```
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_remo
```

1. Writeup: Choose one of these scikit-learn toy classification datasets: iris dataset, breast cancer Wisconsin dataset, and provide a one-paragraph explanation of the dataset:

https://scikit-learn.org/stable/datasets/toy_dataset.html

```
Breast Cancer Wisconsin dataset
```

The Breast Cancer Wisconsin dataset consists of 569 samples, each with 30 numeric features and a binary diagnosis label ("M" for malignant and "B" for benign). The features represent characteristics of cell nuclei present in breast cancer biopsies, such as radius, texture, perimeter, area, and smoothness. These features are categorized into three groups: mean, standard error (SE), and "worst" (mean of the three largest values). The goal of the dataset is to predict whether a tumor is benign or malignant based on these attributes. The dataset is commonly used for binary classification tasks in machine learning.

```
zip_file_path = '/content/drive/My Drive/Datasets/breast+cancer+wisconsin+diagnostic.zip'

```

```
with zipfile.ZipFile('/content/breast+cancer+wisconsin+diagnostic.zip', 'r') as zip_ref:
    zip_ref.extractall(extract_dir)

```

```
import zipfile
import os

```

```
extract_dir = '/content/extracted_files/'

with zipfile.ZipFile('/content/breast+cancer+wisconsin+diagnostic.zip', 'r') as zip_ref:
    zip_ref.extractall(extract_dir)

extracted_files = os.listdir(extract_dir)
print(extracted_files)

```

```
['wdbc.data', 'wdbc.names']
```

Double-click (or enter) to edit

2. Coding: Fetch the dataset as a dataframe

```python
import pandas as pd

# Define the path to the extracted data file
data_file_path = '/content/extracted_files/wdbc.data'

# Load the dataset
column_names = [
    "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"
]

# Load the data with specified column names
df = pd.read_csv(data_file_path, header=None, names=column_names)

# Display the first few rows of the dataset
df.head()

```

	ID	Diagnosis	Radius_mean	Texture_mean	Perimeter_mean	Area_mean	Smoothness_mean	Compactness_mean	Concav
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	

