Libraries

```
# NumPy for arrays and math functions
import numpy as np

# Pandas for data manipulation
import pandas as pd

# Matplotlib and Seaborn for plotting
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler  # Normalizes features to a specific range (e.g., 0 to 1)
# Split data into training and testing sets
from sklearn.model_selection import train_test_split

# Standardize features by removing the mean and scaling to unit variance
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV  # Optimizes hyperparameters using cross-validation techniques
# PCA for dimensionality reduction
from sklearn.decomposition import PCA

# Randomized search for hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV

from sklearn.neural_network import MLPClassifier  # Brings in the Multi-Layer Perceptron classifier for neural network modeling

# Evaluation metrics: classification report and confusion matrix
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Manage warnings
import warnings
warnings.filterwarnings("ignore", message="The total space of parameters .*")
```

*Importing Google Drive *

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Import dataset

```
dataframe = pd.read_csv('/content/drive/MyDrive/ML Work/H /breast-cancer.csv')
dataframe.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	con 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

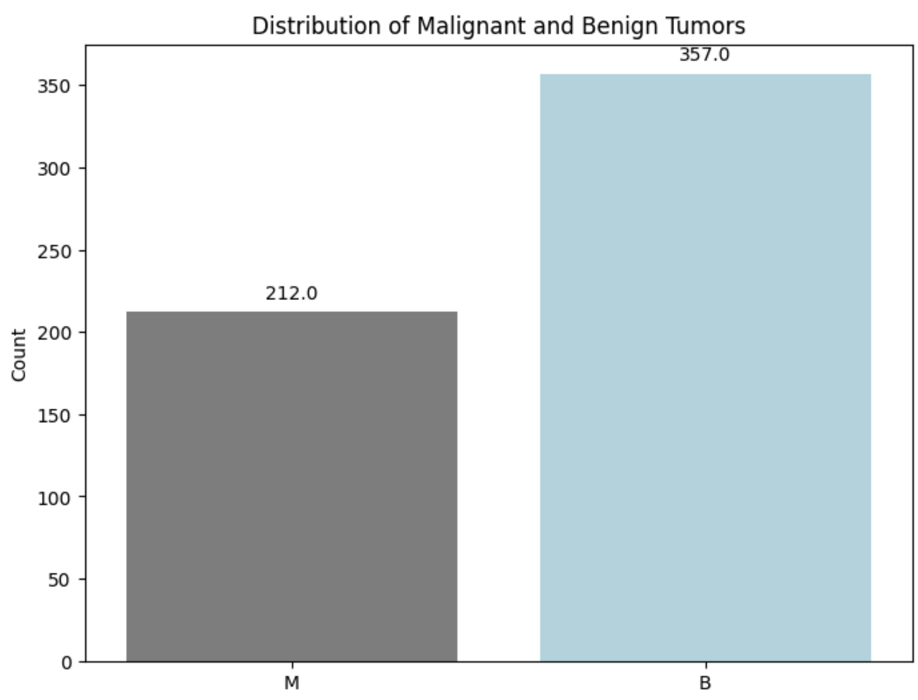
5 rows × 32 columns

```
dataframe.info()
```

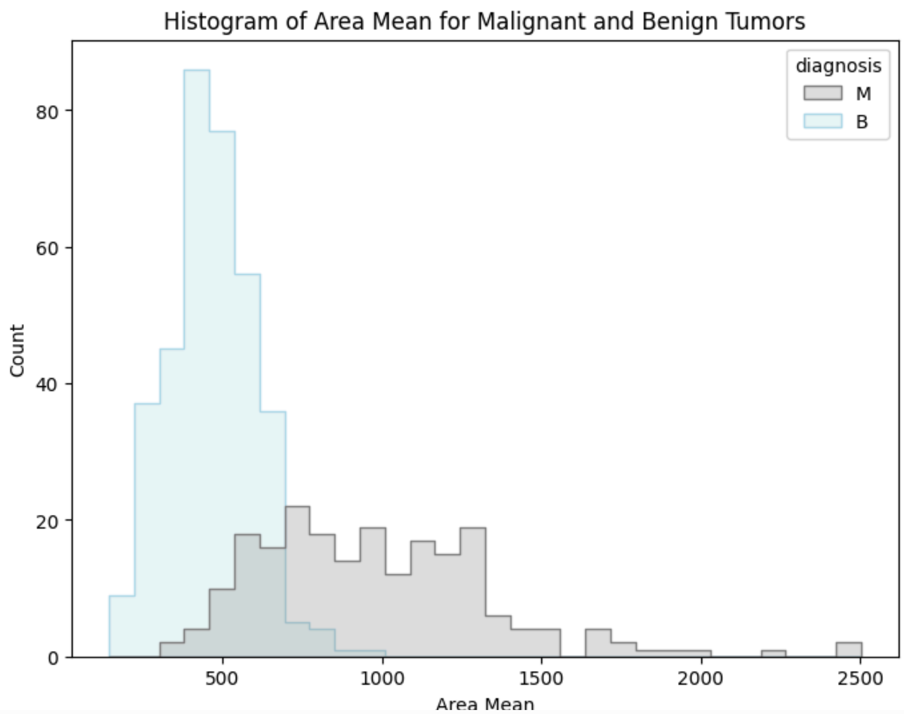
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    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
```

