Homework #2 Part 2

(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.**
   a. **Summary Statistics of the breast-cancer-wisconsin data:**
   After load the data, we assigned the column name based on given information with the following code:

```
# Define the base features
base_features = ["radius", "texture", "perimeter", "area", "smoothness", "compactness", "concavity", "concave points",
    "symmetry", "fractal dimension"]

# Define the computations
computations = ["Mean", "SE", "Worst"]

# Create the header list
headers = ["ID number", "Diagnosis"]

# Append computed features to the headers
for computation in computations:
    for feature in base_features:
        headers.append(f"{computation} {feature}")

headers

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

   The table below is the corresponding outputs.

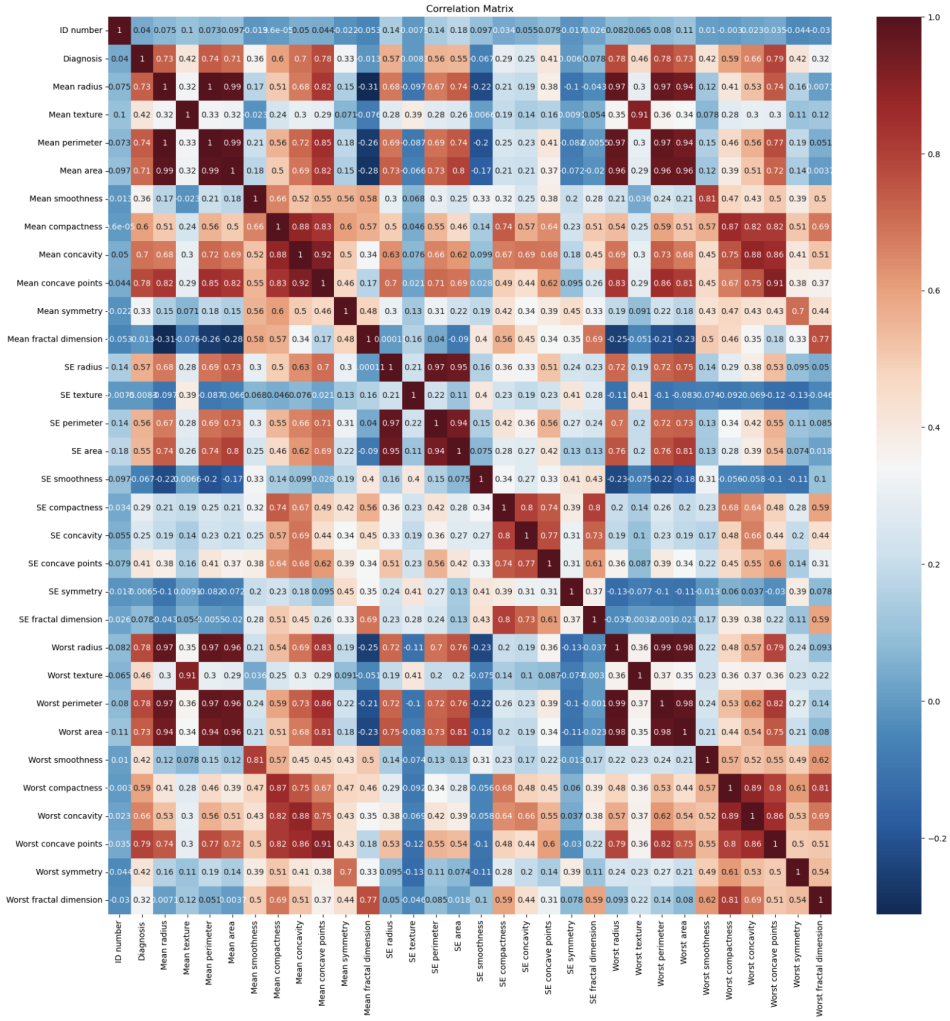
	ID number	Mean radius	Mean texture	Mean perimeter	Mean area	Mean smoothness	Mean compactness	Mean concavity	Mean concave points	Mean symmetry	...	Worst radius	Wo texti
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.000000	569.000(
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	...	16.269190	25.6772
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	...	4.833242	6.1462
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	...	7.930000	12.020(
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	...	13.010000	21.080(
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	...	14.970000	25.410(
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	...	18.790000	29.720(
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	...	36.040000	49.540(

   b. **Correlation matrix of the breast-cancer-wisconsin data:**

To provide a clearer understanding of the correlation matrix, we created the heatmaps. The corresponding codes and outputs are as below:

