mice_protein_expression

June 21, 2022

1 Mice Protein Expression (Introduction and Description)

Here, we examine 77 protein expression levels measured in the cerebral cortex of mice. The mice are split into many separate categories. First, the mice either possess a trisomy (resulting in down syndrome) or non-trisomic. Second, the mice are either stimulated to learn or note stimulated to learn. Third, the mice are either injected with saline or are injected with Memantine, which is used to treat Alzheimer's. Together, this results in 8 separate classes: 1. c-CS-s: control mice, stimulated to learn, injected with saline (9 mice) 2. c-CS-m: control mice, stimulated to learn, injected with memantine (10 mice) 3. c-SC-s: control mice, not stimulated to learn, injected with saline (9 mice) 4. c-SC-m: control mice, not stimulated to learn, injected with memantine (10 mice) 5. t-CS-s: trisomy mice, stimulated to learn, injected with saline (7 mice) 6. t-CS-m: trisomy mice, stimulated to learn, injected with memantine (9 mice) 7. t-SC-s: trisomy mice, not stimulated to learn, injected with saline (9 mice) 8. t-SC-m: trisomy mice, not stimulated to learn, injected with memantine (9 mice)

Thus, the question at hand is *can we cluster mice together using K-Means Clustering to effectively separate the 8 classes?*. To do this, we will use K-Means and permute the cluster indices to find the most accurate labels that correspond. This provides us a semi-parametric mean to classify mice protein expression levels. This will likely be a difficult task since there are a large number of predictors and relatively obscure classes. The notebook is organized as follows: 1. **Preamble**, where we load modules and create classes for loading, summarizing, and visualizing data. 2. **Preprocessing**, where we prepare data for modeling by removing missing values and describing the data. 3. **Feature Reduction**, where we perform dimensionality reduction on the dataset via PCA. 4. **Model Building**, where we build the K-Means model and prepare it for evaluation 5. **Model Evaluation**, where we predict the cluster and class of the test data on the K-Means model 6. **Conclusion**, where we discuss where the model performs well, fails, and delve further into its performance.

1.1 Data Source

The dataset is availabel online at Kaggle and can be found here

1.2 Github

If you are interested, this worked is posted on my GitHub. A link to that can be found here.

1.3 Preamble (Loading Modules and Creating a Data Class)

Here, we will load a number of necessary modules. `sklearn` will be used heavily for data reduction, model building, and model evaluation. `pandas` is used to read in and handle data as a `DataFrame()`

object. `seaborn` and `matplotlib` provide impressive plotting capabilities, and `klib` provides some ease of access tools and visualizations for data-processing.

Our Data() class is used to store, split, and perform data reduction, while the DataTesting() class is used primarily to produce classification metrics and confusion matrix plots.

[90]:
```python
import numpy as np

# Data description, cleaning, and visualization
import klib
import pandas as pd

import seaborn as sns
from matplotlib import pyplot as plt
%matplotlib inline

# Feature Reduction and preprocessing
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder

# Model Creation (Hierarchical)
from sklearn.cluster import KMeans

# Model evaluation
from sklearn.metrics import silhouette_score
from sklearn.metrics import classification_report
from yellowbrick.cluster import SilhouetteVisualizer

# Data splitting
from sklearn.model_selection import train_test_split

from itertools import permutations
```

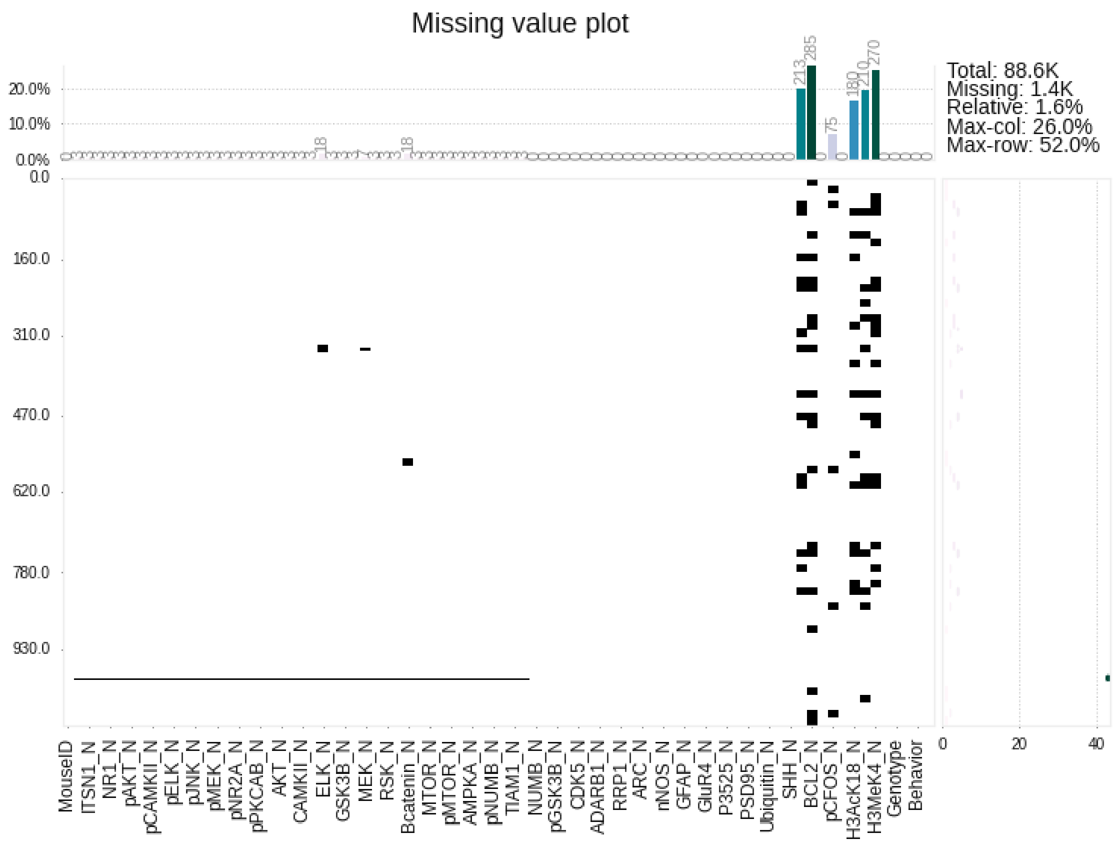
[91]:
```python
class Data():
    def __init__(self, data, target, train_proportion=0.75):
        """Class for loading, storing, splitting, and describing data.

        Parameters
        ----------
        data: Pandas Dataframe
            The full dataset prior to splitting
        target: String
            The column name of the target variable
        train_proportion: float, default=0.75
            The proportion of data to dedicate to a training set. The remainder␣
↪will be dedicated to a testing set
        """
        data_x = data.drop(labels=target, axis=1)
        data_y = data[target]

        self.full = data
        self.train_proportion = train_proportion

        self.features = data_x.columns
        self.target = target

        # Standardize the features
        scaler = StandardScaler()
        data_x = scaler.fit_transform(data_x)

        self.x_train, self.x_test, self.y_train, self.y_test = train_test_split(
            data_x, data_y, train_size=train_proportion, random_state=11235813
        )

    def __str__(self):
        nrows, ncols = self.full.shape
        return f"""\
Data({nrows} Observations x {ncols} Variables)
   X Train:    {self.x_train.shape}
   Y Train:    {self.y_train.shape}
   X Test:     {self.x_test.shape}
   Y Test:     {self.y_test.shape}
   Train Size: {self.train_proportion}\
"""

    def pca(self, explained_variance_threshold=0.99):
        """Perform PCA on the training data to transform it into the principal
        components. This transformation is applied to the training data and only
        the most significant components are kept.

        This operation is done inplace, so the training and testing features
        will automatically be pruned.

        Parameters
        ----------
        explained_variance_threshold: float, default=0.98
            The cutoff value for [percentage of] explained variance. All
            remaining PCs will be discarded.
        """
        # Perform PCA and transform our training data
        pca = PCA(n_components=explained_variance_threshold, svd_solver="full")
        x_train = pca.fit_transform(self.x_train)

        # Determine the feature names of the reduced data
        n_components = pca.components_.shape[0]

        # Get the index of the most important feature for each component
        feature_index = [np.abs(pca.components_[i]).argmax() for i in␣
 ↪range(n_components)]

        # Extract the new feature names
        self.features = [self.features[feature_index[i]] for i in␣
 ↪range(n_components)]

        # Remove the unselected features from the data
        self.x_train = self.x_train[:, feature_index]
        self.x_test = self.x_test[:, feature_index]




class DataTesting():
    def __init__(self, labels_pred, labels_truth, decoder: dict=None):
        """
        Class for testing our a fitted model given truth and predicted labels.

        Parameters
        ----------
        labels_pred: np.ndarray
            A numpy array containing the predicted labels from the model
        labels_truth: np.ndarray
            A numpy array containing the true labels from the model
        decoder: dict, default=None
            An optional dictionary used to decode the label values. Has the
            format decoder[encoded_label] -> label_name
        """
        assert isinstance(labels_pred, np.ndarray) and isinstance(labels_truth,␣
 ↪np.ndarray), "Labels must be a NumPy array."
        assert labels_pred.shape == labels_truth.shape, "Labels must be of␣
 ↪equal shape."
        self.pred = labels_pred
        self.truth = labels_truth
        self.decoder = {key: val for key, val in decoder.items()}

    def report(self):
        """
        Returns a report on evaluation metrics for the model such as
        specicifity and sensitivity.

        Returns
        -------
        report_df: pd.DataFrame
            The resulting model report in the form of a pandas dataframe
        """
        report = classification_report(
            self.truth,
            self.pred,
            output_dict=True
        )

        # Make a dataframe and rename the columnns
        report_df = pd.DataFrame.from_dict(report)
        report_df.rename(
            columns={str(key): val for key, val in self.decoder.items()},
            inplace=True
        )

        return report_df

    def confusion_matrix(self):
        """
        Computes the confusion matrix for the model.

        Returns
        -------
        confusion_matrix: pd.DataFrame
            The confusion matrix
        """
        n_features = len(np.unique(self.truth))
        matrix = np.zeros((n_features, n_features))
        for pred, truth in zip(self.pred, self.truth):
            matrix[pred, truth] += 1

        confusion_matrix = pd.DataFrame(matrix)
        confusion_matrix.rename(columns=self.decoder, inplace=True)
        confusion_matrix.rename(index=self.decoder, inplace=True)

        return confusion_matrix

    def confusion_matrix_plot(self, confusion_matrix=None):
        """
        Returns a figure corresponding to the model's confusion matrix.

        Arguments
        ---------
        confusion_matrix: None or pd.DataFrame, default=None
            The confusion matrix to plot against. If None, it will generate
            one from the class attributes
        """
        if confusion_matrix is None:
            confusion_matrix = self.confusion_matrix()

        fig, ax = plt.subplots(figsize=(11, 10), tight_layout=True)
        sns.heatmap(
            data=confusion_matrix,
            robust=True,
            cmap=sns.dark_palette("#69d", reverse=False, as_cmap=True),
            annot=True,
            fmt=".100g",
            linewidth=0.5,
            ax=ax,
            square=True
        )

        ax.set_xlabel("True Label")
        ax.set_ylabel("Predicted Label")
        ax.set_xticklabels(ax.get_xticklabels(), rotation=45)

        return fig, ax
```

1.4 Preprocessing (Loading Data and Handling Missing Values)

Here, we load the data in from its relative path and investigate for missing values. We will also remove some unecessary or redundant dimensions.

