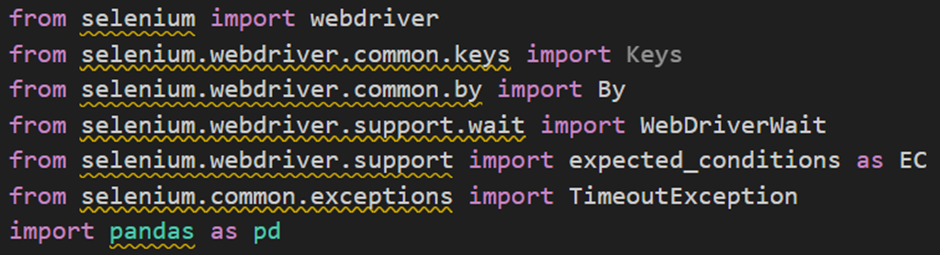
**Code Explanation**

**Web Scraping**



*FIGURE 1: WEB SCRAPING MODULES*

1. `from selenium import webdriver`: This imports the `webdriver` module from the Selenium library, which allows interaction with web browsers.2. `from selenium.webdriver.common.keys import Keys`: This imports the `Keys` class from the `webdriver.common.keys` module, which is used for sending keyboard keys like Enter, Escape, etc.3. `from selenium.webdriver.common.by import By`: This imports the `By` class from the `webdriver.common.by` module, which is used to locate elements on a webpage.4. `from selenium.webdriver.support.wait import WebDriverWait`: This imports the `WebDriverWait` class from the `webdriver.support.wait` module, which allows waiting for certain conditions to occur before proceeding with the code execution.5. `from selenium.webdriver.support import expected\_conditions as EC`: This imports the `expected\_conditions` module from the `webdriver.support` package, which provides a set of predefined conditions to wait for.6. `from selenium.common.exceptions import TimeoutException`: This imports the `TimeoutException` class from the `selenium.common.exceptions` module, which is raised when a timeout occurs during waiting.7. `import pandas as pd`: This imports the Pandas library, which is used for data manipulation and analysis.

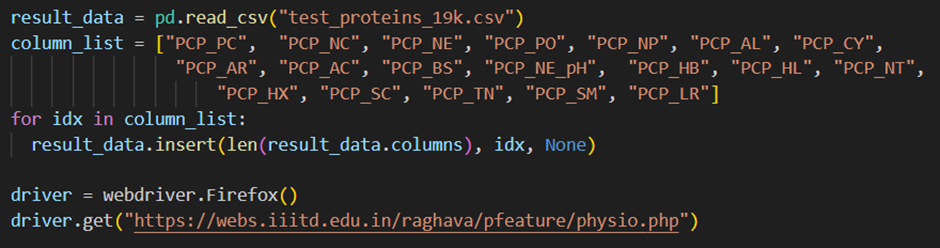
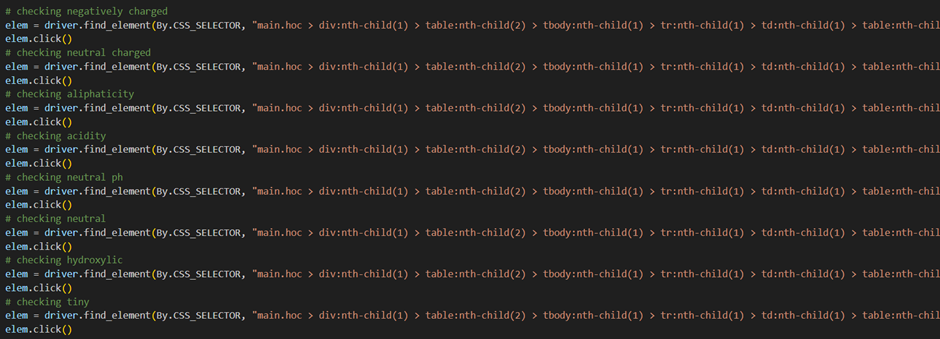


Figure 2 : - Making a data frame to store values

1. result\_data = pd.read\_csv("test\_proteins\_19k.csv")`result\_data`: This is a variable that will hold the DataFrame created by reading the CSV file.`pd.read\_csv("test\_proteins\_19k.csv")`: This function from the Pandas library reads the CSV file named "test\_proteins\_19k.csv" and returns a DataFrame. The DataFrame is assigned to the variable `result\_data`.2. column\_list = ["PCP\_PC", "PCP\_NC", "PCP\_NE", "PCP\_PO", "PCP\_NP", "PCP\_AL", "PCP\_CY", "PCP\_AR", "PCP\_AC", "PCP\_BS", "PCP\_NE\_pH", "PCP\_HB", "PCP\_HL", "PCP\_NT", "PCP\_HX", "PCP\_SC", "PCP\_TN", "PCP\_SM", "PCP\_LR"]`column\_list`: This is a variable that holds a list of column names (strings) that we want to add to the DataFrame.3. for idx in column\_list:`for idx in column\_list:`: This is a for loop that iterates over each element in `column\_list`. In each iteration, `idx` will take the value of the current element from `column\_list`.4. result\_data.insert(len(result\_data.columns), idx, None)`result\_data.insert(...)`: This function inserts a new column into the DataFrame.`len(result\_data.columns)`: This gets the current number of columns in the DataFrame. The new column will be inserted at this position, effectively adding it to the end of the DataFrame.`idx`: This is the name of the new column to be added, which comes from the current element in the `column\_list` during the iteration.`None`: This sets the default value of all the entries in the new column to `None`.5. driver = webdriver.Firefox()`driver`: This is a variable that will hold the instance of the Firefox web browser.`webdriver.Firefox()`: This creates a new instance of the Firefox web browser using the Selenium `webdriver`. This browser instance will be controlled programmatically by Selenium.6. driver.get("https://webs.iiitd.edu.in/raghava/pfeature/physio.php")`driver.get(...)`: This method instructs the Selenium-controlled browser to navigate to a specified URL.`"https://webs.iiitd.edu.in/raghava/pfeature/physio.php"`: This is the URL that the browser will navigate to. In this case, it is the URL of a web page related to a tool or feature hosted on the IIIT-Delhi website.



*Figure 23 Navigating the HTML structure*

1. elem = driver.find\_element(By.CSS\_SELECTOR, "main.hoc > div:nth-child(1) > table:nth-child(2) > tbody:nth-child(1) > tr:nth-child(1) > td:nth-child(1) > table:nth-child(7) > tbody:nth-child(1) > tr:nth-child(2) > td:nth-child(2) > input:nth-child(1)")`elem`: This variable will hold the web element that is found by the Selenium `find\_element` method.`driver.find\_element(...)`: This method is used to locate a single web element on the web page. It returns the first matching element.`By.CSS\_SELECTOR`: This specifies that the method should use a CSS selector to locate the element. The `By` class is part of Selenium and is used to specify the method of locating elements.`"main.hoc > div:nth-child(1) > table:nth-child(2) > tbody:nth-child(1) > tr:nth-child(1) > td:nth-child(1) > table:nth-child(7) > tbody:nth-child(1) > tr:nth-child(2) > td:nth-child(2) > input:nth-child(1)"`: This is the CSS selector string used to precisely locate the desired element within the web page. The selector specifies a path through the DOM (Document Object Model) to find the element. The CSS selector string can be broken down as follows:- `main.hoc`: Selects a `<main>` element with the class `hoc`.- `> div:nth-child(1)`: Selects the first child `<div>` of the `main.hoc` element.- `> table:nth-child(2)`: Selects the second child `<table>` of the first `<div>`.- `> tbody:nth-child(1)`: Selects the first child `<tbody>` of the second `<table>`.- `> tr:nth-child(1)`: Selects the first child `<tr>` of the first `<tbody>`.- `> td:nth-child(1)`: Selects the first child `<td>` of the first `<tr>`.- `> table:nth-child(7)`: Selects the seventh child `<table>` of the first `<td>`.- `> tbody:nth-child(1)`: Selects the first child `<tbody>` of the seventh `<table>`.- `> tr:nth-child(2)`: Selects the second child `<tr>` of the first `<tbody>`.- `> td:nth-child(2)`: Selects the second child `<td>` of the second `<tr>`.- `> input:nth-child(1)`: Selects the first child `<input>` of the second `<td>`. 2. elem.click()`elem.click()`: This method is called on the web element stored in the `elem` variable. It simulates a mouse click on the located element, which is typically used to interact with buttons, links, or other clickable elements on a web page.

