Homework #2 Part 2

Neil Jiang, Hayley Huang, Foster Mosden, Icy Wang

Note: This is a team homework assignment. Discussing this homework with your classmates outside your MSBA team is a **violation** of the Honor Code.
If you borrow code from somewhere else, please add a comment in your code to make it clear what the source of the code is (e.g., a URL would sufficient). If you borrow code and you don't provide the source, it is a violation of the Honor Code.

Total grade: _____ out of ___70___ points

ATTENTION: HW2 has two parts. Please first complete the Quiz "HW2_Part1" on Canvas. Then, proceed with Part 2 in the following page. You will need to submit (a) a PDF file with your answers and screenshots of Python code snippets as well as Rapidminer repositories and (b) the Python code and Rapidminer repositories.

(70 points) [Mining publicly available data. Please implement the following models with both Rapidminer and Python]

Please use the dataset on breast cancer research from this link:
http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data [Note: Rapidminer can import .data files in the same way it can import .csv files. For Python please read the data *directly from the URL* **without** downloading the file on your local disk.**] The description of the data and attributes can be found at this link:**
http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.names and is also provided as in the appendix of this homework assignment.

Each record of the data set represents a different case of breast cancer. Each case is described with 30 real-valued attributes: attribute 1 represents case id, attributes 3-32 represent various physiological characteristics, and attribute 2 represents the type (benign B or malignant M) .

50 Points (Python):

a) (10 points) Load the data. Then, explore the data by reporting summary statistics and a correlation matrix. Show your code.

```
# load data from the provided URL
import pandas as pd
import numpy as np

df =
pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data', header=None)

df.columns = ['id', 'diagnosis', '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']
```

```
# explore the data by reporting summary statistics and a correlation matrix
df.describe()
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave_points_mean	symmetry_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave_points_worst	symmetry_worst	fractal_dimension_worst
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	...	16.269190	25.677223	107.261213	880.583128	0.132369	0.254265	0.272188	0.114606	0.290076	0.083946
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	...	4.833242	6.146258	33.602542	569.356993	0.022832	0.157336	0.208624	0.065732	0.061867	0.018061
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	...	7.930000	12.020000	50.410000	185.200000	0.071170	0.027290	0.000000	0.000000	0.156500	0.055040
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	...	13.010000	21.080000	84.110000	515.300000	0.116600	0.147200	0.114500	0.064930	0.250400	0.071460
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	...	14.970000	25.410000	97.660000	686.500000	0.131300	0.211900	0.226700	0.099930	0.282200	0.080040
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	...	18.790000	29.720000	125.400000	1084.000000	0.146000	0.339100	0.382900	0.161400	0.317900	0.092080
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	...	36.040000	49.540000	251.200000	4254.000000	0.222600	1.058000	1.252000	0.291000	0.663800	0.207500

8 rows × 31 columns

In order to build a model to predict diagnoses from the provided data set, we first need to examine the data. In order to do this we first print off basic summary statistics for the data set across all attributes. The data appears to be complete, and without any unexpected values. The code and sample output can be seen above.

Data ranges seem normal

```
#is a classifier problem
df.diagnosis.unique()
```

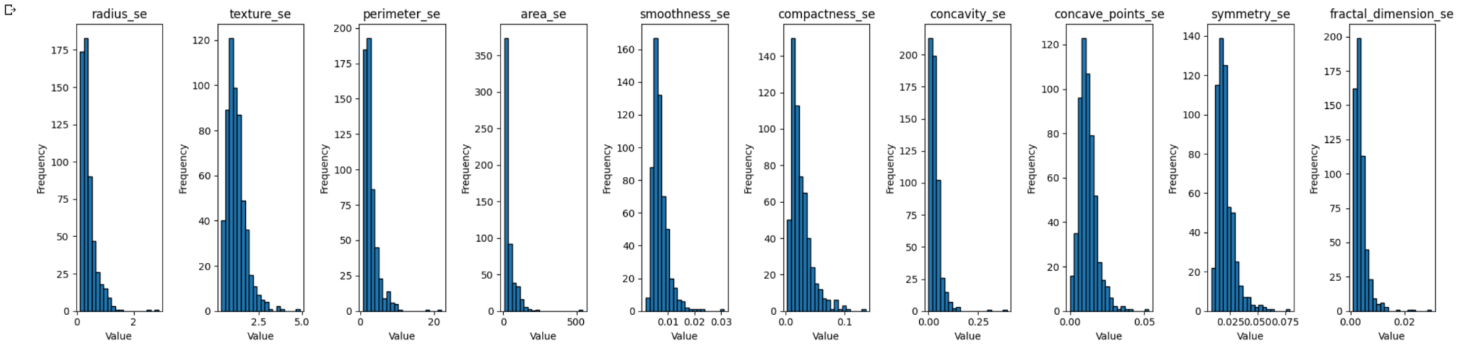
```
array(['M', 'B'], dtype=object)
```

Target variable is binary so classification problem.

```
[6] # Calculate skewness for each column
    skewness_scores = df.skew(numeric_only=True)

    # Print or return the skewness scores
    print(skewness_scores)
```

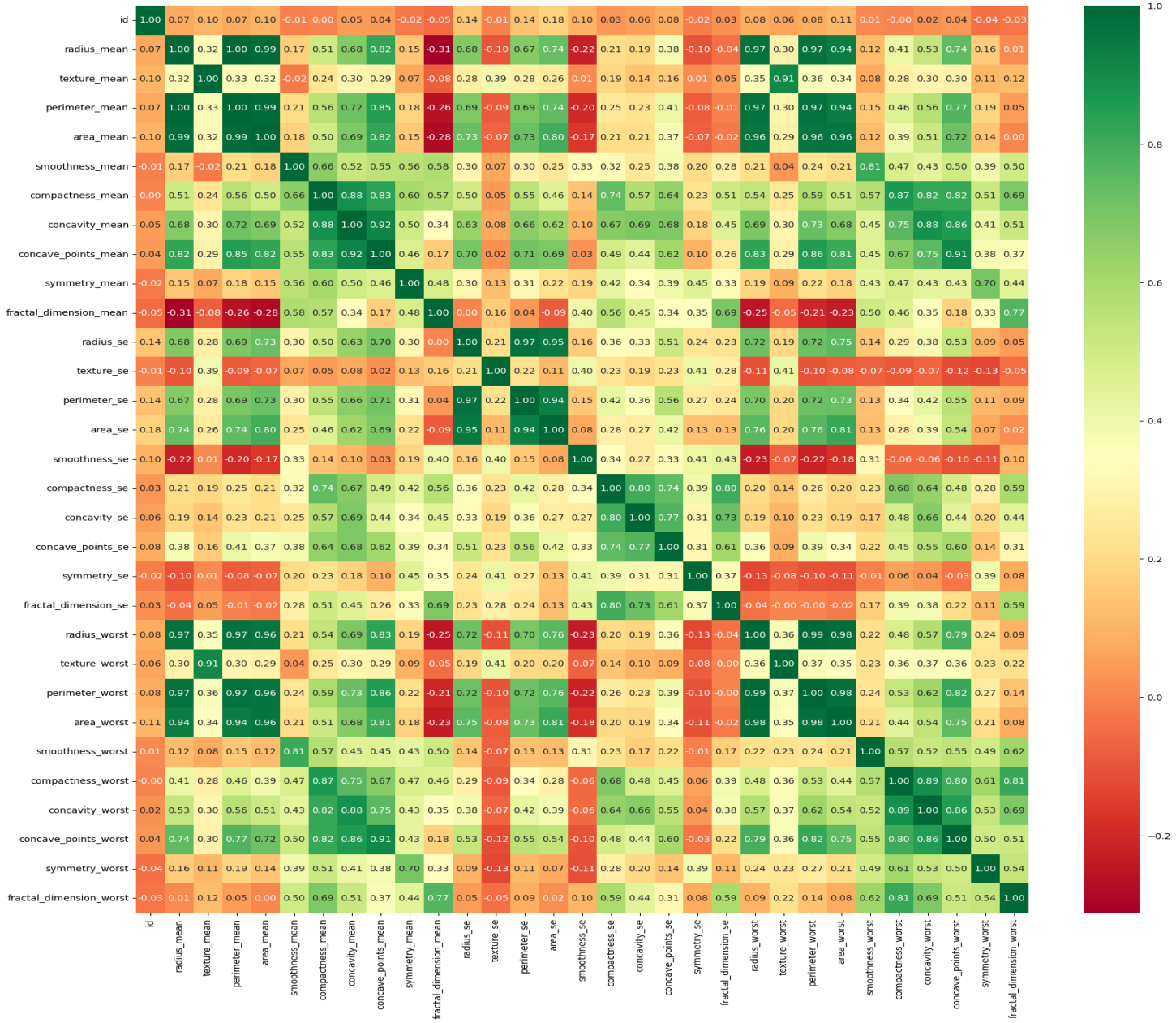
```
id                         6.473752
radius_mean                0.942380
texture_mean               0.650450
perimeter_mean             0.990650
area_mean                  1.645732
smoothness_mean            0.456324
compactness_mean           1.190123
concavity_mean             1.401180
concave_points_mean        1.171180
symmetry_mean              0.725609
fractal_dimension_mean     1.304489
radius_se                  3.088612
texture_se                 1.646444
perimeter_se               3.443615
area_se                    5.447186
smoothness_se              2.314450
compactness_se             1.902221
concavity_se               5.110463
concave_points_se          1.444678
symmetry_se                2.195133
fractal_dimension_se       3.923969
radius_worst               1.103115
texture_worst              0.498321
perimeter_worst            1.128164
area_worst                 1.859373
smoothness_worst           0.415426
compactness_worst          1.473555
concavity_worst            1.150237
concave_points_worst       0.492616
symmetry_worst             1.433928
fractal_dimension_worst    1.662579
dtype: float64
```

