

1 Exploratory Data Analysis (EDA)

1.1. Understanding the data

1.2. Impuding missing values

1.1 Understand the data

In [178...]

```
# Import all the tools we need
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
```

In [179...]

```
df_original = pd.read_excel('/content/cancer.xlsx')
```

In [180...]

```
df = df_original.copy()
df.head()
```

Out[180]:

	index	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	com
0	1	842302	17.99	10.38	122.80	1001.0	0.11840	
1	2	842517	20.57	17.77	132.90	1326.0	0.08474	
2	3	84300903	19.69	21.25	130.00	1203.0	0.10960	
3	4	84348301	11.42	20.38	77.58	386.1	0.14250	
4	5	84358402	20.29	14.34	135.10	1297.0	0.10030	

5 rows × 36 columns

1. Understanding the data

In [181...]

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 36 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   index            569 non-null    int64  
 1   id               569 non-null    int64  
 2   radius_mean      569 non-null    float64 
 3   texture_mean     569 non-null    float64 
 4   perimeter_mean   569 non-null    float64 
 5   area_mean        569 non-null    float64 
 6   smoothness_mean  569 non-null    float64 
 7   compactness_mean 569 non-null    float64 
 8   concavity_mean   552 non-null    float64 
 9   concave_points_mean 550 non-null    float64 
 10  symmetry_mean   568 non-null    float64 
 11  fractal_dimension 569 non-null    float64 
 12  radius_se        569 non-null    float64 
 13  texture_se       569 non-null    float64 
 14  perimeter_se    569 non-null    float64 
 15  area_se          569 non-null    float64 
 16  smoothness_se   569 non-null    float64 
 17  compactness_se  569 non-null    float64 
 18  concavity_se    554 non-null    float64 
 19  concave_points_se 550 non-null    float64 
 20  symmetry_se     569 non-null    float64 
 21  fractal_dimension_se 569 non-null    float64 
 22  radius_worst    569 non-null    float64 
 23  texture_worst   569 non-null    float64 
 24  texture_worst.1 569 non-null    float64 
 25  area_worst       569 non-null    float64 
 26  smoothness_worst 569 non-null    float64 
 27  compactness_worst 568 non-null    float64 
 28  concavity_worst 562 non-null    float64 
 29  concave_points_worst 563 non-null    float64 
 30  symmetry_worst  569 non-null    float64 
 31  fractal_dimension_worst 569 non-null    float64 
 32  N Stage          569 non-null    object  
 33  6th Stage        569 non-null    object  
 34  differentiate    569 non-null    object  
 35  diagnosis        569 non-null    object  
dtypes: float64(30), int64(2), object(4)
memory usage: 160.2+ KB
```

In [182... df.columns

```
Out[182]: Index(['index', 'id', 'radius_mean', 'texture_mean', 'perimeter_mean',
   'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
   'concave_points_mean', 'symmetry_mean', 'fractal_dimension',
   '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',
   'texture_worst.1', 'area_worst', 'smoothness_worst',
   'compactness_worst', 'concavity_worst', 'concave_points_worst',
   'symmetry_worst', 'fractal_dimension_worst', 'N Stage', '6th Stage',
   'differentiate', 'diagnosis'],
  dtype='object')
```

In [183... df.describe()

Out[183]:

	index	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness
count	569.000000	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000
mean	285.000000	3.037183e+07	14.127292	19.289649	91.969033	654.889104	
std	164.400426	1.250206e+08	3.524049	4.301036	24.298981	351.914129	
min	1.000000	8.670000e+03	6.981000	9.710000	43.790000	143.500000	
25%	143.000000	8.692180e+05	11.700000	16.170000	75.170000	420.300000	
50%	285.000000	9.060240e+05	13.370000	18.840000	86.240000	551.100000	
75%	427.000000	8.813129e+06	15.780000	21.800000	104.100000	782.700000	
max	569.000000	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	

8 rows × 32 columns

In [184... df.shape

Out[184]: (569, 36)

In [185... df.ndim

Out[185]: 2

In [186... df.isnull().sum()

```
Out[186]: index      0
          id        0
          radius_mean    0
          texture_mean    0
          perimeter_mean   0
          area_mean       0
          smoothness_mean   0
          compactness_mean   0
          concavity_mean     17
          concave_points_mean 19
          symmetry_mean      1
          fractal_dimension    0
          radius_se         0
          texture_se         0
          perimeter_se        0
          area_se            0
          smoothness_se        0
          compactness_se        0
          concavity_se        15
          concave_points_se     19
          symmetry_se          0
          fractal_dimension_se   0
          radius_worst        0
          texture_worst        0
          texture_worst.1      0
          area_worst           0
          smoothness_worst      0
          compactness_worst     1
          concavity_worst       7
          concave_points_worst   6
          symmetry_worst        0
          fractal_dimension_worst 0
          N Stage             0
          6th Stage            0
          differentiate        0
          diagnosis            0
          dtype: int64
```

1.2 Imputing missing values

```
In [187... df["concavity_mean"].fillna(df["concavity_mean"].median(), inplace = True)
df["concave_points_mean"].fillna(df["concave_points_mean"].mean(), inplace = True)
df["symmetry_mean"].fillna(df["symmetry_mean"].median(), inplace = True)
df["compactness_worst"].fillna(df["compactness_worst"].mean(), inplace = True)
df["concavity_se"].fillna(df["concavity_se"].mean(), inplace = True)
df["concave_points_se"].fillna(df["concave_points_se"].median(), inplace = True)
df["concavity_worst"].fillna(df["concavity_worst"].median(), inplace = True)
df["concave points_worst"].fillna(df["concave points_worst"].median(), inplace = True)

In [188... df=df.drop(['index','id'],axis=1)