[92]:
```
# Specify the location of our data
fname = r"../Data/data_cortex_nuclear.csv"

# Load the data as a DataFrame
df = pd.read_csv(fname)

# Examine the data for missing values
klib.missingval_plot(df, figsize=(12, 8))
```

[92]:
```
GridSpec(6, 6)
```



```
[93]: # Find all the NA's in the dataframe
      df_nas = df.isna().sum().sort_values(ascending=False)
      df_nas = df_nas[df_nas > 0]
      print(df_nas.to_string())

      # Remove the genotype, treatment, and behavior columns (All in Class)
      df.drop(labels=["Genotype", "Treatment", "Behavior"], axis=1, inplace=True)

      # We can also drop MouseID because they are all unique
      if len(df["MouseID"]) == len(df):
          df.drop(labels="MouseID", axis=1, inplace=True)

      # We will first drop data from columns with more than 50 NAs
      df.dropna(axis=1, thresh=len(df) - 50, inplace=True)

      # Drop the remaining rows with NAs present
      df.dropna(axis=0, how="any", inplace=True)

      print(f"Remaining NA Values: {np.sum(df.isna().to_numpy())}")
```

```
BCL2_N          285
```

```
H3MeK4_N      270
BAD_N         213
EGR1_N        210
H3AcK18_N     180
pCFOS_N        75
ELK_N          18
Bcatenin_N     18
MEK_N           7
P38_N           3
JNK_N           3
TRKA_N          3
RSK_N           3
SOD1_N          3
MTOR_N          3
RAPTOR_N        3
pMTOR_N         3
DSCR1_N         3
AMPKA_N         3
GSK3B_N         3
pNUMB_N         3
DYRK1A_N        3
TIAM1_N         3
pP70S6_N        3
NR2B_N          3
APP_N           3
ERK_N           3
PKCA_N          3
NR1_N           3
NR2A_N          3
pAKT_N          3
pBRAF_N         3
CREB_N          3
pCAMKII_N       3
pCREB_N         3
pELK_N          3
BDNF_N          3
pJNK_N          3
pERK_N          3
pMEK_N          3
pNR1_N          3
pNR2A_N         3
pNR2B_N         3
pPKCAB_N        3
pRSK_N          3
AKT_N           3
BRAF_N          3
CAMKII_N        3
ITSN1_N         3
```

```
Remaining NA Values: 0
```

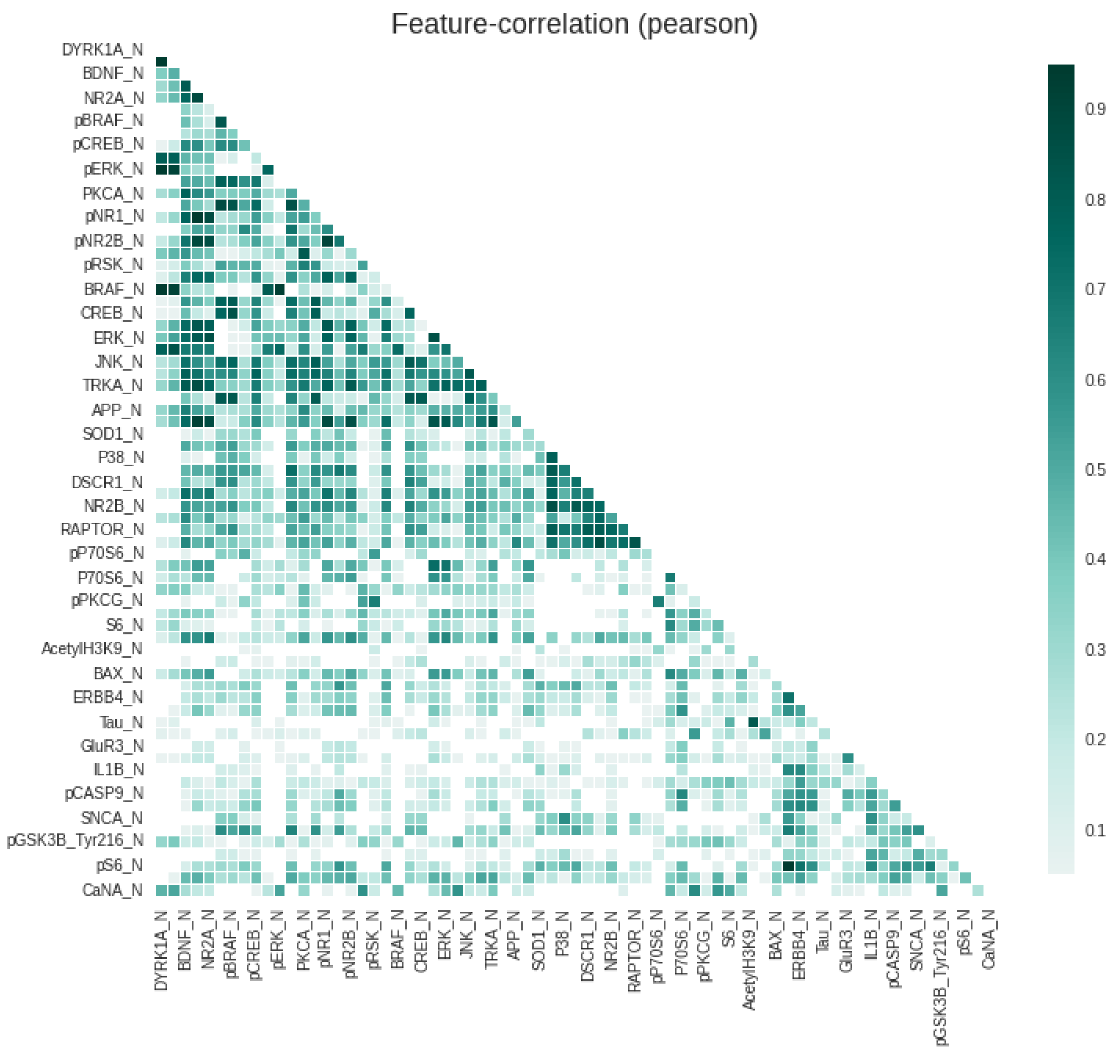
1.5 Preprocessing (Describing and Visualizing Data)

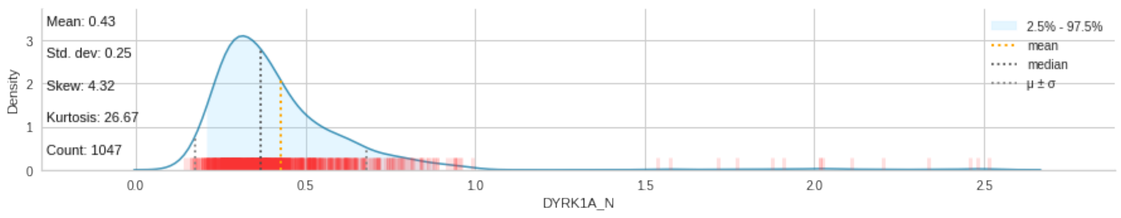
We should also look at a correlation matrix to see if some features co-vary. These will likely be removed in PCA as there provide little additional information in the data. We can also examine density estimates to make sure there are no data inconsistencies.

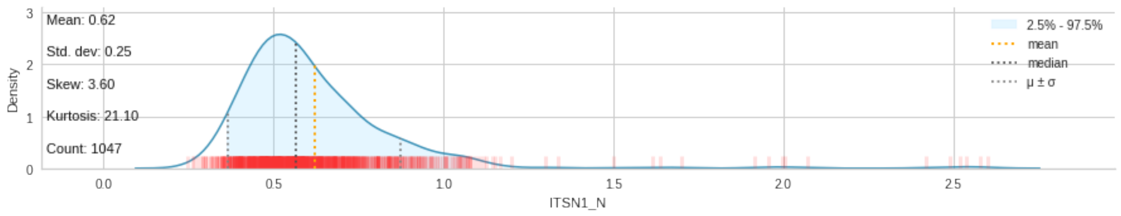
```
[104]: klib.corr_plot(df, split="pos", method="pearson", annot=False)
       klib.dist_plot(df, showall=True)
```

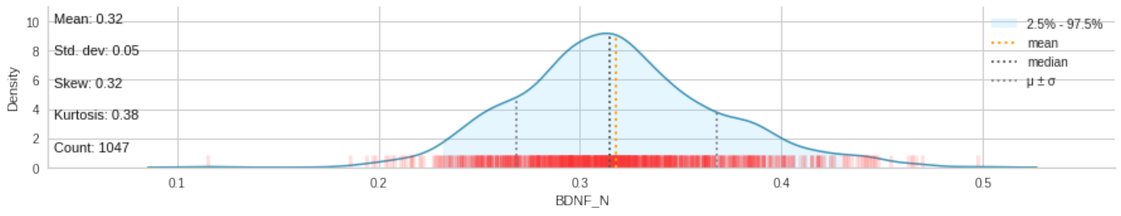
```
Displaying positive correlations. Specify a positive "threshold" to limit the
results further.

/home/benjamin/.local/lib/python3.10/site-packages/seaborn/axisgrid.py:409:
RuntimeWarning: More than 20 figures have been opened. Figures created through
the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly
closed and may consume too much memory. (To control this warning, see the
rcParam `figure.max_open_warning`).
  fig = plt.figure(figsize=figsize)
```

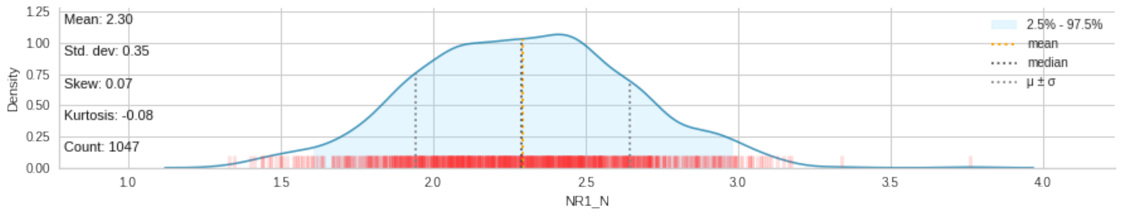
```
[104]: <AxesSubplot:xlabel='CaNA_N', ylabel='Density'>
```

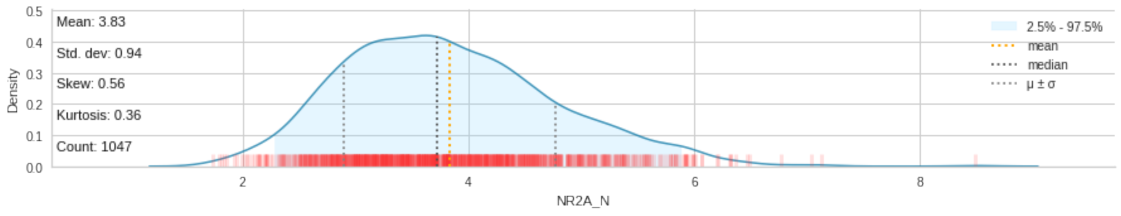

Feature-correlation (pearson)
DYRK1A_N
BDNF_N
NR2A_N
pBRAF_N
pCREB_N
pERK_N
PKCA_N
pNR1_N
pNR2B_N
pRSK_N
BRAF_N
CREB_N
ERK_N
JNK_N
TRKA_N
APP_N
SOD1_N
P38_N
DSCR1_N
NR2B_N
RAPTOR_N
pP70S6_N
P70S6_N
pPKCG_N
S6_N
AcetylH3K9_N
BAX_N
ERBB4_N
Tau_N
GluR3_N
IL1B_N
pCASP9_N
SNCA_N
pGSK3B_Tyr216_N
pS6_N
CaNA_N
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

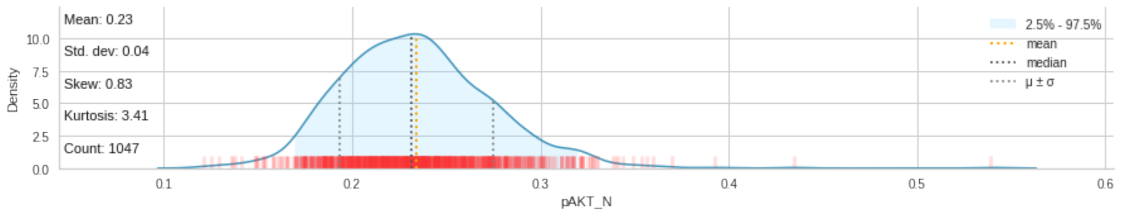

Mean: 0.43
Std. dev: 0.25
Skew: 4.32
Kurtosis: 26.67
Count: 1047
Density
DYRK1A_N
2.5% - 97.5%
mean
median
μ ± σ

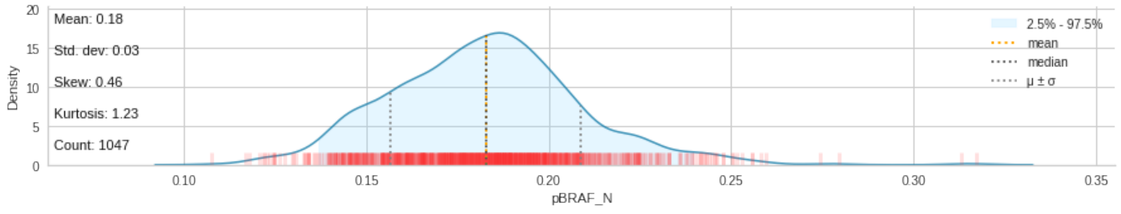

Mean: 0.62
Std. dev: 0.25
Skew: 3.60
Kurtosis: 21.10
Count: 1047
Density
ITSN1_N
2.5% - 97.5%
mean
median
μ ± σ

