Decision Tree Model

March 17, 2022

[1]:
```
import sys
# !{sys.executable} -m pip install -U pandas-profiling[notebook]
# !jupyter nbextension enable --py widgetsnbextension
# !pip install matplotlib
# !pip install graphviz
```

[2]:
```
import os
import numpy as np
import pandas as pd
```

[5]:
```
df= pd.read_csv(r'C:\Users\zdehg\Downloads\archive\DASS_data_21.02.19\data.
 ↪csv',  error_bad_lines=False, warn_bad_lines=False, sep=r'\t' )
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\util\_decorators.py:311:
ParserWarning: Falling back to the 'python' engine because the 'c' engine does
not support regex separators (separators > 1 char and different from '\s+' are
interpreted as regex); you can avoid this warning by specifying engine='python'.
  return func(*args, **kwargs)
C:\ProgramData\Anaconda3\lib\site-
packages\IPython\core\interactiveshell.py:3444: FutureWarning: The
warn_bad_lines argument has been deprecated and will be removed in a future
version.


  exec(code_obj, self.user_global_ns, self.user_ns)
C:\ProgramData\Anaconda3\lib\site-
packages\IPython\core\interactiveshell.py:3444: FutureWarning: The
error_bad_lines argument has been deprecated and will be removed in a future
version.


  exec(code_obj, self.user_global_ns, self.user_ns)
```