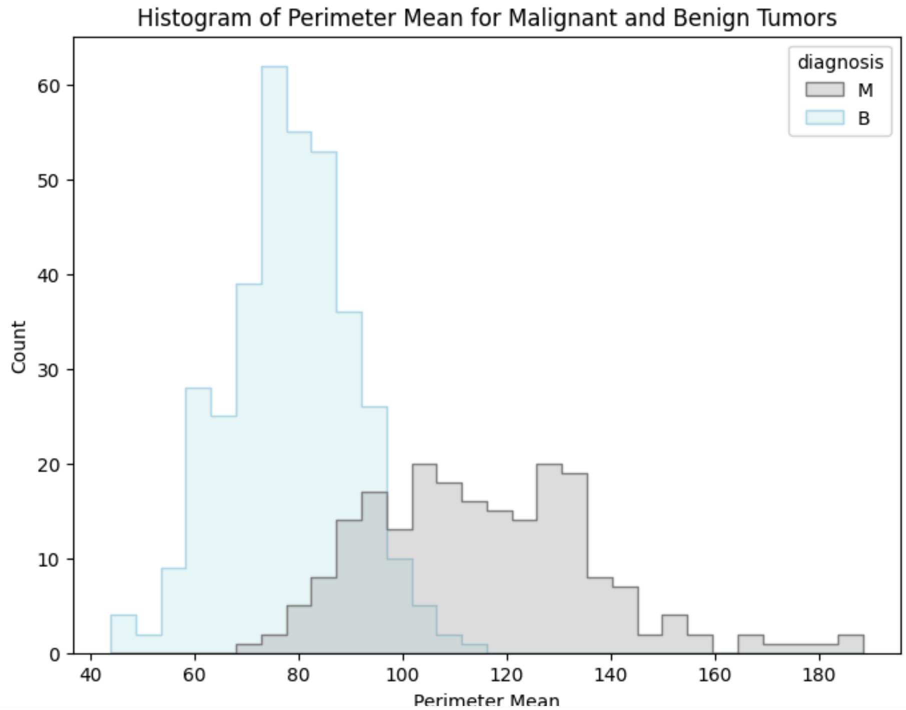
```
plt.figure(figsize=(8, 6))
ax = sns.countplot(x='diagnosis', data=dataframe, hue='diagnosis', palette=['gray', 'lightblue'])
plt.title('Distribution of Malignant and Benign Tumors')
plt.xlabel('Diagnosis')
plt.ylabel('Count')

# Annotate counts on each bar
for p in ax.patches:
    height = p.get_height()
    ax.annotate(f'{height}', (p.get_x() + p.get_width() / 2., height),
                ha='center', va='center', xytext=(0, 10), textcoords='offset points')

plt.show()
```



```
plt.figure(figsize=(8, 6))

# Creating a histogram for 'area_mean' with separation based on 'diagnosis'
sns.histplot(data=dataframe, x='area_mean', hue='diagnosis', element='step', palette=['gray', 'lightblue'], bins=30)

plt.title('Histogram of Area Mean for Malignant and Benign Tumors')
plt.xlabel('Area Mean')
plt.ylabel('Count')
plt.show()
```



