ISDS 574 - GROUP 6

BREAST CANCER

Craig Joseph Albuquerque
Rucha Prasad Nilangekar
Ashutosh Anilkumar Dubey
Nanditha Kolakaluru
Vinh Vuong
Thuy Nguyen

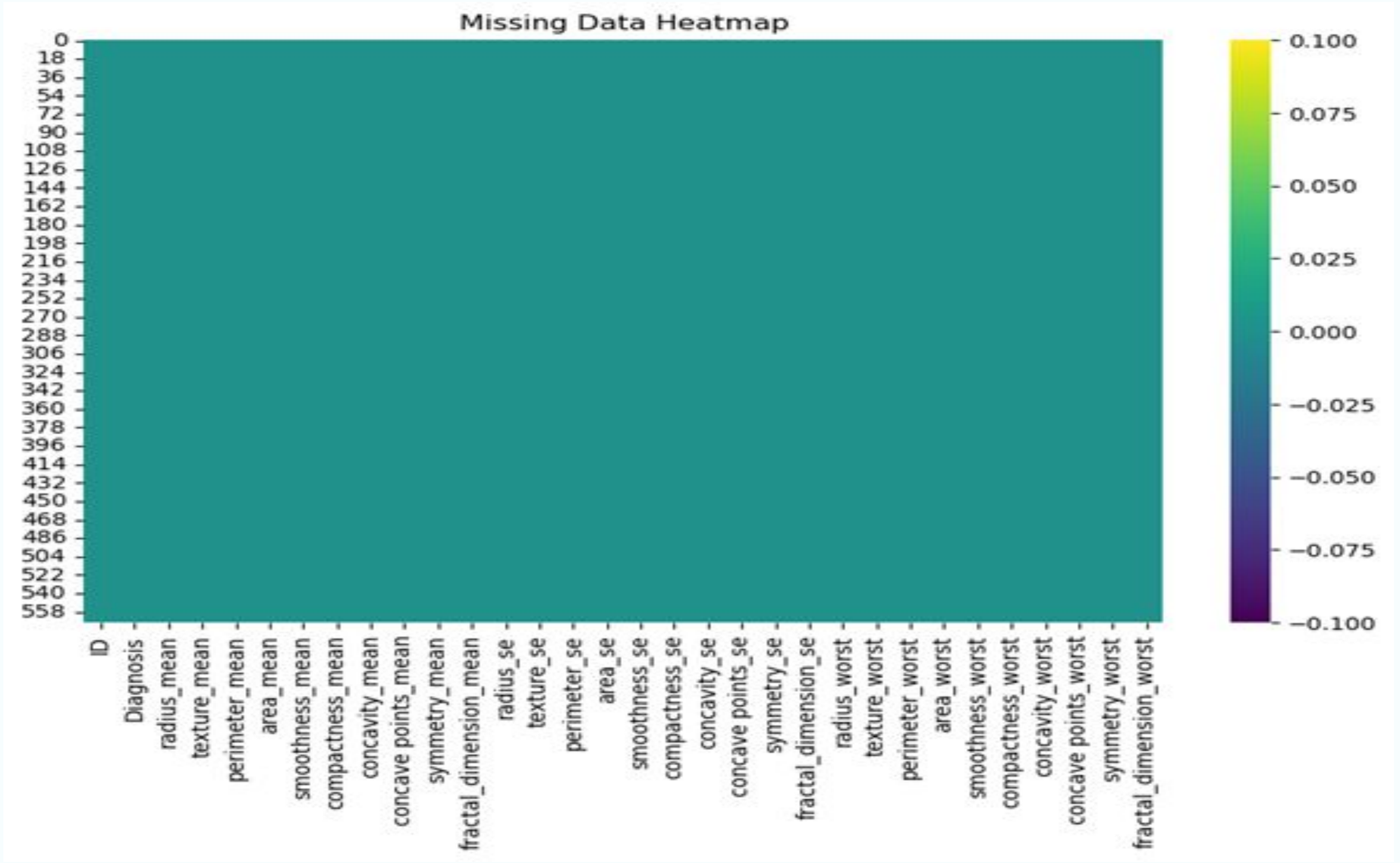
ABOUT THE DATASET

➤ ***Goal: Improve diagnosis accuracy and objectivity when using non-invasive breast cancer diagnosis***

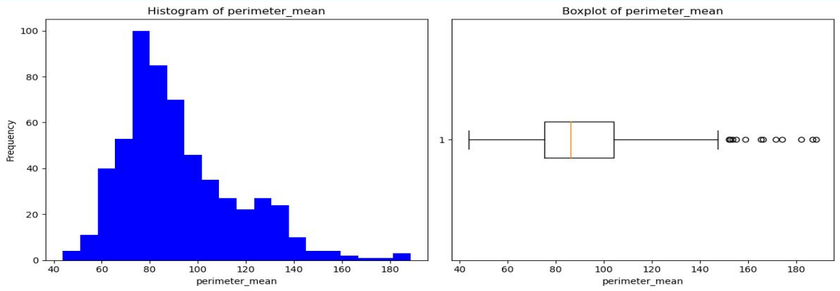
- **Method**: Analyzed nuclear features of cell nuclei from images of the tissue sample collected via Fine Needle Aspiration (FNA)
    - 30 features
        - 10 categories for each cell nucleus
        - 3 measured indicators for each category
            - mean value
            - standard error
            - extreme value (largest or worst value depending on the category)
    - 569 images
    - Outcome variable = Diagnosis
        - B = benign
        - M = malignant

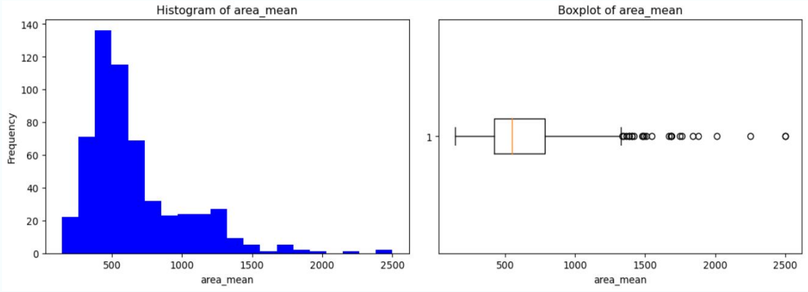
Cell Nuclei Categories
radius
texture
perimeter
area
smoothness
compactness
concavity
concave points
symmetry
fractal dimension

