Student Name: Oyatokun Modupeola

Student ID: C0895705

Assignment: Breast Cancer Data Analysis and Streamlit App

This report outlines the steps taken in analyzing a breast cancer dataset. The project encompasses loading the dataset, examining its structure, cleaning and preparing the data, visualizing relationships, and constructing a machine learning model to make predictions.

The first step is to create a new directory for the breast cancer assignment in VS Code. Initialized a Git repository.

Created and activated a virtual environment

Loading Libraries and Dataset

Importing necessary libraries and loading the dataset.

tensorflow==2.15.0
pandas
numpy
scikit-learn
tensorboard
matplotlib
streamlit
scikeras
seaborn

Load the dataset (the data set was gotten from Kaggle)

```
## Load the dataset from kaggle from a CSV File
data=pd.read_csv("data.csv")
data.head()
```
0.0s

Exploring the Dataset

The first step in exploring the dataset involves viewing the first few rows and getting summary statistics to understand the data's structure and basic characteristics.

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809
...	...	...	...	...	...	...	...	...	...	...
564	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726
565	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752
566	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590
567	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397
568	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587

569 rows × 31 columns

Data Cleaning and Preparation

Data cleaning and preparation are essential steps to ensure the dataset is ready for analysis. This may involve dropping irrelevant columns or encoding categorical variables.

Dropping irrelevant columns. The 'Unnamed: 32' column is dropped.

```
### Drop irrelevant columns
data=data.drop(['Unnamed: 32','id'],axis=1)
data
```

0.0s

The code data.diagnosis.replace (to_replace=dict(M=1, B=0), inplace=True) replaces 'M' with 1 and 'B' with 0 in the 'diagnosis' column of the DataFrame data. This is typically done to convert categorical values (like 'M' and 'B') into numerical values (1 and 0) which are easier to work with in machine learning models. By performing this replacement, the 'diagnosis' column now contains numerical values (1 for 'Malignant' and 0 for 'Benign'), making it suitable for training machine learning models or performing statistical analysis where numerical inputs are expected.

Replacing categorical variables to Numerical

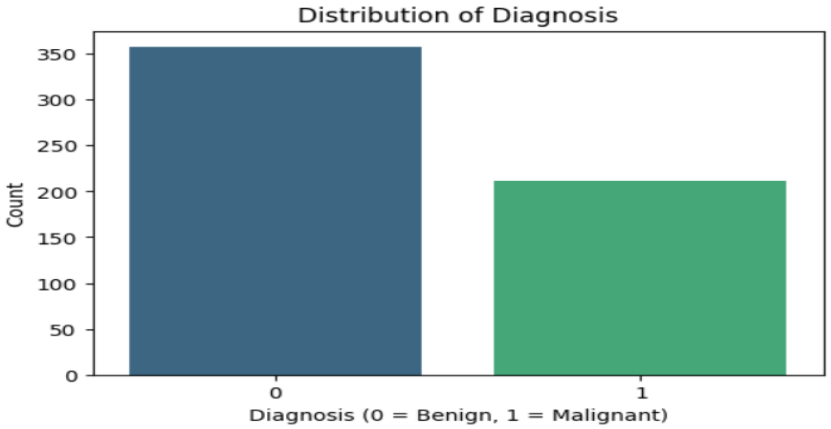
```
# The 'diagnosis' column was replaced (M is mapped to 1 and B to 0).
data.diagnosis.replace(to_replace = dict(M = 1, B = 0), inplace = True)
data
```

