0
symmetry_mean	0
fractal_dimension_mean	0
radius_se	0
texture_se	0
perimeter_se	0
area_se	0
smoothness_se	0
compactness_se	0
concavity_se	0
concave points_se	0
symmetry_se	0
fractal_dimension_se	0
radius_worst	0
texture_worst	0
perimeter_worst	0
area_worst	0
smoothness_worst	0
compactness_worst	0
concavity_worst	0
concave points_worst	0
symmetry_worst	0
fractal_dimension_worst	0
Unnamed: 32	569

dtype: int64

```

1 numeric_df = df.select_dtypes(include=['float64', 'int64'])
2 correlation_matrix = numeric_df.corr()
3 print(correlation_matrix)

```

perimeter_worst	0.816322	0.269493
area_worst	0.747419	0.209146
smoothness_worst	0.547691	0.493838
compactness_worst	0.801080	0.614441
concavity_worst	0.855434	0.532520
concave points_worst	1.000000	0.502528
symmetry_worst	0.502528	1.000000
fractal_dimension_worst	0.511114	0.537848
Unnamed: 32	NaN	NaN

	fractal_dimension_worst	Unnamed: 32
id	-0.029866	NaN
radius_mean	0.007066	NaN
texture_mean	0.119205	NaN
perimeter_mean	0.051019	NaN
area_mean	0.003738	NaN
smoothness_mean	0.499316	NaN
compactness_mean	0.687382	NaN
concavity_mean	0.514930	NaN
concave points_mean	0.368661	NaN
symmetry_mean	0.438413	NaN
fractal_dimension_mean	0.767297	NaN
radius_se	0.049559	NaN
texture_se	-0.045655	NaN
perimeter_se	0.085433	NaN
area_se	0.017539	NaN
smoothness_se	0.101480	NaN
compactness_se	0.590973	NaN
concavity_se	0.439329	NaN
concave points_se	0.310655	NaN
symmetry_se	0.078079	NaN
fractal_dimension_se	0.591328	NaN
radius_worst	0.093492	NaN
texture_worst	0.219122	NaN
perimeter_worst	0.138957	NaN
area_worst	0.079647	NaN
smoothness_worst	0.617624	NaN
compactness_worst	0.810455	NaN
concavity_worst	0.686511	NaN
concave points_worst	0.511114	NaN
symmetry_worst	0.537848	NaN
fractal_dimension_worst	1.000000	NaN
Unnamed: 32	NaN	NaN

[32 rows x 32 columns]

```

1 number_columns = df.select_dtypes(include=['int', 'float']).columns
2 print(number_columns)
3 # Columns with categorical data
4 categorical_columns = df.select_dtypes(include=['object']).columns
5 print("Categorical Columns:")
6 print(categorical_columns)
7

