sharonbuhlungu-577878-mlg382-st

May 17, 2024

===

SUMMATIVE TEST:	Total:	Duration:
Machine Learning 382	90 marks	120 + 15 minutes

Examiner:	Moderator:
Cheteni E.	Tavagwisa C.

===

0.0.1 Instructions

1. General - Once you finish you need to upload your solution to AssessmentQ, you will also need to click `Finish`. Do this before the allocated time as indicated on AssessmentQ. - No explanation of the questions may be asked or shall be given. - Candidates shall not communicate or attempt to communicate with anyone during the test time, and candidates shall not conduct themselves in an improper or unseemly manner. - No instructions or directives of the invigilator shall be disregarded; if any of the instructions are disobeyed, candidates shall expose themselves to disqualification from future test.

2. Practical - All practical work must be saved on `AssessmentQ` on the completion of the test, using the file name convention `StudentFullName(student_i.d)_MLG382_ST.ipynb` and `StudentFullName(student_i.d)_MLG382_ST.html (or .pdf)`. It is your responsibility to ensure this. - Submit both `.ipynb` and `.html` and/or `.pdf` files. - Make sure to run all the code cells before converting to either `.html` or `.pdf` file. - Submit on AssessmentQ before the indicated time expires.

===

[1]:
```python
# importing datetime module for now()
import datetime as dt
# using now() to get current time
d = dt.datetime.now().day
m = dt.datetime.now().month
y = dt.datetime.now().year
print (f"Date :{d}/{m}/{y}")
```

Date :17/5/2024

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] 1a. Import libraries for data manipulation? 2 marks

[1]:
```
import warnings
warnings.filterwarnings('ignore') # optional to handle warnings
import numpy as np
import pandas as pd
```

1b. Import libraries for data visualization? 2 marks

[2]:
```
import matplotlib as mt
import seaborn as sns
```

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

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

[8]:
```
df= pd.read_csv('cancer_data.csv')
df.head()
```

[8]:
```
         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

[10]:
```python
#Code + comment [2]
df.shape
# YOUR CODE HERE!
```

[10]:
```
(569, 33)
```

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

[17]:
```
#Code plus comment [2]
df.isnull().sum()

# YOUR CODE HERE!
```

[17]:
```
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
```

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

[26]:
```
# dropping the irrelevant features for prediction [2]
df.drop(columns=['id','Unnamed: 32'], axis=1)
```

```
# Comment results [1]
```

[26]:

```
    diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0           M        17.99         10.38          122.80     1001.0
1           M        20.57         17.77          132.90     1326.0
2           M        19.69         21.25          130.00     1203.0
3           M        11.42         20.38           77.58      386.1
4           M        20.29         14.34          135.10     1297.0
..        …          …            …             …          …
564         M        21.56         22.39          142.00     1479.0
565         M        20.13         28.25          131.20     1261.0
566         M        16.60         28.08          108.30      858.1
567         M        20.60         29.33          140.10     1265.0
568         B         7.76         24.54           47.92      181.0

     smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0            0.11840           0.27760         0.30010              0.14710
1            0.08474           0.07864         0.08690              0.07017
2            0.10960           0.15990         0.19740              0.12790
3            0.14250           0.28390         0.24140              0.10520
4            0.10030           0.13280         0.19800              0.10430
..               …                 …               …                    …
564          0.11100           0.11590         0.24390              0.13890
565          0.09780           0.10340         0.14400              0.09791
566          0.08455           0.10230         0.09251              0.05302
567          0.11780           0.27700         0.35140              0.15200
568          0.05263           0.04362         0.00000              0.00000

     symmetry_mean  …  radius_worst  texture_worst  perimeter_worst  \
0           0.2419  …        25.380          17.33           184.60
1           0.1812  …        24.990          23.41           158.80
2           0.2069  …        23.570          25.53           152.50
3           0.2597  …        14.910          26.50            98.87
4           0.1809  …        22.540          16.67           152.20
..             …  …          …              …                …
564         0.1726  …        25.450          26.40           166.10
565         0.1752  …        23.690          38.25           155.00
566         0.1590  …        18.980          34.12           126.70
567         0.2397  …        25.740          39.42           184.60
568         0.1587  …         9.456          30.37            59.16

     area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0        2019.0           0.16220            0.66560           0.7119
1        1956.0           0.12380            0.18660           0.2416
2        1709.0           0.14440            0.42450           0.4504
3         567.7           0.20980            0.86630           0.6869
4        1575.0           0.13740            0.20500           0.4000
```

