Task 1

a. In this task, I worked with two CSV files, "train.csv" and "unique_m.csv," which were contained in a zip file. These files did not have any meaningful keys for merging or linking. I understood that "unique_m.csv" is for a different research purpose and focused on "train.csv." I read both files, but split train.csv into training and testing sets, and applied various regression models.

However, I consistently obtained a score of around 0.73 or 0.74 for Linear Regression, Ridge Regression with different parameters, and Lasso Regression.

Lasso Regression or Ridge Regression: These are linear regression variants with L1 (Lasso) and L2 (Ridge) regularization, respectively. They are useful when dealing with many features and we want to reduce overfitting or perform feature selection. Lasso can set some coefficients to zero, effectively selecting a subset of important features. Ridge shrinks the coefficients towards zero but does not set them exactly to zero.

The scores were as follows:

- Linear Regression Score: 0.7376
- Ridge (default parameters) Score: 0.74
- Ridge (alpha=10) Score: 0.73
- Ridge (alpha=0.1) Score: 0.74
- Ridge (alpha=0.01) Score: 0.74
- Ridge (alpha=0.001) Score: 0.74
- Lasso Score: 0.71, Number of features used with Lasso: 47

Since the scores were not satisfactory, I decided to use the RandomForestRegressor, even though it took more time to run. The Random Forest Regressor yielded a score of 0.93, indicating a promising performance.

Random Forest Regressor Score: 0.9305

Random Forest is an ensemble method that can handle both linear and nonlinear relationships. It's particularly useful for capturing complex interactions between features. It can also provide feature importance scores.

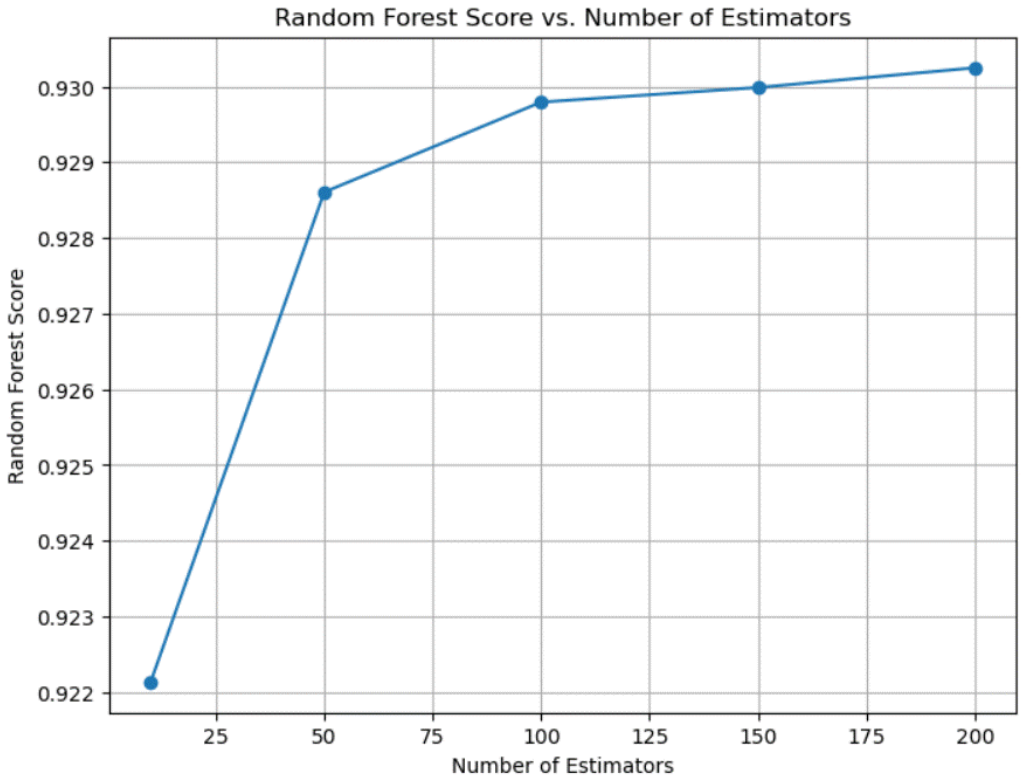
Some default parameters of the RandomForestRegressor were mentioned, including:

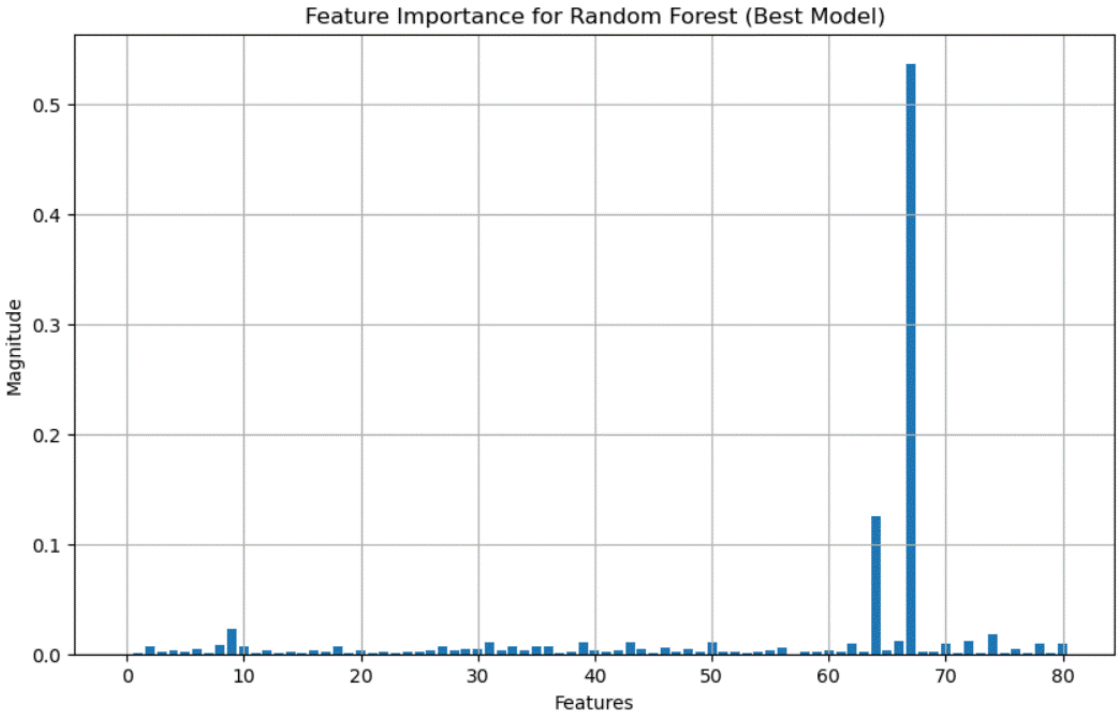
- n_estimators: The number of trees in the forest.
- max_depth: The maximum depth of the individual decision trees.
- min_samples_split: The minimum number of samples required to split an internal node.
- min_samples_leaf: The minimum number of samples required to be at a leaf node.
- max_features: The number of features to consider when looking for the best split.

b. To further optimize the Random Forest Regressor, I experimented with different values of the "n_estimators" parameter, specifically [10, 50, 100, 200, 300]. I created two plots to visualize the relationship between the number of estimators and the Random Forest Score, as well as the features and their magnitude. The "Gmeans_density" feature had the highest value among the columns.

From the plots, it was evident that the best Random Forest Score (0.9299) was achieved when "n_estimators" was set to 200.

Best Random Forest Score: 0.9302484131218491 (n_estimators = 200)