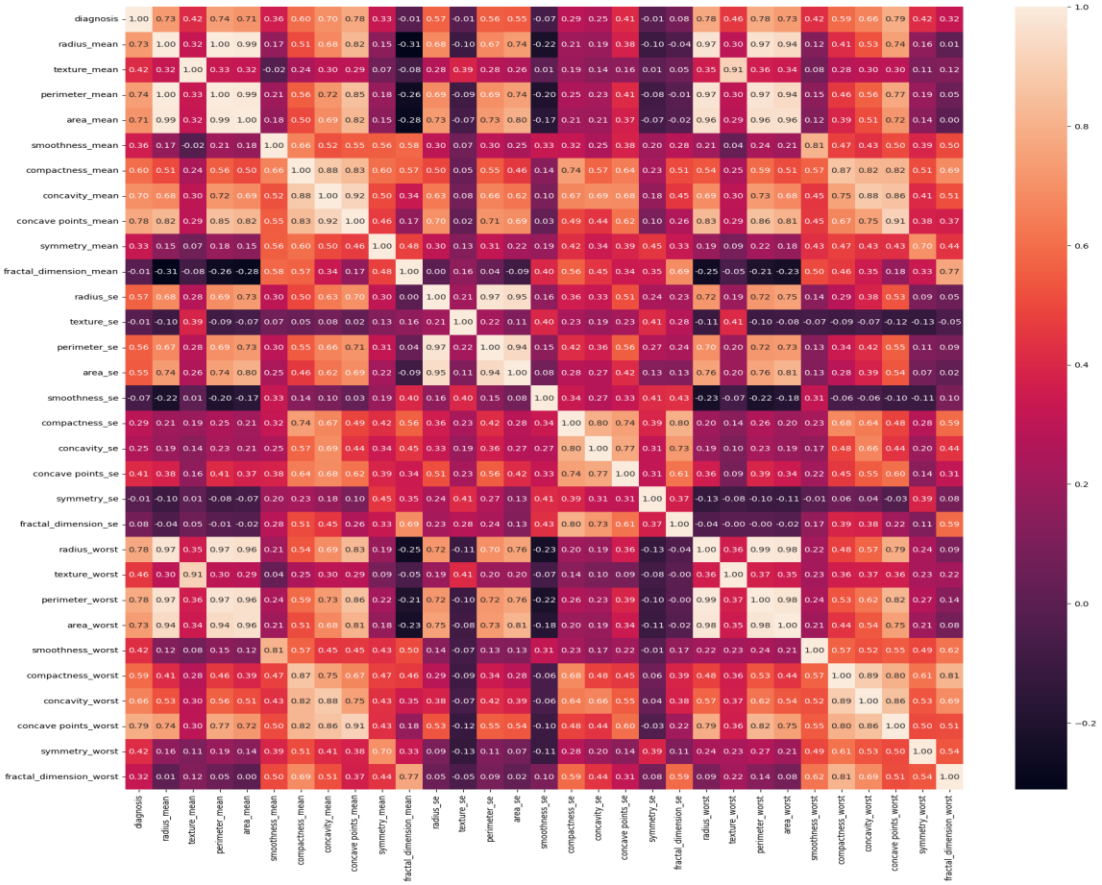
0.0s

Data Visualization

Visualize the distribution of the target variable

```
# Visualize the distribution of the target variable

# Set the figure size
plt.figure(figsize=(6, 4))

# Create a count plot for the 'diagnosis' column with a 'viridis' color palette
sns.countplot(x='diagnosis', data=data, palette='viridis')

# Set the title and labels of the plot
plt.title('Distribution of Diagnosis')
plt.xlabel('Diagnosis (0 = Benign, 1 = Malignant)')
plt.ylabel('Count')

# Display the plot
plt.show()
```





This visualization is particularly useful for understanding relationships between different features in dataset, which can guide feature selection, model building, and further analysis.The heatmap provides a visual representation of how strongly each variable is correlated with every other variable in the dataset.

Dividing the dataset to independent and dependent features. This is done by dropping the target variable which in the case is diagnosis.

```
## DiVide the dataset into independent and dependent features
X=data.drop('diagnosis',axis=1)
y=data['diagnosis']
```

Then split the dataset to train and test

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

Pickle is a Python module that implements binary protocols for serializing and deserializing Python objects.

Purpose Pickle

Reusability: By saving these objects to files, you can reuse the fitted label_encoder_diagnosis and scaler without needing to refit them each time you run your code. This is useful for consistent preprocessing in different sessions or environments.

Efficiency: Loading a serialized object is faster than refitting it, which can save time, especially when dealing with large datasets or complex preprocessing steps.

Sharing: Serialized objects can be easily shared with others, allowing them to use the same preprocessing steps without needing access to the original data.

```
##pickle

pickle.dump(label_encoder_diagnosis, open('label_encoder_diagnosis.pkl','wb'))

pickle.dump(scaler,open('scaler.pkl','wb' ))
```

Features Selection

The select_features function employs SelectKBest with the f_classif scoring function to identify the top k features most pertinent to the target variable (y_train). It fits the SelectKBest object to the training data, then transforms both the training and testing datasets to retain only the selected features. The function returns these transformed datasets and the fitted feature selector. This

approach reduces the data's dimensionality, emphasizes the most significant features, and can enhance the performance of machine learning models.

```
#Selecting the best Features

from sklearn.feature_selection import SelectKBest, f_classif

def select_features(X_train, y_train, X_test, k=10):
    fs = SelectKBest(score_func=f_classif, k=k)
    fs.fit(X_train, y_train)
    X_train_fs = fs.transform(X_train)
    X_test_fs = fs.transform(X_test)
    return X_train_fs, X_test_fs, fs

# Assuming the preprocessed data is saved in variables X_train, y_train, X_test
X_train_fs, X_test_fs, fs = select_features(X_train, y_train, X_test)
```