[23]:
```
from sklearn import tree
help(tree.DecisionTreeClassifier)
```

```
Help on class DecisionTreeClassifier in module sklearn.tree._classes:

class DecisionTreeClassifier(sklearn.base.ClassifierMixin, BaseDecisionTree)
```

```
 |  DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features=None, random_state=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None,
ccp_alpha=0.0)
 |
 |  A decision tree classifier.
 |
 |  Read more in the :ref:`User Guide <tree>`.
 |
 |  Parameters
 |  ----------
 |  criterion : {"gini", "entropy"}, default="gini"
 |      The function to measure the quality of a split. Supported criteria are
 |      "gini" for the Gini impurity and "entropy" for the information gain.
 |
 |  splitter : {"best", "random"}, default="best"
 |      The strategy used to choose the split at each node. Supported
 |      strategies are "best" to choose the best split and "random" to choose
 |      the best random split.
 |
 |  max_depth : int, default=None
 |      The maximum depth of the tree. If None, then nodes are expanded until
 |      all leaves are pure or until all leaves contain less than
 |      min_samples_split samples.
 |
 |  min_samples_split : int or float, default=2
 |      The minimum number of samples required to split an internal node:
 |
 |      - If int, then consider `min_samples_split` as the minimum number.
 |      - If float, then `min_samples_split` is a fraction and
 |        `ceil(min_samples_split * n_samples)` are the minimum
 |        number of samples for each split.
 |
 |      .. versionchanged:: 0.18
 |         Added float values for fractions.
 |
 |  min_samples_leaf : int or float, default=1
 |      The minimum number of samples required to be at a leaf node.
 |      A split point at any depth will only be considered if it leaves at
 |      least ``min_samples_leaf`` training samples in each of the left and
 |      right branches.  This may have the effect of smoothing the model,
 |      especially in regression.
 |
 |      - If int, then consider `min_samples_leaf` as the minimum number.
 |      - If float, then `min_samples_leaf` is a fraction and
 |        `ceil(min_samples_leaf * n_samples)` are the minimum
 |        number of samples for each node.
```

```
|
|       .. versionchanged:: 0.18
|          Added float values for fractions.
|
|   min_weight_fraction_leaf : float, default=0.0
|       The minimum weighted fraction of the sum total of weights (of all
|       the input samples) required to be at a leaf node. Samples have
|       equal weight when sample_weight is not provided.
|
|   max_features : int, float or {"auto", "sqrt", "log2"}, default=None
|       The number of features to consider when looking for the best split:
|
|           - If int, then consider `max_features` features at each split.
|           - If float, then `max_features` is a fraction and
|             `int(max_features * n_features)` features are considered at each
|             split.
|           - If "auto", then `max_features=sqrt(n_features)`.
|           - If "sqrt", then `max_features=sqrt(n_features)`.
|           - If "log2", then `max_features=log2(n_features)`.
|           - If None, then `max_features=n_features`.
|
|       Note: the search for a split does not stop until at least one
|       valid partition of the node samples is found, even if it requires to
|       effectively inspect more than ``max_features`` features.
|
|   random_state : int, RandomState instance or None, default=None
|       Controls the randomness of the estimator. The features are always
|       randomly permuted at each split, even if ``splitter`` is set to
|       ``"best"``. When ``max_features < n_features``, the algorithm will
|       select ``max_features`` at random at each split before finding the best
|       split among them. But the best found split may vary across different
|       runs, even if ``max_features=n_features``. That is the case, if the
|       improvement of the criterion is identical for several splits and one
|       split has to be selected at random. To obtain a deterministic behaviour
|       during fitting, ``random_state`` has to be fixed to an integer.
|       See :term:`Glossary <random_state>` for details.
|
|   max_leaf_nodes : int, default=None
|       Grow a tree with ``max_leaf_nodes`` in best-first fashion.
|       Best nodes are defined as relative reduction in impurity.
|       If None then unlimited number of leaf nodes.
|
|   min_impurity_decrease : float, default=0.0
|       A node will be split if this split induces a decrease of the impurity
|       greater than or equal to this value.
|
|       The weighted impurity decrease equation is the following::
|
```

```
 |          N_t / N * (impurity - N_t_R / N_t * right_impurity
 |                              - N_t_L / N_t * left_impurity)
 |
 |      where ``N`` is the total number of samples, ``N_t`` is the number of
 |      samples at the current node, ``N_t_L`` is the number of samples in the
 |      left child, and ``N_t_R`` is the number of samples in the right child.
 |
 |      ``N``, ``N_t``, ``N_t_R`` and ``N_t_L`` all refer to the weighted sum,
 |      if ``sample_weight`` is passed.
 |
 |      .. versionadded:: 0.19
 |
