modelling

May 19, 2024

1 Cancer Prediction Using Machine-Learning Models

==

Objective :

- This analysis aims to observe which features are most helpful in predicting malignant or benign cancer and to see general trends that may aid us in model selection and hyper parameter selection. The goal is to **classify whether the breast cancer is benign (B) or malignant (M)**. To achieve this i have used **machine learning classification methods** to fit a function that can predict the discrete class of new input.

==

- **Structure of the Test :**

1. Importing Dependencies (library & packages)
2. Data Preparation –> (Load And Check Data)
3. Data Exploration & Analysis
4. Data Partitioning & Feature scaling
5. Machine Learning Model Selection & Performance Evaluation
   - Logistic Regression (LR)
   - GradientBoostingClassifier (GB)
   - RandomForestClassifier (RF)

- Perform Comparative Analysis of each & every **3 classification algorithms** & then conclude to the best-model.

==

1. Importing Dependencies [4]

[3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
```

```python
import warnings
warnings.filterwarnings('ignore') # optional to handle warnings
```

==
2. Data preparation (Load & Check Data) [12]

2a. Import cancer_data.csv dataset and view first 5 rows? 2 marks

[4]:
```python
df = pd.read_csv("../data/cancer_data.csv")
df.head(5)
```

[4]:
```
         id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0    842302         M        17.99         10.38          122.80     1001.0
1    842517         M        20.57         17.77          132.90     1326.0
2  84300903         M        19.69         21.25          130.00     1203.0
3  84348301         M        11.42         20.38           77.58      386.1
4  84358402         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   …  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0  …          17.33           184.60      2019.0            0.1622
1  …          23.41           158.80      1956.0            0.1238
2  …          25.53           152.50      1709.0            0.1444
3  …          26.50            98.87       567.7            0.2098
4  …          16.67           152.20      1575.0            0.1374

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0             0.6656           0.7119                0.2654          0.4601
1             0.1866           0.2416                0.1860          0.2750
2             0.4245           0.4504                0.2430          0.3613
3             0.8663           0.6869                0.2575          0.6638
4             0.2050           0.4000                0.1625          0.2364

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
1                  0.08902          NaN
2                  0.08758          NaN
3                  0.17300          NaN
4                  0.07678          NaN

[5 rows x 33 columns]
```

2b. Determine the dimension of breast cancer dataset and comment? 2 marks

[5]:
```
print(f"The dimension of this dataset is: {df.shape}")
```

```
The dimension of this dataset is: (569, 33)
```

We can see that there are 569 rows and 33 columns in this dataset

<p>2c. Check for missing values and comment? 2 marks</p>

[6]:
```
print(df.isnull().sum())
```

```
id                           0
diagnosis                    0
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
Unnamed: 32                569
dtype: int64
```

There seems to be no missing values in this dataset besides in the 'Unamed 32' column which has 569 missing values, therefore no imputation for missing values in the other columns is required.

2d. Drop irrelevant columns, if any, from the breast cancer dataset which can not be used to predict breast cancer? 3 marks

[7]:
```
df2 = df.drop(['Unnamed: 32'], axis=1)
```

I have decided to drop the 'Unamed 32' column because out of the other columns, it seems to have multiplle NaN values and doesn't seem like it well contribute to preidcting breast cancer.

2e. Check for duplicate rows and comment. 3 marks

[8]:
```
df2.duplicated().sum()
```

[8]:
```
0
```

There are no duplicates in this dataset, this tell us that this is either a very well kept dataset or it has already been preprocessed

==
3. Data Exploration & Analysis [34]

3a. Rename the 'diagnosis' column to 'label'? 11 marks

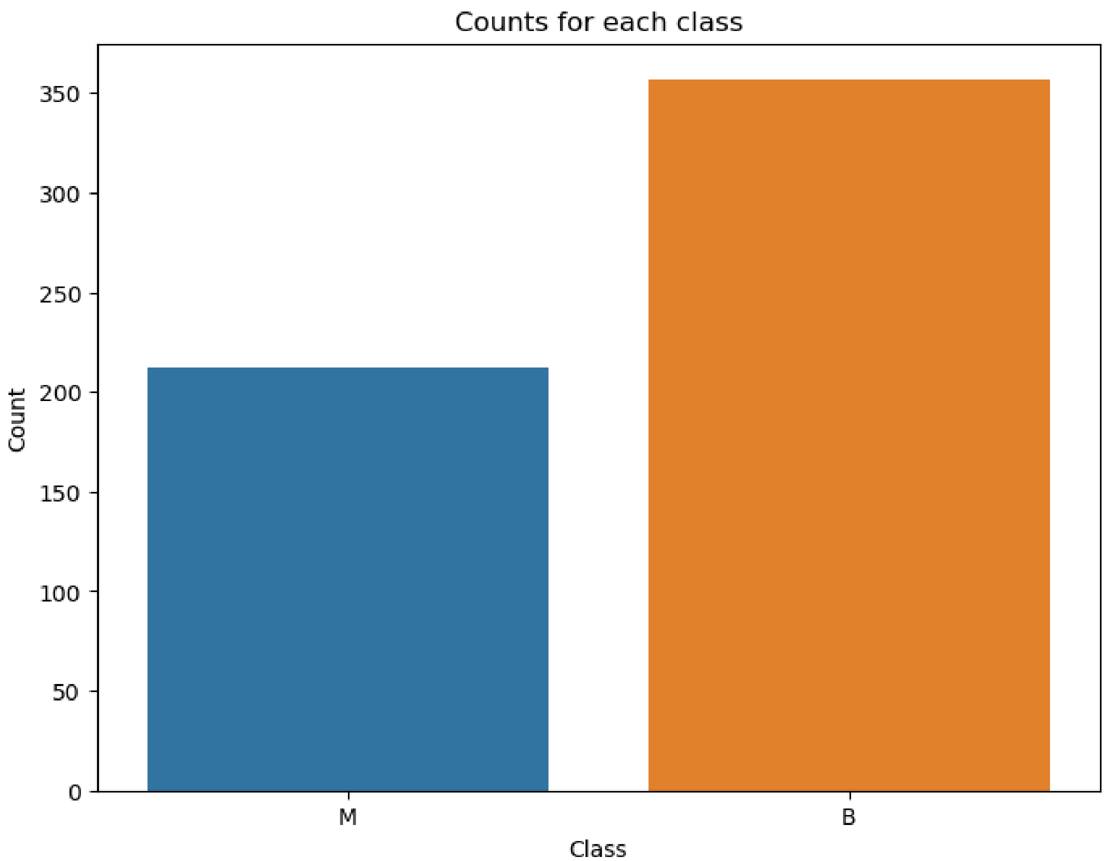
`diagnosis` is the column which we are going to predict , which says if the cancer is `M = malignant` or `B = benign`.

- i. Rename `diagnosis` to `label`. [2]
- ii. Plot a `countplot` of the label to show counts for each class, include annotations (`chart title` and `x and y-axis titles`).[3]
- iii. Convert `string` expressions to `int` because it will be necessary when training your model. `Malignant = 1` ,`Benign = 0`. [3]
- iv. Confirm number of `malignant` and `benign` cases and comment. [3]

[9]:
```
# renaming the title of properties as per need of prediction [2]
df2.rename(columns={'diagnosis': 'label'}, inplace=True)
```

[10]:
```
# countplot