```
#check the SE variables which are highly skewed
import pandas as pd
import matplotlib.pyplot as plt

# Assuming df is your DataFrame
columns_to_check = ['radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se', 'compactness_se', 'concavity_se', 'concave_points_se', 'symmetry_se', 'fractal_dimension_se']

# Create a figure with subplots
fig, axes = plt.subplots(nrows=1, ncols=len(columns_to_check), figsize=(20, 5))

# Iterate through columns and plot histograms
for i, column in enumerate(columns_to_check):
    axes[i].hist(df[column], bins=20, edgecolor='k')
    axes[i].set_title(column)
    axes[i].set_xlabel('Value')
    axes[i].set_ylabel('Frequency')

plt.tight_layout()
plt.show()
```



High skewness in data. High skewness in a predictor variable can pose challenges for logistic regression. It may require data transformation, consideration of non-linear effects, and careful model evaluation to ensure the model's validity and interpretability. We'll give our model a higher iteration limit for it to converge with highly skewed data.

```
# Graphical correlation matrix
df = pd.DataFrame(data)
corr_matrix = df.corr()
plt.figure(figsize=(32,32))
sn.heatmap(corr_matrix, annot=True)
plt.show()
```



The correlation matrix demonstrates a high correlation in data. While we are focusing on predictability rather than coefficient interpretability in our models, it is crucial to address multicollinearity of our data to ensure that linearity holds for our logistic regression model. This will provide results with greater generalization performance.
k-NN is a non-parametric and instance-based algorithm, so the impact of correlated predictors is not a significant concern.

b) (12 points) Perform a predictive modeling analysis on this dataset to predict the type (benign B or malignant M) using a k-NN technique (for k=3) and the Logistic Regression technique. Please be specific about what other parameters you specified for your models. Briefly discuss your modeling process (e.g., validation technique, any preprocessing steps, parameters used to build the models, etc.) and show your code. Report the estimated coefficients of the Logistic Regression technique.

b.1 kNN

splitting the data to do holdout testing

```
[9] from sklearn.model_selection import train_test_split
    from sklearn.discriminant_analysis import StandardScaler
    X_knn = df.drop(['id','diagnosis'],axis=1)
    y_knn = df.diagnosis

    X_train, X_test, y_train, y_test = train_test_split(X_knn,y_knn,test_size=0.3,random_state=1, stratify=y_knn)

    # Normalization/Standardization
    # Instantiate StandardScaler
    sc = StandardScaler()
    # Fitting the StandardScaler
    sc.fit(X_train)

    # Transforming the datasets
    X_train_std = sc.transform(X_train)
    X_test_std = sc.transform(X_test)
```

X contains the feature variables, excluding the 'id' and 'diagnosis' columns from the DataFrame df.
y contains the target variable, which is the 'diagnosis' column from df.

In order to conduct holdout testing, we used Train_test_split() to split dataset into training and testing subsets, with a test size of 30% (test_size=0.3), and specified stratify=y to ensure that the class distribution in the target variable 'diagnosis' is preserved in both the training and testing sets.

We also created standardized versions of X variables to ensure that the features have similar scales.
X_train_std = sc.transform(X_train): Standardized the training features.
X_test_std = sc.transform(X_test): Standardized the testing features using the same scaling parameters (mean and standard deviation) obtained from the training data.

b.2 kNN modeling

```
# Perform a predictive modeling analysis on this dataset to predict the type (benign B or malignant M) using a k-NN technique (for k=3)

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score


knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_std,y_train)
y_pred = knn.predict(X_test_std)