 |  min_impurity_split : float, default=0
 |      Threshold for early stopping in tree growth. A node will split
 |      if its impurity is above the threshold, otherwise it is a leaf.
 |
 |      .. deprecated:: 0.19
 |         ``min_impurity_split`` has been deprecated in favor of
 |         ``min_impurity_decrease`` in 0.19. The default value of
 |         ``min_impurity_split`` has changed from 1e-7 to 0 in 0.23 and it
 |         will be removed in 1.0 (renaming of 0.25).
 |         Use ``min_impurity_decrease`` instead.
 |
 |  class_weight : dict, list of dict or "balanced", default=None
 |      Weights associated with classes in the form ``{class_label: weight}``.
 |      If None, all classes are supposed to have weight one. For
 |      multi-output problems, a list of dicts can be provided in the same
 |      order as the columns of y.
 |
 |      Note that for multioutput (including multilabel) weights should be
 |      defined for each class of every column in its own dict. For example,
 |      for four-class multilabel classification weights should be
 |      [{0: 1, 1: 1}, {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1}] instead of
 |      [{1:1}, {2:5}, {3:1}, {4:1}].
 |
 |      The "balanced" mode uses the values of y to automatically adjust
 |      weights inversely proportional to class frequencies in the input data
 |      as ``n_samples / (n_classes * np.bincount(y))``
 |
 |      For multi-output, the weights of each column of y will be multiplied.
 |
 |      Note that these weights will be multiplied with sample_weight (passed
 |      through the fit method) if sample_weight is specified.
 |
 |  ccp_alpha : non-negative float, default=0.0
 |      Complexity parameter used for Minimal Cost-Complexity Pruning. The
 |      subtree with the largest cost complexity that is smaller than
 |      ``ccp_alpha`` will be chosen. By default, no pruning is performed. See
```

```
|      :ref:`minimal_cost_complexity_pruning` for details.
|
|      .. versionadded:: 0.22
|
|  Attributes
|  ----------
|  classes_ : ndarray of shape (n_classes,) or list of ndarray
|      The classes labels (single output problem),
|      or a list of arrays of class labels (multi-output problem).
|
|  feature_importances_ : ndarray of shape (n_features,)
|      The impurity-based feature importances.
|      The higher, the more important the feature.
|      The importance of a feature is computed as the (normalized)
|      total reduction of the criterion brought by that feature.  It is also
|      known as the Gini importance [4]_.
|
|      Warning: impurity-based feature importances can be misleading for
|      high cardinality features (many unique values). See
|      :func:`sklearn.inspection.permutation_importance` as an alternative.
|
|  max_features_ : int
|      The inferred value of max_features.
|
|  n_classes_ : int or list of int
|      The number of classes (for single output problems),
|      or a list containing the number of classes for each
|      output (for multi-output problems).
|
|  n_features_ : int
|      The number of features when ``fit`` is performed.
|
|  n_outputs_ : int
|      The number of outputs when ``fit`` is performed.
|
|  tree_ : Tree instance
|      The underlying Tree object. Please refer to
|      ``help(sklearn.tree._tree.Tree)`` for attributes of Tree object and
|      :ref:`sphx_glr_auto_examples_tree_plot_unveil_tree_structure.py`
|      for basic usage of these attributes.
|
|  See Also
|  --------
|  DecisionTreeRegressor : A decision tree regressor.
|
|  Notes
|  -----
|  The default values for the parameters controlling the size of the trees
```

```
|  (e.g. ``max_depth``, ``min_samples_leaf``, etc.) lead to fully grown and
|  unpruned trees which can potentially be very large on some data sets. To
|  reduce memory consumption, the complexity and size of the trees should be
|  controlled by setting those parameter values.
|
|  The :meth:`predict` method operates using the :func:`numpy.argmax`
|  function on the outputs of :meth:`predict_proba`. This means that in
|  case the highest predicted probabilities are tied, the classifier will
|  predict the tied class with the lowest index in :term:`classes_`.
|
|  References
|  ----------
|
|  .. [1] https://en.wikipedia.org/wiki/Decision_tree_learning
|
|  .. [2] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification
|         and Regression Trees", Wadsworth, Belmont, CA, 1984.
|
|  .. [3] T. Hastie, R. Tibshirani and J. Friedman. "Elements of Statistical
|         Learning", Springer, 2009.
|
|  .. [4] L. Breiman, and A. Cutler, "Random Forests",
|         https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
|
|  Examples
|  --------
|  >>> from sklearn.datasets import load_iris
|  >>> from sklearn.model_selection import cross_val_score
|  >>> from sklearn.tree import DecisionTreeClassifier
|  >>> clf = DecisionTreeClassifier(random_state=0)
|  >>> iris = load_iris()
|  >>> cross_val_score(clf, iris.data, iris.target, cv=10)
|  …                             # doctest: +SKIP
|  …
|  array([ 1.     ,  0.93…,  0.86…,  0.93…,  0.93…,
|          0.93…,  0.93…,  1.     ,  0.93…,  1.      ])
|
|  Method resolution order:
|      DecisionTreeClassifier
|      sklearn.base.ClassifierMixin
