Machine Learning
Course Project Report
(Phase-II)

Title of the project: AutoMPG

Student: Gayanthika Shankar (gayanthika.s-26@scds.saiuniversity.edu.in)

ML Category: Regression

---

1. Introduction

This project aims to develop a model for predicting the city-cycle fuel efficiency (miles per gallon, mpg) of automobiles. The significance of the Auto MPG dataset lies in its practical application in the automotive industry and environmental research. With growing concerns about fuel consumption and emissions, understanding the factors that influence a vehicle's fuel efficiency is crucial. This dataset provides insights into how different technical specifications of a car affect its mpg, aiding manufacturers in designing more fuel-efficient vehicles.

Problem statement: Predict the fuel efficiency (miles per gallon) of automobiles based on their various technical specifications

2. Dataset and Features

- Details of the dataset

The AutoMPG has a total of 398 samples(instances) with some missing values (6) in the 'horsepower' attribute. There are nine attributes (seven features) to work with. The attributes are elaborated in the following table:

Variable Name (type)	**Role**	**Description**
displacement (cont)	Feature	The engine displacement volume in cubic inches
mpg (cont)	Target	The city-cycle fuel efficiency in miles per gallon
cylinders (int)	Feature	The number of cylinders in the engine
horsepower(cont)	Feature	The engine horsepower
weight(cont)	Feature	The weight of the vehicle in pounds
acceleration(cont)	Feature	The time it takes in seconds to accelerate from 0 to 60 mph
model_year(int)[1]	Feature	The year the car was manufactured
origin(int)	Feature	The manufacturer's country[2] of origin
car_name (categorical)[3]	ID	The name of the car model

- EDA (Exploratory Data Analysis)

The dataset contains 398 rows and 9 columns. Let us take a look at the first five rows of the dataset:

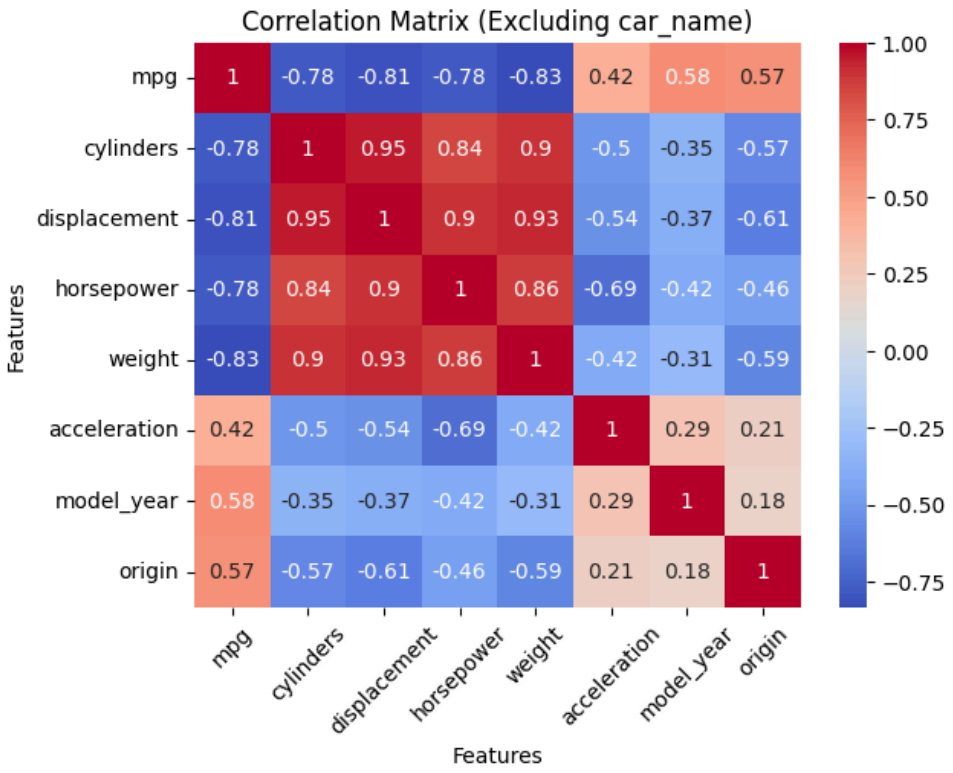
[1] int: multi-valued discrete
[2] 1: USA, 2: Europe, 3: Asia
[3] categorical: string