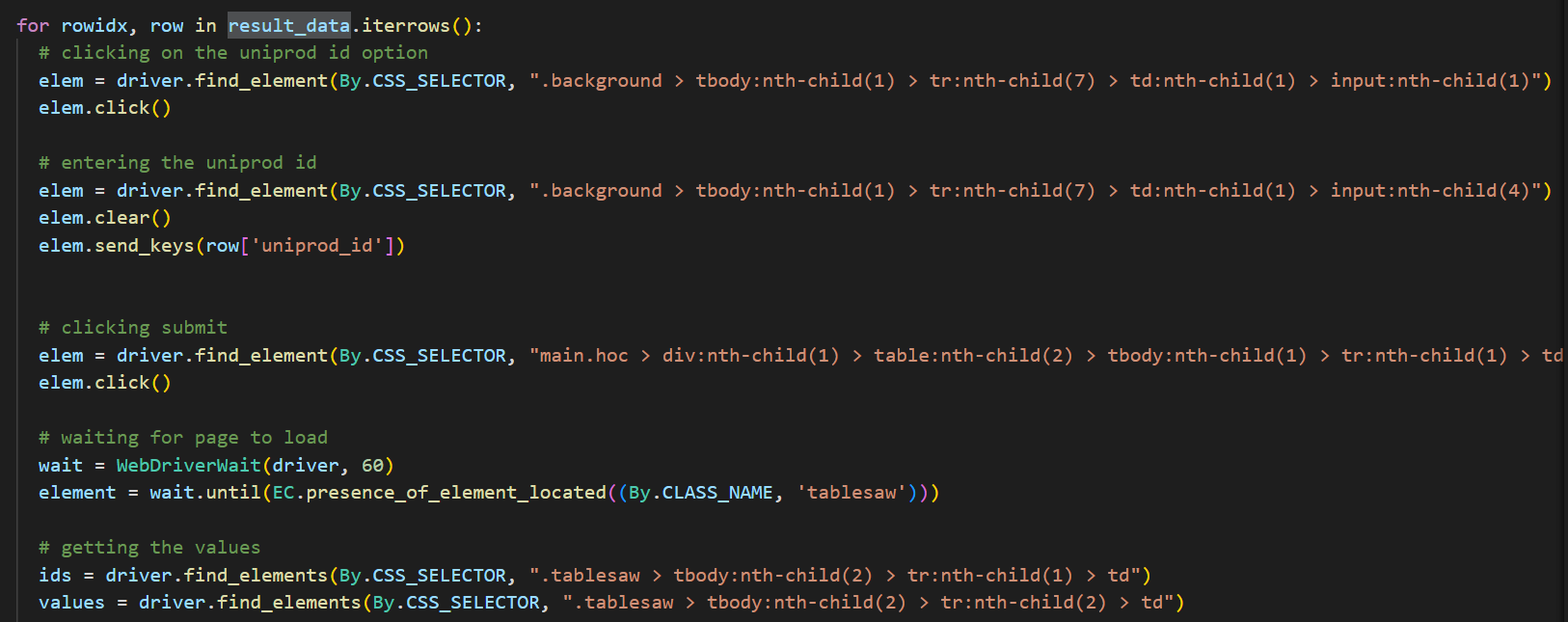


Figure: 27 Locating elements in the website

1. for rowidx, row in result\_data.iterrows():`for rowidx, row in result\_data.iterrows():`: This is a for loop that iterates over the rows of the DataFrame `result\_data`. - `result\_data.iterrows()`: This method generates an iterator over the DataFrame rows as (index, Series) pairs. - `rowidx`: This variable will store the index of the current row. - `row`: This variable will store the Series object representing the current row of the DataFrame.2. elem.click()`elem.click()`: This method is called on the web element stored in the `elem` variable. It simulates a mouse click on the located element, which is typically used to interact with buttons, links, or other clickable elements on a web page.3. elem.clear()`elem.clear()`: This method is called on the web element stored in the `elem` variable. It clears any existing text or value in the input field, making it ready for new input.4. elem.send\_keys(row['uniprod\_id'])`elem.send\_keys(...)`: This method is called on the web element stored in the `elem` variable. It simulates typing into the input field.`row['uniprod\_id']`: This accesses the value in the 'uniprod\_id' column of the current row in the DataFrame. This value is sent as input to the located input element.5. wait = WebDriverWait(driver, 60)`wait`: This variable will hold an instance of the `WebDriverWait` class. `WebDriverWait(driver, 60)`: This initializes a `WebDriverWait` object with the following parameters: - `driver`: The Selenium WebDriver instance controlling the browser. - `60`: The maximum number of seconds to wait for a condition to be met. In this case, it will wait up to 60 seconds. 6. element = wait.until(EC.presence\_of\_element\_located((By.CLASS\_NAME, 'tablesaw')))`element`: This variable will hold the web element that is found by the `until` method.`wait.until(...)`: This method is called on the `WebDriverWait` instance to wait until a specified condition is met. It returns the web element once the condition is satisfied.`EC.presence\_of\_element\_located(...)`: This is the condition to wait for, which is provided by the `expected\_conditions` (imported as `EC`) module in Selenium. This particular condition waits for the presence of an element located by a specific method.`(By.CLASS\_NAME, 'tablesaw')`: This is a tuple specifying the method of locating the element and the value to locate it by: - `By.CLASS\_NAME`: Specifies that the element should be located by its class name. - `'tablesaw'`: The class name to locate the element by.7. ids = driver.find\_elements(By.CSS\_SELECTOR, ".tablesaw > tbody:nth-child(2) > tr:nth-child(1) > td")`ids`: This variable will hold a list of web elements found by the `find\_elements` method.`driver.find\_elements(...)`: This method is used to locate multiple web elements on the web page. It returns a list of matching elements.