plt.figure(figsize=(8, 6))
sns.countplot(x='label', data=df2)
plt.title('Counts for each class')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```



Categorical data contain variables with text labels rather than numeric. The number of possible values is often limited to a fixed set. You need to change these into some numeric values to represent the text.

[11]:
```
# label encoding
label_encoder = LabelEncoder()
df2['label'] = label_encoder.fit_transform(df2['label'])
```

[12]:
```
# confirming counts of respective classes
counts = df2['label'].value_counts()
print("Value Counts:\n", counts)

malignant_cases = counts.get(1, 0)
benign_cases = counts.get(0, 0)

print(f'Malignant cases: {malignant_cases}')
print(f'Benign cases: {benign_cases}')
```

```
Value Counts:
 label
```

```
0    357
1    212
Name: count, dtype: int64
Malignant cases: 212
Benign cases: 357
```

As we can see in the value counts mini table, feature 0 has 357 counts and feature 1 has 212 counts which matches the counts we have. (Benign cases is 0 and Maligant cases is 1)

- **Variable/Attribute Description**
  Label–> **(M= malignant , B = Benign)**

  Ten real-valued features are computed for each cell nucleus:

1. radius (mean of distances from center to points on the perimeter)
2. texture (standard deviation of gray-scale values)
3. perimeter
4. area
5. smoothness (local variation in radius lengths)
6. compactness (perimeter^2 / area - 1.0)
7. concavity (severity of concave portions of the contour)
8. concave points (number of concave portions of the contour)
9. symmetry
10. fractal dimension ("coastline approximation" - 1) ____

- The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

3b. Compute a 5-number summary of all features against the label. (i.e. min, 25%, 50%, 75%, max only). 2 marks

[13]:
```
df2_summary = df2['label'].describe()
print(df2_summary)
```

```
count    569.000000
mean       0.372583
std        0.483918
min        0.000000
25%        0.000000
50%        0.000000
75%        1.000000
max        1.000000
Name: label, dtype: float64
```

3c. Compute correlation of the entire dataset and observe features with 'corr value' greater than '60%'. 4 marks

