Course: CSCE 5215 Machine Learning

Professor: Zeenat Tariq

Assignment: 1

Instructions:

1. Wherever you are asked to code, insert a text block below your code block and explain what you have coded as per your own understanding.
2. If the code is provided by us, execute it, and add below a text block and provide your explanation in your own words.
3. Submit both the .ipynb and pdf files to canvas.
4. **The similarity score should be less than 15%.**

Task 1 (15 points)

What is the significance of data visualization in machine learning?

Ans:

- Data visualization is a method of displaying data in a way that makes its relationships and insights more understandable.
- Histogram, Bar lot, Line graphs are some of the examples of Data Visualization.
- Data visualization simplifies the process of understanding the insights of the relationship among variables because data is presented as graphs, scatter plots, pie charts, histograms, and other visual representations. It is an important tool for Exploratory Data Analysis(EDA). *It can help in analyzing and interpreting large and complex amounts of data in the form of visual representations which help save time.*It also helps in grouping attributes together. *It helps in optimizing computational resources.*Data Visualization gives the ability to find the outliers, and expectations in data.

What is the significance of preprocessing the data? How can we represent the data for the Machine Learning Algorithm?

Ans:

- It drops or manipulates data before using it to give better results.
- Accuracy of the machine learning algorithm increases by using data preprocessing because the data after preprocessing is cleaned.
- Data in the ML Algorithm is used in the form of two-dimensional table i.e., rows (represent observations) and columns(represent features).

Task 2: KNN (25 points)

What is the significance of the parameter K in KNN?

How can we find a suitable value for k?

Ans:

- Choosing K is important to improve the accuracy and efficiency of an algorithm.
- If we choose the K value is really high then there will be a case of underfitting.
- If we choose the K value to be really low then there will be a case of overfitting.
- K value can be found using experience or based on trial and error method.

Implement the code below and provide an explanation for each section of the code based on your understanding.

General Imports

```
In [1]: ### use this cell for general imports
        # import panda
        import pandas as pd
        # import seaborn
        import seaborn as sns
        # import numpy
        import numpy as np
        # import matplotlib
        import matplotlib.pyplot as plt
```

- imported the required libraries
- Numpy for mathematical operations
- matplotlib for plotting graphs
- seaborn for visualization

Load the Dataset.

Use any dataset of your choice.

Split the dataset into X and y.

```
In [2]: from sklearn.datasets import load_breast_cancer
        breast_cancer = load_breast_cancer()
        X = breast_cancer.data
        y = breast_cancer.target
```

- Imported the breast cancer dataset and assigned feature to X and targets to y.

Create a dataframe(df) from X and y. Display the tail of df.

```
In [3]: breast_cancer.feature_names
```

```
Out[3]: array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
               'mean smoothness', 'mean compactness', 'mean concavity',
               'mean concave points', 'mean symmetry', 'mean fractal dimension',
               'radius error', 'texture error', 'perimeter error', 'area error',
               'smoothness error', 'compactness error', 'concavity error',
               'concave points error', 'symmetry error',
               'fractal dimension error', 'worst radius', 'worst texture',
               'worst perimeter', 'worst area', 'worst smoothness',
               'worst compactness', 'worst concavity', 'worst concave points',
               'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

```
In [4]: data = pd.DataFrame(data=X, columns=breast_cancer.feature_names)
        data['target'] = y

        df_tail = data.tail()
        print(df_tail)
```

```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
564        21.56         22.39          142.00     1479.0          0.11100   
565        20.13         28.25          131.20     1261.0          0.09780   
566        16.60         28.08          108.30      858.1          0.08455   
567        20.60         29.33          140.10     1265.0          0.11780   
568         7.76         24.54           47.92      181.0          0.05263   

     mean compactness  mean concavity  mean concave points  mean symmetry  \
564           0.11590         0.24390              0.13890         0.1726   
565           0.10340         0.14400              0.09791         0.1752   
566           0.10230         0.09251              0.05302         0.1590   
567           0.27700         0.35140              0.15200         0.2397   
568           0.04362         0.00000              0.00000         0.1587   

     mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
564                 0.05623  ...          26.40           166.10      2027.0   
565                 0.05533  ...          38.25           155.00      1731.0   
566                 0.05648  ...          34.12           126.70      1124.0   
567                 0.07016  ...          39.42           184.60      1821.0   
568                 0.05884  ...          30.37            59.16       268.6   

     worst smoothness  worst compactness  worst concavity  \
564           0.14100            0.21130           0.4107   
565           0.11660            0.19220           0.3215   
566           0.11390            0.30940           0.3403   
567           0.16500            0.86810           0.9387   
568           0.08996            0.06444           0.0000   

     worst concave points  worst symmetry  worst fractal dimension  target  
564                0.2216          0.2060                  0.07115       0  
565                0.1628          0.2572                  0.06637       0  
566                0.1418          0.2218                  0.07820       0  
567                0.2650          0.4087                  0.12400       0  
568                0.0000          0.2871                  0.07039       1  