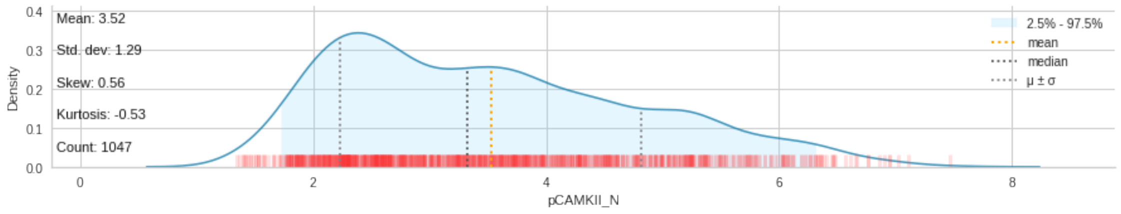

Mean: 0.32
Std. dev: 0.05
Skew: 0.32
Kurtosis: 0.38
Count: 1047
Density
BDNF_N
2.5% - 97.5%
mean
median
μ ± σ

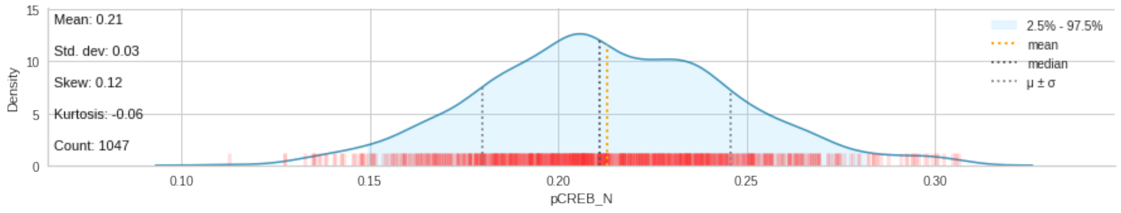

Mean: 2.30
Std. dev: 0.35
Skew: 0.07
Kurtosis: -0.08
Count: 1047
Density
NR1_N
2.5% - 97.5%
mean
median
μ ± σ

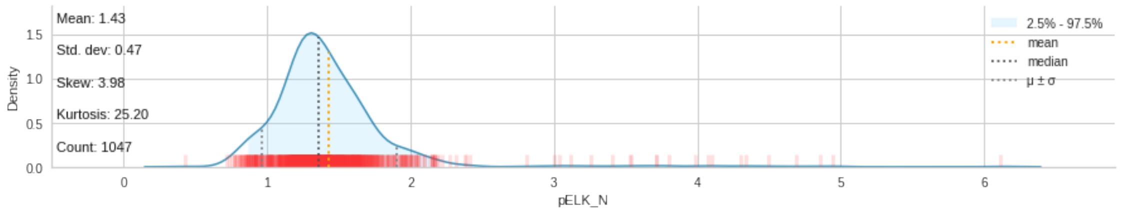

Mean: 3.83
Std. dev: 0.94
Skew: 0.56
Kurtosis: 0.36
Count: 1047
Density
NR2A_N
2.5% - 97.5%
mean
median
μ ± σ

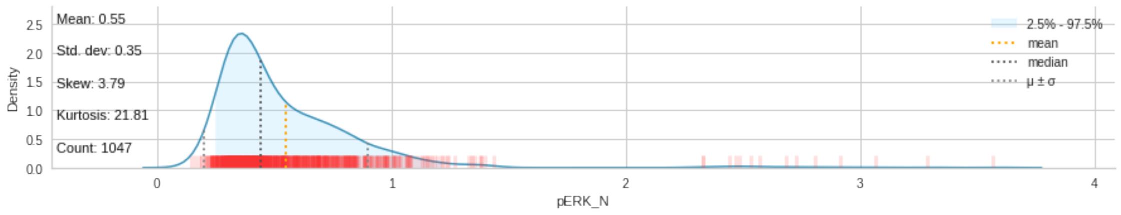

Mean: 0.23
Std. dev: 0.04
Skew: 0.83
Kurtosis: 3.41
Count: 1047
Density
pAKT_N
2.5% - 97.5%
mean
median
μ ± σ

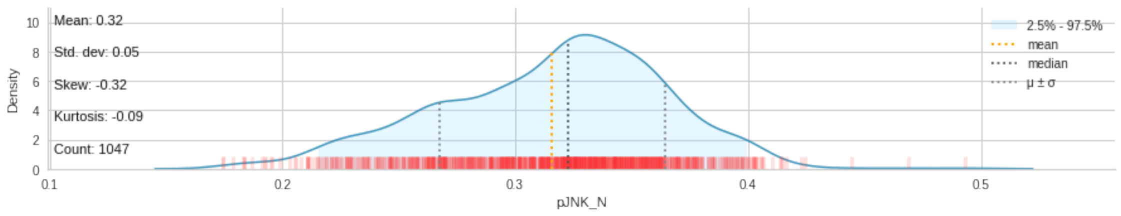

Mean: 0.18
Std. dev: 0.03
Skew: 0.46
Kurtosis: 1.23
Count: 1047
Density
pBRAF_N
2.5% - 97.5%
mean
median
μ ± σ

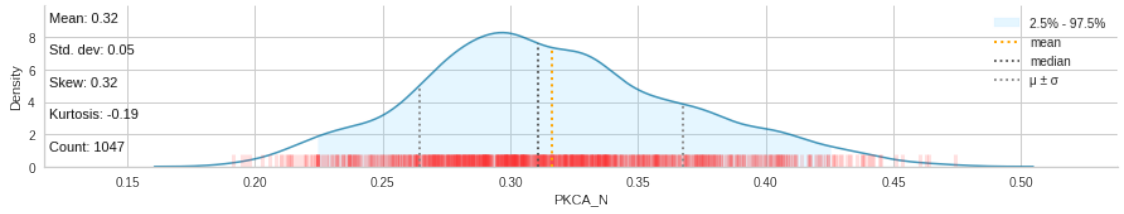

Mean: 3.52
Std. dev: 1.29
Skew: 0.56
Kurtosis: -0.53
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
pCAMKII_N

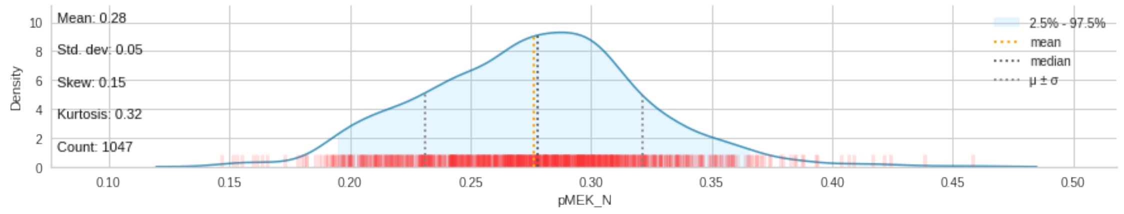

Mean: 0.21
Std. dev: 0.03
Skew: 0.12
Kurtosis: -0.06
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
pCREB_N

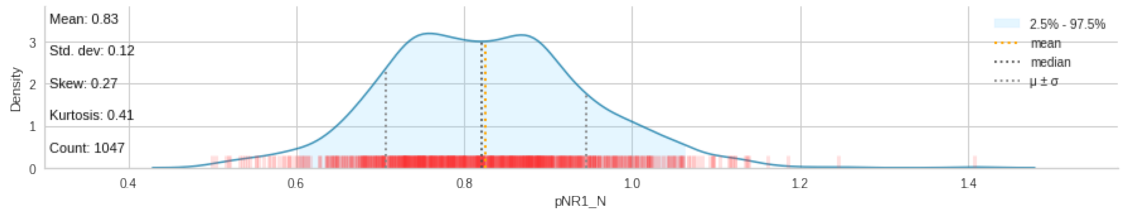

Mean: 1.43
Std. dev: 0.47
Skew: 3.98
Kurtosis: 25.20
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
pELK_N

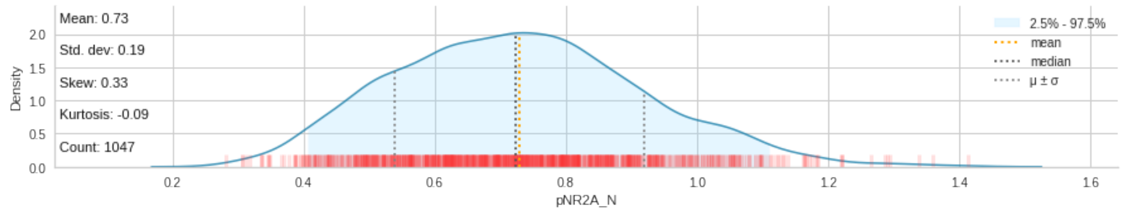

Mean: 0.55
Std. dev: 0.35
Skew: 3.79
Kurtosis: 21.81
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
pERK_N

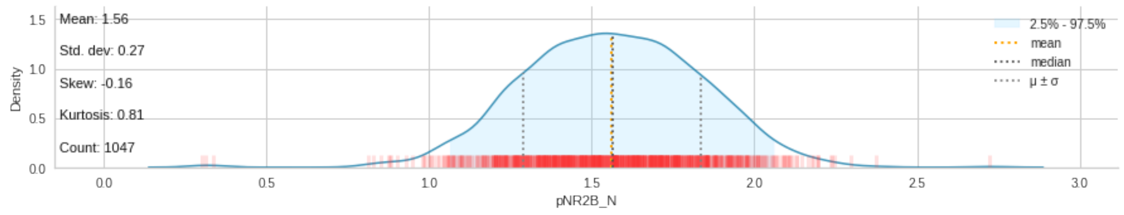

Mean: 0.32
Std. dev: 0.05
Skew: -0.32
Kurtosis: -0.09
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
pJNK_N

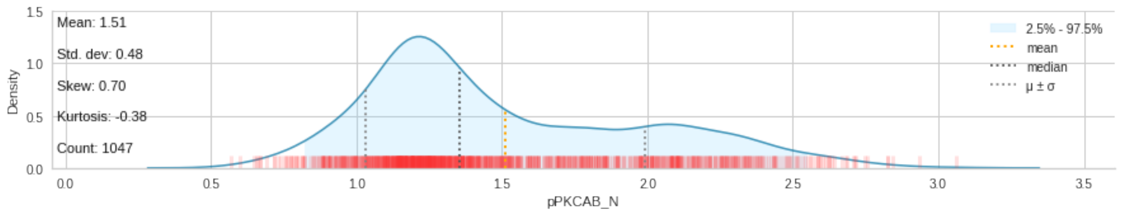

Mean: 0.32
Std. dev: 0.05
Skew: 0.32
Kurtosis: -0.19
Count: 1047
Density
PKCA_N
2.5% - 97.5%
mean
median
μ ± σ

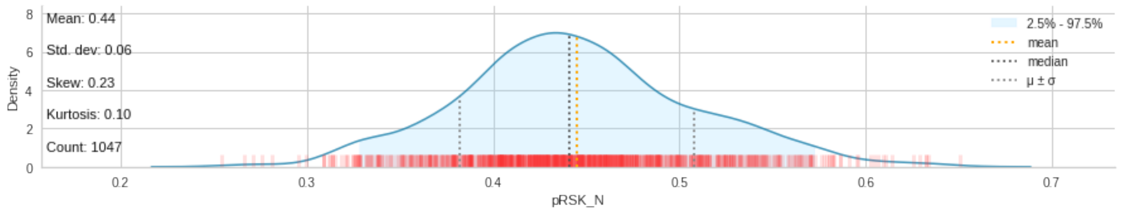

Mean: 0.28
Std. dev: 0.05
Skew: 0.15
Kurtosis: 0.32
Count: 1047
Density
pMEK_N
2.5% - 97.5%
mean
median
μ ± σ

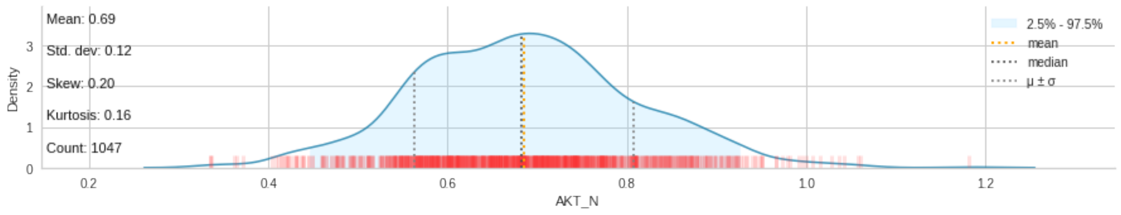

Mean: 0.83
Std. dev: 0.12
Skew: 0.27
Kurtosis: 0.41
Count: 1047
Density
pNR1_N
2.5% - 97.5%
mean
median
μ ± σ