[14]:
```
# correlation matrix
correlation_matrix = df2.corr()
high_correlation = correlation_matrix[correlation_matrix.abs() > 0.6]
```

```python
print("Correlation matrix with values greater than 60%:\n", high_correlation)
```

```
Correlation matrix with values greater than 60%:
                          id     label  radius_mean  texture_mean  \
id                       1.0       NaN          NaN           NaN
label                    NaN  1.000000     0.730029           NaN
radius_mean              NaN  0.730029     1.000000           NaN
texture_mean             NaN       NaN          NaN      1.000000
perimeter_mean           NaN  0.742636     0.997855           NaN
area_mean                NaN  0.708984     0.987357           NaN
smoothness_mean          NaN       NaN          NaN           NaN
compactness_mean         NaN       NaN          NaN           NaN
concavity_mean           NaN  0.696360     0.676764           NaN
concave points_mean      NaN  0.776614     0.822529           NaN
symmetry_mean            NaN       NaN          NaN           NaN
fractal_dimension_mean   NaN       NaN          NaN           NaN
radius_se                NaN       NaN     0.679090           NaN
texture_se               NaN       NaN          NaN           NaN
perimeter_se             NaN       NaN     0.674172           NaN
area_se                  NaN       NaN     0.735864           NaN
smoothness_se            NaN       NaN          NaN           NaN
compactness_se           NaN       NaN          NaN           NaN
concavity_se             NaN       NaN          NaN           NaN
concave points_se        NaN       NaN          NaN           NaN
symmetry_se              NaN       NaN          NaN           NaN
fractal_dimension_se     NaN       NaN          NaN           NaN
radius_worst             NaN  0.776454     0.969539           NaN
texture_worst            NaN       NaN          NaN      0.912045
perimeter_worst          NaN  0.782914     0.965137           NaN
area_worst               NaN  0.733825     0.941082           NaN
smoothness_worst         NaN       NaN          NaN           NaN
compactness_worst        NaN       NaN          NaN           NaN
concavity_worst          NaN  0.659610          NaN           NaN
concave points_worst     NaN  0.793566     0.744214           NaN
symmetry_worst           NaN       NaN          NaN           NaN
fractal_dimension_worst  NaN       NaN          NaN           NaN

                         perimeter_mean  area_mean  smoothness_mean  \
id                                  NaN        NaN              NaN
label                          0.742636   0.708984              NaN
radius_mean                    0.997855   0.987357              NaN
texture_mean                        NaN        NaN              NaN
perimeter_mean                 1.000000   0.986507              NaN
area_mean                      0.986507   1.000000              NaN
smoothness_mean                     NaN        NaN         1.000000
compactness_mean                    NaN        NaN         0.659123
concavity_mean                 0.716136   0.685983              NaN
```

concave points_mean	0.850977	0.823269	NaN
symmetry_mean	NaN	NaN	NaN
fractal_dimension_mean	NaN	NaN	NaN
radius_se	0.691765	0.732562	NaN
texture_se	NaN	NaN	NaN
perimeter_se	0.693135	0.726628	NaN
area_se	0.744983	0.800086	NaN
smoothness_se	NaN	NaN	NaN
compactness_se	NaN	NaN	NaN
concavity_se	NaN	NaN	NaN
concave points_se	NaN	NaN	NaN
symmetry_se	NaN	NaN	NaN
fractal_dimension_se	NaN	NaN	NaN
radius_worst	0.969476	0.962746	NaN
texture_worst	NaN	NaN	NaN
perimeter_worst	0.970387	0.959120	NaN
area_worst	0.941550	0.959213	NaN
smoothness_worst	NaN	NaN	0.805324
compactness_worst	NaN	NaN	NaN
concavity_worst	NaN	NaN	NaN
concave points_worst	0.771241	0.722017	NaN
symmetry_worst	NaN	NaN	NaN
fractal_dimension_worst	NaN	NaN	NaN

	compactness_mean	concavity_mean	\
id	NaN	NaN	
label	NaN	0.696360	
radius_mean	NaN	0.676764	
texture_mean	NaN	NaN	
perimeter_mean	NaN	0.716136	
area_mean	NaN	0.685983	
smoothness_mean	0.659123	NaN	
compactness_mean	1.000000	0.883121	
concavity_mean	0.883121	1.000000	
concave points_mean	0.831135	0.921391	
symmetry_mean	0.602641	NaN	
fractal_dimension_mean	NaN	NaN	
radius_se	NaN	0.631925	
texture_se	NaN	NaN	
perimeter_se	NaN	0.660391	
area_se	NaN	0.617427	
smoothness_se	NaN	NaN	
compactness_se	0.738722	0.670279	
concavity_se	NaN	0.691270	
concave points_se	0.642262	0.683260	
symmetry_se	NaN	NaN	
fractal_dimension_se	NaN	NaN	
radius_worst	NaN	0.688236	

texture_worst	NaN	NaN
perimeter_worst	NaN	0.729565
area_worst	NaN	0.675987
smoothness_worst	NaN	NaN
compactness_worst	0.865809	0.754968
concavity_worst	0.816275	0.884103
concave points_worst	0.815573	0.861323
symmetry_worst	NaN	NaN
fractal_dimension_worst	0.687382	NaN

	concave points_mean	…	radius_worst \
id	NaN	…	NaN
label	0.776614	…	0.776454
radius_mean	0.822529	…	0.969539
texture_mean	NaN	…	NaN
perimeter_mean	0.850977	…	0.969476
area_mean	0.823269	…	0.962746
smoothness_mean	NaN	…	NaN
compactness_mean	0.831135	…	NaN
concavity_mean	0.921391	…	0.688236
concave points_mean	1.000000	…	0.830318
symmetry_mean	NaN	…	NaN
fractal_dimension_mean	NaN	…	NaN
radius_se	0.698050	…	0.715065
texture_se	NaN	…	NaN
perimeter_se	0.710650	…	0.697201
area_se	0.690299	…	0.757373
smoothness_se	NaN	…	NaN
compactness_se	NaN	…	NaN
concavity_se	NaN	…	NaN
concave points_se	0.615634	…	NaN
symmetry_se	NaN	…	NaN
fractal_dimension_se	NaN	…	NaN
radius_worst	0.830318	…	1.000000
texture_worst	NaN	…	NaN
perimeter_worst	0.855923	…	0.993708
area_worst	0.809630	…	0.984015
smoothness_worst	NaN	…	NaN
compactness_worst	0.667454	…	NaN
concavity_worst	0.752399	…	NaN
concave points_worst	0.910155	…	0.787424
symmetry_worst	NaN	…	NaN
fractal_dimension_worst	NaN	…	NaN