[5 rows x 31 columns]
```

- Creating a data frame for the breast cancer for complete dataset and printing the last 5 rows for the features.

Change a column name of your choice. For example, change the target or label column to true_value

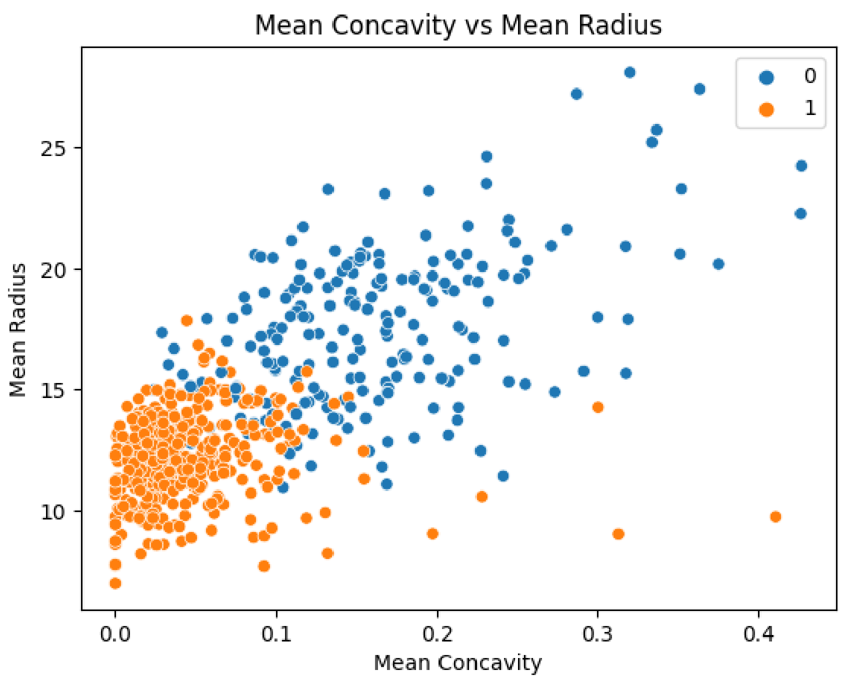
In [5]:
```
data = data.rename(columns={'target': 'target_value'})

print(data.tail())
```

```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
564        21.56         22.39          142.00     1479.0          0.11100
565        20.13         28.25          131.20     1261.0          0.09780
566        16.60         28.08          108.30      858.1          0.08455
567        20.60         29.33          140.10     1265.0          0.11780
568         7.76         24.54           47.92      181.0          0.05263

     mean compactness  mean concavity  mean concave points  mean symmetry  \
564           0.11590         0.24390              0.13890         0.1726
565           0.10340         0.14400              0.09791         0.1752
566           0.10230         0.09251              0.05302         0.1590
567           0.27700         0.35140              0.15200         0.2397
568           0.04362         0.00000              0.00000         0.1587

     mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
564                 0.05623  ...          26.40           166.10      2027.0
565                 0.05533  ...          38.25           155.00      1731.0
566                 0.05648  ...          34.12           126.70      1124.0
567                 0.07016  ...          39.42           184.60      1821.0
568                 0.05884  ...          30.37            59.16       268.6

     worst smoothness  worst compactness  worst concavity  \
564           0.14100            0.21130           0.4107
565           0.11660            0.19220           0.3215
566           0.11390            0.30940           0.3403
567           0.16500            0.86810           0.9387
568           0.08996            0.06444           0.0000

     worst concave points  worst symmetry  worst fractal dimension  \
564                0.2216          0.2060                  0.07115
565                0.1628          0.2572                  0.06637
566                0.1418          0.2218                  0.07820
567                0.2650          0.4087                  0.12400
568                0.0000          0.2871                  0.07039

     target_value
564             0
565             0
566             0
567             0
568             1

[5 rows x 31 columns]
```

- Renamed target as target_value.

Using Matplotlib make 2 visualizations of your choice and describe your understanding of each plot

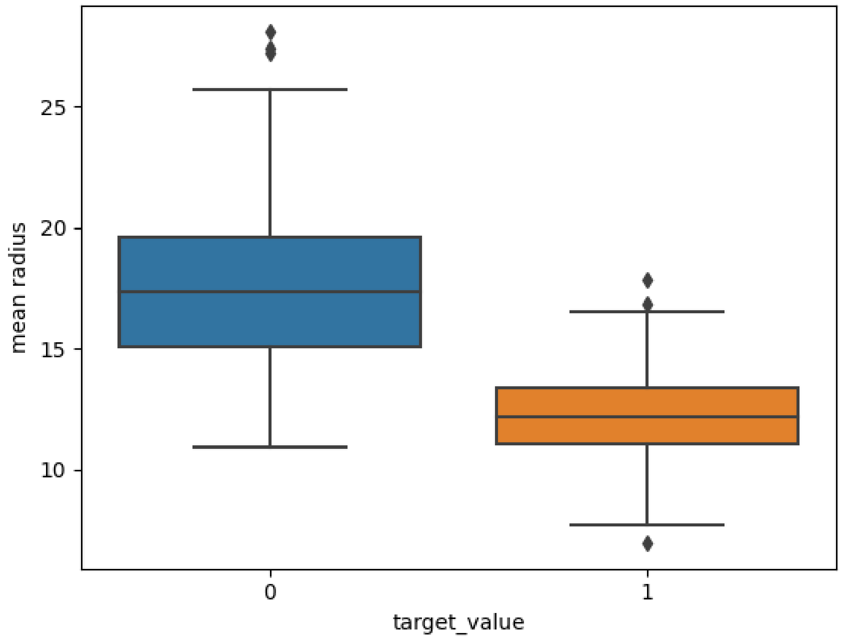
In [6]:
```python
#Visualization 1: Type your code here
# scatter plot
sns.scatterplot(x=X[:, 6],y=X[:, 0],data=data, hue=y) # "hue" = label name

plt.xlabel("Mean Concavity")
plt.ylabel("Mean Radius")
plt.title("Mean Concavity vs Mean Radius")