print('Accuracy score for k=3 is: ',accuracy_score(y_test,y_pred))
```

```
Accuracy score for k=3 is:  0.9590643274853801
```

Initiated k-NN classifier with

- k=3, which implies that the classification decision is based on the labels of the three nearest neighbors in the feature space.
- Other default parameters
    - weights='uniform', # Default: 'uniform' (all neighbors have equal weight)
    - algorithm='auto', # Default: 'auto' (automatic selection of algorithm)

- leaf_size=30, # Default: 30 (for BallTree or KDTree algorithms)
- p=2, # Default: 2 (Euclidean distance metric)
- metric='minkowski', # Default: 'minkowski' (p-norm distance metric)
- metric_params=None, # Default: None (no additional parameters for the metric)
- n_jobs=None # Default: None (no parallelism)

Fitted 3NN with standardized Xs because kNN calculates distances between data points. Standardization helps prevent features with larger scales from dominating the distance calculations.

b.3 Logistics Regression

b.3.1 Multicollinearity check

In the data exploration part, we checked the correlation matrix of the attributes in the dataset and found out that there are many variables that are highly correlated with each other. We have calculated the VIF score for the dataset and the corresponding VIF scores are as follows:

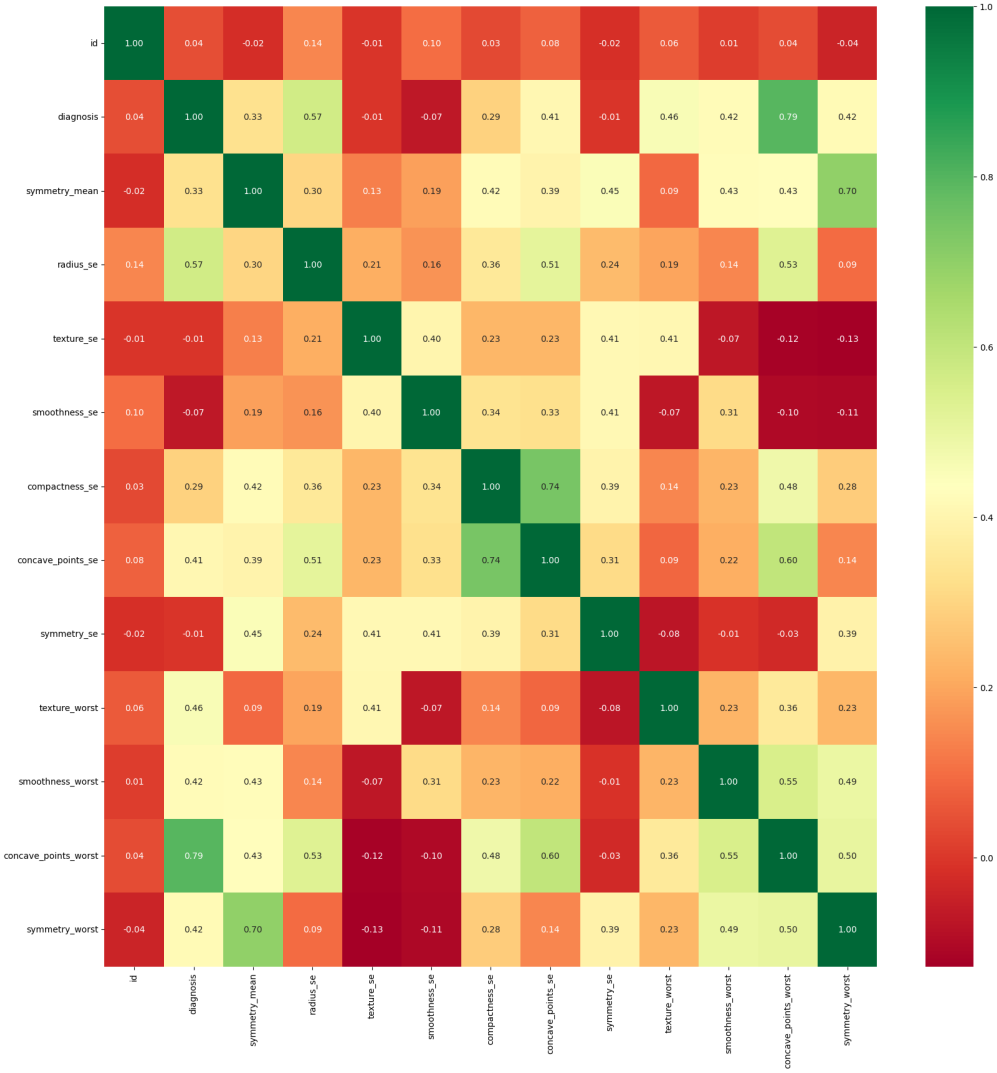
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
import pandas as pd

# Assume df is your DataFrame
# df = pd.read_csv('your_data.csv')

# Only keep numerical columns for VIF calculation
numerical_cols = df.select_dtypes(include=[float, int])

# Add a constant column for the intercept term
numerical_cols['Intercept'] = 1

# Calculate VIF for each variable
vif_data = pd.DataFrame()
vif_data['feature'] = numerical_cols.columns
vif_data['VIF'] = [variance_inflation_factor(numerical_cols.values, i)
                   for i in range(len(numerical_cols.columns))]
```

	feature	VIF
0	id	1.098563
1	radius_mean	3806.464754
2	texture_mean	11.901803
3	perimeter_mean	3786.400488
4	area_mean	347.968386
5	smoothness_mean	8.196794
6	compactness_mean	50.527114
7	concavity_mean	70.785037
8	concave_points_mean	60.060406
9	symmetry_mean	4.220809
10	fractal_dimension_mean	15.759755
11	radius_se	75.653561
12	texture_se	4.219350
13	perimeter_se	70.363596
14	area_se	41.688437
15	smoothness_se	4.052116
16	compactness_se	15.378147
17	concavity_se	15.699567
18	concave_points_se	11.592109
19	symmetry_se	5.179871
20	fractal_dimension_se	9.725655
21	radius_worst	799.418629
22	texture_worst	18.570023
23	perimeter_worst	405.023630
24	area_worst	337.319359
25	smoothness_worst	10.925873
26	compactness_worst	37.010829
27	concavity_worst	31.971022
28	concave_points_worst	36.791231
29	symmetry_worst	9.525054
30	fractal_dimension_worst	18.863976
31	Intercept	1868.614798

Because highly correlated attributes violate the independence assumption of the regression model, we conducted feature selection where highly correlation features are dropped in the dataset. The threshold that we chose for determining strong correlation is an absolute value of larger than 0.9 because we experimented with different thresholds and 0.9 generates better prediction accuracy. Below is the detailed process:

```python
# Calculate the correlation of each feature (except first two columns) with the target variable
correlation_with_target = df.iloc[:, 2:].corrwith(df['diagnosis']).abs()

# Create the correlation matrix for all columns except the first two
corrmatrix = df.iloc[:, 2:].corr().abs()

# Get pairs of highly correlated features
high_corr_var = np.where(corrmatrix > 0.9)
high_corr_var = [(corrmatrix.columns[x], corrmatrix.columns[y]) for x, y in zip(*high_corr_var) if x != y and x < y]

# Drop one of each pair based on correlation with target
for var1, var2 in high_corr_var:
    if correlation_with_target[var1] > correlation_with_target[var2]:
        df.drop(var2, axis=1, inplace=True, errors='ignore')
    else:
        df.drop(var1, axis=1, inplace=True, errors='ignore')
```

After feature selection, we conducted a correlation matrix again to look at the dataset:



After dropping the highly-correlated columns, similar to kNN, we used Train_test_split() to split the dataset into training and testing subsets, with a test size of 30% (test_size=0.3), and specified stratify=y to ensure that the class distribution in the target variable 'diagnosis' is preserved in both the training and testing sets.

```
[15] X = df.drop(['id','diagnosis'],axis=1)
     y = df.diagnosis

     X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=1, stratify=y)
```

b.4 Logistics Regression - modeling

```
# Perform a predictive modeling analysis on this dataset to predict the type (benign B or malignant M) using Logistic Regression technique
# X unstandardized
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn import metrics

clf = LogisticRegression(multi_class='auto',  # multi_class='auto' indicates that the classifier should handle multi-class classification automatically
                                        C=1,                 # C=1 specifies the regularization strength (smaller values of C correspond to stronger regularization)
                                        max_iter=200)       # maximum number of iterations taken for the solvers to converge

clf.fit(X_train,y_train)
y_pred_lr = clf.predict(X_test)

print('attributes are:', X_train.columns)
print('The weights of the attributes are:', clf.coef_)
print('The weights of the intercepts are:', clf.intercept_)
print('Accuracy score for Logistic Regression is: ',accuracy_score(y_test,y_pred_lr))
```

```
attributes are: Index(['smoothness_mean', 'compactness_mean', 'symmetry_mean',
       'fractal_dimension_mean', 'radius_se', 'texture_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave_points_se', 'symmetry_se',
       'fractal_dimension_se', 'texture_worst', 'perimeter_worst',
       'smoothness_worst', 'compactness_worst', 'concavity_worst',
       'concave_points_worst', 'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
The weights of the attributes are: [[ 0.24247968  0.42307757  0.35733679  0.0760836   1.26813522 -0.46876978
   0.02785743  0.02316518  0.07972108  0.0412188   0.08322154  0.00526121
   0.24376425  0.18267176  0.43448503  1.12887997  1.75559378  0.62008415
   1.04895537  0.2045002 ]]
The weights of the intercepts are: [-28.01300139]
Accuracy score for Logistic Regression is:  0.9473684210526315
```

Initiated Logistics Regression with

- multi_class='auto': This parameter specifies that the logistic regression classifier should automatically handle multi-class classification. When set to 'auto', it selects the appropriate strategy based on the number of classes in the target variable which is suitable for binary classification (benign vs. malignant).
- C=1: This parameter controls the inverse of the regularization strength. Smaller values of C indicate stronger regularization, which helps prevent overfitting. A value of 1 suggests moderate regularization.
- max_iter=200: This parameter defines the maximum number of iterations taken for the solver to converge. Logistic regression uses optimization algorithms to find the optimal coefficients for the model. Setting a higher value for max_iter ensures that the solver has sufficient iterations to converge, especially when dealing with our highly skewed Xs.