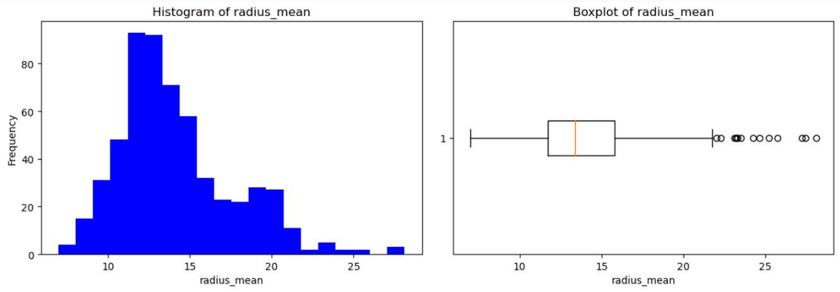
DATA PREPROCESSING - STEP 1

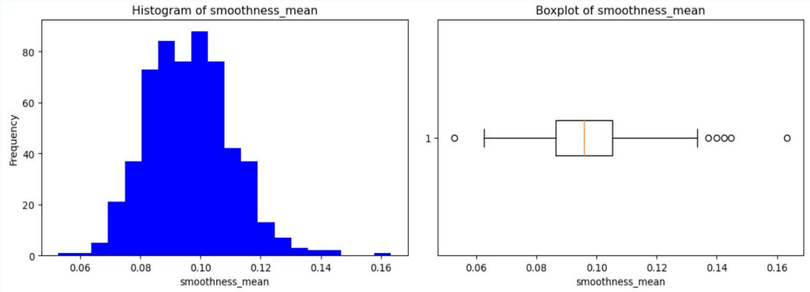


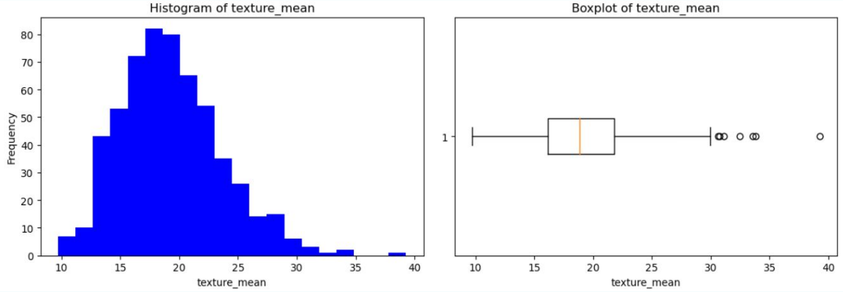
DATA PREPROCESSING - STEP 2

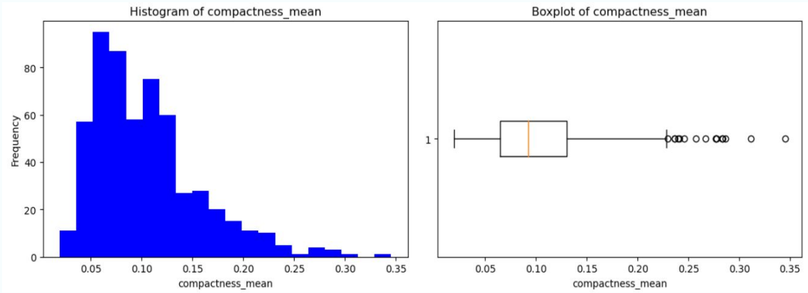












- All 30 independent variables have outliers

DATA PREPROCESSING - STEP 3 & 4

STEP 3: Check Frequency Table For Categorical Variables

```
frequency_table = dat2['Diagnosis'].value_counts()

print(frequency_table)
```

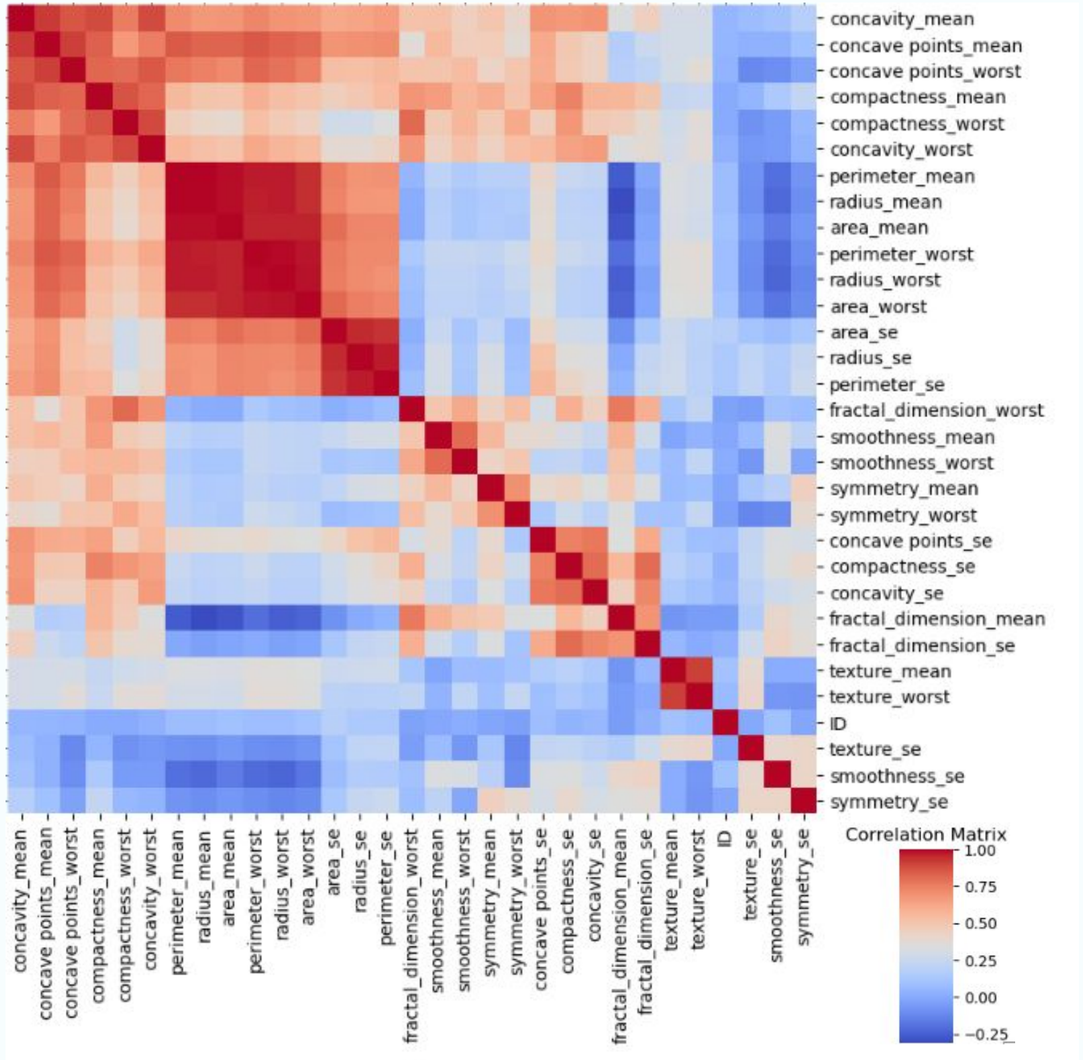
```
Diagnosis
B    357
M    212
Name: count, dtype: int64
```