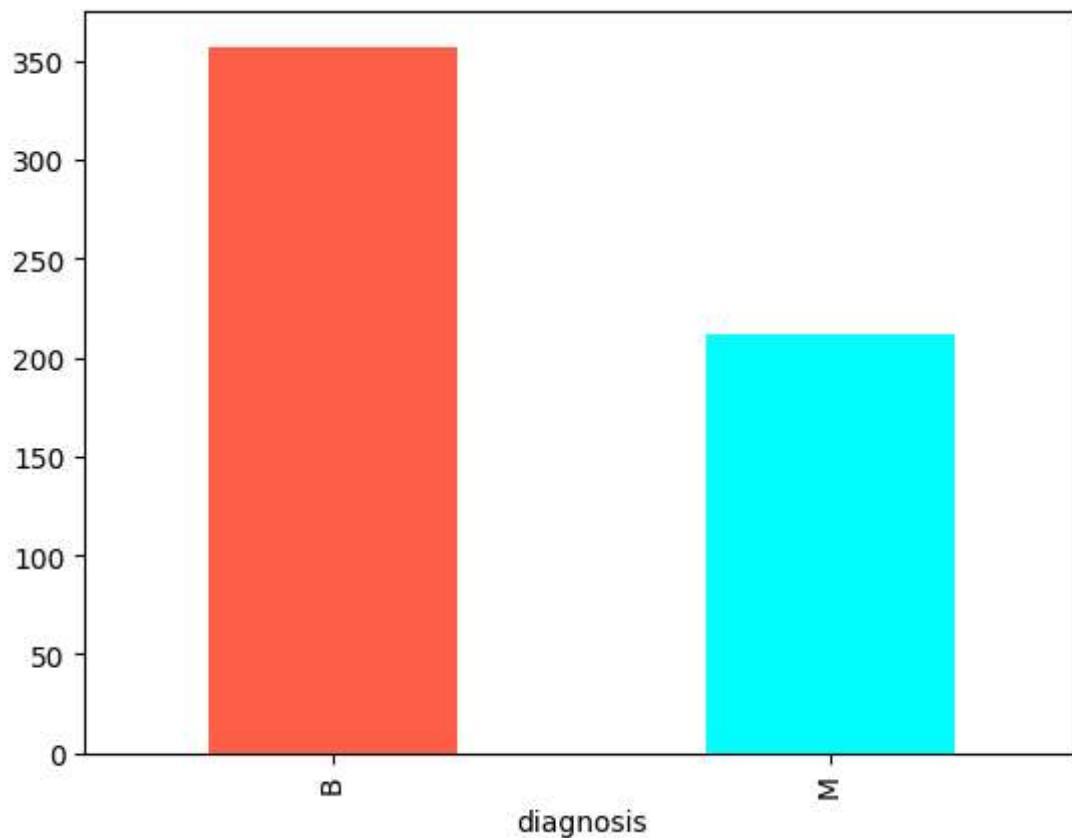
In [189... df.isnull().sum()
```

```
Out[189]: 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     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  
texture_worst.1          0  
area_worst               0  
smoothness_worst         0  
compactness_worst        0  
concavity_worst          0  
concave points_worst    0  
symmetry_worst           0  
fractal_dimension_worst 0  
N Stage                  0  
6th Stage                0  
differentiate            0  
diagnosis                0  
dtype: int64
```

```
In [190... df["diagnosis"].value_counts()
```

```
Out[190]: diagnosis  
B    357  
M    212  
Name: count, dtype: int64
```

```
In [191... df["diagnosis"].value_counts().plot(kind="bar",color=["tomato", "cyan"]);
```



2 Feature Selection

2.1. Correlation heatmap

2.2. Dropping features

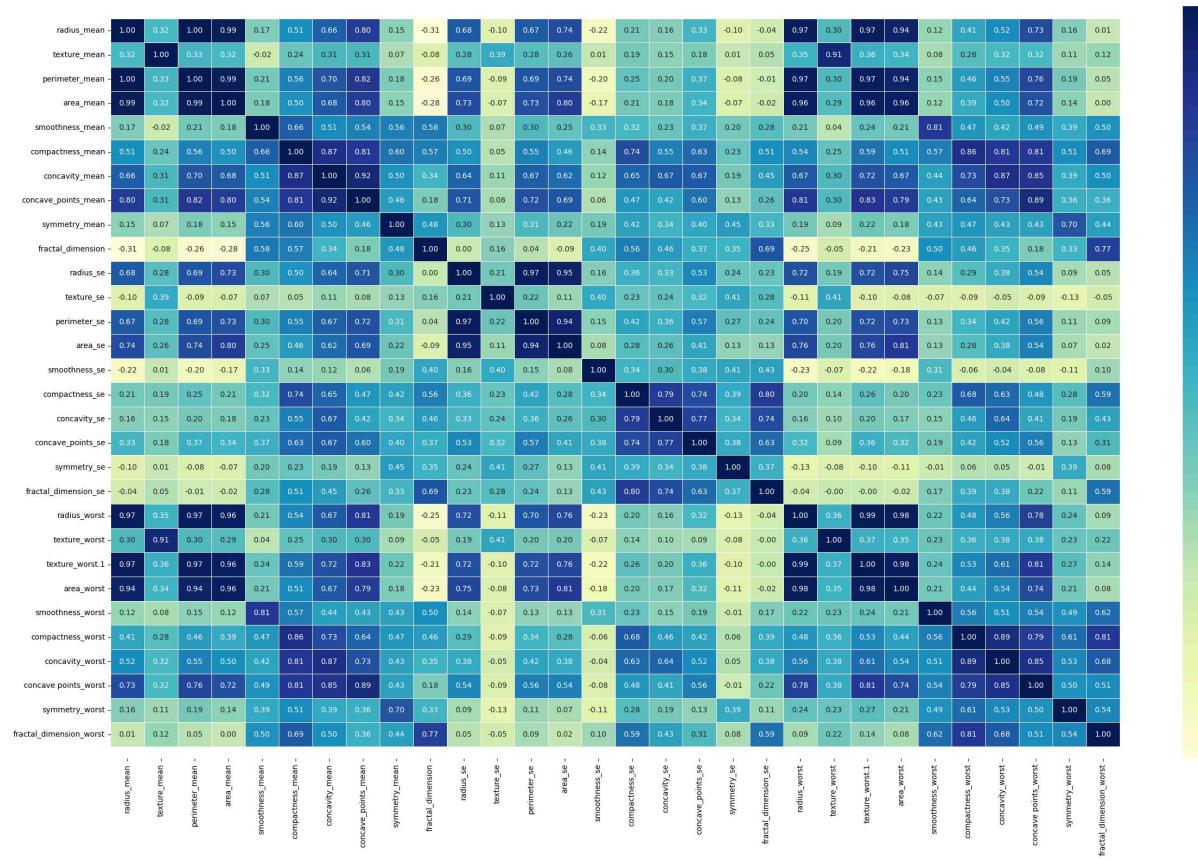
2.3. Dealing with outliers

2.4. Converting Categorical features to Numerical features

2.1 Correlation heatmap