Fitted LR with unstandardized Xs.

b.4 Validation Techniques

Version 1:

With the train_test_split method above, we used Holdout testing – a model evaluation technique used in machine learning, where a dataset is split into two distinct subsets: a training set and a testing (holdout) set. The training set is used to train the machine learning model, while the testing set is reserved for assessing the model's performance. By comparing the model's predictions on the testing set to the actual target values, holdout testing provides an

estimate of how well the model generalizes to unseen data. It helps gauge the model's real-world predictive accuracy and guides decisions on hyperparameter tuning and model selection.

Version 2:

To get a more robust assessment of the model's generalization performance, we applied 5-fold cross-validation – a widely used technique in machine learning for evaluating the performance of a model. In this approach, the dataset is divided into five equal-sized subsets or "folds." The model is trained and evaluated five times, each time using a different fold as the testing set and the remaining four folds for training. This process ensures that every data point is used for both training and validation.

```
[18] # Perform a predictive modeling analysis on this dataset to predict the type (benign B or malignant M) using Logistic Regression technique
     # with cross validation
     from sklearn.linear_model import LogisticRegression
     from sklearn.model_selection import cross_val_score
     from sklearn import metrics

     #for kNN:
     sc = StandardScaler()
     sc.fit(X_knn)

     X_knn_std = sc.transform(X_knn)

     # for LR:
     X = df.drop(['id','diagnosis'],axis=1)
     y = df.diagnosis
```

Standardize X and y without splitting

kNN with cross validation:

```
19] #kNN with cross validation
    clf_knn = KNeighborsClassifier(n_neighbors=3)

    scores=cross_val_score(clf_knn,               # specify the model to use to fit the data
                                                  # in this case, the kNN classifier
                           X_knn_std,  # the data to fit. Can be for example a list, or an array
                                                  # the features used for training are petal length and petal width
                           y,          # the target variable to try to predict
                           cv=5)                  # specify the number of folds (if the estimator is a classifier and y is either binary or multiclass,
                                                  # sStratifiedKFold is used)

    print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
    # prints the accuracy scores for each fold
    print(scores)

    # F-1 scores
    # print the mean F1 score and its confidence interval
    scores_f1=cross_val_score(clf_knn,              # the model to use to fit the data
                              X_knn_std, # the data to fit. Can be for example a list, or an array
                              y,         # the target variable to try to predict
                              cv=5,                # specify the number of folds (if the estimator is a classifier and y is either binary or
                                                   # multiclass, StratifiedKFold is used)
                              scoring='f1_macro')   # the macro-average F1 score should be used as the scoring metric

    # prints the F1 scores for each fold
    print("F1-score: %0.2f (+/- %0.2f)" % (scores_f1.mean(), scores_f1.std() * 2)) # returns an array of scores of the estimator for each run
    print(scores_f1)
```

```
Accuracy: 0.96 (+/- 0.04)
[0.97368421 0.95614035 0.98245614 0.94736842 0.92920354]
F1-score: 0.95 (+/- 0.04)
[0.97186343 0.95310572 0.98095556 0.94153846 0.92421194]
```

LR with cross validation:

```
] #LR with cross validation
  clf_lr = LogisticRegression(multi_class='auto',  # multi_class='auto' indicates that the classifier should handle multi-class classification automatically
                                          C=1,                # C=1 specifies the regularization strength (smaller values of C correspond to stronger regularization)
                                          max_iter=200)       #  maximum number of iterations for optimization

  scores=cross_val_score(clf_lr,                # specify the model to use to fit the data
                                                # in this case, the logistic regression classifier
                         X,  # the data to fit. Can be for example a list, or an array
                                                # the features used for training are petal length and petal width
                         y,           # the target variable to try to predict
                         cv=5)                  # specify the number of folds (if the estimator is a classifier and y is either binary or multiclass,
                                                # sStratifiedKFold is used)

  print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
  # prints the accuracy scores for each fold
  print(scores)

  # F-1 scores
  # print the mean F1 score and its confidence interval
  scores_f1=cross_val_score(clf_lr,               # the model to use to fit the data
                            X, # the data to fit. Can be for example a list, or an array
                            y,          # the target variable to try to predict
                            cv=5,               # specify the number of folds (if the estimator is a classifier and y is either binary or
                                                 # multiclass, StratifiedKFold is used)
                            scoring='f1_macro')   # the macro-average F1 score should be used as the scoring metric

  # prints the F1 scores for each fold
  print("F1-score: %0.2f (+/- %0.2f)" % (scores_f1.mean(), scores_f1.std() * 2)) # returns an array of scores of the estimator for each run
  print(scores_f1)
```

```
Accuracy: 0.95 (+/- 0.04)
[0.92982456 0.93859649 0.98245614 0.93859649 0.96460177]
F1-score: 0.95 (+/- 0.04)
[0.92205128 0.93369339 0.98115079 0.934348   0.96277997]
```

The final evaluation metric is typically an average of the five individual performance scores (Accuracy and F1 here) with standard deviation, providing a reliable estimate of the model's overall accuracy and reducing the impact of randomness in the data split.

All result metrics will be further discussed in the following sections:

c) (13 points) Compare the generalization performance of the k-NN model with the Logistic Regression model. Make sure you report the confusion matrix, the predictive accuracy, precision, recall, and f-measure. Briefly discuss the results and show your code.

c.1 Generalization performance of the kNN model (train_test_split method and cross validation method)

c.1.1 kNN model- Train_test_split method result

```
# generalization performance of the kNN model.
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

print('Accuracy score for k=3 is: ',accuracy_score(y_test,y_pred))
print('Confusion matrix for k-NN is: \n',confusion_matrix(y_test,y_pred))
print('Classification report for k-NN is: \n',classification_report(y_test,y_pred))
```

```
Accuracy score for k=3 is:  0.9590643274853801
Confusion matrix for k-NN is:
 [[106   1]
 [  6  58]]
Classification report for k-NN is:
              precision    recall  f1-score   support

           B       0.95      0.99      0.97       107
           M       0.98      0.91      0.94        64

    accuracy                           0.96       171
   macro avg       0.96      0.95      0.96       171
weighted avg       0.96      0.96      0.96       171
```

In the confusion matrix:
The element at '[0][0]'(106) represents the number of true negatives (TN).
The element at '[0][1]'(1) represents the number of false positives (FP).
The element at '[1][0]'(6) represents the number of false negatives (FN).
The element at '[1][1]'(58) represents the number of true positives (TP).