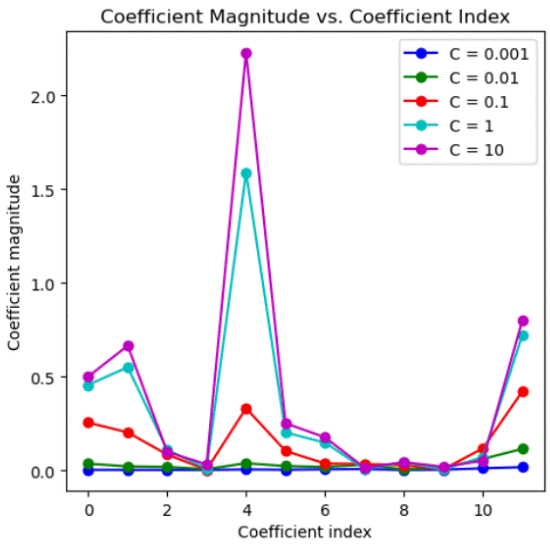
Task 2

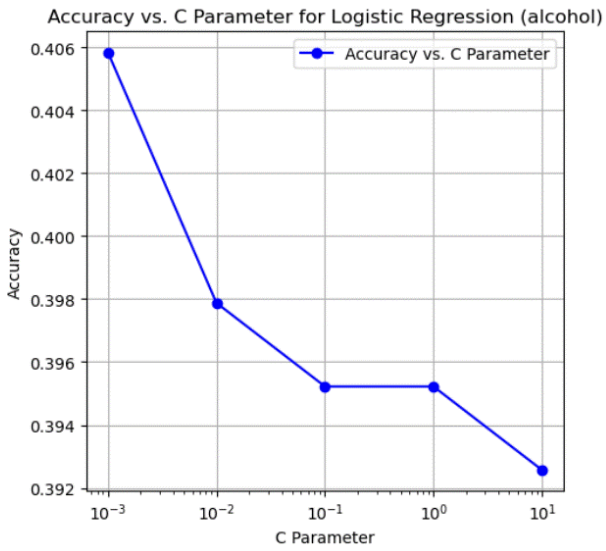
a. I started by reading a data file from the URL "https://archive.ics.uci.edu/ml/machine-learning-databases/00373/drug_consumption.data" and organized the columns correctly. I then mapped the seven classes to numerical values and selected two features, "alcohol" and "caffeine," which I considered as the target variable (y), while 12 columns became the features (X). I used LogisticRegression and LinearSVC for classification and calculated the accuracy for each:

- Accuracy for Logistic Regression (alcohol): 0.3952
- Accuracy for Logistic Regression (caffeine): 0.5703
- Accuracy for LinearSVC (alcohol): 0.4005
- Accuracy for LinearSVC (caffeine): 0.5650

Again, the scores were not satisfactory**!**

b. To optimize one of the default parameters, "C" values, in LogisticRegression, I conducted a manual optimization by trying at least five different values: [0.001, 0.01, 0.1, 1, 10]. I then plotted the coefficient magnitude vs the coefficient index and the accuracy vs the "C" parameter for Logistic Regression (alcohol).





C. In this step, I trained a multiclass classification model to predict 16 output features, excluding the two features from part a. I used a for loop along with LogisticRegression and calculated the accuracy for each of the 16 output features.

- Accuracy for amphetamines: 0.4960
- Accuracy for amyl nitrite: 0.6817
- Accuracy for benzodiazepine: 0.5385
- Accuracy for cannabis: 0.7321
- Accuracy for chocolate: 0.4164
- Accuracy for cocaine: 0.4138
- Accuracy for caffeine: 0.5703
- Accuracy for crack: 0.8859
- Accuracy for ecstasy: 0.5464
- Accuracy for heroin: 0.8594
- Accuracy for ketamine: 0.8170
- Accuracy for legal highs: 0.6207
- Accuracy for LSD: 0.5782
- Accuracy for methadone: 0.7374
- Accuracy for mushrooms: 0.5013
- Accuracy for nicotine: 0.3926

As you see, accuracy for crack, heroin and ketamine is almost fine.

Classification Report for Classes CL0 to CL6:

Class	Accuracy	Precision	Recall	F1-Score	Support
CL0	0.65	0.63	0.35	0.45	502
CL1	0.23	0.23	0.06	0.10	206
CL2	0.18	0.23	0.04	0.06	252
CL3	0.27	0.28	0.16	0.20	204
CL4	0.25	0.30	0.05	0.09	155
CL5	0.32	0.29	0.03	0.05	135
CL6	0.71	0.62	0.93	0.74	1,775

Overall Accuracy: 0.5249

the accuracy for predicting these features ranges from low to moderate. The classification performance varies widely across different classes within each feature, indicating that the model may perform well for some categories while struggling with others. Further model tuning or more data may be required to improve the accuracy for these features, particularly for the classes where it is currently low

d. For binary classification and three states (s1, s2, s3), I identified the states for different drug classes.