```
summary_stats = df.describe()
print("Summary Statistics:")
print(summary_stats)

correlation_matrix = df.iloc[:,2:].corr()  # Calculate the correlation matrix
correlation_matrix

plt.figure(figsize=(20, 20))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Matrix Heatmap")
plt.show()
```

Heatmap Results:



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.

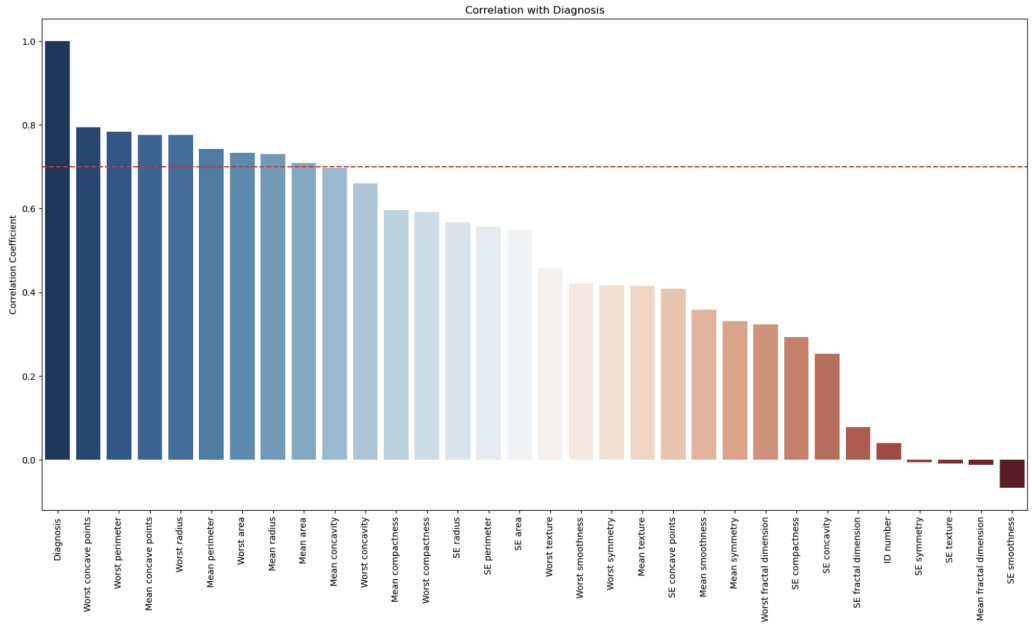
Validation Process:

1. Check Correlation with Target Variable:

After building the heated map to demonstrate the **correlation** between attributes and our target variable Diagnosis, we have found some attributes that are highly correlated. When independent variables are highly correlated, multicollinearity will be a problem. This can lead to unstable coefficient estimates,

making it difficult to determine the individual impact of predictors on the response. As shown in the following bar graph, if we determine certain thresholds, the variables with higher correlations could be eliminated or perform regularizations, including either L1 Lasso regression or L2 ridge. For convenience, we suspected to use a threshold of 0.7 as it is a conventional choice.

```
import matplotlib.pyplot as plt
import seaborn as sns

# Compute correlation of all columns with each other
full_correlation_matrix = df.corr()

# Select the 'Diagnosis' row or column from the full correlation matrix
correlation_with_diagnosis = full_correlation_matrix['Diagnosis'].sort_values(ascending=False)

# Plotting
plt.figure(figsize=(20, 10))
sns.barplot(x=correlation_with_diagnosis.index, y=correlation_with_diagnosis.values, palette="RdBu_r")

# Add a horizontal line at y=0.7
plt.axhline(y=0.7, color='r', linestyle='--')

plt.title("Correlation with Diagnosis")
plt.xticks(rotation=90)
plt.ylabel('Correlation Coefficient')
plt.show()
```



The screenshot demonstrates how we perform the validation for eliminating attributes that are highly correlated with our target variable.

```
correlation_with_diagnosis = df.corr()['Diagnosis']
columns_to_drop = correlation_with_diagnosis[correlation_with_diagnosis.abs() > 0.7].index.drop('Diagnosis')
df.drop(columns_to_drop, axis=1, inplace=True)
```

2. Detect Missing Values:

During the process, we performed a thorough evaluation of our datasets, paying close attention to any potential **missing values** in our datasets. This helps guarantee the completeness and accuracy of our dataset, consequently avoiding mistakes in any future analysis or modeling using the dataset.

After performing a thorough evaluation of our datasets, we can see, as shown in the table, there is no missing value included in our given dataset, which means our data is in a good position to be used in our models.

```
# 1.1 Handling missing values
# For this dataset, there aren't any missing values. But let's check to be sure.
missing_values = df.isnull().sum()
```

```
missing_values
```

```
ID number                  0
Diagnosis                  0
Mean radius                0
Mean texture               0
Mean perimeter             0
Mean area                  0
Mean smoothness            0
Mean compactness           0
Mean concavity             0
Mean concave points        0
Mean symmetry              0
Mean fractal dimension     0
SE radius                  0
SE texture                 0
SE perimeter               0
SE area                    0
SE smoothness              0
SE compactness             0
SE concavity               0
SE concave points          0
SE symmetry                0
SE fractal dimension       0
Worst radius               0
Worst texture              0
Worst perimeter            0
Worst area                 0
Worst smoothness           0
Worst compactness          0
Worst concavity            0
Worst concave points       0
Worst symmetry             0
Worst fractal dimension    0
dtype: int64
```

Parameters used:

For the k-NN model, the following parameters were used:

- n_neighbors: 3 (This means the model looks at the three nearest points to make a prediction.)
- metric: Minkowski (This is a generalization of the Euclidean and Manhattan distances.)
- p: 2 (With p=2, the Minkowski metric becomes the Euclidean distance.)
- weights: Uniform (This means all points in the neighborhood are weighted equally.)

For the Logistic Regression model, we used:

- max_iter: 10000 (This is the maximum number of iterations for the solvers to converge.)
- random_state: 42 (For reproducibility.)

Estimated Coefficients of the Logistic Regression:

The weights of the attributes are: [[-3.52934663 -0.09482721 0.58631845 -0.0246688 0.13048482 0.64178528
0.8910687 0.37953675 0.17469563 0.04902986 -0.11077869 -1.58405815
0.13606778 0.11972166 0.01855867 0.13044177 0.18841705 0.05151257
0.06434593 0.01356018 -3.33077087 0.30646393 0.12944361 0.05161542
0.23831075 1.87744164 2.36375563 0.69796482 0.70934228 0.19373244]]

The weights of the intercepts are: [-0.58020583]

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.

k-NN performance:

```
Accuracy (out-of-sample): 0.96
Accuracy (in-sample): 0.99
F1 score (out-of-sample):  0.9555629802873371
F1 score (in-sample)    :  0.9864973978653675
Kappa score (out-of-sample):  0.9112083673318003
Kappa score (in-sample)    :  0.9729964447580536
              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
```

Logistic Regression:

```
Accuracy (in-sample): 0.99
Accuracy (out-of-sample): 0.97
F1 score (in-sample): 0.9821428571
F1 score (out-of-sample): 0.9647058824
Kappa score (in-sample): 0.9716904826
Kappa score (out-of-sample): 0.9437314906

Detailed Classification Report (out-of-sample):

              precision    recall  f1-score   support

           B       0.97      0.99      0.98        71
           M       0.98      0.95      0.96        43

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

Accuracy rate is the proportion of correct diagnostics in the dataset. In our example, 97% of out-of-sample instances are correctly classified by the logistic regression model, and 96% of out-of-sample instances are correctly classified by the KNN model. Both the KNN model and the logistic regression model have good accuracy rates, while logistic regression performs better in the out-of-sample accuracy. In addition, both models are not overfitting in this case since the difference in model accuracy between the in-sample and out-of-sample is minor.

Recall (malignant) in the mathematical perspective, $TP/(TP + FN)$, provides the proportion of actual positives that were correctly classified by the model; Recall (benign) in the mathematical perspective, $TN/(TN + FP)$, provides the proportion of actual negatives that were correctly classified by the model; The higher recall score indicates that the model classified most of the positive(negative) samples correctly.

According to the outcome in the K-NN model, a 99% recall for the "benign" classification means that 99% of the actual benign samples were correctly identified and classified as benign by the model, and only 1% of the benign samples were missed or incorrectly classified as malignant. A 91% recall for the "malignant" classification means that 91% of the actual malignant samples were correctly identified and classified as

malignant by the model, and only 9% of the malignant samples were missed or incorrectly classified as benign.

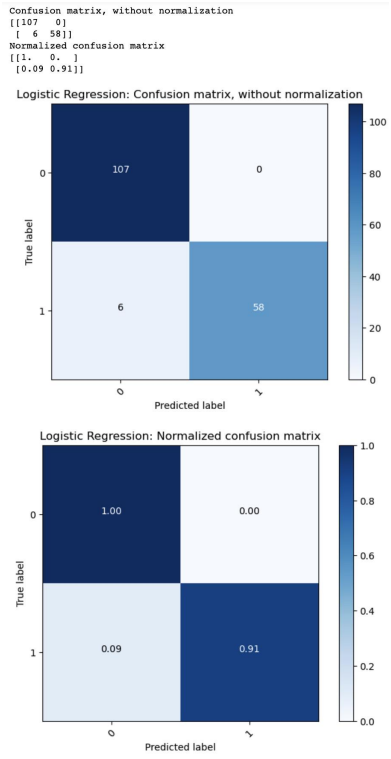
As for the logistic model, a 99% recall for the "benign" classification means that 99% of the actual benign samples were correctly identified and classified as benign by the model, and only 1% of the benign samples were missed or incorrectly classified as malignant. A 95% recall for the "malignant" classification means that 95% of the actual malignant samples were correctly identified and classified as malignant by the model, and only 5% of the malignant samples were missed or incorrectly classified as benign.

As a result, the logistic regression model performs better in the out-of-sample recall metric.

Precision (positive) measures how many of the positive predictions made by a classification model are actually positive. It's a ratio of correctly predicted positive observations to the total predicted positives. Both the KNN model and logistic regression model have a precision (malignant) of 98%. In other words, when the model predicts a patient's cancer is malignant, there's a 98% chance that the cancer is truly being malignant. The KNN model has a precision (benign) of 95%. In other words, when the model predicts a patient's cancer is benign, there's a 95% chance that the cancer is truly benign. The logistic regression model has a precision (benign) of 97%. In other words, when the model predicts a patient's cancer is benign, there's a 97% chance that the cancer is truly benign. In our example, the logistic regression model performs better in the out-of-sample precision metric.