The predictive accuracy is 0.96; the precision(B) is 0.95, the precision(M) is 0.98; the recall(B) is 0.99, the recall(M) is 0.91, the f1-score(B) is 0.97, the f1-score(M) is 0.94.

c.1.2 kNN model- Cross validation method result

```
Accuracy: 0.95 (+/- 0.04)
[0.92982456 0.93859649 0.98245614 0.93859649 0.96460177]
F1-score: 0.95 (+/- 0.04)
[0.92205128 0.93369339 0.98115079 0.934348   0.96277997]
```

The mean of the accuracy of the kNN model is 0.95 with a SD of .04.
The mean of the F1-score of the kNN model is 0.95 with a SD of .04.

c.2 Generalization performance of the Logistic model (train_test_split method and cross validation method)

c.2.1 LR- train_test_split method result

```
7] # generalization performance of the Logistic Regression model.

   print('Accuracy score for Logistic Regression is: ',accuracy_score(y_test,y_pred_lr))
   print('Confusion matrix for Logistic Regression is: \n',confusion_matrix(y_test,y_pred_lr))
   print('Classification report for Logistic Regression is: \n',classification_report(y_test,y_pred_lr))
```

```
Accuracy score for Logistic Regression is:  0.9473684210526315
Confusion matrix for Logistic Regression is:
 [[105   2]
 [  7  57]]
Classification report for Logistic Regression is:
              precision    recall  f1-score   support

           0       0.94      0.98      0.96       107
           1       0.97      0.89      0.93        64

    accuracy                           0.95       171
   macro avg       0.95      0.94      0.94       171
weighted avg       0.95      0.95      0.95       171
```

In the confusion matrix:
The element at '[0][0]'(105) represents the number of true negatives (TN).
The element at '[0][1]'(2) represents the number of false positives (FP).
The element at '[1][0]'(7) represents the number of false negatives (FN).
The element at '[1][1]'(57) represents the number of true positives (TP).

Logistic Regression

```
[16] df['diagnosis'] = df['diagnosis'].replace({'M': 1, 'B': 0})
```

The predictive accuracy is 0.95; the precision(B) is 0.95, the precision(M) is 0.97; the recall(B) is 0.98, the recall(M) is 0.91, the f1-score(B) is 0.96, the f1-score(M) is 0.94.

c.1.2 LR- Cross validation method result

```
Accuracy: 0.95 (+/- 0.04)
[0.92982456 0.93859649 0.98245614 0.93859649 0.96460177]
F1-score: 0.95 (+/- 0.04)
[0.92205128 0.93369339 0.98115079 0.934348   0.96277997]
```

The mean of the accuracy of the Logistics Regression model is 0.95 with a SD of .04.
The mean of the F1-score of the Logistics Regression model is 0.95 with a SD of .04.

c.3 Comparison of the generalization performance of the k-NN model and the Logistic Regression model:

- Using the train-test split method, k-NN scored higher in both predictive accuracies and f1-scores; we can know that the k-NN model's generalization performance is better than the Logistic Regression model.
- Using cross validation method, the mean of the accuracy of the kNN model and the Logistics Regression model both have means of 0.95 with a SD of .04, the mean of the F1 score of the kNN model and the Logistics Regression model both have means of 0.95 with a SD of .04.
- From an application in real world perspective, false negatives in cancer detection (i.e., failing to detect a malignant case) can be life-threatening, as it may result in a delayed diagnosis and treatment, so we want to minimize the false negatives(FN) of our models. In this case, the FN of the kNN models is 6, while for the Logistic Model, the FN is 7. From this perspective, we should prefer the kNN model over the logistic model.
- To sum up, the generalization performance of the k-NN model is better than the Logistic Regression model.

d) (15 points) What generalization performance metric would you prefer to use in order to choose the best performing model in this context and why? Please be clear about any assumptions you might make when you choose the generalization performance metric you would prefer.

a. In the context of cancer diagnosis, recall is often the preferred metric for assessing the generalization performance of a model because the cost of missing a true positive is extremely high—delayed treatment could worsen patient outcomes. More specifically, a Type 1 error, or false positive, occurs when a healthy individual is incorrectly diagnosed with cancer. This could lead to unnecessary emotional distress and additional medical procedures, such as biopsies and MRIs, which are not only costly but can also divert medical resources from more urgent cases. On the other hand, a Type 2 error, or false negative, happens when an individual with cancer is mistakenly identified as healthy. This can be far more dangerous, as it could result in delayed treatment and allow the disease to progress to a more severe stage.

However, it's important to note that maximizing recall alone can lead to an excessive number of false positives. While a false positive may be less harmful than a false negative, it can still strain medical resources and cause unnecessary stress for patients. Additional diagnostic tests are often required to confirm a positive diagnosis, which could be resource-intensive. So, while recall is crucial in this setting, a balanced approach that also considers other metrics like precision could provide a more comprehensive view of the model's performance.

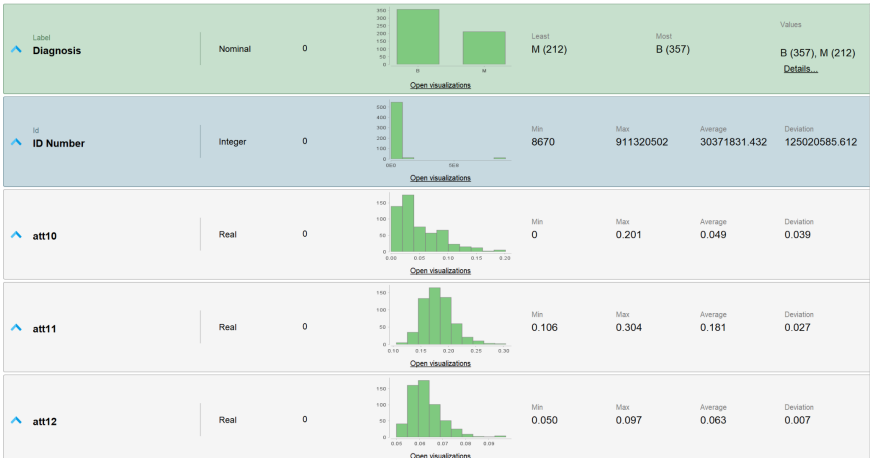
20 Points (Rapidminer):

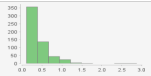
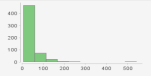
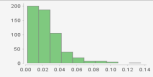
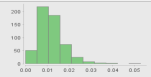
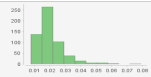
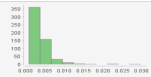
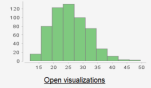
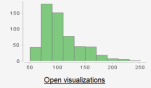
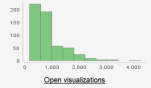
Perform a predictive modeling analysis on this dataset to predict the type (benign B or malignant M) using a k-NN technique (for k=3) and the Logistic Regression technique. Compare the generalization performance of the k-NN model with the Logistic Regression model. Make sure you report the confusion matrix, the predictive accuracy, precision, recall, and f-measure.

