**API Documentation**

**Descriptive Analysis**

The Property Data Analysis API explained the project program's various functions, including the measure of variability and tendency and frequency analysis on the key continuous and categorical variables. It provides endpoints to perform various tasks related to analyzing property data.

**Function 1**: **Import the library ‘pandas’, ‘matplotlib’, ‘numpy’**

‘panda’ – import panda as pd

Description: Pandas is a Python library used for data manipulation and analysis. It would be a helpful library for us to clean, transform and analyze the structured data (input CSV file).

‘matplotlib’ – import matplotlib as plt

Description: matplotlib is a library that is used for plotting diagrams. It would be helpful for us to visualize our findings through line charts, box plots, histograms, bar charts, etc.

‘numpy’ – import numpy as np

Description: Numpy is a package for scientific computing in Python. It would be helpful for use to handle the arrays and matrices of the dataset. It provides support for arrays, mathematical functions, and linear algebra operations.

**Function 2: pd.read\_csv (Filename)**

Description:

The function will read over the input file (property.csv). The roperty.csv has columns of legal type, current value, current value improvement, previous value, previous value improvement, and tax levy. The function skips the header row and returns the first few rows of the property dataset.

Parameters:

Filename(str): The name of the input file (property.csv).

Filepath(str): ‘content/drive/MyDrive/INF1340/Sources/property.csv

Return:

Dataframe: property.csv

Raises:

FileNotFoundError: when the specified file is not found, the FileNotFoundError appears.

**Function 3: dropna()**

Description:

dropna() function removes rows containing missing values (NaN or None) in the property dataset.

Parameters:

None

Return:

Dataframe: property (after removing rows with missing values)

**Function 4: Statistics.mean()**

Description:

Statistics.mean() computes the mean of the CURRENT\_LAND\_VALUE/TAX\_LEVY columns in the dataset property.

Parameters:

data [‘column name’]: The data with the CURRENT\_LAND\_VALUE /TAX\_LEVY column ready for analysis.

Return:

Mean\_CurrentValue(float): The mean value of the CURRENT\_LAND\_VALUE /TAX\_LEVY column.

**Function 5: Statistics.mode()**

Description:

Statistics.mode() computes the mean of the CURRENT\_LAND\_VALUE/TAX\_LEVY columns in the dataset property.

Parameters:

data [‘column name’]: The data that has the CURRENT\_LAND\_VALUE/TAX\_LEVY column ready for analyze.

Return:

Mode\_CurrentValue(float): The mode value of the CURRENT\_LAND\_VALUE/TAX\_LEVY column.

**Function 6: Statistics.median()**

Description:

Statistics.median () function computes the median of the CURRENT\_LAND\_VALUE columns in the dataset property.

Parameters:

data [‘column name’]: The data with the CURRENT\_LAND\_VALUE/TAX\_LEVY column is ready for analysis.

Return:

Median\_CurrentValue(float): The median value of the CURRENT\_LAND\_VALUE/TAX\_LEVY column.

**Function 7: plt.hist()**

Description:

plt.hist() function generates the histogram for the CURRENT\_LAND\_VALUE/TAX\_LEVY in the dataset property.

Parameters:

data [‘column name’]: The data with the CURRENT\_LAND\_VALUE/TAX\_LEVY column is ready for the plot.

custom\_bins(list): a list of custom bin edges specified for the histogram. Because of the spread nature of the Current land value and tax levy, we generated the logarithmic histogram to visualize the distribution of the CURRENT\_LAND\_VALUE/TAX\_LEVY.

Return:

Histogram Visualizatioin: the histogram shows the frequency distribution of CURRENT\_LAND\_VALUE/TAX\_LEVY with logarithmic scaling on the x-axis.

**Function 8: Statistics. max()**

Description:

Statistics. max() computes the maximum value of the CURRENT\_LAND\_VALUE/TAX\_LEVY columns in the dataset property.

Parameters:

data [‘column name’]: The data with the CURRENT\_LAND\_VALUE/TAX\_LEVY column is ready for analysis.

Return:

Max\_CurrentValue(float): The maximum value of the CURRENT\_LAND\_VALUE/TAX\_LEVY column.

**Function 9: Statistics. min()**

Description:

Statistics. min() computes the minimum value of the CURRENT\_LAND\_VALUE/TAX\_LEVY columns in the dataset property.

Parameters:

data [‘column name’]: The data with the CURRENT\_LAND\_VALUE/TAX\_LEVY column is ready for analysis.