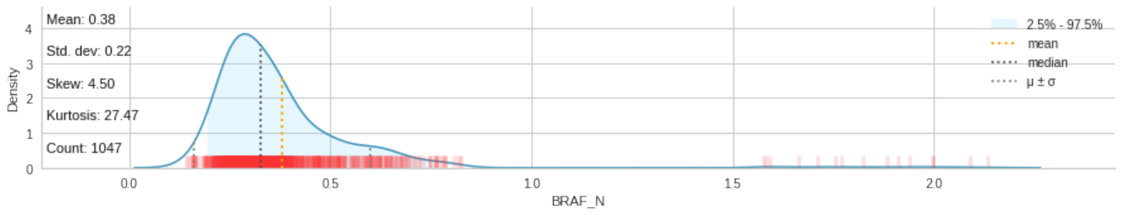

Mean: 0.73
Std. dev: 0.19
Skew: 0.33
Kurtosis: -0.09
Count: 1047
Density
pNR2A_N
2.5% - 97.5%
mean
median
μ ± σ

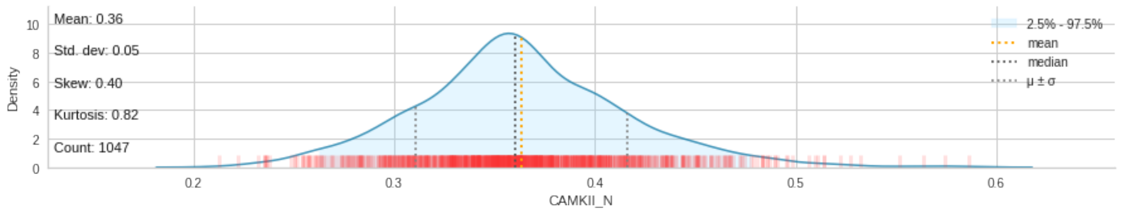

Mean: 1.56
Std. dev: 0.27
Skew: -0.16
Kurtosis: 0.81
Count: 1047
Density
pNR2B_N
2.5% - 97.5%
mean
median
μ ± σ

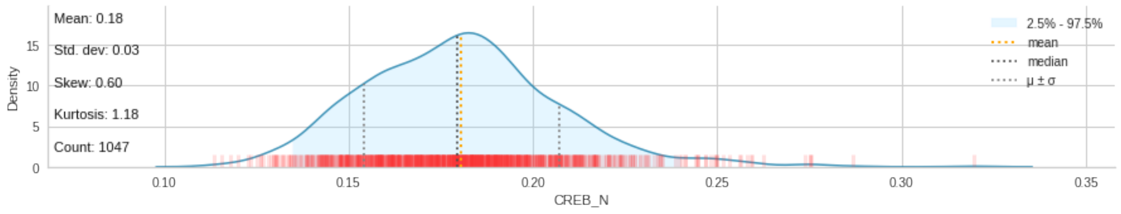

Mean: 1.51
Std. dev: 0.48
Skew: 0.70
Kurtosis: -0.38
Count: 1047
Density
pPKCAB_N
2.5% - 97.5%
mean
median
μ ± σ

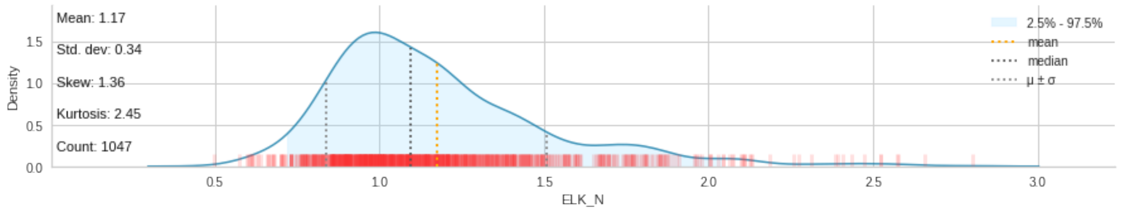

Mean: 0.44
Std. dev: 0.06
Skew: 0.23
Kurtosis: 0.10
Count: 1047
Density
pRSK_N
2.5% - 97.5%
mean
median
μ ± σ

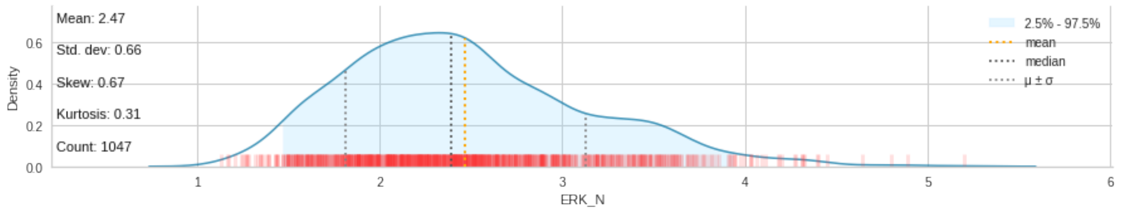

Mean: 0.69
Std. dev: 0.12
Skew: 0.20
Kurtosis: 0.16
Count: 1047
Density
AKT_N
2.5% - 97.5%
mean
median
μ ± σ

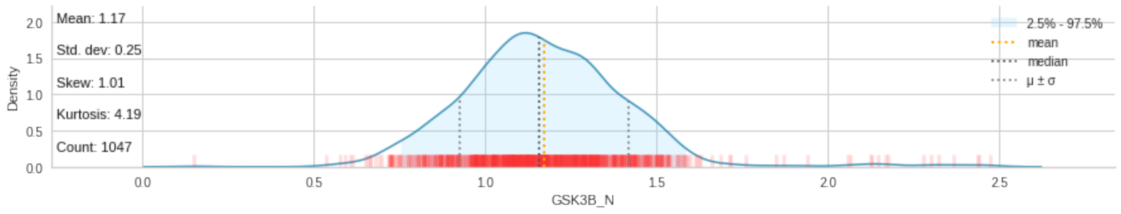

Mean: 0.38
Std. dev: 0.22
Skew: 4.50
Kurtosis: 27.47
Count: 1047
Density
BRAF_N
2.5% - 97.5%
mean
median
μ ± σ

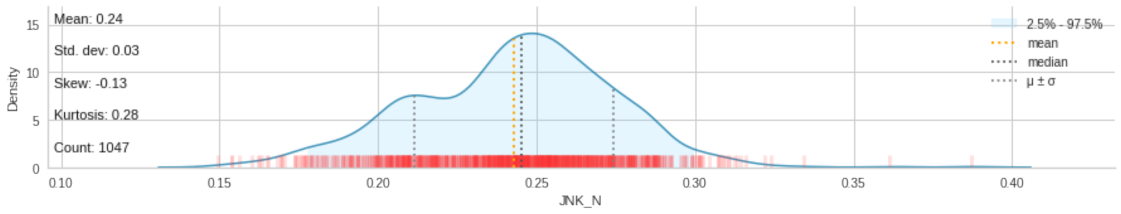

Mean: 0.36
Std. dev: 0.05
Skew: 0.40
Kurtosis: 0.82
Count: 1047
Density
CAMKII_N
2.5% - 97.5%
mean
median
μ ± σ

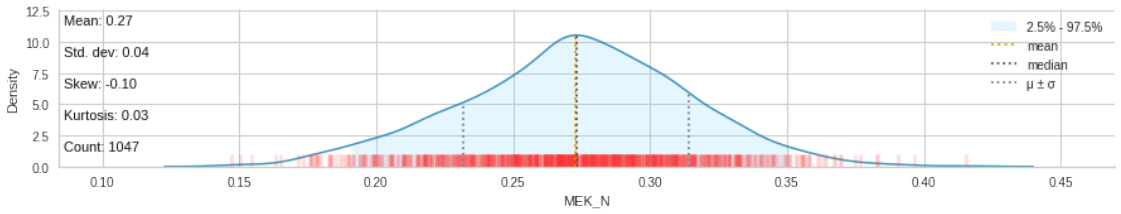

Mean: 0.18
Std. dev: 0.03
Skew: 0.60
Kurtosis: 1.18
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
CREB_N

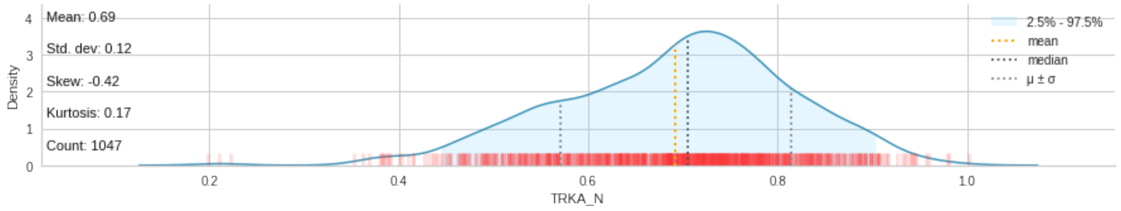

Mean: 1.17
Std. dev: 0.34
Skew: 1.36
Kurtosis: 2.45
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
ELK_N

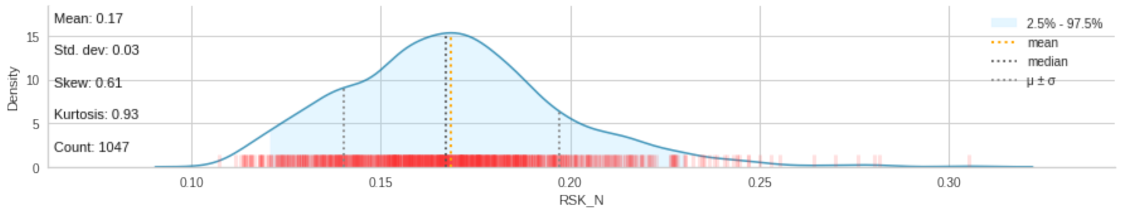

Mean: 2.47
Std. dev: 0.66
Skew: 0.67
Kurtosis: 0.31
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
ERK_N

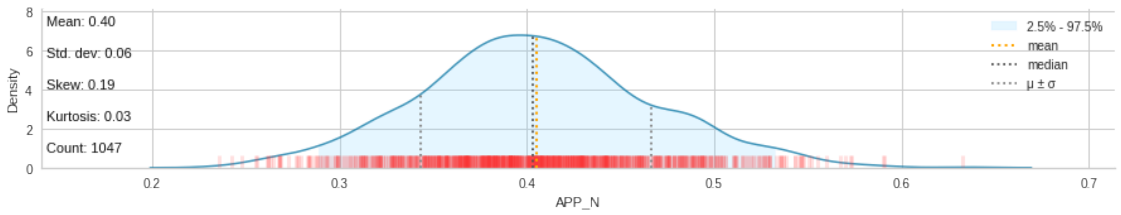

Mean: 1.17
Std. dev: 0.25
Skew: 1.01
Kurtosis: 4.19
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
GSK3B_N

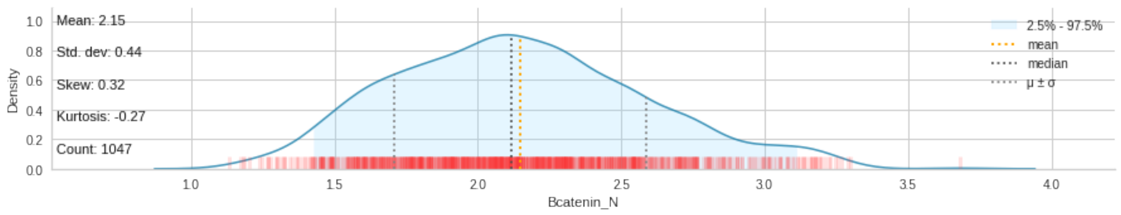

Mean: 0.24
Std. dev: 0.03
Skew: -0.13
Kurtosis: 0.28
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
JNK_N

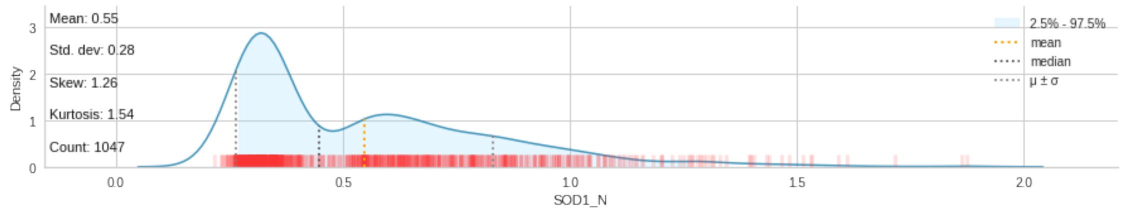

Mean: 0.27
Std. dev: 0.04
Skew: -0.10
Kurtosis: 0.03
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
MEK_N


Mean: 0.69
Std. dev: 0.12
Skew: -0.42
Kurtosis: 0.17
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
TRKA_N

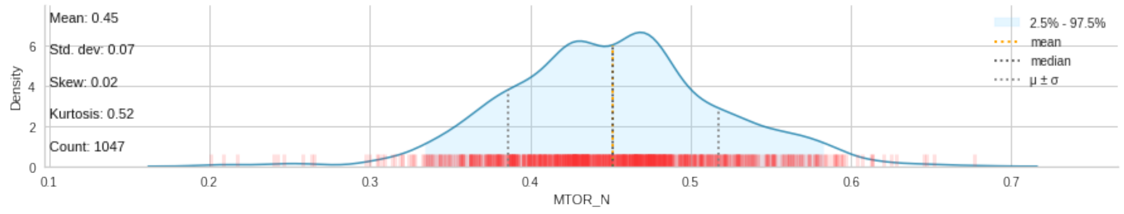

Mean: 0.17
Std. dev: 0.03
Skew: 0.61
Kurtosis: 0.93
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
RSK_N


