```
import pandas as pd


df = pd.read_csv('wdbc.data', header=None, names=column_names)

base_features = [
    'radius', 'texture', 'perimeter', 'area', 'smoothness',
    'compactness', 'concavity', 'concave_points', 'symmetry', 'fractal_dimen
]

column_names = ['ID', 'diagnosis']

for feature in base_features:
    column_names.append(f'{feature}_mean')

for feature in base_features:
    column_names.append(f'{feature}_se')

for feature in base_features:
    column_names.append(f'{feature}_worst')
```

```
print(df.head())
```

```
         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

   ...  radius_worst  texture_worst  perimeter_worst  area_worst  \
0  ...         25.38          17.33           184.60      2019.0
1  ...         24.99          23.41           158.80      1956.0
2  ...         23.57          25.53           152.50      1709.0
3  ...         14.91          26.50            98.87       567.7
4  ...         22.54          16.67           152.20      1575.0

   smoothness_worst  compactness_worst  concavity_worst  concave_points_worst
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   symmetry_worst  fractal_dimension_worst
0          0.4601                  0.11890
```

```
1          0.2750                  0.08902
2          0.3613                  0.08758
3          0.6638                  0.17300
4          0.2364                  0.07678

[5 rows x 32 columns]
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   ID                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 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           569 non-null    float64
 9   concave_points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   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             569 non-null    float64
 19  concave_points_se        569 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  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave_points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

```
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
dtype: int64
```

```
from sklearn.model_selection import train_test_split

X = df.drop(['ID', 'diagnosis'], axis=1)
y = df['diagnosis']

y = y.map({'M': 1, 'B': 0})

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ran
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Logistic Regression:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_scor

log_reg_model = LogisticRegression(random_state=42, solver='liblinear')

log_reg_model.fit(X_train_scaled, y_train)

y_train_pred = log_reg_model.predict(X_train_scaled)
y_test_pred = log_reg_model.predict(X_test_scaled)
```

```
y_test_pred = log_reg_model.predict(X_test_scaled)

training_accuracy = accuracy_score(y_train, y_train_pred)
training_precision = precision_score(y_train, y_train_pred)
training_recall = recall_score(y_train, y_train_pred)
training_f1 = f1_score(y_train, y_train_pred)
training_error = 1 - training_accuracy

print(f"Training Accuracy: {training_accuracy:.4f}")
print(f"Training Precision: {training_precision:.4f}")
print(f"Training Recall: {training_recall:.4f}")
print(f"Training F1-score: {training_f1:.4f}")
print(f"Training Error: {training_error:.4f}")

print("\n" + "-"*30 + "\n")

test_accuracy = accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred)
test_f1 = f1_score(y_test, y_test_pred)
test_error = 1 - test_accuracy

print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Test Precision: {test_precision:.4f}")
print(f"Test Recall: {test_recall:.4f}")
print(f"Test F1-score: {test_f1:.4f}")
print(f"Test Error: {test_error:.4f}")

print("\n" + "-"*30 + "\n")
```

```
Training Accuracy: 0.9874
Training Precision: 0.9932
Training Recall: 0.9732
Training F1-score: 0.9831
Training Error: 0.0126

------------------------------

Test Accuracy: 0.9825
Test Precision: 0.9688
Test Recall: 0.9841
Test F1-score: 0.9764
Test Error: 0.0175

------------------------------
```

Decision Tree Classifier:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_scor

dt_model = DecisionTreeClassifier(random_state=42)

dt_model.fit(X_train_scaled, y_train)
```

```
y_train_pred_dt = dt_model.predict(X_train_scaled)
y_test_pred_dt = dt_model.predict(X_test_scaled)

training_accuracy_dt = accuracy_score(y_train, y_train_pred_dt)
training_precision_dt = precision_score(y_train, y_train_pred_dt)
training_recall_dt = recall_score(y_train, y_train_pred_dt)
training_f1_dt = f1_score(y_train, y_train_pred_dt)
training_error_dt = 1 - training_accuracy_dt

print(f"Training Accuracy: {training_accuracy_dt:.4f}")
print(f"Training Precision: {training_precision_dt:.4f}")
print(f"Training Recall: {training_recall_dt:.4f}")
print(f"Training F1-score: {training_f1_dt:.4f}")
print(f"Training Error: {training_error_dt:.4f}")

print("\n" + "-"*30 + "\n")

test_accuracy_dt = accuracy_score(y_test, y_test_pred_dt)
test_precision_dt = precision_score(y_test, y_test_pred_dt)
test_recall_dt = recall_score(y_test, y_test_pred_dt)
test_f1_dt = f1_score(y_test, y_test_pred_dt)
test_error_dt = 1 - test_accuracy_dt

print(f"Test Accuracy: {test_accuracy_dt:.4f}")
print(f"Test Precision: {test_precision_dt:.4f}")
print(f"Test Recall: {test_recall_dt:.4f}")
print(f"Test F1-score: {test_f1_dt:.4f}")
print(f"Test Error: {test_error_dt:.4f}")

print("\n" + "-"*30 + "\n")
```

```
Training Accuracy: 1.0000
Training Precision: 1.0000
Training Recall: 1.0000
Training F1-score: 1.0000
Training Error: 0.0000

------------------------------

Test Accuracy: 0.9415
Test Precision: 0.8955
Test Recall: 0.9524
Test F1-score: 0.9231
Test Error: 0.0585

------------------------------
```

Logistic Regression Model

- Training Performance:
    - Accuracy: 0.9874

    - Precision: 0.9932
    - Recall: 0.9732
    - F1-score: 0.9831
    - Error: 0.0126
- Test Performance:
    - Accuracy: 0.9825
    - Precision: 0.9688
    - Recall: 0.9841
    - F1-score: 0.9764
    - Error: 0.0175

Decision Tree Classifier

- Training Performance:
    - Accuracy: 1.0000
    - Precision: 1.0000
    - Recall: 1.0000
    - F1-score: 1.0000
    - Error: 0.0000
- Test Performance:
    - Accuracy: 0.9415
    - Precision: 0.8955
    - Recall: 0.9524
    - F1-score: 0.9231
    - Error: 0.0585

Analysis:

- Logistic Regression: The model has good accuracy and precision scores in both training and testing dataset. ~1% error in the calculation. Hence no problem of overfitting or underfitting.
- Decision Tree: The model has good scores in training dataset but drastic performance dip in testing dataset. This is due to overfitting of features in decision tree mode.
- Relavant issues in ML:

1. Scaling: The issue occurs when the skewness of the dataset is vast, so we scale down the values for the ease of reading the data
2. High Correlation: Due to high correlation of data, the logistic regression model fails, but decisioin tree model is able to perform reasonably well under such conditions but it also may suffer from this issue.