Return:

Min\_CurrentValue(float): The minimum value of the CURRENT\_LAND\_VALUE/TAX\_LEVY column.

The difference between Max\_CurrentValue(float) and Min\_CurrentValue(float) is the range of the CURRENT\_LAND\_VALUE/TAX\_LEVY.

**Function 10: Statistics. var()**

Description:

Statistics. var () computes the variance of the CURRENT\_LAND\_VALUE/TAX\_LEVY columns in the dataset property.

Parameters:

data [‘column name’]: The data with the CURRENT\_LAND\_VALUE/TAX\_LEVY column is ready for analysis.

Return:

Var\_CurrentValue(float): The variance of the CURRENT\_LAND\_VALUE/TAX\_LEVY column.

**Function 11: Statistics. std()**

Description:

Statistics. std () computes the standard deviation of the CURRENT\_LAND\_VALUE/TAX\_LEVY columns in the dataset property.

Parameters:

data [‘column name’]: The data with the CURRENT\_LAND\_VALUE/TAX\_LEVY column is ready for analysis.

Return:

Var\_CurrentValue(float): The standard deviation of the CURRENT\_LAND\_VALUE/TAX\_LEVY column.

**Function 12 : data.plot(kind = ‘box’)**

decription:

data.plt(kind = ‘box’) creates the box plot of the CURRENT\_LAND\_VALUE/TAX\_LEVY columns in the dataset property.

Parameters:

data [‘column name’]: The data with the CURRENT\_LAND\_VALUE/TAX\_LEVY column is ready for analysis.

Return:

Boxplot Visualization: the boxplot shows the frequency distribution of CURRENT\_LAND\_VALUE/TAX\_LEVY.

**Function 13: data.value\_counts()**

Description:

data.value\_counts function counts the occurrence of each legal type

Parameters:

data [‘column name’]: The LEGAL\_TYPE/ZONE\_CLASSIFICATION column data is ready for analysis.

Return:

The dictionary shows the legal type with the occurrence of each legal type/Zone classification.

**Function 14: data.value\_counts (). plot (kind = ‘pie’)**

Description:

data.value\_counts ().plot(kind = ‘pie’) creates the legal type/Zone classification pie chart.

Parameters:

data [‘column name’]: The data with the LEGAL\_TYPE/ZONE\_CLASSIFICATION column is ready for analysis.

Return:

Pie chart visualization: the pie chart shows the distribution of each legal type/Zone classification.

**Function 15: data.groupby().mean().reset\_index()**

Description:

data.groupby().mean().reset\_index() Calculates the average tax rate by each legal type /average growth rate by each zone classification

Parameters:

data [‘column name’]: The data with the LEGAL\_TYPE/ZONE\_CLASSIFICATION column is ready for analysis.

data [‘column name’]: The data with the CURRENT\_LAND\_VALUE/TAX\_LEVY column is ready for analysis.

Return:

The dictionary shows the average tax rate by each legal type and average growth rate by each zone classification.

**Function 16: plt.bar()**

Description:

plt.bar() function creates the bar chart of the average tax rate by legal type and average growth rate by zone classification.

Parameters:

data [‘column name’]: The LEGAL\_TYPE/ZONE\_CLASSIFICATION column data is ready for analysis.

data [‘column name’]: The data with the CURRENT\_LAND\_VALUE/TAX\_LEVY column is ready for analysis.

Return:

Bar chart visualization: the bar chart shows the average tax rate across the legal type and average growth rate across the zone classification.

**Function 17 Statistics.skew()**

Description:

Statistics. skew () computes the skewness of the CURRENT\_LAND\_VALUE/TAX\_LEVY columns in the dataset property.

Parameters:

data [‘column name’]: The data with the CURRENT\_LAND\_VALUE/TAX\_LEVY column is ready for analysis.

Return:

skewness\_CurrentValue(float): The skewness of the CURRENT\_LAND\_VALUE/TAX\_LEVY column.

**Function 18 Statistics.kurtosis()**

Description:

Statistics. kurtosis () computes the kurtosis of the CURRENT\_LAND\_VALUE/TAX\_LEVY columns in the dataset property.

Parameters:

data [‘column name’]: The data with the CURRENT\_LAND\_VALUE/TAX\_LEVY column is ready for analysis.

Return:

kurtosis \_CurrentValue(float): The kurtosis of the CURRENT\_LAND\_VALUE/TAX\_LEVY column.