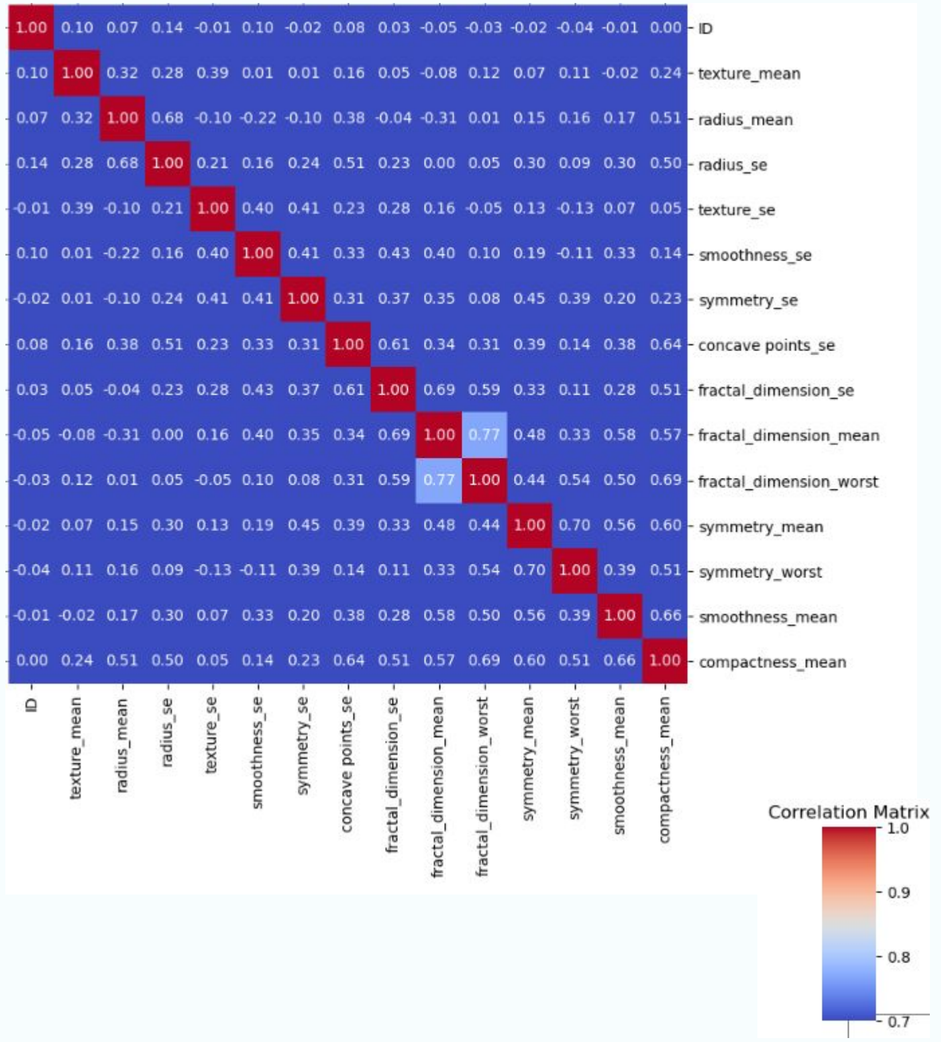
STEP 4: Create Dummy Variables for Categorical Variables

No categorical predictors in our dataset.

DATA PREPROCESSING - STEP 5

DROPPED VARIABLES





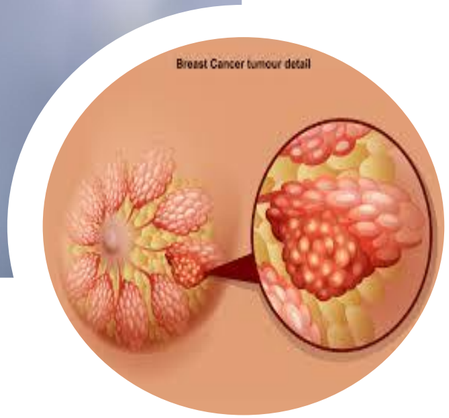
CLEANED DATASET

- 14 independent variables

	smoothness_se	fractal_dimension_worst
texture_mean	symmetry_se	symmetry_mean
radius_mean	concave points_se	symmetry_worst
radius_se	fractal_dimension_se	smoothness_mean
texture_se	fractal_dimension_mean	compactness_mean

- 569 observations

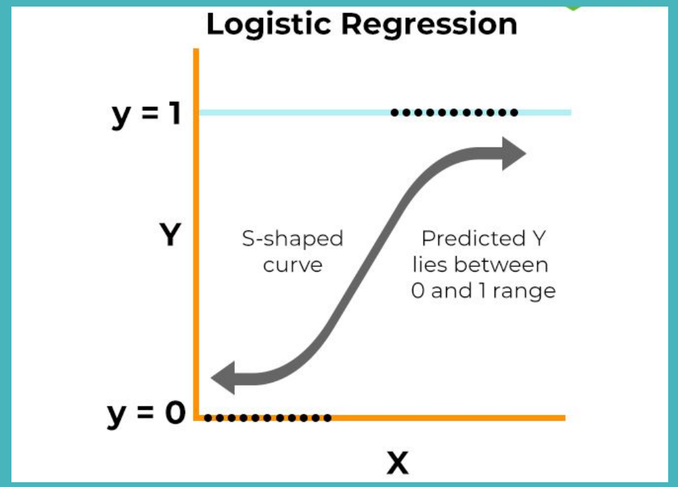
CUTOFF SELECTION



Test two level of prediction:

- **Lower** cutoff ➠ doubtful if the tumor is malignant and need further investigation
- **Higher** cutoff ➠ higher confidence in our diagnosis
- **Sensitivity** vs **specificity**