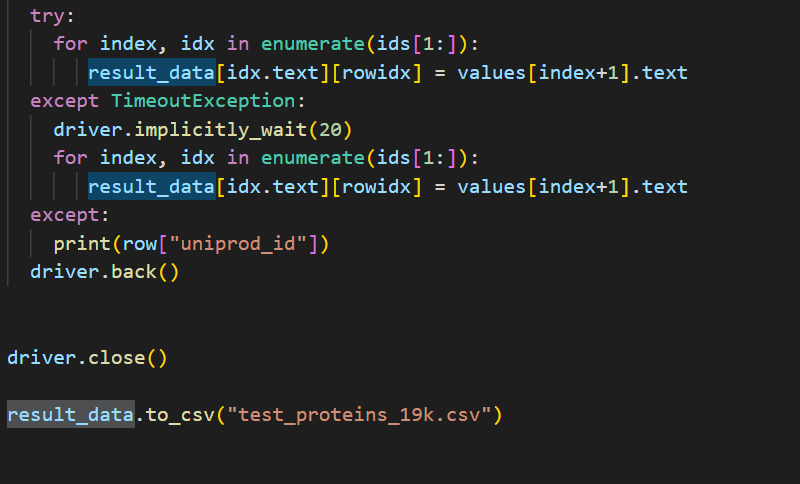
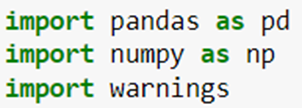


Figure 28 Exception Handling

1. try:`try:`: This starts a try block to handle exceptions that may occur within the block of code that follows.2. for index, idx in enumerate(ids[1:]):`for index, idx in enumerate(ids[1:]):`: This is a for loop that iterates over the elements in `ids` starting from the second element (index 1). - `enumerate(ids[1:])`: This function provides both the index and the value of each element in the iterable `ids[1:]`. - `index`: This variable holds the index of the current iteration. - `idx`: This variable holds the current element from `ids`.3. result\_data[idx.text][rowidx] = values[index+1].text`result\_data[idx.text][rowidx]`: This accesses the cell in the DataFrame `result\_data` at the column named `idx.text` and the row indexed by `rowidx`.`values[index+1].text`: This gets the text content of the element in `values` at the position `index + 1`.`=`: This assigns the text content from `values` to the corresponding cell in the DataFrame.4. except TimeoutException:`except TimeoutException:`: This starts an except block to handle `TimeoutException` errors that might occur in the try block.5. driver.implicitly\_wait(20)`driver.implicitly\_wait(20)`: This sets an implicit wait of 20 seconds for the WebDriver. This means that the WebDriver will wait up to 20 seconds for elements to become available before throwing an exception.6. for index, idx in enumerate(ids[1:]):`for index, idx in enumerate(ids[1:]):`: This is a for loop that iterates over the elements in `ids` starting from the second element (index 1). This loop is identical to the one in the try block and will be executed if a `TimeoutException` is caught.7. result\_data[idx.text][rowidx] = values[index+1].text`result\_data[idx.text][rowidx]`: This accesses the cell in the DataFrame `result\_data` at the column named `idx.text` and the row indexed by `rowidx`.`values[index+1].text`: This gets the text content of the element in `values` at the position `index + 1`.`=`: This assigns the text content from `values` to the corresponding cell in the DataFrame. This line is identical to the one in the try block.8. except:`except:`: This starts a general except block to handle any other exceptions that might occur.9. print(row["uniprod\_id"])`print(row["uniprod\_id"])`: This prints the value of the 'uniprod\_id' column from the current row of the DataFrame. This is used for debugging purposes to identify which row caused an exception.10. driver.back()`driver.back()`: This method navigates the browser back to the previous page in the browser history. This is useful if the script needs to return to a prior state to continue processing the next row.11. driver.close()`driver.close()`: This method closes the current browser window. If it's the only open window, it will shut down the WebDriver session.12. result\_data.to\_csv("test\_proteins\_19k.csv")`result\_data.to\_csv("test\_proteins\_19k.csv")`: This method saves the DataFrame `result\_data` to a CSV file named "test\_proteins\_19k.csv". It writes the DataFrame contents to the file, including any updates made during the script execution.

**Data Pre-processing and Model Building**



*Figure 29 Modules*

1. `pandas` is a popular data manipulation and analysis library in Python. It provides data structures and functions necessary to work with structured data, particularly in the form of dataframes.2. `numpy` is a library used for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.3. `warnings` is a built-in Python module used to handle warnings in a program. It allows developers to control how warnings are displayed or ignored.



Figure 30: - Importing dataset

1. `pd.read\_excel(...)`: This is a function provided by the pandas library to read data from an Excel file into a DataFrame. - The `read\_excel()` function takes the file path of the Excel file as its main argument. - In this case, the file name provided is 'mixture\_3.xlsx'. Pandas will attempt to locate this file in the current working directory unless the full file path is provided.2. `'mixture\_3.xlsx'`: This is the name of the Excel file from which data will be read. It should be located in the same directory as the script or Jupyter Notebook, unless the full path to the file is provided.3. `df`: This is the name assigned to the pandas DataFrame that will hold the data read from the Excel file. It's a common convention to name DataFrames `df` when they contain general tabular data, but you can choose any valid variable name.4. Explanation: - When this line of code is executed, pandas will read the data from the Excel file 'mixture\_3.xlsx' and store it in the DataFrame `df`. Each sheet in the Excel file will be read into a separate DataFrame if the Excel file contains multiple sheets. This DataFrame can then be used for further analysis, processing, or visualization within the Python environment.



Figure 31:- Getting first five entries

1. `df.head()`: This pandas DataFrame method retrieves the first few rows of the DataFrame `df`. - By default, it returns the first 5 rows. However, you can specify the number of rows by passing an integer argument, such as `df.head(10)` for the first 10 rows. - This method is commonly used to quickly inspect the structure and content of the DataFrame, facilitating initial data exploration and analysis.



Figure 32: - Detecting NaN values

1. `df.isna()`: This part of the code returns a DataFrame of the same shape as `df` where each element is `True` if the corresponding element in `df` is NaN (missing), and `False` otherwise. - The `isna()` method is used to identify missing values in the DataFrame.2. `.sum()`: This part of the code calculates the sum of boolean values (True = 1, False = 0) along each column axis. - Since True is equivalent to 1 and False is equivalent to 0 when summing, this effectively counts the number of missing values in each column. - By default, `sum()` sums along the column axis (axis=0). - This is helpful for quickly assessing the completeness of the data and determining if any columns contain missing values that need to be handled (e.g., imputed or dropped) before further analysis.



Figure 33: - checking for duplicates

1. `df.duplicated()`: This part of the code returns a Series of boolean values indicating whether each row in the DataFrame `df` is a duplicate of a previous row.2. Explanation: - When this line of code is executed, it computes the total number of duplicate rows in the DataFrame `df`. - This information is useful for identifying and possibly removing duplicate rows from the dataset to ensure data integrity and avoid biases in analysis or modeling caused by redundant information.



Figure 34: - checking dimensions of the data frame

1. `df.shape`: This is an attribute of the pandas DataFrame that returns a tuple representing the dimensions of the DataFrame. - The `shape` attribute returns two values: the number of rows and the number of columns, respectively.



Figure 35: - Dropping unnecessary columns

1. `df.drop(columns=['Unnamed: 0','uniprod\_id','Entry Name'])`: This is a pandas DataFrame method used to remove columns from the DataFrame. - The `drop()` method is used to drop specified labels (rows or columns) from the DataFrame. - The `columns` parameter specifies the labels of the columns to be dropped. In this case, the list `['Unnamed: 0','uniprod\_id','Entry Name']` contains the names of the columns to be removed: 'Unnamed: 0', 'uniprod\_id', and 'Entry Name'. - By default, the `drop()` method drops rows with the specified labels. However, to drop columns, you need to specify `axis=1` (or `axis='columns'`), which indicates that the labels to be dropped are column names.