Save the selected features

```
# Save the best features selected
with open('feature_selector.pkl', 'wb') as f:
    pickle.dump(fs, f)

# Save the selected features data
with open('selected_features_data.pkl', 'wb') as f:
    pickle.dump((X_train_fs, X_test_fs, y_train, y_test), f)
```

Model Building and Grid Search CV for Model Tuning

This process is crucial for improving model performance by finding the optimal hyperparameters for the neural network classifier.

This is done using scikit-learn's GridSearchCV and MLPClassifier

GridSearchCV: This is a module from scikit-learn that allows exhaustive search over specified parameter values for an estimator.

MLPClassifier: This is a Multi-Layer Perceptron classifier, which is a type of artificial neural network model.

Parameter_space: This dictionary defines the hyperparameters to be tuned and their possible values.

Hidden_layer_sizes: Defines the architecture of the neural network. Different configurations of hidden layers and neurons are specified.

Activation: Activation functions for the hidden layers.

Solver: Optimization algorithms for weight adjustment.

Alpha: Regularization term to prevent overfitting.

Learning_rate: Strategies for learning rate adaptation.

MLPClassifier(max_iter=100): Initializes the MLPClassifier with a maximum of 100 iterations.

GridSearchCV(mlp, parameter_space, n_jobs=-1, cv=3):

mlp: The estimator (MLPClassifier) to be optimized.

parameter_space: The hyperparameter space to search.

n_jobs=-1: Uses all available CPUs to perform the computation.

cv=3: Uses 3-fold cross-validation.

Fits the grid search model on the training data. print the Outputs the best combination of hyperparameters found. (**return clf**) Returns the fitted grid search object.

Model Building and Grid Search CV for Model Tuning

```
#import the libraries
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier

def grid_search_cv(X_train, y_train):
    parameter_space = {
        'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
        'activation': ['tanh', 'relu'],
        'solver': ['sgd', 'adam'],
        'alpha': [0.0001, 0.05],
        'learning_rate': ['constant','adaptive'],
    }

    mlp = MLPClassifier(max_iter=100)
    clf = GridSearchCV(mlp, parameter_space, n_jobs=-1, cv=3)
    clf.fit(X_train, y_train)

    print('Best parameters found:\n', clf.best_params_)
    return clf
```

Saving the Model:

- The trained model (clf) is saved to a file named clf_model.pkl using joblib.dump().
- joblib.dump(clf, 'clf_model.pkl'): Serializes and saves the trained model to the specified file.

```
# load selected features data is saved in variables X_train_fs, y_train
with open('selected_features_data.pkl', 'rb') as f:
    X_train_fs, X_test_fs, y_train, y_test = pickle.load(f)
clf = grid_search_cv(X_train_fs, y_train)
#saved the trained model

# clf is trained MLPClassifier model
import joblib
from sklearn.neural_network import MLPClassifier

joblib.dump(clf, 'clf_model.pkl')
```

Loading Selected Features Data

- The selected feature data and corresponding labels, which were previously saved in a file named selected_features_data.pkl, are loaded using pickle.load().
- with open('selected_features_data.pkl', 'rb') as f: Opens the file in binary read mode ('rb').
- X_train_fs, X_test_fs, y_train, y_test: These variables store the loaded training and testing feature sets and labels.

Performing Grid Search Cross-Validation

- The grid_search_cv function is called with the training features (X_train_fs) and training labels (y_train).
- This function performs hyperparameter tuning using GridSearchCV and returns the best estimator (clf), which is a trained MLPClassifier model.

Implementing an Artificial Neural Network (ANN) Model

Making Predictions and Evaluating the Model

```
#Model Evaluation
from sklearn.metrics import classification_report, accuracy_score

def train_evaluate_ann(X_train_fs, y_train, X_test_fs, y_test, clf):
    y_pred = clf.predict(X_test_fs)
    print("Accuracy: ", accuracy_score(y_test, y_pred))
    print(classification_report(y_test, y_pred))
```

Defining the train_evaluate_ann Function

X_train_fs: The training feature set (with selected features).

y_train: The training target set.

X_test_fs: The testing feature set (with selected features).

y_test: The testing target set.

clf: The trained classifier model.