|      BaseDecisionTree
|      sklearn.base.MultiOutputMixin
|      sklearn.base.BaseEstimator
|      builtins.object
|
|  Methods defined here:
|
|  __init__(self, *, criterion='gini', splitter='best', max_depth=None,
```

```
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features=None, random_state=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None,
ccp_alpha=0.0)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  fit(self, X, y, sample_weight=None, check_input=True,
X_idx_sorted='deprecated')
 |      Build a decision tree classifier from the training set (X, y).
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The training input samples. Internally, it will be converted to
 |          ``dtype=np.float32`` and if a sparse matrix is provided
 |          to a sparse ``csc_matrix``.
 |
 |      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
 |          The target values (class labels) as integers or strings.
 |
 |      sample_weight : array-like of shape (n_samples,), default=None
 |          Sample weights. If None, then samples are equally weighted. Splits
 |          that would create child nodes with net zero or negative weight are
 |          ignored while searching for a split in each node. Splits are also
 |          ignored if they would result in any single class carrying a
 |          negative weight in either child node.
 |
 |      check_input : bool, default=True
 |          Allow to bypass several input checking.
 |          Don't use this parameter unless you know what you do.
 |
 |      X_idx_sorted : deprecated, default="deprecated"
 |          This parameter is deprecated and has no effect.
 |          It will be removed in 1.1 (renaming of 0.26).
 |
 |          .. deprecated :: 0.24
 |
 |      Returns
 |      -------
 |      self : DecisionTreeClassifier
 |          Fitted estimator.
 |
 |  predict_log_proba(self, X)
 |      Predict class log-probabilities of the input samples X.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
```

```
 |          The input samples. Internally, it will be converted to
 |          ``dtype=np.float32`` and if a sparse matrix is provided
 |          to a sparse ``csr_matrix``.
 |
 |      Returns
 |      -------
 |      proba : ndarray of shape (n_samples, n_classes) or list of n_outputs
such arrays if n_outputs > 1
 |          The class log-probabilities of the input samples. The order of the
 |          classes corresponds to that in the attribute :term:`classes_`.
 |
 |  predict_proba(self, X, check_input=True)
 |      Predict class probabilities of the input samples X.
 |
 |      The predicted class probability is the fraction of samples of the same
 |      class in a leaf.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input samples. Internally, it will be converted to
 |          ``dtype=np.float32`` and if a sparse matrix is provided
 |          to a sparse ``csr_matrix``.
 |
 |      check_input : bool, default=True
 |          Allow to bypass several input checking.
 |          Don't use this parameter unless you know what you do.
 |
 |      Returns
 |      -------
 |      proba : ndarray of shape (n_samples, n_classes) or list of n_outputs
such arrays if n_outputs > 1
 |          The class probabilities of the input samples. The order of the
 |          classes corresponds to that in the attribute :term:`classes_`.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.ClassifierMixin:
 |
 |  score(self, X, y, sample_weight=None)
 |      Return the mean accuracy on the given test data and labels.
 |
 |      In multi-label classification, this is the subset accuracy
 |      which is a harsh metric since you require for each sample that
```

```
 |      each label set be correctly predicted.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          Test samples.
 |
 |      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
 |          True labels for `X`.
 |
 |      sample_weight : array-like of shape (n_samples,), default=None
 |          Sample weights.
 |
 |      Returns
 |      -------
 |      score : float
 |          Mean accuracy of ``self.predict(X)`` wrt. `y`.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.base.ClassifierMixin:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from BaseDecisionTree:
 |
 |  apply(self, X, check_input=True)
 |      Return the index of the leaf that each sample is predicted as.
 |
 |      .. versionadded:: 0.17
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input samples. Internally, it will be converted to
 |          ``dtype=np.float32`` and if a sparse matrix is provided
 |          to a sparse ``csr_matrix``.
 |
 |      check_input : bool, default=True
 |          Allow to bypass several input checking.
 |          Don't use this parameter unless you know what you do.
 |
 |      Returns
 |      -------
```

```
 |      X_leaves : array-like of shape (n_samples,)
 |          For each datapoint x in X, return the index of the leaf x
 |          ends up in. Leaves are numbered within
 |          ``[0; self.tree_.node_count)``, possibly with gaps in the
 |          numbering.
 |
 |  cost_complexity_pruning_path(self, X, y, sample_weight=None)
 |      Compute the pruning path during Minimal Cost-Complexity Pruning.
 |