LOGISTIC REGRESSION



a.k.a. **Log Odds**

or **Logit**

Intercept

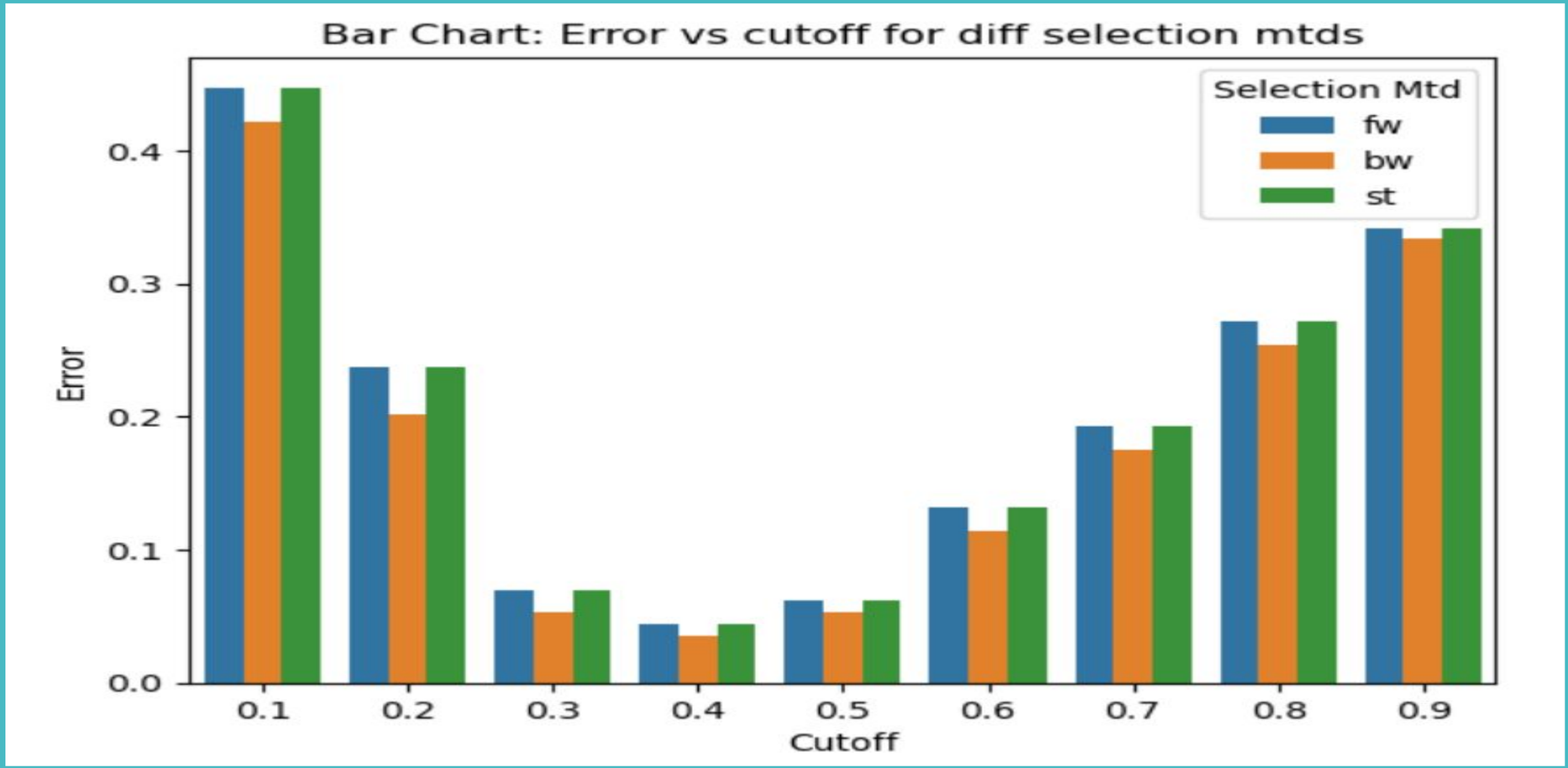
$$\log\left(\frac{P}{1-P}\right) = \beta_0 + \beta_1 X$$

Backward Selection

- radius_mean
- texture_mean
- smoothness_mean
- fractal_dimension_mean
- concave points_se
- symmetry_se
- fractal_dimension_se
- symmetry_worst
- fractal_dimension_worst
- texture_se
- smoothness_se
- *compactness_mean*
- symmetry_mean
- radius_se

Forward /Stepwise Selection

LOGISTIC REGRESSION



LOGISTIC REGRESSION

Selection Mtd	cutoff	error	Sensitivity	Specificity	Accuracy
bw	0.3	0.052632	0.928571	0.958333	0.947368
bw	0.4	0.035088	0.928571	0.986111	0.964912
★ st	0.3	0.070175	0.952381	0.916667	0.929825
st	0.4	0.043860	0.904762	0.986111	0.956140

➠ We select 0.3 cutoff due to higher Sensitivity.

$$\text{Sensitivity} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

LOGISTIC REGRESSION: ODDS RATIO

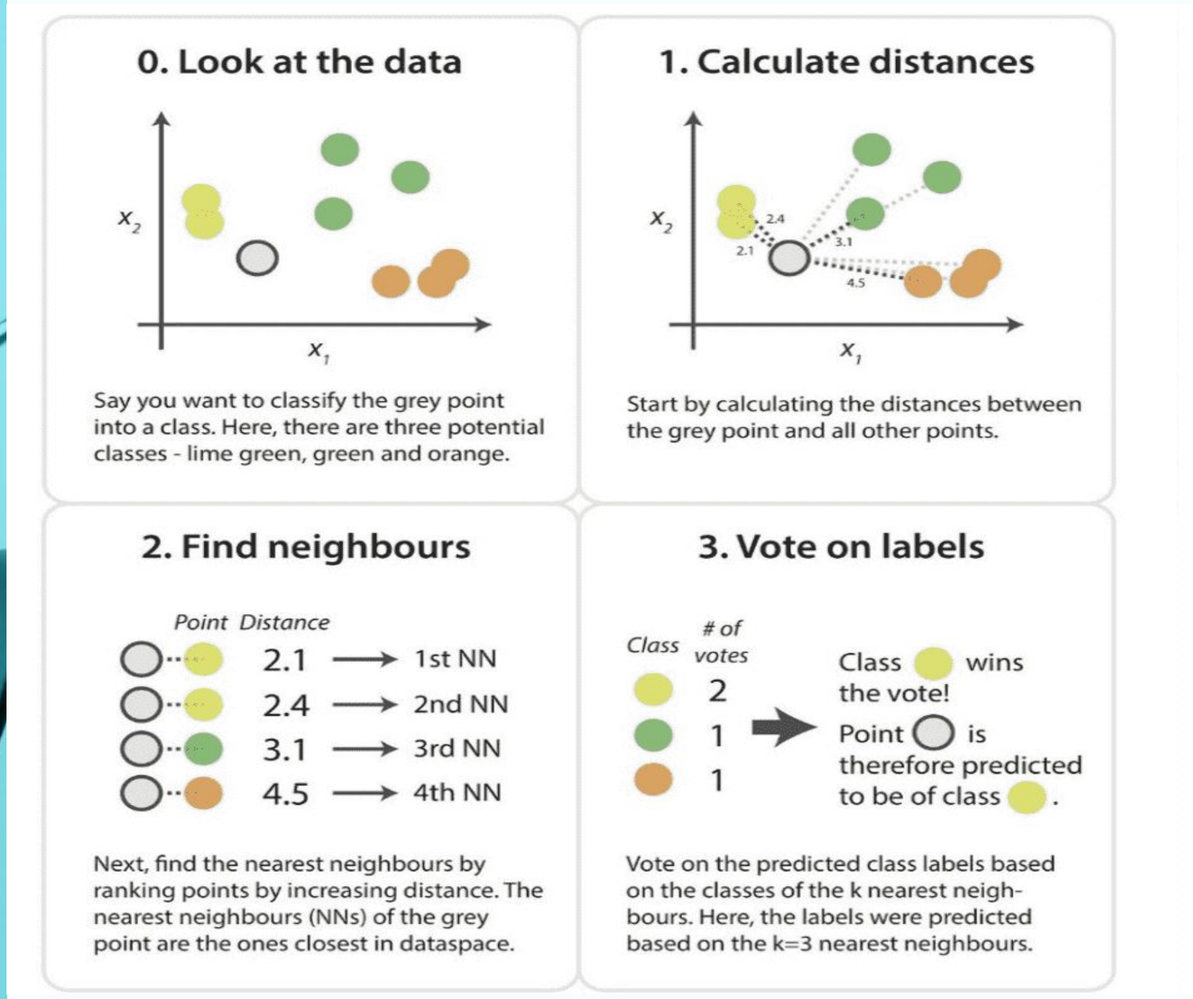
Forward/Stepwise Selection

```
1  radius_mean has a odds ratio of 1004.2
2  texture_mean has a odds ratio of 21.16
3  smoothness_mean has a odds ratio of 10.68
4  compactness_mean has a odds ratio of 0.41
5  symmetry_mean has a odds ratio of 10.3
6  fractal_dimension_mean has a odds ratio of 0.45
7  radius_se has a odds ratio of 0.69
8  texture_se has a odds ratio of 21.02
9  smoothness_se has a odds ratio of 10.02
```