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino

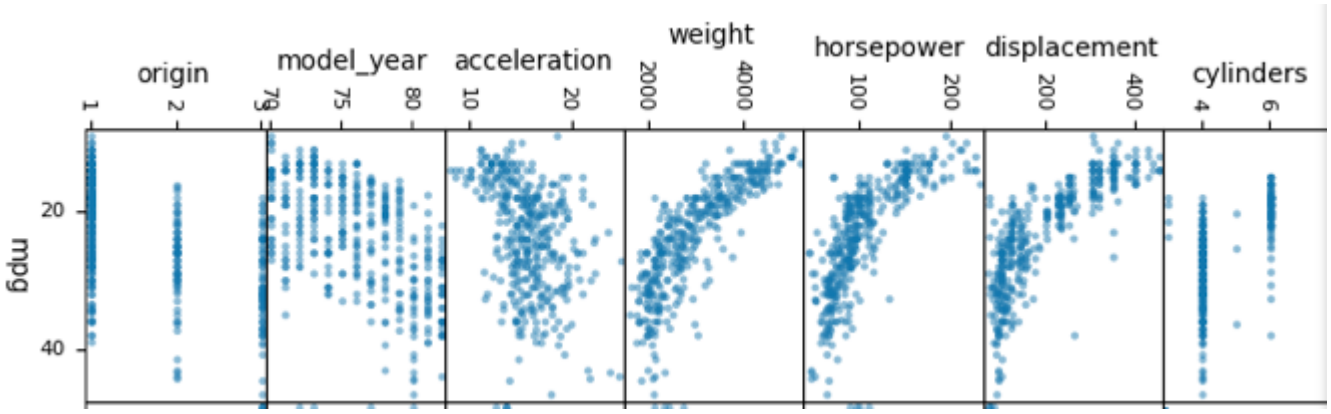
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    object
 4   weight        398 non-null    float64
 5   acceleration  398 non-null    float64
 6   model_year    398 non-null    int64
 7   origin        398 non-null    int64
 8   car_name      398 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

We observe that the columns are of mixed data types, float64 and int64. Notice that 'horsepower' has the type object, which is later corrected into float64. The dataset contains six missing values, all of which belong to horsepower. These values fall under the category of MCAR, where the likelihood of a data point being missing is completely random and not influenced by any other variables in the dataset. The missing values only make up 1.507% of the dataset (<5% of data), and are randomly distributed across the dataset. Hence, we drop the rows containing the missing values as part of data cleaning. Additionally, the column 'car_name' is dropped as part of feature selection as each name was unique and it would provide no information on predicting the mpg values.

To understand the relationship between the features in the dataset, we utilize visual graphical notations such as histograms, the correlation matrix, as well as a scatter plot.



We see that there is a strong negative correlation between displacement, horsepower, weight, and cylinders. This implies that as any one of those variables increases, the mpg decreases. However, displacement, horsepower, weight, and cylinders have strong positive correlations between themselves and this violates the non-multicollinearity assumption of linear regression. Multicollinearity hinders the performance and accuracy of the regression model. To combat this issue, we perform feature selection and/or PCA. The other variables (i.e. acceleration, model and origin) are not highly correlated with each other.



The image above is a rotated section of the scatter plot produced in the notebook. Observe that there are non-linear as well as linear correlations (which is explored through the line of best fit). This suggests that polynomial regression could be utilized to capture the relationship and produce an adequate model. Our dependent variable (mpg) is slightly skewed to the right. Further into feature selection, we cannot remove acceleration, weight, horsepower, or displacement due to their prominent effect on mpg. 'cylinders' has a negative, linear relationship with, mpg and most of all, 51.3% of the vehicles in the dataset have 4 cylinders. Thus we cannot remove 'cylinders'. 'model_year' and 'origin' have a positive linear relationship, yet cannot be dropped due to the correlation with the cars brand and cannot be generalized. Car technology advancements over time can lead to better fuel efficiency. Similarly, car manufacturers from certain regions might be known for focusing on fuel efficiency. These factors can influence mpg even if the correlation isn't perfect.

