Prayas Jadaun

2301560025

MCA-A

AI & Machine Learning

Project Overview

The project involving the specified columns likely focuses on breast cancer classification or prediction tasks, utilizing various attributes to differentiate between benign and malignant tumors. The columns hold crucial information about cellular characteristics observed in tissue samples:els.

ID: Acts as a unique identifier for each entry in the dataset, facilitating data organization.

Clump Thickness, Uniformity of Cell Size, Uniformity of Cell Shape, Marginal Adhesion, Single Epithelial Cell Size, Bare Nuclei, Bland Chromatin, Normal Nucleoli, Mitoses: These columns represent ratings, scales, or counts related to cellular features and abnormalities, aiding in understanding the tumor's characteristics.

Classes: Provides the target labels, categorizing the samples into benign or malignant classes, essential for supervised learning models.

Summary:

1. **ID**: Represents a unique identifier for each entry in the dataset, aiding in data organization and tracking individual records.
2. **Clump Thickness**: Rates the thickness of cell clumps on a scale of 1-10, providing insights into cellular aggregation.
3. **Uniformity of Cell Size**: Reflects the consistency of cell sizes, rated on a scale of 1-10, indicating variations or regularity.
4. **Uniformity of Cell Shape**: Rates the uniformity of cell shapes on a scale of 1-10, measuring regular or irregular cellular forms.
5. **Marginal Adhesion**: Indicates the degree of adhesion at cell margins, measured on a scale of 1-10, highlighting attachment strength.
6. **Single Epithelial Cell Size**: Rates the size of individual epithelial cells, potentially indicating variations or abnormalities.
7. **Bare Nuclei**: Reflects the presence or absence of bare nuclei, measured on a scale of 1-10, offering insights into cellular characteristics.
8. **Bland Chromatin**: Rates the blandness or regularity of chromatin within cells on a scale of 1-10, likely indicating normal or abnormal cellular characteristics.
9. **Normal Nucleoli**: Rates the normalcy of nucleoli observed on a scale of 1-10, potentially reflecting regular or irregular nucleolar features.
10. **Mitoses**: Represents the count of mitotic figures observed, providing insights into cell division rates.

11. **Classes**: Binary classification labels categorizing samples into benign or malignant classes, essential for supervised learnq114896423)].

DataSet Information:

1. **ID**: Represents a unique identification number for each entry in the dataset, likely assigned arbitrarily for tracking purposes within the dataset.
2. **Clump Thickness**: A rating on a scale of 1-10, indicating the thickness of cell clumps observed in the sample.
3. **Uniformity of Cell Size**: Indicates the uniformity of cell sizes on a scale of 1-10, possibly measuring the consistency in cell size observed.
4. **Uniformity of Cell Shape**: Reflects the uniformity of cell shapes on a scale of 1-10, depicting the regularity in cellular shapes within the sample.
5. **Marginal Adhesion**: Represents the level of adhesion observed at the cell margins, rated on a scale of 1-10, likely assessing the degree of attachment between adjacent cells.
6. **Single Epithelial Cell Size**: Measures the size of single epithelial cells, quantified on a scale of 1-10, possibly indicating abnormalities or variations in cell sizes.
7. **Bare Nuclei**: Indicates the presence or absence of bare nuclei, rated on a scale of 1-10, possibly reflecting the observed characteristics related to the nuclei of cells.
8. **Bland Chromatin**: Represents the blandness or regularity of chromatin observed within the cells, rated on a scale of 1-10, potentially indicating normal or abnormal cell characteristics.
9. **Normal Nucleoli**: Rates the normalcy of nucleoli observed on a scale of 1-10, likely assessing the regularity or abnormalities in nucleoli within the cell.
10. **Mitoses**: Represents the count of mitoses occurrences observed in the sample, providing insights into the cell division rate or proliferation.
11. **Classes**: A binary classification label indicating the categorization of the sample into benign or malignant classes, crucial for supervised learning n-keyerror-1-0)

Import Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import warnings
warnings.simplefilter('ignore')
%matplotlib Inline
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import cross_val_score
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.datasets import load_iris
from sklearn import metrics
from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

