DECISION TREE

**Description:** An algorithm that examines the password and determines / decides if it is weak based on the following criterias:

DECISION TREE INDEPENDENT FACTORS / VARIABLES

* Whether people with birth dates in the database entered their age as a password

(is not possible! Because **Active Directory or RADIUS does not provide any attributes to store instant messengers, birth date, anniversary, additional addresses, profession, hobbies, gender etc. To add this attributes typically you will need to customize the Active Directory schema which is a complicated operation that requires significant affords.**

Reference: <https://www.carddavserver.com/features/create_attribute_ad/#:~:text=Active%20Directory%20does%20not%20provide,operation%20that%20requires%20significant%20affords>.)

* Some or all of the name and surname of the password owner
* Sample Password List that consists of the potential password typing scenarios that can be guessed by the attacker human or bot (for example, “1q2w3e4r5t6y7u8ı9o0p\*ğ-ü” according to Turkish-Q keyboard layout)

**Plain Text:**

1. Start
2. Parameters:

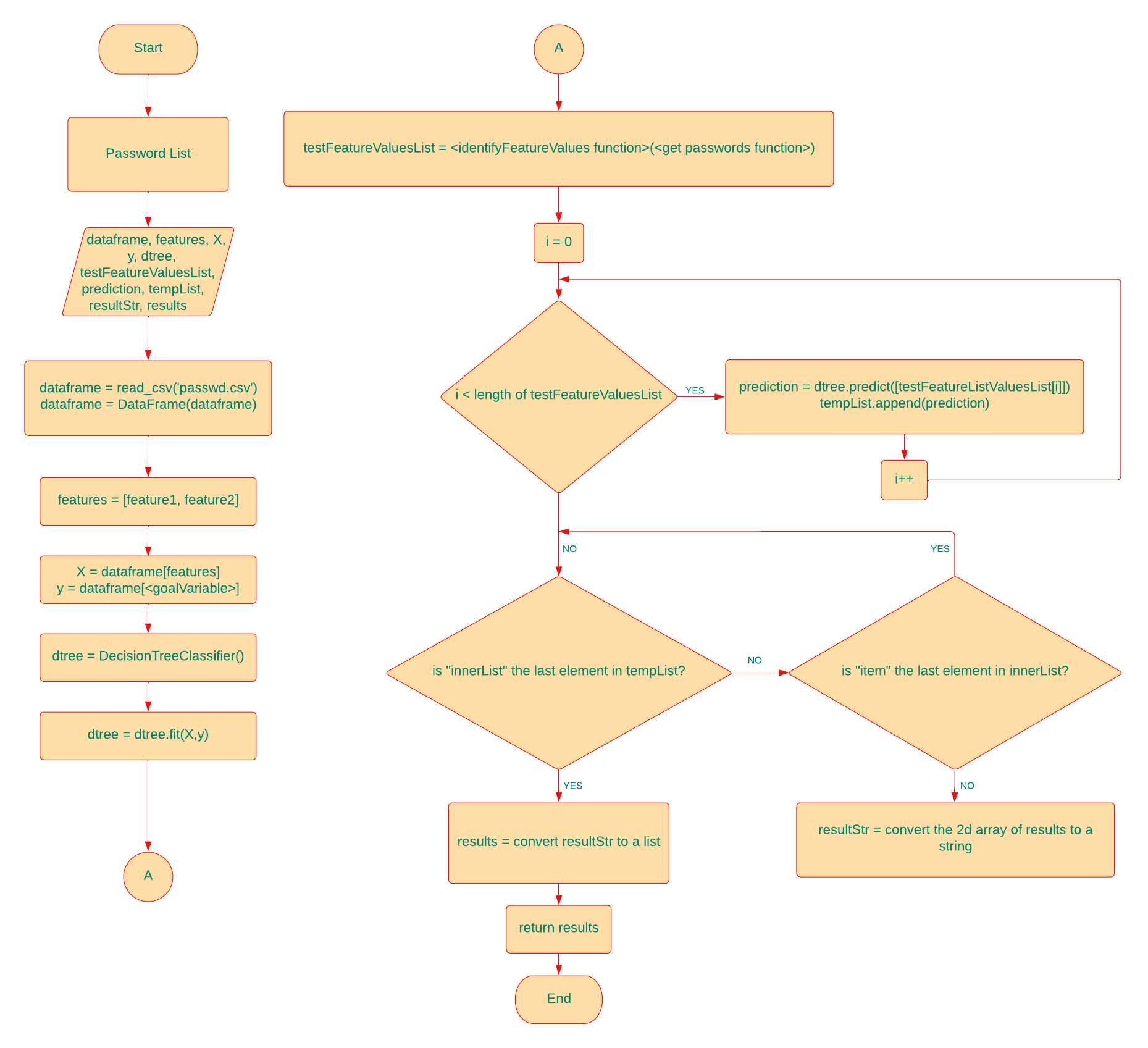
* **Password List** fetched from the user authentication database of the enterprise network in phase 1