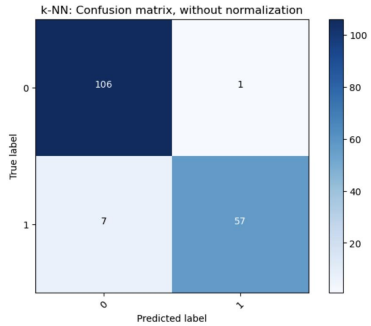
F1 score integrates both recall and precision. KNN has a 95% out-of-sample F1 score, and logistic regression has a 96% out-of-sample F1 score, so logistic regression performs slightly better than the K-NN model in terms of out-of-sample F1 score in terms of achieving a harmonious balance between precision and recall on unseen data.

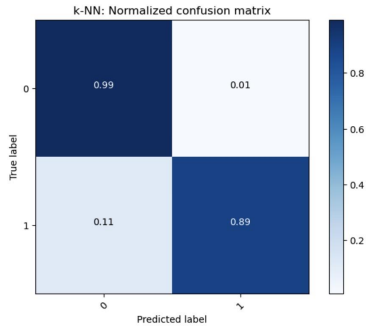
Confusion Matrix:



In the **Logistic Regression model**, 100% of instances predicted are True Positive, 0% of instances predicted are False Positive, 9% of instances predicted are False Negative, and 91% of instances predicted are True Negative.

```
Confusion matrix, without normalization
[[106   1]
 [  7  57]]
Normalized confusion matrix
[[0.99 0.01]
 [0.11 0.89]]
```





In the **KNN model**, 99% of instances predicted are True Positive, 1% of instances predicted are False Positive, 11% of instances predicted are False Negative, and 89% of instances predicted are True Negative.

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.

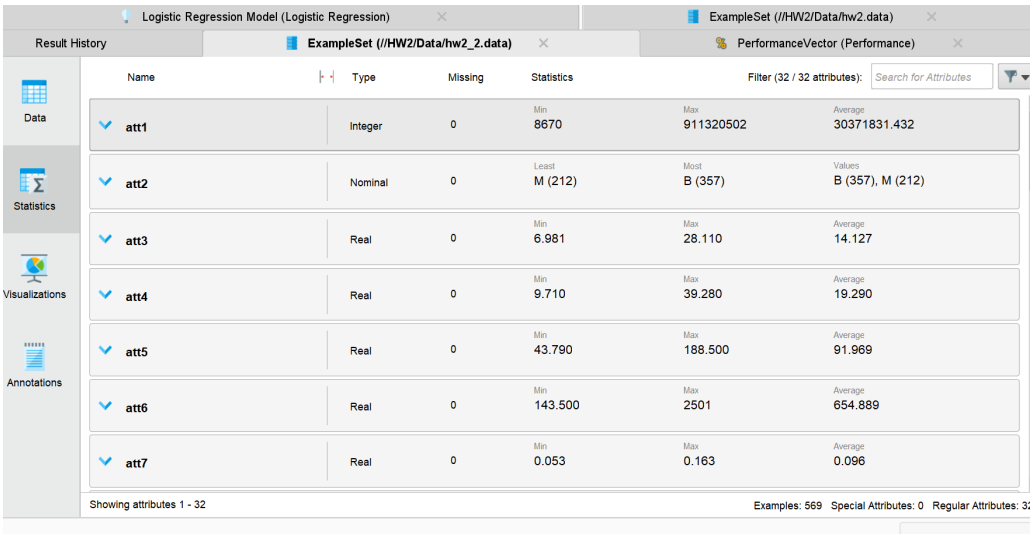
In this context, we prioritize using recall to select the best-performing model.
While we are dealing with a cancer diagnosis, if our model consistently under-identifies actual cancer cases, it could lead to a significant number of patients not receiving timely treatment, with potentially life-threatening outcomes.

Missing a positive diagnosis (false negative) would be far more costly than a false positive. A low recall would mean the model has missed many positive cases, resulting in a high volume of false negatives, so we want to maximize the recall of a model. When cancer is actually malignant but diagnosed as benign, the consequences can be fatal. Given the high costs associated with such misdiagnoses, it's imperative that we emphasize **recall** as the generalization performance metric.