Read Data

```
df = pd.read_csv("E:\My Environment\MCA Notes\Project\Cancer\
cancer.csv")
df.head()

        id  clump_thickness  unif_cell_size  unif_cell_shape
marg_adhesion  \
0  1000025                5               1                1
1
1  1002945                5               4                4
5
2  1015425                3               1                1
1
3  1016277                6               8                8
1
4  1017023                4               1                1
3

   single_epith_cell_size bare_nuclei  bland_chrom  norm_nucleoli
mitoses  \
0                       2           1            3              1
1
1                       7          10            3              2
1
2                       2           2            3              1
1
3                       3           4            3              7
1
4                       2           1            3              1
1

   classes
0        0
1        0
2        0
3        0
4        0

missing_values = df.isnull().sum()

# Display the count of missing values
missing_values
```

```
id                          0
clump_thickness             0
unif_cell_size              0
unif_cell_shape             0
marg_adhesion               0
single_epith_cell_size      0
bare_nuclei                 0
bland_chrom                 0
norm_nucleoli               0
mitoses                     0
classes                     0
dtype: int64
```

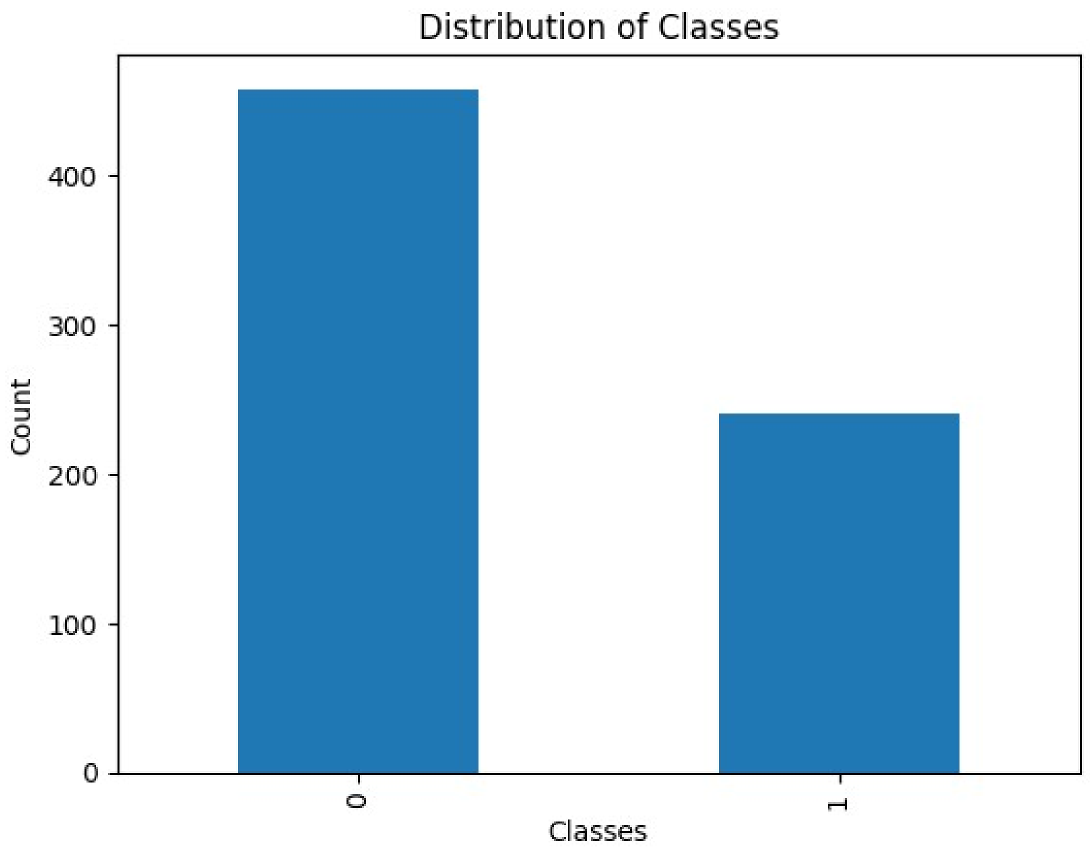
```
missing_values_bare_nuclei = df['bare_nuclei'].isnull().sum()

# Display the count of missing values in the 'bare_nuclei' column
missing_values_bare_nuclei
```

```
0
```