2. `df=`: This assigns the modified DataFrame (after dropping the specified columns) back to the variable `df`. - By reassigning the DataFrame to the same variable (`df`), the original DataFrame is replaced with the modified DataFrame that no longer contains the specified columns. - This operation is useful for removing unnecessary or redundant columns from the DataFrame, which can streamline data processing, reduce memory usage, and improve the clarity of the DataFrame structure for subsequent analysis.



Figure 36: - Summary statistics for data frame

1. `df.describe()`: This is a pandas DataFrame method used to generate descriptive statistics for the DataFrame. - The `describe()` method computes various summary statistics for each numerical column in the DataFrame, including count, mean, standard deviation, minimum value, 25th percentile (Q1), median (50th percentile), 75th percentile (Q3), and maximum value. - The output provides valuable insights into the distribution, central tendency, and spread of the data within each numerical column, facilitating exploratory data analysis (EDA) and helping to identify potential issues or patterns in the data.

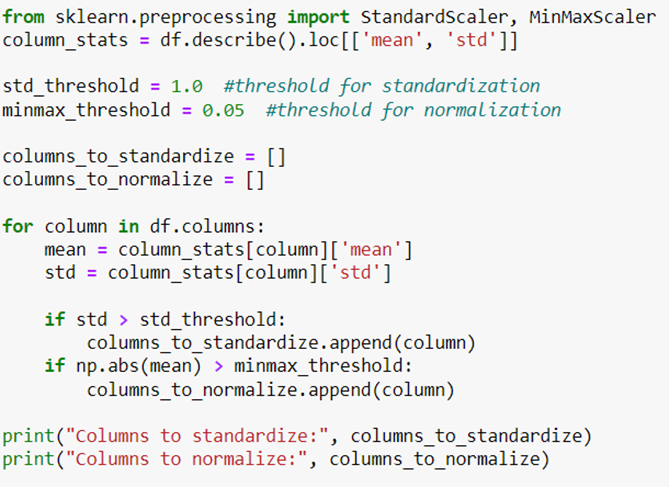


Figure 37: - Segregating columns to normalize and Standardize

1. `from sklearn.preprocessing import StandardScaler, MinMaxScaler`

- This line imports two classes from the `sklearn.preprocessing` module. `StandardScaler` is used for standardizing features by removing the mean and scaling to unit variance, while `MinMaxScaler` scales features to a given range (default is 0 to 1).

2. `column\_stats = df.describe().loc[['mean', 'std']]`

- This line calculates summary statistics for the dataframe `df` using the `describe()` method, which provides metrics like mean, standard deviation, min, max, and quartiles for each column.

- The result is then filtered to include only the 'mean' and 'std' (standard deviation) rows. `loc` is used for label-based indexing to select these specific rows.

3. `std\_threshold = 1.0 #threshold for standardization`

- This line defines a threshold for standardization. If the standard deviation of a column is greater than this value, the column will be considered for standardization.

4. `minmax\_threshold = 0.05 #threshold for normalization`

- This line defines a threshold for normalization. If the absolute mean of a column is greater than this value, the column will be considered for normalization.

5. `columns\_to\_standardize = []`

- This initializes an empty list to store the names of columns that need to be standardized.

6. `columns\_to\_normalize = []`

- This initializes an empty list to store the names of columns that need to be normalized.

7. `for column in df.columns:`

- This starts a loop that iterates over each column name in the dataframe `df`.

8. `mean = column\_stats[column]['mean']`

- Inside the loop, this line retrieves the mean value of the current column from `column\_stats`.

9. `std = column\_stats[column]['std']`

- Similarly, this line retrieves the standard deviation value of the current column from `column\_stats`.

10. `if std > std\_threshold:`

- This checks if the standard deviation of the current column is greater than the predefined `std\_threshold`.

11. `columns\_to\_standardize.append(column)`

- If the condition in the previous line is true, the column name is added to the `columns\_to\_standardize` list.

12. `if np.abs(mean) > minmax\_threshold:`

- This checks if the absolute mean of the current column is greater than the predefined `minmax\_threshold`.

13. `columns\_to\_normalize.append(column)`

- If the condition in the previous line is true, the column name is added to the `columns\_to\_normalize` list.

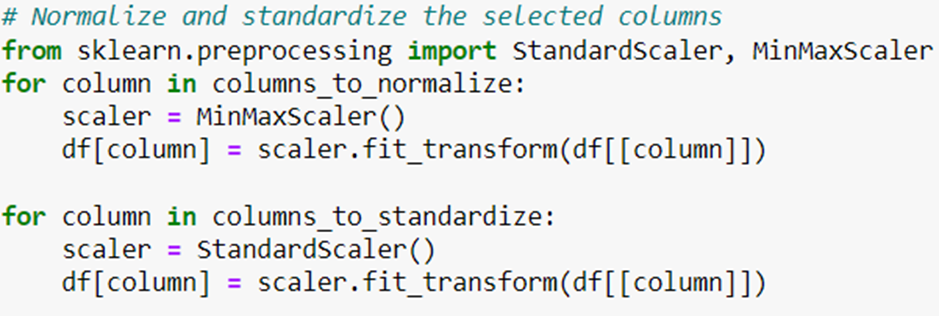


Figure 38: - Normalizing and Standardization of columns

1. `for column in columns\_to\_normalize:` - This initiates a loop over each column name stored in the `columns\_to\_normalize` list. These are the columns identified earlier as needing normalization based on the predefined criteria.2. `scaler = MinMaxScaler()` - Inside the loop, this creates an instance of the `MinMaxScaler` class, which will be used to scale the values of the current column to a specified range (by default, 0 to 1).3. `df[column] = scaler.fit\_transform(df[[column]])` - This line applies the `fit\_transform()` method of the scaler object to normalize the values of the current column in the dataframe `df`. The `fit\_transform()` method calculates the minimum and maximum values of the data and then scales the values accordingly.4. `for column in columns\_to\_standardize:` - This initiates another loop, this time over each column name stored in the `columns\_to\_standardize` list. These are the columns identified earlier as needing standardization based on the predefined criteria.5. `scaler = StandardScaler()` - Inside this loop, an instance of the `StandardScaler` class is created. This scaler will standardize the values of the current column by subtracting the mean and dividing by the standard deviation.6. `df[column] = scaler.fit\_transform(df[[column]])` - Similar to the normalization process, this line applies the `fit\_transform()` method of the `StandardScaler` object to standardize the values of the current column in the dataframe `df`.

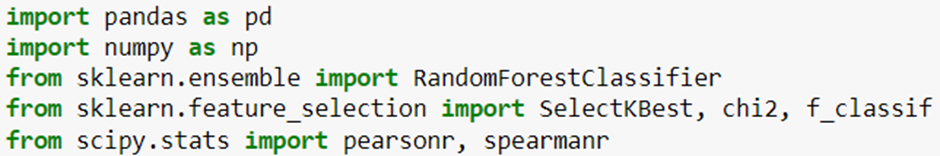


Figure 39: - Importing necessary modules

1. `from sklearn.ensemble import RandomForestClassifier` - This line imports the `RandomForestClassifier` class from the `sklearn.ensemble` module. RandomForest is a popular ensemble learning method used for classification tasks. It builds multiple decision trees during training and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

2. `from sklearn.feature\_selection import SelectKBest, chi2, f\_classif` - This line imports several feature selection techniques from the `sklearn.feature\_selection` module. `SelectKBest` is a feature selection method that selects the top k features based on a specified scoring function. `chi2` is a scoring function based on the chi-squared statistic, often used for feature selection with categorical target variables. `f\_classif` is a scoring function based on F-value between label/feature for classification tasks.3. `from scipy.stats import pearsonr, spearmanr` - This line imports the `pearsonr` and `spearmanr` functions from the `scipy.stats` module. These functions are used to compute the Pearson correlation coefficient and Spearman rank-order correlation coefficient, respectively. These coefficients measure the strength and direction of the linear and monotonic relationship between two variables.