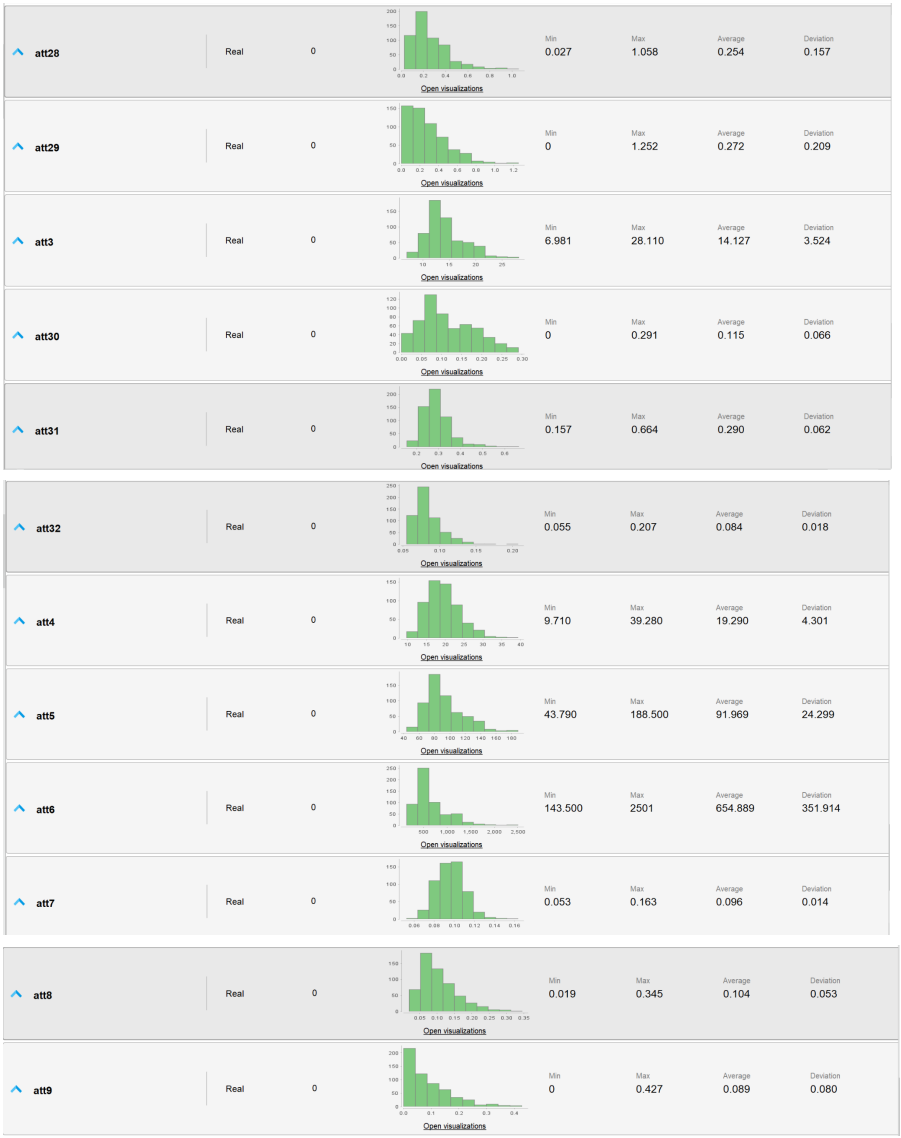
a) [20 points] Please show below screenshots of the models you have built using Rapidminer, the results, and the parameters you have specified.

a. [8 points] Data Preview

i. Screenshot of the summary statistics table in Rapidminer.



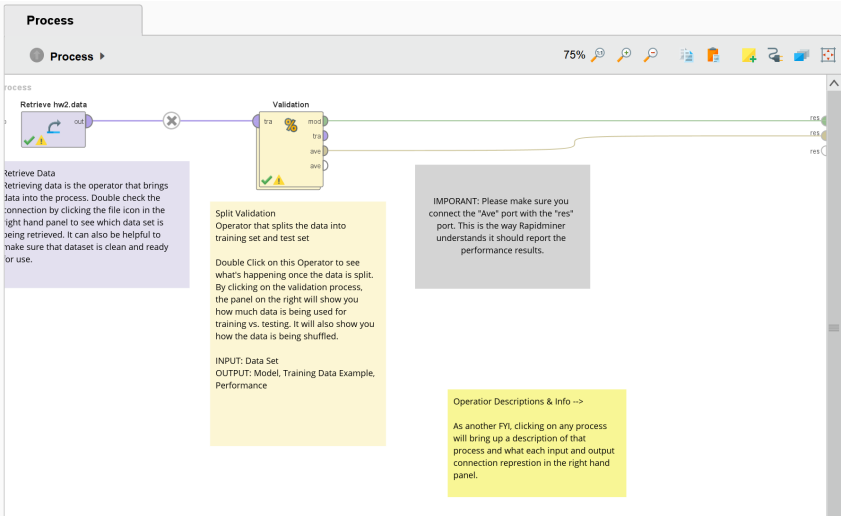
att13	Real	0	 Open visualizations	Min 0.112	Max 2.873	Average 0.405	Deviation 0.277
att14	Real	0	 Open visualizations	Min 0.360	Max 4.885	Average 1.217	Deviation 0.552
att15	Real	0	 Open visualizations	Min 0.757	Max 21.980	Average 2.866	Deviation 2.022
att16	Real	0	 Open visualizations	Min 6.802	Max 542.200	Average 40.337	Deviation 45.491
att17	Real	0		Min 0.002	Max 0.031	Average 0.007	Deviation 0.003
att18	Real	0	 Open visualizations	Min 0.002	Max 0.135	Average 0.025	Deviation 0.018
att19	Real	0	 Open visualizations	Min 0	Max 0.396	Average 0.032	Deviation 0.030
att20	Real	0	 Open visualizations	Min 0	Max 0.053	Average 0.012	Deviation 0.006
att21	Real	0	 Open visualizations	Min 0.008	Max 0.079	Average 0.021	Deviation 0.008
att22	Real	0		Min 0.001	Max 0.030	Average 0.004	Deviation 0.003
att23	Real	0	 Open visualizations	Min 7.930	Max 36.040	Average 16.269	Deviation 4.833
att24	Real	0	 Open visualizations	Min 12.020	Max 49.540	Average 25.677	Deviation 6.146
att25	Real	0	 Open visualizations	Min 50.410	Max 251.200	Average 107.261	Deviation 33.603
att26	Real	0	 Open visualizations	Min 185.200	Max 4254	Average 880.583	Deviation 569.357
att27	Real	0		Min 0.071	Max 0.223	Average 0.132	Deviation 0.023

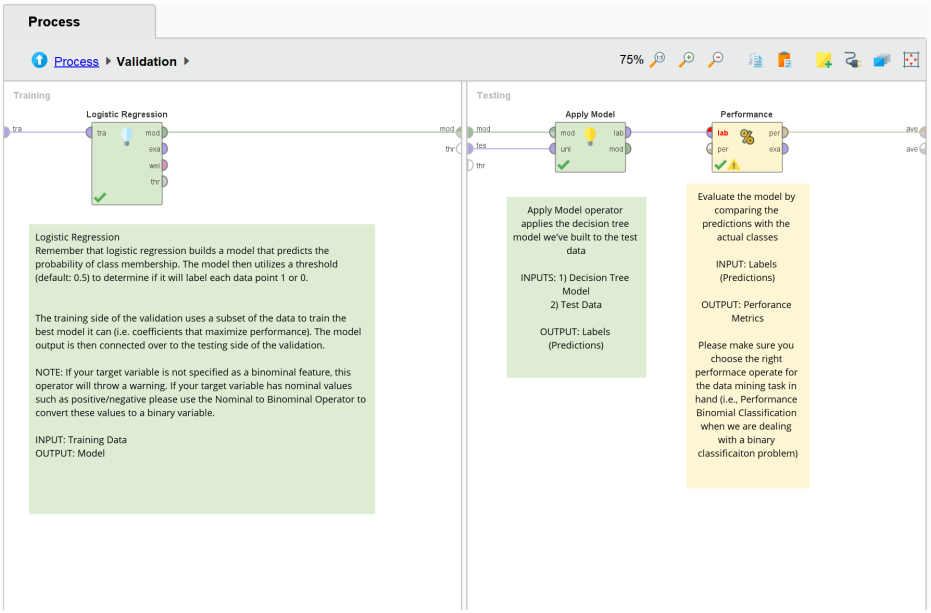


b. [8 points] Logistic Regression

i. Screenshots for Logistic Regression Model Setup (Rapidminer Processes)

(Insert Screenshots here – 2 screenshots are expected here; one for the upper layer and one inside the validation technique)





ii. Screenshot for Logistic Regression Performance
(Insert Screenshot here)