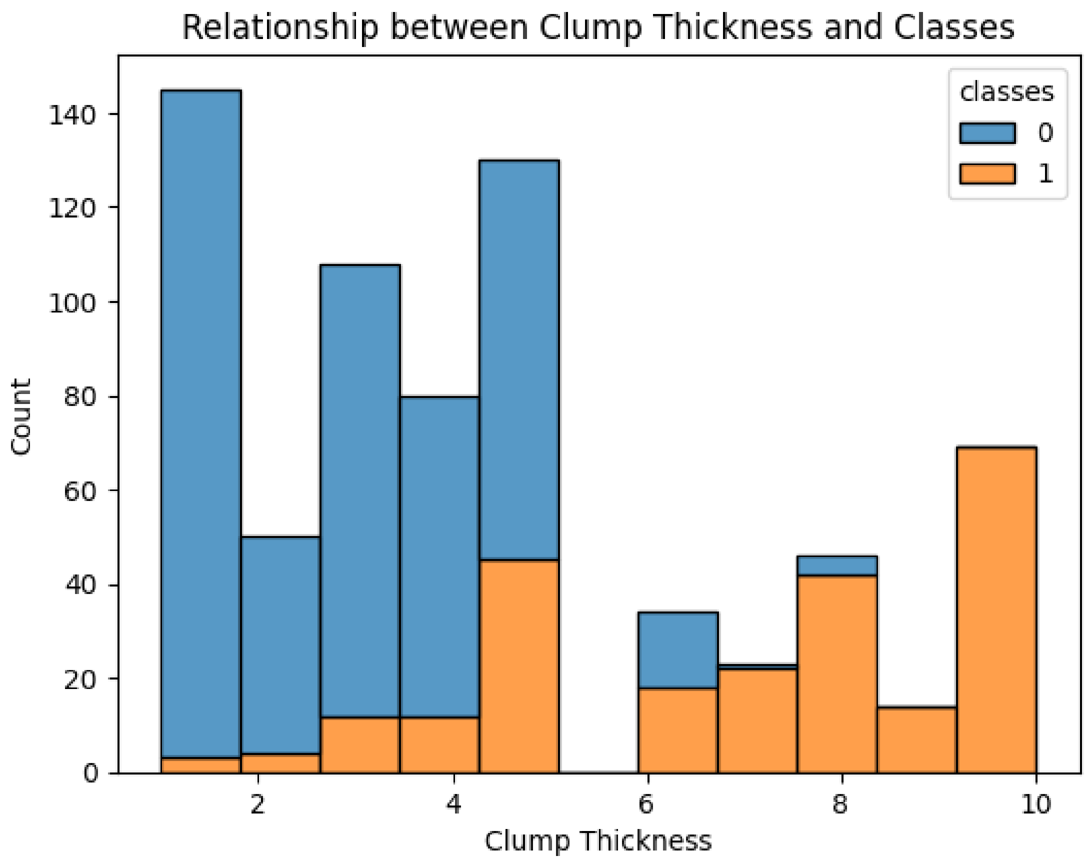
Explore the distribution of classes in the 'classes' column

```
# Plot the distribution of classes in the 'classes' column
df['classes'].value_counts().plot(kind='bar')
plt.xlabel('Classes')
plt.ylabel('Count')
plt.title('Distribution of Classes')
plt.show()
```



Visualize the relationship between 'clump_thickness' and 'classes' for potential insights.