```

```

Index(['id', 'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
      'smoothness_mean', 'compactness_mean', 'concavity_mean',
      'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
      'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
      'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
      'fractal_dimension_se', 'radius_worst', 'texture_worst',
      'perimeter_worst', 'area_worst', 'smoothness_worst',
      'compactness_worst', 'concavity_worst', 'concave points_worst',
      'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
Categorical Columns:
Index(['diagnosis'], dtype='object')

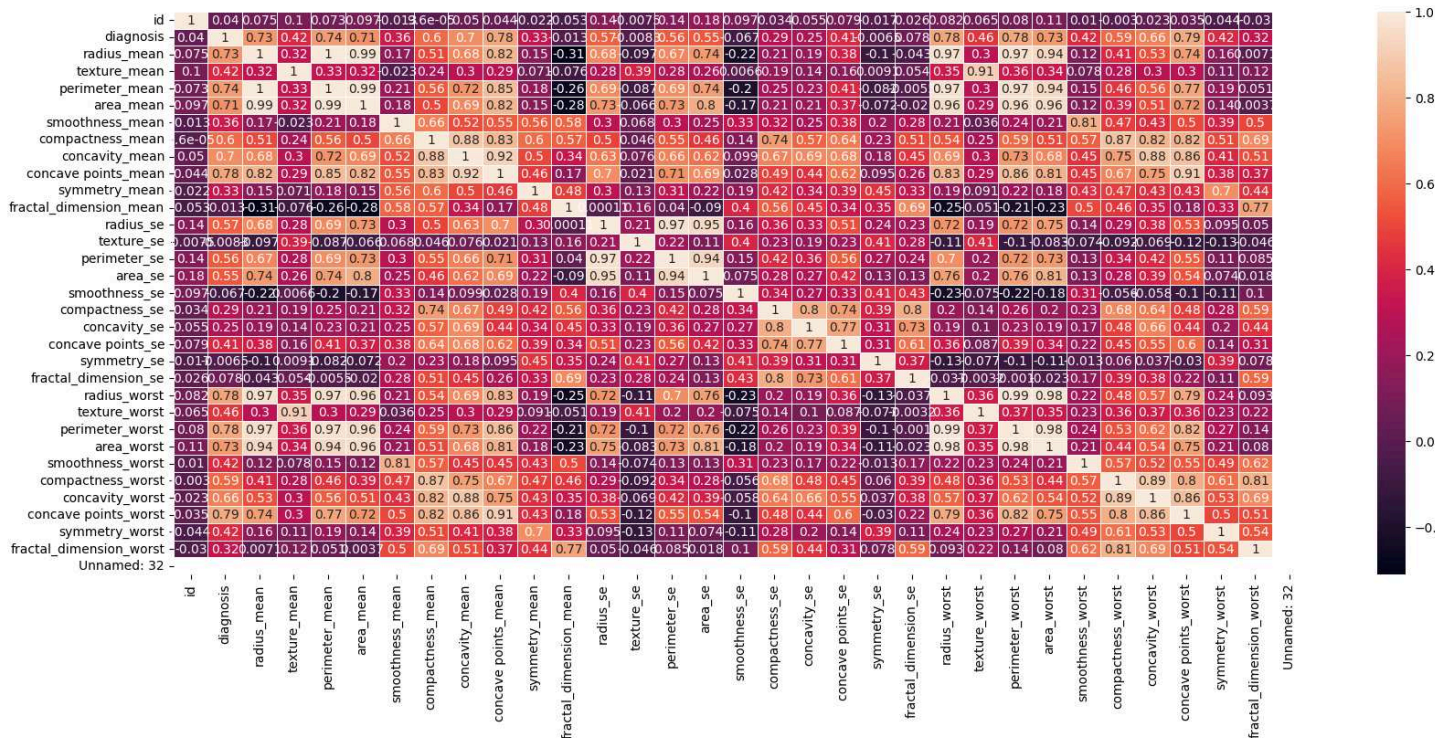
```

```

1
2 df= df
3 le = LabelEncoder()
4 for columns in df.columns:
5     if df[columns].dtype == 'object':
6         df[columns] = le.fit_transform(df[columns])
7 f,ax = plt.subplots(figsize=(20, 8))
8 sns.heatmap(df.corr(), annot=True, linewidths=.5,ax=ax)

```

<Axes: >



```
1 number_columns = df.select_dtypes(include=['int', 'float']).columns
2 num_features = number_columns.copy()
3 num_features
4
```

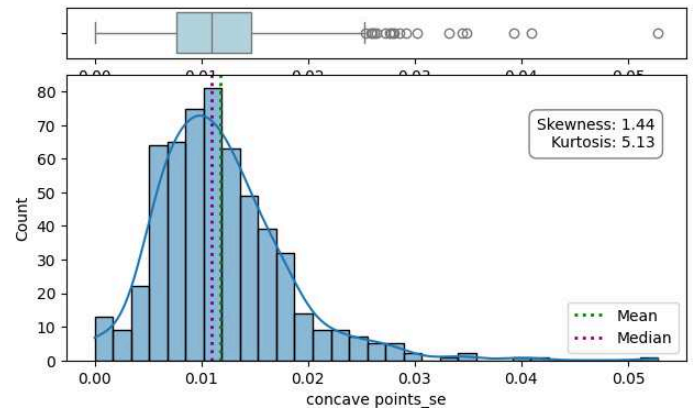
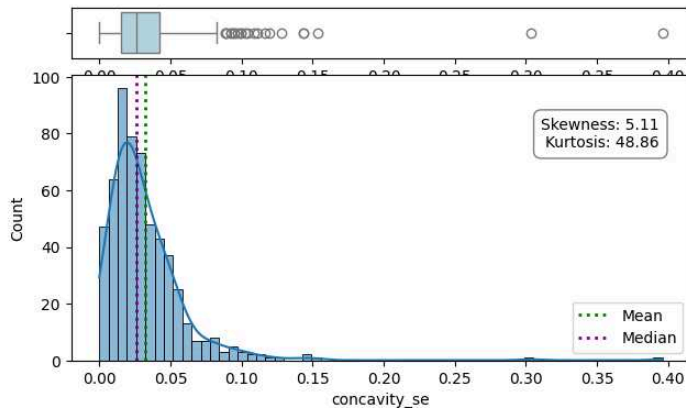
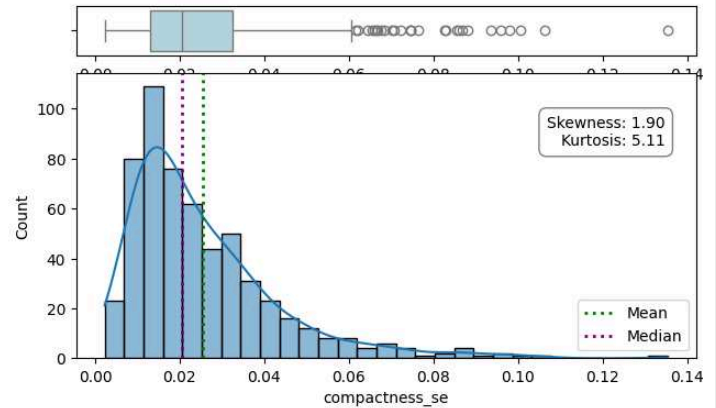
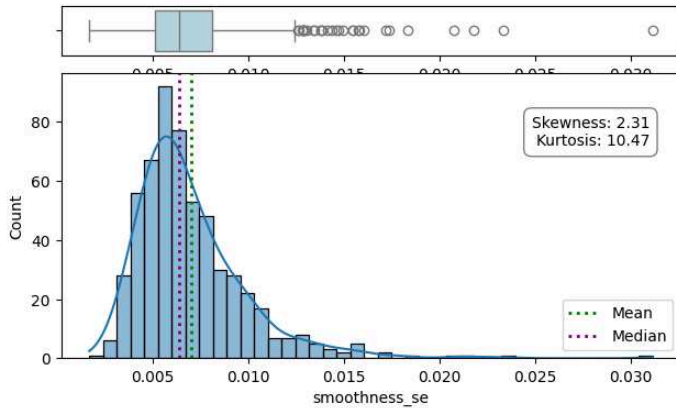
```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