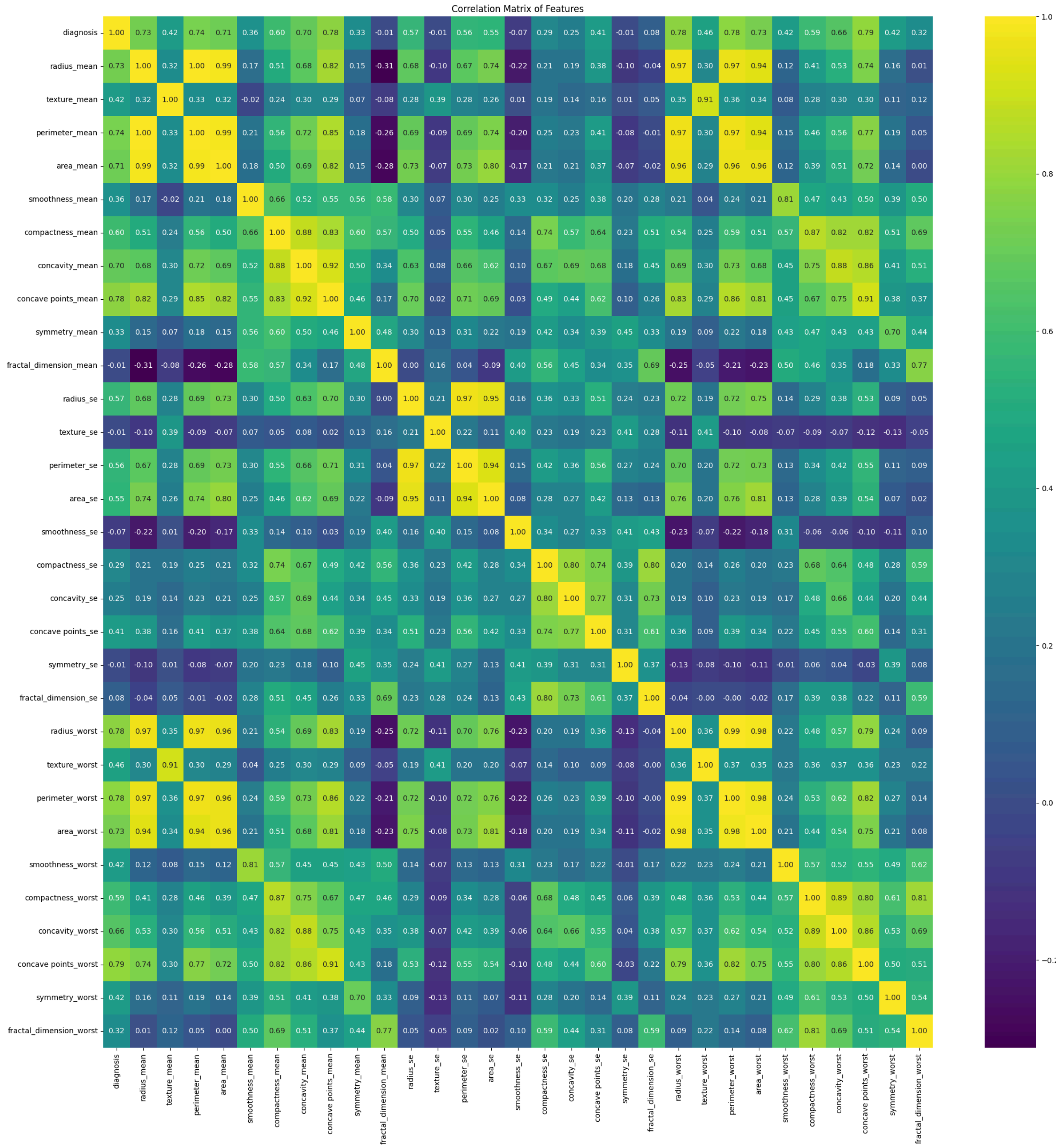

```
plt.figure(figsize=(8, 6))

# Creating a histogram for 'perimeter_mean' with separation based on 'diagnosis'
sns.histplot(data=dataframe, x='perimeter_mean', hue='diagnosis', element='step', palette=['gray', 'lightblue'], bins=30)

plt.title('Histogram of Perimeter Mean for Malignant and Benign Tumors')
plt.xlabel('Perimeter Mean')
plt.ylabel('Count')
plt.show()
```




```
dataframe['diagnosis'] = (dataframe['diagnosis'] == 'M').astype(int) #encode the label into 1/0
```