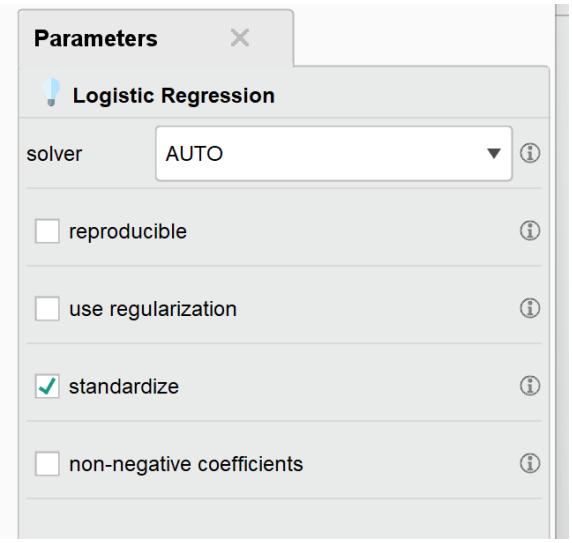
accuracy: 93.57%

	true M	true B	class precision
pred. M	71	9	88.75%
pred. B	2	89	97.80%
class recall	97.26%	90.82%	

iii. Screenshot for Logistic Regression Results (Coefficients)
(Insert Screenshot here)

Attribute	Coefficient	Std. Coefficient	Std. Error	z-Value	p-Value
att3	518.435	1826.989	132.630	3.909	0.000
att4	-9.239	-39.735	4.967	-1.860	0.063
att5	-21.003	-510.344	18.623	-1.128	0.259
att6	-3.605	-1268.592	0.502	-7.180	0.000
att7	-4519.994	-63.570	1479.560	-3.055	0.002
att8	4930.328	260.384	1055.083	4.673	0.000
att9	-2253.026	-179.611	600.822	-3.750	0.000
att10	-2859.136	-110.943	1126.839	-2.537	0.011
att11	1513.301	41.486	412.598	3.668	0.000
att12	-5288.088	-37.336	1740.680	-3.038	0.002
att13	-210.842	-58.469	258.776	-0.815	0.415
att14	20.691	11.414	26.912	0.769	0.442
att15	80.959	163.687	26.617	3.042	0.002
att16	-8.500	-386.687	2.028	-4.192	0.000
att17	4311.539	12.945	5657.258	0.762	0.446
att18	-8222.406	-147.248	1780.715	-4.617	0.000
att19	6206.293	187.344	692.147	8.967	0
att20	-22943.263	-141.566	3601.386	-6.371	0.000
att21	7479.816	61.831	1958.086	3.820	0.000
att22	68305.169	180.740	14239.251	4.797	0.000
att23	-143.477	-693.458	43.061	-3.332	0.001
att24	-5.890	-36.202	3.742	-1.574	0.115
att25	-3.485	-117.103	4.217	-0.826	0.409
att26	0.975	554.899	0.392	2.488	0.013
att27	1443.635	32.962	1352.968	1.067	0.286
att28	733.190	115.357	241.707	3.033	0.002
att29	-570.769	-119.076	221.309	-2.579	0.010
att30	-650.638	-42.768	539.544	-1.206	0.228
att31	-1680.257	-103.953	203.574	-8.254	0.000
att32	-6092.790	-110.044	2137.709	-2.850	0.004
Intercept	206.830	-121.045	281.231	0.735	0.462

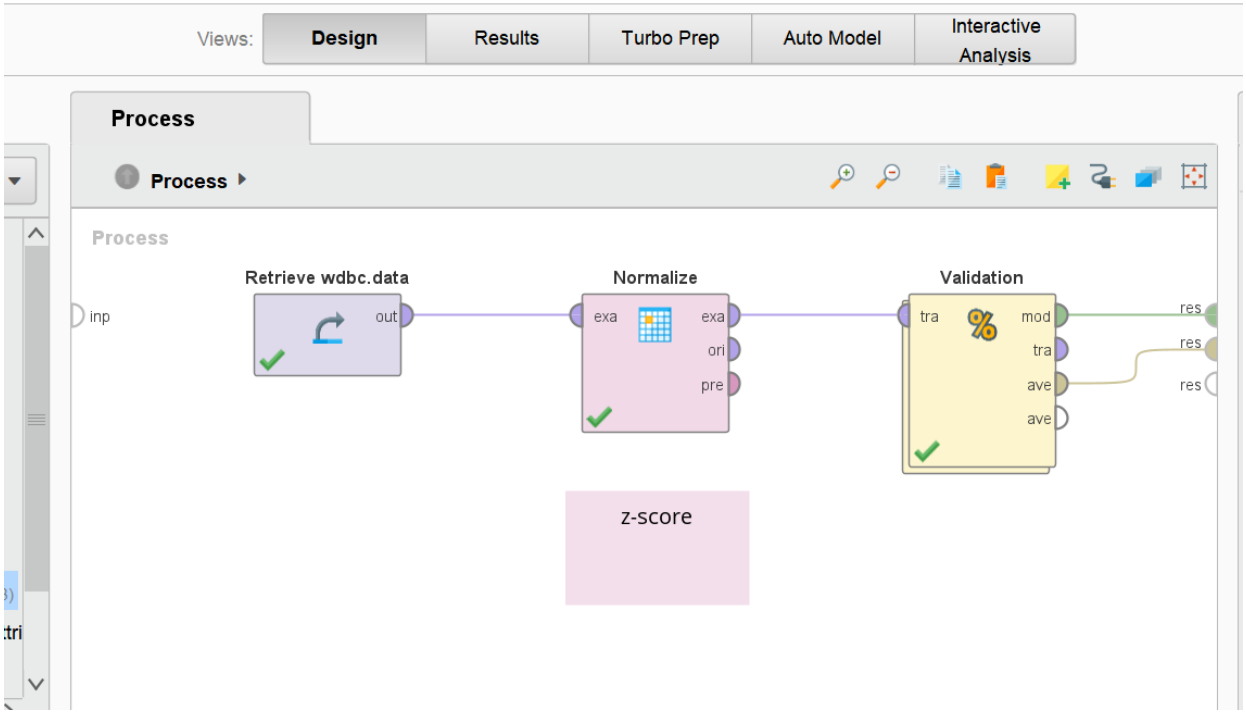
iv. Screenshot for Logistic Regression Rapidminer Operator Parameters (click on Logistic Regression operator and then take a screenshot of the Parameters window on the right) (Insert Screenshot here)

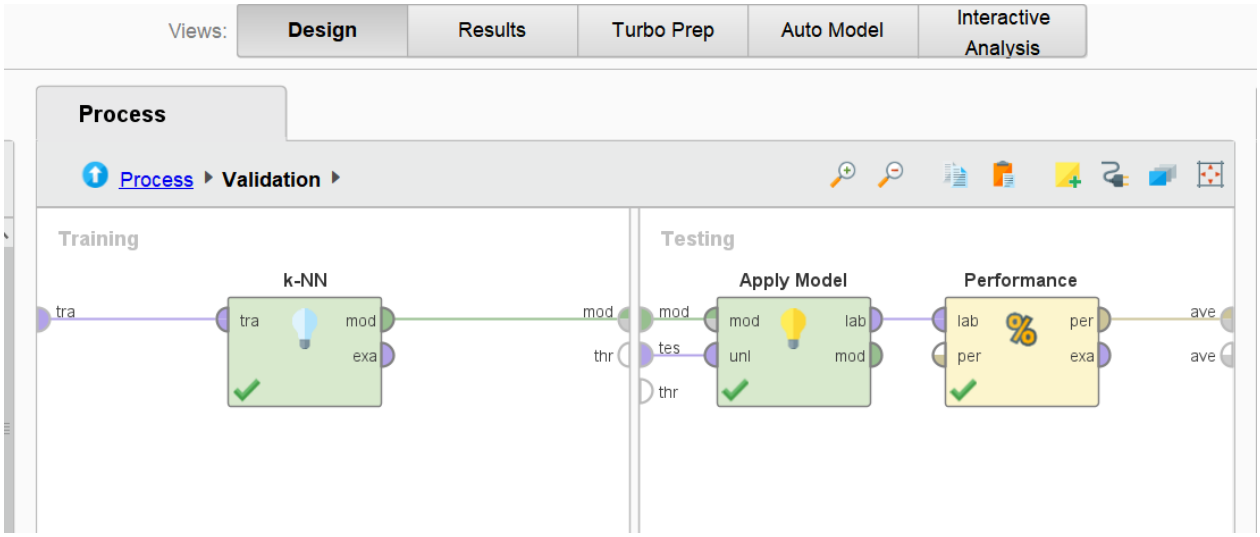


c. [8 points] kNN

i. Screenshots for kNN Model Setup (Rapidminer Processes)