```
1 import matplotlib.pyplot as plt
2 import matplotlib.gridspec as gridspec
3 import seaborn as sns
4 import numpy as np
5
6 selected_features = ['smoothness_se', 'compactness_se', 'concavity_se', 'concave points_se']
7
8 n_cols = 2
9 n_rows = 2
10
11 fig = plt.figure(figsize=(16, 10))
12 outer = gridspec.GridSpec(n_rows, n_cols, wspace=0.2, hspace=0.4)
13
14 for i, col_name in enumerate(selected_features):
15     inner = gridspec.GridSpecFromSubplotSpec(2, 1, subplot_spec=outer[i],
16                                             wspace=0.1, hspace=0.1, height_ratios=(0.15, 0.85))
17
18     ax_box = plt.Subplot(fig, inner[0])
19     sns.boxplot(data=df, x=col_name, color='lightblue', ax=ax_box)
20     ax_box.set_xlabel('')
21     fig.add_subplot(ax_box)
22
23     mean_value = df[col_name].mean()
24     median_value = df[col_name].median()
25     skewness = df[col_name].skew()
26     kurt = df[col_name].kurt()
27
28     ax_hist = plt.Subplot(fig, inner[1])
29     sns.histplot(data=df, x=col_name, kde=True, ax=ax_hist)
```

```

29 sns.histplot(data=df, x=col_name, kde=True, ax=ax_hist)
30 ax_hist.axvline(mean_value, color='green', linestyle='dotted', linewidth=2, label='Mean')
31 ax_hist.axvline(median_value, color='purple', linestyle='dotted', linewidth=2, label='Median')
32 ax_hist.legend(loc='lower right', fontsize=10)
33
34 x_pos = 0.25 if skewness < 0 else 0.95
35
36 ax_hist.text(x_pos, 0.85, f"Skewness: {skewness:.2f}\nKurtosis: {kurt:.2f}",
37             transform=ax_hist.transAxes, verticalalignment='top', horizontalalignment='right',
38             bbox=dict(facecolor='white', edgecolor='gray', boxstyle='round,pad=0.5'),
39             fontsize=10)
40 fig.add_subplot(ax_hist)
41
42 plt.show()