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**

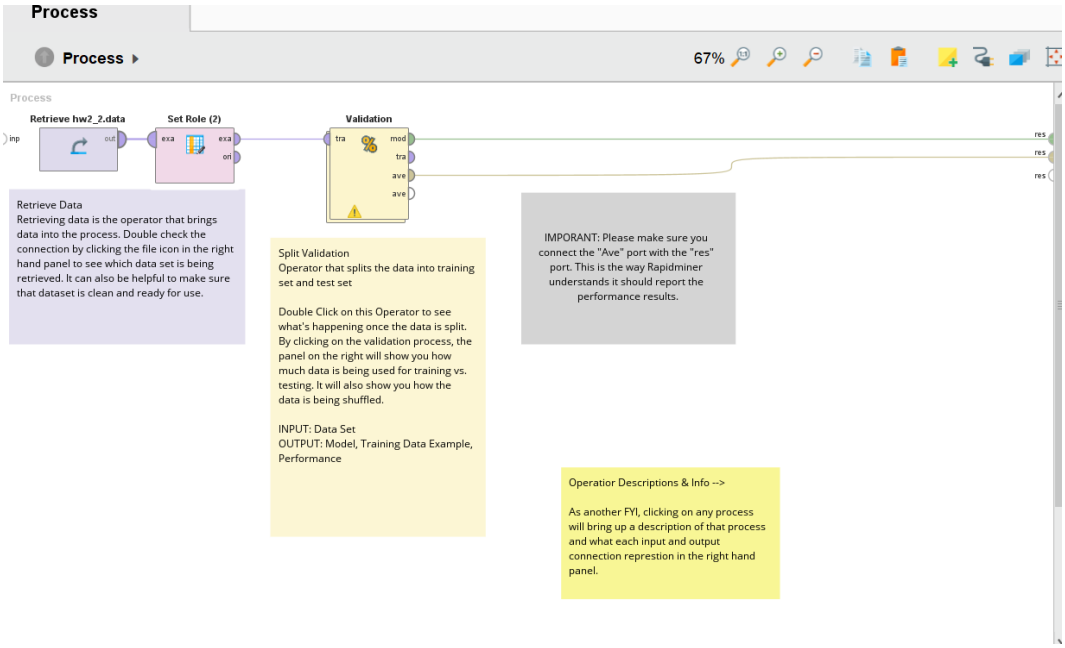
i.

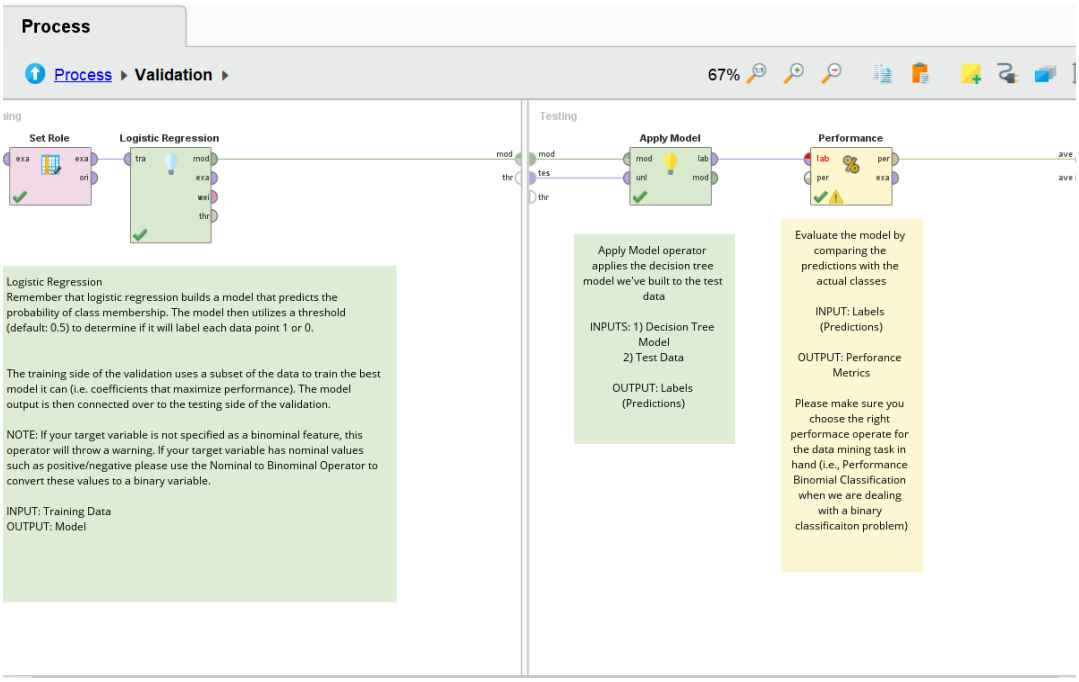


b. [8 points] Logistic Regression

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

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





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

Accuracy

Table View Plot View

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%	

Precision

Table View Plot View

precision: 97.80% (positive class: B)

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

Recall

Table View Plot View

recall: 90.82% (positive class: B)

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

F1 Measure

Table View Plot View

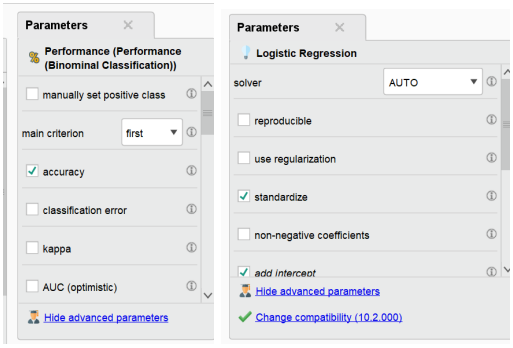
f_measure: 94.18% (positive class: B)

	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 Shreenshot here)

