EMORY UNIVERSITY

Introduction to Business Analytics [MSBA]

Homework #2 Part 2

Dola Qiu, Seth Abayo, Lanston Chen, Yizhou Sun, Yihua Wang

(put your full names above (incl. any nicknames))

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.**

Import library

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

plt.rcParams['axes.labelsize'] = 14   # fontsize of the x any y labels
plt.rcParams['xtick.labelsize'] = 12 # fontsize of the x tick labels
plt.rcParams['ytick.labelsize'] = 12 # fontsize of the y tick labels

# Sklearn imports
from sklearn import linear_model
from sklearn.metrics import confusion_matrix, f1_score, accuracy_score,precision_score, recall_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
#Scipy imports
from scipy import stats

#import itertools
import itertools

# Suppress warnings
import warnings
warnings.filterwarnings("ignore")
```

Read the data from URL & Add column names

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

```python
column_names = ['ID','Diagnosis']

# Add mean real-valued features
mean_features = ['Mean_Radius', 'Mean_Texture', 'Mean_Perimeter', 'Mean_Area',
                 'Mean_Smoothness', 'Mean_Compactness', 'Mean_Concavity',
                 'Mean_Concave_Points', 'Mean_Symmetry', 'Mean_Fractal_Dimension']

# Add standard error feature names
se_features = ['SE_Radius', 'SE_Texture', 'SE_Perimeter', 'SE_Area',
               'SE_Smoothness', 'SE_Compactness', 'SE_Concavity',
               'SE_Concave_Points', 'SE_Symmetry', 'SE_Fractal_Dimension']

# Add worst feature names
worst_features = ['Worst_Radius', 'Worst_Texture', 'Worst_Perimeter', 'Worst_Area',
                  'Worst_Smoothness', 'Worst_Compactness', 'Worst_Concavity',
                  'Worst_Concave_Points', 'Worst_Symmetry', 'Worst_Fractal_Dimension']