3. Methods

We prepare three datasets for the model.

1) The regular dataset
2) The standardized dataset
3) The PCA reduced dataset

These are created to perform multiple experiments and come up with the best model.

3.1 Baseline - Linear Regression

We apply multivariate linear regression as there are multiple explanatory variables (cylinders, displacement, horsepower weight etc) influencing the target variable.

We work with df_filtered, which contains 392 rows and 8 columns, from which we create our three datasets, X, X_std, and X_pca. The mpg column is stored as y (the target variable) and the remaining are stored under X as the input features. We split the dataset into 75% training and 25% testing, we can then analyze the data to find the equation of a flat, multidimensional surface that best fits the data points. Most importantly the R^2 score of the model is calculated to reflect how well the model fits the data.

Scatter Plots of the actual versus predicted value as well as histograms of the residual errors are utilized to further explore the accuracy of the model.

Apart from studying the model's ability to fit the data, we want to evaluate the models ability to generalize new data, this is done by calculating the cross-validation score. Cross-validation tests a model's generalizability by splitting data into folds. It iteratively uses one fold for testing and the rest for training. Analyzing the mean and standard deviation across folds assesses the model's average performance, ensuring it generalizes well to unseen data.

The cross-validation score of the regular dataset for multivariate linear regression is 0.81 ± 0.04 and 0.64 ± 0.21 for the entire dataset. The significant difference between the CV score (0.81) and the entire dataset score (0.64) suggests potential overfitting. The model might be memorizing noise or irrelevant patterns in the training data, leading to better performance on the training data itself (entire dataset score) but potentially worse performance on unseen data (reflected in the CV score). Therefore, we perform regularization to resolve overfitting.

LASSO stands for 'Least Absolute Shrinkage and Selection Operator'. It tries to eliminate the least important features by setting their weights close to zero. The LASSO score for the regular is higher than the standardized data, followed by the PCA reduced data performs the worst with a score of 0.7524.

Ridge regression excels at handling datasets with many features, like ours, by keeping its model parameters under control. This makes it a great choice for our situation. It tries to reduce the weights as much as possible. Ridge performs better than LASSO, with scores of 0.7990, 0.7976, and 0.7541.

ElasticNet is a combination of L1 and L2 regularization. The weightage given to L1 and L2 can be controlled by 'r'. It performs slightly poorer than Ridge but better than LASSO with R^2 scores of 0.7996, 0.7888, 0.7530.

The above method is implemented for three datasets we've created. For X, we implement basic linear regression. For X_std, we utilize StandardScaler to apply feature scaling, which produces a mean 0 and variance 1 to all features. For X_pca we apply principal component analysis to reduce the dimensionality, bringing the dimensions for 7 to 4

Moving forward, we apply polynomial regression. Polynomial regression is a type of regression analysis used to model non-linear relationships between a dependent variable and an independent variable. We implement polynomial regression with degree 2, as higher degrees lead to a negative R^2 score (which suggests that the model deteriorates). The R^2 scores of the polynomial regression for all three datasets are higher than that of multivariate linear regression and those of the scores of regularization.

3.2 Support Vector Machines

Support vector machines for regression aims to find a hyperplane in a high-dimensional space that best fits the training data. It is a type of supervised learning algorithm.

Kernel SVM regression tackles non-linear relationships. It projects data into a higher dimension where a simple linear model works, effectively creating a non-linear model in the original space.

A linear kernel assumes a linear relationship between features and the target variable and treats data points in their original form. Fitting the regular data gave a score of 0.6626 for the regular data, 0.7803 for the standardized data, and 0.7484 for the PCA reduced data.

A polynomial kernel allows the model to capture non-linear relationships by creating new features based on polynomial interactions between existing features. The scores produced for this kernel were the lowest produced out of all models for all three datasets, the lowest being 0.2661 for the PCA reduced data.