Figure 40: - Counting unique values

1. `df['affects']` - This part of the code accesses the column named 'affects' in the DataFrame `df`. In pandas, you can access columns of a DataFrame using square brackets and the column name as a string within the brackets.2. `.value\_counts()` - This method is applied to the 'affects' column obtained in the previous step. It counts the occurrences of unique values in that column and returns the counts as a Series object, with the unique values as the index labels.

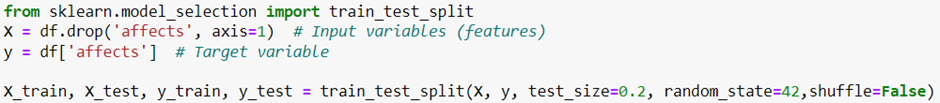


Figure 41: - Splitting of dataset in train and test

1. `from sklearn.model\_selection import train\_test\_split` - This line imports the `train\_test\_split` function from the `sklearn.model\_selection` module. This function is commonly used to split datasets into two subsets: one for training a machine learning model and the other for testing its performance.2. `X = df.drop('affects', axis=1)` - This line creates a DataFrame `X` which contains the input variables or features for the machine learning model. It is created by dropping the column named 'affects' from the original DataFrame `df`. The `axis=1` parameter specifies that the column should be dropped along the columns axis (i.e., horizontally).3. `y = df['affects']` - This line creates a Series `y` which contains the target variable for the machine learning model. It selects the column named 'affects' from the original DataFrame `df`. The target variable is the variable we want to predict using the input variables/features.4. `X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42,shuffle=False)` - This line splits the input features (`X`) and the target variable (`y`) into training and testing sets using the `train\_test\_split` function. - `X\_train` and `y\_train` represent the features and target variable for the training set, respectively. - `X\_test` and `y\_test` represent the features and target variable for the testing set, respectively. - `test\_size=0.2` specifies that 20% of the data will be reserved for testing, and the remaining 80% will be used for training. - `random\_state=42` ensures reproducibility of the split, as the same random seed will always produce the same random splitting of the data. - `shuffle=False` specifies that the data should not be shuffled before splitting. By default, `train\_test\_split` shuffles the data before splitting, but setting `shuffle=False` ensures that the data is split in sequential order.

Figure 42: - Checking Dependent and Independent variables

1. `X.head()` - This line accesses the first few rows of the DataFrame `X`, which contains the input variables or features for the machine learning model. The `head()` method in pandas returns the first n rows of a DataFrame (by default, the first 5 rows if no value is specified).2. `y.head()` - This line accesses the first few rows of the Series `y`, which contains the target variable for the machine learning model. The `head()` method in pandas returns the first n elements of a Series (by default, the first 5 elements if no value is specified).

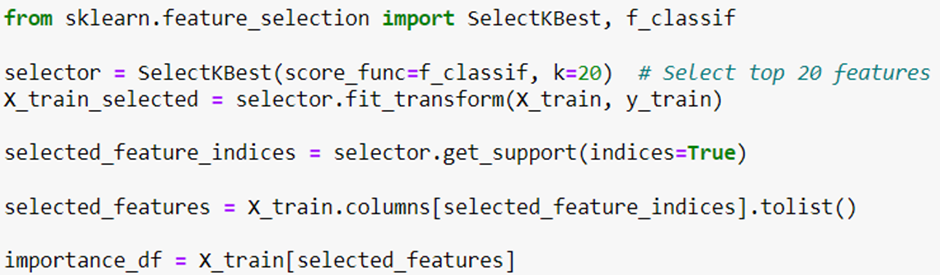


Figure 43: - Feature selection

1. `from sklearn.feature\_selection import SelectKBest, f\_classif`

- This line imports the `SelectKBest` class and the `f\_classif` function from the `sklearn.feature\_selection` module. `SelectKBest` is a feature selection method that selects the top k features based on a specified scoring function. `f\_classif` is one of the scoring functions available, which computes the ANOVA F-value between the feature and target for classification tasks.

2. `selector = SelectKBest(score\_func=f\_classif, k=20)`

- This line initializes an instance of the `SelectKBest` class with the `f\_classif` scoring function and specifies `k=20`, indicating that we want to select the top 20 features based on their ANOVA F-values.

3. `X\_train\_selected = selector.fit\_transform(X\_train, y\_train)`

- This line fits the feature selector (`selector`) to the training data (`X\_train` and `y\_train`) using the `fit\_transform()` method. This method computes the scores and selects the top k features based on the specified scoring function (`f\_classif` in this case). It then transforms the training data to include only these selected features.

4. `selected\_feature\_indices = selector.get\_support(indices=True)`

- This line retrieves the indices of the selected features from the fitted feature selector using the `get\_support()` method with `indices=True`. The `get\_support()` method returns a boolean mask indicating which features were selected (`True` for selected features, `False` for unselected features), and `indices=True` specifies that we want to retrieve the indices of the selected features.

5. `selected\_features = X\_train.columns[selected\_feature\_indices].tolist()`

- This line selects the names of the selected features from the column names of the training data (`X\_train`) using the indices obtained in the previous step. It converts the selected feature names to a list.

6. `importance\_df = X\_train[selected\_features]`

- This line creates a new DataFrame (`importance\_df`) containing only the selected features from the training data (`X\_train`) based on their importance scores. It selects columns from `X\_train` corresponding to the names of the selected features.

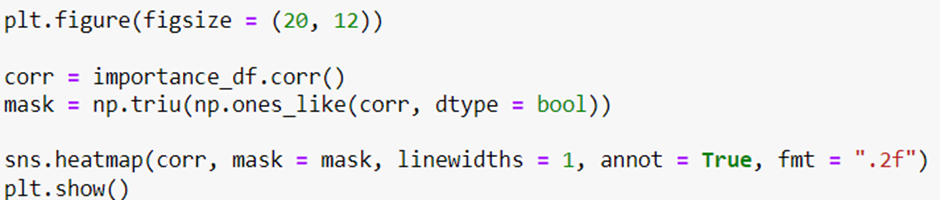


Figure 44: - Calculating correlational matrix

1. `plt.figure(figsize=(20, 12))`

- This line creates a new figure for the heatmap plot with a specified size. The `figsize` parameter specifies the width and height of the figure in inches. In this case, it sets the width to 20 inches and the height to 12 inches.

2. `corr = importance\_df.corr()`

- This line calculates the correlation matrix (`corr`) for the DataFrame `importance\_df`. The correlation matrix contains the pairwise correlation coefficients between all pairs of features in the DataFrame. Each element (i, j) in the matrix represents the correlation between the ith and jth features.

3. `mask = np.triu(np.ones\_like(corr, dtype=bool))`

- This line creates a mask (`mask`) to hide the lower triangular part of the correlation matrix. The `np.triu()` function creates an upper triangular matrix with ones and zeros elsewhere. `np.ones\_like(corr, dtype=bool)` creates a matrix of ones with the same shape as the correlation matrix `corr`. By applying `np.triu()` to this matrix, we get an upper triangular matrix of ones and zeros. This mask is used to hide the lower triangular part of the heatmap.