```
..        …          …           …         …
564     2027.0     0.14100     0.21130     0.4107
565     1731.0     0.11660     0.19220     0.3215
566     1124.0     0.11390     0.30940     0.3403
567     1821.0     0.16500     0.86810     0.9387
568      268.6     0.08996     0.06444     0.0000

     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
..                    …              …                       …
564                0.2216          0.2060                  0.07115
565                0.1628          0.2572                  0.06637
566                0.1418          0.2218                  0.07820
567                0.2650          0.4087                  0.12400
568                0.0000          0.2871                  0.07039

[569 rows x 31 columns]
```

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

[29]:
```
# YOUR CODE HERE! [3]

df.duplicated().sum()
```

[29]: 0

==

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]

[38]:
```
# i. renaming the title of properties as per need of prediction [2]

df.columns= df.columns.str.replace('diagnosis','label')
```

```
df.columns
    # YOUR CODE HERE!
```

[38]:
```
Index(['id', 'label', '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', 'Unnamed: 32'],
      dtype='object')
```

[ ]:
```
# ii. Countplot [3]

sns.load_dataset('cancer_data.csv')
sns.countplot(x='label',data=df)

# YOUR CODE HERE!
```

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.

[83]:
```
# iii. Label encoding [3]
#df['label']= df['label'].astype('int')

df['label'].value_counts()

encoding_data= pd.get_dummies(df,columns=['label'])
encoding_data







# YOUR CODE HERE!
```

[83]:
```
         id  radius_mean  texture_mean  perimeter_mean  area_mean  \
0    842302        17.99         10.38          122.80     1001.0
1    842517        20.57         17.77          132.90     1326.0
2  84300903        19.69         21.25          130.00     1203.0
3  84348301        11.42         20.38           77.58      386.1
4  84358402        20.29         14.34          135.10     1297.0
```

```
..       …         …          …          …        …
564   926424       21.56        22.39        142.00    1479.0
565   926682       20.13        28.25        131.20    1261.0
566   926954       16.60        28.08        108.30     858.1
567   927241       20.60        29.33        140.10    1265.0
568    92751        7.76        24.54         47.92     181.0

     smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0            0.11840           0.27760         0.30010              0.14710
1            0.08474           0.07864         0.08690              0.07017
2            0.10960           0.15990         0.19740              0.12790
3            0.14250           0.28390         0.24140              0.10520
4            0.10030           0.13280         0.19800              0.10430
..               …                …               …                   …
564          0.11100           0.11590         0.24390              0.13890
565          0.09780           0.10340         0.14400              0.09791
566          0.08455           0.10230         0.09251              0.05302
567          0.11780           0.27700         0.35140              0.15200
568          0.05263           0.04362         0.00000              0.00000

     symmetry_mean  …  area_worst  smoothness_worst  compactness_worst  \
0           0.2419  …      2019.0           0.16220            0.66560
1           0.1812  …      1956.0           0.12380            0.18660
2           0.2069  …      1709.0           0.14440            0.42450
3           0.2597  …       567.7           0.20980            0.86630
4           0.1809  …      1575.0           0.13740            0.20500
..             …  …         …               …                  …
564         0.1726  …      2027.0           0.14100            0.21130
565         0.1752  …      1731.0           0.11660            0.19220
566         0.1590  …      1124.0           0.11390            0.30940
567         0.2397  …      1821.0           0.16500            0.86810
568         0.1587  …       268.6           0.08996            0.06444