plt.show()
```



- Scatter plot for x-axis as 'Mean Concavity' and y-axis as 'Mean Radius'.
- The orange dots are target with value 1, and blue dots are target value 0.
- From the figure we can see that they ar bit hard to separate because in the between the values are getting mixed.

In [7]:
```
#Visualization 2: Type your code here
#box plot
sns.boxplot(x='target_value',y='mean radius',data = data)
```

Out[7]:
```
<Axes: xlabel='target_value', ylabel='mean radius'>
```



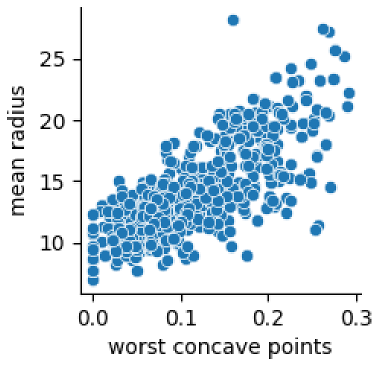
- Box plot for x-axis as 'target_value' and y-axis as 'mean radius'. ### For target_value=0 [Blue Box]
- Min: 11
- Max: 26
- Median: 17
- Q1: 15
- Q3: 20 ### For target_value=1 [Orange Box]
- Min: 8

- Max: 16
- Median: 12
- Q1: 11
- Q3: 14

- Rhombus dots above max value are the outliers.

In [8]:
```
#Visualization-3
#pairplot
sns.pairplot(data, x_vars=['worst concave points'], y_vars=['mean radius'], markers='o')
```

Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x7ea28aa3b9d0>
```



- Pair plot between age vs bp.

Import the necessary library and create two KNN models (a classifier and a regressor) using Hyperparameters tuning (GridSearchCV).

Make sure you have correct input (For example: same class distribution)

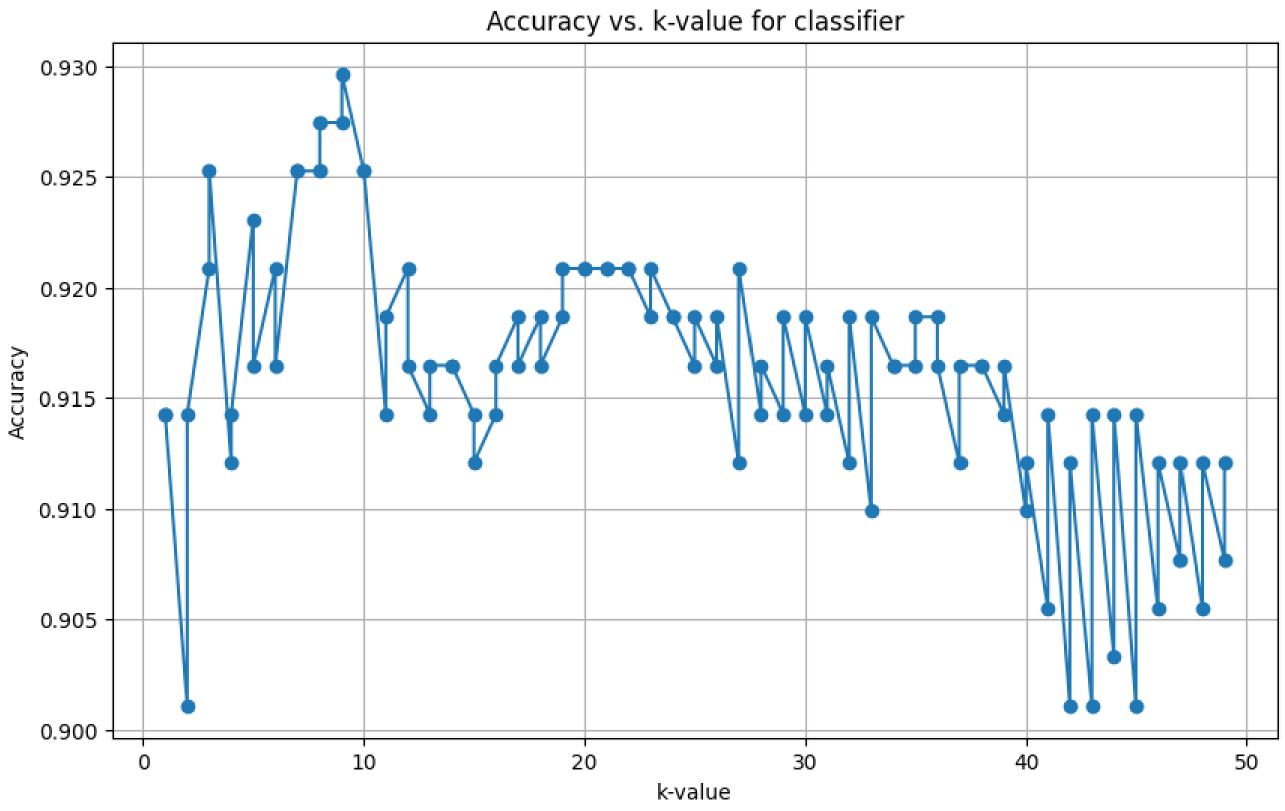
Describe how you found the k value AND plot the k value