**Diagnostic Analysis**

**Function 1**: **Import the library ‘pandas’, ‘matplotlib’, ‘numpy’**

‘panda’ – import panda as pd

Description: Pandas is a Python library used for data manipulation and analysis. It would be a helpful library for us to clean, transform and analyze the structured data (input CSV file).

‘matplotlib’ – import matplotlib as plt

Description: matplotlib is a library that is used for plotting diagrams. It would be helpful for us to visualize our findings through line charts, box plots, histograms, bar charts, etc.

‘numpy’ – import numpy as np

Description: Numpy is a package for scientific computing in Python. It would be helpful for use to handle the arrays and matrices of the dataset. It provides support for arrays, mathematical functions, and linear algebra operations.

**Function 2: pd.read\_csv (Filename)**

Description: The function will read over the input file (property.csv). The roperty.csv has columns of legal type, current value, current value improvement, previous value, previous value improvement, and tax levy. The function skips the header row and returns the first few rows of the property dataset.

Parameters:

Filename(str): The name of the input file (property.csv).

Filepath(str): ‘content/drive/MyDrive/INF1340/Sources/property.csv

Return:

Dataframe: property.csv

**Function 3: dropna()**

Description: dropna() function removes rows containing missing values (NaN or None) in the property dataset.

Parameters:

None

Return:

Dataframe: property (after removing rows with missing values)

**Function 4: groupby().mean()**

Description:

This function groups the data by a specified column (e.g., LEGAL\_TYPE or ZONE\_CLASSIFICATION) and calculates the mean of numeric columns for each group.

Parameters:

column\_name: The column name for grouping, such as 'LEGAL\_TYPE' or 'ZONE\_CLASSIFICATION'.

Return:

A DataFrame with the mean values of numeric columns for each group specified by the column\_name.

**Function 5: plot(kind='bar')**

Description:

This function creates a bar plot for the grouped data. It visualizes the average values of either 'CURRENT\_LAND\_VALUE' or 'CURRENT\_IMPROVEMENT\_VALUE' for each legal type.

Parameters:

kind='bar': Specifies the type of plot as a bar chart.

color (optional, used in the second plot): Sets the color of the bars, e.g., 'orange' for the 'CURRENT\_IMPROVEMENT\_VALUE' plot.

Return:

Bar plot visualizations for 'CURRENT\_LAND\_VALUE' and 'CURRENT\_IMPROVEMENT\_VALUE' by legal type.

**Function 6: boxplot(data=property, x='COLUMN\_NAME')**

Description:

Creates a box plot for the specified column in the property DataFrame. This plot is used to visualize the distribution and identify outliers in 'CURRENT\_LAND\_VALUE' and 'CURRENT\_IMPROVEMENT\_VALUE'.

Parameters:

data=property: The DataFrame containing the data to be visualized.

x='COLUMN\_NAME': The specific column for which the box plot is generated, either 'CURRENT\_LAND\_VALUE' or 'CURRENT\_IMPROVEMENT\_VALUE'.

Return:

Box plot visualizations for 'CURRENT\_LAND\_VALUE' and 'CURRENT\_IMPROVEMENT\_VALUE'.

**Function 7: sort\_values(by='COLUMN\_NAME', ascending=False)**

Description:

Sorts the DataFrame based on the values in the specified column. Used here to arrange the grouped data in descending order of 'CURRENT\_LAND\_VALUE'.

Parameters:

by='COLUMN\_NAME': The column name to sort by, such as 'CURRENT\_LAND\_VALUE'.

ascending=False: Sorting order, set to False for descending order.

Return:

A DataFrame sorted by the specified column in descending order.

**Function 8: head(20)**

Description:

Selects the top 20 rows from the sorted DataFrame. It's used to focus the analysis on the top regions based on average property values.

Parameters:

20: Number of top rows to select.

Return:

A DataFrame containing the top 20 rows from the sorted data.

**Function 9: stats.f\_oneway(\*grouped\_data\_by\_type)**

Description:

Performs a one-way ANOVA test using scipy.stats on the grouped data. This test is used to determine if there are any statistically significant differences between the means of different zone classifications.

Parameters:

\*grouped\_data\_by\_type: Variable arguments representing the grouped data by 'ZONE\_CLASSIFICATION', each as a list of values.

Return:

ANOVA test result with F-statistic and p-value, indicating whether the means of different groups are significantly different.