# combine all columns
all_columns = column_names + mean_features + se_features + worst_features

# assign column names to dataframe
data.columns = all_columns
```

	ID	Diagnosis	Mean_Radius	Mean_Texture	Mean_Perimeter	Mean_Area	Mean_Smoothness	Mean_Compactness	Mean_Concavity	Mean_Concave_Points	...	Worst_Radius	Worst_Texture	Worst_Perimeter	Wor
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.60	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.80	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	152.50	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	98.87	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	152.20	

Explore the data by reporting summary statistics

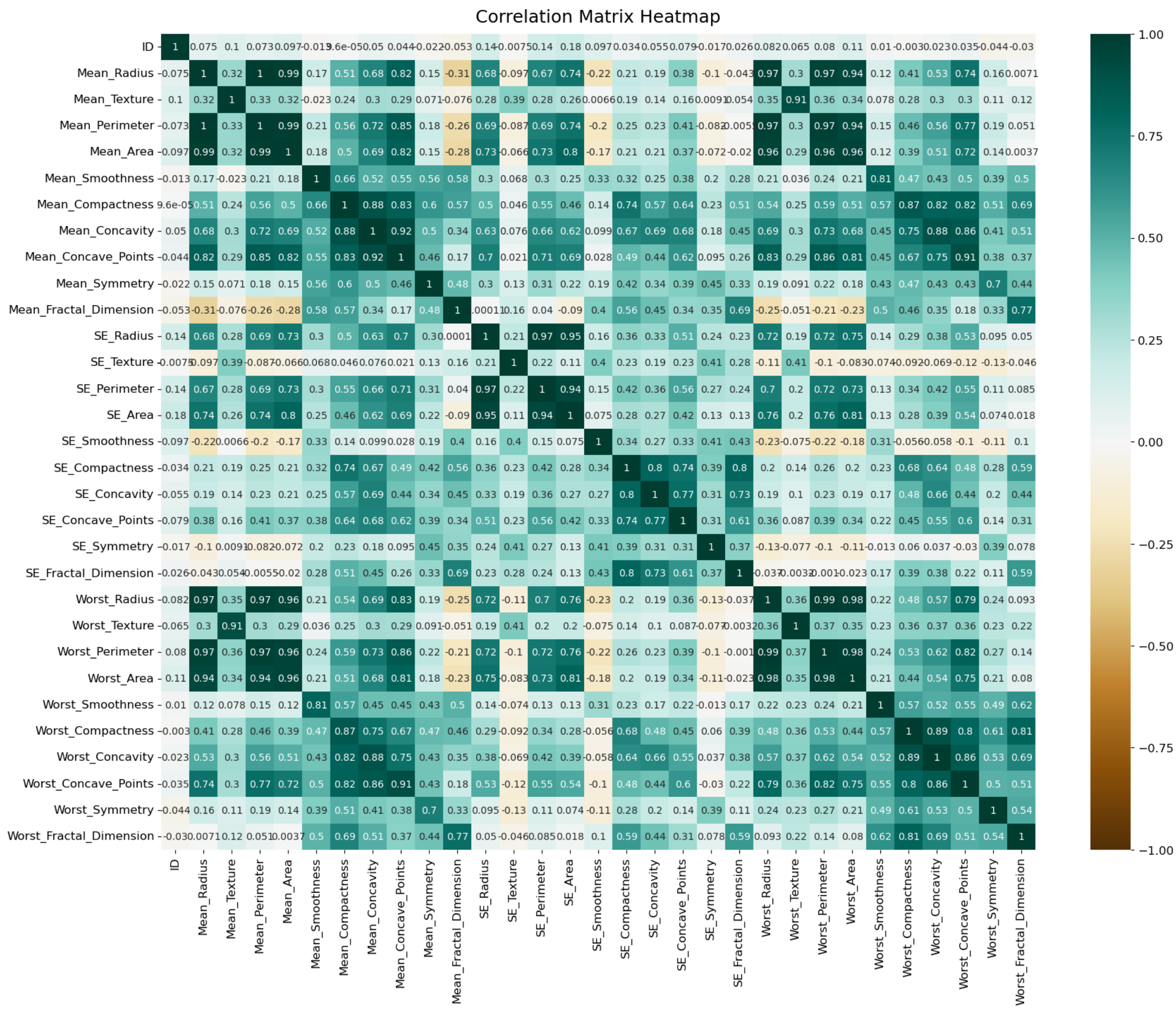
```python
# Using describe function to compute summary statistics of entire df. It automatically ignore cat features
# Slice the dataset to skip first column as it is an ID
data.iloc[:,1:].describe()[1:] # Skip first row as it returns instance counts and it is the same across all features (569)
```

	Mean_Radius	Mean_Texture	Mean_Perimeter	Mean_Area	Mean_Smoothness	Mean_Compactness	Mean_Concavity	Mean_Concave_Points	Mean_Symmetry	Mean_Fractal_Dimension	...	Worst_Radius	Worst_Textu
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	...	16.269190	25.6772
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	...	4.833242	6.1462
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	...	7.930000	12.0200
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	...	13.010000	21.0800
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	...	14.970000	25.4100
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	...	18.790000	29.7200
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	...	36.040000	49.5400

7 rows × 30 columns

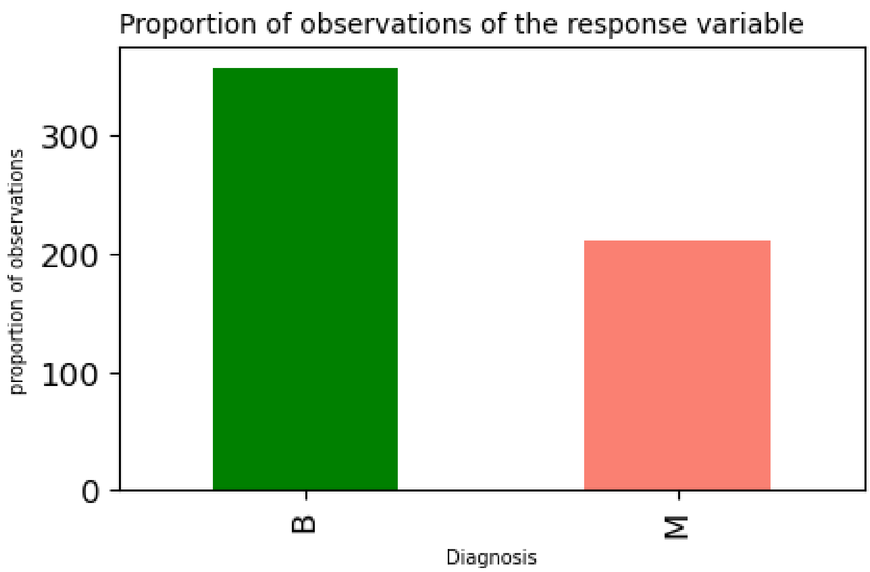
Explore the data by correlation matrix

```python
# Plot heatmap correlation matrix on a graph Reference: https://medium.com/@szabo.bibor/how-to-create-a-seaborn-correlation-heatmap-in-python-834c0686b88e
plt.figure(figsize=(20,15))

heatmap = sns.heatmap(data.corr(), vmin=-1, vmax=1, annot=True, cmap='BrBG')

heatmap.set_title('Correlation Matrix Heatmap', fontdict={'fontsize': 18}, pad=12)
```



Explore the data by check class imbalance

```
fig = plt.figure(figsize=(5,3))
ax = fig.add_subplot(111)
data['Diagnosis'].value_counts().plot(kind='bar',
                                        ax=ax,
                                        color=['green','salmon'])