```
In [192]:  
numeric_df = df.select_dtypes(include=['number'])  
corr_matrix = numeric_df.corr()  
fig, ax = plt.subplots(figsize=(30, 18))  
ax = sns.heatmap(corr_matrix,  
                  annot=True,  
                  linewidths=0.5,  
                  fmt=".2f",  
                  cmap="YlGnBu");  
bottom, top = ax.get_ylim()  
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[192]: (30.5, -0.5)



In [193...]: df1 = df.copy()

In [194...]: len(df1.columns)

Out[194]: 34

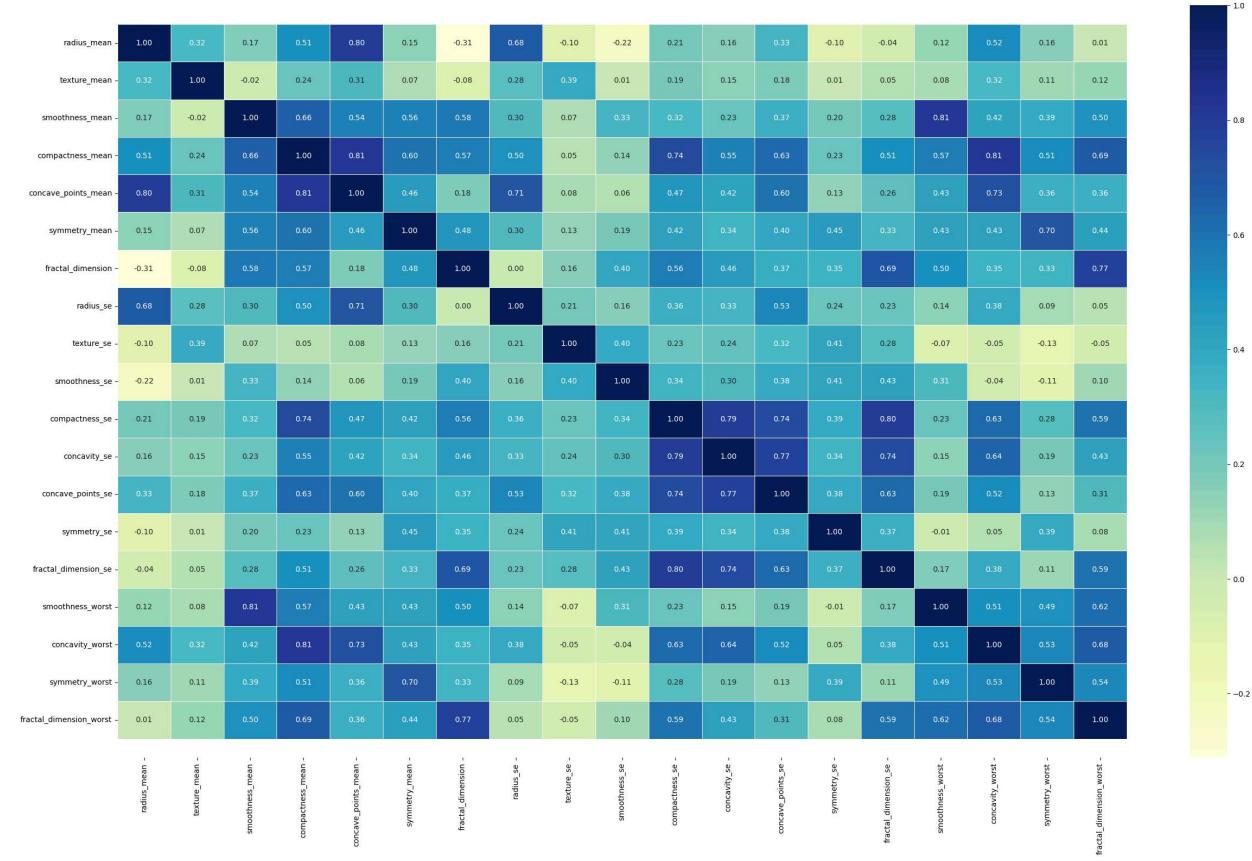
2.2 Dropping features

Dropping columns having correlation more than 0.85

In [195...]: df1.drop(['perimeter_mean', 'area_mean', 'radius_worst', 'texture_worst', 'texture_worst.1'])

```
numeric_df1 = df1.select_dtypes(include=['number'])
corr_matrix = numeric_df1.corr()
fig, ax = plt.subplots(figsize=(30, 18))
ax = sns.heatmap(corr_matrix,
                  annot=True,
                  linewidths=0.5,
                  fmt=".2f",
                  cmap="YlGnBu");
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

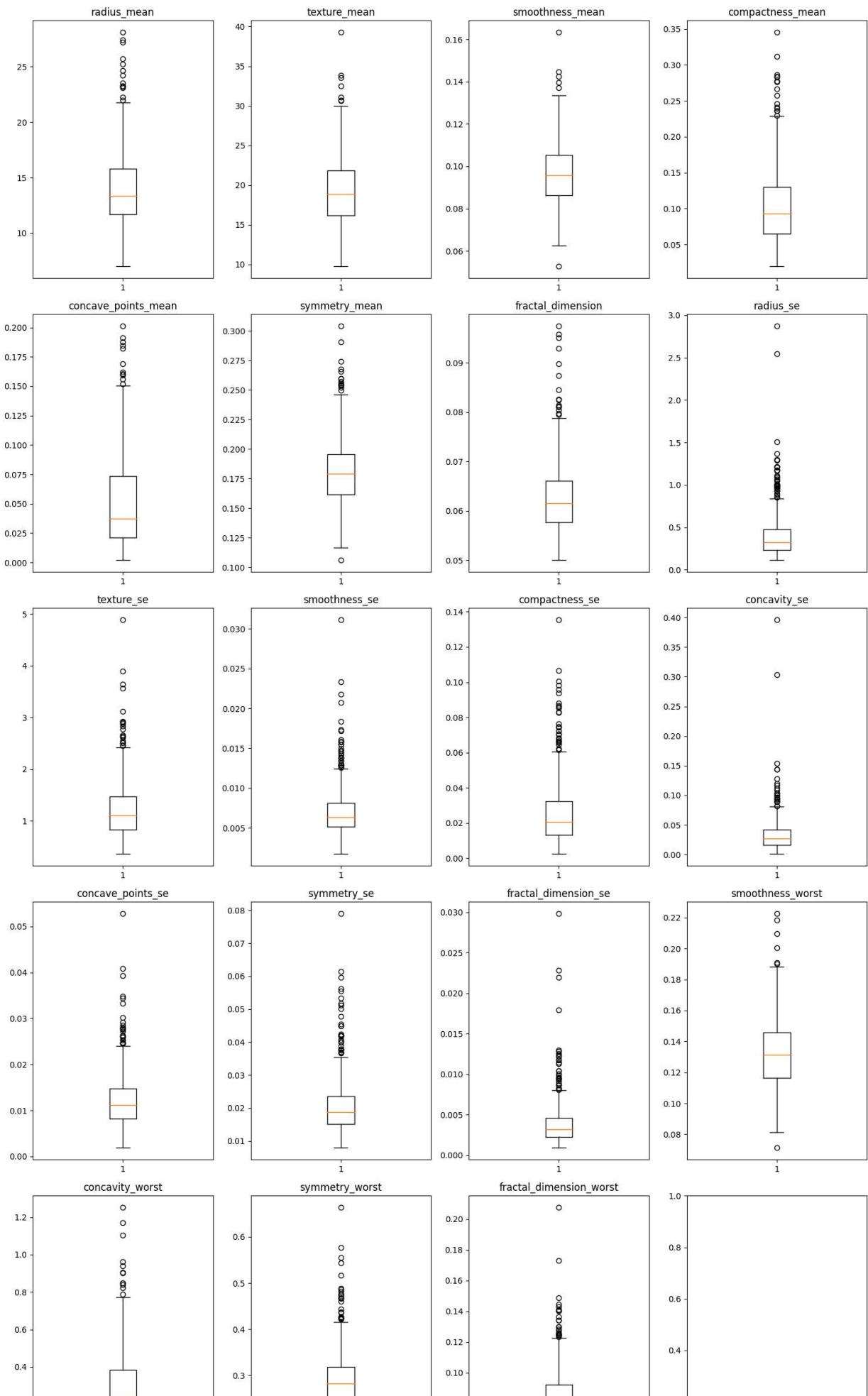
Out[196]: (19.5, -0.5)

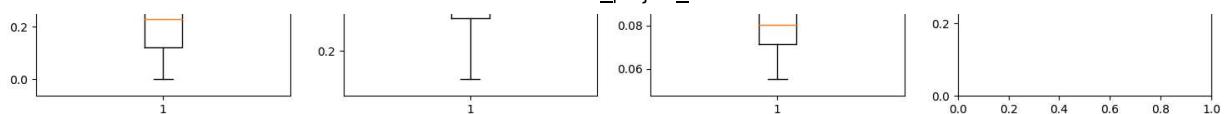


2.3 Dealing with outliers