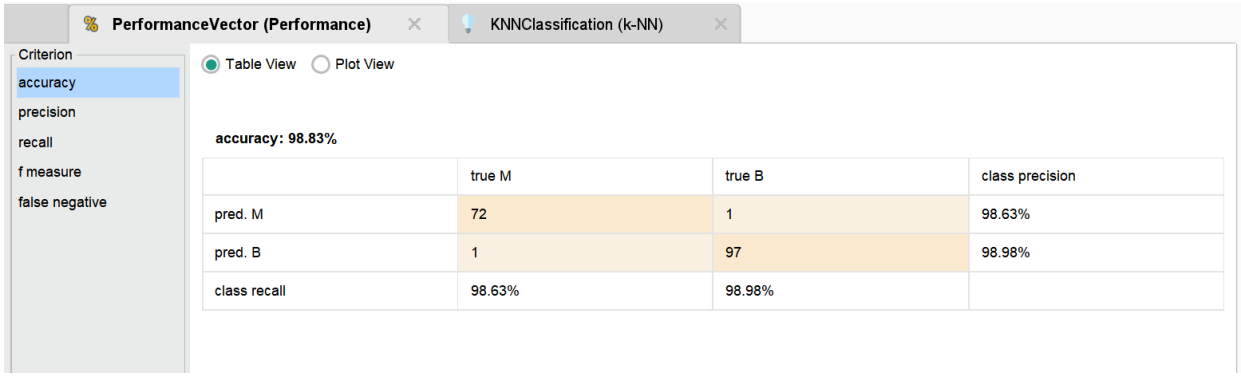
(Insert Screenshots here– 2 screenshots are expected here; one for the upper layer and one inside the validation technique)

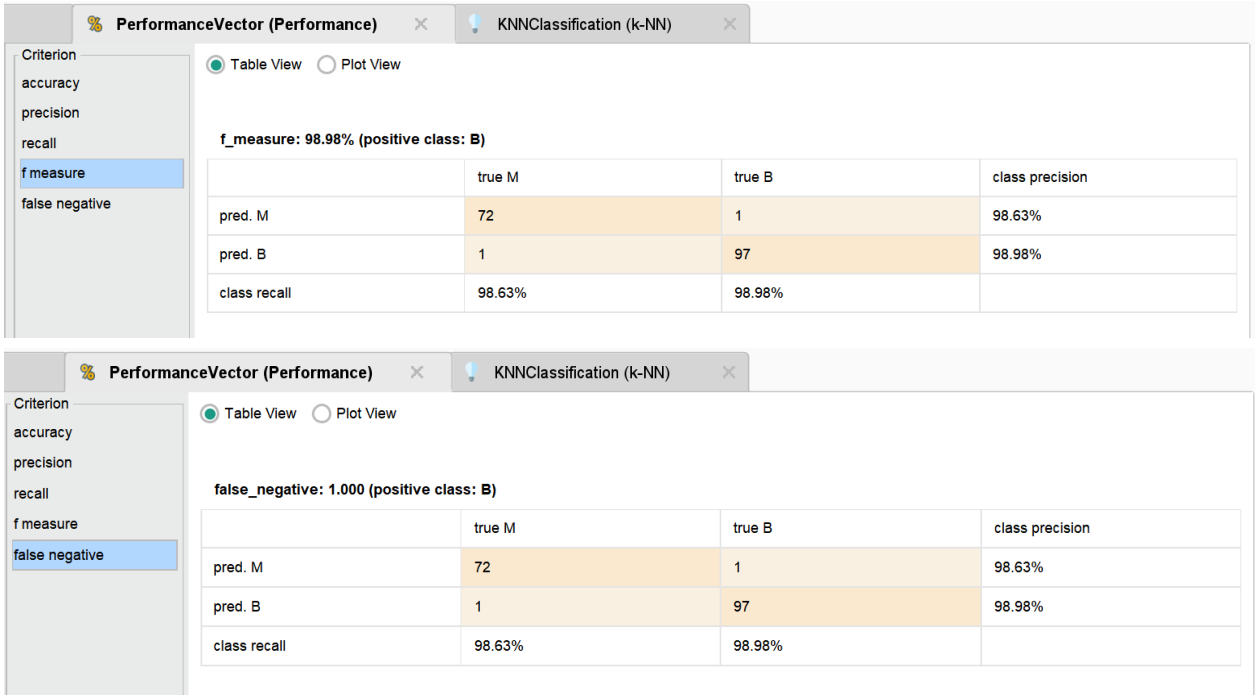




ii. Screenshot for kNN Performance

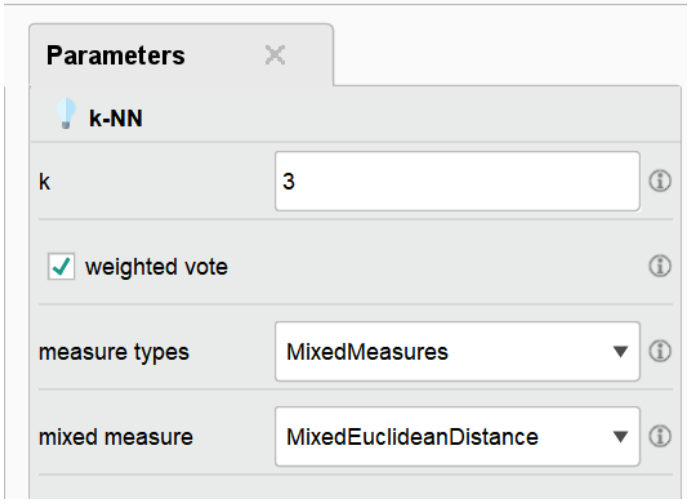
(Insert Screenshot here)





iii. Screenshot for kNN Rapidminer Operator Parameters (click on kNN operator and then take a screenshot of the Parameters windows on the right)

(Insert Screenshot here)



Appendix (Data Description)

1.Title: Wisconsin Diagnostic Breast Cancer (WDBC)

Results:

- predicting field 2, diagnosis: B = benign, M = malignant

2. Number of instances: 569

3. Number of attributes: 32 (ID, diagnosis, 30 real-valued input features)

4. Attribute information

1) ID number
2) Diagnosis (M = malignant, B = benign)
3-32)

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

a) radius (mean of distances from center to points on the perimeter)
b) texture (standard deviation of gray-scale values)
c) perimeter
d) area
e) smoothness (local variation in radius lengths)
f) compactness (perimeter^2 / area - 1.0)
g) concavity (severity of concave portions of the contour)
h) concave points (number of concave portions of the contour)
i) symmetry
j) 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.

All feature values are recorded with four significant digits.

5. Missing attribute values: none

6. Class distribution: 357 benign, 212 malignant