The RBF kernel excels at finding hyperplanes in high-dimensional space that separate even very non-linear data. It achieves this by measuring similarity based on data point distance in the original space. The RBF kernel resulted in the highest scores for the three datasets out of all three kernels: 0.7102, 0.8209, and 0.8190.

3.3 Decision Tree

Decision trees are non-parametric ML algorithms capable of performing both regression and classification. Non-parametric models are those who have a variable number of parameters. The algorithm used to train decision trees is the CART algorithm, which is a greedy algorithm. While they are considered to be a better algorithm than the ones performed previously, decision trees are sensitive to data rotation and have high variance. Therefore, small changes in hyperparameters or the data may produce very different models.

Fitting the datasets, we see that the score for the regular and standardized datasets are equal (both being approximately 0.8077), this is because decision trees are indifferent to standardization or any sort of feature scaling. We can also observe that the score for the PCA reduced data is much lower (around 0.6761).

3.4 Random Forest

Random forests are an ensemble of decision trees and are generally trained via the bagging method. Instead of searching for the best feature when splitting a node, it searched for the best feature among a random subset of features. Random forests make it easy to measure the recreational importance of each feature and are considered as one of the best machine learning algorithms.

The scores for the regular and standardized datasets are similar (around 0.8745) and is higher than that of the PCA reduced dataset (0.8602)

3.5 AdaBoost

Boosting is an ensemble method combining several weak learners into a strong learner. The general idea is to train predictors sequentially, where each predictor tries to correct its predecessor. There are two boosting methods, we will look at the first one: AdaBoost.

AdaBoost – also known as AdaptiveBoosting – pays more attention to the training instances that the predecessor underfits, and hence results in new predictors focusing more and more on the hard cases. This is done by assigning more weight to the misclassified training instances.

The scores when the datasets are fitted are as follows: 0.8847, 0.8819, and 0.8743.