In [27]:
```
df_drug['s1_amphetamines'] = df_drug['amphetamines'].map({'CL0': 0, 'CL1': 0, 'CL2': 1, 'CL3': 1, 'CL4': 1, 'CL5': 1, 'CL6':
df_drug['s1_cannabis'] = df_drug['cannabis'].map({'CL0': 0, 'CL1': 0, 'CL2': 1, 'CL3': 1, 'CL4': 1, 'CL5': 1, 'CL6': 1})
df_drug['s1_ecstasy'] = df_drug['ecstasy'].map({'CL0': 0, 'CL1': 0, 'CL2': 1, 'CL3': 1, 'CL4': 1, 'CL5': 1, 'CL6': 1})

df_drug['s2_amphetamines'] = df_drug['amphetamines'].map({'CL0': 0, 'CL1': 0, 'CL2': 0, 'CL3': 1, 'CL4': 1, 'CL5': 1, 'CL6':
df_drug['s2_cannabis'] = df_drug['cannabis'].map({'CL0': 0, 'CL1': 0, 'CL2': 1, 'CL3': 0, 'CL4': 1, 'CL5': 1, 'CL6': 1})
df_drug['s2_ecstasy'] = df_drug['ecstasy'].map({'CL0': 0, 'CL1': 0, 'CL2': 1, 'CL3': 0, 'CL4': 1, 'CL5': 1, 'CL6': 1})

df_drug['s3_amphetamines'] = df_drug['amphetamines'].map({'CL0': 0, 'CL1': 0, 'CL2': 0, 'CL3': 0, 'CL4': 1, 'CL5': 1, 'CL6':
df_drug['s3_cannabis'] = df_drug['cannabis'].map({'CL0': 0, 'CL1': 0, 'CL2': 0, 'CL3': 0, 'CL4': 1, 'CL5': 1, 'CL6': 1})
df_drug['s3_ecstasy'] = df_drug['ecstasy'].map({'CL0': 0, 'CL1': 0, 'CL2': 0, 'CL3': 0, 'CL4': 1, 'CL5': 1, 'CL6': 1})
```

I then ran binary classification and calculated the accuracy for each state:

- Accuracy for s1_amphetamines: 0.7135

- Accuracy for s1_cannabis: 0.9788
- Accuracy for s1_ecstasy: 0.7347
- Accuracy for s2_amphetamines: 0.8117
- Accuracy for s2_cannabis: 0.9443
- Accuracy for s2_ecstasy: 0.7507
- Accuracy for s3_amphetamines: 0.8912
- Accuracy for s3_cannabis: 0.9257
- Accuracy for s3_ecstasy: 0.8859

Comment in details:

With considering "Never Used", "Used over a Decade Ago", "Used in Last Decade", means there is not any frequent usage, and when we add new options to Class 0, the meaning of this model is not clear exactly to find certain usage.

s1_amphetamines (Accuracy: 0.7135):

The accuracy for predicting amphetamines in the first state (s1) is around 71.35%.

This accuracy suggests a moderately successful classification for amphetamines in the first state.

s1_cannabis (Accuracy: 0.9788):

The accuracy for predicting cannabis in the first state (s1) is impressively high at 97.88%.

This suggests that the model performs exceptionally well in identifying cannabis use in the first state.

s1_ecstasy (Accuracy: 0.7347):

The accuracy for predicting ecstasy in the first state (s1) is approximately 73.47%.

While not as high as the accuracy for cannabis, this result still indicates a reasonable performance for ecstasy classification in the first state.

s2_amphetamines (Accuracy: 0.8117):

The accuracy for predicting amphetamines in the second state (s2) is around 81.17%.

This represents a relatively high accuracy for amphetamines in the second state.

s2_cannabis (Accuracy: 0.9443):

The accuracy for predicting cannabis in the second state (s2) is quite high at 94.43%.