```
In [197]: num_cols = 4 # Number of columns in each row of subplots
num_rows = (len(df1.columns) + num_cols - 1) // num_cols # Calculate the number of rows
num_rows -= 1
# Create subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5*num_rows))
# Flatten axes if needed
if num_rows == 1:
    axes = [axes]
# Plot boxplots for each numerical column
for i, column in enumerate(df1.columns):
    if df1[column].dtype != 'O': # Check if column is numeric
        row_index = i // num_cols
        col_index = i % num_cols
        axes[row_index][col_index].boxplot(df1[column])
        axes[row_index][col_index].set_title(column)
# Adjust Layout and display plots
plt.tight_layout()
plt.show()
```

ML_project_final





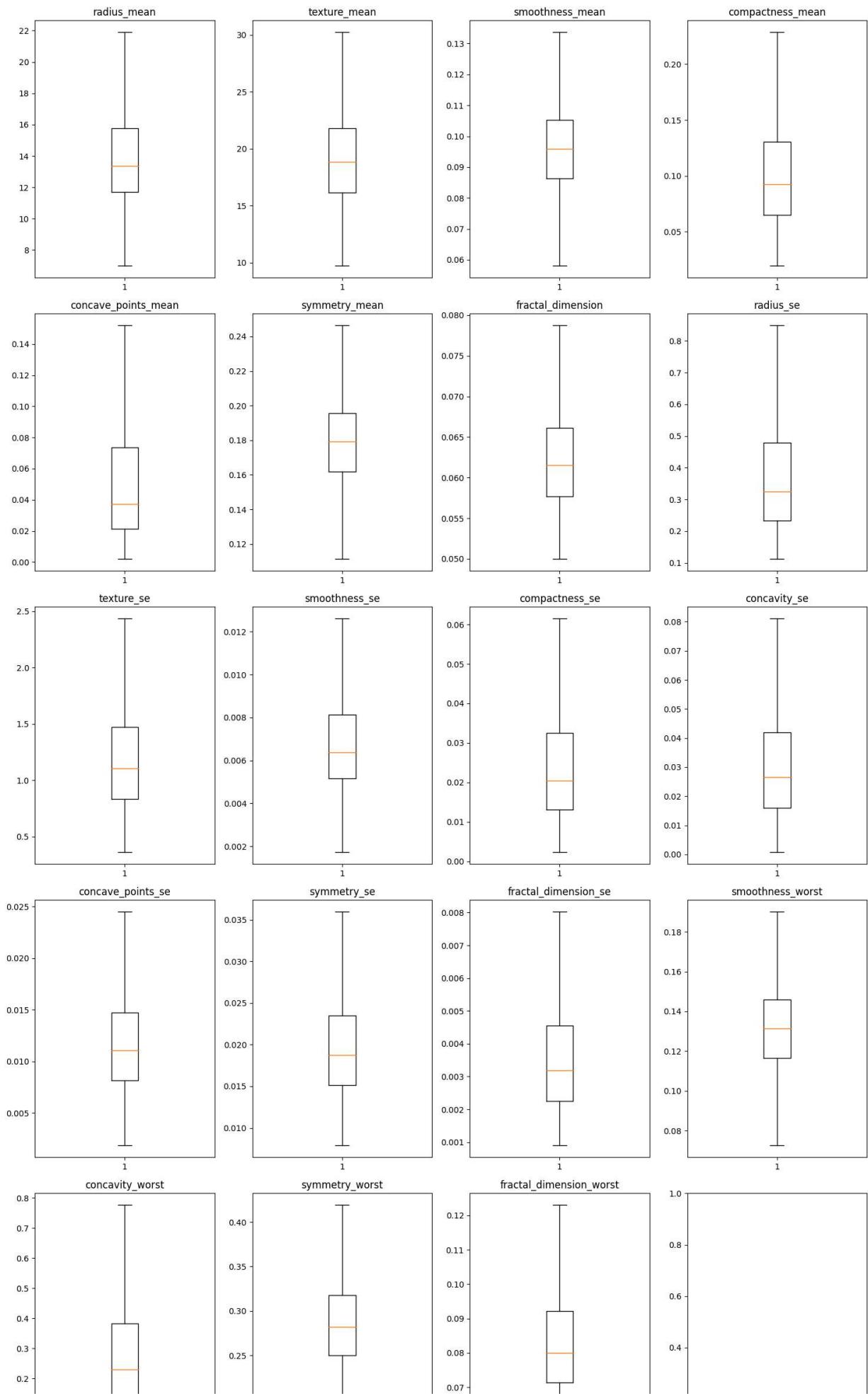
In [198...]

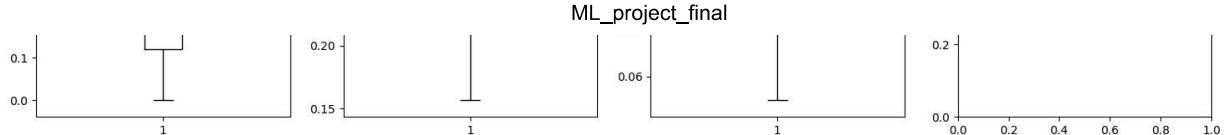
```
for i in df1.columns:
    if(df1[i].dtype != object):
        IQR = df1[i].quantile(0.75) - df1[i].quantile(0.25)
        upper_bound = df1[i].quantile(0.75) + 1.5*IQR
        lower_bound = df1[i].quantile(0.25) - 1.5*IQR
        df1.loc[df1[i] >= upper_bound,i] = upper_bound
        df1.loc[df1[i] <= lower_bound,i] = lower_bound
```

In [199...]