 |      See :ref:`minimal_cost_complexity_pruning` for details on the pruning
 |      process.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The training input samples. Internally, it will be converted to
 |          ``dtype=np.float32`` and if a sparse matrix is provided
 |          to a sparse ``csc_matrix``.
 |
 |      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
 |          The target values (class labels) as integers or strings.
 |
 |      sample_weight : array-like of shape (n_samples,), default=None
 |          Sample weights. If None, then samples are equally weighted. Splits
 |          that would create child nodes with net zero or negative weight are
 |          ignored while searching for a split in each node. Splits are also
 |          ignored if they would result in any single class carrying a
 |          negative weight in either child node.
 |
 |      Returns
 |      -------
 |      ccp_path : :class:`~sklearn.utils.Bunch`
 |          Dictionary-like object, with the following attributes.
 |
 |          ccp_alphas : ndarray
 |              Effective alphas of subtree during pruning.
 |
 |          impurities : ndarray
 |              Sum of the impurities of the subtree leaves for the
 |              corresponding alpha value in ``ccp_alphas``.
 |
 |  decision_path(self, X, check_input=True)
 |      Return the decision path in the tree.
 |
 |      .. versionadded:: 0.18
 |
 |      Parameters
 |      ----------
```

```
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input samples. Internally, it will be converted to
 |          ``dtype=np.float32`` and if a sparse matrix is provided
 |          to a sparse ``csr_matrix``.
 |
 |      check_input : bool, default=True
 |          Allow to bypass several input checking.
 |          Don't use this parameter unless you know what you do.
 |
 |      Returns
 |      -------
 |      indicator : sparse matrix of shape (n_samples, n_nodes)
 |          Return a node indicator CSR matrix where non zero elements
 |          indicates that the samples goes through the nodes.
 |
 |  get_depth(self)
 |      Return the depth of the decision tree.
 |
 |      The depth of a tree is the maximum distance between the root
 |      and any leaf.
 |
 |      Returns
 |      -------
 |      self.tree_.max_depth : int
 |          The maximum depth of the tree.
 |
 |  get_n_leaves(self)
 |      Return the number of leaves of the decision tree.
 |
 |      Returns
 |      -------
 |      self.tree_.n_leaves : int
 |          Number of leaves.
 |
 |  predict(self, X, check_input=True)
 |      Predict class or regression value for X.
 |
 |      For a classification model, the predicted class for each sample in X is
 |      returned. For a regression model, the predicted value based on X is
 |      returned.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input samples. Internally, it will be converted to
 |          ``dtype=np.float32`` and if a sparse matrix is provided
 |          to a sparse ``csr_matrix``.
 |
```

```
 |      check_input : bool, default=True
 |          Allow to bypass several input checking.
 |          Don't use this parameter unless you know what you do.
 |
 |      Returns
 |      -------
 |      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
 |          The predicted classes, or the predict values.
 |
 |  ----------------------------------------------------------------------
 |  Readonly properties inherited from BaseDecisionTree:
 |
 |  feature_importances_
 |      Return the feature importances.
 |
 |      The importance of a feature is computed as the (normalized) total
 |      reduction of the criterion brought by that feature.
 |      It is also known as the Gini importance.
 |
 |      Warning: impurity-based feature importances can be misleading for
 |      high cardinality features (many unique values). See
 |      :func:`sklearn.inspection.permutation_importance` as an alternative.
 |
 |      Returns
 |      -------
 |      feature_importances_ : ndarray of shape (n_features,)
 |          Normalized total reduction of criteria by feature
 |          (Gini importance).
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |
 |  __getstate__(self)
 |
 |  __repr__(self, N_CHAR_MAX=700)
 |      Return repr(self).
 |
 |  __setstate__(self, state)
 |
 |  get_params(self, deep=True)
 |      Get parameters for this estimator.
 |
 |      Parameters
 |      ----------
 |      deep : bool, default=True
 |          If True, will return the parameters for this estimator and
 |          contained subobjects that are estimators.
 |
```

```
 |      Returns
 |      -------
 |      params : dict
 |          Parameter names mapped to their values.
 |
 |  set_params(self, **params)
 |      Set the parameters of this estimator.
 |
 |      The method works on simple estimators as well as on nested objects
 |      (such as :class:`~sklearn.pipeline.Pipeline`). The latter have
 |      parameters of the form ``<component>__<parameter>`` so that it's
 |      possible to update each component of a nested object.
 |
 |      Parameters
 |      ----------
 |      **params : dict
 |          Estimator parameters.
 |
 |      Returns
 |      -------
 |      self : estimator instance
 |          Estimator instance.
```