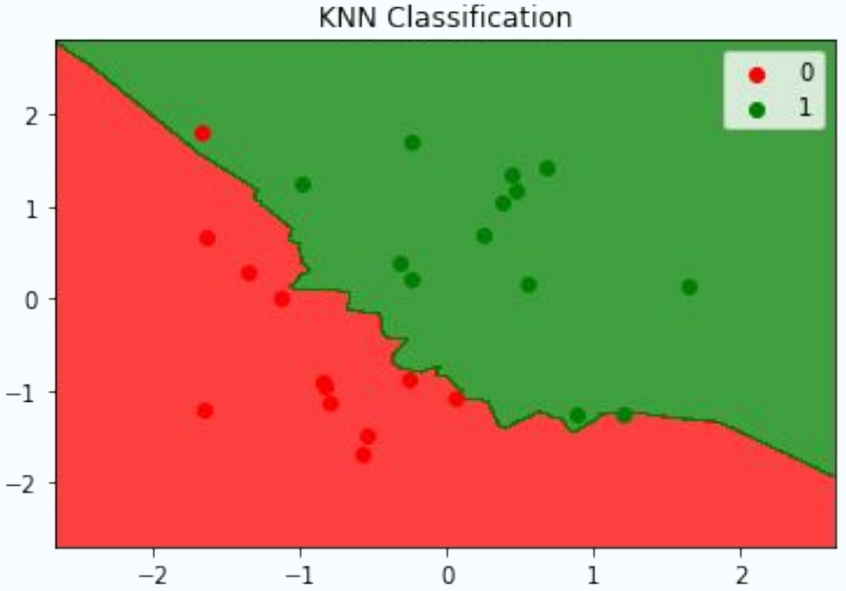
Backward Selection

```
1  radius_mean has a odds ratio of 599.95
2  texture_mean has a odds ratio of 21.66
3  smoothness_mean has a odds ratio of 7.86
4  compactness_mean has a odds ratio of 2.92
5  symmetry_mean has a odds ratio of 0.39
6  fractal_dimension_mean has a odds ratio of 14.19
7  radius_se has a odds ratio of 0.61
8  texture_se has a odds ratio of 7.34
9  smoothness_se has a odds ratio of 0.35
10  concave points_se has a odds ratio of 0.62
11  symmetry_se has a odds ratio of 16.02
12  fractal_dimension_se has a odds ratio of 9.72
```