**Function 10: stats.ttest\_ind(strata\_data.dropna(), land\_data.dropna(), equal\_var=False)**

Description:

Performs a two-sample t-test to compare the means of 'CURRENT\_LAND\_VALUE' between 'STRATA' and 'LAND' legal types. It's used to assess if there's a significant difference in land values between these two types.

Parameters:

strata\_data.dropna(): Data for 'STRATA' legal type with NaN values removed.

land\_data.dropna(): Data for 'LAND' legal type with NaN values removed.

equal\_var=False: Indicates that the variances of the two groups are assumed to be different (Welch's t-test).

Return:

T-test result, including t-statistic and p-value, indicating the statistical significance of the difference in means between the two legal types.

**Predictive Analysis**

The Predictive Analysis part in the API documentation uses various functions from different packages to perform three analyses and make predictions on different variables. By analyzing historical data, identifying relevant features and training models, it provides valuable insights for potential buyers or sellers of properties regarding investments and development.

**Task 1: Linear regression (Deitel and Deitel, 2019)**

This task refers to chapter 15 from the following book for the scatterplot and linear regression prediction.

Reference: Deitel, P., Deitel, H.M. (2019). Intro to Python for Computer Science and Data Science: Learning to Program with AI, Big Data and the Cloud. Pearson Education Limited

**Function 1**: **Import the library ‘pandas’, ‘matplotlib’, ‘seaborn’**

‘panda’ – import panda as pd

Description: Pandas is a Python library used for data manipulation and analysis. It would be a helpful library for us to clean, transform and analyze the structured data (input CSV file).

‘matplotlib’ – import matplotlib as plt

Description: matplotlib is a library that is used for plotting diagrams. It would be helpful for us to visualize our findings through line charts, box plots, histograms, bar charts, etc.

‘seaborn’ – import seaborn as sns

Description: Seaborn is a data visualization library for Python built on top of Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

**Function 2: pd.read\_csv (Filename)**

Description:

The function will read over the input file (property.csv). The roperty.csv has columns of legal type, current value, current value improvement, previous value, previous value improvement, and tax levy. The function skips the header row and returns the first few rows of the property dataset.

Parameters:

Filename(str): The name of the input file (property.csv).

Filepath(str): ‘content/drive/MyDrive/INF1340/property.csv

Return:

Dataframe: property.csv

**Function 3: drop()**

Description:

drop() function removes certain columns in the property dataset.

Parameters: columns=[‘TAX\_ASSESSMENT\_YEAR’]

Return: filterd\_VanPro data frame, which contains no tax assessment year column

**Function 4: filtered\_VanPro['YEAR\_BUILT'].astype(), filtered\_VanPro['BIG\_IMPROVEMENT\_YEAR'].astype()**

Description:

astype() transfer the columns into a specified type.

Parameters:

int: It transfers values in the ‘YEAR\_BUILT’ and ‘BIG\_IMPROVEMENT\_YEAR’ into integer types.

Return:

filtered\_VanPro: With two columns type changed into integers.

**Function 5: dropna()**

Description:

It drops NaN values for the related data frame.

Parameters: none

Return: filtered\_VanPro, which contains no missing values.

**Function 6: scatterplot()**

Description:

It is a function in seaborn used to visualize the relationship between two continuous variables and identify patterns. Here we use a for loop to go through every element in the feature list and plot the relationship between each feature and the current improvement value.

Parameters:

data=filtered\_VanPro\_cleaned, x=feature,

y='CURRENT\_IMPROVEMENT\_VALUE', hue='CURRENT\_IMPROVEMENT\_VALUE',

palette='cool', legend=False

Return: a scatterplot

**Function 7: sm.add\_constant()**

Description:

Being part of the statsmodels library, this function is used to add a constant term (intercept) into a set of independent variables.

Parameters: X\_feature

Return: X\_feature\_with\_const, a new data frame with constants

**Function 8: sm.OLS.fit.summary()**

Description:

It is from the ‘statsmodels’ library in Python stands for Ordinary Least Squares, and it is used to perform linear regression analysis. Then the ‘fit’ method is called to estimate the coefficients of the model using the OLS method. Finally the ‘summary’ method is used to obtain a detailed summary of the regression results.

Parameters: y\_target, X\_feature\_with\_const

Return: A detailed summary of the regression results includes coefficients, standard errors, t-statistics, p-values, and other relevant statistics.

**Function 9: train\_test\_splilt()**

Description:

It is a function in machine learning libraries to split a dataset into two subsets: one for training a model and the other for testing the model. The random\_state in the parameter ensures that the randomness is reproducible.