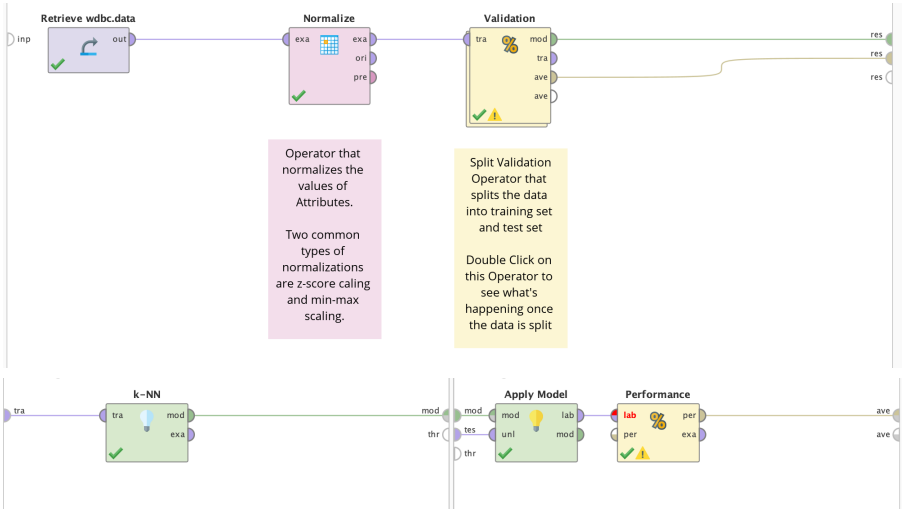
Attribute	Coefficient
att1	0.000
att3	336.976
att4	-6.534
att5	4.266
att6	-3.550
att7	-4144.690
att8	3285.021
att9	-1728.287
att10	-2281.412
att11	1580.948
att12	-2736.834
att13	15.275
att14	41.495
att15	72.582
att16	-11.486
att17	-1791.856
att18	-6098.389
att19	5211.568
att20	-22867.640
att21	10288.291
att22	57312.933
att23	-113.308
att24	-7.845
att25	-6.395
att26	1.096
att27	1561.472
att28	534.299
att29	-506.592
att30	-497.763
att31	-1779.926
att32	-5162.489
Intercept	-5.372

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 Shreenshot here)



c. [8 points] kNN

i. Screenshots for kNN Model Setup (Rapidminer Processes)



ii. Screenshot for kNN Performance

accuracy: 98.83%

	true M	true B	class precision
pred. M	72	1	98.63%
pred. B	1	97	98.98%
class recall	98.63%	98.98%	

precision: 98.98% (positive class: B)

	true M	true B	class precision
pred. M	72	1	98.63%
pred. B	1	97	98.98%
class recall	98.63%	98.98%	

recall: 98.98% (positive class: B)

	true M	true B	class precision
pred. M	72	1	98.63%
pred. B	1	97	98.98%
class recall	98.63%	98.98%	

f_measure: 98.98% (positive class: B)

	true M	true B	class precision
pred. M	72	1	98.63%
pred. B	1	97	98.98%
class recall	98.63%	98.98%	

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

Analysis and Interpretation:

Accuracy: Accuracy represents the overall correctness of the model across both categories (Malignant 'M' and Benign 'B'). The k-NN model demonstrates a superior accuracy of 98.83% compared to the Logistic Regression's 93.57%. This suggests that k-NN is better equipped to correctly classify both malignant and benign tumors in the given dataset.

Precision (for Malignant 'M'): Precision is particularly critical when the cost of a false positive is high. In this context, a false positive would mean incorrectly classifying a benign tumor as malignant. While both models exhibit commendable precision, the k-NN model, with a precision of 98.98%, slightly surpasses the Logistic Regression model's 97.8%. This means that the k-NN model has a higher likelihood of correctly identifying malignant tumors when it predicts them.

Recall (for Malignant 'M'): Recall becomes crucial when the cost of a false negative is significant. In medical scenarios like cancer detection, a false negative (failing to identify a malignant tumor) can have grave consequences. The k-NN model has a recall of 98.98%, outperforming the Logistic Regression model's 90.82%. This suggests that the k-NN model is more adept at capturing all the malignant cases present in the dataset, reducing the risk of missed malignant tumors.

F-Score: The F-Score provides a balance between precision and recall, especially vital when the datasets are imbalanced. A higher F-Score for k-NN (98.98%) compared to Logistic Regression (94.18%) indicates that k-NN maintains a superior balance between reducing both false positives and false negatives for malignant tumors.

Conclusion : For the Wisconsin Diagnostic Breast Cancer dataset, the k-NN model consistently outperforms the Logistic Regression model across all evaluated metrics. Its outstanding performance in both precision and recall ensures that not only are most malignant tumors correctly identified (high recall), but also that there's a very low likelihood of falsely classifying benign tumors as malignant (high precision). This balanced performance is further underscored by the k-NN model's high F-Score. While the Logistic Regression model still provides robust results, in this analysis, the k-NN model emerges as the more reliable classifier for distinguishing between malignant (M) and benign (B) tumors.