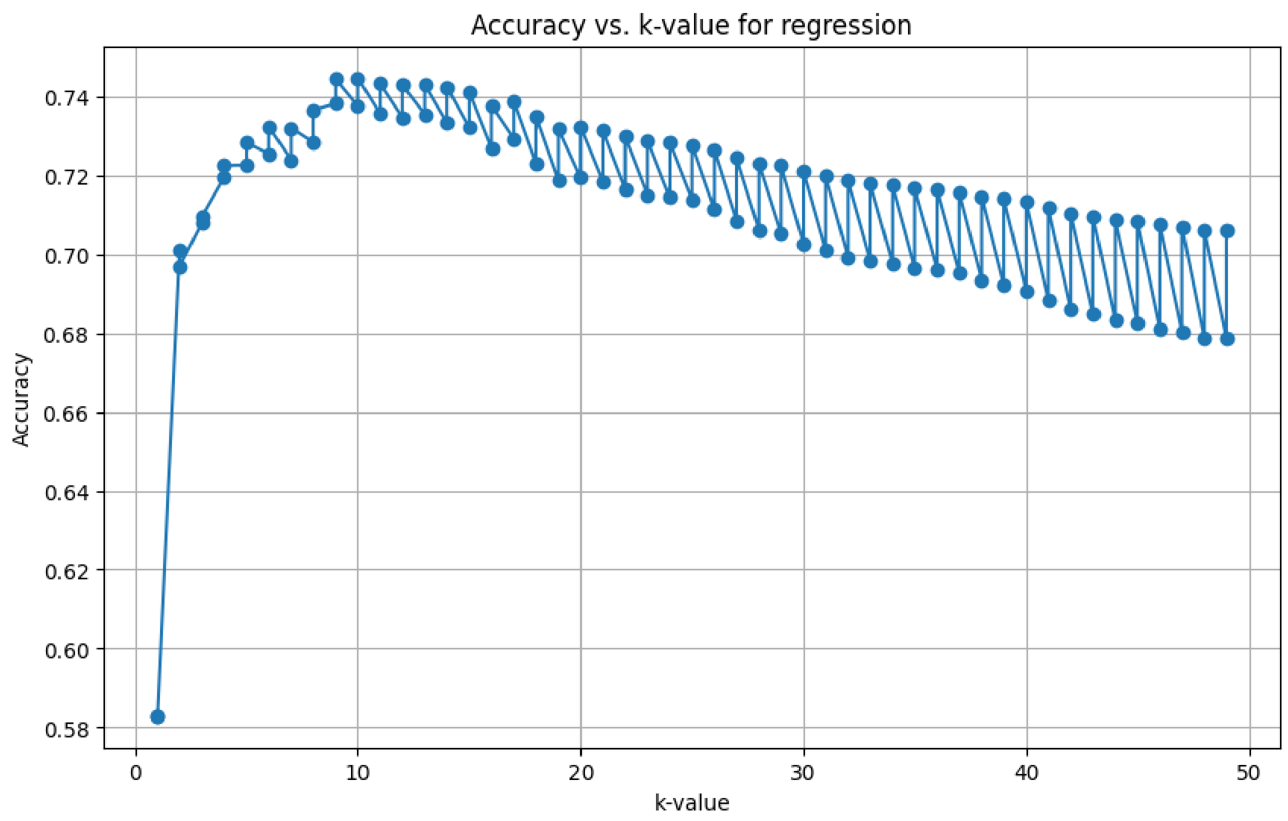
In [9]:
```
# Implement models, fit, and train
```

In [11]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.metrics import accuracy_score, mean_squared_error, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split


# 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)


# Parameter grid for hyperparameter tuning
parameter_grid = {
    'n_neighbors': list(range(1, 50)),
    'weights': ['uniform', 'distance'],
}

# For classification
knn_classifier = KNeighborsClassifier()
classifier_cv = GridSearchCV(knn_classifier, parameter_grid, cv=5)
classifier_cv.fit(X_train, y_train)
best_k_classifier = classifier_cv.best_estimator_
print('Best k value for classifier:', best_k_classifier.n_neighbors)

# Calculate the accuracy for the best classifier
y_predict_classification = best_k_classifier.predict(X_test)
accuracy_classification = accuracy_score(y_test, y_predict_classification)
print('Accuracy for classification:', accuracy_classification)

# For regression
knn_regression = KNeighborsRegressor()
regression_cv = GridSearchCV(knn_regression, parameter_grid, cv=5)
regression_cv.fit(X_train, y_train)
```

```python
best_k_regression = regression_cv.best_estimator_
print('Best k value for regressor:', best_k_regression.n_neighbors)

# Calculate the Mean Squared Error (MSE) for the best regressor
y_predict_regression = best_k_regression.predict(X_test)
mse = mean_squared_error(y_test, y_predict_regression)
print('MSE for regression:', mse)

# k value vs accuracy graph for the classifier
k_values = [param['n_neighbors'] for param in classifier_cv.cv_results_['params']]
accuracy_values = classifier_cv.cv_results_['mean_test_score']

plt.figure(figsize=(10, 6))
plt.plot(k_values, accuracy_values, marker='o')
plt.title('Accuracy vs. k-value for classifier')
plt.xlabel('k-value')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()

# k value vs accuracy graph for the regression
k_values = [param['n_neighbors'] for param in regression_cv.cv_results_['params']]
accuracy_values = regression_cv.cv_results_['mean_test_score']


plt.figure(figsize=(10, 6))
plt.plot(k_values, accuracy_values, marker='o')
plt.title('Accuracy vs. k-value for regression')
plt.xlabel('k-value')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```

```
Best k value for classifier: 9
Accuracy for classification: 0.9473684210526315
Best k value for regressor: 10
MSE for regression: 0.029148951679321022
```





- First create a parameter grid for hyper-parameter tuning.
- Then initialize the KNNClassifier with hyper-parameter tuning.
- Fit the model with X_train and y_train
- Finding the best k value for the classifier
- Calculating the accuracy of the KNN Classifier by predicting the k-value
- Repeated for regression also by calculating mse.
- Plot the graph between k-value and accuracy for classification and regression.

Calculate accuracy for the best estimator (classifiers) and show confusion matrix and classification report

Print the MSE for the regression model

In [12]:
```
# Accuracy for the best classifier
y_predict_classification = best_k_classifier.predict(X_test)
accuracy_classification = accuracy_score(y_test, y_predict_classification)
print('Accuracy for classification:', accuracy_classification)

# Confusion_matrix and classification_report
confusion = confusion_matrix(y_test, y_predict_classification)
classification_rep = classification_report(y_test, y_predict_classification, zero_division=1)

print('Confusion Matrix:')
print(confusion)
print('\nClassification Report:')
print(classification_rep)

# MSE for the regression model
print('MSE for regression:', mse)
```

```
Accuracy for classification: 0.9473684210526315
Confusion Matrix:
[[38  5]
 [ 1 70]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.88      0.93        43
           1       0.93      0.99      0.96        71

    accuracy                           0.95       114
   macro avg       0.95      0.93      0.94       114
weighted avg       0.95      0.95      0.95       114

MSE for regression: 0.029148951679321022
```

- The accuracy of the classifier is calculated using test and predicted data.
- Similarly, calculating the confusion matrix and classification report using test and prediction data.