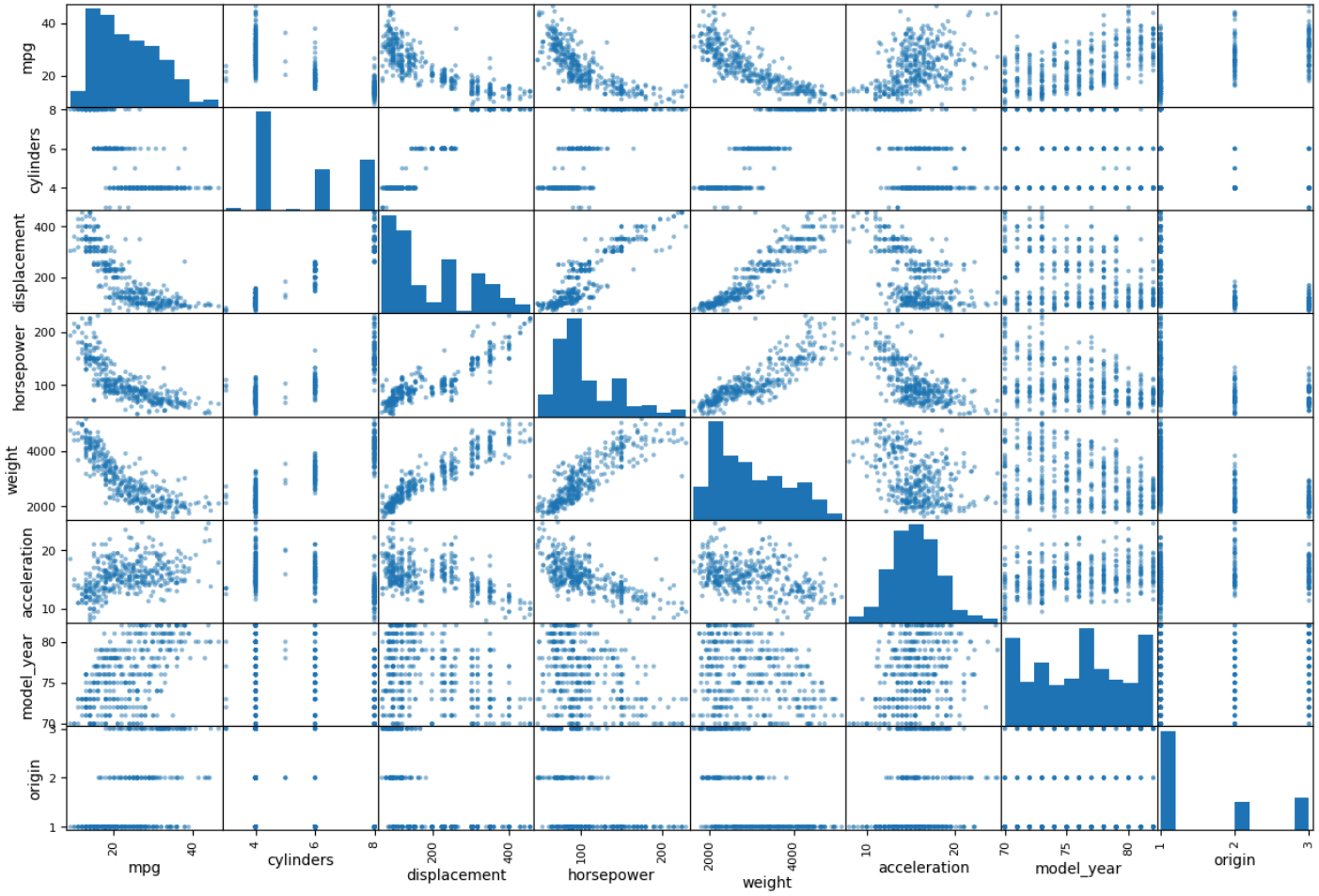
3.6 Gradient Boosting

Gradient boosting is the second boosting method, and like AdaBoost, it works sequentially adding predictors to the ensemble. Yet instead of tweaking the instance weights at every iteration, gradient boosting tries to fit the new predictor to the residual errors made by the previous predictors.

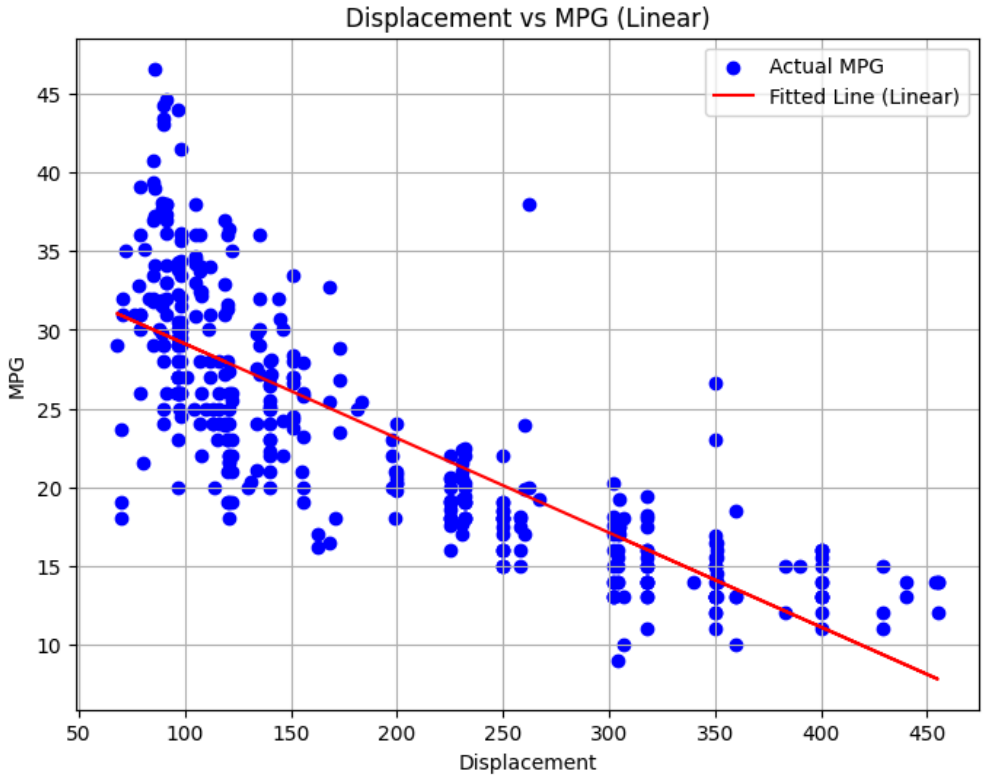
The scores when the datasets are fitted are as follows: 0.8453, 0.8485, and 0.8070.