Parameters: X\_feature, y\_target, random\_state=11

Return: Get four sets of data: X\_train, X\_test, y\_train, y\_test

**Function 10:LinearRegression(), LinearRegression.fit()**

Description:

The first part, LinearRegression() constructs a linear regression model, it creates an empty instance of the linear regression model. Then .fit() trains the model using the training data. It will learn the coefficients that minimize the sum of squared differences between the predicted and actual target variable values.

Parameters: X=X\_train, y=y\_train

Return: A model ready to make predictions on new, unseen data

**Function 11:predict()**

Description:

It takes a feature matrix as input and returns the corresponding predicted target variable values.

Parameters: X\_test

Return: a variable contains the predicted values for the target variable based on the model’s understanding of the patterns in the testing set.

**Function 12:sm.r2\_score()**

Description: a metric that measures how close the data are to the fitted regression line.

Parameters: expected, predicted

Return: The R-squared score.

**Task 2: Logistic regression (Grover, 2023)**

This task refers to chapter 19 from the following book for the logistic regression prediction part.

Reference: Grover, D. (2023). Python for Data & Analytics: A Business-Oriented Approach. United States: Prospect Press.

**Function 1**: **Import the library ‘pandas’, ‘matplotlib’, ‘seaborn’**

The same as in Task 1 Function 1

**Function 2: pd.read\_csv (Filename)**

The same as in Task 1 Function 2

**Function 3: drop()**

The same as in Task 1 Function 3

**Function 4: filtered\_VanPro['YEAR\_BUILT'].astype(), filtered\_VanPro['BIG\_IMPROVEMENT\_YEAR'].astype()**

The same as in Task 1 Function 4

**Function 5: dropna()**

The same as in Task 1 Function 5

**Function 6: apply()**

Description:

It creates a new binary column in the DataFrame and the value is 1 for element ‘x’ greater than or equal to 0, otherwise, the value 0 is assigned.

Parameters: lambda x: 1 if x >= 0 else 0

Return: A binary column ‘IMPROVEMENT\_VALUE\_CHANGE\_BINARY’

**Function 7: str.split(expand=True)**

Description:

It splits the values in a column into multiple columns. Each new column will correspond to a part of the original string.

Parameters:

filtered\_VanPro\_cleaned['PROPERTY\_POSTAL\_CODE']

Return:

Two new columns each contain the first and last 3 codes correspondingly

**Function 8: rename()**

Description:

It renames the column by assigning new names to the columns previously created through the split operation.

Parameters:

columns={0: 'First\_Three\_Code', 1: 'Last\_Three\_Code'}

Return:

Two new columns created above (name 0 and 1) renamed (‘First\_Three\_Code’ and ‘Last\_Three\_Code’)

**Function 9: pd.concat()**

Description: It concatenates (joins) the DataFrame with columns after comma along the columns axis (axis=1).

Parameters: [filtered\_VanPro\_cleaned, split\_columns], axis=1

Return: The original DataFrame filtered\_VanPro\_cleaned with split\_columns created above

**Function 10: unique()**

Description: It is used to obtain the unique values in the column from the DataFrame.

Parameters: filtered\_VanPro\_cleaned["First\_Three\_Code"]

Return: An array containing all the unique values present in the ‘First\_Three\_Code’ column of the DataFrame ‘filtered\_VanPro\_cleaned’

**Function 11: str.replace()**

Description: It creates a new binary column, and the values in this new column are derived from the column in parameter based on conditions defined by regular expressions. If the original value in ‘'First\_Three\_Code'’ starts with ‘V5’, it is replaced with ‘0’. If it starts with ‘V6’, it is replaced with ‘1’.

Parameters: filtered\_VanPro\_cleaned['First\_Three\_Code'], in the first bracket we have: r'^V5.\*$', '0'; in the second bracket we have r'^V6.\*$', '1'

Return: A binary-encoded format with only 0 and 1

**Function 12: logreg1 = Logit().fit().summary()**

Description: It uses the ‘Logit’ class to perform logistic regression with the independent variables represented by ‘X’ and the dependent variable by ‘y’.

Parameters: y,X, where y is filtered\_VanPro\_cleaned['IMPROVEMENT\_VALUE\_CHANGE\_BINARY'],

and X is filtered\_VanPro\_cleaned[['CURRENT\_LAND\_VALUE','PREVIOUS\_LAND\_VALUE','TAX\_LEVY','HOUSE\_AGE','First\_Three\_Code\_Binary']]

Return: Results of the logistic regression analysis, summary() shows information including coefficients, standard errors, p-values, and other relevant statistics

**Function 13: train\_test\_split()**

The same as in Task 1 Function 9, with random\_state=22

**Function 14: LogisticRegression().fit()**

The same as in Task 1 Function 10, only to change from linear regression to logistic regression

**Function 15: logisticRegression.predict\_proba()**

Description: It is used to obtain predicted probabilities for each class using a trained logistic regression model.