	texture_worst	perimeter_worst	area_worst \
id	NaN	NaN	NaN
label	NaN	0.782914	0.733825
radius_mean	NaN	0.965137	0.941082

texture_mean	0.912045	NaN	NaN
perimeter_mean	NaN	0.970387	0.941550
area_mean	NaN	0.959120	0.959213
smoothness_mean	NaN	NaN	NaN
compactness_mean	NaN	NaN	NaN
concavity_mean	NaN	0.729565	0.675987
concave points_mean	NaN	0.855923	0.809630
symmetry_mean	NaN	NaN	NaN
fractal_dimension_mean	NaN	NaN	NaN
radius_se	NaN	0.719684	0.751548
texture_se	NaN	NaN	NaN
perimeter_se	NaN	0.721031	0.730713
area_se	NaN	0.761213	0.811408
smoothness_se	NaN	NaN	NaN
compactness_se	NaN	NaN	NaN
concavity_se	NaN	NaN	NaN
concave points_se	NaN	NaN	NaN
symmetry_se	NaN	NaN	NaN
fractal_dimension_se	NaN	NaN	NaN
radius_worst	NaN	0.993708	0.984015
texture_worst	1.000000	NaN	NaN
perimeter_worst	NaN	1.000000	0.977578
area_worst	NaN	0.977578	1.000000
smoothness_worst	NaN	NaN	NaN
compactness_worst	NaN	NaN	NaN
concavity_worst	NaN	0.618344	NaN
concave points_worst	NaN	0.816322	0.747419
symmetry_worst	NaN	NaN	NaN
fractal_dimension_worst	NaN	NaN	NaN

	smoothness_worst	compactness_worst	concavity_worst \
id	NaN	NaN	NaN
label	NaN	NaN	0.659610
radius_mean	NaN	NaN	NaN
texture_mean	NaN	NaN	NaN
perimeter_mean	NaN	NaN	NaN
area_mean	NaN	NaN	NaN
smoothness_mean	0.805324	NaN	NaN
compactness_mean	NaN	0.865809	0.816275
concavity_mean	NaN	0.754968	0.884103
concave points_mean	NaN	0.667454	0.752399
symmetry_mean	NaN	NaN	NaN
fractal_dimension_mean	NaN	NaN	NaN
radius_se	NaN	NaN	NaN
texture_se	NaN	NaN	NaN
perimeter_se	NaN	NaN	NaN
area_se	NaN	NaN	NaN
smoothness_se	NaN	NaN	NaN

```
compactness_se                    NaN           0.678780          0.639147
concavity_se                      NaN                NaN          0.662564
concave points_se                 NaN                NaN               NaN
symmetry_se                       NaN                NaN               NaN
fractal_dimension_se              NaN                NaN               NaN
radius_worst                      NaN                NaN               NaN
texture_worst                     NaN                NaN               NaN
perimeter_worst                   NaN                NaN          0.618344
area_worst                        NaN                NaN               NaN
smoothness_worst             1.000000                NaN               NaN
compactness_worst                 NaN           1.000000          0.892261
concavity_worst                   NaN           0.892261          1.000000
concave points_worst              NaN           0.801080          0.855434
symmetry_worst                    NaN           0.614441               NaN
fractal_dimension_worst      0.617624           0.810455          0.686511

                      concave points_worst  symmetry_worst  \
id                                     NaN             NaN
label                             0.793566             NaN
radius_mean                       0.744214             NaN
texture_mean                           NaN             NaN
perimeter_mean                    0.771241             NaN
area_mean                         0.722017             NaN
smoothness_mean                        NaN             NaN
compactness_mean                  0.815573             NaN
concavity_mean                    0.861323             NaN
concave points_mean               0.910155             NaN
symmetry_mean                          NaN        0.699826
fractal_dimension_mean                 NaN             NaN
radius_se                              NaN             NaN
texture_se                             NaN             NaN
perimeter_se                           NaN             NaN
area_se                                NaN             NaN
smoothness_se                          NaN             NaN
compactness_se                         NaN             NaN
concavity_se                           NaN             NaN
concave points_se                 0.602450             NaN
symmetry_se                            NaN             NaN
fractal_dimension_se                   NaN             NaN
radius_worst                      0.787424             NaN
texture_worst                          NaN             NaN
perimeter_worst                   0.816322             NaN
area_worst                        0.747419             NaN
smoothness_worst                       NaN             NaN
compactness_worst                 0.801080        0.614441
concavity_worst                   0.855434             NaN
concave points_worst              1.000000             NaN
symmetry_worst                         NaN        1.000000
```

```
fractal_dimension_worst                   NaN             NaN

                         fractal_dimension_worst
id                                           NaN
label                                        NaN
radius_mean                                  NaN
texture_mean                                 NaN
perimeter_mean                               NaN
area_mean                                    NaN
smoothness_mean                              NaN
compactness_mean                        0.687382
concavity_mean                               NaN
concave points_mean                          NaN
symmetry_mean                                NaN
fractal_dimension_mean                  0.767297
radius_se                                    NaN
texture_se                                   NaN
perimeter_se                                 NaN
area_se                                      NaN
smoothness_se                                NaN
compactness_se                               NaN
concavity_se                                 NaN
concave points_se                            NaN
symmetry_se                                  NaN
fractal_dimension_se                         NaN
radius_worst                                 NaN
texture_worst                                NaN
perimeter_worst                              NaN
area_worst                                   NaN
smoothness_worst                        0.617624
compactness_worst                       0.810455
concavity_worst                         0.686511
concave points_worst                         NaN
symmetry_worst                               NaN
fractal_dimension_worst                 1.000000

[32 rows x 32 columns]
```