Mean: 0.40
Std. dev: 0.06
Skew: 0.19
Kurtosis: 0.03
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
APP_N

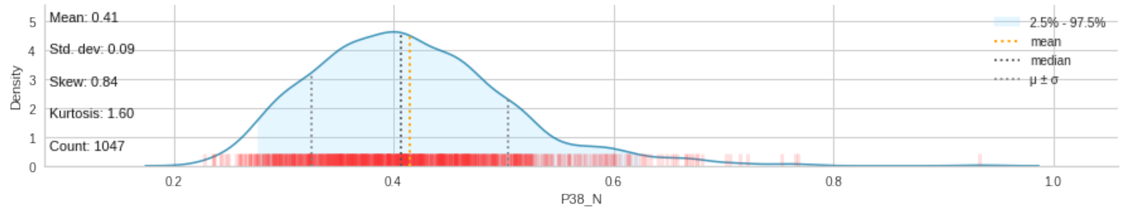

Mean: 2.15
Std. dev: 0.44
Skew: 0.32
Kurtosis: -0.27
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
Bcatenin_N



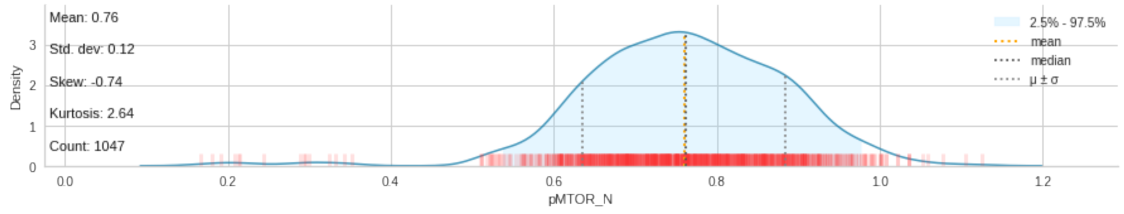
Mean: 0.55
Std. dev: 0.28
Skew: 1.26
Kurtosis: 1.54
Count: 1047
Density
SOD1_N
2.5% - 97.5%
mean
median
μ ± σ



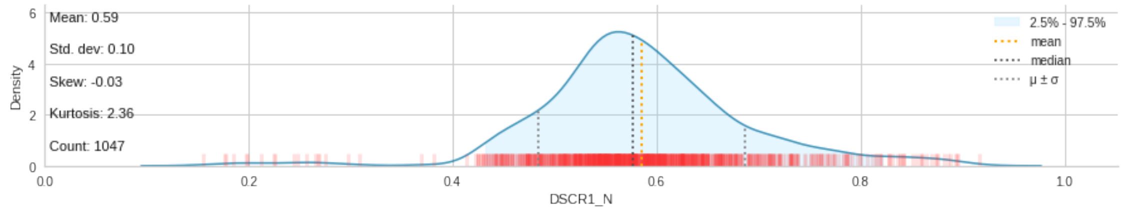
Mean: 0.45
Std. dev: 0.07
Skew: 0.02
Kurtosis: 0.52
Count: 1047
Density
MTOR_N
2.5% - 97.5%
mean
median
μ ± σ



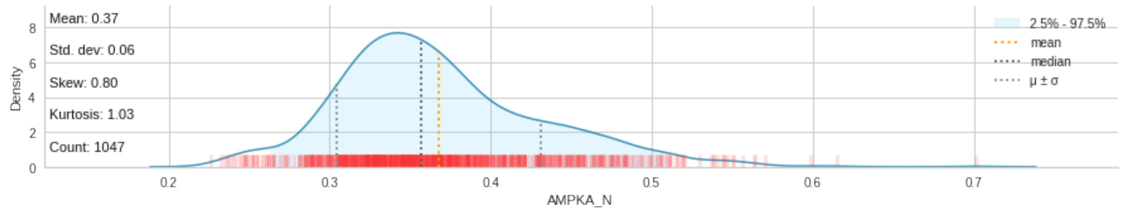
Mean: 0.41
Std. dev: 0.09
Skew: 0.84
Kurtosis: 1.60
Count: 1047
Density
P38_N
2.5% - 97.5%
mean
median
μ ± σ



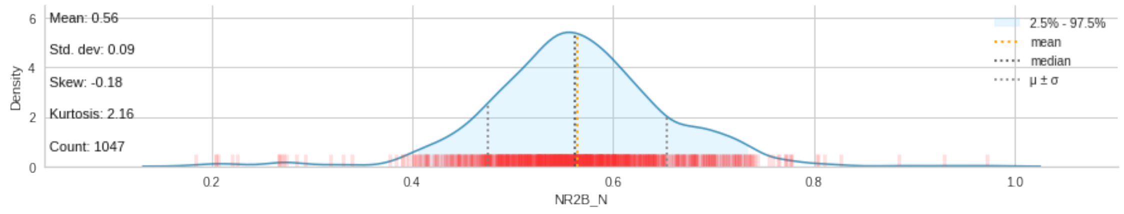
Mean: 0.76
Std. dev: 0.12
Skew: -0.74
Kurtosis: 2.64
Count: 1047
Density
pMTOR_N
2.5% - 97.5%
mean
median
μ ± σ



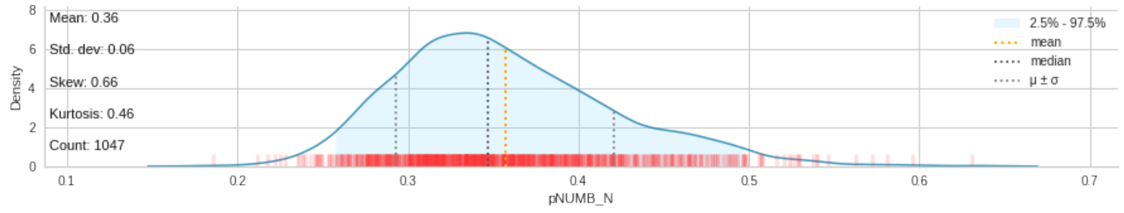
Mean: 0.59
Std. dev: 0.10
Skew: -0.03
Kurtosis: 2.36
Count: 1047
Density
DSCR1_N
2.5% - 97.5%
mean
median
μ ± σ



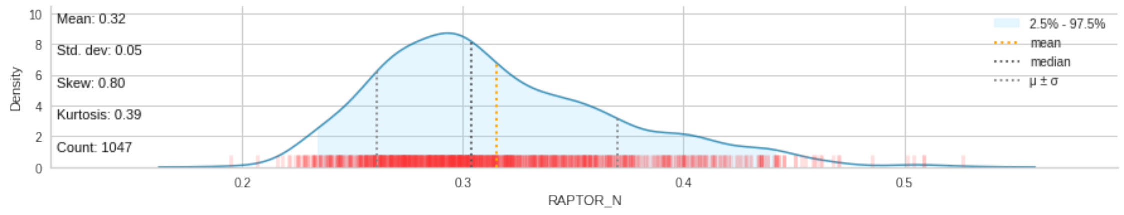
Mean: 0.37
Std. dev: 0.06
Skew: 0.80
Kurtosis: 1.03
Count: 1047
Density
AMPKA_N
2.5% - 97.5%
mean
median
μ ± σ



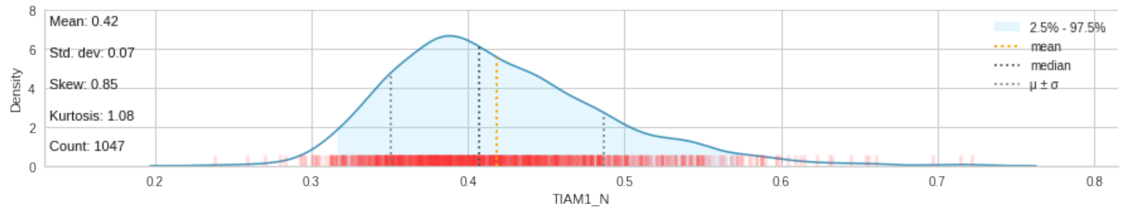
Mean: 0.56
Std. dev: 0.09
Skew: -0.18
Kurtosis: 2.16
Count: 1047
Density
NR2B_N
2.5% - 97.5%
mean
median
μ ± σ



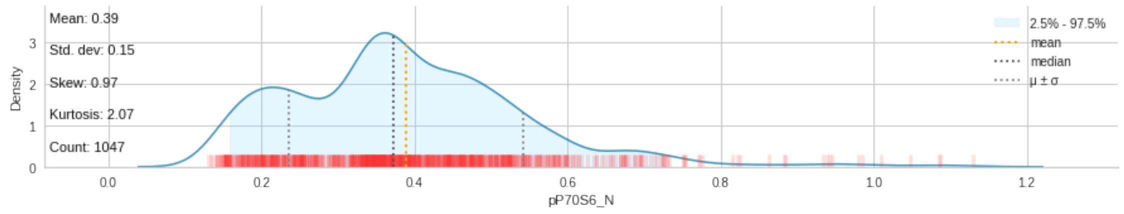
Mean: 0.36
Std. dev: 0.06
Skew: 0.66
Kurtosis: 0.46
Count: 1047
Density
pNUMB_N
2.5% - 97.5%
mean
median
μ ± σ

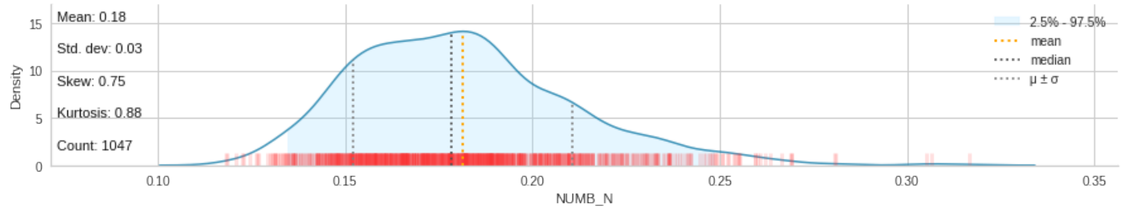


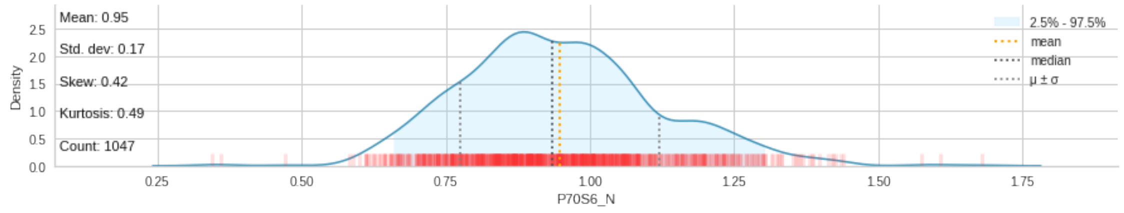
Mean: 0.32
Std. dev: 0.05
Skew: 0.80
Kurtosis: 0.39
Count: 1047
Density
RAPTOR_N
2.5% - 97.5%
mean
median
μ ± σ

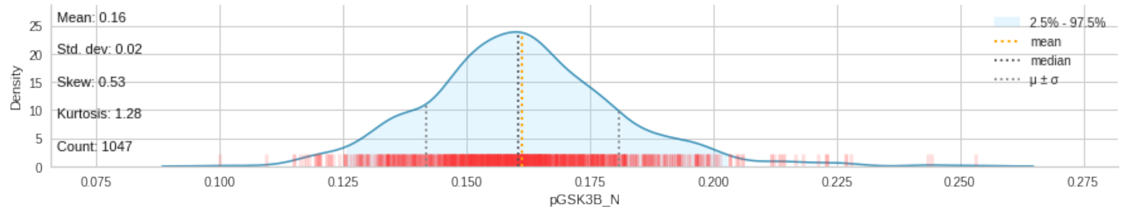


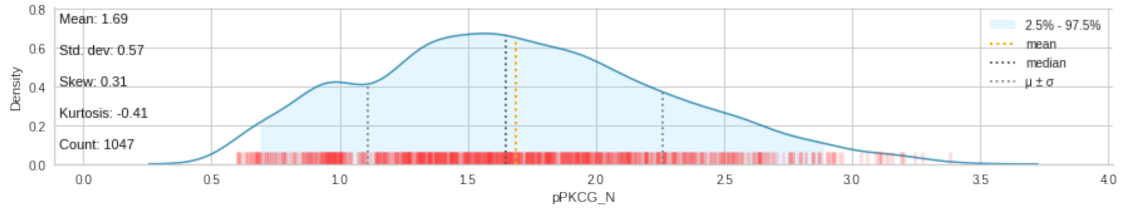
Mean: 0.42
Std. dev: 0.07
Skew: 0.85
Kurtosis: 1.08
Count: 1047
Density
TIAM1_N
2.5% - 97.5%
mean
median
μ ± σ