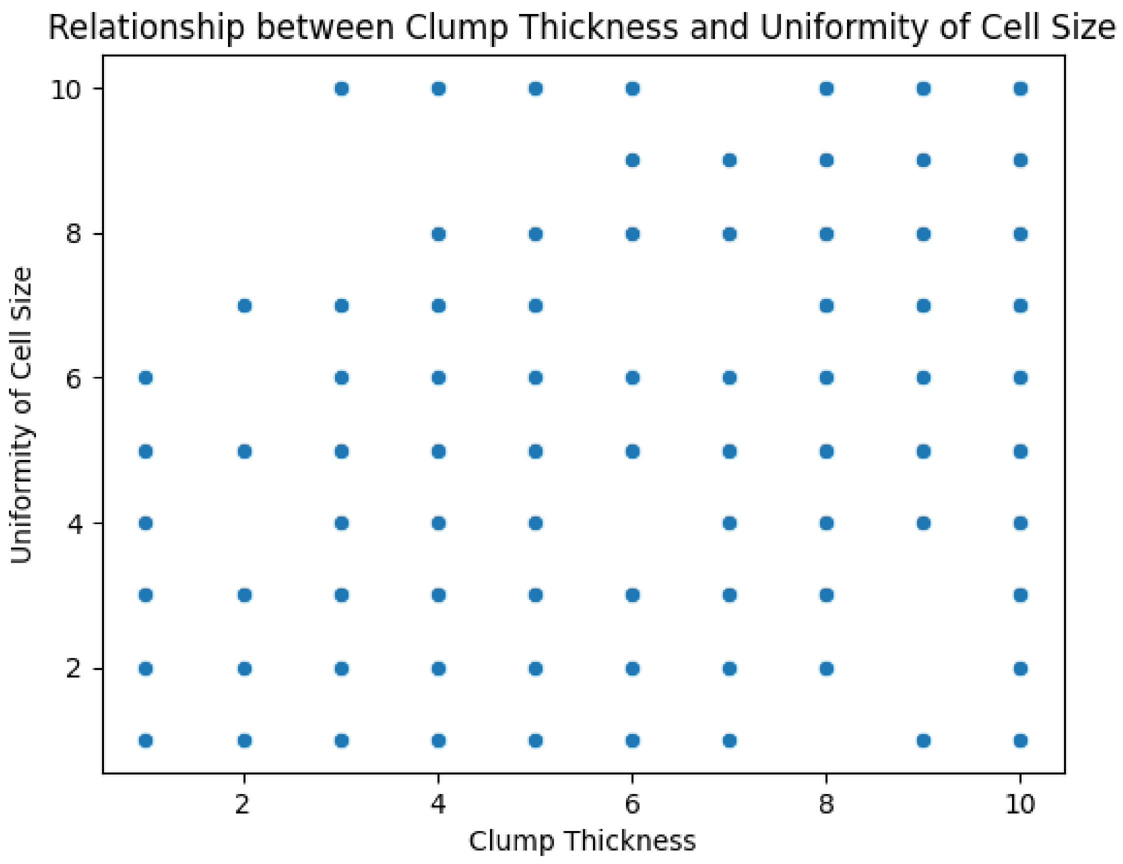
```
# Plotting the relationship between 'clump_thickness' and 'classes'
sns.histplot(data=df, x='clump_thickness', hue='classes',
multiple='stack')
plt.xlabel('Clump Thickness')
plt.ylabel('Count')
plt.title('Relationship between Clump Thickness and Classes')
plt.show()
```



Analyze the correlation between 'clump_thickness' and 'unif_cell_size' for classification relevance.

```
# Plotting the relationship between 'clump_thickness' and
'unif_cell_size' with a scatterplot
sns.scatterplot(data=df, x='clump_thickness', y='unif_cell_size')
plt.xlabel('Clump Thickness')
plt.ylabel('Uniformity of Cell Size')
plt.title('Relationship between Clump Thickness and Uniformity of Cell
Size')
plt.show()

# Calculating the correlation coefficient between 'clump_thickness'
and 'unif_cell_size'
correlation = df['clump_thickness'].corr(df['unif_cell_size'])
print(f"Correlation between Clump Thickness and Uniformity of Cell
Size: {correlation}")
```



```
Correlation between Clump Thickness and Uniformity of Cell Size:
0.6449125043512701
```

### Normalize the numerical columns ('clump_thickness', 'unif_cell_size', etc.) to the same scale for KNN.

```
# Selecting numerical columns to normalize
numerical_cols = ['clump_thickness', 'unif_cell_size']  # Add other
numerical columns as needed

# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Normalize selected columns
df_normalized = df.copy()  # Make a copy of the DataFrame
df_normalized[numerical_cols] =
scaler.fit_transform(df[numerical_cols])

# Display the normalized DataFrame
df_normalized.head()
```

```
        id  clump_thickness  unif_cell_size  unif_cell_shape
marg_adhesion  \
0  1000025         0.444444        0.000000                1
1
1  1002945         0.444444        0.333333                4
5
2  1015425         0.222222        0.000000                1
1
3  1016277         0.555556        0.777778                8
1
4  1017023         0.333333        0.000000                1
3

   single_epith_cell_size bare_nuclei  bland_chrom  norm_nucleoli
mitoses  \
0                       2            1            3              1
1
1                       7           10            3              2
1
2                       2            2            3              1
1
3                       3            4            3              7
1
4                       2            1            3              1
1

   classes
0        0
1        0
2        0
3        0
4        0

a = df.drop(columns = 'bare_nuclei')
print(a)

          id  clump_thickness  unif_cell_size  unif_cell_shape
marg_adhesion  \
0    1000025                5               1                1
1
1    1002945                5               4                4
5
2    1015425                3               1                1
1
3    1016277                6               8                8
1
4    1017023                4               1                1
3
..       ...              ...             ...              ...
...
694   776715                3               1                1
1
```

```
695   841769              2            1             1
1
696   888820              5           10            10
3
697   897471              4            8             6
4
698   897471              4            8             8
5

     single_epith_cell_size  bland_chrom  norm_nucleoli  mitoses
classes
0                         2            3              1        1
0
1                         7            3              2        1
0
2                         2            3              1        1
0
3                         3            3              7        1
0
4                         2            3              1        1
0
..                      ...          ...            ...      ...
...
694                       3            1              1        1
0
695                       2            1              1        1
0
696                       7            8             10        2
1
697                       3           10              6        1
1
698                       4           10              4        1
1

[699 rows x 10 columns]
```