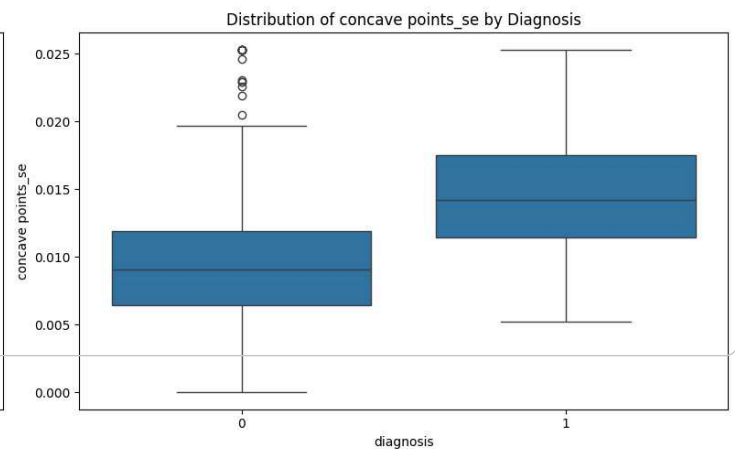
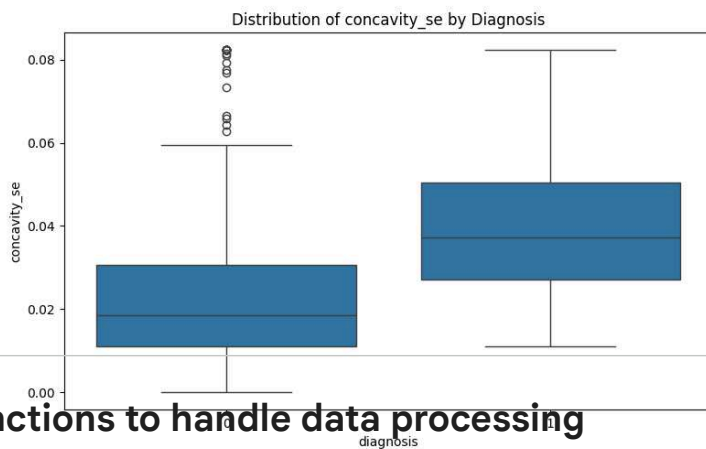
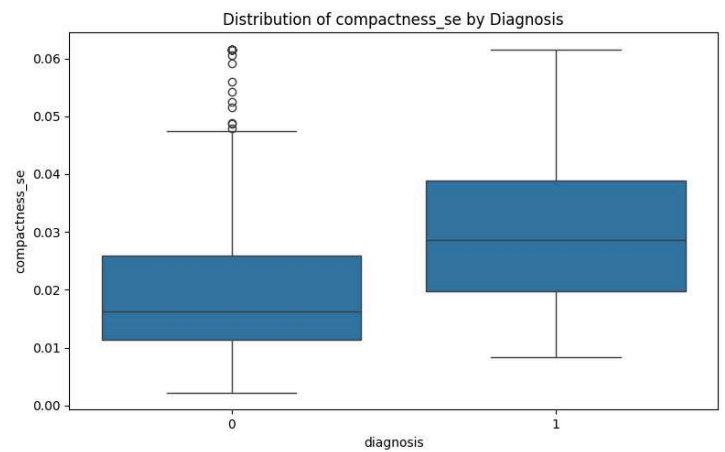
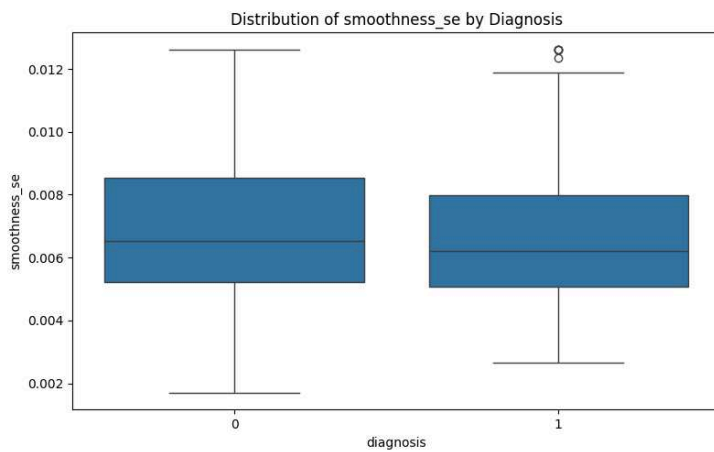
```



```

1 import numpy as np
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4
5 selected_features = ['smoothness_se', 'compactness_se', 'concavity_se', 'concave points_se']
6
7 n_cols = 2
8 n_rows = 2
9
10 fig, ax = plt.subplots(n_rows, n_cols, figsize=(16, 10))
11 ax = ax.flatten()
12
13 for i, feature in enumerate(selected_features):
14     sns.boxplot(data=df, x='diagnosis', y=feature, ax=ax[i])
15     ax[i].set_title(f'Distribution of {feature} by Diagnosis')
16
17 plt.tight_layout()
18 plt.show()

```



Functions to handle data processing