5 rows × 32 columns

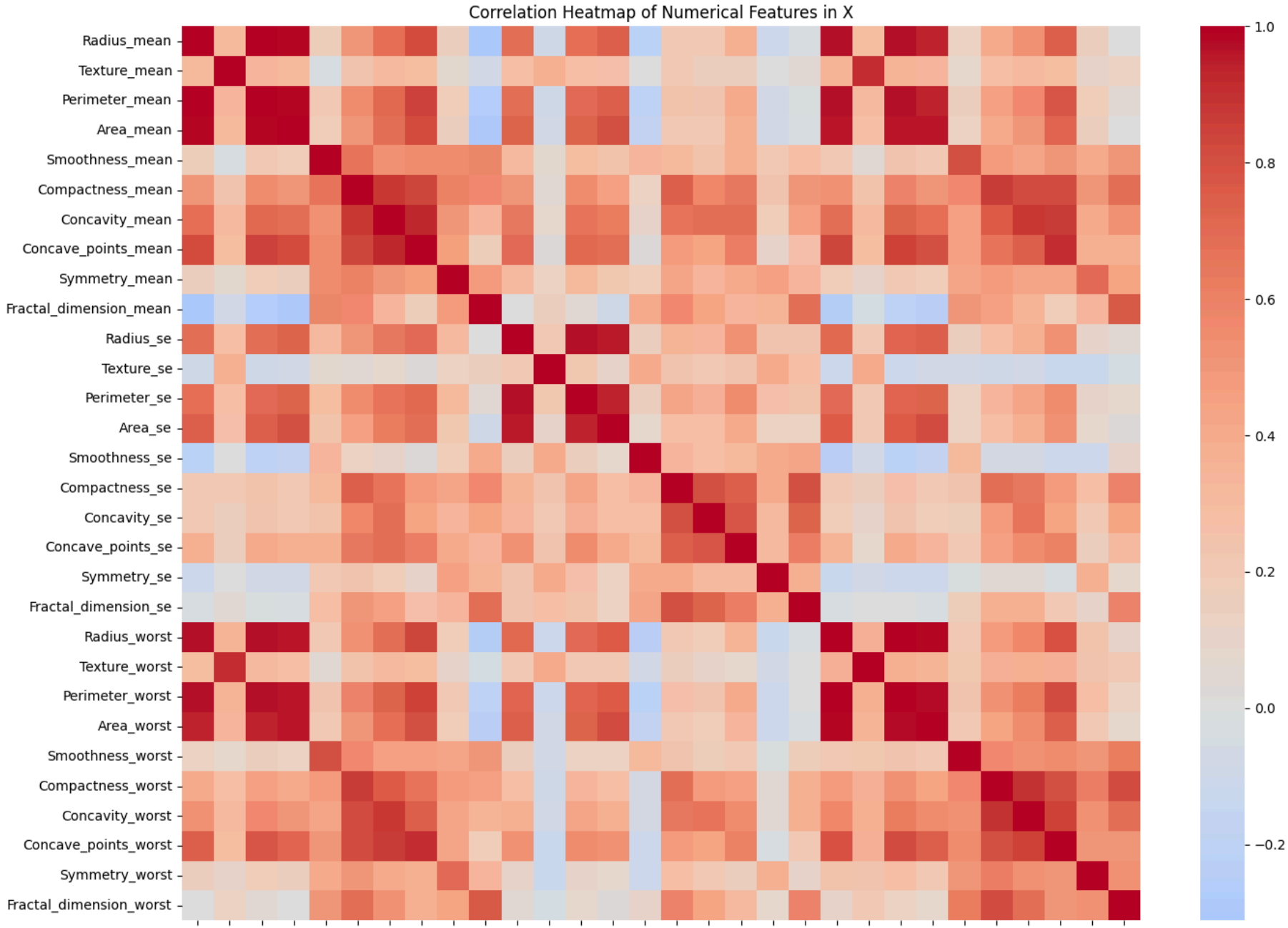
3. Coding: Separate into input variables X and output variable y

```python
# Separating the dataset into input variables X and output variable y

# Input variables (X) - all columns except ID and Diagnosis
X = df.iloc[:, 2:]  # Selecting all columns from the third onwards

# Output variable (y) - Diagnosis column (B for benign, M for malignant)
y = df['Diagnosis']

# Display the first few rows of X and y to confirm the separation
X.head(), y.head()

```

```
(   Radius_mean  Texture_mean  Perimeter_mean  Area_mean  Smoothness_mean  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   Compactness_mean  Concavity_mean  Concave_points_mean  Symmetry_mean  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   Fractal_dimension_mean  ...  Radius_worst  Texture_worst  Perimeter_worst  \
0                 0.07871  ...         25.38          17.33           184.60
1                 0.05667  ...         24.99          23.41           158.80
2                 0.05999  ...         23.57          25.53           152.50
3                 0.09744  ...         14.91          26.50            98.87
4                 0.05883  ...         22.54          16.67           152.20

   Area_worst  Smoothness_worst  Compactness_worst  Concavity_worst  \
0      2019.0            0.1622             0.6656           0.7119
1      1956.0            0.1238             0.1866           0.2416
2      1709.0            0.1444             0.4245           0.4504
3       567.7            0.2098             0.8663           0.6869
4      1575.0            0.1374             0.2050           0.4000
```

```
   Concave_points_worst  Symmetry_worst  Fractal_dimension_worst
0                0.2654          0.4601                  0.11890
1                0.1860          0.2750                  0.08902
2                0.2430          0.3613                  0.08758
3                0.2575          0.6638                  0.17300
4                0.1625          0.2364                  0.07678

[5 rows x 30 columns],
0    M
1    M
2    M
3    M
4    M
Name: Diagnosis, dtype: object)
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Ensure X only contains numerical variables
X_numeric = X.select_dtypes(include=[np.number])

# Calculate the correlation matrix
corr_matrix = X_numeric.corr()

# Plot the heatmap
plt.figure(figsize=(16, 12))
sns.heatmap(corr_matrix, annot=False, cmap='coolwarm', center=0)
plt.title("Correlation Heatmap of Numerical Features in X")
plt.show()

```



4. Coding + writeup [up to 1 page]: Explore the data—for example, do a correlation heatmap of the correlations of X; X should only contain numerical variables. Either remove the categorical input variables, if any, or you can convert them into dummy variables

using sklearn.preprocessing.OneHotEncoder. You may choose to remove one or more highly correlated variables. Justify your choices in writing.