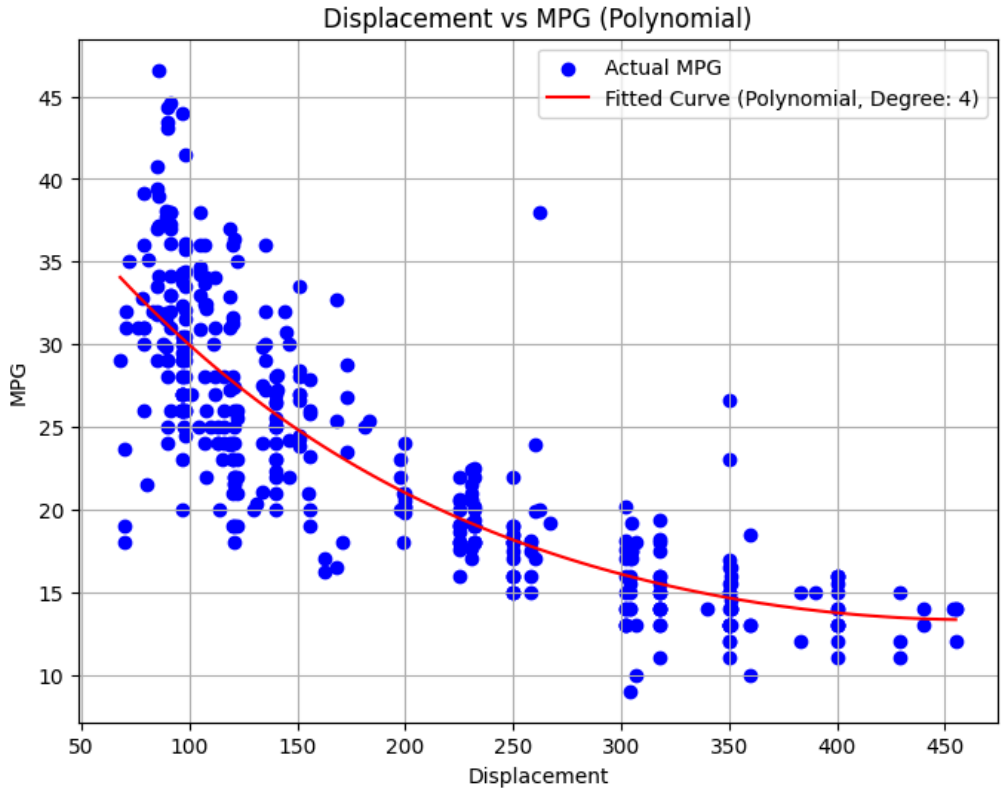
4. Results

- Scatterplots of features



- Fitting a linear and polynomial line of best fit between one of the features and the target variable to get an understanding of what model would work best with the data (as part of EDA)





- R^2 cross validated scores of all models (training data - model accuracy)

(Note: highest scores are italicized for each model and the highest R^2 score is underlined)