```

1 import numpy as np
2 import pandas as pd
3 from sklearn.impute import KNNImputer
4 from sklearn.preprocessing import LabelEncoder
5 target_column_name = 'diagnosis'
6
7 def drop_highly_correlated_columns(df, target_column_name, threshold=0.9):
8     corr_matrix = df.drop(columns=[target_column_name]).corr().abs()
9     upper_tri = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
10    to_drop = [column for column in upper_tri.columns if any(upper_tri[column] > threshold)]
11    df.drop(columns=to_drop, inplace=True)
12    return df
13
14 def drop_zero_variance_columns(df, target_column_name):
15     df = df.loc[:, df.drop(columns=[target_column_name]).apply(pd.Series.nunique) != 1]
16     return df
17
18
19 def drop_high_percentage_of_missing_values_columns(df, target_column_name, threshold=0.5):
20     missing_percentage = df.drop(columns=[target_column_name]).isna().mean()
21     sparse_columns = missing_percentage[missing_percentage > threshold].index.tolist()
22     df.drop(columns=sparse_columns, inplace=True)
23     return df
24
25 def drop_id_columns(df, target_column_name):
26     id_columns = []
27     for col in df.columns:
28         if col != target_column_name: # Skip the target column
29             unique_values = df[col].nunique()
30             if unique_values == len(df):
31                 id_columns.append(col)
32
33     df.drop(columns=id_columns, inplace=True)
34     return df
35
36 def handle_missing_data(df, target_column_name):
37     df.dropna(subset=[target_column_name], inplace=True)
38
39     for col in df.columns:

```

```

40         if col != target_column_name:
41             if df[col].dtype == 'object':
42                 df[col].fillna(df[col].mode()[0], inplace=True)
43             else:
44                 imputer = KNNImputer(n_neighbors=5)
45                 df[col] = imputer.fit_transform(df[[col]]):[:, 0]
46     return df
47
48 def drop_outliers(df, target_column_name):
49     for col in df.columns:
50         if col != target_column_name: # Skip the target column
51             if pd.api.types.is_numeric_dtype(df[col]) or pd.api.types.is_float_dtype(df[col]):
52                 Q1 = df[col].quantile(0.25)
53                 Q3 = df[col].quantile(0.75)
54                 IQR = Q3 - Q1
55                 whisker_width = 1.5
56                 lower_whisker = Q1 - (whisker_width * IQR)
57                 upper_whisker = Q3 + (whisker_width * IQR)
58                 index = df[col][(df[col] < lower_whisker) | (df[col] > upper_whisker)].index
59                 df.drop(index, inplace=True)
60     return df
61
62 def crappnig_outliers(df, target_column_name):
63     for col in df.columns:
64         if col != target_column_name: # Skip the target column
65             if pd.api.types.is_numeric_dtype(df[col]) or pd.api.types.is_float_dtype(df[col]):
66                 Q1 = df[col].quantile(0.25)
67                 Q3 = df[col].quantile(0.75)
68                 IQR = Q3 - Q1
69                 whisker_width = 1.5
70                 lower_whisker = Q1 - (whisker_width * IQR)
71                 upper_whisker = Q3 + (whisker_width * IQR)
72                 df[col] = np.where(df[col] > upper_whisker, upper_whisker, np.where(df[col] < lower_whisker, lower_whisker, df[col]
73     return df
74
75 def encode_categorical_data(df):
76     categorical_columns = df.select_dtypes(include=['object']).columns
77
78     for col in categorical_columns:
79         unique_values = df[col].nunique()
80         if unique_values > 5:
81             encoded_data = pd.get_dummies(df[col], prefix=col, drop_first=True)
82             df.drop(columns=[col], inplace=True)
83             df = pd.concat([df, encoded_data], axis=1)
84         else:
85             label_encoder = LabelEncoder()
86             df[col] = label_encoder.fit_transform(df[col])
87     return df
88

```

```

1 target_column_name = 'diagnosis'
2 df = encode_categorical_data(df)
3 df = drop_high_percentage_of_missing_values_columns(df, target_column_name)
4 df = drop_id_columns(df, target_column_name)
5 df = handle_missing_data(df, target_column_name)
6 df = crappnig_outliers(df, target_column_name)
7 df = drop_highly_correlated_columns(df, target_column_name)
8
9

```