4. `sns.heatmap(corr, mask=mask, linewidths=1, annot=True, fmt=".2f")`

- This line creates a heatmap plot using the Seaborn library (`sns`). It visualizes the correlation matrix (`corr`) with the specified mask (`mask`) to hide the lower triangular part.

- `linewidths=1` sets the width of the lines separating each cell in the heatmap to 1 pixel.

- `annot=True` displays the correlation coefficients in each cell of the heatmap.

- `fmt=".2f"` specifies the format for displaying the correlation coefficients as floating-point numbers with two decimal places.

5. `plt.show()`

- This line displays the heatmap plot. It shows the correlation between different features in the dataset visually, with darker colors indicating stronger correlations.

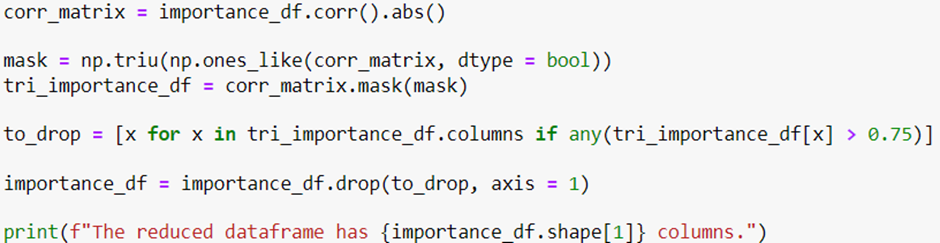


Figure 45: - Dropping highly correlated columns

1. `corr\_matrix = importance\_df.corr().abs()`

- This line calculates the absolute correlation matrix (`corr\_matrix`) for the DataFrame `importance\_df`. The `corr()` function computes the pairwise correlation coefficients between all pairs of features in the DataFrame, and `.abs()` returns the absolute values of these coefficients. This is done to consider both positive and negative correlations.

2. `mask = np.triu(np.ones\_like(corr\_matrix, dtype=bool))`

- This line creates a mask (`mask`) to hide the upper triangular part of the absolute correlation matrix `corr\_matrix`. The `np.triu()` function creates an upper triangular matrix with ones and zeros elsewhere. `np.ones\_like(corr\_matrix, dtype=bool)` creates a matrix of ones with the same shape as `corr\_matrix`. By applying `np.triu()` to this matrix, we get an upper triangular matrix of ones and zeros. This mask is used to hide the upper triangular part of the correlation matrix.

3. `tri\_importance\_df = corr\_matrix.mask(mask)`

- This line applies the mask `mask` to the absolute correlation matrix `corr\_matrix` using the `mask()` method. It sets the values in `corr\_matrix` to NaN (Not a Number) where the mask is True (i.e., in the upper triangular part of the matrix). This effectively removes the upper triangular part of the correlation matrix.

4. `to\_drop = [x for x in tri\_importance\_df.columns if any(tri\_importance\_df[x] > 0.75)]`

- This line creates a list `to\_drop` containing the names of columns in the DataFrame `tri\_importance\_df` that have at least one correlation coefficient greater than 0.75 (indicating high correlation with at least one other feature). It iterates over the columns of `tri\_importance\_df` and checks if any correlation coefficient in each column is greater than 0.75 using a list comprehension.

5. `importance\_df = importance\_df.drop(to\_drop, axis=1)`

- This line drops the columns specified in the list `to\_drop` from the DataFrame `importance\_df`. It removes the columns with high correlation coefficients, which helps in reducing multicollinearity and improves the performance of machine learning models.

6. `print(f"The reduced dataframe has {importance\_df.shape[1]} columns.")`

- This line prints out the number of columns remaining in the reduced DataFrame `importance\_df` after dropping the highly correlated columns. It uses an f-string to include the number of columns in the output message.



Figure 46: - Accessing column names

This line of code accesses the column names of the DataFrame `importance\_df`.

In pandas, the `columns` attribute of a DataFrame contains the names of its columns. By calling `importance\_df.columns`, you are retrieving an index object that contains the names of all the columns in the DataFrame `importance\_df`.

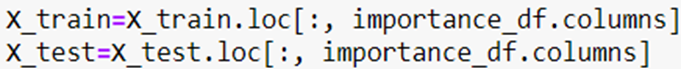


Figure 47 : Transforming the dataframe

1. `X\_train = X\_train.loc[:, importance\_df.columns]`

- This line selects columns from the DataFrame `X\_train` based on the column names present in the DataFrame `importance\_df`.

- `X\_train.loc[:, importance\_df.columns]` uses label-based indexing (`loc`) to select all rows (`:`) and only the columns that are present in `importance\_df.columns`.

2. `X\_test = X\_test.loc[:, importance\_df.columns]`

- This line does the same as the previous line, but for the DataFrame `X\_test`. It selects the same subset of columns as selected in `X\_train`, ensuring that both training and testing datasets have the same features for consistency in model training and evaluation.

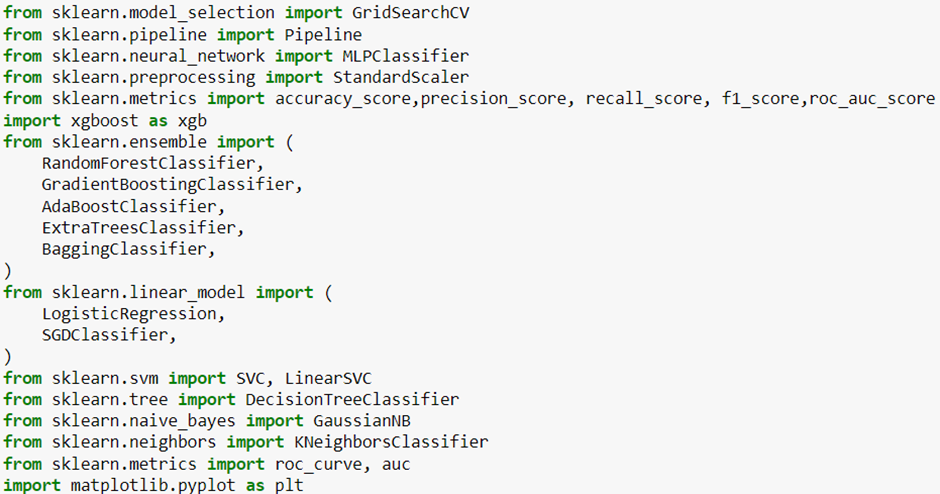


Figure 48: - Importing necessary modules

1. `from sklearn.model\_selection import GridSearchCV`

- This line imports the `GridSearchCV` class from the `sklearn.model\_selection` module. `GridSearchCV` is a technique used for tuning hyperparameters of a machine learning model by exhaustively searching through a specified grid of hyperparameters and cross-validating the model's performance.

2. `from sklearn.pipeline import Pipeline`

- This line imports the `Pipeline` class from the `sklearn.pipeline` module. `Pipeline` is a tool for chaining multiple machine learning algorithms into one, so that you can use them as a single estimator. It is useful for combining preprocessing steps (like scaling or feature selection) with a predictive model.

3. `from sklearn.preprocessing import StandardScaler`

- This line imports the `StandardScaler` class from the `sklearn.preprocessing` module. `StandardScaler` is used for standardizing features by removing the mean and scaling to unit variance. It is often used as a preprocessing step before feeding data into machine learning algorithms.

4. `from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score`

- This line imports several metric functions from the `sklearn.metrics` module. These functions are used for evaluating the performance of classification models.

- `accuracy\_score` calculates the accuracy of the model.

- `precision\_score` calculates the precision of the model.