Task 3: Decision Tree (30 points)

What are the advantages of using Decision Tree classifiers for classification tasks? Provide examples of scenarios where Decision Trees perform well.

Ans:

- The Decision Tree is easy to understand and interpret.
- If there are errors or missing values then also we can create a decision tree.
- Decision Tree does not require normalization or scaling of data.
- The decision tree is very easy to explain.
- It can be used for both categorical and numeric values.
- Used for both classification as well as regression.
- It is fast and efficient in training and predicting.

Example:

In order to estimate the possibility that a borrower would repay a loan, decision trees can be used in the loan approval process. Lenders can utilize this data to make well-informed choices regarding whether or not to approve a loan application.

Lenders would use a historical dataset of loan applications and their results to train a decision tree for loan approval. The borrower's credit score, income, debt-to-income ratio, and work history may all be elements in the dataset. Whether or not the borrower repaid the loan would be the target variable in the dataset.

If (the borrower's credit score is greater than 700 and their debt-to-income ratio is less than 35%),

{then approve the loan application}.

Else if (the borrower's employment history is greater than 2 years and their income is greater than $50,000),

{then approve the loan application}.

Else, {decline the loan application}.

Implement the below code and provide explanation for the code(each section) as per your understanding

Import the necessary libraries.

In [14]:
```python
from sklearn.datasets import load_breast_cancer
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
```

- In the above code, we imported all the required libraries used for data manipulation (numpy, panda), visualization (matplotlib),and other required libraries.

Load the breast cancer dataset from the sklearn library.

```
In [15]: # Load the Breast Cancer dataset
         breast_cancer = load_breast_cancer()
```

- The above code is to get the breast cancer data from sklearn dataset.

Split the dataset into features (X) and target variable (y).

```
In [16]: #Complete the code for X and y below
         X = breast_cancer.data
         y = breast_cancer.target
```

- 'X' is the feature data and 'Y' is the target from the breast cancer dataset.

Create a dataframe(df) from X and y. Display the tail of df.

```
In [17]: df = pd.DataFrame(data=X, columns=breast_cancer.feature_names)
         df['target'] = y
         df.tail()
```

Out[17]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	wor perimet
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	166.
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	155.
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12	126.
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	39.42	184.
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	30.37	59.

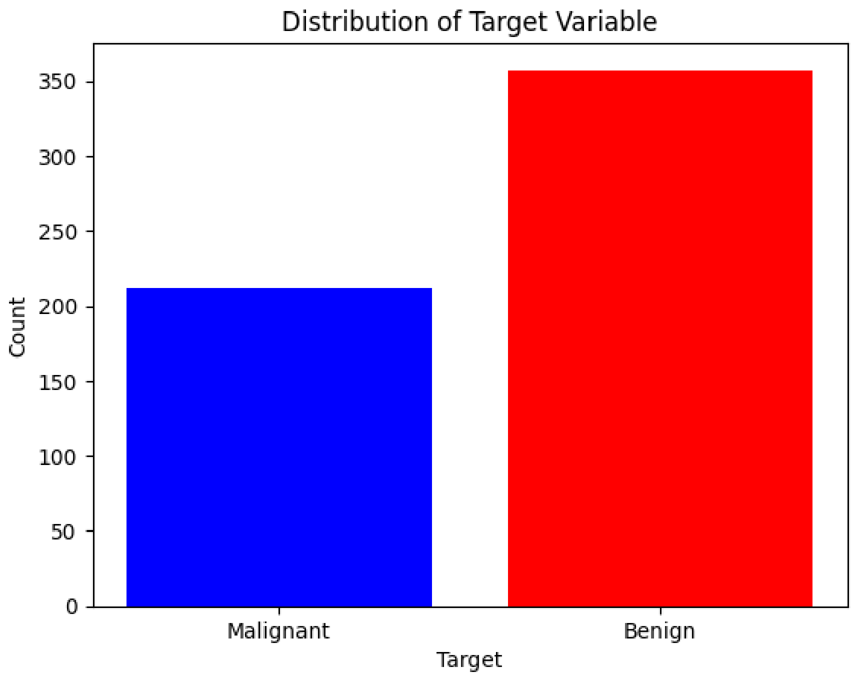
5 rows × 31 columns

- A new column of the data frame is created in the features_names which is labeled as target.
- Tail displays the last 5 rows from the dataset.

Use Matplotlib, Seaborn, or Plotly libraries to visualize the distribution of the target variable using a bar plot.

The plot should represent malignant and benign counts with blue and red bars.