```
num_cols = 4 # Number of columns in each row of subplots
num_rows = (len(df1.columns) + num_cols - 1) // num_cols # Calculate the number of rows
num_rows -= 1
# Create subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5*num_rows))
# Flatten axes if needed
if num_rows == 1:
    axes = [axes]
# Plot boxplots for each numerical column
for i, column in enumerate(df1.columns):
    if df1[column].dtype != 'O': # Check if column is numeric
        row_index = i // num_cols
        col_index = i % num_cols
        axes[row_index][col_index].boxplot(df1[column])
        axes[row_index][col_index].set_title(column)
# Adjust layout and display plots
plt.tight_layout()
plt.show()
```

ML_project_final





2.4 Categorical to numerical

In [200...]: df1.columns

```
Out[200]: Index(['radius_mean', 'texture_mean', 'smoothness_mean', 'compactness_mean',
       'concave_points_mean', 'symmetry_mean', 'fractal_dimension',
       'radius_se', 'texture_se', 'smoothness_se', 'compactness_se',
       'concavity_se', 'concave_points_se', 'symmetry_se',
       'fractal_dimension_se', 'smoothness_worst', 'concavity_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'N Stage', '6th Stage',
       'differentiate', 'diagnosis'],
      dtype='object')
```

In [201...]: #find the categorical features

```
for i in df1.columns:
    if(df1[i].dtype == 'O'):
        print(i, "\t", df1[i].nunique())
```

N Stage	3
6th Stage	5
differentiate	4
diagnosis	2

Performing ONE HOT ENCODING

In [202...]: # Perform one-hot encoding

```
one_hot_encoded1 = pd.get_dummies(df['N Stage'])
one_hot_encoded2 = pd.get_dummies(df['6th Stage'])
one_hot_encoded3 = pd.get_dummies(df['differentiate'])
# Concatenate the one-hot encoded columns with the original DataFrame
df_encoded = pd.concat([df1, one_hot_encoded1, one_hot_encoded2, one_hot_encoded3], axis=1)
df_encoded.drop(['N Stage', '6th Stage', 'differentiate'], axis=1, inplace=True)
df_encoded.columns
```

```
Out[202]: Index(['radius_mean', 'texture_mean', 'smoothness_mean', 'compactness_mean',
       'concave_points_mean', 'symmetry_mean', 'fractal_dimension',
       'radius_se', 'texture_se', 'smoothness_se', 'compactness_se',
       'concavity_se', 'concave_points_se', 'symmetry_se',
       'fractal_dimension_se', 'smoothness_worst', 'concavity_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'diagnosis', 'N1', 'N2',
       'N3', 'IIA', 'IIB', 'IIIA', 'IIIB', 'IIIC', 'Moderately differentiated',
       'Poorly differentiated', 'Undifferentiated', 'Well differentiated'],
      dtype='object')
```

In [203...]: df_encoded['diagnosis'] = df_encoded['diagnosis'].map({'B': 0, 'M': 1})
df_encoded["diagnosis"].value_counts()

```
Out[203]: diagnosis
0    357
1    212
Name: count, dtype: int64
```

3 Modelling

We're going to try 6 different machine learning models:

1. Logistic Regression
2. K-Nearest Neighbours Classifier
3. Random Forest Classifier
4. Decision Tree
5. Support Vector Machine
6. Naive Bayes

In [204...]

```
X = df_encoded.drop("diagnosis", axis=1)

y = df_encoded["diagnosis"]

# Split data into train and test sets
np.random.seed(42)

# Split into train & test set
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2)
```

In [205...]

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
import numpy as np

# Put models in a dictionary
models = {"Logistic Regression": LogisticRegression(),
          "KNN": KNeighborsClassifier(),
          "Random Forest": RandomForestClassifier(),
          "Decision Tree": DecisionTreeClassifier(),
          "SVM": SVC(),
          "Naive Bayes": GaussianNB()}

# Create a function to fit and score models
def fit_and_score(models, X_train, X_test, y_train, y_test):
    """
    Fits and evaluates given machine learning models.
    models : a dict of different Scikit-Learn machine learning models
    X_train : training data (no labels)
    X_test : testing data (no labels)
    y_train : training labels
    y_test : test labels
    """
    # Set random seed
    np.random.seed(42)
    # Make a dictionary to keep model scores
```

```

model_scores = {}
model_accuracy = {}
report = {}
y_prediction = {}

# Loop through models
for name, model in models.items():
    # Fit the model to the data
    model.fit(X_train, y_train)
    # Evaluate the model and append its score to model_scores
    model_scores[name] = model.score(X_test, y_test)
    # Accuracy
    y_pred = model.predict(X_test)
    y_prediction[name] = y_pred
    model_accuracy[name] = accuracy_score(y_test, y_pred)
    report[name] = classification_report(y_test, y_pred, target_names=['malignant'])

return model_scores, model_accuracy, report, y_prediction

