1. Load the data from ‘ex1data1.txt’. The first column is the population of a city and the second column is the profit of a food truck in that city. A negative value for profit indicates a loss

Perform the following:

1. Visualize the data using the appropriate plot.
2. Print the description of the data.
3. Check if attributes have a linear relationship, and apply the Linear Regression model. (Train/Test split = 80/20)
4. Find and print the regression parameters.
5. Find the goodness of the model using R-squared (Co-efficient of Determination).
6. Compute MSE for the instances in the test set.
7. Load the data from ‘ex1data2.txt’ contains a training set of housing prices in Portland, Oregon. The first column is the size of the house (in square feet), the second column is the number of bedrooms, and the third column is the price of the house.

Perform the following:

1. Print the description of the data.
2. Apply the Linear Regression model. (Train/Test split = 80/20)
3. Find and print the regression parameters.
4. Find the goodness of the model using R2.
5. Compute MSE for the instances in the test set.
6. Load the data from ‘canada\_per\_capita\_income.csv’. Use this to build a regression model and predict the per capita income for Canadian citizens in the year 2020. Predict Canada’s per capita income in the year 2020. (Expected Output - 41288.69409442)
7. Suppose that you are the administrator of a university department and you want to determine each applicant’s chance of admission based on their results on two exams. You have historical data (“ex2data1.txt”) from previous applicants that you can use as a training set for logistic regression. For each training example, you have the applicant’s scores on two exams and the admissions decision. Your task is to build a classification model that estimates an applicant’s probability of admission based the scores from those two exams.
8. Load the data from ‘HR\_comma\_sep.csv’. Use this to build a logistic regression model and compute the accuracy of model.
9. Load the data from ‘Naive-Bayes-Classification-Data.csv’. Use this to build a Naïve Bayes classifier and compute the accuracy of model.

<https://heartbeat.comet.ml/naive-bayes-classifier-in-python-using-scikit-learn-13c4deb83bcf>

1. Load Iris data-set. Use this to build a Naïve Bayes classifier and find the accuracy of the model.
2. Load Iris data-set. Use this to build a SVM classifier and find the accuracy of the model.

<https://www.kaggle.com/code/arshid/support-vector-machine-on-iris-flower-dataset/notebook>

<https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python>

1. Load Breast Cancer Wisconsin (Diagnostic) data-set. Use this to build a SVM classifier and find the accuracy of the model.
2. Load the data from Iris data-set. Use this to build a Decision Tree model and compute the accuracy of model.
3. Load the data from Titanic data-set. Use this to build a Decision Tree model and compute the accuracy of model.
4. Load Titanic data-set. Use this to build kNN classifier and compute the accuracy of model.
5. Load Iris data-set. Use this to build kNN classifier and compute the accuracy of model.
6. Load Iris data-set. Perform k-means Clustering over it.

Linear Regression- Simple, Multiple, Polynomial, Logistic Regression, Regularization, Naïve Bayes Classifier, Decision Tree Classifier, SVM, kNN, K-means Clustering, Perceptron, Hierarchical Clustering, Cross Validation

Scikit-learn provides three types of Naive Bayes algorithms:

1. Gaussian Naive Bayes (GaussianNB): This algorithm assumes that the features are normally distributed and calculates the mean and standard deviation of each feature for each class. It then uses Bayes' theorem to calculate the posterior probability of each class given the input features.
2. Multinomial Naive Bayes (MultinomialNB): This algorithm is used for discrete data such as text. It assumes that the input features are counts and uses Bayes' theorem to calculate the probability of each class given the input features.
3. Bernoulli Naive Bayes (BernoulliNB): This algorithm is also used for discrete data such as text. It assumes that the input features are binary (i.e., 0 or 1) and uses Bayes' theorem to calculate the probability of each class given the input features.

All three algorithms are implemented in scikit-learn's **naive\_bayes** module. You can create an instance of each algorithm by calling the corresponding class (**GaussianNB**, **MultinomialNB**, or **BernoulliNB**) and then use the **fit()** method to train the model and the **predict()** method to make predictions.

SVC Parameters

The **SVC** (Support Vector Classification) class in scikit-learn provides a number of parameters that you can use to customize your SVM classifier. Here are some of the most commonly used parameters:

1. **C**: This is the regularization parameter that controls the trade-off between achieving a low training error and a low testing error. A smaller value of **C** will result in a wider margin, allowing more errors in the training set, but potentially improving the generalization performance. A larger value of **C** will result in a narrower margin, trying to correctly classify all training examples, but may lead to overfitting. The default value is **C=1.0**.
2. **kernel**: This parameter specifies the type of kernel function to be used for the SVM classifier. There are several types of kernel functions available in scikit-learn, such as 'linear', 'poly', 'rbf', 'sigmoid', etc. The choice of kernel function depends on the nature of the data and the problem you are trying to solve. The default kernel is 'rbf' (Radial Basis Function).
3. **degree**: This parameter is used only for the polynomial kernel function ('poly') and specifies the degree of the polynomial. The default value is **degree=3**.
4. **gamma**: This parameter is used for the 'rbf', 'poly', and 'sigmoid' kernel functions and controls the width of the Gaussian kernel. A smaller value of **gamma** will result in a wider kernel, allowing more points to be considered as similar, potentially leading to underfitting. A larger value of **gamma** will result in a narrower kernel, considering only nearby points, potentially leading to overfitting. The default value is **gamma='scale'**, which uses the inverse of the number of features (**1 / (n\_features \* X.var())**) as the value of **gamma**.
5. **coef0**: This parameter is used for the 'poly' and 'sigmoid' kernel functions and controls the constant in the decision function. The default value is **coef0=0.0**.
6. **shrinking**: This parameter controls whether to use the shrinking heuristic to speed up the optimization process. The default value is **shrinking=True**.
7. **probability**: This parameter controls whether to enable probability estimates. If **True**, the classifier will predict the probability of the sample being in each class. The default value is **probability=False**.
8. **tol**: This parameter controls the tolerance for stopping criterion. The default value is **tol=1e-3**.

These are just some of the most commonly used parameters in **SVC**. You can find more information on the **SVC** documentation page in scikit-learn's website.