Prediction:

y_pred = clf.predict(X_test_fs):

This line uses the predict method of the classifier clf to generate predictions for the test feature matrix X_test_fs.

y_pred will contain the predicted labels for the test set.

Accuracy Calculation:

accuracy_score(y_test, y_pred) calculates the accuracy of the predictions by comparing the true labels (y_test) with the predicted labels (y_pred).

classification_report(y_test, y_pred) generates a detailed classification report that includes precision, recall, and F1-score for each class, as well as overall metrics such as macro and weighted averages.

```
# Assuming the grid search CV model is saved in variable clf
train_evaluate_ann(X_train_fs, y_train, X_test_fs, y_test, clf)
```

✓ 0.0s

```
Accuracy:  0.9824561403508771
              precision    recall  f1-score   support

           0       0.99      0.99      0.99        71
           1       0.98      0.98      0.98        43

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

Accuracy measures the proportion of correct predictions (both true positives and true negatives) out of all predictions

In this case, the accuracy is approximately 98.25%, meaning that the classifier correctly predicted the labels for about 98.25% of the test samples.

Prediction.ipynb

import libraries

```
#import libraries
import pandas as pd
import joblib
import pickle
```

Load the saved models

```
#Load the save models
model = joblib.load('clf_model.pkl')
sc = joblib.load('scaler.pkl')

with open('feature_selector.pkl', 'rb') as f:
    fs = pickle.load(f)
```

This function, predict_cancer, is designed to predict whether an input sample indicates breast cancer using a pre-trained machine learning model.This function takes raw input data, scales it, selects relevant features, and uses a pre-trained model to predict the probability that the input data indicates breast cancer.

By Inputting the input data,

The percentage of the prediction gives

```
# Predict and print the result
prediction_proba = predict_cancer(input_data)
if prediction_proba < 0.5:
    print('Likely to be Malignant')
else:
    print('Likely Benign')
```

✓ 0.0s

```
Likely Benign
```

Application for Deployment - Building a Streamlit App Locally

Develop a simple Streamlit application where users can engage with the breast cancer dataset and observe model predictions. Integrate model predictions and user interaction seamlessly within the Streamlit app.

```python
#Building a Streamlit App Locally
# app.py
import streamlit as st
import pandas as pd
import joblib
import pickle

# Load the trained model
clf = joblib.load('clf_model.pkl')

# Load the scaler
sc = joblib.load('scaler.pkl')

# Load the feature selector
with open('feature_selector.pkl', 'rb') as f:
    fs = pickle.load(f)

# Load the dataset to get column names and statistics for user inputs
df = pd.read_csv('data.csv')

st.title('Breast Cancer Prediction App')
st.write('This is a simple web app to predict breast cancer.')

# User input
st.sidebar.header('User Input Parameters')
```

```python
# Interactive Prediction
st.write("## Make Predictions")
def user_input_features():
    features = {}
    # Load the preprocessed dataset to get feature names
    df_features = pd.read_csv('preprocessed_breast_cancer.csv')
    feature_names = df_features.columns[df_features.columns != 'diagnosis']  # Exclude 'diagnosis'

    for feature in feature_names:
        min_value = float(df_features[feature].min())
        max_value = float(df_features[feature].max())
        mean_value = float(df_features[feature].mean())
        features[feature] = st.sidebar.slider(feature, min_value, max_value, mean_value)

    return pd.DataFrame(features, index=[0])

input_df = user_input_features()

# Load scaler and model
scaler = pickle.load(open('scaler.pkl', 'rb'))
clf = joblib.load('clf_model.pkl')

# Apply model to make predictions
scaled_input = scaler.transform(input_df)
selected_features = fs.transform(scaled_input)
prediction = clf.predict(selected_features)