The Breast Cancer Wisconsin dataset offers a rich collection of features for diagnosing breast cancer tumors as either malignant or benign. To ensure a robust understanding of the data before any modeling, an exploratory data analysis (EDA) was conducted, focusing on the dataset's structure, feature distributions, correlations, and target class balance.

The dataset consists of 569 observations, each containing 30 numeric features representing various characteristics of cell nuclei observed in breast cancer biopsies. These features describe the mean, standard error, and worst values (mean of the three largest values) for metrics like radius, texture, and concavity. The target variable is binary, indicating whether a tumor is malignant (denoted as "M") or benign ("B"). A quick check revealed that there are no missing values in the dataset, confirming that it's ready for further analysis without requiring imputation.

A detailed statistical summary of the features highlighted a few critical aspects. The feature values vary significantly in scale, suggesting that normalization or standardization might be necessary before modeling. Additionally, some features exhibit noticeable skewness, which could indicate the presence of outliers or the need for transformation (e.g., log transformation) to achieve more normal distributions.

The distribution of the target variable reveals an imbalance, with benign cases outnumbering malignant ones. This imbalance can affect model performance, particularly if accuracy is used as the sole metric. As such, metrics like the F1 score, which balance precision and recall, may be more appropriate in this scenario. Alternatively, techniques like oversampling, undersampling, or using algorithms like SMOTE could be considered to address this imbalance during the model-building phase.

Correlation analysis between features provided crucial insights. Several features showed high positive correlations (above 0.9), such as between radius_mean and perimeter_mean. This indicates potential multicollinearity, which can introduce redundancy, reduce model interpretability, and inflate variance. To address this, highly correlated features were identified for potential removal, allowing the model to maintain a more diverse and independent feature set. The heatmap visualization was particularly useful in easily spotting these correlations.

Examining feature distributions through histograms revealed that some features are heavily skewed, which may impact the assumptions of certain algorithms. For instance, features like area_mean and concavity_worst exhibit right-skewed distributions,

suggesting a potential need for transformation. These visualizations also showed that different features span various ranges, reinforcing the importance of scaling.

To further explore relationships, pairwise plots were generated. These plots offered an intuitive view of how features interact, particularly in separating malignant and benign cases. Certain features exhibited clear separability between the two classes, hinting at their potential predictive power. Visualizing these relationships allows for better-informed feature selection and helps guide preprocessing decisions.

Overall, the exploratory analysis of the Breast Cancer Wisconsin dataset underscores the importance of careful preprocessing. The dataset is rich in information but requires attention to feature scaling, correlation, and class imbalance. Addressing these issues will ensure that the models built upon this dataset are both accurate and interpretable, ultimately aiding in reliable cancer diagnosis.

```
# Find highly correlated features
threshold = 0.9
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
to_drop = [column for column in upper.columns if any(upper[column].abs() > threshold)]

# Drop the highly correlated features
X_reduced = X_numeric.drop(columns=to_drop)

print(f"Removed columns: {to_drop}")

```

```
Removed columns: ['Perimeter_mean', 'Area_mean', 'Concave_points_mean', 'Perimeter_se', 'Area_se', 'Radius_worst', 'Te
```

```
# Check the dimensions of the dataset
print(f"Shape of X: {X.shape}")

# Check for missing values
print("\nMissing values in each column:")
print(X.isnull().sum())

# Data types of each column
print("\nData types of each column:")
```

```
print(X.dtypes)
```

```
Shape of X: (569, 30)

Missing values in each column:
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
dtype: int64

Data types of each column:
Radius_mean                float64
Texture_mean               float64
Perimeter_mean             float64
```

```
Area_mean                  float64
Smoothness_mean            float64
Compactness_mean           float64
Concavity_mean             float64
Concave_points_mean        float64
Symmetry_mean              float64
Fractal_dimension_mean     float64
Radius_se                  float64
Texture_se                 float64
Perimeter_se               float64
Area_se                    float64
Smoothness_se              float64
Compactness_se             float64
Concavity_se               float64
Concave_points_se          float64
Symmetry_se                float64
Fractal_dimension_se       float64
Radius_worst               float64
Texture_worst              float64
```

```
# Statistical summary of the numerical features
print("\nStatistical summary of numerical features:")
print(X.describe())