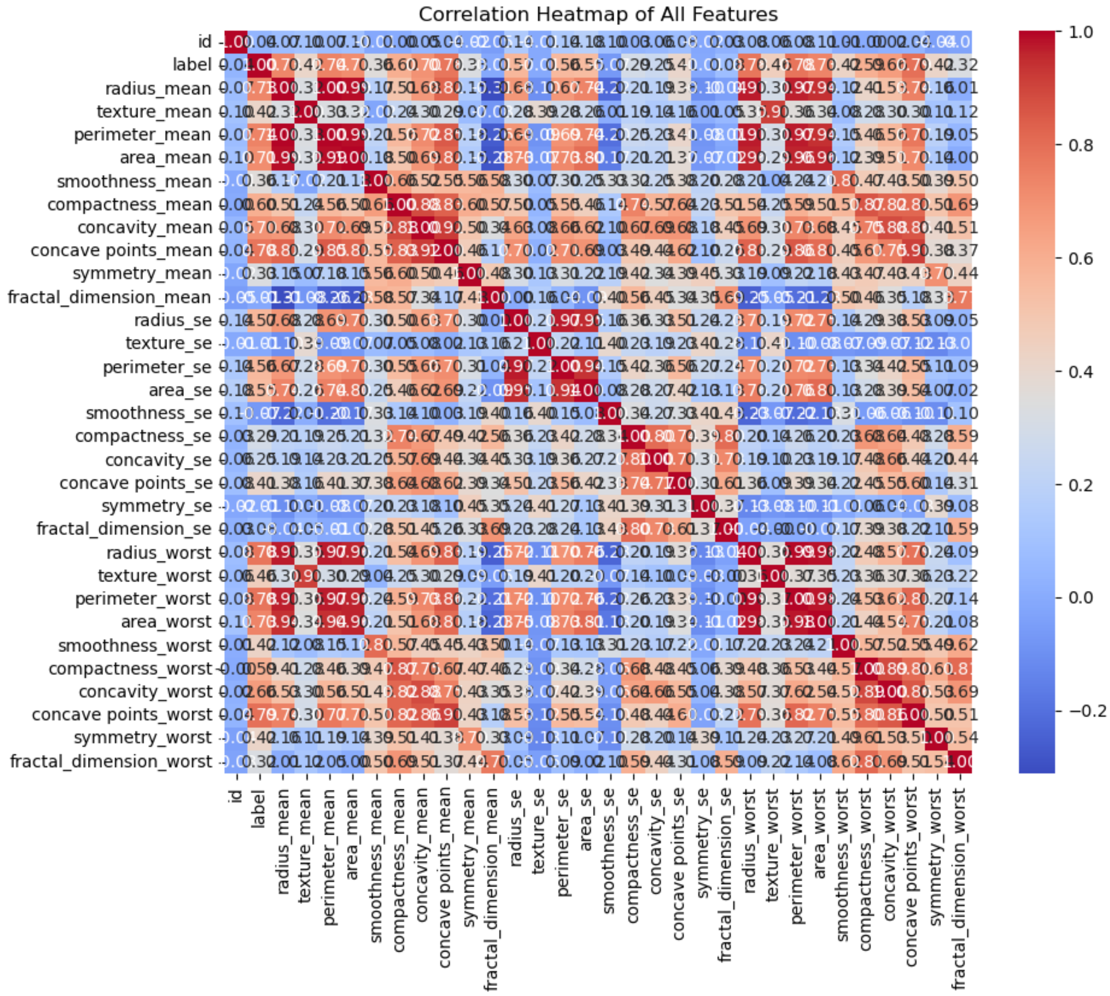
Visualization of data is an imperative aspect to understand data and also to explain the data to another person. Python has several interesting visualization libraries that can help an individual to achieve this.