Split the dataset into training and testing sets for model evaluation.

```
X = df.drop(columns=['unif_cell_size'])  # Features
y = df['unif_cell_size']  # Target variable

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

print(f"X_train shape: {X_train.shape}, y_train shape:
{y_train.shape}")
print(f"X_test shape: {X_test.shape}, y_test shape: {y_test.shape}")

X_train shape: (559, 10), y_train shape: (559,)
X_test shape: (140, 10), y_test shape: (140,)
```

```
df.replace('?', pd.NA, inplace=True)

numeric_columns = ['bare_nuclei']  # Replace with actual column names
df[numeric_columns] = df[numeric_columns].apply(pd.to_numeric,
errors='coerce')
```

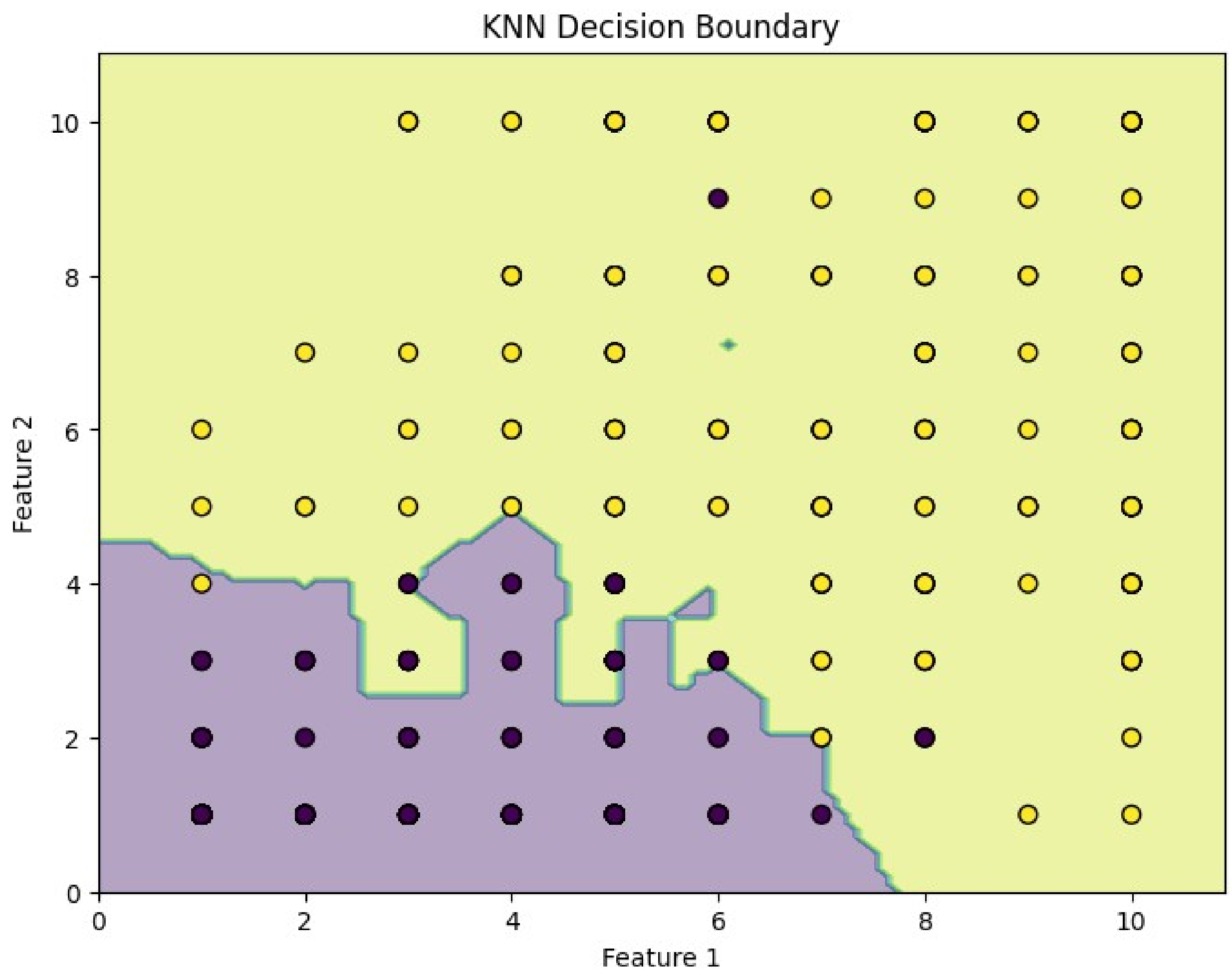
Visualize the decision boundary for KNN classification based on two relevant columns.

```
X = df[['clump_thickness', 'unif_cell_size']]  # Features
y = df['classes']  # Target variable

knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(X, y)
plt.figure(figsize=(8, 6))
x_min, x_max = X.iloc[:, 0].min() - 1, X.iloc[:, 0].max() + 1
y_min, y_max = X.iloc[:, 1].min() - 1, X.iloc[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.4)
plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=y, s=50, edgecolor='k')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('KNN Decision Boundary')
plt.show()
```