```

In [206...]

```

model_scores, model_accuracy, report, y_prediction = fit_and_score(models=models,
                                                                X_train=X_train,
                                                                X_test=X_test,
                                                                y_train=y_train,
                                                                y_test=y_test)

# for key in model_scores.keys():
#     print(f'{key}\t{model_scores[key]}')

max_length = max(len(key) for key in model_scores.keys())
for key, value in model_scores.items():
    print(f'{key.ljust(max_length)}\t{value}')

for key in report.keys():
    print(key, '\t', report[key])

```

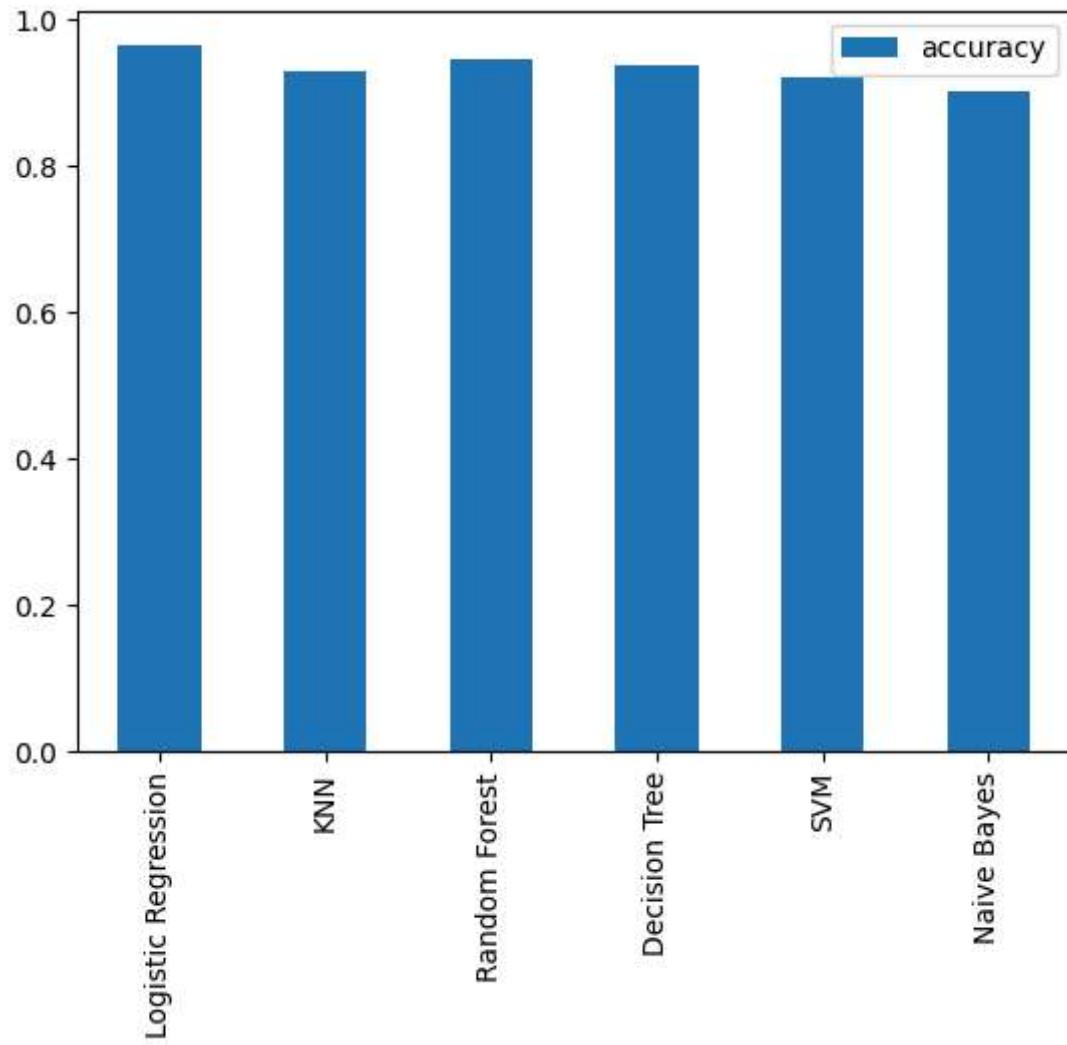
/usr/local/lib/python3.10/dist-packages/scikit-learn/_linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/_linear_model.html#logistic-regression

Logistic Regression	0.9649122807017544
KNN	0.9298245614035088
Random Forest	0.9473684210526315
Decision Tree	0.9385964912280702
SVM	0.9210526315789473
Naive Bayes	0.9035087719298246
Logistic Regression	precision recall f1-score support
malignant	0.97 0.97 0.97 71
benign	0.95 0.95 0.95 43
accuracy	0.96 0.96 0.96 114
macro avg	0.96 0.96 0.96 114
weighted avg	0.96 0.96 0.96 114
KNN	precision recall f1-score support
malignant	0.93 0.96 0.94 71
benign	0.93 0.88 0.90 43
accuracy	0.93 0.92 0.92 114
macro avg	0.93 0.92 0.92 114
weighted avg	0.93 0.93 0.93 114
Random Forest	precision recall f1-score support
malignant	0.95 0.97 0.96 71
benign	0.95 0.91 0.93 43
accuracy	0.95 0.94 0.95 114
macro avg	0.95 0.94 0.94 114
weighted avg	0.95 0.95 0.95 114
Decision Tree	precision recall f1-score support
malignant	0.96 0.94 0.95 71
benign	0.91 0.93 0.92 43
accuracy	0.93 0.94 0.94 114
macro avg	0.93 0.94 0.93 114
weighted avg	0.94 0.94 0.94 114
SVM	precision recall f1-score support
malignant	0.92 0.96 0.94 71
benign	0.93 0.86 0.89 43
accuracy	0.92 0.91 0.92 114
macro avg	0.92 0.91 0.91 114
weighted avg	0.92 0.92 0.92 114
Naive Bayes	precision recall f1-score support
malignant	0.88 0.99 0.93 71
benign	0.97 0.77 0.86 43
accuracy	0.92 0.88 0.90 114
macro avg	0.92 0.88 0.89 114
weighted avg	0.91 0.90 0.90 114

In [207...]

```
model_compare = pd.DataFrame(model_scores, index=["accuracy"])
model_compare.T.plot.bar();
```