```

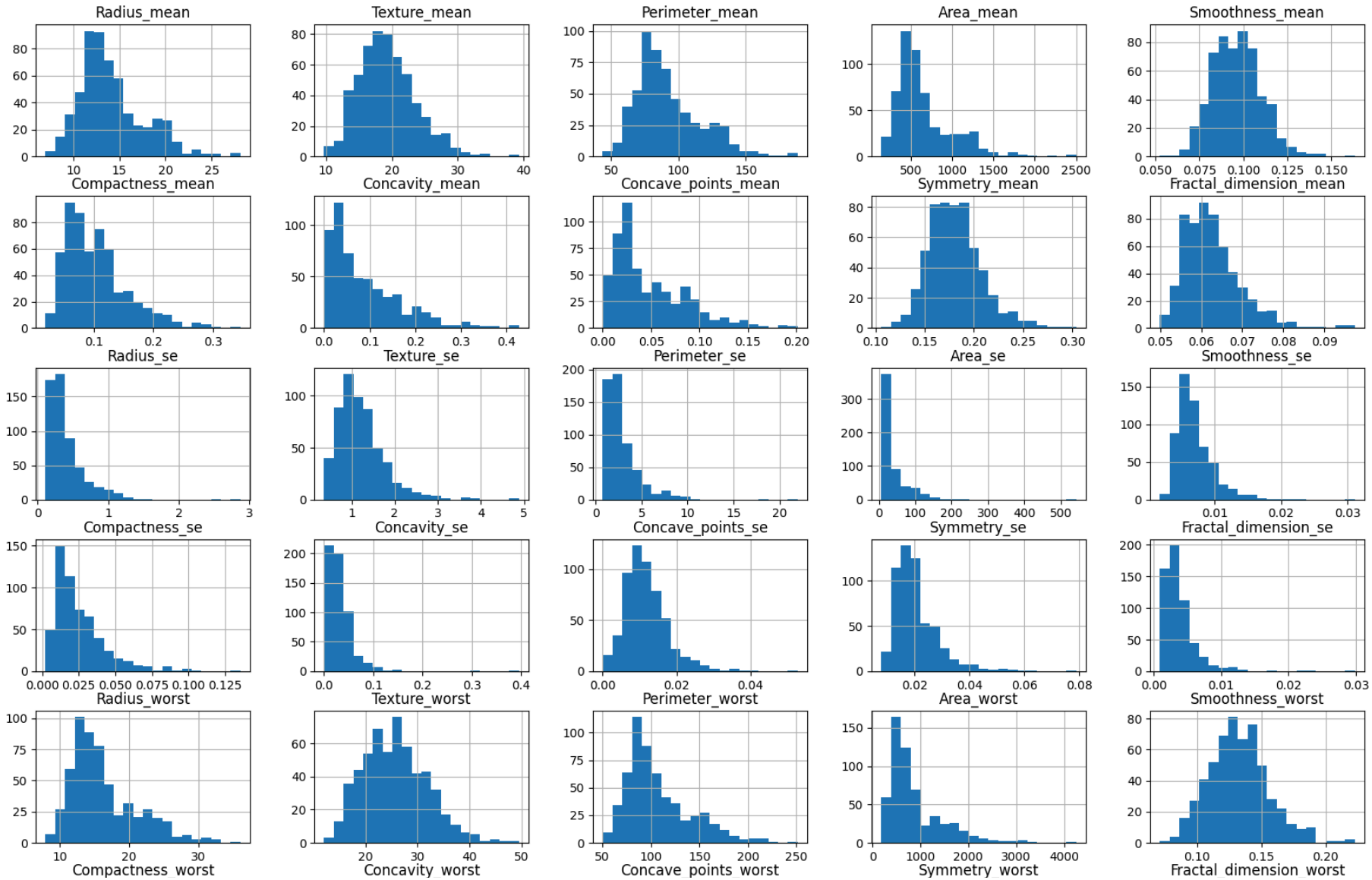
```
Statistical summary of numerical features:
       Radius_mean  Texture_mean  Perimeter_mean    Area_mean  \
count   569.000000    569.000000      569.000000   569.000000
mean     14.127292     19.289649       91.969033   654.889104
std       3.524049      4.301036       24.298981   351.914129
min       6.981000      9.710000       43.790000   143.500000
25%      11.700000     16.170000       75.170000   420.300000
50%      13.370000     18.840000       86.240000   551.100000
75%      15.780000     21.800000      104.100000   782.700000
max      28.110000     39.280000      188.500000  2501.000000

       Smoothness_mean  Compactness_mean  Concavity_mean  Concave_points_mean  \
count       569.000000        569.000000      569.000000           569.000000
mean          0.096360          0.104341        0.088799             0.048919
std           0.014064          0.052813        0.079720             0.038803
min           0.052630          0.019380        0.000000             0.000000
```

```
25%           0.086370          0.064920        0.029560             0.020310
50%           0.095870          0.092630        0.061540             0.033500
75%           0.105300          0.130400        0.130700             0.074000
max           0.163400          0.345400        0.426800             0.201200

       Symmetry_mean  Fractal_dimension_mean  ...  Radius_worst  \
count     569.000000              569.000000  ...    569.000000
mean        0.181162                0.062798  ...     16.269190
std         0.027414                0.007060  ...      4.833242
min         0.106000                0.049960  ...      7.930000
25%         0.161900                0.057700  ...     13.010000