Model	**Regular Data**	**Standardized Data**	**PCA reduced data**
Linear Regression - Multivariate	*0.8098000094768565 ± 0.04126349533447571*5	*0.8098000094768565 ± 0.04126349533447571*5	0.7879570765274106 ± 0.05358772930830503
L1/LASSO Regression	0.8102056233282484 ± 0.044353483893364624	*0.8113134003442186 ± 0.045151165052256685*	0.7874287588856392 ± 0.049902938106662235
L2/Ridge Regression	0.809844428069445 ± 0.04138764747125511	*0.8100693512202227 ± 0.04168156429118428*	0.7880232953177015 ± 0.05334050517178408
Elastic Net Regression	*0.8098970263248063 ± 0.0442810343710217*	0.8074635134627067 ± 0.040897749058376336	0.7879623888169068 ± 0.049044660606572824
Polynomial Regression	0.8098000094768565 ± 0.041263495334475715	*0.8098000094768567 ± 0.04126349533447642*	0.7879570765274106 ± 0.05358772930830503
Linear SVM	0.5945186010161079 ± 0.16244801627195232	*0.8000779287396718 ± 0.04558675902460143*	0.7814353946934328 ± 0.05643938844648777
Polynomial SVM	*0.6404596352936835 ± 0.06996740476652724*	0.29653799386192736 ± 0.10261047493897021	0.21677550483783298 ± 0.09963091725292822
RBF SVM	0.7012425467204537 ± 0.07929292814032703	*0.8347493182260568 ± 0.05345007295891266*	0.8294173898925348 ± 0.05459477420359437
Decision Tree	*0.7902699045449595 +/- 0.07133062238701596*	*0.7902699045449595 +/- 0.07133062238701596*	0.6761780460784608 +/- 0.157255085678348
Random Forest	*0.8585241017455395 ± 0.04463758305933999*	0.8584581709134035 ± 0.04474131567228862	0.8330067624709073 ± 0.05156124903386635
AdaBoost	0.854977338219596 ± 0.048540171250264885	*0.8576495751634475 ± 0.04729878741978559*	0.8188734409873912 ± 0.06462310474636518

Gradient Boosting	*0.8474069198387986 ± 0.04942931269290906*6	0.8456366858408841 ± 0.049706541274652315	0.8171593198049314 ± 0.06548738012471672

- R^2 scores of all models (testing data - model generalization)

(Note: highest scores are italicized for each model and the highest R^2 score is underlined)

Model	**Regular Data**	**Standardized Data**	**PCA reduced data**
Linear Regression - Multivariate	*0.7988908872869827*	0.798890887286982	0.7540019357925587
L1/LASSO Regression	*0.7976667768749909*	0.7944282199102781	0.7524750518117207
L2/Ridge Regression	*0.7990253547730505*	0.7976058398848342	0.7541838209622151
Elastic Net Regression	*0.7996378546043167*	0.788886980658066	0.7553095427854379
Polynomial Regression	0.8589493832283672	0.8589493832263562	*0.8666281843663011*
Linear SVM	0.6626344429353861	*0.7803959546531063*	0.7484203677634447
Polynomial SVM	*0.6579830253884889*	0.346144246374533	0.2661154010946981
RBF SVM	0.7102030559488015	*0.8209935120539685*	0.8190764424356886
Decision Tree	*0.8077493898600785*	*0.8077493898600785*	0.6761493557111735
Random Forest	0.8742778138657534	*0.8745500405394342*	0.860222278953527
AdaBoost	*0.8847129235449682*	0.881987536007458	0.8743430966654674
Gradient Boosting	0.8453715021388497	*0.848568403782159*	0.8070889193313175

5. Conclusion I

AdaBoost gave the best model with an R^2 of the regular data being around 0.8847, and the highest scores out of all the other models. Random Forest followed closely behind with a consistent average score of 0.8693[4]. This highlights the strength of Random Forests in capturing complex relationships within the data through its ensemble learning approach.

Polynomial Regression follows next, achieving the highest average scores (around 0.8615) across all data pre-processing techniques. Gradient Boosting also displayed promising results, delivering consistent scores ranging from 0.8071 to 0.8847. As mentioned before, this technique iteratively builds a model by learning from the errors of previous models, potentially explaining its strong performance.

Interestingly, Decision Trees performed well on both the regular and standardized data (likely due to their ability to handle non-linear relationships), but their performance suffered significantly with PCA dimensionality reduction (score of 0.6761). This suggests that PCA might have removed crucial information for the decision tree algorithm.

Out of the three regularization techniques, Ridge regression produces the next best model with an average R^2 score being around 0.7828. This could be due to its ability to balance between reducing model complexity. Elastic Net regression followed closely behind (average score of 0.78127), with LASSO showing the lowest average score (0.7815). This highlights its limitations in capturing nonlinear patterns. These methods show consistent performance trends across different preprocessing techniques, with PCA reduced data resulting in lower scores due to potential loss of important features.