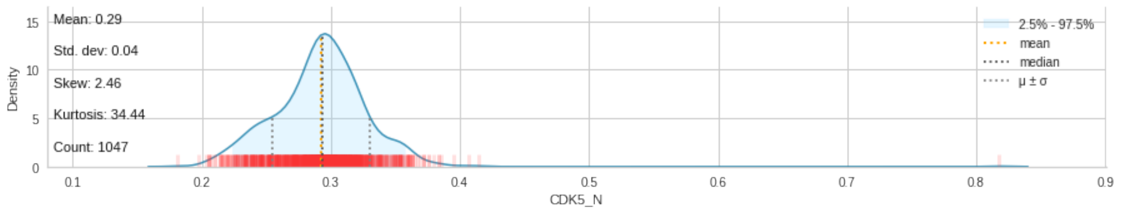

Mean: 0.39
Std. dev: 0.15
Skew: 0.97
Kurtosis: 2.07
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
pP70S6_N

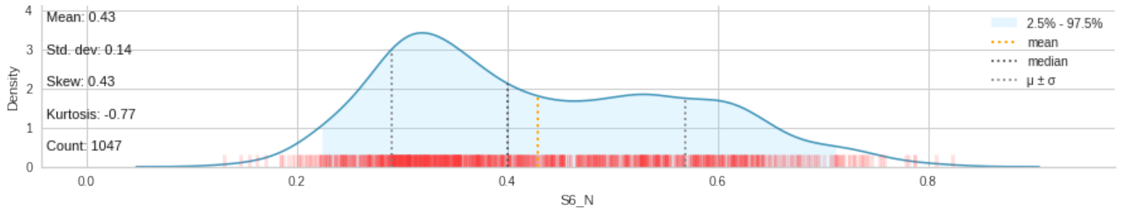

Mean: 0.18
Std. dev: 0.03
Skew: 0.75
Kurtosis: 0.88
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
NUMB_N

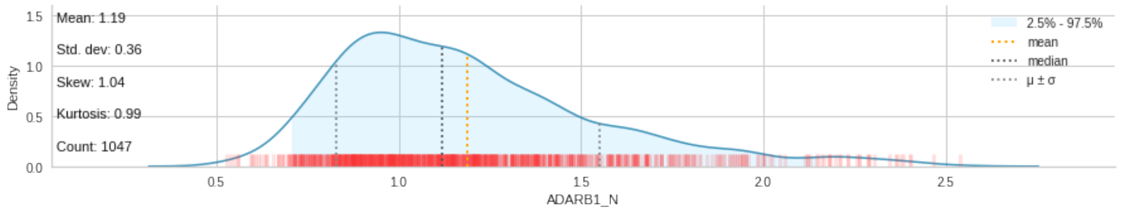

Mean: 0.95
Std. dev: 0.17
Skew: 0.42
Kurtosis: 0.49
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
P70S6_N

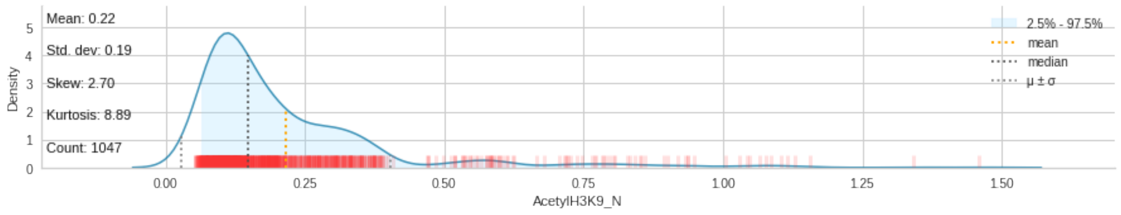

Mean: 0.16
Std. dev: 0.02
Skew: 0.53
Kurtosis: 1.28
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
pGSK3B_N

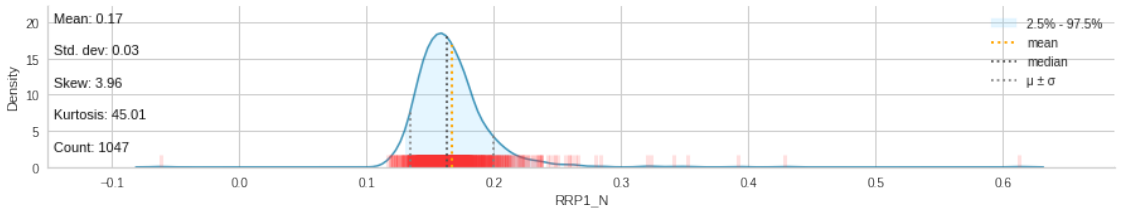

Mean: 1.69
Std. dev: 0.57
Skew: 0.31
Kurtosis: -0.41
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
pPKCG_N

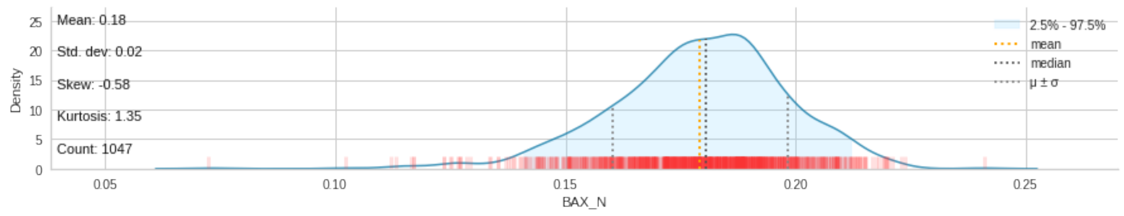

Mean: 0.29
Std. dev: 0.04
Skew: 2.46
Kurtosis: 34.44
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
CDK5_N


Mean: 0.43
Std. dev: 0.14
Skew: 0.43
Kurtosis: -0.77
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
S6_N

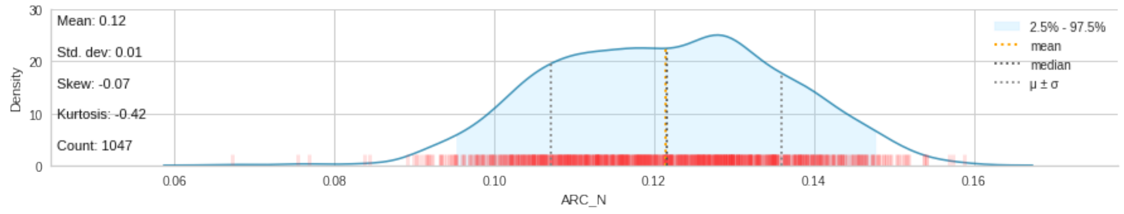

Mean: 1.19
Std. dev: 0.36
Skew: 1.04
Kurtosis: 0.99
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
ADARB1_N


Mean: 0.22
Std. dev: 0.19
Skew: 2.70
Kurtosis: 8.89
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
AcetylH3K9_N

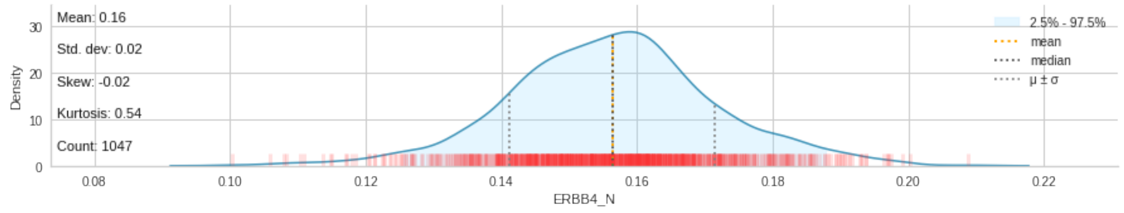

Mean: 0.17
Std. dev: 0.03
Skew: 3.96
Kurtosis: 45.01
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
RRP1_N



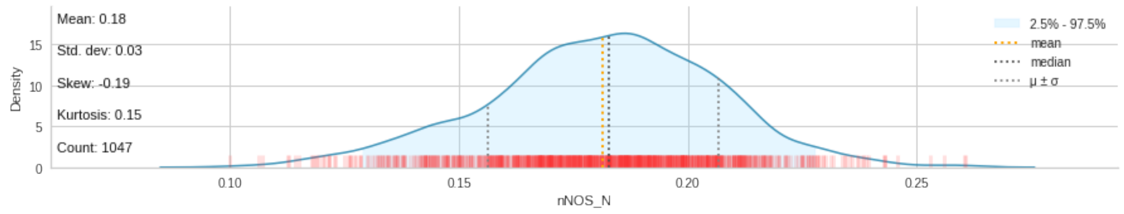
Mean: 0.18
Std. dev: 0.02
Skew: -0.58
Kurtosis: 1.35
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
BAX_N



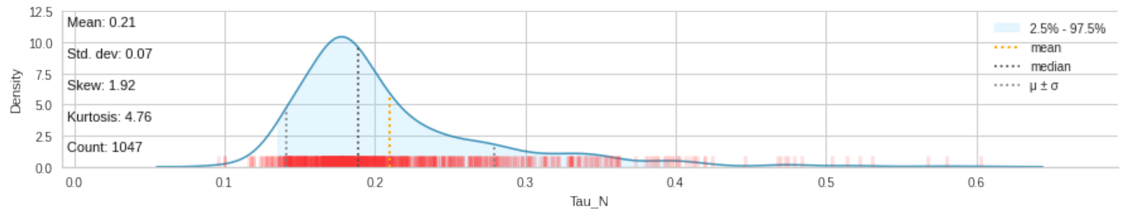
Mean: 0.12
Std. dev: 0.01
Skew: -0.07
Kurtosis: -0.42
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
ARC_N



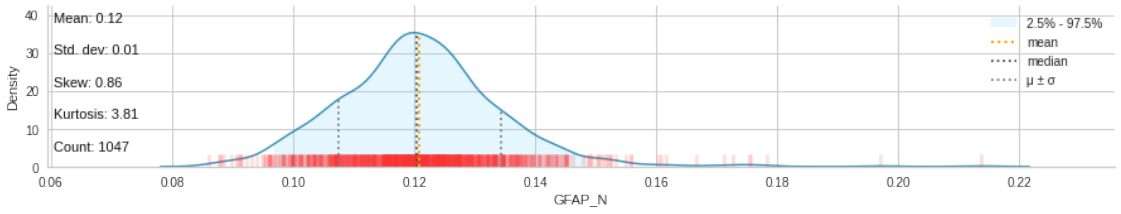
Mean: 0.16
Std. dev: 0.02
Skew: -0.02
Kurtosis: 0.54
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
ERBB4_N

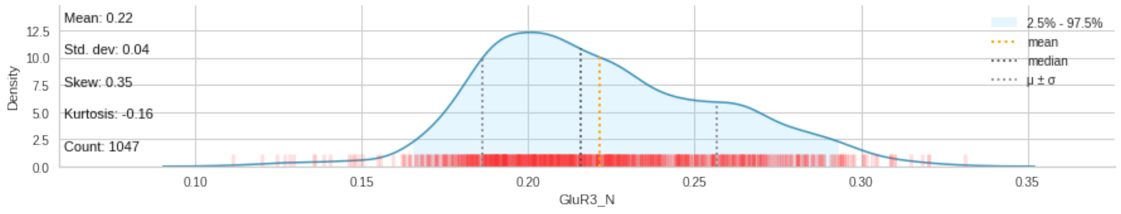


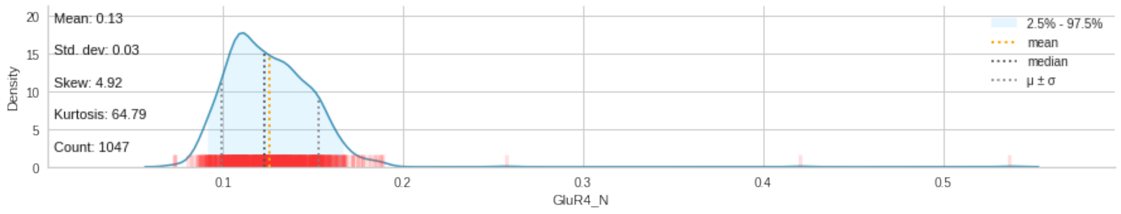
Mean: 0.18
Std. dev: 0.03
Skew: -0.19
Kurtosis: 0.15
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
nNOS_N