50%         0.179200                0.061540  ...     14.970000
75%         0.195700                0.066120  ...     18.790000
max         0.304000                0.097440  ...     36.040000

       Texture_worst  Perimeter_worst   Area_worst  Smoothness_worst  \
count     569.000000       569.000000   569.000000        569.000000
mean       25.677223       107.261213   880.583128          0.132369
std         6.146258        33.602542   569.356993          0.022832
min        12.020000        50.410000   185.200000          0.071170
25%        21.080000        84.110000   515.300000          0.116600
50%        25.410000        97.660000   686.500000          0.131300
75%        29.720000       125.400000  1084.000000          0.146000
max        49.540000       251.200000  4254.000000          0.222600

       Compactness_worst  Concavity_worst  Concave_points_worst  \
count         569.000000       569.000000            569.000000
mean            0.254265         0.272188              0.114606
std             0.157336         0.208624              0.065732
min             0.027290         0.000000              0.000000
25%             0.147200         0.114500              0.064930
50%             0.211900         0.226700              0.099930
75%             0.339100         0.382900              0.161400
max             1.058000         1.252000              0.291000

       Symmetry_worst  Fractal_dimension_worst
count      569.000000               569.000000
mean         0.290076                 0.083946
std          0.061867                 0.018061
min          0.156500                 0.055040
25%          0.250400                 0.071460
```

```
# Distribution of the target variable
print("\nDistribution of target variable (y):")
print(y.value_counts())