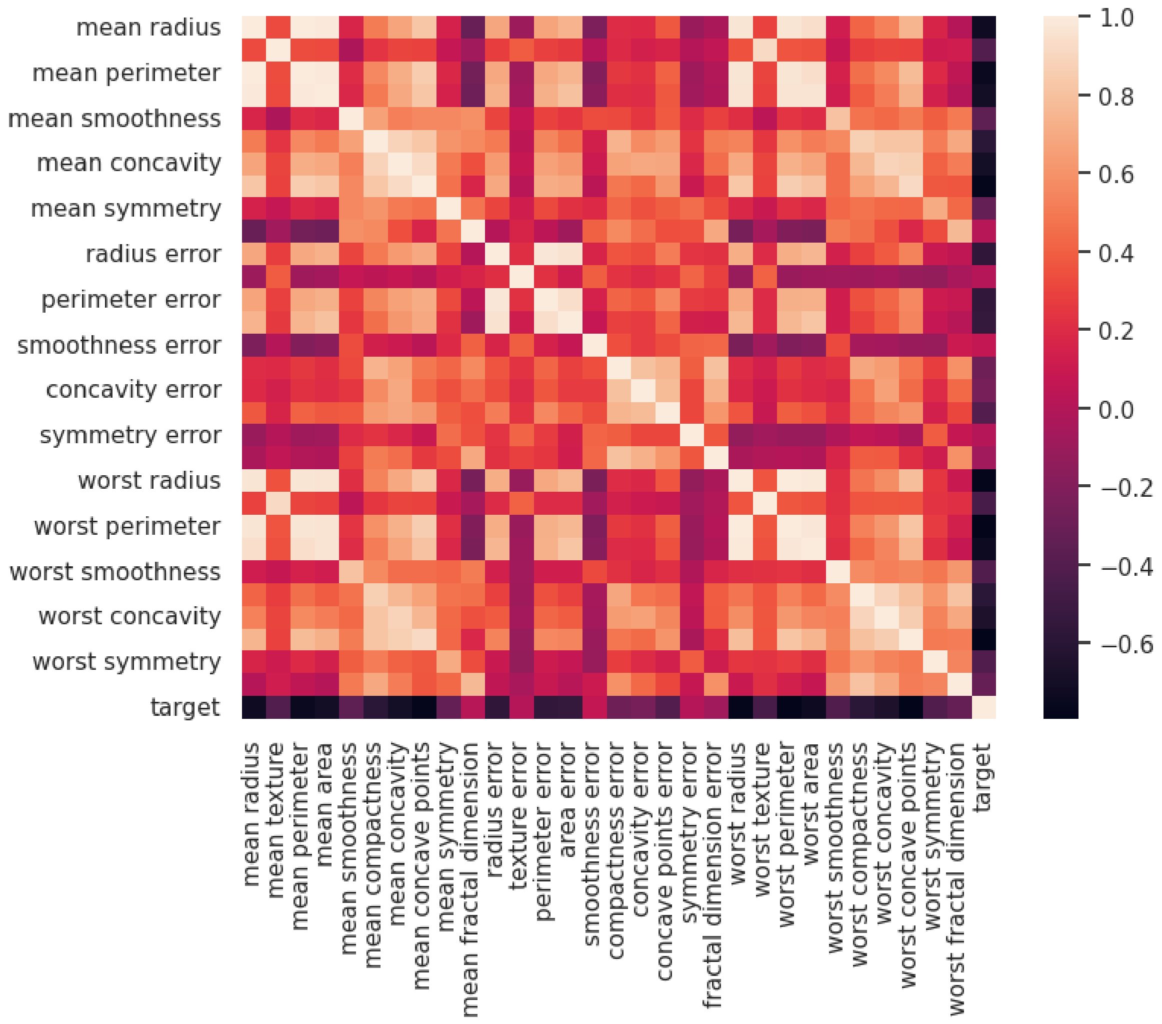
```
In [18]: # Plot the distribution of the target variable
         # For reference the figure should look like the below figure.
         from matplotlib import color_sequences

         #only keep the data with target either '0' or '1'
         df_filtered = df[df['target'].isin([0, 1])]
         target_counts = df_filtered['target'].value_counts()
         # target_counts = df['target'].value_counts()


         plt.bar(target_counts.index, target_counts.values, color=['red', 'blue'])

         plt.xlabel("Target")
         plt.ylabel("Count")
         plt.title("Distribution of Target Variable")

         plt.xticks([0,1], ['Malignant', 'Benign'])

         plt.show()
```



- Plotting the graph for target(Malignant, Benign)[types of breast_cancet] vs. count in blue for Malignant and red for Benign.

Plot the correlation matrix using a heatmap(use library of your choice). What is your understanding of the figure

In [19]:
```
# Plot the correlation matrix
# For reference the figure should look like the below figure.

sns.set(rc={'figure.figsize':(8,6)})
sns.heatmap(df.corr(), annot=False)
plt.show()
```



- In the heatmap, if the color is light then those features are highly related to each other.
- Like if the color is white then the features are very much related.
- Diagonal is white because a feature is always fully related to itself.

Create a dataframe from features that you think are important. Please explain shortly why you think those features are important

In [20]:
```
#Create a new dataframe(new_df) should have the selected feature
new_df= df[['worst radius','worst perimeter','worst area']]
```

- I chose these features because they are highly related to each other. (I came to know by seeing the heatmap above, i.e., they have a light color)

Assign new_df to X and df['target'] to y.

In [21]:
```
X=new_df
y=df['target']
```

Split the data into training and testing sets, with a ratio of 70% for training and 30% for testing, using the train_test_split function. Ensure to consider the important aspects of the split.

What is the main reason behind data splitting??

- Splitting is done to evaluate the performance of the model, ensuring that the accuracy of the model is high.

In [22]:
```
# Split the data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0,shuffle=True,stratify
```

Import the necessary library and create a decision tree classifier object using Hyperparameters: criterion='gini',max_depth=5, min_samples_split=2 and min_samples_leaf=1

In [23]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import precision_score, recall_score, f1_score

decision_tree_clf = DecisionTreeClassifier(criterion='gini', max_depth=5, min_samples_split=2, min_samples_le
```

Fit the training data into the model using fit function.

In [24]:
```
# Train the decision tree classifier
decision_tree_clf.fit(X_train, y_train)
```

Out[24]:
```
▾       DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5)
```

- Fitting X_Train and y_train in decision tree classifier using fit function.

Using the predict function, make prediction on test data.

In [25]:
```
# Make predictions on the test set
y_pred = decision_tree_clf.predict(X_test)
y_pred
```

Out[25]:
```
array([0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0,
       1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1])
```

- Making prediction on X_test data.

Calculate the accuracy score for the model using the sklearn library..

In [26]:
```
# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8362573099415205
```

- The accuracy for the model is **84.7%**

Without using any inbuilt function, calculate the precision of the model.

In [27]:
```
# Calculate the precision of the model
def calculate_precision(y_true, y_pred):
    #Complete the function
    value1 = sum((y_true == 1) & (y_pred == 1))
    value2 = sum((y_true == 0) & (y_pred == 1))
    precision = value1 / (value1 + value2)
    return precision

precision = calculate_precision(y_test, y_pred)
print("Precision:", precision)
```