4 Cross validation on Logistic Regression Model

In [208...]

```
from sklearn.model_selection import cross_val_score
# Initialize logistic regression model
model = LogisticRegression(max_iter=1000)

# Perform cross-validation
# Here, cv=5 means 5-fold cross-validation
# You can adjust cv parameter as needed
scores = cross_val_score(model, X, y, cv=5)

# Print the cross-validation scores
print("Cross-validation scores:", scores)

# Print the mean and standard deviation of the cross-validation scores
print("Mean accuracy:", scores.mean())
print("Standard deviation of accuracy:", scores.std())
```

Cross-validation scores: [0.89473684 0.92982456 0.97368421 0.93859649 0.9380531]

Mean accuracy: 0.934979040521658

Standard deviation of accuracy: 0.025165965701401824

In [209...]

y_prediction

Out[209]:

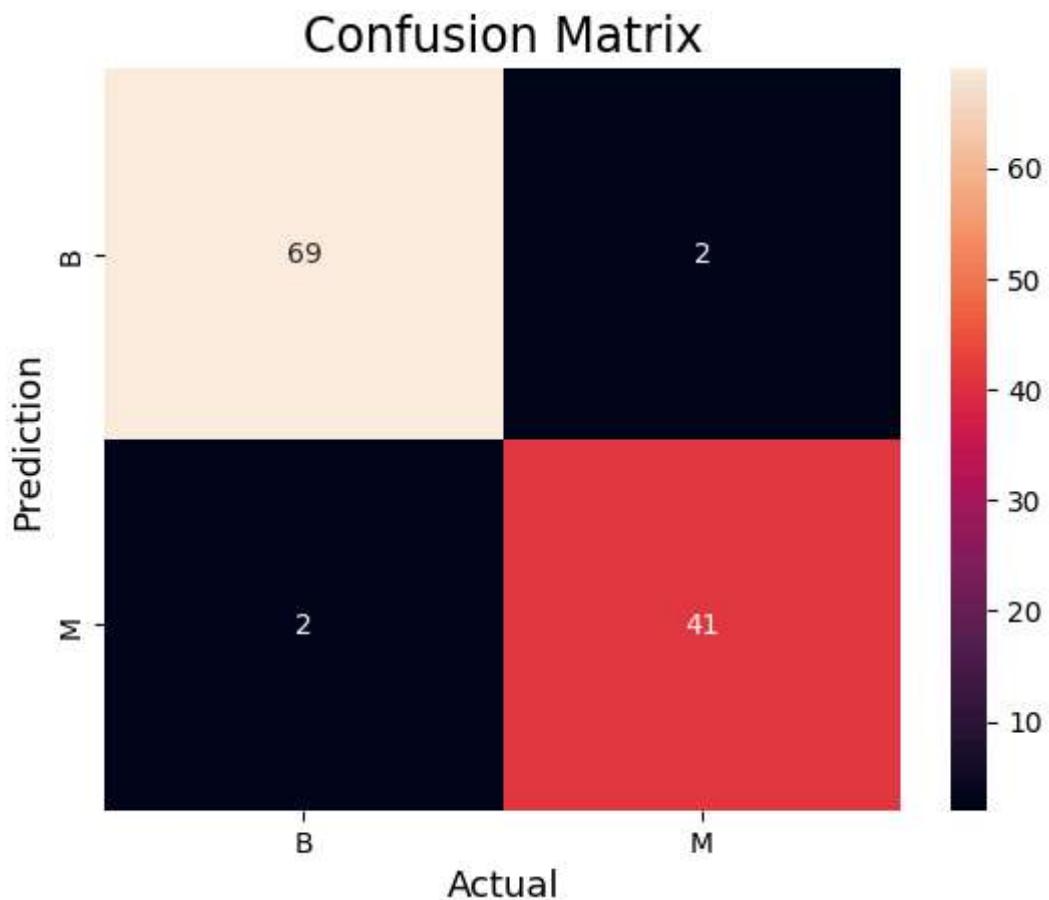
```
{'Logistic Regression': array([0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0,
0, 1, 0, 0,
1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0,
1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1,
1, 0, 1, 1]),
'KNN': array([0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1,
0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1,
1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1,
0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0,
1, 0, 0, 1]),
'Random Forest': array([0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1,
0, 0,
1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1,
1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,
1, 0, 0, 1]),
'Decision Tree': array([0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1,
0, 0,
1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1,
1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,
1, 0, 0, 1]),
'SVM': array([0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,
1, 0, 1, 1]),
'Naive Bayes': array([0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
1, 0, 0, 1])}
```

In [210...]

```
actual      = np.array(y_test)
predicted  = np.array(y_prediction['Logistic Regression'])
cm         = confusion_matrix(actual,predicted)
```

In [211...]

```
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['B','M'],
            yticklabels=['B','M'])
plt.ylabel('Prediction', fontsize=13)
plt.xlabel('Actual', fontsize=13)
plt.title('Confusion Matrix', fontsize=17)
plt.show()
```



5 Pipeline

```
In [212]: # pipeline making
```

```
In [213]: df1
```

Out[213]:

	radius_mean	texture_mean	smoothness_mean	compactness_mean	concave_points_mean	symmetry_mean
0	17.99	10.38	0.118400	0.22862	0.147100	
1	20.57	17.77	0.084740	0.07864	0.070170	
2	19.69	21.25	0.109600	0.15990	0.127900	
3	11.42	20.38	0.133695	0.22862	0.105200	
4	20.29	14.34	0.100300	0.13280	0.104300	
...
564	21.56	22.39	0.111000	0.11590	0.138900	
565	20.13	28.25	0.097800	0.10340	0.097910	
566	16.60	28.08	0.084550	0.10230	0.053020	
567	20.60	29.33	0.117800	0.22862	0.151970	
568	7.76	24.54	0.057975	0.04362	0.050092	

569 rows × 23 columns

In [214...]

df1['diagnosis'].value_counts()

Out[214]:

```
diagnosis
B    357
M    212
Name: count, dtype: int64
```

In [215...]

df1['diagnosis'] = np.where((df1['diagnosis'] == 'M'), 1, 0)

In [216...]

df1['diagnosis'].value_counts()

Out[216]:

```
diagnosis
0    357
1    212
Name: count, dtype: int64
```

In [217...]

df1

Out[217]:

	radius_mean	texture_mean	smoothness_mean	compactness_mean	concave_points_mean	symmetry_mean
0	17.99	10.38	0.118400	0.22862	0.147100	
1	20.57	17.77	0.084740	0.07864	0.070170	
2	19.69	21.25	0.109600	0.15990	0.127900	
3	11.42	20.38	0.133695	0.22862	0.105200	
4	20.29	14.34	0.100300	0.13280	0.104300	
...
564	21.56	22.39	0.111000	0.11590	0.138900	
565	20.13	28.25	0.097800	0.10340	0.097910	
566	16.60	28.08	0.084550	0.10230	0.053020	
567	20.60	29.33	0.117800	0.22862	0.151970	
568	7.76	24.54	0.057975	0.04362	0.050092	

569 rows × 23 columns

In [218...]