# set title and labels
ax.set_title('Proportion of observations of the response variable',
             fontsize=10, loc='left')
ax.set_xlabel('Diagnosis',
              fontsize=7)
ax.set_ylabel('proportion of observations',
              fontsize=7)
```



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.**

Model processing (which both used in KNN and Logistic Regression)

```
# Extract X and y
X = data.drop(columns=['Diagnosis','ID'],axis=1)
y = data.Diagnosis
X.head()
X_train,X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,stratify=y, random_state=42)
```

Split the data to test set and train set,
Random state = 42
Strategy = y (Maintains same label distribution in training and test sets as in original data)
Also drop the 'ID' column, which obviously make no sense the predict the target value

KNN predict model

- Scaling data for KNN (Below codes were adopted from class code example)

```
# Instantiate StandardScaler
sc = StandardScaler()
# Fitting the StandardScaler
sc.fit(X_train)

# Transforming the datasets
X_train_std = sc.transform(X_train) # Perform standardization of train set X attributes by centering and scaling
                                     # This line uses the transform method of the sc object to standardize the features in the tra
X_test_std = sc.transform(X_test)   # Perform standardization of test set X attributes by centering and scaling
                                     # Similarly, this line standardizes the features in the testing set.
                                     # Importantly, it uses the same mean and standard deviation values that were computed from the
```

Link:https://colab.research.google.com/drive/1Tk3iWD1MgSIUrhbvrEobmcA5PNG7Njkw?usp=sharing#scrollTo=Fe_Mfq1QsP3g

- Fit the model

```
# K-NN model
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_std, y_train)
```

Use the parameters K = 3

Logistic Regression predict model

- Fit the model

```
#Logistic regression modelm
log_reg = linear_model.LogisticRegression(solver='lbfgs',max_iter=500)
log_reg.fit(X_train, y_train)
```

  - *Use the 'lbfgs' parameter as solver, which is a quasi-Newton optimization method: 'lbfgs' is good for small datasets but struggle for larger datasets.*
  - *Max_iter = 500, which sets the maximum number of iterations taken for the solver to converge*

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.**

Create confusion matrix graph function (Below codes were adopted from class code example)

```
# Function that prints and plots the confusion matrix.
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]          # devide absolute number of observations with sum across columns to get the relative percentage of observations
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)                 # shows the confusion matrix in the console
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))                                 # add tick marks to the confusion matrix
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'                                     # choose format depending on whether the confusion matrix is normalizaed or not
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):  # loop that adds the value to each cell of the confusion matrix
        plt.text(j, i, format(cm[i, j], fmt),                              # we reformat how the cell values are displayed accroding to the variable fmt we defined before
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Link:https://colab.research.google.com/drive/1Tk3iWD1MgSIUrhbvrEobmcA5PNG7Njkw?usp=sharing#scrollTo=Fe_Mfq1QsP3g

Predict the model

```
# k-NN and Logistic predictions
k_nn_pred = knn.predict(X_test_std)
log_reg_pred = log_reg.predict(X_test)

# k-NN Compute confusion matrix to evaluate the accuracy of a classification
knn_cnf_matrix = confusion_matrix(y_test, k_nn_pred)
knn_accuracy = accuracy_score(y_test, k_nn_pred)
knn_report = classification_report(y_test, k_nn_pred)


# Logistic Regression Compute confusion matrix to evaluate the accuracy of a classification
log_reg_cnf_matrix = confusion_matrix(y_test, log_reg_pred)
log_reg_accuracy = accuracy_score(y_test, log_reg_pred)
log_reg_report = classification_report(y_test, log_reg_pred)
```