Appendix (Additional Code):

```
# Define the headers
base_features = [
    "radius",
    "texture",
    "perimeter",
    "area",
    "smoothness",
    "compactness",
    "concavity",
    "concave points",
    "symmetry",
    "fractal dimension"
]

# Define the computations
computations = ["Mean", "SE", "Worst"]

# Create the header list
headers = ["ID number", "Diagnosis"]

# Append computed features to the headers
for computation in computations:
    for feature in base_features:
        headers.append(f"{computation} {feature}")

headers

df.columns = headers

df.head()

# Retrieve features/attributes of dataset iris
X = df.iloc[:, 2:32]

# Retrieve target variable of dataset
y= df.iloc[:,1]


# SPLIT + NORMALIZATION + looking for missing values

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder

# 1.1 Handling missing values
# For this dataset, there aren't any missing values. But let's check to be sure.
missing_values = df.isnull().sum()

# 1.2 Convert Categorical to Numerical
```

```
encoder = LabelEncoder()
df['Diagnosis'] = encoder.fit_transform(df['Diagnosis'])  # M will be 1 and B will be 0

correlation_with_diagnosis = df.corr()['Diagnosis']
columns_to_drop = correlation_with_diagnosis[correlation_with_diagnosis.abs() > 0.7].index.drop('Diagnosis')
df.drop(columns_to_drop, axis=1, inplace=True)

# 1.3 Split Data
X = df.iloc[:, 2:].values  # Features (excluding ID and Diagnosis)
y = df['Diagnosis'].values  # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# 1.4 Standardize Features
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)

missing_values


############################################ Distribution Target Variable
#########################################

# Count how many data points we have for each label of the target variable
print('Labels counts in y:', np.bincount(y))            # The bincount function from the numpy library counts the
number of occurrences of each value in the input array.
                                        # Counts how many samples belong to each class in the target variable y (the
original dataset before splitting).
print('Labels counts in y_train:', np.bincount(y_train)) # Counts the occurrences of each class in the training subset
(y_train) of the target variable.
print('Labels counts in y_test:', np.bincount(y_test))

############################################### Train the Model
###############################################

# Import the Necessary Library
from sklearn import neighbors

# Setting up the kNN Classifier
# Code initializes a kNN classifier with specific parameters
knn = neighbors.KNeighborsClassifier(n_neighbors=3,        # n_neighbors is the k in the kNN
                            p=2,                # power parameter for the Minkowski metric.
                            metric='minkowski',    # the default metric is minkowski, which is a generalization of the
Euclidean distance
                                        # with p=2 is equivalent to the standard Euclidean distance.
                                        # with p=1 is equivalent to the Mahattan distance.
                            n_jobs=-1,          # the number of parallel jobs to run for neighbors search; -1 means using
all processors
```

```
                              weights='uniform')     # all points in each neighborhood are weighted equally (default). We would choose 'distance' if
                                                     # we wanted to apply a similarity-moderated kNN
# Train the model
knn = knn.fit(X_train_std, y_train)

############################################### Evaluate the Model
###############################################
from sklearn.metrics import accuracy_score, f1_score, cohen_kappa_score, classification_report #The required functions for evaluation metrics are imported

# Estimate the predicted values by applying the kNN algorithm
y_pred = knn.predict(X_test_std)           # make predictions for test set
                                  # The classifier (knn) makes predictions on the standardized test data
y_pred_insample = knn.predict(X_train_std)  # make predictions for train set (for educational purposes, to demonstrate the difference)
                                  # The classifier (knn) makes predictions on the standardized train data

############################################### Evaluate the Model
###############################################
from sklearn.metrics import accuracy_score, f1_score, cohen_kappa_score, classification_report #The required functions for evaluation metrics are imported

# Estimate the predicted values by applying the kNN algorithm
y_pred = knn.predict(X_test_std)           # make predictions for test set
                                  # The classifier (knn) makes predictions on the standardized test data
y_pred_insample = knn.predict(X_train_std)  # make predictions for train set (for educational purposes, to demonstrate the difference)
                                  # The classifier (knn) makes predictions on the standardized train data

# Accuracy
print('Accuracy (out-of-sample): %.2f' % accuracy_score(y_test, y_pred))        # Accuracy is calculated and printed for both the test (out-of-sample) dataset.
print('Accuracy (in-sample): %.2f' % accuracy_score(y_train, y_pred_insample))  # Accuracy is calculated and printed for both the training (in-sample) dataset.

# F1 score
# The F1 score is a harmonic mean of precision and recall, providing a balance between the two metrics
print('F1 score (out-of-sample): ', f1_score(y_test, y_pred, average='macro'))          # average='macro' calculate metrics for each label, and find their unweighted mean
print('F1 score (in-sample)    : ', f1_score(y_train, y_pred_insample, average='macro')) # The average='macro' argument calculates the metric independently for each class and then takes the average, not considering label imbalance.

# Kappa score
# Cohen's Kappa score measures the agreement between the predictions and the actual values, accounting for the possibility of agreement occurring by chance
# It's especially useful when the classes are imbalanced
print('Kappa score (out-of-sample): ', cohen_kappa_score(y_test, y_pred))              # computes Cohen’s kappa: a statistic that measures inter-annotator agreement
```

```
print('Kappa score (in-sample)    : ', cohen_kappa_score(y_train, y_pred_insample))      # (i.e., agreement between predictions and actual values of target variables)

# Build a text report showing the main classification metrics (out-of-sample performance)
# classification_report function provides a comprehensive report displaying key metrics

target_names = np.unique(y_test).astype(str).tolist()

print(classification_report(y_test, y_pred, target_names=target_names))         # builds a text report showing the main classification metrics (precision, recall, f1-score)
                                                    # The target_names argument provides names for each of the classes

############################################## Logistic Regression ################################

from __future__ import division, print_function, unicode_literals

# Import necessary libraries and modules
# Matplotlib inline allows the output of plotting commands will be displayed inline (within the notebook)
%matplotlib inline
from sklearn.datasets import load_iris  # iris dataset
                                # imports the load_iris function from the sklearn.datasets module
from sklearn import linear_model        # the sklearn.linear_model module implements generalized linear models.


from sklearn.model_selection import train_test_split # splits arrays or matrices into random train and test subsets
from sklearn.metrics import accuracy_score, f1_score, classification_report # the sklearn.metrics module includes performance metrics

##############################################    Split the Data
##############################################

# Split validation
# train_test_split is used to split the dataset into training (X_train, y_train) and testing (X_test, y_test) subsets
X_train, X_test, y_train, y_test = train_test_split(X,  # dataset to be split ; X represents the feature matrix
                                    y,  # dataset to be split ; y represents the target variable

                                    test_size=0.4,  # a float number between 0.0 and 1.0 representing the proportion of the dataset to include in the test split
                                                # test_size=0.4 specifies that 40% of the data will be used for testing, and the remaining 60% for training
                                    random_state=1, # controls the shuffling for reproducible output

                                    stratify=y)

from sklearn.linear_model import LogisticRegression

clf = linear_model.LogisticRegression(multi_class = 'auto', C=1e5, solver = 'lbfgs', max_iter=100)

clf = clf.fit(X_train, y_train)                         # model induction using the train data
```

```
print('The weights of the attributes are:', clf.coef_)      # reports coefficients of the features in the decision function
                                                    # the coefficients in clf.coef_ are printed in the same order as the columns of the input feature matrix X
                                                    # these coefficients represent the weight or importance of each feature in the logistic regression model's decision function
print('The weights of the intercepts are:', clf.intercept_) # reports intercepts in the decision function

##################################### Apply the Logistic Regression Model
#####################################

# We now apply the trained logistic regression model to the test set
y_pred = clf.predict(X_test)            # generate classification prediction and store them in y_pred
                                # in scikit-learn's LogisticRegression the default threshold for the .predict() method is 0.5
y_pred_prob = clf.predict_proba(X_test)  # estimate class probabilities

# Print the first elements of the arrays containing predictions, predicted class probabilities,
# and the sum of predicted probabilities for the first test sample
print('The predictions are:', y_pred[0], y_pred_prob[0], np.sum(y_pred_prob[0])) # prints first elements of arrays

################################### Evaluate the Logistic Regression Model
###################################

# Build a text report showing the main classification metrics (out-of-sample performance)
print(classification_report(y_test, y_pred, target_names=target_names)) # builds a text report showing the main classification metrics
                                                        # (such as precision, recall, f1-score)
```