     concavity_worst  concave points_worst  symmetry_worst  \
0             0.7119                0.2654          0.4601
1             0.2416                0.1860          0.2750
2             0.4504                0.2430          0.3613
3             0.6869                0.2575          0.6638
4             0.4000                0.1625          0.2364
..               …                   …              …
564           0.4107                0.2216          0.2060
565           0.3215                0.1628          0.2572
566           0.3403                0.1418          0.2218
567           0.9387                0.2650          0.4087
568           0.0000                0.0000          0.2871

     fractal_dimension_worst  Unnamed: 32  label_B  label_M
```

```
0                     0.11890          NaN    False     True
1                     0.08902          NaN    False     True
2                     0.08758          NaN    False     True
3                     0.17300          NaN    False     True
4                     0.07678          NaN    False     True
..                        …          …       …       …
564                   0.07115          NaN    False     True
565                   0.06637          NaN    False     True
566                   0.07820          NaN    False     True
567                   0.12400          NaN    False     True
568                   0.07039          NaN     True    False

[569 rows x 34 columns]
```

[80]:
```python
# iv. Confirming counts of respective classes [3]

df['label'].value_counts()

# YOUR CODE HERE!
```

[80]:
```
label
B    357
M    212
Name: count, dtype: int64
```

- **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

[90]:
```python
# 5 number summary [2]
#!pip install pandas
df['label'].describe().loc[['min','25%','50%','75%','max']]

# YOUR CODE HERE!
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[90], line 3
      1 # 5 number summary [2]
      2 #!pip install pandas
----> 3 df['label'].describe().loc[['min','25%','50%','75%','max']]

File ~\anaconda3\Lib\site-packages\pandas\core\indexing.py:1153, in␣
↪_LocationIndexer.__getitem__(self, key)
   1150 axis = self.axis or 0
   1152 maybe_callable = com.apply_if_callable(key, self.obj)
-> 1153 return self._getitem_axis(maybe_callable, axis=axis)

File ~\anaconda3\Lib\site-packages\pandas\core\indexing.py:1382, in _LocIndexer
↪_getitem_axis(self, key, axis)
   1379     if hasattr(key, "ndim") and key.ndim > 1:
   1380         raise ValueError("Cannot index with multidimensional key")
-> 1382     return self._getitem_iterable(key, axis=axis)
   1384 # nested tuple slicing
   1385 if is_nested_tuple(key, labels):

File ~\anaconda3\Lib\site-packages\pandas\core\indexing.py:1322, in _LocIndexer
↪_getitem_iterable(self, key, axis)
   1319 self._validate_key(key, axis)
   1321 # A collection of keys
-> 1322 keyarr, indexer = self._get_listlike_indexer(key, axis)
   1323 return self.obj._reindex_with_indexers(
   1324     {axis: [keyarr, indexer]}, copy=True, allow_dups=True
   1325 )

File ~\anaconda3\Lib\site-packages\pandas\core\indexing.py:1520, in _LocIndexer
↪_get_listlike_indexer(self, key, axis)
   1517 ax = self.obj._get_axis(axis)
   1518 axis_name = self.obj._get_axis_name(axis)
-> 1520 keyarr, indexer = ax._get_indexer_strict(key, axis_name)
   1522 return keyarr, indexer

File ~\anaconda3\Lib\site-packages\pandas\core\indexes\base.py:6115, in Index.
↪_get_indexer_strict(self, key, axis_name)
   6112 else:
   6113     keyarr, indexer, new_indexer = self._reindex_non_unique(keyarr)
```

```
-> 6115 self._raise_if_missing(keyarr, indexer, axis_name)
   6117 keyarr = self.take(indexer)
   6118 if isinstance(key, Index):
   6119     # GH 42790 - Preserve name from an Index

File ~\anaconda3\Lib\site-packages\pandas\core\indexes\base.py:6176, in Index.
 ↪_raise_if_missing(self, key, indexer, axis_name)
   6174     if use_interval_msg:
   6175         key = list(key)
-> 6176     raise KeyError(f"None of [{key}] are in the [{axis_name}]")
   6178 not_found = list(ensure_index(key)[missing_mask.nonzero()[0]].unique())
   6179 raise KeyError(f"{not_found} not in index")