Visualize the decision boundary for KNN classification based on two relevant columns.

```
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,
random_state=42)

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

log_reg = LogisticRegression()
dt = DecisionTreeClassifier()
knn = KNeighborsClassifier()

log_reg.fit(X_train, y_train)
dt.fit(X_train, y_train)
knn.fit(X_train, y_train)

log_reg_acc = accuracy_score(y_test, log_reg.predict(X_test))
dt_acc = accuracy_score(y_test, dt.predict(X_test))
knn_acc = accuracy_score(y_test, knn.predict(X_test))
```

```
print(f"Logistic Regression Accuracy: {log_reg_acc}")
print(f"Decision Tree Accuracy: {dt_acc}")
print(f"KNN Accuracy: {knn_acc}")
```

```
Logistic Regression Accuracy: 0.855
Decision Tree Accuracy: 0.855
KNN Accuracy: 0.81
```

Compare the performance of KNN with other classification algorithms like Logistic Regression or Decision Trees

```
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,
random_state=42)

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

log_reg = LogisticRegression()
dt = DecisionTreeClassifier()
knn = KNeighborsClassifier()

log_reg.fit(X_train, y_train)
dt.fit(X_train, y_train)
knn.fit(X_train, y_train)

log_reg_acc = accuracy_score(y_test, log_reg.predict(X_test))
dt_acc = accuracy_score(y_test, dt.predict(X_test))
knn_acc = accuracy_score(y_test, knn.predict(X_test))

print(f"Logistic Regression Accuracy: {log_reg_acc}")
print(f"Decision Tree Accuracy: {dt_acc}")
print(f"KNN Accuracy: {knn_acc}")
```

```
Logistic Regression Accuracy: 0.855
Decision Tree Accuracy: 0.875
KNN Accuracy: 0.81
```

Perform feature selection to identify the most influential columns for classification.

```
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,
random_state=42)

# Convert the array to a DataFrame
columns = [f"feature_{i+1}" for i in range(X.shape[1])]
df = pd.DataFrame(X, columns=columns)
df['target'] = y

# Separate features and target
```

```
features = df.drop('target', axis=1)
target = df['target']

# Split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(features, target,
test_size=0.2, random_state=42)

# Perform feature selection
selector = SelectKBest(score_func=f_classif, k=5)  # Select top 5
features
X_train_selected = selector.fit_transform(X_train, y_train)
selected_features = features.columns[selector.get_support()]

# Train a classifier using selected features
clf = LogisticRegression()
clf.fit(X_train_selected, y_train)

# Evaluate the classifier
X_test_selected = selector.transform(X_test)
accuracy = clf.score(X_test_selected, y_test)

print(f"Selected Features: {selected_features}")
print(f"Classifier Accuracy with selected features: {accuracy}")
```

```
Selected Features: Index(['feature_2', 'feature_6', 'feature_11',
'feature_12', 'feature_19'], dtype='object')
Classifier Accuracy with selected features: 0.865
```