`df1.to_csv("cleanedCancerDate.csv")`

In [219...]

`X = df1.drop("diagnosis", axis=1)`
`Y = df1["diagnosis"]`

In [220...]

`Y`

Out[220]:

```

0      1
1      1
2      1
3      1
4      1
 ..
564    1
565    1
566    1
567    1
568    0
Name: diagnosis, Length: 569, dtype: int64

```

In [221...]

`from sklearn.model_selection import train_test_split`

```
In [222]: X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_st
```

```
In [223]: from sklearn.pipeline import make_pipeline  
from sklearn.preprocessing import OneHotEncoder, StandardScaler  
from sklearn.compose import make_column_transformer
```

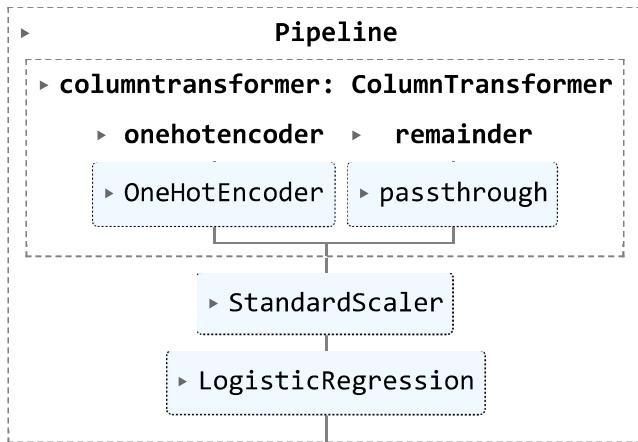
```
In [224]: column_trans = make_column_transformer(  
    (OneHotEncoder(sparse_output=False), ['N Stage', '6th Stage', 'differentiate']),  
    remainder='passthrough'  
)  
scaler=StandardScaler()
```

```
In [225]: from sklearn.linear_model import LogisticRegression  
clf = LogisticRegression()
```

```
In [226... pipe=make_pipeline(column_trans,scaler,clf)
```

```
In [227]: pipe.fit(X_train,Y_train)
```

Out[227]: ▶ Pipeline



```
In [228]: y_pred_dt=pipe.predict(X_test)
```

```
In [229]: y_pred_dt
```

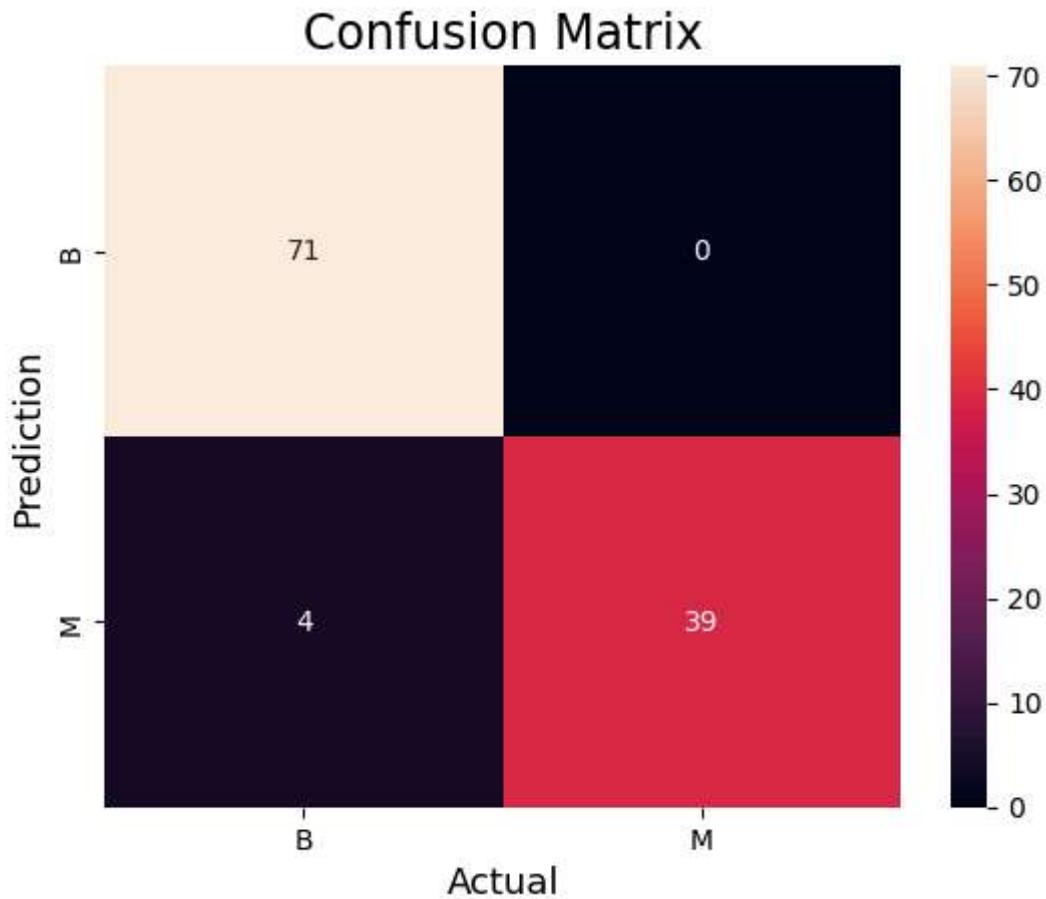
```
In [230]: from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score  
  
accuracy = accuracy_score(Y_test, y_pred_dt)  
  
# Calculate precision  
precision = precision_score(Y_test, y_pred_dt)  
  
# Calculate recall  
recall = recall_score(Y_test, y_pred_dt)  
  
# Calculate F1 score
```

```
f1 = f1_score(Y_test, y_pred_dt)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Accuracy: 0.9649122807017544
Precision: 1.0
Recall: 0.9069767441860465
F1 Score: 0.951219512195122
```

```
In [232...]: cm = confusion_matrix(Y_test,y_pred_dt)
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['B','M'],
            yticklabels=['B','M'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()
```



```
In [231...]: import pickle
pickle.dump(pipe, open('LogisticRegression_cancer.pkl','wb'))
```