st.write("### Prediction")
st.write("Benign" if prediction[0] == 0 else "Malignant")
```

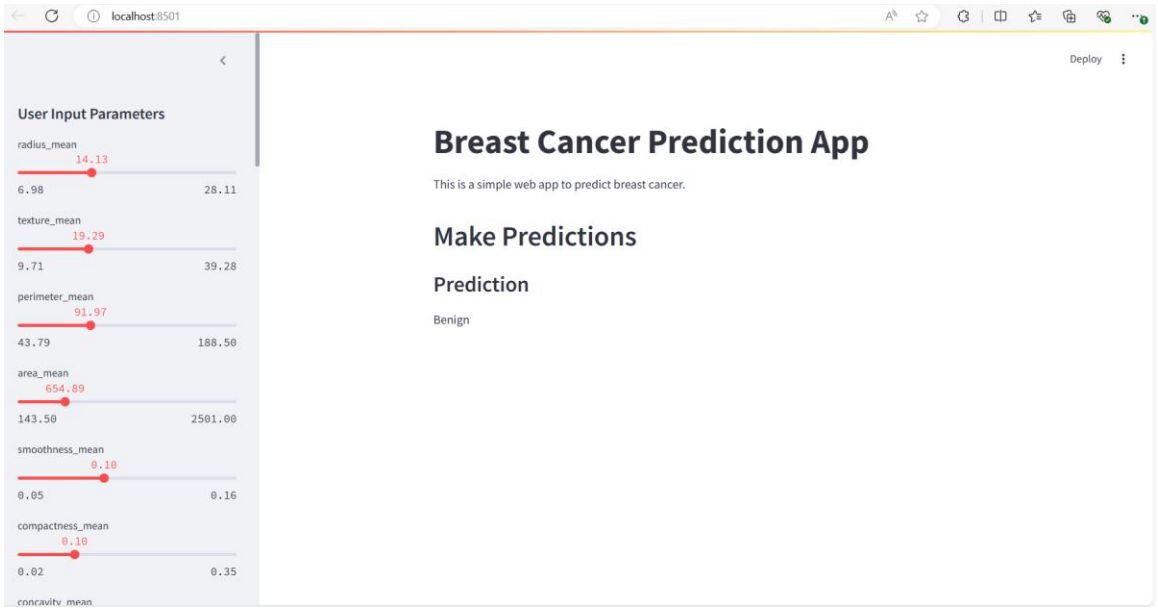
```
59    # Save the file,
60    # streamlit run app.py

PROBLEMS 77   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   JUPYTER

  command:

    streamlit run c:/ANN ASSIGNMENT/app.py [ARGUMENTS]
PS C:\ANN ASSIGNMENT> streamlit run app.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.6.164:8501
```

Deployment

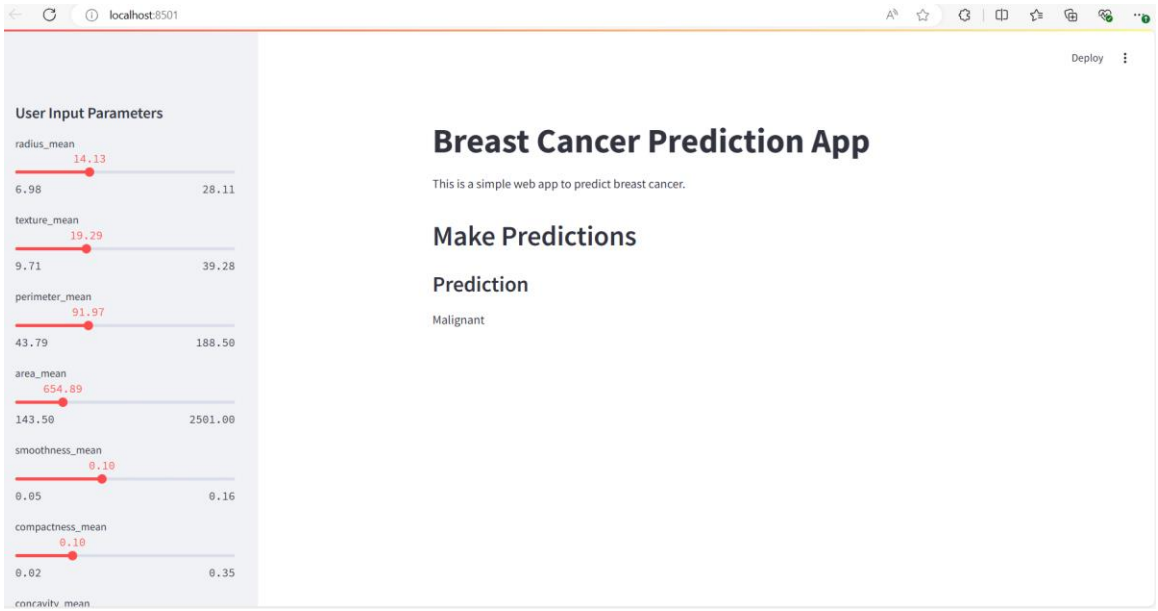
Local URL: http://localhost:8501

Two predictions were made by changing the input parameter.

Prediction - Benign



Prediction - Malignant



GitHub Repository Setup: Commit code and push changes to GitHub

GitHub link: Dupsynusi/Breast-Cancer-Prediction-App (github.com)