- `recall\_score` calculates the recall of the model.

- `f1\_score` calculates the F1 score of the model.

- `roc\_auc\_score` calculates the area under the ROC curve of the model.

5. `import xgboost as xgb`

- This line imports the XGBoost library and aliases it as `xgb`. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.

6. `from sklearn.ensemble import (...)`

- This line imports several ensemble learning algorithms from the `sklearn.ensemble` module. Ensemble learning methods combine multiple individual models to improve predictive performance. The imported ensemble methods include:

- `RandomForestClassifier`: A random forest classifier.

- `GradientBoostingClassifier`: A gradient boosting classifier.

- `AdaBoostClassifier`: An AdaBoost classifier.

- `ExtraTreesClassifier`: An extra-trees classifier.

- `BaggingClassifier`: A bagging classifier.

7. `from sklearn.linear\_model import (...)`

- This line imports several linear model classifiers from the `sklearn.linear\_model` module. Linear model classifiers are algorithms that make predictions based on linear relationships between input features and target variables. The imported linear model classifiers include:

- `LogisticRegression`: Logistic regression classifier.

- `SGDClassifier`: Stochastic Gradient Descent classifier.

8. `from sklearn.svm import SVC, LinearSVC`

- This line imports the Support Vector Classifier (SVC) and the Linear Support Vector Classifier (LinearSVC) from the `sklearn.svm` module. SVM classifiers are supervised learning models used for classification tasks.

9. `from sklearn.tree import DecisionTreeClassifier`

- This line imports the `DecisionTreeClassifier` class from the `sklearn.tree` module. `DecisionTreeClassifier` is a classification model based on the decision tree algorithm.

10. `from sklearn.naive\_bayes import GaussianNB`

- This line imports the `GaussianNB` class from the `sklearn.naive\_bayes` module. `GaussianNB` is a classification algorithm based on Bayes' theorem with the assumption of independence between features.

11. `from sklearn.neighbors import KNeighborsClassifier`

- This line imports the `KNeighborsClassifier` class from the `sklearn.neighbors` module. `KNeighborsClassifier` is a classification algorithm based on the k-nearest neighbors algorithm, which assigns labels to data points based on the majority class of their k-nearest neighbors.

12. `from sklearn.metrics import roc\_curve, auc`

- This line imports the `roc\_curve` and `auc` functions from the `sklearn.metrics` module. These functions are used to compute Receiver Operating Characteristic (ROC) curves and Area Under the Curve (AUC) scores, which are common metrics for evaluating the performance of binary classification models.



Figure 49: - Building and training models. Using GridSearchCV for best hyper-parameter values

1. `classifiers = [...]`

- This block initializes a list called `classifiers` containing tuples. Each tuple represents a machine learning classifier along with its hyperparameters for tuning.

2. `results\_df = pd.DataFrame(columns=["Classifier", "Test Score", "Precision", "Recall", "F1-score", "ROC Score"])`

- This line initializes an empty DataFrame named `results\_df` with columns for storing evaluation metrics of different classifiers. The columns include: Classifier name, Test Score, Precision, Recall, F1-score, and ROC score.

3. `for classifier, clf, param\_grid in classifiers:`

- This initiates a loop over each tuple in the `classifiers` list. Each tuple contains the name of the classifier (`classifier`), the classifier object (`clf`), and a dictionary of hyperparameters to be tuned (`param\_grid`).

4. `pipeline = Pipeline([...])`

- This line creates a machine learning pipeline using the `Pipeline` class from scikit-learn. The pipeline consists of two steps:

- StandardScaler: Standardizes the features by removing the mean and scaling to unit variance.

- Classifier: The machine learning classifier specified in the `clf` variable.

5. `grid\_search = GridSearchCV(pipeline, param\_grid, cv=5)`

- This line initializes a `GridSearchCV` object named `grid\_search` for hyperparameter tuning. It takes three main arguments:

- `pipeline`: The machine learning pipeline.

- `param\_grid`: A dictionary of hyperparameters to be tuned.

- `cv=5`: The number of cross-validation folds to be used for evaluation.

6. `grid\_search.fit(X\_train, y\_train)`

- This line fits the `GridSearchCV` object to the training data (`X\_train` and `y\_train`). It performs an exhaustive search over the hyperparameter grid defined in `param\_grid` and evaluates the performance of each combination of hyperparameters using cross-validation.

7. `best\_estimator = grid\_search.best\_estimator\_`

- This line retrieves the best estimator found by the grid search, which is the model with the optimal combination of hyperparameters that resulted in the highest cross-validation score.



Figure 50

1. `sorted\_results = results\_df.sort\_values("Test Score", ascending=False)`

- This line creates a new DataFrame `sorted\_results` by sorting the `results\_df` DataFrame based on the values in the column "Test Score" in descending order (`ascending=False`).

- Sorting in descending order ensures that the classifiers with the highest test scores appear at the top of the DataFrame.

2. `sorted\_results`

- This line simply displays the `sorted\_results` DataFrame, showing the classifiers along with their corresponding evaluation metrics sorted by test score in descending order.

**Pickling of Top Models**

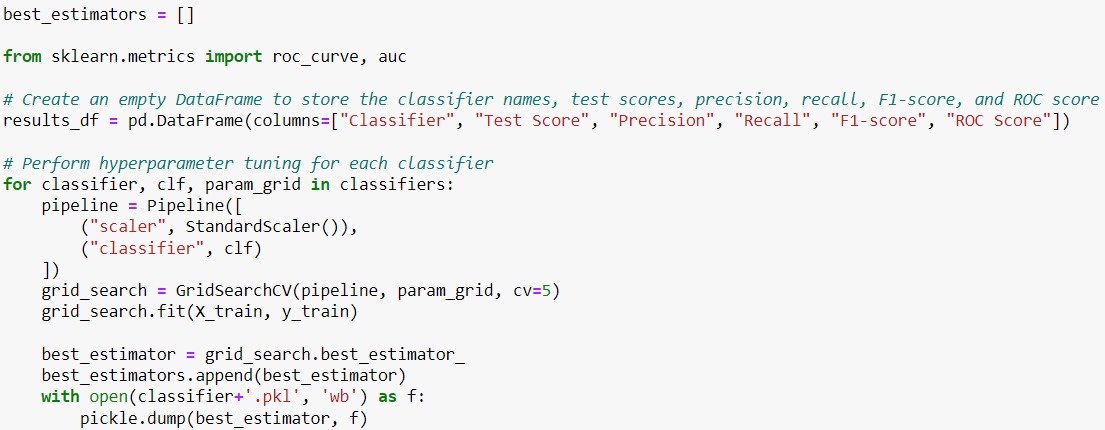


Figure 51: Pickling of the best models

best\_estimators.append(best\_estimator)

- Appends the best model found to the best\_estimators list.

1. with open(classifier+'.pkl', 'wb') as f:

- Opens a file in write-binary mode ('wb') with the name of the classifier (stored in classifier) and the .pkl extension. The file is opened as f.

2. pickle.dump(best\_estimator, f)

- Uses the pickle.dump function to serialize and save the best model (best\_estimator) to the file f.

Explanation:

- This code performs hyperparameter tuning for a list of classifiers, stores the best models found, and saves each best model to a .pkl file.

- It iterates over each classifier, creates a pipeline with data scaling and the classifier, and performs a grid search with cross-validation to find the best hyperparameters.

- The best model for each classifier is appended to a list and saved to a file for later use. This ensures that the best-performing models are easily accessible for future predictions or evaluations.