3d. Visualize the correlation between features using heatmap. 11 marks

i. Heatmap of all features, include annotations, title and correlation values must be to 2 decimal places. [5]

ii. Heatmap of all features with 60% corr value and above, include annotations, title and correlation values must be to 2 decimal places. [6]

[15]:
```python
plt.figure(figsize=(10, 8))
sns.heatmap(df2.corr(), annot=True, fmt='.2f', cmap='coolwarm', cbar=True)
plt.title('Correlation Heatmap of All Features')
plt.show()
```

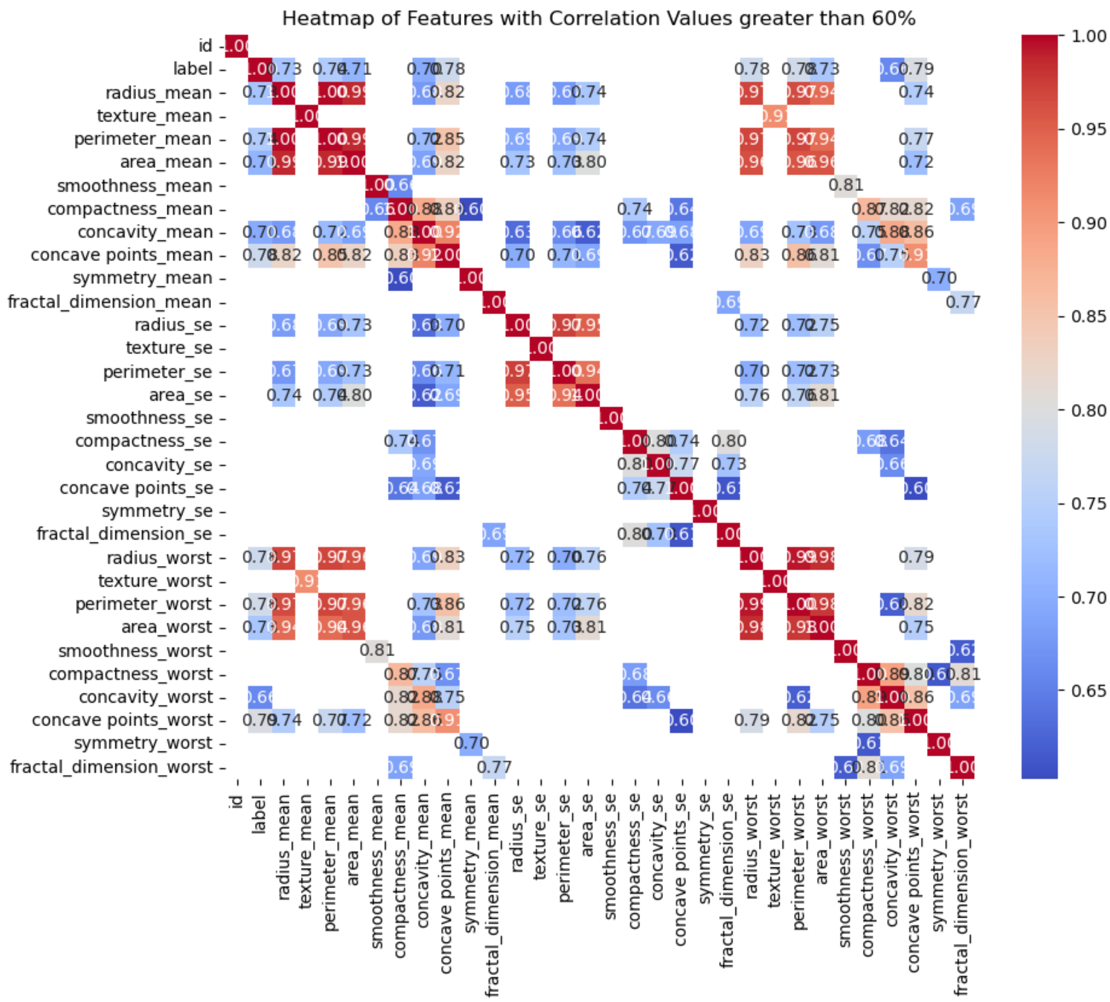


- First, set a limit value. Here set it to `0.6`. Display features with relationship against the target greater than `|0.6|`.

[16]:
```python
target_corr = correlation_matrix['label'][correlation_matrix['label'].abs() > 0.
 ↪6]
filtered_corr_matrix = high_correlation.dropna(how='all').dropna(axis=1,␣
 ↪how='all')

plt.figure(figsize=(10, 8))
sns.heatmap(filtered_corr_matrix, annot=True, fmt='.2f', cmap='coolwarm',␣
 ↪cbar=True)
```

```
plt.title('Heatmap of Features with Correlation Values greater than 60%')
plt.show()
```



3e. Use pairplot to visualize features with 60% and higher correlation value against the label. 3 marks