K-Nearest Neighbors



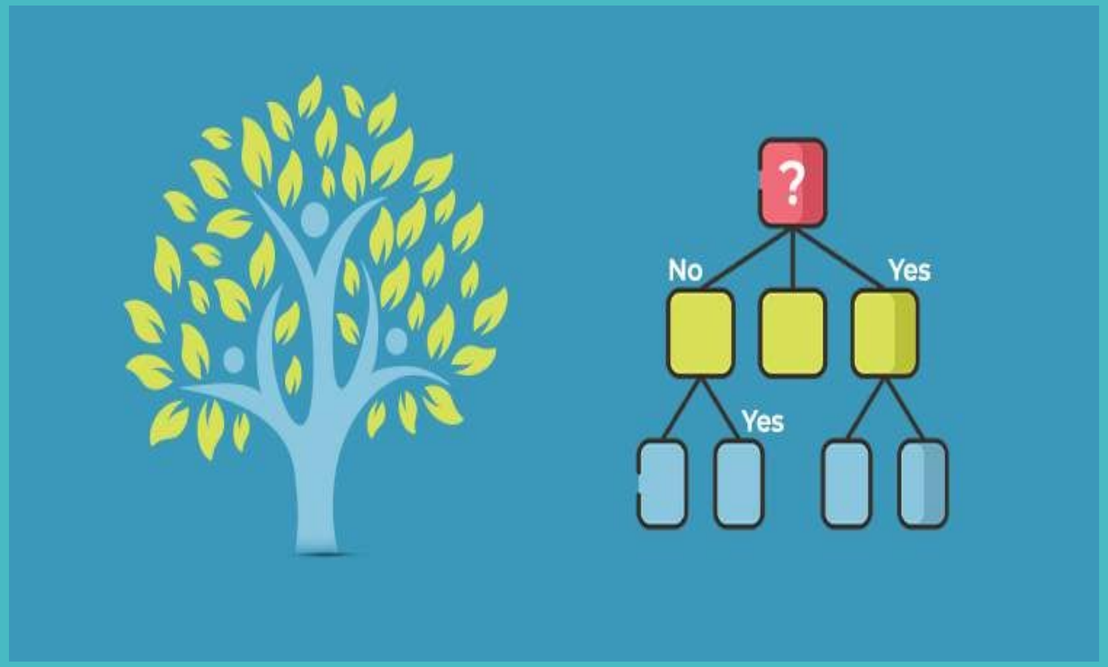
- 14 variables Selected for KNN



K-Nearest Neighbors

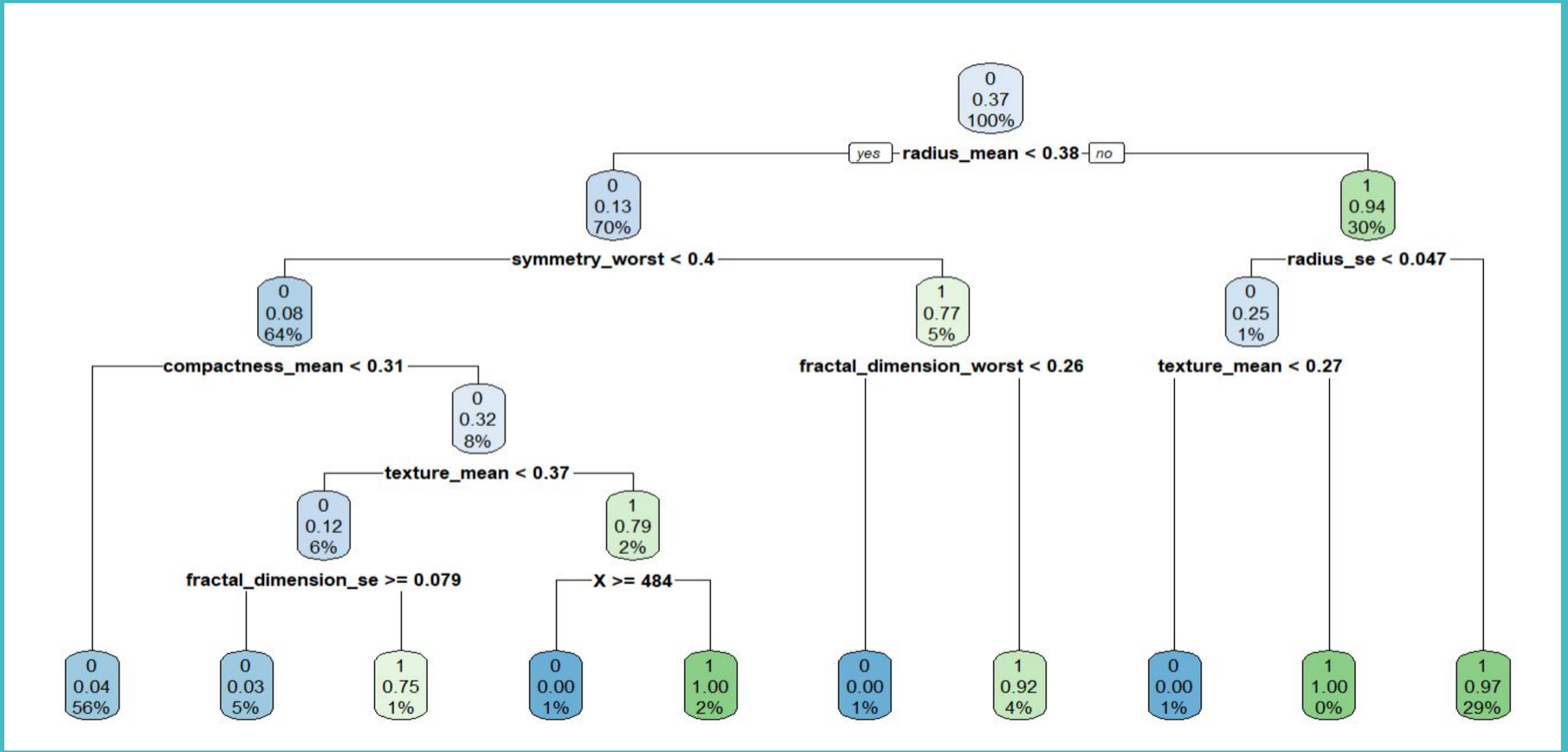
	Best_k	Best_Accuracy	Best_Sensitivity	Best_Specificity	least_error
0.4	23	0.958042	0.909091	0.988636	0.041958
0.5	9	0.958042	0.890909	1.0	0.041958
0.6	9	0.951049	0.872727	1.0	0.048951
0.7	7	0.944056	0.854545	1.0	0.055944
0.9	1	0.916084	0.909091	0.920455	0.083916

CLASSIFICATION AND REGRESSION TREE



- rpart package is used to implement the CART algorithm.
- For classification tasks CART creates a decision tree which is built to maximize the purity of the classes in each leaf. Using Gini impurity or cross-entropy.

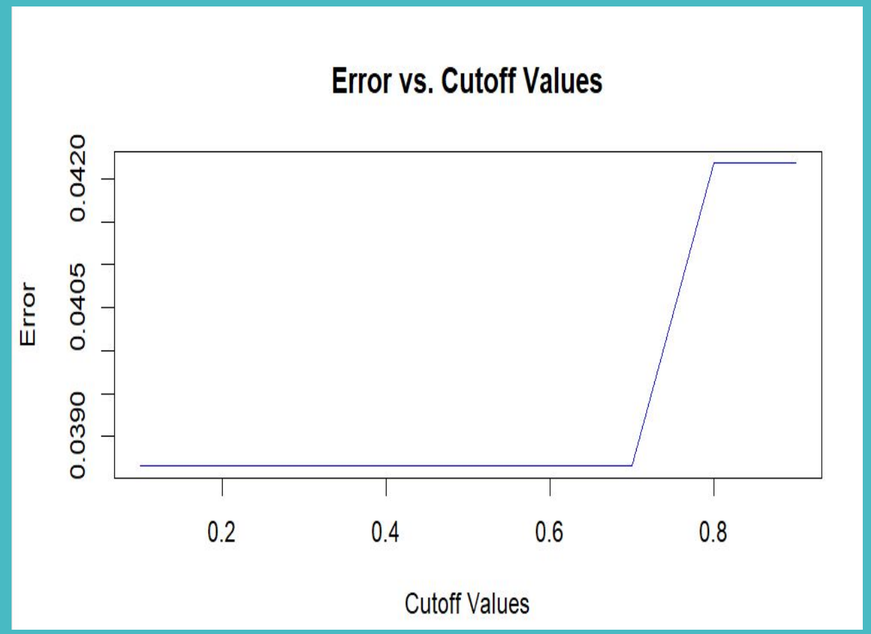
CLASSIFICATION AND REGRESSION TREE



➠ Both Min Error and Best-Pruned tree have similar results.

CLASSIFICATION AND REGRESSION TREE

	Cutoff	Error	Sensitivity	Specificity	FPR	FNR
1	0.1	0.10193322	0.9386792	0.8739496	0.12605042	0.06132075
2	0.2	0.05799649	0.9198113	0.9551821	0.04481793	0.08018868
3	0.3	0.05096661	0.9103774	0.9719888	0.02801120	0.08962264
4	0.4	0.05096661	0.9103774	0.9719888	0.02801120	0.08962264
5	0.5	0.05096661	0.9103774	0.9719888	0.02801120	0.08962264
6	0.6	0.05096661	0.9103774	0.9719888	0.02801120	0.08962264
7	0.7	0.05096661	0.9103774	0.9719888	0.02801120	0.08962264
8	0.8	0.06502636	0.8584906	0.9803922	0.01960784	0.14150943
9	0.9	0.06502636	0.8584906	0.9803922	0.01960784	0.14150943



KEY FINDINGS

	Methods				
	Logistic Regression			kNN	CART
	Forward	Backward	Stepwise		
Cutoff	0.3	0.3	0.3	0.4	0.3
Validation Error (%)	7.02	5.26	7.02	4.19	5.09
Sensitivity (%)	95.24	92.86	95.24	90.91	91.04
Specificity (%)	91.67	95.83	91.67	98.86	97.2

KEY FINDINGS

Variables contribution to the outcome:

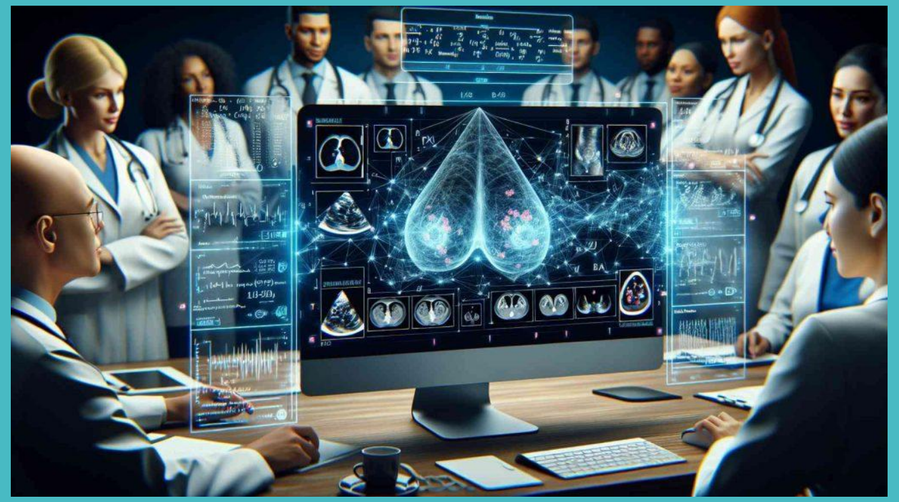
- radius_mean
- texture_se
- texture_mean
- smoothness_mean
- symmetry_mean
- smoothness_se
- radius_se
- compactness_mean
- fractal_dimension_mean

COMPARISON WITH PRIOR RESEARCH

	Our Findings	Breast cancer diagnosis based on feature extraction using a hybrid of K-means and support vector machine algorithms (Zheng et al., 2014)	Breast Cancer Diagnosis Using an Unsupervised Feature Extraction Algorithm Based on Deep Learning (Xiao et al., 2018)
Method	Logistic Regression - Stepwise	Hybrid of K-means and support vector machine algorithms (K-SVM)	A deep learning based unsupervised feature extraction algorithm (SAE-SVM) ● the stacked auto-encoders ● a support vector machine model
Feature Dimension	9	6	15
Accuracy (%)	92.98	97.38	98.25

IMPLICATIONS & APPLICATIONS

- Accessible Diagnosis
- Reduce healthcare professionals workload
- Potentially improve overall diagnostic accuracy and early detection.
- Practical method for assisting doctors
- Influence healthcare policy and funding
- Integration with other technologies like AI



LIMITATION & FUTURE WORKS

LIMITATIONS:

- **LOGISTIC REGRESSION:** linear decision boundary, multicollinearity, outlier sensitivity.
- **KNN:** computational complexity, optimal value of K, no model training
- **CART:** overfitting, high Variance, instability

FUTURE WORKS:

- Improve existing diagnostic and prognostic systems
- Generalize the approach

THANK YOU!

CODE - LOGISTIC REGRESSION

Loop for running multiple features, cutoff

```
for features,feature_name in zip(selected_features,Selected_features_names):

    best_ct = 0
    best_accuracy = 0
    best_spec = 0
    best_sen = 0
    error_min = 1000

    df1 = df[all_features]
    X = df1.iloc[:,features]
    y = df['Diagnosis']
    xtrain, xtest, ytrain, ytest = train_test_split(X,y, test_size=0.2, random_state=1)
    model = LogisticRegression( penalty= 'l2', max_iter = 10000,  verbose=0)
    model.fit(xtrain, ytrain)

    print(f'##########{features}#############')
    for ct in cts:
        print(f'----------------{ct}----------------')
        #predict proba of model
        y_pred_prob = model.predict_proba(xtest)

        ypred = []

        #evaluvate the proba
        for i in range(0, (len( y_pred_prob))):
            # print(y_pred_prob[i][1])
            if y_pred_prob[i][1] > ct:
                result = 1
            else:
                result = 0
            ypred.append(result)
            # print(result)

        accuracy = accuracy_score(ytest, ypred)
        conf_matrix = confusion_matrix(ytest, ypred)
        # Extract TP, TN, FP, FN
        TN = conf_matrix[0, 0]
        FP = conf_matrix[0, 1]
        FN = conf_matrix[1, 0]
        TP = conf_matrix[1, 1]
```

```
        sensitivity = TP / (TP + FN)
        specificity = TN / (TN + FP)
        tpr = TP / (TP + FN)
        fpr = FP / (FP + TN)
        error = (FP + FN) / (TP + TN + FP + FN)

        #----In the confusion matrix obtained using confusion_matrix(ytest, ypred),
        #      the rows represent the actual classes, and the columns represent the
        #      predicted classes. The confusion matrix is a 2x2 matrix for binary
        #      classification problems. Let's break down the elements:

        # Extract coefficients
        coefficients = model.coef_[0]

        # Calculate odds ratio
        odds_ratio = np.exp(coefficients)
        odds_ratio = [round(num, 2) for num in odds_ratio]

        for i in range(0,len(features)):
            print(f'{i+1}  {all_features[i]} has a odds ratio of {odds_ratio[i]}')

        if error < error_min:
            error_min = error
            best_accuracy = accuracy
            best_ct= ct
            best_spec = specificity
            best_sen = sensitivity

        feature_selection_list.append(feature_name)
        ct_list.append(ct)
        error_list.append(error)
        sensitivity_list.append(sensitivity)
        specificity_list.append(specificity)
        accuracy_list.append(accuracy)

    result_df.loc[feature_name, 'Best_cutoff'] = best_ct
    result_df.loc[feature_name, 'least_error'] = error_min
    result_df.loc[feature_name, 'Best_Sensitivity'] = best_sen
    result_df.loc[feature_name, 'Best_Specificity'] = best_spec
    result_df.loc[feature_name, 'Best_Accuracy'] = best_accuracy

result_df
ind_results_df = pd.DataFrame({'Selection Mtd':feature_selection_list,'cutoff':ct_list,'error':error_list,
                               'Sensitivity':sensitivity_list,'Specificity':specificity_list, 'Accuracy':accuracy_list})
```

CODE - LOGISTIC REGRESSION

Variable Selection

```
model = LogisticRegression(max_iter = 10000)

# Initialize the forward feature selector
sfs_fw = SequentialFeatureSelector(model, forward=True, k_features='best', scoring='accuracy', cv=5, verbose = 0)
sfs_bk = SequentialFeatureSelector(model, forward=False, k_features='best', scoring='accuracy', cv=5, verbose = 0)
ffs = SequentialFeatureSelector(model, k_features='best', scoring='accuracy', cv=5)
# efs = ExhaustiveFeatureSelector(model, min_features=1, max_features=(len(df.columns) - 3),scoring='accuracy', cv=5)

# Fit the feature selector to your data
sfs_fw.fit(xtrain, ytrain)
sfs_bk.fit(xtrain, ytrain)
ffs.fit(xtrain, ytrain)
# efs.fit(xtrain, ytrain)

# Get the selected feature indices
selected_features_indices_sfs_fw = sfs_fw.k_feature_idx_
selected_features_indices_sfs_bk = sfs_bk.k_feature_idx_
selected_features_indices_ffs = ffs.k_feature_idx_
```