Print the Performance

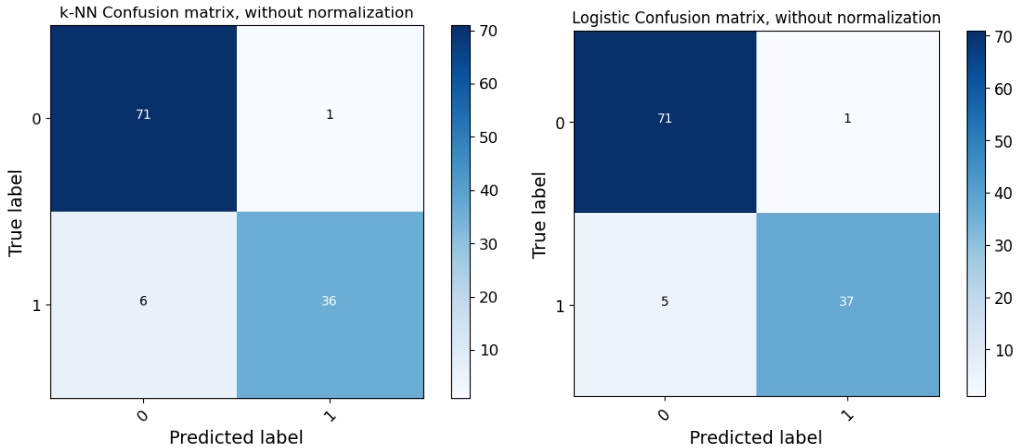
```
#print KNN performance
print("K-NN Evaluation:")
print("Confusion Matrix:\n", knn_cnf_matrix)
print("Accuracy:", knn_accuracy)
print("Classification Report:\n", knn_report)

#print KNN performance
print("\nLogistic Regression Evaluation:")
print("Confusion Matrix:\n", log_reg_cnf_matrix)
print("Accuracy:", log_reg_accuracy)
print("Classification Report:\n", log_reg_report)
```

```
K-NN Evaluation:
Confusion Matrix:
 [[71  1]
 [ 6 36]]
Accuracy: 0.9385964912280702
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.99      0.95        72
           1       0.97      0.86      0.91        42

    accuracy                           0.94       114
   macro avg       0.95      0.92      0.93       114
weighted avg       0.94      0.94      0.94       114
```

```
Logistic Regression Evaluation:
Confusion Matrix:
 [[71  1]
 [ 5 37]]
Accuracy: 0.9473684210526315
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.99      0.96        72
           1       0.97      0.88      0.93        42

    accuracy                           0.95       114
   macro avg       0.95      0.93      0.94       114
weighted avg       0.95      0.95      0.95       114
```

Plot the confusion matrix graph

```
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(knn_cnf_matrix,
                      classes=class_names,
                      title='k-NN Confusion matrix, without normalization')

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(log_reg_cnf_matrix,
                      classes=class_names,
                      title='Logistic Confusion matrix, without normalization')
```



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.**

Given that the context is cancer detection, reducing false negatives is crucial because failing to identify a true cancer case could be life-threatening. Therefore, the primary metric I'd focus on is "Recall" for the positive class (1), which measures the model's ability to correctly identify all positive cases. Higher recall minimizes the chance of false negatives.

Assumptions:
1. We assume that a false negative is significantly more costly than a false positive.
2. We also assume that both models have been adequately tuned and validated, and the datasets used are representative of the population.

In this case context:
1. Based on these assumptions and the provided confusion matrix, both models have similar but not identical recall scores for the positive class (1).
2. Logistic Regression has a slightly higher recall of 0.88 compared to K-NN's 0.86.

In a broader context:
1. Logistic Regression offers easier interpretability, which is crucial for medical applications where explaining the model's decisions can be as important as the decision itself.
2. Logistic Regression is computationally less intensive than K-NN, making it more scalable for larger datasets.
3. Logistic Regression handles imbalanced classes better with proper regularization and weighting, often seen in medical datasets like cancer prediction.

Based on both the specific dataset metrics and general considerations, I'd recommend using Logistic Regression for cancer detection.