[24]:
```
def clasVar(df):
    conditions = [
    (df['Depression'] >= 14) ,
    (df['Anxiety'] >= 10)   ,
    (df['Stress'] >= 19),
    ((df['Depression'] < 14)  & (df['Anxiety'] < 10) & (df['Stress'] < 19)),
    ]
    values = ['Severly Depressed','Severely axious', 'Severly Stressed',␣
↪'Normal']
    df['diagnosis Category'] = np.select(conditions, values)
    return df
```

[26]:
```
dfNew = clasVar(df)
dfNew.head(10)
```

[26]:

	Q1A	Q1I	Q1E	Q2A	Q2I	Q2E	Q3A	Q3I	Q3E	Q4A	…	Anxiety	\
0	4	28	3890	4	25	2122	2	16	1944	4	…	34	
1	4	2	8118	1	36	2890	2	35	4777	3	…	17	
2	3	7	5784	1	33	4373	4	41	3242	1	…	12	
3	2	23	5081	3	11	6837	2	37	5521	1	…	17	
4	2	36	3215	2	13	7731	3	5	4156	4	…	40	
5	1	18	6116	1	28	3193	2	2	12542	1	…	6	
6	1	20	4325	1	34	4009	2	38	3604	3	…	19	

```
7     1    34  4796     1     9  2618     1    39    5823     1  …         4
8     4     4  3470     4    14  2139     3     1   11043     4  …        39
9     3    38  5187     2    28  2600     4     9    2015     1  …        28

   Anxiety_cat  Stress  Stress_cat  Extraversion  Agreeableness  \
0            4      40           4           0.5            4.5
1            3      27           3           4.5            4.5
2            2      17           1           1.5            3.5
3            3      16           1           2.0            6.0
4            4      29           3           2.0            3.5
5            0      12           0           1.0            6.5
6            3      14           0           3.5            4.0
7            0       6           0           6.0            3.5
8            4      33           3           0.5            4.5
9            4      34           4           0.5            0.5

   Conscientiousness  EmotionalStability  Openness  diagnosis Category
0                4.5                 0.5       6.5   Severly Depressed
1                2.0                 0.5       3.5   Severly Depressed
2                2.0                 4.0       5.0   Severly Depressed
3                6.5                 4.5       6.0   Severly Depressed
4                2.0                 2.0       4.5   Severly Depressed
5                5.5                 6.0       3.5              Normal
6                5.0                 3.5       2.5   Severly Depressed
7                4.0                 3.5       4.0              Normal
8                2.5                 0.5       4.0   Severly Depressed
9                5.0                 0.5       2.5   Severly Depressed

[10 rows x 184 columns]
```

[27]:
```
dfNew = dfNew.loc[:,['Extraversion', 'Agreeableness', 'Conscientiousness',␣
 ↪'EmotionalStability', 'Openness', 'diagnosis Category']]
dfNew
```