Correlation Matrix

```
# Calculating the correlation matrix
corr = dataframe.drop('id', axis=1).corr()

# Setting up the matplotlib figure
plt.figure(figsize=(25, 25))

# Drawing the heatmap with a new color palette (viridis)
sns.heatmap(corr, annot=True, fmt=".2f", cmap='viridis')

# Displaying the plot with a title
plt.title('Correlation Matrix of Features')
plt.show()
```



Feature Importance

```
# Calculate correlation of all features with the target variable
correlation_with_target = dataframe.corr()['diagnosis'].abs()

# Drop the target variable itself from the correlation
correlation_with_target = correlation_with_target.drop('diagnosis')
```

```
# Select the 8 features with the highest correlation
top_8_features_corr = correlation_with_target.sort_values(ascending=False).head(8)
print("Top 8 features based on correlation:")
print(top_8_features_corr)
```

```
Top 8 features based on correlation:
concave points_worst    0.793566
perimeter_worst         0.782914
concave points_mean     0.776614
radius_worst            0.776454
perimeter_mean          0.742636
area_worst              0.733825
radius_mean             0.730029
area_mean               0.708984
Name: diagnosis, dtype: float64
```

```
# List of selected feature names (using the top correlated features for breast cancer classification)
selected_features = [
    'concave points_worst',
    'perimeter_worst',
    'concave points_mean',
    'radius_worst',
    'perimeter_mean',
    'area_worst',
    'radius_mean',
    'area_mean'
]

# Extract the features and target variable
X = dataframe[selected_features]  # Feature set
y = dataframe['diagnosis']          # Target variable

# Verify the extracted data
print("Shape of feature set (X):", X.shape)
print("Shape of target variable (y):", y.shape)
```

```
Shape of feature set (X): (569, 8)
Shape of target variable (y): (569,)
```

Data Set Separation

```
# 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, stratify=y
)

# Verify the shapes of the splits
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (455, 8)
Shape of X_test: (114, 8)
Shape of y_train: (455,)
Shape of y_test: (114,)
```

Data Normalization

```
# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Fit the scaler on the training data and transform both training and testing sets
X_train_normalized = scaler.fit_transform(X_train)
X_test_normalized = scaler.transform(X_test)

# Verify normalization
print("First 5 rows of normalized training data:\n", X_train_normalized[:5])
print("First 5 rows of normalized testing data:\n", X_test_normalized[:5])
```

```
First 5 rows of normalized training data:
 [[0.34278351 0.36550625 0.16515905 0.40056919 0.40709004 0.23712151
  0.42780065 0.27753977]
 [0.32271478 0.18208078 0.18389662 0.19815012 0.24227766 0.08943669
  0.25268588 0.13599152]
 [0.19247423 0.20528911 0.09279324 0.2301672  0.26839887 0.11320291
  0.27776989 0.1573701 ]
 [0.85051546 0.37347477 0.48265408 0.29953753 0.40294382 0.15913783
  0.37479294 0.22969247]
 [0.51202749 0.44568953 0.38424453 0.47598719 0.54115127 0.29930201
  0.55038099 0.40318134]]
First 5 rows of normalized testing data:
 [[0.30783505 0.16599432 0.13036779 0.17395945 0.20420151 0.07994986
  0.20961711 0.11020148]
 [0.7233677  0.57218985 0.65109344 0.62789043 0.65724553 0.44848604
  0.66065597 0.51770944]
 [0.42989691 0.31221674 0.26824056 0.32159374 0.43196738 0.16621608
  0.4348999  0.27359491]
 [0.53780069 0.29428756 0.29020875 0.29882604 0.35028678 0.15758946
  0.35207535 0.2116649 ]
 [0.26676976 0.23048957 0.16222664 0.25329064 0.29403635 0.12790012
  0.3028539  0.1754825 ]]
```