20 Points (Rapidminer):

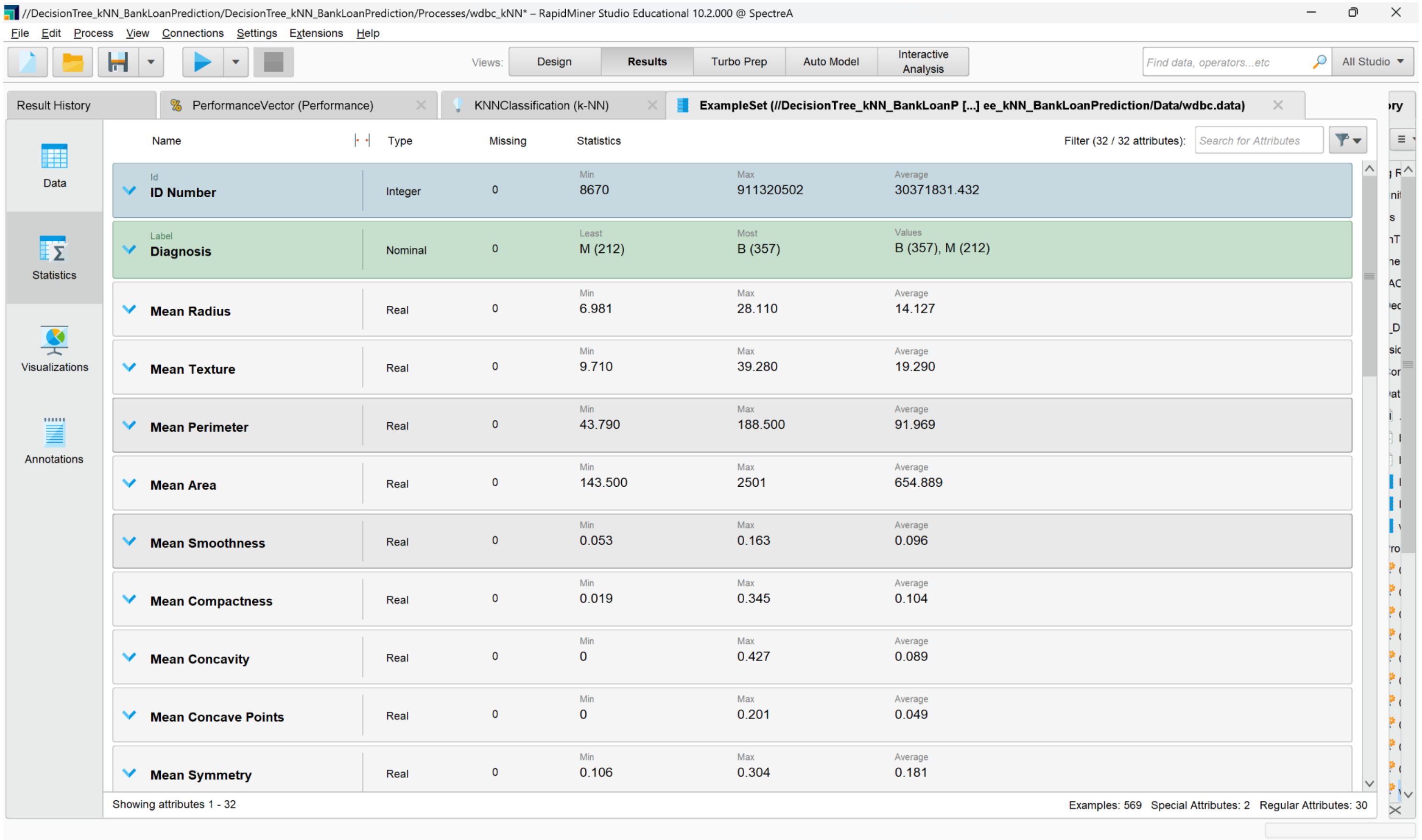
Perform a predictive modeling analysis on this dataset to predict the type (benign B or malignant M) using a k-NN technique (for k=3) and the Logistic Regression technique. Compare the generalization performance of the k-NN model with the Logistic Regression model. Make sure you report the confusion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

a) **[20 points] Please show <u>below</u> screenshots of the models you have built using Rapidminer, the results, and the parameters you have specified.**
   a. **[8 points] Data Preview**

