**Comparison between R and Python codes**

**(Ongoing)**

|  |  |  |
| --- | --- | --- |
| **Function** | **R** | **Python** |
| Install library | install.packages(‘#library’) | pip install #library |
| Import library | library(#library) | import #library as #abbreviation |
| Use library | #library::#function | #abbreviation.#function |
| Read file | read.csv(‘#file’) | pd.read\_csv(‘#file’, header = None) |
| Read head | head(#df, #num) | #df.head(#num) |
| Read tail | tail(#df, #num) | #df.tail(#num) |
| Describe | summary(#df)  glimpse(#df)  str(#df) | #df.describe() |
| Count NA |  | #df.isnull().sum() |
|  |  |  |
| Filter | #df %>%  dplyr::filter(#condition) |  |
| Replace string |  | #df.replace('#string', np.NaN) |
| Impute numeric value |  | #df.fillna(#df.mean(), inplace=True) |
| Impute categorical value |  | for col in #df.columns:  # Check if the column is of object type  if #df [col].dtypes == 'object':  # Impute with the most frequent value  #print(#df[col].value\_counts().index[0])  #df = #df.replace(np.NaN, #df [col].value\_counts().index[0])  # Count the number of NaNs in the dataset and print the counts to verify  #df.isnull().sum() |
| Encoding (Convert non-numeric to numeric) |  | # Import LabelEncoder  import sklearn  from sklearn.preprocessing import LabelEncoder  # Instantiate LabelEncoder  le = LabelEncoder()  # Iterate over all the values of each column and extract their dtypes  for col in #df.columns:  # Compare if the dtype is object  if #df[col].dtypes == 'object':  # Use LabelEncoder to do the numeric transformation  le.fit(#df[col])  #df[col]=le.transform(#df[col]) |
| Min-Max Scaling |  | # Import MinMaxScaler  from sklearn.preprocessing import MinMaxScaler  # Instantiate MinMaxScaler and use it to rescale X\_train and X\_test  scaler = MinMaxScaler(feature\_range = (0, 1))  rescaledX\_train = scaler.fit\_transform(X\_train)  rescaledX\_test = scaler. fit\_transform(X\_test) |
| Train Test Split |  | # Import train\_test\_split  from sklearn.model\_selection import train\_test\_split  # Drop the features 11 and 13 and convert the DataFrame to a NumPy array  #df = #df.drop([11, 13], axis=1).values  # Segregate features and labels into separate variables  X,y = #df[:,0:12] , #df[:,13]  # Split into train and test sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42) |
| Logistic Regression |  | # Import LogisticRegression  from sklearn.linear\_model import LogisticRegression  # Instantiate a LogisticRegression classifier with default parameter values  logreg = LogisticRegression()  # Fit logreg to the train set  logreg.fit(rescaledX\_train, y\_train) |
| Evaluation for Logistic Regression |  | # Import confusion\_matrix  from sklearn.metrics import confusion\_matrix  # Use logreg to predict instances from the test set and store it  y\_pred = logreg.predict(rescaledX\_test)  # Get the accuracy score of logreg model and print it  print("Accuracy of logistic regression classifier: ", ...)  # Print the confusion matrix of the logreg model  print(logreg.score(rescaledX\_test, y\_test))  confusion\_matrix(y\_test, y\_pred) |
| GridSearch |  | # Import GridSearchCV  from sklearn.model\_selection import GridSearchCV  # Define the grid of values for tol and max\_iter  tol = [0.01, 0.001, 0.0001]  max\_iter = [100, 150, 200]  # Create a dictionary where tol and max\_iter are keys and the lists of their values are corresponding values  param\_grid = dict(tol = tol, max\_iter = max\_iter)  # Instantiate GridSearchCV with the required parameters  grid\_model = GridSearchCV(estimator = logreg, param\_grid = param\_grid, cv = 5)  # Use scaler to rescale X and assign it to rescaledX  rescaledX = scaler. fit\_transform (X)  # Fit data to grid\_model  grid\_model\_result = grid\_model. fit(rescaledX, y)  # Summarize results  best\_score, best\_params = grid\_model\_result.best\_score\_, grid\_model\_result.best\_params\_  print("Best: %f using %s" % (best\_score, best\_params)) |
|  |  |  |
| Save file | write.csv(#df, ‘#file’) | #df.to\_csv(‘#file’) |