This indicates a strong performance in identifying cannabis use in the second state.

s2_ecstasy (Accuracy: 0.7507):

The accuracy for predicting ecstasy in the second state (s2) is approximately 75.07%.

While not as high as cannabis, this result still suggests a reasonable performance for ecstasy classification in the second state.

s3_amphetamines (Accuracy: 0.8912):

The accuracy for predicting amphetamines in the third state (s3) is around 89.12%.

This is a high accuracy, indicating a strong ability to classify amphetamines in the third state.

s3_cannabis (Accuracy: 0.9257):

The accuracy for predicting cannabis in the third state (s3) is 92.57%.

This represents a very high accuracy for cannabis use in the third state.

s3_ecstasy (Accuracy: 0.8859):

The accuracy for predicting ecstasy in the third state (s3) is approximately 88.59%.

This result suggests a strong performance for ecstasy classification in the third state.

the best setups, in terms of accuracy, appear to be for **cannabis** in the first and second states, with very high accuracies of 97.88% and 94.43%, respectively. The third best setup is for cannabis in the third state with an accuracy of 92.57%. Overall, the model shows strong performance in identifying cannabis use, especially in the second state, where the accuracy is the highest. **Amphetamines** also exhibit good accuracy, particularly in the third state. **Ecstasy**, while showing reasonable accuracy, doesn't perform as strongly as cannabis or amphetamines in these states.

Task 3

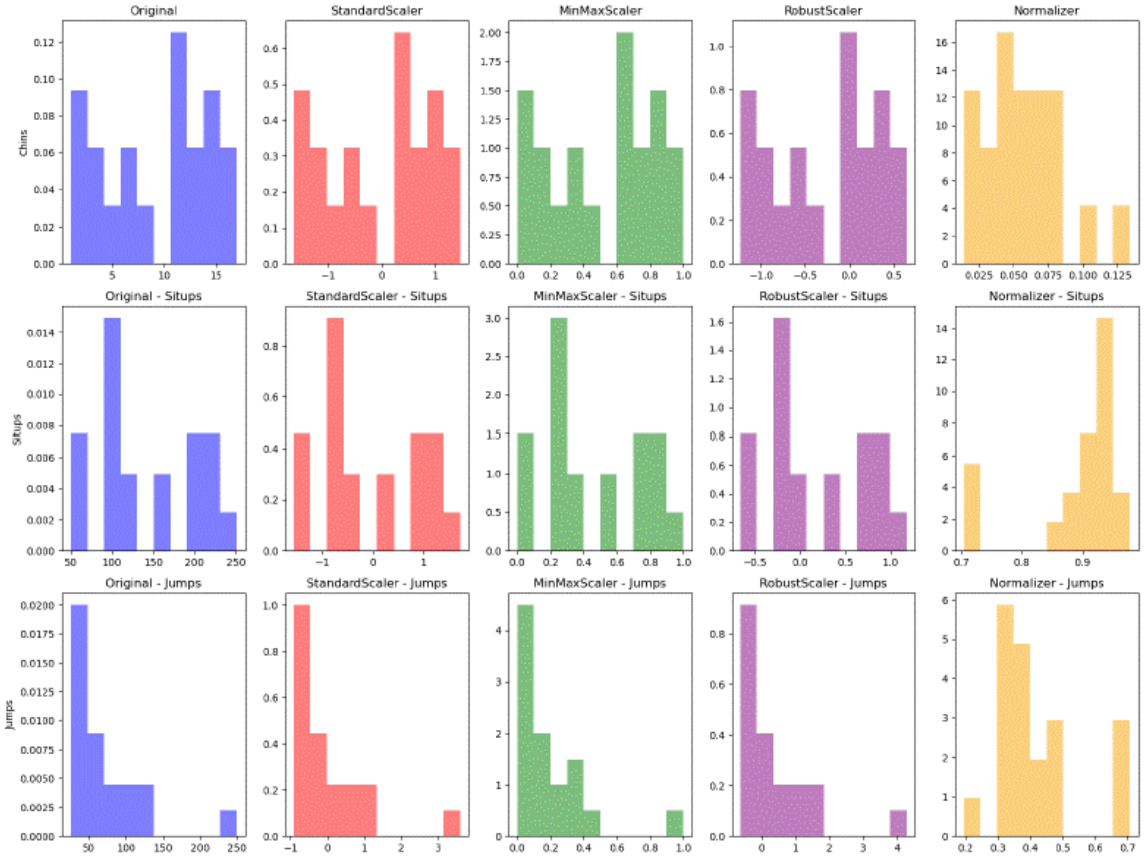
a. I loaded the "linnerud" dataset, which contains 3 features and 3 targets. Here are the feature and target names:

Feature Names: Feature 1: Chins Feature 2: Situps Feature 3: Jumps

Target Names: Target 1: Weight Target 2: Waist Target 3: Pulse

b. To preprocess the data, I applied four different Scalers: StandardScaler, MinMaxScaler, RobustScaler, and Normalizer, individually. Each scaler has its own method of scaling the data. After applying these scalers separately, I created visualizations to compare the original data with the data transformed by each of the four scalers. This allows us to see how the data distribution changes with each scaling method, helping us understand the impact of scaling on the dataset.

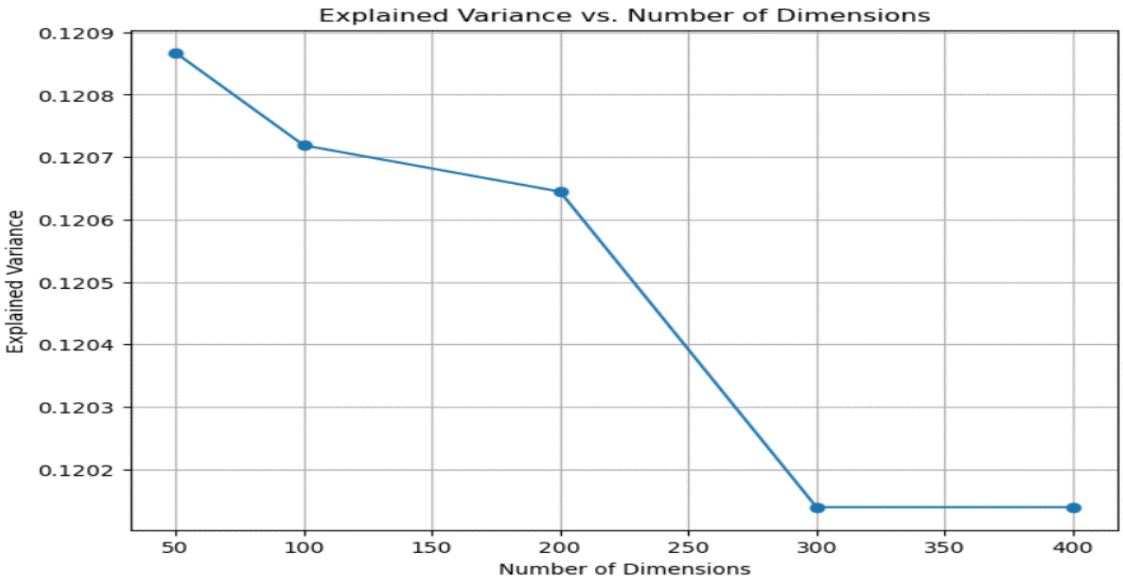
In the image, we see that Normalizer changes the data.



Task 4

a.

1. I loaded the Olivetti Faces dataset using **fetch_olivetti_faces**. This dataset contains facial images, and I prepared the data by extracting the images and their corresponding target labels.
2. To reduce the high dimensionality of the images (4096 features), I applied KernelPCA for dimensionality reduction.
3. I used the Support Vector Classifier (SVC) as a classifier and trained it on the data with different numbers of dimensions obtained after KernelPCA, trying at least 5 different values.
4. This process allowed me to assess how the model's classification score changed as a function of the number of dimensions.



b.The cluster distribution tables for both n_components = 200 and n_components = 100 show that the original clusters are not effectively recovered. In both cases, the true class labels are distributed across multiple clusters, indicating that the clustering process is not accurately capturing the original data structure. To improve cluster recovery, we may need to explore different dimensionality reduction and clustering methods.