Open in Turbo Prep Auto Model   Filter (569 / 569 examples): all

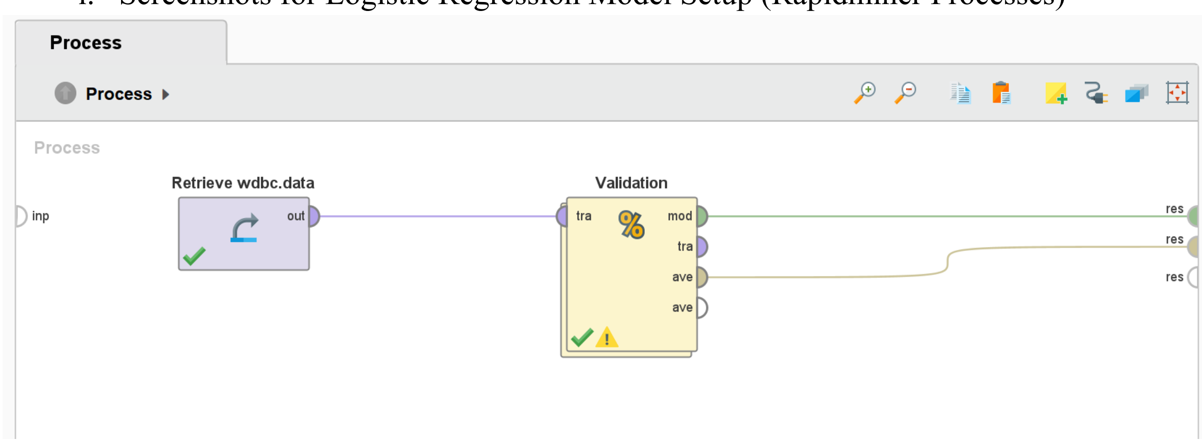
Row No.	ID Number	Diagnosis	Mean Radius	Mean Texture	Mean Peri...	Mean Area	Mean Smo...	Mean Com...	Mean Conc...	Mean Conc...	Mean Sym...	Mean Fract...	Radius
1	842302	M	17.990	10.380	122.800	1001	0.118	0.278	0.300	0.147	0.242	0.079	1.095
2	842517	M	20.570	17.770	132.900	1326	0.085	0.079	0.087	0.070	0.181	0.057	0.543
3	84300903	M	19.690	21.250	130	1203	0.110	0.160	0.197	0.128	0.207	0.060	0.746
4	84348301	M	11.420	20.380	77.580	386.100	0.142	0.284	0.241	0.105	0.260	0.097	0.496
5	84358402	M	20.290	14.340	135.100	1297	0.100	0.133	0.198	0.104	0.181	0.059	0.757
6	843786	M	12.450	15.700	82.570	477.100	0.128	0.170	0.158	0.081	0.209	0.076	0.335
7	844359	M	18.250	19.980	119.600	1040	0.095	0.109	0.113	0.074	0.179	0.057	0.447
8	84458202	M	13.710	20.830	90.200	577.900	0.119	0.165	0.094	0.060	0.220	0.075	0.584
9	844981	M	13	21.820	87.500	519.800	0.127	0.193	0.186	0.094	0.235	0.074	0.306
10	84501001	M	12.460	24.040	83.970	475.900	0.119	0.240	0.227	0.085	0.203	0.082	0.298
11	845636	M	16.020	23.240	102.700	797.800	0.082	0.067	0.033	0.033	0.153	0.057	0.380
12	84610002	M	15.780	17.890	103.600	781	0.097	0.129	0.100	0.066	0.184	0.061	0.506
13	846226	M	19.170	24.800	132.400	1123	0.097	0.246	0.206	0.112	0.240	0.078	0.956
14	846381	M	15.850	23.950	103.700	782.700	0.084	0.100	0.099	0.054	0.185	0.053	0.403
15	84667401	M	13.730	22.610	93.600	578.300	0.113	0.229	0.213	0.080	0.207	0.077	0.212
16	84799002	M	14.540	27.540	96.730	658.800	0.114	0.160	0.164	0.074	0.230	0.071	0.370
17	848406	M	14.680	20.130	94.740	684.500	0.099	0.072	0.074	0.053	0.159	0.059	0.473
18	84862001	M	16.130	20.680	108.100	798.800	0.117	0.202	0.172	0.103	0.216	0.074	0.569
19	849014	M	19.810	22.150	130	1260	0.098	0.103	0.148	0.095	0.158	0.054	0.758
20	8510426	B	13.540	14.360	87.460	566.300	0.098	0.081	0.067	0.048	0.189	0.058	0.270

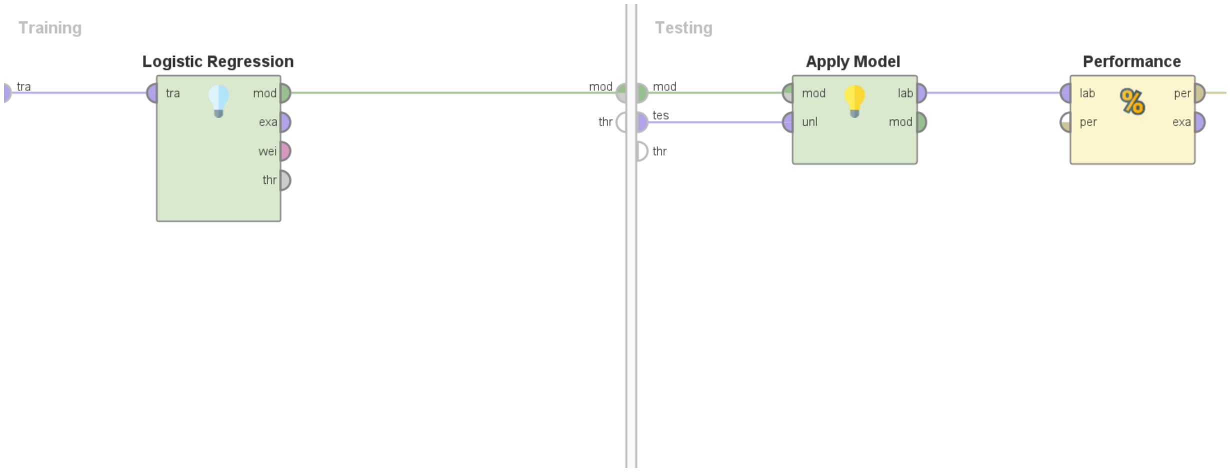
ExampleSet (569 examples,2 special attributes,30 regular attributes)