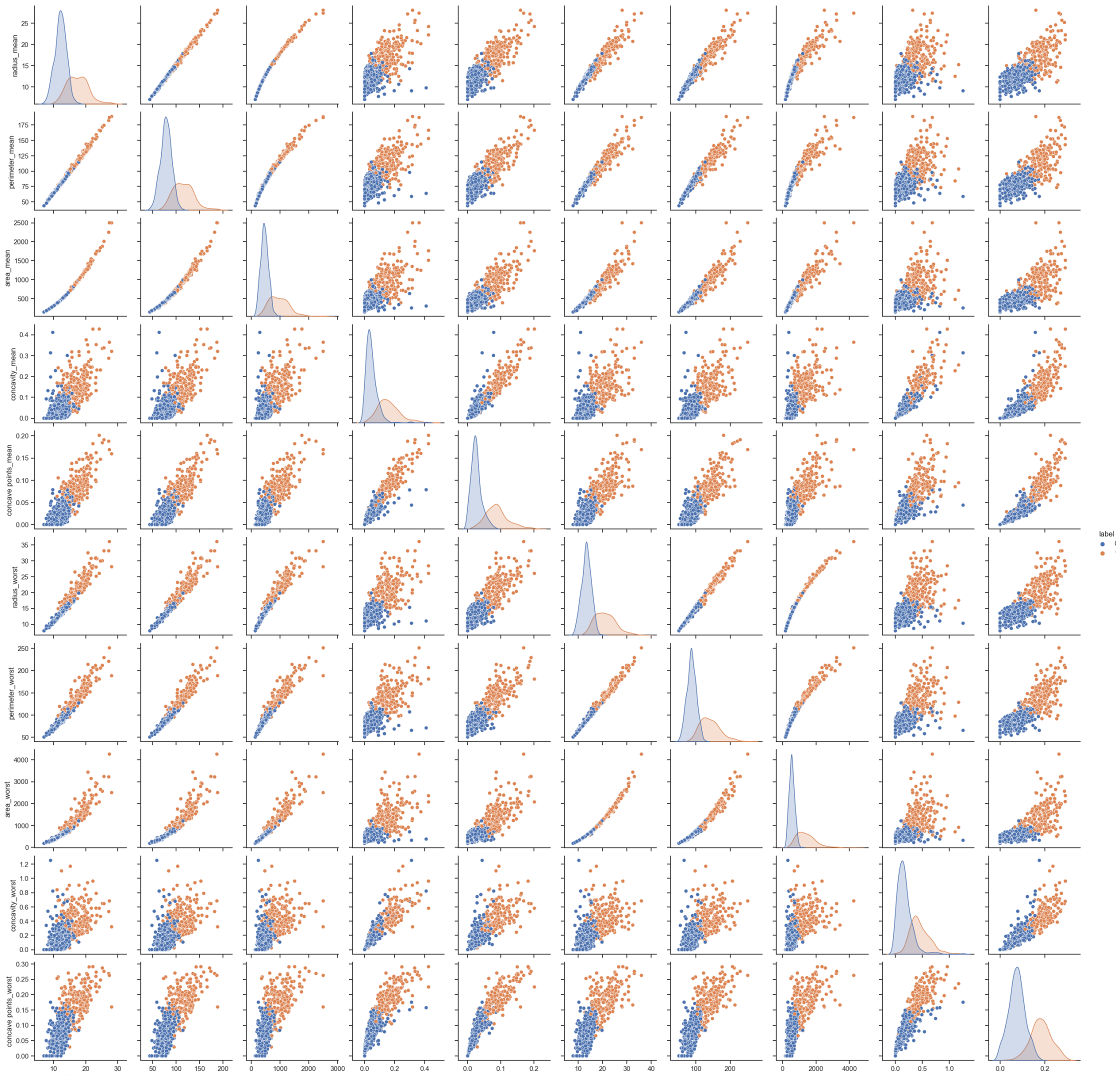
- set `diag_kind = "kde"`
- set `hue = "label"`

[17]:
```
# pairplot for the features with 60% and higher correlation value with the label

high_corr_features = target_corr.index.tolist()
high_corr_features.remove('label')
df_filtered = df2[['label'] + high_corr_features]

sns.set(style="ticks")
```

```python
sns.pairplot(df_filtered, diag_kind="kde", hue="label")
plt.show()
```



<p>3f. Separate features (X) from labels (y) using 60%+ correlatio

[18]:
```python
X = df_filtered.drop('label', axis=1)
y = df_filtered['label']
```

==
4. Data Partitioning and Feature Scaling [10]

- **Splitting the dataset :** The data we use is usually split into training data and test data. The training set contains a known output and the model learns on this data in order to be generalized to other data later on. We have the test dataset (or subset) in order to test our model's prediction on this subset. We will do this using `SciKit-Learn` library in Python using the `train_test_split` method.

`<p><b>4a.</b> Split the dataset into train and test set using the <b>|60%+|</b> correlation fea`

- Split into 80-20%
- Check and verify in `percentages` on the shape of the `X_train` and `X_test` sets.

[19]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

print("Shape: X_train:", X_train.shape)
print("Shape: X_test:", X_test.shape)
```

```
Shape: X_train: (455, 10)
Shape: X_test: (114, 10)
```

4b. Scale your features using StandardScaler method. 4 marks

We look at the data need for standardization, if there are big differences between the data, standardization is required.

Most of the times, your dataset will contain features highly varying in magnitudes, units and range. But since, most of the machine learning algorithms use Eucledian distance between two data points in their computations. We need to bring all features to the same level of magnitudes. This can be achieved by scaling. This means that you're transforming your data so that it fits within a specific scale, like `0-100` or `0-1`. We will use `StandardScaler` method from `Scikit-Learn` library.

[20]:
```
scaler = StandardScaler()
scaled_df = scaler.fit_transform(df2)
scaled_df = pd.DataFrame(scaled_df, columns = df2.columns)
```

==
5. Machine Learning Models Selection and Performance Evaluation [24]

This phase is known as Algorithm selection for Predicting the best results.

You are required to train the following models: - LogisticRegression - GradientBoostingClassifier - RandomForestClassifier

5a. Model Fitting. 11 marks

[21]:
```
# 1. Train LR model
log_regres = LogisticRegression()
log_regres.fit(X_train, y_train)