KeyError: "None of [Index(['min', '25%', '50%', '75%', 'max'], dtype='object')]
 ↪are in the [index]"
```

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

[96]:
```
#correlation matrix [4]
df.corr()


# YOUR CODE HERE!
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[96], line 2
      1 #correlation matrix [4]
----> 2 df.corr()

File ~\anaconda3\Lib\site-packages\pandas\core\frame.py:10704, in DataFrame.
 ↪corr(self, method, min_periods, numeric_only)
  10702 cols = data.columns
  10703 idx = cols.copy()
> 10704 mat = data.to_numpy(dtype=float, na_value=np.nan, copy=False)
  10706 if method == "pearson":
  10707     correl = libalgos.nancorr(mat, minp=min_periods)

File ~\anaconda3\Lib\site-packages\pandas\core\frame.py:1889, in DataFrame.
 ↪to_numpy(self, dtype, copy, na_value)
   1887 if dtype is not None:
   1888     dtype = np.dtype(dtype)
-> 1889 result = self._mgr.as_array(dtype=dtype, copy=copy, na_value=na_value)
   1890 if result.dtype is not dtype:
   1891     result = np.array(result, dtype=dtype, copy=False)
```

```
File ~\anaconda3\Lib\site-packages\pandas\core\internals\managers.py:1656, in␣
↪BlockManager.as_array(self, dtype, copy, na_value)
   1654         arr.flags.writeable = False
   1655 else:
-> 1656     arr = self._interleave(dtype=dtype, na_value=na_value)
   1657     # The underlying data was copied within _interleave, so no need
   1658     # to further copy if copy=True or setting na_value
   1660 if na_value is lib.no_default:

File ~\anaconda3\Lib\site-packages\pandas\core\internals\managers.py:1715, in␣
↪BlockManager._interleave(self, dtype, na_value)
   1713     else:
   1714         arr = blk.get_values(dtype)
-> 1715     result[rl.indexer] = arr
   1716     itemmask[rl.indexer] = 1
   1718 if not itemmask.all():

ValueError: could not convert string to float: 'M'
```

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]

[ ]:
```
# YOUR CODE HERE!  [5]
plt.figure(figsize=(10,5))
c=df.corr()
sns.heatmap(c,cmap=BrBG,annot=True)
```

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

[ ]:
```
# YOUR CODE HERE!  [6]
```

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

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

[3]:
```
# pairplot for the features with 60% and higher correlation value with the␣
↪label  [3]
```

```python
sns.set()
sns.pairplot(dataset,hue='label')
sns.plt.show()
# YOUR CODE HERE!
```

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

[ ]:
```python
# YOUR CODE HERE!
# DataFrame with specified features [1]
X= df.drop('dataset',axis=1)
y=df['dataset']
# X = .... [1]
# y = .... [1]
```

===
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>4a. Split the dataset into train and test set using the |60%+| correlation fea

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

[97]:
```python
# importing train_test_split from scikit-learn [1]
# YOUR CODE HERE!
from sklearn.model_selection import train_test_split
X=df.drop(columns='label','Unnamed: 32')
y= df.['label']
X_test,X_train,y_test,y_train =train_test_split(X,y,train=0.8,random_state=42)

# splitting the data to 80-20% [5]

# YOUR CODE HERE!
```

[97]:
```
           id label  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
..        …   …          …             …               …          …
564    926424     M        21.56         22.39          142.00     1479.0
565    926682     M        20.13         28.25          131.20     1261.0
566    926954     M        16.60         28.08          108.30      858.1
```

```
567    927241      M        20.60         29.33           140.10     1265.0
568     92751      B         7.76         24.54            47.92      181.0

     smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0            0.11840           0.27760         0.30010              0.14710
1            0.08474           0.07864         0.08690              0.07017
2            0.10960           0.15990         0.19740              0.12790
3            0.14250           0.28390         0.24140              0.10520
4            0.10030           0.13280         0.19800              0.10430
..               …                 …               …                    …
564          0.11100           0.11590         0.24390              0.13890
565          0.09780           0.10340         0.14400              0.09791
566          0.08455           0.10230         0.09251              0.05302
567          0.11780           0.27700         0.35140              0.15200
568          0.05263           0.04362         0.00000              0.00000