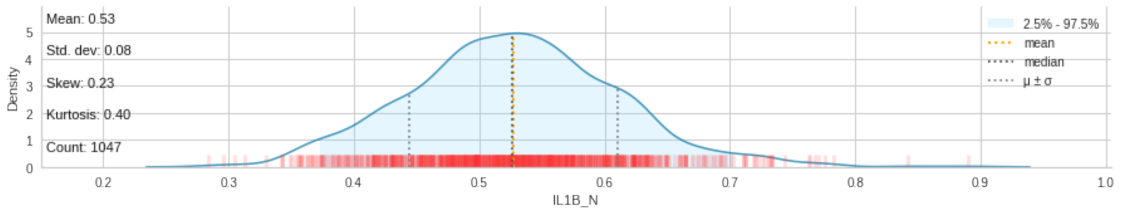


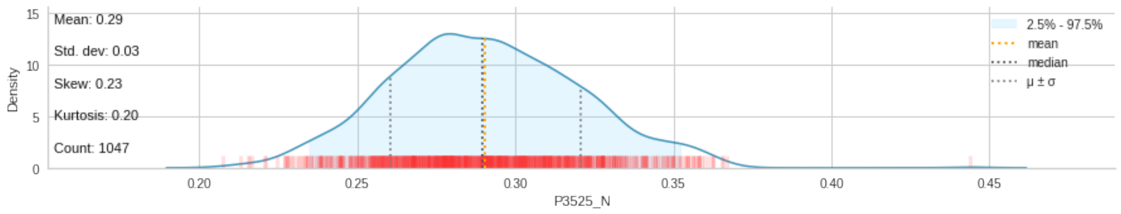
Mean: 0.21
Std. dev: 0.07
Skew: 1.92
Kurtosis: 4.76
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
Tau_N

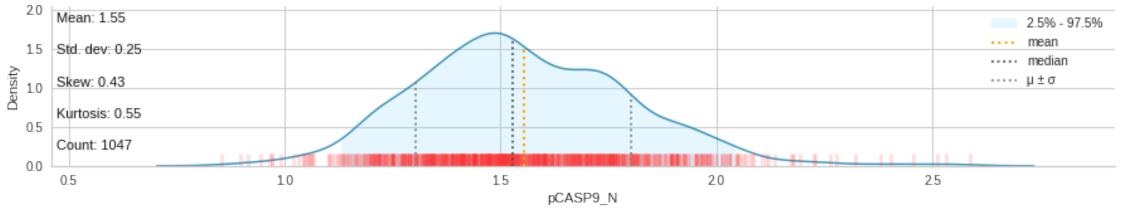

Mean: 0.12
Std. dev: 0.01
Skew: 0.86
Kurtosis: 3.81
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
GFAP_N


Mean: 0.22
Std. dev: 0.04
Skew: 0.35
Kurtosis: -0.16
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
GluR3_N

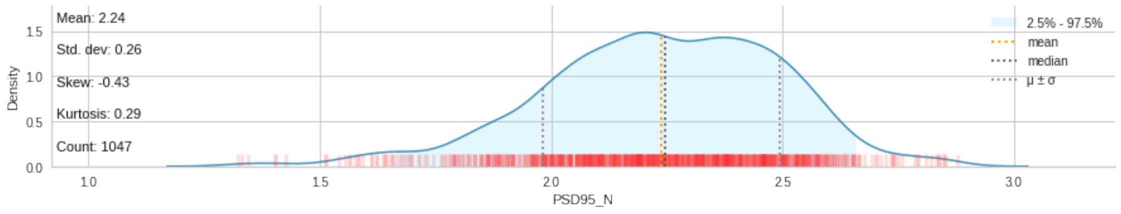

Mean: 0.13
Std. dev: 0.03
Skew: 4.92
Kurtosis: 64.79
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
GluR4_N


Mean: 0.53
Std. dev: 0.08
Skew: 0.23
Kurtosis: 0.40
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
IL1B_N

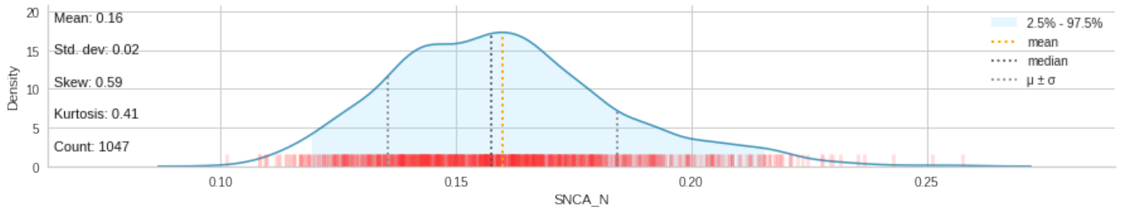

Mean: 0.29
Std. dev: 0.03
Skew: 0.23
Kurtosis: 0.20
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
P3525_N



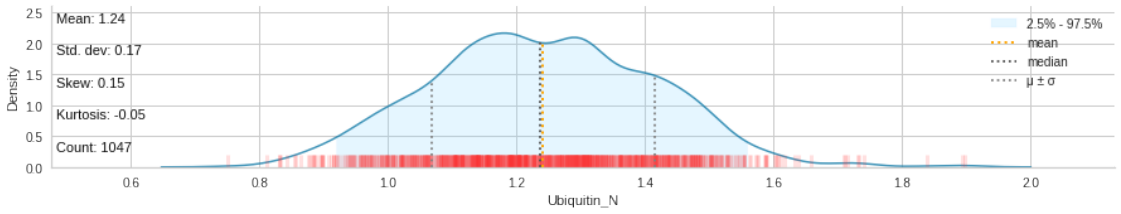
Mean: 1.55
Std. dev: 0.25
Skew: 0.43
Kurtosis: 0.55
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
pCASP9_N



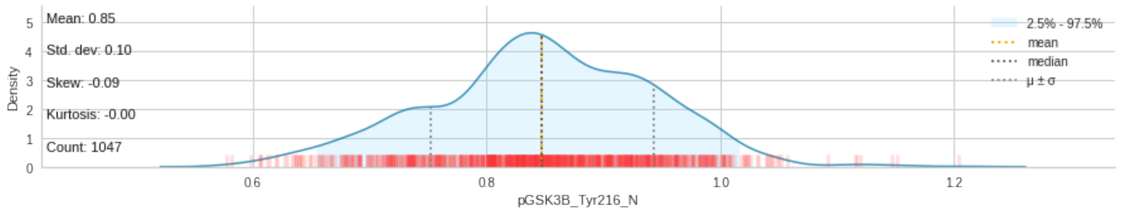
Mean: 2.24
Std. dev: 0.26
Skew: -0.43
Kurtosis: 0.29
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
PSD95_N



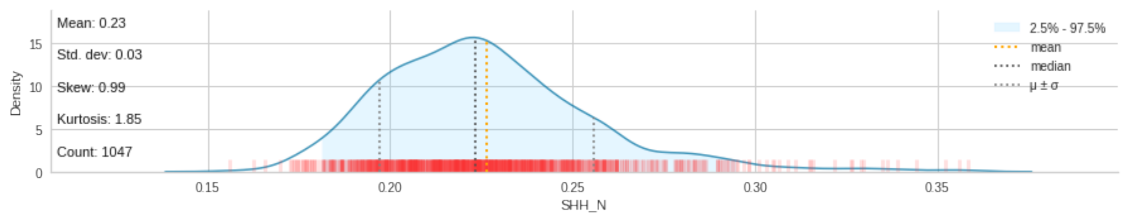
Mean: 0.16
Std. dev: 0.02
Skew: 0.59
Kurtosis: 0.41
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
SNCA_N

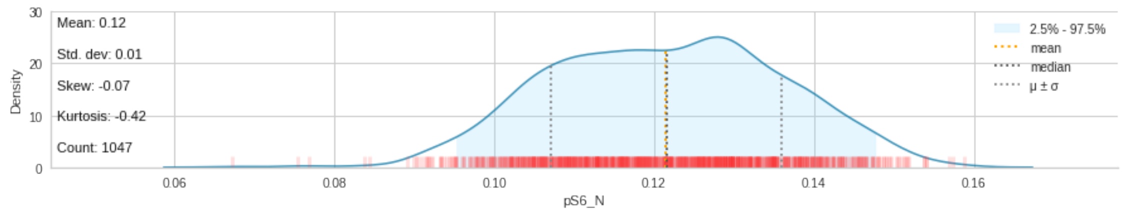


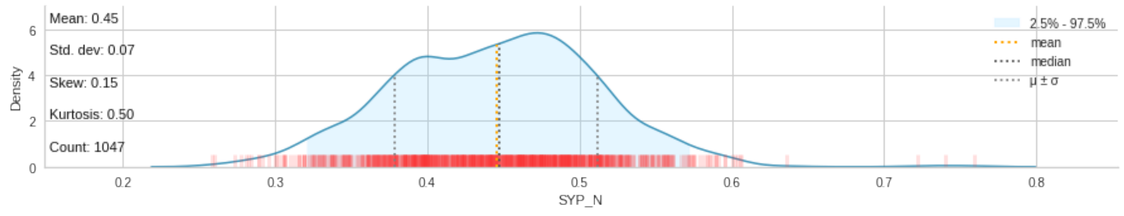
Mean: 1.24
Std. dev: 0.17
Skew: 0.15
Kurtosis: -0.05
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
Ubiquitin_N

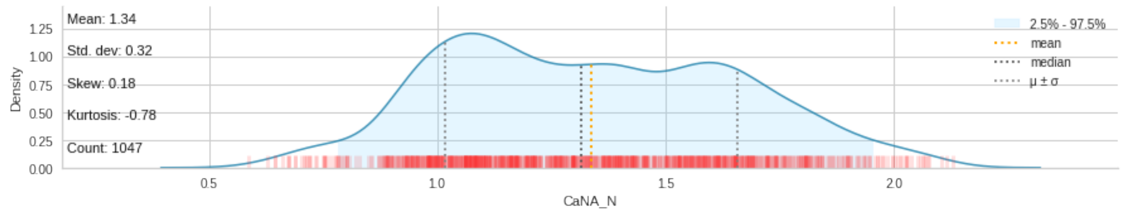


Mean: 0.85
Std. dev: 0.10
Skew: -0.09
Kurtosis: -0.00
Count: 1047
2.5% - 97.5%
mean
median
μ ± σ
Density
pGSK3B_Tyr216_N









1.6 Preprocessing (Data Splitting)

We initialize our Data() class, which will split our data into training and testing sets. Prior to this, it will also standardize the entire dataset.

[95]:
```
data = Data(df, "class", train_proportion=0.70)
print(data)
```

```
Data(1047 Observations x 72 Variables)
   X Train:    (732, 71)
   Y Train:    (732,)
   X Test:     (315, 71)
   Y Test:     (315,)
   Train Size: 0.7
```

1.7 Feature Reduction (Dimensionality Reduction via PCA)

A high dimensional space makes clustering difficult in many algorithms. To remedy this, we perform PCA to work in a *much* lower dimensional space. This should increase the efficiency and performance of our model.

[96]:
```python
data.pca(explained_variance_threshold=0.8)
print(data)
print(data.features)
```

```
Data(1047 Observations x 72 Variables)
   X Train:    (732, 9)
   Y Train:    (732,)
   X Test:     (315, 9)
   Y Test:     (315,)
   Train Size: 0.7
['MEK_N', 'SNCA_N', 'P70S6_N', 'pPKCG_N', 'GFAP_N', 'pPKCG_N', 'Tau_N',
'pGSK3B_Tyr216_N', 'GluR4_N']
```

1.8 Model Building

Here, we will initialize our K-Means model. Additionally, we create a function for comparing cluster indices to the true training labels. We will assign clusters to the labels that provide the highest accuracy and maintain the same mapping when we make predictions from the testing data.