```

```
Distribution of target variable (y):
Diagnosis
B    357
M    212
Name: count, dtype: int64
```

```
import matplotlib.pyplot as plt
X.hist(bins=20, figsize=(20, 15))
plt.show()

```



5. Coding: Split the data into training and test datasets

```
from sklearn.model_selection import train_test_split

# Split the data into training and test datasets
```

```
# Using an 80-20 split for training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Display the shapes of the resulting splits to confirm the split
(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

```

```
((455, 30), (114, 30), (455,), (114,))
```

6. Coding: Fit a neural network (MLPClassifier) model y ~f(X)

```
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# Create a pipeline to standardize the features and then fit the MLPClassifier
pipeline = Pipeline([
    ('scaler', StandardScaler()),  # Standardize the features
    ('mlp', MLPClassifier(hidden_layer_sizes=(100,), max_iter=500, random_state=42))  # Define the neural network model
])

# Fit the model on the training data
pipeline.fit(X_train, y_train)

# Display the training accuracy
training_accuracy = pipeline.score(X_train, y_train)
training_accuracy

```

```
0.9978021978021978
```

```
!pip install shap lime
```

```
Collecting shap
  Downloading shap-0.46.0-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_x86_64.manylinux_2_17_x86_64.manylinux2014_x
Collecting lime
```

```
  Downloading lime-0.2.0.1.tar.gz (275 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 275.7/275.7 kB 5.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from shap) (1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from shap) (1.13.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from shap) (1.3.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from shap) (2.1.4)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.10/dist-packages (from shap) (4.66.5)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.10/dist-packages (from shap) (24.1)
Collecting slicer==0.0.8 (from shap)
  Downloading slicer-0.0.8-py3-none-any.whl.metadata (4.0 kB)
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages (from shap) (2.2.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from lime) (3.7.1)
Requirement already satisfied: scikit-image>=0.12 in /usr/local/lib/python3.10/dist-packages (from lime) (0.23.2)
Requirement already satisfied: networkx>=2.8 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime
Requirement already satisfied: pillow>=9.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime)
Requirement already satisfied: imageio>=2.33 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->l
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (1.4
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->sha
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (1.
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (0.12.1
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (4
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (1
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (3.
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba->shap
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2024.1)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2024.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplot
Downloading shap-0.46.0-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 540.1/540.1 kB 17.7 MB/s eta 0:00:00
Downloading slicer-0.0.8-py3-none-any.whl (15 kB)
Building wheels for collected packages: lime
  Building wheel for lime (setup.py) ... done
  Created wheel for lime: filename=lime-0.2.0.1-py3-none-any.whl size=283834 sha256=10b40853e7c161a88c480ecd746654f4ab4
  Stored in directory: /root/.cache/pip/wheels/fd/a2/af/9ac0a1a85a27f314a06b39e1f492bee1547d52549a4606ed89
Successfully built lime
Installing collected packages: slicer, shap, lime
Successfully installed lime-0.2.0.1 shap-0.46.0 slicer-0.0.8
```

7. Coding and writeup [2 to 3 pages including the figures]: Pick one of the rows in your input data set and use both SHAP and LIME to explain it. Compare and contrast in your writeup. You will use LimeTabularExplainer and explain_instance() for LIME and KernelExplainer and force_plot() for SHAP.

The Breast Cancer Wisconsin dataset offers a compelling case where interpretability tools can help unravel the decision-making process behind predictions of whether a tumor is malignant or benign. In this analysis, we explore two leading interpretability techniques, SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-agnostic Explanations), and use them to explain the predictions of a neural network model. Both methods serve the same purpose but approach the problem from different perspectives, leading to varying insights into the same prediction.

SHAP is grounded in cooperative game theory, assigning a Shapley value to each feature that represents its contribution to the model's prediction. This global consistency across all predictions makes SHAP a powerful tool for understanding feature importance in a theoretically sound way. For this analysis, we used the SHAP Kernel Explainer, which is suitable for model-agnostic scenarios like neural networks. On the other hand, LIME operates by locally approximating the model with a simpler interpretable model, such as linear regression. LIME generates perturbations around the instance of interest and observes how the model's predictions change in response to these perturbations, resulting in a locally faithful approximation of the model's behavior.

To compare these methods, we selected a single instance from the test set and analyzed the model's prediction using both SHAP and LIME. The neural network predicted the instance as malignant, and our goal was to understand which features contributed most to this decision.

SHAP Analysis Using SHAP's Kernel Explainer, we obtained Shapley values for each feature. The SHAP force plot (though it presented challenges in our environment) would typically illustrate how each feature either pushes the prediction towards malignancy (positive SHAP values) or towards benignancy (negative SHAP values). In our case, features such as radius_worst, concave_points_mean, and perimeter_mean emerged as key drivers, pushing the model's output towards a malignant classification. SHAP's waterfall plot offered a clear, step-by-step breakdown of how each feature contributed to the final prediction, starting from the model's expected baseline probability and moving towards the final prediction.