Model Training of MLP

```
# Define the ANN model (MLPClassifier)
mlp = MLPClassifier(max_iter=1000, random_state=42)

# Set up the parameter grid for GridSearchCV
param_grid = {
    'hidden_layer_sizes': [(50,),(75,), (100,), (50, 50)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.001, 0.01],
    'learning_rate': ['constant', 'adaptive']
}


# Perform GridSearchCV
grid_search_ann = GridSearchCV(mlp, param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=2)
grid_search_ann.fit(X_train_normalized, y_train)

# Retrieve the best model and parameters
best_ann = grid_search_ann.best_estimator_
best_params_ann = grid_search_ann.best_params_
print("Best Parameters for ANN:", best_params_ann)

# Make predictions on the test set
y_pred_ann = best_ann.predict(X_test_normalized)

results_df = pd.DataFrame(grid_search_ann.cv_results_)
for index, row in results_df.iterrows():
    print(f"Params: {row['params']} | Mean Test Score: {row['mean_test_score']:.4f} | Std Test Score: {row['std_test_score']:.4f}")
```

```
Params: {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50), 'learning_rate': 'adaptive', 'solver': 'adam'} | Mean
Params: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (50,), 'learning_rate': 'constant', 'solver': 'sgd'} | Mean Test
Params: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (50,), 'learning_rate': 'constant', 'solver': 'adam'} | Mean Tes
Params: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (50,), 'learning_rate': 'adaptive', 'solver': 'sgd'} | Mean Test
Params: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (50,), 'learning_rate': 'adaptive', 'solver': 'adam'} | Mean Tes
Params: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (75,), 'learning_rate': 'constant', 'solver': 'sgd'} | Mean Test
Params: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (75,), 'learning_rate': 'constant', 'solver': 'adam'} | Mean Tes
Params: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (75,), 'learning_rate': 'adaptive', 'solver': 'sgd'} | Mean Test
Params: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (75,), 'learning_rate': 'adaptive', 'solver': 'adam'} | Mean Tes
Params: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver': 'sgd'} | Mean Tes
Params: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver': 'adam'} | Mean Te
Params: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver': 'sgd'} | Mean Tes
Params: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver': 'adam'} | Mean Te
Params: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (50, 50), 'learning_rate': 'constant', 'solver': 'sgd'} | Mean T
Params: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (50, 50), 'learning_rate': 'constant', 'solver': 'adam'} | Mean
Params: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (50, 50), 'learning_rate': 'adaptive', 'solver': 'sgd'} | Mean T
Params: {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (50, 50), 'learning_rate': 'adaptive', 'solver': 'adam'} | Mean
Params: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (50,), 'learning_rate': 'constant', 'solver': 'sgd'} | Mean Test
Params: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (50,), 'learning_rate': 'constant', 'solver': 'adam'} | Mean Test
Params: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (50,), 'learning_rate': 'adaptive', 'solver': 'sgd'} | Mean Test
Params: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (50,), 'learning_rate': 'adaptive', 'solver': 'adam'} | Mean Test
Params: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (75,), 'learning_rate': 'constant', 'solver': 'sgd'} | Mean Test
Params: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (75,), 'learning_rate': 'constant', 'solver': 'adam'} | Mean Test
Params: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (75,), 'learning_rate': 'adaptive', 'solver': 'sgd'} | Mean Test
Params: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (75,), 'learning_rate': 'adaptive', 'solver': 'adam'} | Mean Test
Params: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver': 'sgd'} | Mean Test
Params: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver': 'adam'} | Mean Tes
Params: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver': 'sgd'} | Mean Test
Params: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver': 'adam'} | Mean Tes
Params: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (50, 50), 'learning_rate': 'constant', 'solver': 'sgd'} | Mean Te
Params: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (50, 50), 'learning_rate': 'constant', 'solver': 'adam'} | Mean T
Params: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (50, 50), 'learning_rate': 'adaptive', 'solver': 'sgd'} | Mean Te
Params: {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (50, 50), 'learning_rate': 'adaptive', 'solver': 'adam'} | Mean T
```

```
#  Evaluate the model
test_accuracy_ann = accuracy_score(y_test, y_pred_ann)
print("\nTest Set Accuracy (ANN):", test_accuracy_ann)
print("\nClassification Report (ANN):\n", classification_report(y_test, y_pred_ann))

# Visualize the confusion matrix and classification report
# Confusion Matrix
conf_matrix_ann = confusion_matrix(y_test, y_pred_ann)

plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix_ann, annot=True, fmt='d', cmap='Blues', xticklabels=True, yticklabels=True)
plt.title("ANN Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

```
Test Set Accuracy (ANN): 0.956140350877193

Classification Report (ANN):
               precision    recall  f1-score   support

           0       0.97      0.96      0.97        72
           1       0.93      0.95      0.94        42

    accuracy                           0.96       114
   macro avg       0.95      0.96      0.95       114
weighted avg       0.96      0.96      0.96       114
```

ANN Confusion Matrix