Parameters: X\_test

Return: array with two columns for binary classification tasks (0 and 1), and each row represent the predicted probabilities of the corresponding sample belonging to each class

**Function 16: pd.Series()**

Description: It creates a pandas series from the data in ()

Parameters: y\_test\_proba, which extracts the predicted probabilities for class 1 from the array ‘logicPredict’

Return: a pandas Series object with the predicted probabilities

**Function 17: confusion\_matrix()**

Description: It calculates the confusion matrix for a binary classification problem

Parameters: y\_test, y\_test\_pred

Return: create a confusion matrix with a structure like

array([[ TN, FP],

[ FN, TP]])

Where True Positive (TP) means the model predicted the positive class correctly; True Negative (TN) means the model predicted the negative class correctly, False Positive (FP)means the model predicted the positive class incorrectly, and False Negative (FN)means the model predicted the negative class incorrectly.

**Function 18: cmratio() and displayCM()**

Description: The first function is defined to calculate the ratio given the counts of correct and incorrect predictions, and the second function calculates various metrics such as sensitivity, specificity, precision, negative predictive value (npv) using the ‘cmratio’ function. The formatting options (fmt1) control the width and precision of the displayed values. Finally, it prints the accuracy of the model.

Parameters: confusion\_matrix(y\_test, y\_test\_pred)

Return: This would display the true positive (TP), false negative (FN), false positive (FP), true negative (TN), sensitivity, specificity, precision, and negative predictive value, along with the overall accuracy of the model.

**Task 3: Naive Bayes classifier**

**Function 1**: **Import the library ‘pandas’, ‘matplotlib’, ‘seaborn’**

The same as in Task 1 Function 1

**Function 2: pd.read\_csv (Filename)**

The same as in Task 1 Function 2

**Function 3: drop()**

The same as in Task 1 Function 3

**Function 4: filtered\_VanPro['YEAR\_BUILT'].astype(), filtered\_VanPro['BIG\_IMPROVEMENT\_YEAR'].astype()**

The same as in Task 1 Function 4

**Function 5: dropna()**

The same as in Task 1 Function 5

**Function 6: str.split(expand=True)**

The same as in Task 2 Function 7

**Function 7: rename()**

The same as in Task 2 Function 8

**Function 8: pd.concat()**

The same as in Task 2 Function 9

**Function 9: pd.get\_dummies()**

Description: It performs one-hot encoding on the categorical features of the DataFrame by converting categorical variables into dummy variables

Parameters: VanPro\_legal.drop('LEGAL\_TYPE', axis=1)

Return: a DataFrame where categorical features have been replaced with dummy variables, each unique value in the original column will become a new binary column

**Function 10: train\_test\_split()**

The same as in Task 1 Function 9, with random\_state=33

**Function 11: MultinomialNB()**

Description: It creates an instance of the Multinomial Naive Bayes classifier in Python

Parameters: none

Return: create a classifier for training on the data and making predictions

**Function 12: fit()**

Description: It trains (fits) a Multinomial Naive Bayes classifier on a training dataset represented by features and corresponding labels

Parameters: X\_train, y\_train

Return: create a classifier for making predictions on new data

**Function 13: accuracy\_score()**

Description: It calculates the accuracy of a classification model by comparing the predicted labels to the true labels

Parameters: y\_test, nb\_predictions

Return: a percentage indicating the proportion of correctly predicted instances in the testing set

**Function 14: classification\_report()**

Description: It generates a text report that includes several classification metrics for evaluating the performance of a classification model

Parameters: y\_test, nb\_predictions

Return: a report with the following metrics:

* Precision: The ratio of true positive predictions to the total predicted positives. Precision is a measure of the accuracy of the positive predictions.
* Recall (Sensitivity): The ratio of true positive predictions to the total actual positives. Recall is a measure of how well the model captures all instances of the positive class.
* F1-Score: The harmonic mean of precision and recall. It provides a balance between precision and recall.
* Support: The number of actual occurrences of each class in the specified order.
* Accuracy: The ratio of correctly predicted instances to the total number of instances.