**Voting by Top Models in the Uniprot Human Database**

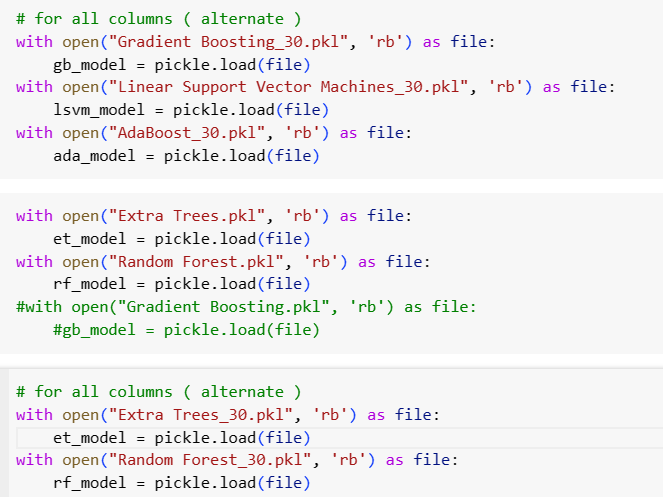


Figure 52: Loading the pickles

1. with open("Gradient Boosting\_30.pkl", 'rb') as file:

- Opens a file named "Gradient Boosting\_30.pkl" in read-binary mode ('rb') and assigns the file object to the variable file.

- This file is expected to contain a previously saved Gradient Boosting model.

2. gb\_model = pickle.load(file)

- Uses the pickle.load function to deserialize and load the model from the file object file.

- The loaded model is assigned to the variable gb\_model.

Explanation:

- This code snippet loads three pre-trained machine learning models from their respective .pkl files.

- For each model, it opens the corresponding file in read-binary mode, uses pickle.load to deserialize the model, and assigns the loaded model to a variable (gb\_model, lsvm\_model, and ada\_model).

- These models can then be used for making predictions or further evaluations in the code that follows.

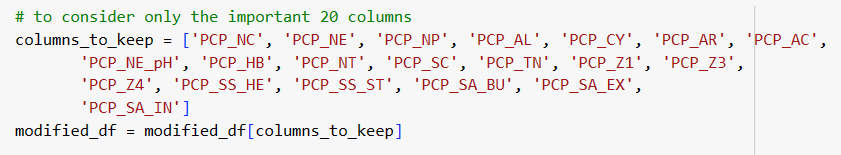


Figure 53: List of important columns

columns\_to\_keep = ['PCP\_AL', 'PCP\_CY', 'PCP\_AR', 'PCP\_NT', 'PCP\_SC', 'PCP\_TN', 'PCP\_Z3', 'PCP\_Z4', 'PCP\_SS\_HE', 'PCP\_SS\_ST', 'PCP\_SA\_EX', 'PCP\_SA\_IN']

- Creates a list named columns\_to\_keep containing the names of 12 columns to retain in the DataFrame.

- These columns are presumably the ones deemed most important for further analysis or model training.

modified\_df = modified\_df[columns\_to\_keep]

- Filters the DataFrame modified\_df to include only the columns specified in the columns\_to\_keep list.

- This is done by indexing modified\_df with columns\_to\_keep.

- The filtered DataFrame, which now only includes the specified 12 columns, is assigned back to modified\_df.

Explanation:

- This code snippet is used to reduce the DataFrame modified\_df to only include a subset of columns that are considered important.

- The first line defines a list of column names that should be kept.

- The second line filters the DataFrame to retain only these specified columns, effectively dropping any other columns not listed.

- The resulting DataFrame, modified\_df, now contains only the 12 important columns specified in columns\_to\_keep, which can simplify further analysis and ensure that only relevant data is used.

The dataframe being mentioned is the database of all the human proteins except the ones used for training and testing for identification of the novel human proteins through voting by top 8 models.

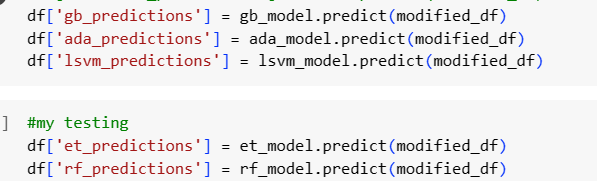


Figure 54: Using the pickled models for predictions

df['gb\_predictions'] = gb\_model.predict(modified\_df)

- This line adds a new column to the DataFrame df named 'gb\_predictions'.

- It uses the Gradient Boosting model (gb\_model) to make predictions on the modified\_df DataFrame (which contains only the important 12 columns).

- The predict method of gb\_model generates predictions based on the input data in modified\_df.

- These predictions are stored in the new 'gb\_predictions' column in df.

Other lines also serve the same purpose, but for the other models.

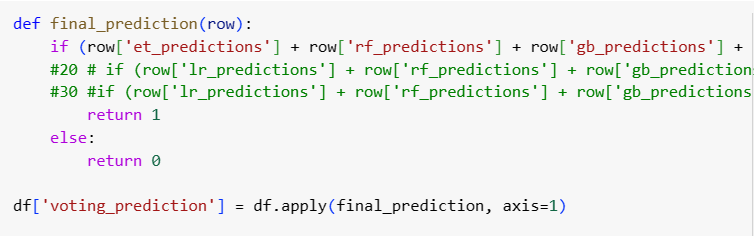


Figure 55: - Setting conditions for voting

1. def final\_prediction(row):

- Defines a function named final\_prediction that takes a single argument row.

- row represents a single row of the DataFrame when the function is applied to it.

2. if (row['et\_predictions'] + row['rf\_predictions'] + row['gb\_predictions'] + row['nb\_predictions'] + row['ada\_predictions'] + row['lsvm\_predictions'] + row['gnb\_predictions'] + row['svm\_predictions']) >= 6:

- Checks if the sum of predictions from various models (Extra Trees (et\_predictions), Random Forest (rf\_predictions), Gradient Boosting (gb\_predictions), Naive Bayes (nb\_predictions), AdaBoost (ada\_predictions), Linear Support Vector Machines (lsvm\_predictions), Gaussian Naive Bayes (gnb\_predictions), and Support Vector Machine (svm\_predictions)) for a given row is greater than or equal to 6.

- Each prediction is presumably a binary value (0 or 1), where 1 indicates a positive prediction.

6. df['voting\_prediction'] = df.apply(final\_prediction, axis=1)

- Applies the final\_prediction function to each row of the DataFrame df.

- The apply method is used with axis=1 to ensure the function is applied row-wise.

- The results of the final\_prediction function for each row are stored in a new column named 'voting\_prediction' in the DataFrame df.

Explanation:

- The code defines a function to compute a final prediction based on a majority vote from multiple model predictions.

- The function sums the predictions of 8 different models for each row.

- If the sum of the predictions is 6 or more, it returns 1 (indicating a positive final prediction).

- If the sum is less than 6, it returns 0 (indicating a negative final prediction).

- This function is then applied to each row of the DataFrame, and the results are stored in the 'voting\_prediction' column, providing an ensemble decision based on the majority vote of the models.



Figure 56

df['voting\_prediction'].sum()

- This line of code calculates the sum of the values in the 'voting\_prediction' column of the DataFrame df.

- The sum() method is called on the 'voting\_prediction' column, which adds up all the values in that column.

Explanation:

In summary, df['voting\_prediction'].sum() returns the number of positive predictions made by the ensemble of models across all rows in the DataFrame.