[97]:
```python
def link_labels(cluster_pred, label_truth, labels=None):
    """Links cluster predictions to labels based on accuracy.

    Parameters
    ----------
    cluster_pred: np.ndarray
        1D array of cluster index predictions
    label_truth: Iterable
        1D iterable of the true labels
    labels: None or iterable of strings
        The potential labels. If None, potential labels are inferred directly
        from label_truth

    Returns
    -------
    best_mapping: dictionary
        The best mapping between a label and a cluster index
    """
    if labels is None:
        labels = pd.unique(label_truth)
    # Maintain a best f1 and mapping
    best_acc = 0
    best_mapping = None

    n = len(cluster_pred)

    # Iterate through each potential permutation
    for perm in permutations(range(len(labels))):
        # Generate a mapping between cluster index and label
        mapping = {labels[a]: i for i, a in enumerate(perm)}

        # Apply the mapping the the labeled data
        cluster_true = np.array([mapping[label] for label in label_truth])

        # Compute the accuracy
        acc = np.sum(cluster_pred == cluster_true) / n

        if acc > best_acc:
            best_acc = acc
            best_mapping = mapping

    return best_mapping
```

[98]:
```
model = KMeans(n_clusters=8, n_init=200, max_iter=2000)
```

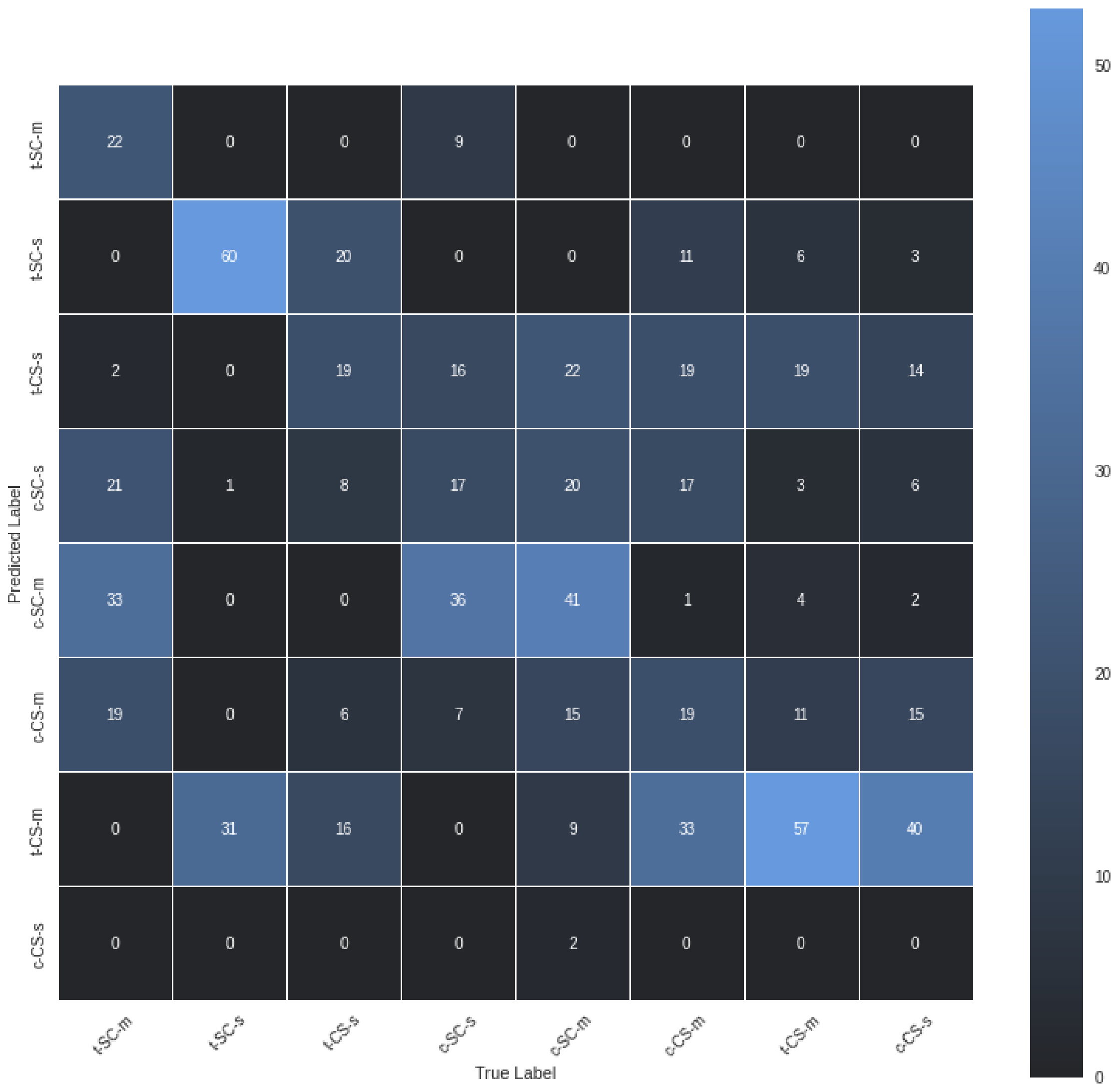
1.9 Model Training and Training Performance

[99]:
```
# Fit the model to the training data
model.fit(data.x_train)

# Extract the cluster indices
clusters_fitted_x = model.labels_
```

1.9.1 Classification

[100]:
```
# Create the best mapping between labels and clusters
mapping = link_labels(clusters_fitted_x, data.y_train)
mapping_r = {v:k for k, v in mapping.items()}
display(mapping)

# Convert the labels to cluster indices
clusters_train_y = np.array([mapping[label] for label in data.y_train])

# Create a confusion matrix
tester = DataTesting(
    labels_pred=clusters_fitted_x,
    labels_truth=clusters_train_y,
    decoder=mapping_r
)

tester.confusion_matrix_plot()
display(tester.report())
```

```
{'t-SC-m': 0,
 't-SC-s': 1,
 't-CS-s': 2,
 'c-SC-s': 3,
 'c-SC-m': 4,
 'c-CS-m': 5,
 't-CS-m': 6,
 'c-CS-s': 7}
              t-SC-m     t-SC-s     t-CS-s     c-SC-s      c-SC-m      c-CS-m  \
precision   0.709677   0.600000   0.171171   0.182796    0.350427    0.206522
recall      0.226804   0.652174   0.275362   0.200000    0.376147    0.190000
f1-score    0.343750   0.625000   0.211111   0.191011    0.362832    0.197917
support    97.000000  92.000000  69.000000  85.000000  109.000000  100.000000

               t-CS-m  c-CS-s  accuracy   macro avg  weighted avg
precision    0.306452     0.0  0.321038    0.315881      0.329073
recall       0.570000     0.0  0.321038    0.311311      0.321038
f1-score     0.398601     0.0  0.321038    0.291278      0.301703
support    100.000000    80.0  0.321038  732.000000    732.000000
```



1.9.2 Cluster Metrics

[101]:
```
# Count the number of elements in each cluster
cluster, count = np.unique(clusters_fitted_x, return_counts=True)

display(pd.DataFrame(count, columns=["Frequency"], index=[f"Cluster {a}" for a␣
 ↪in cluster]))
print(f"KMeans Inertia: {model.inertia_}")
```

```
           Frequency
Cluster 0         31
Cluster 1        100
Cluster 2        111
Cluster 3         93
Cluster 4        117
Cluster 5         92
Cluster 6        186
Cluster 7          2
```

```
KMeans Inertia: 3252.990848746761
```

1.10 Model Evaluation and Results

Now, we will predict from the testing data and make a confusion matrix and classification report similar to the training data.

[102]:
```
# Predict clusters
pred = model.predict(data.x_test)
```

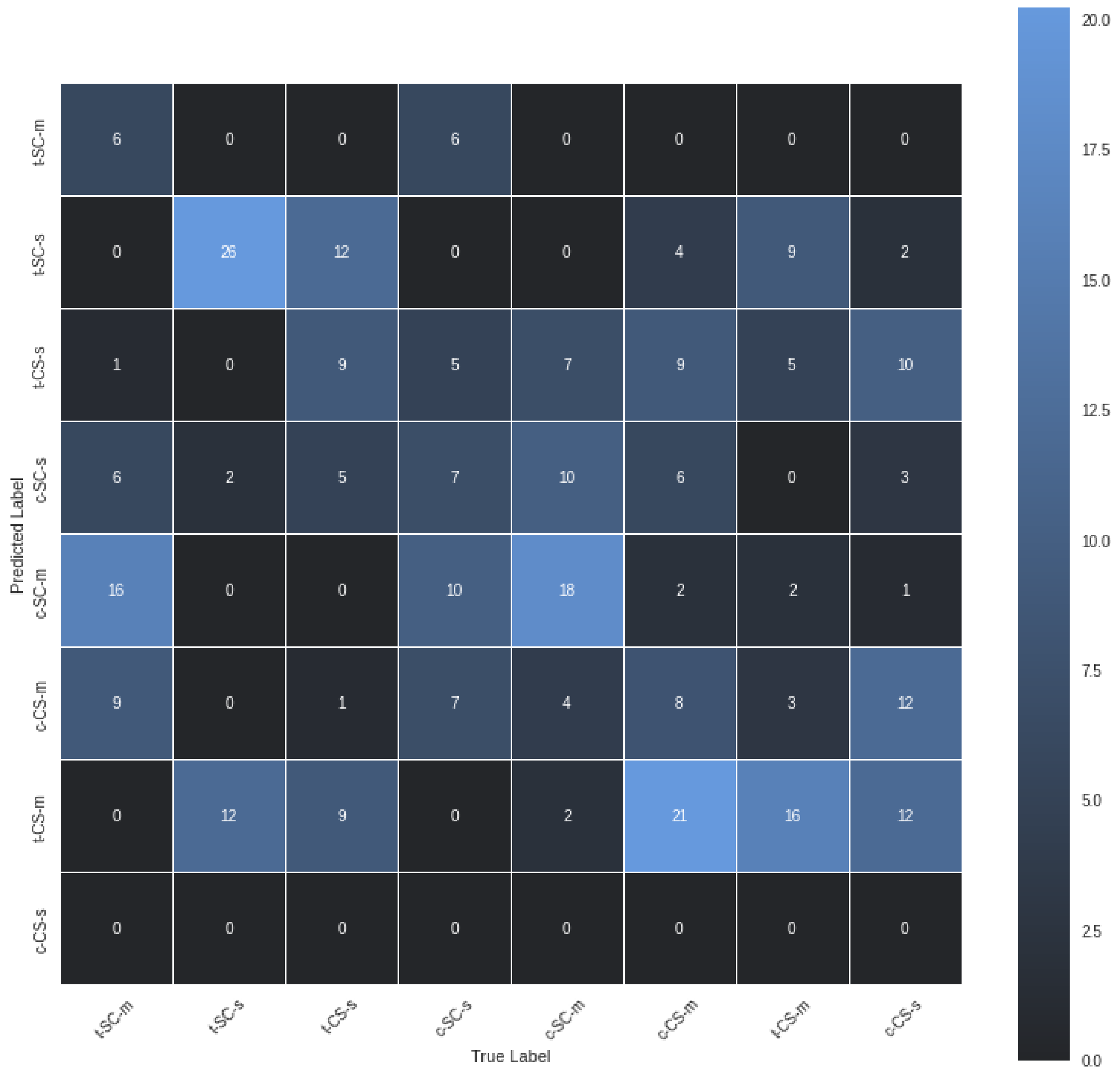
1.10.1 Classification

[103]:
```
# Convert labels to cluster indices
true = np.array([mapping[label] for label in data.y_test])

# Create a confusion matrix
tester = DataTesting(
    labels_pred=pred,
    labels_truth=true,
    decoder=mapping_r
)

tester.confusion_matrix_plot()
display(tester.report())
```

```
/home/benjamin/.local/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1318: 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))
/home/benjamin/.local/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1318: 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))
/home/benjamin/.local/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1318: 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))
```

```
              t-SC-m    t-SC-s    t-CS-s    c-SC-s    c-SC-m    c-CS-m  \
precision   0.500000  0.490566  0.195652  0.179487  0.367347  0.181818
recall      0.157895  0.650000  0.250000  0.200000  0.439024  0.160000
```

```
f1-score     0.240000   0.559140   0.219512   0.189189   0.400000   0.170213
support     38.000000  40.000000  36.000000  35.000000  41.000000  50.000000

               t-CS-m  c-CS-s  accuracy   macro avg  weighted avg
precision    0.222222     0.0  0.285714    0.267137      0.266280
recall       0.457143     0.0  0.285714    0.289258      0.285714
f1-score     0.299065     0.0  0.285714    0.259640      0.258373
support     35.000000    40.0  0.285714  315.000000    315.000000
```



	t-SC-m	t-SC-s	t-CS-s	c-SC-s	c-SC-m	c-CS-m	t-CS-m	c-CS-s
t-SC-m	6	0	0	6	0	0	0	0
t-SC-s	0	26	12	0	0	4	9	2
t-CS-s	1	0	9	5	7	9	5	10
c-SC-s	6	2	5	7	10	6	0	3
c-SC-m	16	0	0	10	18	2	2	1
c-CS-m	9	0	1	7	4	8	3	12
t-CS-m	0	12	9	0	2	21	16	12
c-CS-s	0	0	0	0	0	0	0	0

1.11 Conclusion

Overall, it may initially be difficult to say that the model performed well. With an overall accuracy of 29%, an average precision of 27%, an average recall of 29%, and an average F1 score of 26%. This model performs approximately twice as well as a random model. With 8 class, we expect a random class to have an accuracy of `12.5%`. With a total of 71 features, there are many co-varying features. This means that, while we had a lot of information, much of the information was redundant and did not have predictive power. Additionally, with 1047 total observations, it may be difficult to build a powerful model. A purely supervised approach would likely work much better since the 8 classes may be too similar to confidently distinguish.