```

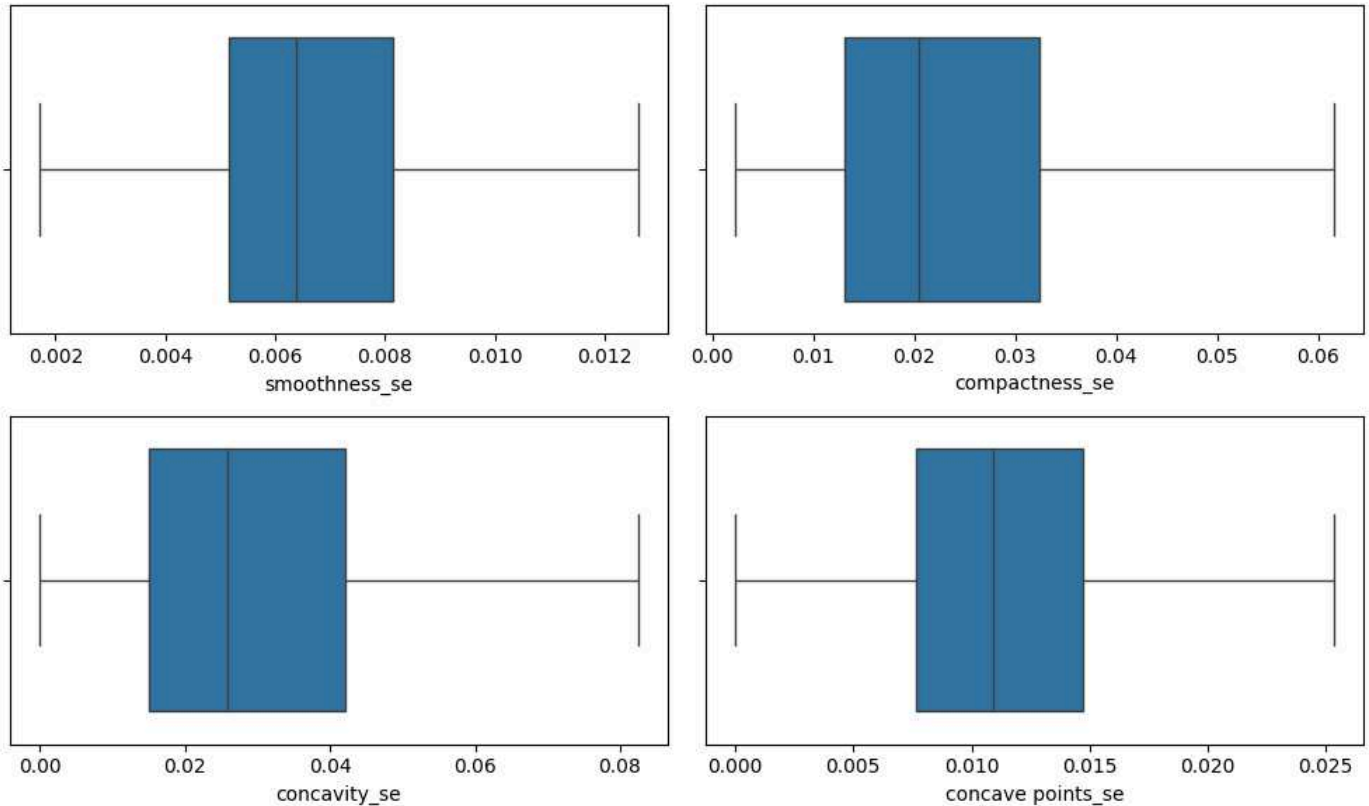
1 import numpy as np
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4
5 selected_features = ['smoothness_se', 'compactness_se', 'concavity_se', 'concave points_se']
6
7 n_cols = 2
8 n_rows = 2
9
10 fig, ax = plt.subplots(n_rows, n_cols, figsize=(10, 6))
11 ax = ax.flatten()
12
13 for i, feature in enumerate(selected_features):
14     sns.boxplot(data=df, x=feature, ax=ax[i])
15

```

```

16 plt.tight_layout()
17 plt.show()

```



_ Train Test Split _ Feature scaling _Build Model