b. [8 points] Logistic Regression

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





ii. Screenshot for Logistic Regression Performance

PerformanceVector

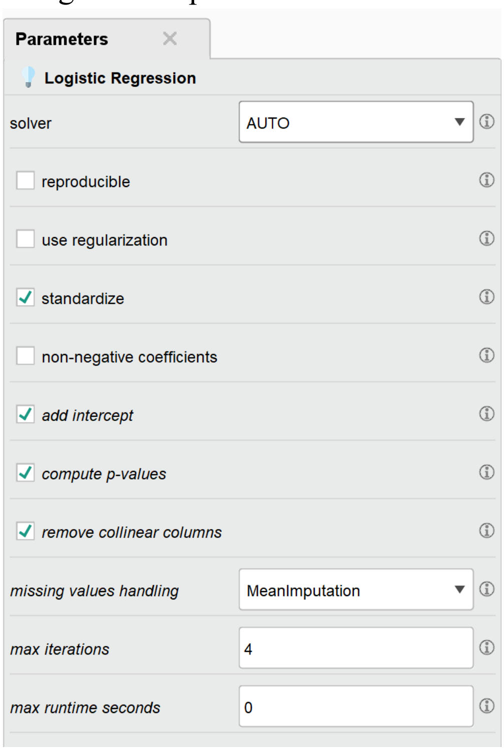
```
PerformanceVector:
accuracy: 96.49%
ConfusionMatrix:
True:   M       B
M:      71      4
B:      2       94
precision: 97.92% (positive class: B)
ConfusionMatrix:
True:   M       B
M:      71      4
B:      2       94
recall: 95.92% (positive class: B)
ConfusionMatrix:
True:   M       B
M:      71      4
B:      2       94
f_measure: 96.91% (positive class: B)
ConfusionMatrix:
True:   M       B
M:      71      4
B:      2       94
```

iii. Screenshot for Logistic Regression Results (Coefficients)

Attribute	Coefficient	Std. Coefficient	Std. Error	z-Value	p-Value
Mean Radius	2.827	9.963	5.780	0.489	0.625
Mean Texture	0.066	0.283	0.193	0.341	0.733
Mean Perimeter	-0.540	-13.118	0.836	-0.646	0.518
Mean Area	0.005 -21.931880393101192	1.713	0.026	0.190	0.849
Mean Smoothness	-21.199	-0.298	64.098	-0.331	0.741
Mean Compactness	49.982	2.640	40.453	1.236	0.217
Mean Concavity	-19.150	-1.527	34.447	-0.556	0.578
Mean Concave Points	-21.932	-0.851	55.665	-0.394	0.694
Mean Symmetry	4.394	0.120	20.738	0.212	0.832
Mean Fractal Dimension	3.279	0.023	163.607	0.020	0.984
Radius SE	-15.422	-4.277	12.269	-1.257	0.209
Texture SE	1.138	0.628	1.277	0.891	0.373
Perimeter SE	0.395	0.798	1.440	0.274	0.784
Area SE	0.057	2.573	0.099	0.571	0.568
Smoothness SE	-126.624	-0.380	206.070	-0.614	0.539
Compactness SE	-42.687	-0.764	73.026	-0.585	0.559
Concavity SE	39.240	1.185	43.217	0.908	0.364
Concave Points SE	-153.697	-0.948	194.760	-0.789	0.430
Symmetry SE	16.331	0.135	85.736	0.190	0.849
Fractal Dimension SE	554.618	1.468	602.610	0.920	0.357

Worst Radius	-1.107	-5.348	1.805	-0.613	0.540
Worst Texture	-0.297	-1.826	0.176	-1.687	0.092
Worst Perimeter	-0.014	-0.477	0.206	-0.069	0.945
Worst Area	0.007	3.853	0.015	0.451	0.652
Worst Smoothness	-7.566	-0.173	40.214	-0.188	0.851
Worst Compactness	7.552	1.188	11.318	0.667	0.505
Worst Concavity	-6.864	-1.432	8.776	-0.782	0.434
Worst Concave Points	-1.055	-0.069	26.861	-0.039	0.969
Worst Symmetry	-12.009	-0.743	12.326	-0.974	0.330
Worst Fractal Dimension	-82.094	-1.483	85.865	-0.956	0.339
Intercept	39.126	1.085	18.061	2.166	0.030