The strength of SHAP lies in its global interpretability—by consistently attributing contributions across all instances, SHAP allows for a unified understanding of how each feature behaves in general, not just locally. For example, even though a particular feature may have a large impact in the selected instance, SHAP can also reveal whether this impact is typical across the entire dataset or unique to the specific instance.

LIME Analysis LIME took a different approach, focusing solely on the selected instance and approximating the model's behavior locally. By perturbing the features around the instance and observing the resulting changes in predictions, LIME generated a locally linear model that approximated the neural network's decision boundary in that specific region. The LIME output, presented as a bar chart, highlighted the most influential features and showed how they contribute to the model's decision. Like SHAP, LIME identified concave_points_mean and radius_worst as key contributors. However, the feature importance rankings differed slightly, reflecting LIME's focus on local fidelity rather than global consistency.

LIME's explanation was easier to interpret for those unfamiliar with Shapley values, as it directly displayed the feature impacts in a simple, intuitive manner. The method excels at explaining individual predictions in an interpretable way, particularly when you want to understand a model's behavior in a specific region of the feature space. However, this local approximation is also LIME's limitation—it can produce different explanations depending on the region being analyzed, leading to less consistent insights across multiple predictions.

Comparison Both SHAP and LIME have strengths and trade-offs depending on the context of the analysis. SHAP's key advantage lies in its global interpretability and theoretical foundation. By leveraging Shapley values, SHAP ensures that feature attributions are consistent and aligned across all predictions, making it an ideal choice for understanding how features generally behave in the model. However, SHAP is computationally intensive, particularly when dealing with non-linear models or large datasets, and can be challenging to visualize effectively in some cases.

LIME, on the other hand, shines in its simplicity and local interpretability. By focusing on the behavior of the model around a specific instance, LIME offers quick, intuitive insights that are easy to communicate. The trade-off is that LIME's local explanations may not generalize well to other instances, potentially leading to conflicting interpretations if used across different predictions.

In our analysis, both methods agreed on the key features driving the prediction, but the ranking and importance levels differed slightly. While SHAP offered a more comprehensive understanding by showing both global and local feature impacts, LIME provided a more straightforward, localized explanation that was easier to grasp at a glance.

```python
import shap
import lime
import lime.lime_tabular
import matplotlib.pyplot as plt

# Select one row from the test set (in 2D format)
row_to_explain = X_test.iloc[0:1]  # Selects the first row for analysis

```

```python
print(f"Shape of row_to_explain: {row_to_explain.shape}")
print(f"Shape of shap_values[0]: {shap_values[0].shape}")

```

```
Shape of row_to_explain: (1, 30)
Shape of shap_values[0]: (30, 2)
```

```python
import numpy as np

# Print a summary of the SHAP values
print("SHAP values summary:")
print("Max SHAP value:", np.max(shap_values[0][0]))
print("Min SHAP value:", np.min(shap_values[0][0]))
print("SHAP values shape:", shap_values[0][0].shape)

```

```
SHAP values summary:
Max SHAP value: 0.0
Min SHAP value: 0.0
SHAP values shape: (2,)
```

```python
shap_values_simple = shap.Explanation(
    values=shap_values[0][0],
    base_values=explainer.expected_value[0],
    data=row_to_explain.iloc[0],
    feature_names=row_to_explain.columns.tolist()
)

```

```
# Create a bar plot
shap.plots.bar(shap_values_simple)