```
Precision: 0.9072164948453608
```

- value1 is for the case where the actual label and prediction are true.
- value 2 is for the case where the actual label is false and the prediction is true.
- Then calculate and return the precision value which is **90.7%**

Calculate Recall and F1-Score using the sklearn library. https://scikit-learn.org/stable/modules/model_evaluation.html

In [28]:
```
from sklearn.metrics import f1_score

Recall= str(recall_score(y_test, y_pred))
F1_Score= str(f1_score(y_test, y_pred))

print('Recall :'+Recall)
print('F1-SCore :'+F1_Score)
```

```
Recall :0.822429906542056
F1-SCore :0.8627450980392157
```

Task-4 : Random Forest (30 points)

What is the intuition behind the Random Forest algorithm? How does it improve the performance of individual Decision Trees?

Ans:

- The intuition behind the Random Forest algorithm is to combine multiple decision trees.
- It improves the performance of individual Decision Trees because it is robust, avoids overfitting, is relatively fast for the ensemble method, and uses majority for classification hence improving accuracy.

Implement the below code and provide explanation for the code(each section) as per your understanding

Load Wine quality dataset from sklearn library.

In [29]:
```
# Load the Wine Quality dataset
wine = load_wine()
```

- It loaded wine dataset.

Split the dataset into features (X) and target variable (y).

In [30]:
```
# Split the dataset into features (X) and target variable (y)
X = wine.data
y = wine.target
```

- Splitting the dataset into X which is the features and y which is the target.

- print the features of the wine dataset.

Create a dataframe(df) from X and y. Display the tail of df.

In [31]:
```
# Convert the dataset to a pandas DataFrame for easier visualization
df = pd.DataFrame(data=X, columns=wine.feature_names)
df['target'] = y
df.tail()
```

Out[31]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	col
173	13.71	5.65	2.45	20.5	95.0	1.68	0.61	0.52	1.06	
174	13.40	3.91	2.48	23.0	102.0	1.80	0.75	0.43	1.41	
175	13.27	4.28	2.26	20.0	120.0	1.59	0.69	0.43	1.35	
176	13.17	2.59	2.37	20.0	120.0	1.65	0.68	0.53	1.46	
177	14.13	4.10	2.74	24.5	96.0	2.05	0.76	0.56	1.35	

- Converting the wine dataset into a data frame

- print the last 5 rows for the dataset.

Using Seaborn Show two visualizations:

- 1. Pairplot - create a matrix of scatter plots for each pair of features in the dataset.
- 1. Boxplot - create a boxplot to visualize the distribution of alcohol content for each wine quality class.

Pairplot:

What is in the diagonal?

Select two features and describe their relationship

In [32]:
```
# Visualization 1: Pairplot
#Check the below figure for your reference

sns.pairplot(df, hue='target',palette='viridis')
```

```
Output hidden; open in https://colab.research.google.com to view.
```

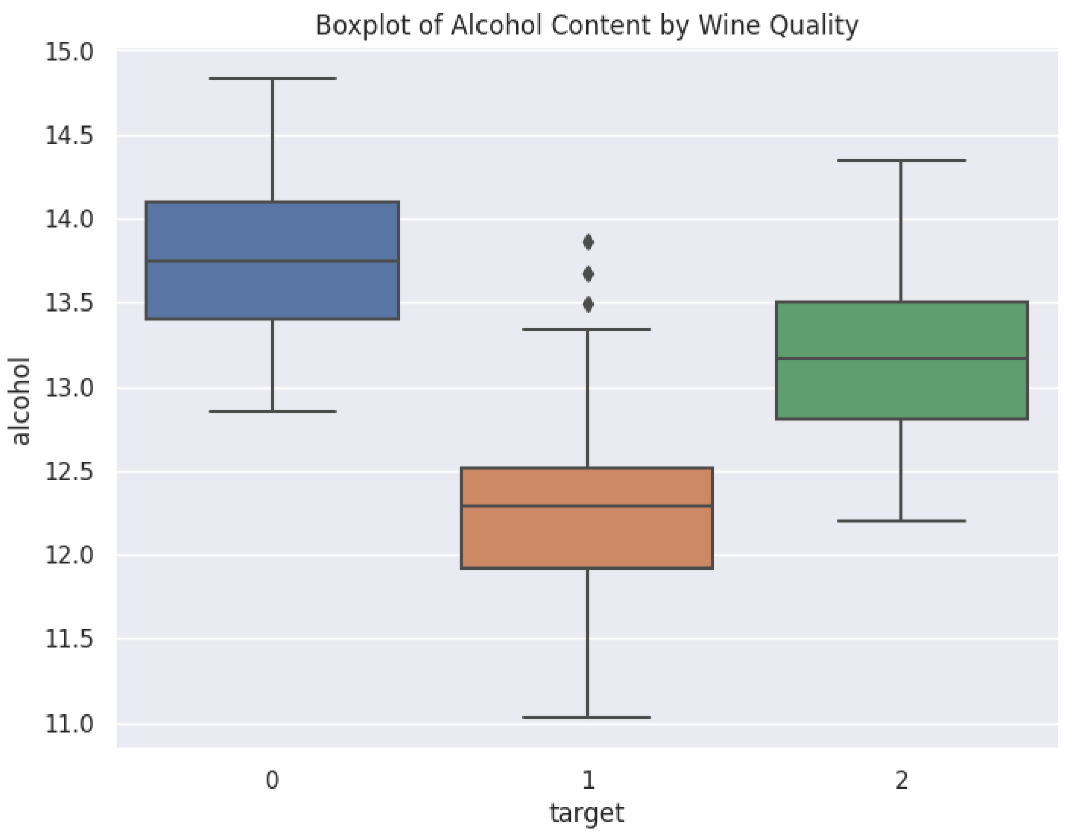
- The diagonal shows the distribution of features in form of histogram.

- The selected features are 'flavanoids', 'hue'.
- As you can see in the above pairplot the both features are not well seperated. So it is little bit hard to seperate because the target point i.e., violet, blue, yellow are getting mixed.

Boxplot:

Define approximately min, max median, Q1, and Q3 from the plot

In [33]:
```
# Visualization 2: Boxplot
#Check the below figure for your refrence
sns.boxplot(x='target', y='alcohol', data=df)
plt.title("Boxplot of Alcohol Content by Wine Quality")
plt.show()
```



Boxplot with target='0' [Blue Box]

- Min: 12.8
- Max: 14.8
- Median: 13.7
- Q1: 13.4
- Q3: 14.1

Boxplot with target='1' [Orange Box]

- Min: 11.1
- Max: 13.4
- Median: 12.3
- Q1: 11.9
- Q3: 12.5

Boxplot with target='2' [Green Box]

- Min: 12.1
- Max: 14.3
- Median: 13.1
- Q1: 12.7
- Q3: 13.5

Split the dataset into Train and Test Sets

In [34]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=123, stratify=y)
```