1. Declare:

* **dataframe** (pandas object) – which is a Pandas table has the train dataset
* **features** (String list) – which indicates the independent variables to be used by the tree
* **X** (dataframe column component object) - which is a variable representing the feature (independent value) columns that will be used as argument for the fit() method of Decision Tree Classifier in order to use in the decision process
* **y** (dataframe column component object) – which is a variable representing the target (dependent value) column that will be used as argument for the fit() method of Decision Tree in order to use in the prediction process
* **dtree** (sklearn.tree.DecisionTreeClassifier object) – which is our decision tree
* **testFeatureValuesList** (2D Integer Array) – which is a 2D array saving the independent values of Password List
* **results** (String) –which has the the strength value of each password taken as an argument to the decision tree

1. Read and assign the table in ‘passwd.csv’ file saving the train data into the **dataframe** variable.
2. Identify the independent variables in a list and assign them to the **features** variable.
3. Assign features columns of the **dataframe** intothe **X** variable**.**
4. Assign goal column of the **dataframe** intothe **y** variable**.**
5. Create a decision tree object by the sklearn.tree.DecisionTreeClassifier class and assign it to the **dtree** variable.
6. Train the model using *fit()* function of DecisionTreeClassifier class, use **X** and **y** variables as arguments. Then assign the trained tree model to **dtree** variable again.
7. Identify the feature values of each password in the password list taken as an argument to decision tree utilizing a function called ‘identifyFeatureValues()’ that you pre-made, then assign them as a list into **testFeatureValuesList** variable.
8. Predict the strength value of each password using the *predict()* function of DecisionTreeClassifier class with our **dtree** object and assign the resultant list into the **results** variable converting the list to String.
9. Return the **results**

**Flow Chart:**



LOGISTIC REGRESSION

**Description:** An algorithm that examines the password and determines / decides if it is weak based on a train dataset which is a pre-made password list.

**Plain Text:**

1. Start
2. Parameters: password list
3. Declare:
   * **dataframe** (pandas object) – which is a Pandas table has the train dataset (pre-made password list for fitting / training the model)
   * **passwords** (2d array) –a rank 2 array savingthe data in the dataframe, in other words saving the passwords and their strength values in the pre-made password list
   * **allpasswords** (String list) - which represents the independent variable column (actual passwords).
   * **X** (spmatrix) - which is a variable representing the feature (independent) columns’ values that will be used as argument for the fit() method of Logistic Regression Classifier in order to use in the decision process
   * **y** (list) – which is a variable representing the target (dependent) column values that will be used as argument for the fit() method of Logistic Regression in order to use in the prediction process
   * **vectorizer** (TfidVectorizer object) – which transforms the passwords that are a token /string to a linear algebra vector to prepare for logistic regression
   * **X\_train, y\_train, X\_test, y\_test** (respectively: spmatrix, list, spmatrix, list) –versions of **X** and **y** created for training and testing
   * **classifier** (LogisticRegression object)–which is our logistic regression classifier
   * **X\_predict, y\_predict** (in order: spmatrix, list) – prediction versions of **X** and **y**

4) Read and assign the table in ‘data.csv’ file saving the train data (pre-made password list) into the **dataframe** variable.

5) Convert the **dataframe** to a rank 2 array called ‘**passwords’.**

6)Shuffle the passwords in **passwords** list randomly for robustness.

7) Split the rank 2 **passwords** array to two as **y** which represents the target column (strength), and as **allpasswords** variable.

8) To prepare the passwords that are a token /string for logistic regression algorithm which is based on Linear Algebra, vectorize each password token using TfidVectorizer class.

To be able to do that, create a vectorizer instance from TfidVectorizer class and assign it to the vectorizer variable.

Vectorize each element of **allpasswords** calling *fit\_transform(vectorizer object)* function with our **vectorizer** instance**,** thenassign into the **X** variable step by step.

9) Split train data set and test dataset through cross validation using *train\_test\_split()* function and assign the values that return from the function respectively into **X\_train**, **X\_test**, **y\_train**, **y\_test** variables.

10) Create a logistic regression classifier instance from sklearn.linear\_model.LogisticRegression class and assign it to **classifier** variable.

11) Train the model using *fit()* function of LogisticRegression class, use **X\_train** and **y\_train** variables as arguments.

12) Score the model using *score()* function of LogisticRegression class, use **X\_test** and **y\_test** variables as arguments.

13) Assign the password list taken as argument to **X\_predict**, and vectorise its elements (passwords) using *transfrom()* function of TfidVectorizer class and our **vectorizer** instance.

14) Use *predict()* function of LogisticRegression class through our **classifier**, take **X\_Predict** as argument, and assign the result list into **y\_predict** variable

15)Return y\_predict

**Flow Chart:**