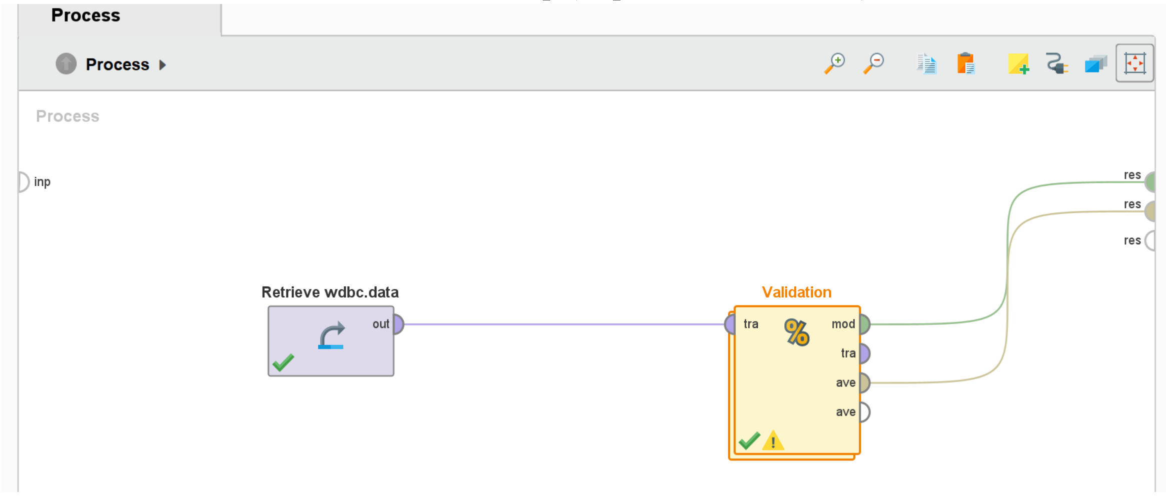
iv. Screenshot for Logistic Regression Rapidminer Operator Parameters (click on Logistic Regression operator and then take a screenshot of the Parameters window on the right)

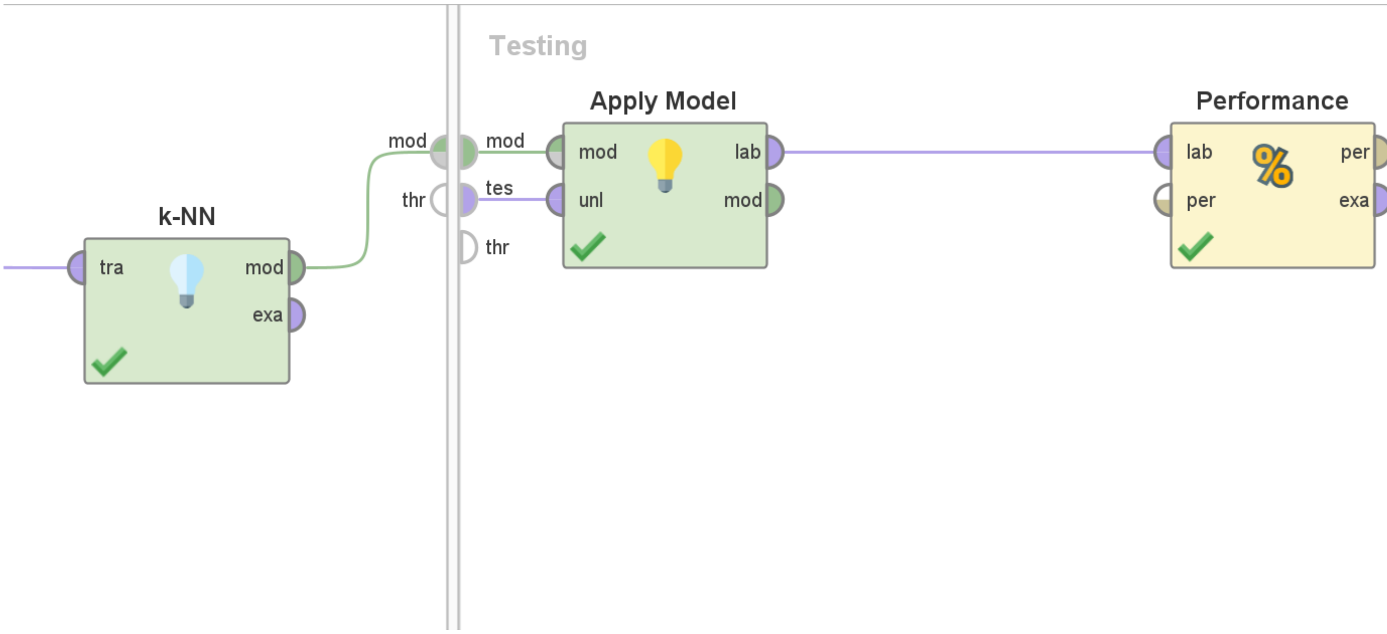


We plug in different numbers of iterations to test the highest accuracy we can get for this model. As a result, we found out that inputting the max iterations as 4 would give the highest accuracy.

c. **[8 points] kNN**

i. Screenshots for kNN Model Setup (Rapidminer Processes)





ii. Screenshot for kNN Performance

PerformanceVector

```
PerformanceVector:
accuracy: 92.98%
ConfusionMatrix:
True:   M       B
M:      70      9
B:      3       89
precision: 96.74% (positive class: B)
ConfusionMatrix:
True:   M       B
M:      70      9
B:      3       89
recall: 90.82% (positive class: B)
ConfusionMatrix:
True:   M       B
M:      70      9
B:      3       89
f_measure: 93.68% (positive class: B)
ConfusionMatrix:
True:   M       B
M:      70      9
B:      3       89
```

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