- Spliting the data into training(70%) and testing(30%).

Create a Random Forest classifier object using Hyperparameters: n_estimators=100, criterion='entropy', max_depth=5, min_samples_split=2, min_samples_leaf=1, random_state=42

Explain each of the hyperparameters (mentioned above) in one sentence. Does the number (42) represent something in the random_state parameter?

Are there any other important parameters for RF other than those mentioned above?

In [35]:
```
# Create a Random Forest Classifier object with n_estimators=100, criterion='entropy', max_depth=5, min_samp
rf_classifier = RandomForestClassifier(
    n_estimators=100,
    criterion='entropy',
    max_depth=5,
    min_samples_split=2,
    min_samples_leaf=1,
    random_state=42
)
```

In [36]:
```
# Train the Random Forest Classifier
rf_classifier.fit(X_train, y_train)
```

Out[36]:
```
▾                RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_depth=5, random_state=42)
```

- Fitting X_train and y_train into random forest classifier using fit() method.

In [37]:
```
# Make predictions on the test set
y_pred = rf_classifier.predict(X_test)
```

In [38]:
```
# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9814814814814815
```

- Predicting the accuracy of random forest classifier.
- The accuracy score is 98.14%
- This means the random forest's prediction is 98.14% accurate when compared to the test data.

Without using any inbuilt function, calculate the Recall of the model.

In [39]:
```
def calculate_recall(y_true, y_pred):
    #Complete the function
    value1 = sum((y_true == 1) & (y_pred == 1))
    value2 = sum((y_true == 1) & (y_pred == 0))

    recall = value1 / (value1 + value2)
    return recall



recall = calculate_recall(y_test, y_pred)
print("Recall:", recall)
```

```
Recall: 0.9523809523809523
```

- value1 is for the case when the actual label is true and the predicted label is also true.
- value2 is for the case when the actual label is true and the predicted label is false.
- Recall the value that we got **0.95**.

Calculate Precision and F1-Score using the sklearn library. https://scikit-learn.org/stable/modules/model_evaluation.html.

In [40]:
```
from sklearn.metrics import precision_score, f1_score

Precision = precision_score(y_test, y_pred, average='weighted')
F1_Score = f1_score(y_test, y_pred, average='weighted')

print('Precision:', Precision)
print('F1-Score:', F1_Score)
```

```
Precision: 0.9824561403508772
F1-Score: 0.9815058961400425
```

Tune RF using GridSearchCV and use the above cells to represent RF prediction results after hyperparameters tunning

In [41]:

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import  recall_score, precision_score, f1_score

# Define the hyperparameter grid for GridSearchCV
param_grid = {
'max_depth': [3, 4,5, 6],
'min_samples_leaf': [0.04, 0.06, 0.08],
'max_features': [0.2, 0.4,0.6, 0.8],
'criterion':['entropy', "gini"],
'min_samples_split':[3,4,5],
}

# Create the RF classifier
rf_classifier = RandomForestClassifier()

# Create GridSearchCV with the classifier and hyperparameter grid
grid_search_cv = GridSearchCV(estimator=rf_classifier,
param_grid=param_grid,
scoring='accuracy',
cv=5,
n_jobs=-1)

# Fit GridSearchCV to the training data
grid_search_cv.fit(X_train, y_train)

# Getting the best possible estimator
best_rf_classifier = grid_search_cv.best_estimator_

# Making the prediction
y_predict_tuned = best_rf_classifier.predict(X_test)

# Calculate values
accuracy_tuned = accuracy_score(y_test, y_predict_tuned)
recall_tuned = recall_score(y_test, y_predict_tuned, average='weighted')
precision_tuned = precision_score(y_test, y_predict_tuned, average='weighted')
f1_score_tuned = f1_score(y_test, y_predict_tuned, average='weighted')


print("Tuned RF Classifier Evaluation:")
print("Accuracy:", accuracy_tuned)
print("Recall:", recall_tuned)
print("Precision:", precision_tuned)
print("F1-Score:", f1_score_tuned)
```

```
Tuned RF Classifier Evaluation:
Accuracy: 0.9814814814814815
Recall: 0.9814814814814815
Precision: 0.9824561403508772
F1-Score: 0.9815058961400425
```

In [42]:

```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```