CODE - K NEAREST NEIGHBORS

K- Nearest Neighbors

```
[ ]  # Set seed for the built-in 'random' module
     random.seed(1)

     # Set seed for NumPy
     np.random.seed(1)

     Y = df_scaled['Diagnosis']
     X = df_scaled[['radius_mean', 'texture_mean', 'smoothness_mean', 'compactness_mean',
            'symmetry_mean', 'fractal_dimension_mean', 'radius_se', 'texture_se',
            'smoothness_se', 'concave points_se', 'symmetry_se',
            'fractal_dimension_se', 'symmetry_worst', 'fractal_dimension_worst']]
     X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=1)
```

```
[ ]  ############################KNN#######################################
     #Find best k for each ct

     cts = [0.4,0.5,0.6, 0.7,0.9]
     result_df = pd.DataFrame(columns=['Best_k','Best_Accuracy','Best_Sensitivity', 'Best_Specificity'], index=cts)
     max_k= round(np.sqrt(df_scaled.shape[0]))

     for ct in cts:
         print(f'##########CT: {ct}############')
         best_k = 0
         best_accuracy = 0
         best_ct = 0
         best_spec = 0
         best_sen = 0
         error_min = 10000
         for k in range(1, max_k, 2):  # iterate through different values of k
             # print(f'for k value {k}')
             knn = KNeighborsClassifier(n_neighbors=k)
             knn.fit(X_train,Y_train)
             ypred_knn_prob = knn.predict_proba(X_test)

             ypred_knn = []
             for i in range(0, len(ypred_knn_prob)):
                 if ypred_knn_prob[i][1] > ct:
                     result = 1
                 else:
                     result = 0
                 ypred_knn.append(result)

             accuracy = accuracy_score(Y_test, ypred_knn)
             conf_matrix = confusion_matrix(Y_test, ypred_knn)

             TN = conf_matrix[0, 0]
             FP = conf_matrix[0, 1]
             FN = conf_matrix[1, 0]
             TP = conf_matrix[1, 1]
```

```
             sensitivity = TP / (TP + FN)
             specificity = TN / (TN + FP)
             tpr = TP / (TP + FN)
             fpr = FP / (FP + TN)
             error = (FP + FN) / (TP + TN + FP + FN)

             if error < error_min:
                 error_min = error

                 best_accuracy = accuracy
                 best_k = k
                 # best_ct= ct
                 best_spec = specificity
                 best_sen = sensitivity
             print(f'error_min is {error_min} for {k}')

             # print(f'accuracy: {accuracy}')
             # print(f'confusion matrix: {conf_matrix}')
             #print(f'sensitivity: {sensitivity}')
             #print(f'specificity: {specificity}')
             # print(f'misclassification error: {error}')
             #print("")
         print(f'best_k: {best_k}')
         print("#######################")
         result_df.loc[ct, 'Best_k'] = best_k
         result_df.loc[ct, 'least_error'] = error_min
         result_df.loc[ct, 'Best_Sensitivity'] = best_sen
         result_df.loc[ct, 'Best_Specificity'] = best_spec
         result_df.loc[ct, 'Best_Accuracy'] = best_accuracy

print("KNN RESULTS:")
result_df
```

CODE - CART

```r
# FUNCTIONS
sen = function(ytest, ypred) {
  ind1 = which(ytest == 1)
  mean(ytest[ind1] == ypred[ind1])
}

spe = function(ytest, ypred) {
  ind1 = which(ytest == 0)
  mean(ytest[ind1] == ypred[ind1])
}

fpr = function(ytest, ypred) {
  ind1 = which(ytest == 0)
  mean(ytest[ind1] != ypred[ind1])
}

fnr = function(ytest, ypred) {
  ind1 = which(ytest == 1)
  mean(ytest[ind1] != ypred[ind1])
}
```

```r
rm(list=ls()); gc()
library(rpart); library(rpart.plot)

#Read in the data
dat = read.csv('wdbc_preped_scaled.csv', stringsAsFactors = T, head=T)
K = nrow(dat)


# Classification Tree with rpart
fit = rpart(Diagnosis ~ ., method="class", data=dat, minsplit=5 , cp = 0.001)
```

```r
# Best Pruned Tree
ind = which.min(fit$cptable[,"xerror"]) # xerror: cross-validation error
se1 = fit$cptable[ind,"xstd"]/sqrt(K) # 1 standard error
xer1 = min(fit$cptable[,"xerror"]) + se1 # targeted error: min + 1 SE
ind0 = which.min(abs(fit$cptable[1:ind,"xerror"] - xer1)) # select the tree g
pfit.bp = prune(fit, cp = fit$cptable[ind0,"CP"])
#pfit.bp = prune(fit, cp = 0.05)
rpart.plot(pfit.bp, main = 'Best Pruned Tree')

## How to predict? I am taking best pruned tree as an example.
yhat = predict(pfit.bp, dat, type = "class") # replace "dat" by validation da
err.bp = mean(yhat != dat$Diagnosis)

# if you want to use a cutoff not equal to 0.5
prob1 = predict(pfit.bp, dat, type = "prob")[,2]
pred.class = as.numeric(prob1 > .8)
#ytest = dat$Diagnosis # Be careful! Check the variable type of your outcome
#err.bp.newCut = mean(pred.class != ytest)
#err.bp.newCut
```

```r
# Minimum Error Tree Model
pfit.me = prune(fit, cp = fit$cptable[which.min(fit$cptable[,"xerror"]),"CP"])
rpart.plot(pfit.me, main = 'Min Error Tree')


## Min Error tree Prediction
yhat_me = predict(pfit.me, dat, type = "class") # replace "dat" by validation data if you have it
err.me = mean(yhat_me != dat$Diagnosis)

# if you want to use a cutoff not equal to 0.5 (Min error)
prob2 = predict(pfit.me, dat, type = "prob")[,2]
pred.class = as.numeric(prob1 > .4)
ytest_2 = as.numeric(dat$Diagnosis)
err.me.newCut = mean(pred.class != ytest)
```

CODE - CART

```r
# Set a sequence of cutoff values
cutoff_values <- seq(0.1, 0.9, by = 0.1)

# Initialize vectors to store results
err_vec <- numeric(length(cutoff_values))
sen_vec <- numeric(length(cutoff_values))
spe_vec <- numeric(length(cutoff_values))
fpr_vec <- numeric(length(cutoff_values))
fnr_vec <- numeric(length(cutoff_values))

# Loop over cutoff values for the
for (i in seq_along(cutoff_values)) {
  # Obtain predictions using the current cutoff
  pred.class <- as.numeric(prob2 > cutoff_values[i])

  # Calculate error and performance metrics
  err_vec[i] <- mean(pred.class != ytest_2)
  sen_vec[i] <- sen(ytest_2, pred.class)
  spe_vec[i] <- spe(ytest_2, pred.class)
  fpr_vec[i] <- fpr(ytest_2, pred.class)
  fnr_vec[i] <- fnr(ytest_2, pred.class)
}

# Create a data frame
result_df <- data.frame(
  Cutoff = cutoff_values,
  Error = err_vec,
  Sensitivity = sen_vec,
  Specificity = spe_vec,
  FPR = fpr_vec,
  FNR = fnr_vec
)

# Print the data frame
print(result_df)
```