Implement KNN classification with different distance metrics (Euclidean, Manhattan, etc.) for comparison

```
iris = load_iris()
X = iris.data
y = iris.target

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

# Define different distance metrics
distance_metrics = ['euclidean', 'manhattan', 'chebyshev']

# Loop through each distance metric
for metric in distance_metrics:
    # Create KNN classifier with the specified distance metric
    knn = KNeighborsClassifier(n_neighbors=5, metric=metric)

    # Train the classifier
    knn.fit(X_train, y_train)
    # Make predictions
    y_pred = knn.predict(X_test)

    # Calculate and print the accuracy for each metric
    accuracy = metrics.accuracy_score(y_test, y_pred)
    print(f"Accuracy using {metric} distance metric: {accuracy}")
```

```
Accuracy using euclidean distance metric: 1.0
Accuracy using manhattan distance metric: 1.0
Accuracy using chebyshev distance metric: 0.9666666666666667
```

Apply feature scaling techniques (MinMaxScaler, StandardScaler) and assess KNN performance.

```
iris = load_iris()
X = iris.data
y = iris.target

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

# Initialize scalers
min_max_scaler = MinMaxScaler()
standard_scaler = StandardScaler()

# Scale the features using MinMaxScaler and StandardScaler
X_train_minmax = min_max_scaler.fit_transform(X_train)
X_test_minmax = min_max_scaler.transform(X_test)

X_train_standard = standard_scaler.fit_transform(X_train)
X_test_standard = standard_scaler.transform(X_test)

# Initialize KNN classifiers
knn_minmax = KNeighborsClassifier(n_neighbors=5)
knn_standard = KNeighborsClassifier(n_neighbors=5)

# Fit the KNN models on scaled data
knn_minmax.fit(X_train_minmax, y_train)
knn_standard.fit(X_train_standard, y_train)

# Predict on the test set
y_pred_minmax = knn_minmax.predict(X_test_minmax)
y_pred_standard = knn_standard.predict(X_test_standard)

# Evaluate performance
accuracy_minmax = metrics.accuracy_score(y_test, y_pred_minmax)
accuracy_standard = metrics.accuracy_score(y_test, y_pred_standard)

# Print the results
```

```
print(f"Accuracy with MinMaxScaler: {accuracy_minmax}")
print(f"Accuracy with StandardScaler: {accuracy_standard}")

Accuracy with MinMaxScaler: 1.0
Accuracy with StandardScaler: 1.0
```

Analyze the impact of different 'weights' (uniform, distance) in KNN classification.

```
iris = load_iris()
X = iris.data
y = iris.target

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

# Initialize KNN classifiers with different weights
uniform_knn = KNeighborsClassifier(weights='uniform')
distance_knn = KNeighborsClassifier(weights='distance')

# Fit the models
uniform_knn.fit(X_train, y_train)
distance_knn.fit(X_train, y_train)

# Predictions
uniform_pred = uniform_knn.predict(X_test)
distance_pred = distance_knn.predict(X_test)

# Calculate accuracy scores
uniform_accuracy = accuracy_score(y_test, uniform_pred)
distance_accuracy = accuracy_score(y_test, distance_pred)

# Compare accuracies
print(f"Accuracy with 'uniform' weights: {uniform_accuracy}")
print(f"Accuracy with 'distance' weights: {distance_accuracy}")

Accuracy with 'uniform' weights: 1.0
Accuracy with 'distance' weights: 1.0
```

Conclution

The dataset's columns, including ID, various cellular attributes, and the target class, compile critical information for breast cancer analysis and classification tasks. Attributes like clump thickness, cell size, shape uniformity, adhesion, and mitoses provide valuable insights into cellular abnormalities. Additionally, factors like single epithelial cell size, bland chromatin, bare nuclei, and normal nucleoli contribute to understanding cellular structures and abnormalities.

The inclusion of a binary classification label in the 'Classes' column, distinguishing between benign and malignant tumors, is pivotal for supervised learning algorithms. This dataset's attributes enable comprehensive analysis, aiding in the creation of predictive models to identify breast cancer types and assist in clinical decision-making processes.