```

1 from sklearn.model_selection import GridSearchCV, train_test_split
2 from sklearn.metrics import accuracy_score, recall_score, precision_score, confusion_matrix, classification_report
3 from sklearn.decomposition import PCA
4 from sklearn.impute import SimpleImputer
5 from sklearn.linear_model import LogisticRegression
6 from sklearn.svm import SVC
7 from sklearn.neighbors import KNeighborsClassifier
8 from sklearn.naive_bayes import GaussianNB
9 from sklearn.tree import DecisionTreeClassifier
10 from sklearn.ensemble import RandomForestClassifier
11 import matplotlib.pyplot as plt
12 from matplotlib.colors import ListedColormap
13 import numpy as np
14
15 colors = ['blue', 'orange']
16 target_column_name = 'diagnosis'
17
18 def splitting_data(df, target_column_name):
19     X = df.drop(columns=[target_column_name])
20     y = df[target_column_name]
21     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
22     return X_train, X_test, y_train, y_test
23
24 def scaling_features(X_train, X_test):
25     imputer = SimpleImputer(strategy='mean')
26     X_train_imputed = imputer.fit_transform(X_train)
27     X_test_imputed = imputer.transform(X_test)
28
29     pca = PCA(n_components=2)
30     X_train_pca = pca.fit_transform(X_train_imputed)
31     X_test_pca = pca.transform(X_test_imputed)
32     return X_train_pca, X_test_pca
33
34 def perform_cv(model, params, X, y):
35     grid = GridSearchCV(model, params, cv=5, scoring='accuracy', n_jobs=-1)
36     grid.fit(X, y)
37     return grid.best_estimator_
38

```

```

39 def evaluate_classification(X_train_pca, X_test_pca, y_train, y_test, algorithm):
40     if algorithm == 'Logistic Regression':
41         classifier = LogisticRegression(max_iter=1000)
42     elif algorithm == 'SVM':
43         classifier = SVC(kernel="rbf", gamma=0.5, C=1.0)
44     elif algorithm == 'K-Nearest Neighbor':
45         classifier = perform_cv(KNeighborsClassifier(), {'n_neighbors': [3, 5, 7, 9, 11]}, X_train_pca, y_train)
46     elif algorithm == 'Naive Bayes':
47         classifier = GaussianNB()
48     elif algorithm == 'Decision Tree':
49         classifier = DecisionTreeClassifier()
50     elif algorithm == 'Random Forest':
51         classifier = perform_cv(RandomForestClassifier(random_state=42), {'n_estimators': [50, 100, 150, 200, 250]}, X_train_pca, y
52
53     classifier.fit(X_train_pca, y_train)
54     y_pred = classifier.predict(X_test_pca)
55
56     print(f"Algorithm: {algorithm}")
57     print("Accuracy:", accuracy_score(y_test, y_pred))
58     print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
59     print("Classification Report:\n", classification_report(y_test, y_pred))
60
61     datasets = [(X_train_pca, y_train, "Train set"), (X_test_pca, y_test, "Test set")]
62
63     for X_set, y_set, title_suffix in datasets:
64         X1, X2 = np.meshgrid(np.arange(X_set[:, 0].min() - 1, X_set[:, 0].max() + 1, 0.05),
65                             np.arange(X_set[:, 1].min() - 1, X_set[:, 1].max() + 1, 0.05))
66         plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
67                     alpha=0.75, cmap=ListedColormap(colors))
68         plt.xlim(X1.min(), X1.max())
69         plt.ylim(X2.min(), X2.max())
70         for i, j in enumerate(np.unique(y_set)):
71             plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
72                         c=ListedColormap(colors)(i), label=j)
73         plt.title(f"{algorithm} Classification ({title_suffix})")
74         plt.legend()
75         plt.show()

```

visualization to train data and test data

```

1 X_train, X_test, y_train, y_test = splitting_data(df, "diagnosis")
2 X_train_pca, X_test_pca = scaling_features(X_train, X_test)
3
4

```

```

1 evaluate_classification(X_train_pca, X_test_pca, y_train, y_test, "SVM")

```

```
Algorithm: SVM
Accuracy: 0.8947368421052632
Confusion Matrix:
[[65  6]
 [ 6 37]]
```

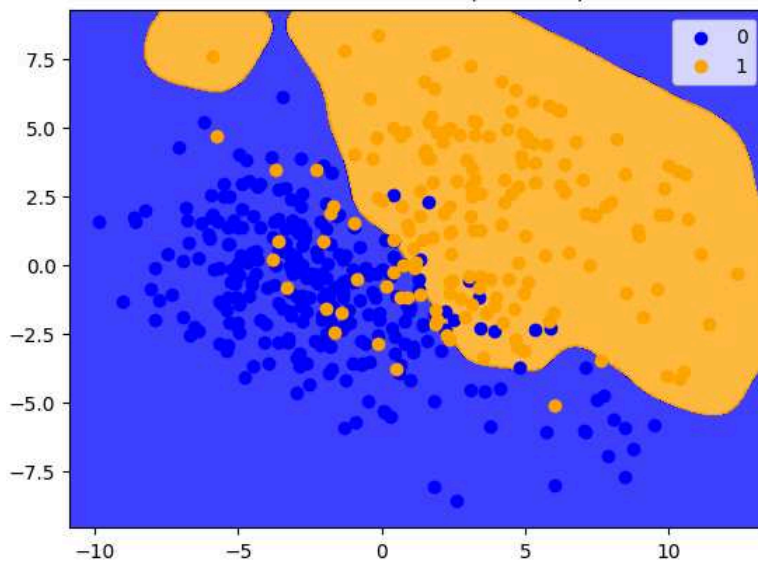
```
Classification Report:
              precision    recall  f1-score   support