 # confusion matrix

```
from sklearn.metrics import confusion_matrix
import itertools

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)
    plt.title(title)
```

```
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

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

# Compute confusion matrix to evaluate the accuracy of a classification
cnf_matrix = confusion_matrix(y_test, y_pred)
#Determine the way floating point numbers are displayed
np.set_printoptions(precision=2)
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix,
                      classes= np.unique(y_test).astype(str).tolist(),
                      title='k-NN: Confusion matrix, without normalization')

# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix,
                      classes=np.unique(y_train).astype(str).tolist(),
                      normalize=True,
                      title='k-NN: Normalized confusion matrix')

plt.show()

from sklearn.linear_model import LogisticRegression

# Train the logistic regression model
log_reg = LogisticRegression(random_state=42, max_iter=10000)   # You can adjust the hyperparameters as
needed
log_reg.fit(X_train_std, y_train)

# Predict using the logistic regression model
y_pred_logistic = log_reg.predict(X_test_std)

# Compute confusion matrix for the logistic regression model
cnf_matrix_logistic = confusion_matrix(y_test, y_pred_logistic)

# Plot non-normalized confusion matrix for logistic regression
plt.figure()
```

```
plot_confusion_matrix(cnf_matrix_logistic,
                      classes=np.unique(y_test).astype(str).tolist(),
                      title='Logistic Regression: Confusion matrix, without normalization')

# Plot normalized confusion matrix for logistic regression
plt.figure()
plot_confusion_matrix(cnf_matrix_logistic,
                      classes=np.unique(y_train).astype(str).tolist(),
                      normalize=True,
                      title='Logistic Regression: Normalized confusion matrix')

plt.show()
```

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 recoded with four significant digits.

5. Missing attribute values: none

6. Class distribution: 357 benign, 212 malignant