     …  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0    …          17.33           184.60      2019.0           0.16220
1    …          23.41           158.80      1956.0           0.12380
2    …          25.53           152.50      1709.0           0.14440
3    …          26.50            98.87       567.7           0.20980
4    …          16.67           152.20      1575.0           0.13740
..   …            …                …           …                 …
564  …          26.40           166.10      2027.0           0.14100
565  …          38.25           155.00      1731.0           0.11660
566  …          34.12           126.70      1124.0           0.11390
567  …          39.42           184.60      1821.0           0.16500
568  …          30.37            59.16       268.6           0.08996

     compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0              0.66560           0.7119                0.2654          0.4601
1              0.18660           0.2416                0.1860          0.2750
2              0.42450           0.4504                0.2430          0.3613
3              0.86630           0.6869                0.2575          0.6638
4              0.20500           0.4000                0.1625          0.2364
..                 …                …                     …               …
564            0.21130           0.4107                0.2216          0.2060
565            0.19220           0.3215                0.1628          0.2572
566            0.30940           0.3403                0.1418          0.2218
567            0.86810           0.9387                0.2650          0.4087
568            0.06444           0.0000                0.0000          0.2871

     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
..                       …          …
564                   0.07115          NaN
565                   0.06637          NaN
566                   0.07820          NaN
567                   0.12400          NaN
568                   0.07039          NaN

[569 rows x 33 columns]
```

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.

[ ]:
```python
# 4 marks
# YOUR CODE HERE! [3]

from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()

X = df[['Weight', 'Volume']]

scaledX = scale.fit_transform(X)

print(scaledX)

# confirm the results [1]
# YOUR CODE HERE!
```

==

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

[ ]:
```python
# Loading libraries for the models to be trained [5]
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import accuracy_score
from sklearn.datasets import load_digits


# YOUR CODE HERE!
```

[ ]:
```
# 10 marks for model training

# 1. Train LR model [2]
model = LogisticRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print(f'the model accuracy: {accuracy}')

# 2. Train GB model [2]
gbc = GradientBoostingClassifier(n_estimators=300,learning_rate=0.
 ↪05,random_state=100,max_features=5 )
gbc.fit(train_X, train_y)
pred_y = gbc.predict(test_X)
acc = accuracy_score(test_y, pred_y)

# 5. Train RF model [2]
forest_model = RandomForestRegressor(random_state=1)
forest_model.fit(train_X, train_y)
melb_preds = forest_model.predict(val_X)
print(mean_absolute_error(val_y, melb_preds))
```

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

[ ]:
```
# 5 marks for making predictions

# Make LR predictions [1]
# YOUR CODE HERE!

# Make RF predictions [1]
# YOUR CODE HERE!

# Make KNN predictions [1]
# YOUR CODE HERE!
```

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`

[ ]:
```
# Importing 3 metrics [3]

# YOUR CODE HERE!

# Accuracy scores of 3 trained models [3]

# YOUR CODE HERE!

# Compute confusion matrix [4]
cm = confusion_matrix(actual,predicted)
cm = confusion_matrix(actual,predicted)
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['M','Not M'],
            yticklabels=['B','Not B'])
plt.xlabel('Prediction',fontsize=13)
plt.ylabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()
```

==

6. Conclusion [6]

6a. Interpret values obtained from the confusion matrix. 4 marks

Comment: [4]

- False Positive
- False Negative

[ ]:
```
# - ***YOUR COMMENT HERE!*** [4]
```

6b. Conclusive remarks on the best model. 2 marks

- Your comment is not limited to what you obtained but additional interpretation will be credited.

[ ]:
```
# - ***YOUR COMMENT HERE!*** [2]
```

==

1.0.1 Thank you for completing this Test!

##