[27]:
```
        Extraversion  Agreeableness  Conscientiousness  EmotionalStability  \
0                0.5            4.5                4.5                 0.5
1                4.5            4.5                2.0                 0.5
2                1.5            3.5                2.0                 4.0
3                2.0            6.0                6.5                 4.5
4                2.0            3.5                2.0                 2.0
…                 …             …                  …                   …
39770            2.0            5.0                3.5                 2.5
39771            2.5            3.0                2.5                 2.0
39772            5.0            3.5                6.5                 3.5
39773            1.5            2.0                3.5                 1.5
39774            5.0            5.0                2.5                 1.5
```

```
       Openness diagnosis Category
0           6.5   Severly Depressed
1           3.5   Severly Depressed
2           5.0   Severly Depressed
3           6.0   Severly Depressed
4           4.5   Severly Depressed
…         …                 …
39770       3.5   Severly Depressed
39771       3.5   Severly Depressed
39772       4.5              Normal
39773       3.0   Severly Depressed
39774       5.5   Severly Depressed

[39775 rows x 6 columns]
```

[28]:
```
dfNew.describe()
```

[28]:
```
       Extraversion  Agreeableness  Conscientiousness  EmotionalStability  \
count  39775.000000   39775.000000       39775.000000        39775.000000
mean       2.967278       4.040377           3.730660            2.738290
std        1.555143       1.229776           1.491769            1.527743
min        0.000000       0.000000           0.000000            0.000000
25%        1.500000       3.500000           2.500000            1.500000
50%        3.000000       4.000000           3.500000            2.500000
75%        4.000000       5.000000           5.000000            3.500000
max        7.000000       7.000000           7.000000            7.000000

           Openness
count  39775.000000
mean       4.101634
std        1.335035
min        0.000000
25%        3.500000
50%        4.000000
75%        5.000000
max        7.000000
```

[30]:
```
from sklearn.model_selection import train_test_split

# Split dataset into training set and test set
# Our class column is Creditability here and everything else will be used as␣
↪features
class_col_name='diagnosis Category'

feature_names=dfNew.columns[dfNew.columns != class_col_name ]
# 70% training and 30% test
```

```
X_train, X_test, y_train, y_test = train_test_split(dfNew.loc[:,␣
 ↪feature_names], dfNew[class_col_name], test_size=0.25,random_state=1)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
(29831, 5) (29831,) (9944, 5) (9944,)
```