     0       0.92       0.92       0.92        71
     1       0.86       0.86       0.86        43

 accuracy          0.89
 macro avg         0.89
 weighted avg      0.89
```

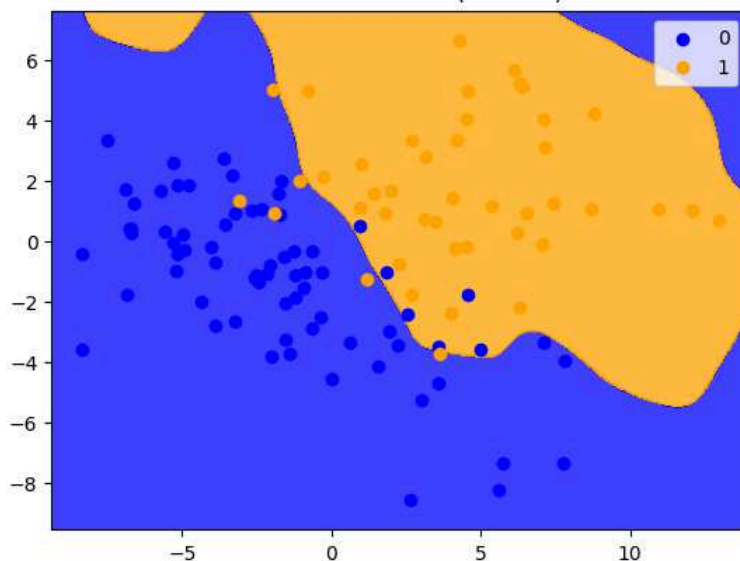
```
/tmp/ipython-input-228660307.py:71: UserWarning: '*' argument looks like a single numeric RGB or RGBA sequence, which should be avoided
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```

SVM Classification (Train set)



```
/tmp/ipython-input-228660307.py:71: UserWarning: '*' argument looks like a single numeric RGB or RGBA sequence, which should be avoided
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```

SVM Classification (Test set)



```
1 confusion = evaluate_classification(X_train_pca, X_test_pca, y_train, y_test, 'K-Nearest Neighbor')
2 print(confusion)
```

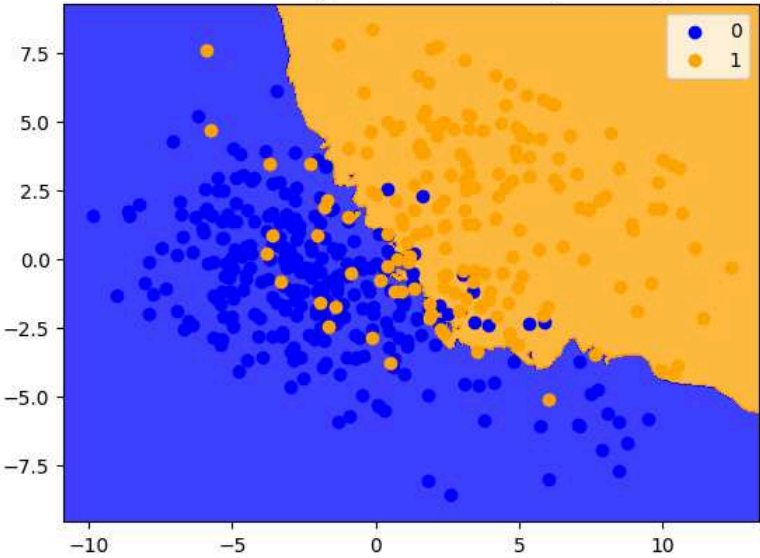
Algorithm: K-Nearest Neighbor
Accuracy: 0.9298245614035088
Confusion Matrix:
[[68 3]
 [5 38]]

Classification Report:

	precision	recall	f1-score	support
0	0.93	0.96	0.94	71
1	0.93	0.88	0.90	43
accuracy			0.93	114
macro avg	0.93	0.92	0.92	114
weighted avg	0.93	0.93	0.93	114

/tmp/ipython-input-228660307.py:71: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],

K-Nearest Neighbor Classification (Train set)



/tmp/ipython-input-228660307.py:71: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],

K-Nearest Neighbor Classification (Test set)