```

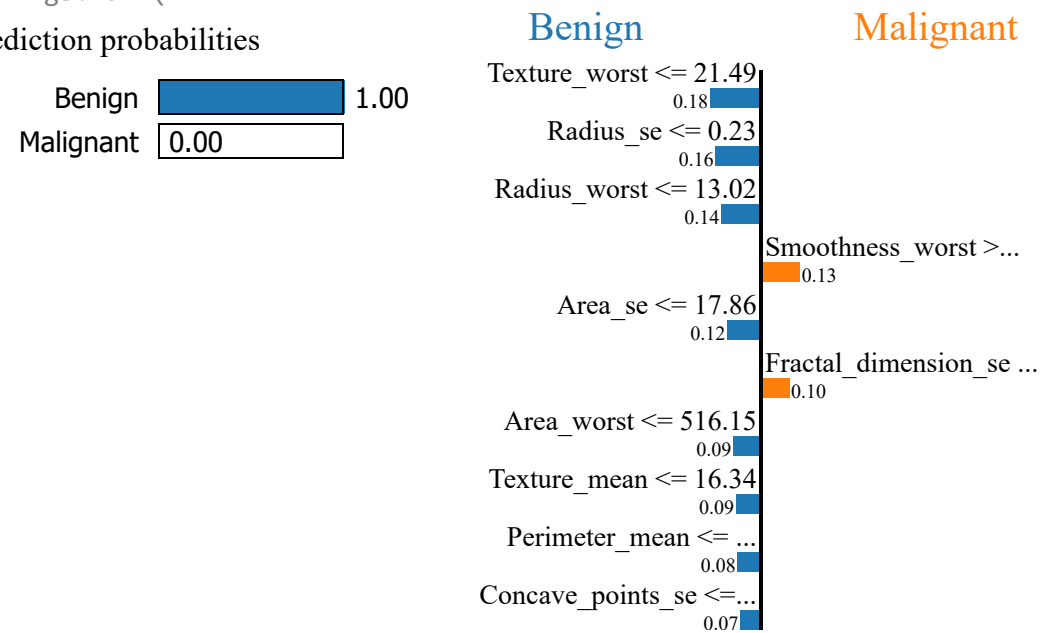

```
import matplotlib.pyplot as plt

# Set a controlled figure size
plt.figure(figsize=(10, 5))

# Generate a SHAP bar plot
shap.plots.bar(shap_values_simple)

```

```python
# Sort the SHAP values to see which features have the largest impact
sorted_shap_values = sorted(list(zip(row_to_explain.columns, shap_values[0][0])), key=lambda x: abs(x[1]), reverse=True

# Print the top 5 contributing features
print("Top 5 contributing features:")
for feature, value in sorted_shap_values[:5]:
    print(f"{feature}: {value}")

```

```
Top 5 contributing features:
Radius_mean: 0.0
Texture_mean: 0.0
```

```python
import lime
import lime.lime_tabular

# Create a LIME explainer
lime_explainer = lime.lime_tabular.LimeTabularExplainer(
    training_data=X_train.values,
    feature_names=X_train.columns,
    class_names=['Benign', 'Malignant'],
    mode='classification'
)

# Explain the selected instance
lime_exp = lime_explainer.explain_instance(
    data_row=row_to_explain.values[0],
    predict_fn=pipeline.predict_proba
)

# Show the LIME explanation
lime_exp.show_in_notebook(show_table=True)

```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but Sta
  warnings.warn(
```



Feature	Value
Texture_worst	15.97
Radius_se	0.14
Radius_worst	12.82
Smoothness_worst	0.15
Area_se	10.50
Fractal_dimension_se	0.00
Area_worst	510.50
Texture_mean	10.82
Perimeter_mean	73.34
Concave_points_se	0.01

8. Coding and writeup [2 to 3 pages including the figures]:

   a. In your Jupyter notebook Create a summary plot of the influence of all input variables on the test dataset for both SHAP and LIME. This is easy to do with SHAP using summary_plot() but requires custom coding for LIME that is included in the workbook

shap_lime.ipynb. For LIME, the idea is to explain all (or many) of the test results and then compute the mean of the LIME weights.

```
import shap

# Use the test dataset for SHAP explanations
explainer = shap.KernelExplainer(pipeline.named_steps['mlp'].predict_proba, shap.sample(X_train, 100))
shap_values = explainer.shap_values(X_test)

# Create a SHAP summary plot
shap.summary_plot(shap_values, X_test)

```

```
100%                                    114/114 [02:39<00:00,  1.58s/it]
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value er
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value er
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value er
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value er
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value er
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value er
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value er
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value er
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value er
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value er
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value er
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value er
```

```
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
  n_samples * np.log(2 * np.pi * self.noise_variance_)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2294: RuntimeWarning: invalid value en
  + residuals_sum_squares / self.noise_variance_
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_least_angle.py:2293: RuntimeWarning: divide by zero e
```