[31]:
```
#because the graphviz package was not available in my system I used this method␣
 ↪to show the decision tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_text
clf = tree.DecisionTreeClassifier(random_state=0 , max_depth=5)
decision_tree = clf.fit(X_train, y_train)
r = export_text(decision_tree, ['Extraversion', 'Agreeableness',␣
 ↪'Conscientiousness', 'EmotionalStability', 'Openness'])
print(r)
```

```
|--- EmotionalStability <= 3.25
|   |--- EmotionalStability <= 2.25
|   |   |--- EmotionalStability <= 1.25
|   |   |   |--- Extraversion <= 4.75
|   |   |   |   |--- Conscientiousness <= 3.25
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |   |--- Conscientiousness >  3.25
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |--- Extraversion >  4.75
|   |   |   |   |--- Conscientiousness <= 3.25
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |   |--- Conscientiousness >  3.25
|   |   |   |   |   |--- class: Severly Depressed
|   |   |--- EmotionalStability >  1.25
|   |   |   |--- Conscientiousness <= 4.25
|   |   |   |   |--- Extraversion <= 4.25
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |   |--- Extraversion >  4.25
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |--- Conscientiousness >  4.25
|   |   |   |   |--- Extraversion <= 3.75
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |   |--- Extraversion >  3.75
|   |   |   |   |   |--- class: Severly Depressed
|   |--- EmotionalStability >  2.25
|   |   |--- Extraversion <= 3.75
|   |   |   |--- Conscientiousness <= 3.25
|   |   |   |   |--- Extraversion <= 2.75
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |   |--- Extraversion >  2.75
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |--- Conscientiousness >  3.25
```

```
|   |   |   |   |--- Openness <= 2.75
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |   |--- Openness >  2.75
|   |   |   |   |   |--- class: Severly Depressed
|   |   |--- Extraversion >  3.75
|   |   |   |--- EmotionalStability <= 2.75
|   |   |   |   |--- Conscientiousness <= 2.75
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |   |--- Conscientiousness >  2.75
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |--- EmotionalStability >  2.75
|   |   |   |   |--- Conscientiousness <= 1.75
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |   |--- Conscientiousness >  1.75
|   |   |   |   |   |--- class: Severly Depressed
|--- EmotionalStability >  3.25
|   |--- EmotionalStability <= 5.25
|   |   |--- Extraversion <= 2.75
|   |   |   |--- Conscientiousness <= 2.75
|   |   |   |   |--- Openness <= 2.75
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |   |--- Openness >  2.75
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |--- Conscientiousness >  2.75
|   |   |   |   |--- Extraversion <= 1.25
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |   |--- Extraversion >  1.25
|   |   |   |   |   |--- class: Severly Depressed
|   |   |--- Extraversion >  2.75
|   |   |   |--- Conscientiousness <= 3.75
|   |   |   |   |--- EmotionalStability <= 3.75
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |   |--- EmotionalStability >  3.75
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |--- Conscientiousness >  3.75
|   |   |   |   |--- EmotionalStability <= 3.75
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |   |--- EmotionalStability >  3.75
|   |   |   |   |   |--- class: Normal
|   |--- EmotionalStability >  5.25
|   |   |--- Extraversion <= 2.25
|   |   |   |--- Conscientiousness <= 5.25
|   |   |   |   |--- EmotionalStability <= 5.75
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |   |--- EmotionalStability >  5.75
|   |   |   |   |   |--- class: Normal
|   |   |   |--- Conscientiousness >  5.25
|   |   |   |   |--- Agreeableness <= 2.25
```

```
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |   |--- Agreeableness >  2.25
|   |   |   |   |   |--- class: Normal
|   |   |--- Extraversion >  2.25
|   |   |   |--- Conscientiousness <= 3.75
|   |   |   |   |--- Conscientiousness <= 1.75
|   |   |   |   |   |--- class: Severly Depressed
|   |   |   |   |--- Conscientiousness >  1.75
|   |   |   |   |   |--- class: Normal
|   |   |   |--- Conscientiousness >  3.75
|   |   |   |   |--- EmotionalStability <= 5.75
|   |   |   |   |   |--- class: Normal
|   |   |   |   |--- EmotionalStability >  5.75
|   |   |   |   |   |--- class: Normal
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:70:
FutureWarning: Pass feature_names=['Extraversion', 'Agreeableness',
'Conscientiousness', 'EmotionalStability', 'Openness'] as keyword args. From
version 1.0 (renaming of 0.25) passing these as positional arguments will result
in an error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "
```

[32]:
```
y_pred = clf.predict(X_test)
```

[33]:
```
from sklearn.metrics import confusion_matrix
cf=confusion_matrix(y_test, y_pred)
print ("Confusion Matrix")
print(cf)
```

```
Confusion Matrix
[[ 818    0 1160    0]
 [ 171    0  921    0]
 [ 320    0 6431    0]
 [  14    0  109    0]]
```

[34]:
```
from sklearn.metrics import classification_report
from sklearn import metrics

print(classification_report(y_test, y_pred))
```

```
                   precision    recall  f1-score   support

           Normal       0.62      0.41      0.50      1978
  Severely axious       0.00      0.00      0.00      1092
Severly Depressed       0.75      0.95      0.84      6751
 Severly Stressed       0.00      0.00      0.00       123

         accuracy                           0.73      9944
```

```
   macro avg       0.34      0.34      0.33      9944
weighted avg       0.63      0.73      0.67      9944
```

```
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[ ]:

```
```