# 2. Train GB model
grad_boost_class = GradientBoostingClassifier()
grad_boost_class.fit(X_train, y_train)

# 3. Train RF model
rand_forest_class = RandomForestClassifier()
rand_forest_class.fit(X_train, y_train)
```

[21]: RandomForestClassifier()

5b. Compute the predictions of the trained models. 3 marks

[22]:
```
y_lr_train_prediction = log_regres.predict(X_train)
y_lr_test_prediction = log_regres.predict(X_test)

y_boost_train_prediction = grad_boost_class.predict(X_train)
y_boost_test_prediction = grad_boost_class.predict(X_test)

y_forest_train_prediction = rand_forest_class.predict(X_train)
y_forest_test_prediction = rand_forest_class.predict(X_test)
```

5c.Evaluate model performance using Accuracy score, and Confusion Matrix. 10 marks

- Accuracy scores for all 3 models and view in a dataframe sorted by `accuracy_score`
- Confusion matrix of the best model based on `accuracy score`

[23]:
```
accuracy_scores = {
    'Logistic Regression': accuracy_score(y_lr_test_prediction,␣
 ↪y_lr_train_prediction),
    'Gradient Boosting': accuracy_score(y_boost_test_prediction,␣
 ↪y_boost_train_prediction),
    'Random Forest': accuracy_score(y_forest_test_prediction,␣
 ↪y_forest_test_prediction)
}

accuracy_df = pd.DataFrame.from_dict(accuracy_scores, orient='index',␣
 ↪columns=['Accuracy Score'])
accuracy_df = accuracy_df.sort_values(by='Accuracy Score', ascending=False)

print("Accuracy Scores for all three models:")
print(accuracy_df)

best_model = accuracy_df.index[0]

# compute confusion matrix
conf_matrix = confusion_matrix(y_test, eval(f'y_pred_{best_model.replace(" ",␣
 ↪"_")}'))
print(f"\nConfusion Matrix of the best model ({best_model}):")
print(conf_matrix)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[23], line 2
      1 accuracy_scores = {
----> 2     'Logistic Regression':␣
 ↪accuracy_score(y_lr_test_prediction, y_lr_train_prediction),
```

```
      3     'Gradient Boosting': accuracy_score(y_boost_test_prediction,␣
↪y_boost_train_prediction),
      4     'Random Forest': accuracy_score(y_forest_test_prediction,␣
↪y_forest_test_prediction)
      5 }
      7 accuracy_df = pd.DataFrame.from_dict(accuracy_scores, orient='index',␣
↪columns=['Accuracy Score'])
      8 accuracy_df = accuracy_df.sort_values(by='Accuracy Score',␣
↪ascending=False)

File c:
↪\Users\User\miniconda3\envs\datascience_python\lib\site-packages\sklearn\utils\_param_valid
↪py:192, in validate_params.<locals>.decorator.<locals>.wrapper(*args, **kwargs)
    187 validate_parameter_constraints(
    188     parameter_constraints, params, caller_name=func.__qualname__
    189 )
    191 try:
--> 192     return func(*args, **kwargs)
    193 except InvalidParameterError as e:
    194     # When the function is just a wrapper around an estimator, we allow
    195     # the function to delegate validation to the estimator, but we␣
↪replace
    196     # the name of the estimator by the name of the function in the error
    197     # message to avoid confusion.
    198     msg = re.sub(
    199         r"parameter of \w+ must be",
    200         f"parameter of {func.__qualname__} must be",
    201         str(e),
    202     )

File c:
↪\Users\User\miniconda3\envs\datascience_python\lib\site-packages\sklearn\metrics\_classific
↪py:221, in accuracy_score(y_true, y_pred, normalize, sample_weight)
    155 """Accuracy classification score.
    156
    157 In multilabel classification, this function computes subset accuracy:
   (…)
    217 0.5
    218 """
    220 # Compute accuracy for each possible representation
--> 221 y_type, y_true, y_pred = _check_targets(y_true, y_pred)
    222 check_consistent_length(y_true, y_pred, sample_weight)
    223 if y_type.startswith("multilabel"):

File c:
↪\Users\User\miniconda3\envs\datascience_python\lib\site-packages\sklearn\metrics\_classific
↪py:86, in _check_targets(y_true, y_pred)
     59 def _check_targets(y_true, y_pred):
```

```
     60     """Check that y_true and y_pred belong to the same classification␣
  ↪task.
     61 
     62     This converts multiclass or binary types to a common shape, and␣
  ↪raises a
   (…)
     84     y_pred : array or indicator matrix
     85     """
---> 86     check_consistent_length(y_true, y_pred)
     87     type_true = type_of_target(y_true, input_name="y_true")
     88     type_pred = type_of_target(y_pred, input_name="y_pred")

File c:
  ↪\Users\User\miniconda3\envs\datascience_python\lib\site-packages\sklearn\utils\validation.
  ↪py:397, in check_consistent_length(*arrays)
    395 uniques = np.unique(lengths)
    396 if len(uniques) > 1:
--> 397     raise ValueError(
    398         "Found input variables with inconsistent numbers of samples: %r
    399         % [int(l) for l in lengths]
    400     )

ValueError: Found input variables with inconsistent numbers of samples: [114,␣
  ↪455]
```