The kernel-based RBF SVM achieved a respectable average R^2 score of 0.7356, indicating its capability of handling non-linear relationships by projecting data into a higher-dimensional space. Linear SVM, on the other hand,

[4] The scores mentioned in this report are approximates

underperformed compared to its kernel counterpart (average score: 0.7304), suggesting the data might exhibit nonlinear patterns that the linear model couldn't capture effectively.

Finally, Polynomial SVM displayed the poorest performance across all data pre-processing techniques, with scores dropping significantly (0.6579, 0.3461, and 0.2661). This indicates potential overfitting and poor generalization.

In conclusion, the findings suggest that AdaBoost, Random Forest, and Gradient Boosting are strong contenders for further investigation due to their high performance and ability to handle complex data relationships.

5. Hyperparameter Tuning

Hyperparameters are parameters that are not directly learnt within estimators, and therefore must be set prior to training. The process of tuning such hyperparameters is vital in building a machine learning system simply because we want the best performing model. However, if overperformed, the performance of the model would be worse than what was measured during cross-validation. This is due to the system being tuned to perform well on the validation data and not on unknown datasets.

There are two ways to implement hyperparameter tuning: GridSearch and RandomSearch. GridSearch considers all hyperparameter combinations. This algorithm works by selecting a hyperparameter and testing various values for it. It uses cross-validation to assess the performance of these combinations and ultimately identifies the optimal set of hyperparameters. RandomSearch can sample a given number of candidates. Typically, a budget is allocated independently of the number of parameters and their possible values in RandomSearch. It's important to note that including parameters that have minimal impact on performance does not reduce efficiency significantly.

5.1 SVM with RBF kernel

● Explanation of the hyperparameters tuned

Since we are working with SVR (Support Vector Regression) in this project, the hyperparameter we will be tuning is epsilon, along with C and gamma.

Epsilon sets a tolerance margin where errors within this margin are not penalized, enhancing the model's robustness to noise. The regularization parameter, C, balances low training error and model complexity. A smaller C value results in a wider margin but permits more violations, creating a simpler model with more errors. In the context of the RBF kernel, Gamma controls the decision boundary's curvature, with high values potentially leading to overfitting.

● Parameter grid

'C': [1, 2, 3, 4, 5, 6, 7, 8, 9],

'gamma': array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.]),

'epsilon': [0,1, 2, 3, 4, 5, 6, 7, 8, 9]

We will be using GridSearch as the performance may be slightly worse in RandomSearch if there is any noise present and also that it doesn't explore the entire search space – rather a subset of it.

● Result obtained for the best configuration

The best estimators for SVM with RBF kernel (SVR): SVR(C=9, epsilon=1, gamma=0.2)

The best R2 score for SVM with RBF kernel (SVR): 0.8708278284655335

● Result for the 25% testing dataset

R2 score of 25% data: -0.002159972721777592

5.2 Decision Trees

● Explanation of the hyperparameters tuned

max_leaf_nodes limits the number of leaf nodes in the tree.

max_depth limits the maximum depth of the tree

min_samples_split is the minimum number of samples required to split an internal node.

Both max_leaf_nodes and max_depth should be lowered to reduce overfitting, while min_samples_split should be increased/have a higher value.

● Parameter grid

'max_leaf_nodes': [10, 20, 30, 40],
'max_depth': [5, 10, 15],
'min_samples_split': [2, 5, 10]

● Result obtained for the best configuration

The best estimators for Decision Trees: DecisionTreeRegressor(max_depth=5, max_leaf_nodes=30, min_samples_split=10, random_state=42)

The best R2 score for Decision Trees: 0.8172823526866762

● Result for the 25% testing dataset

R2 score of 25% testing data: 0.8178987872769241

5.3 Random Forest

● Explanation of the hyperparameters tuned

'max_leaf_nodes' limits the number of leaf nodes in each decision tree within the forest.
'n_estimators' is the number of trees in the forest. Usually, more trees lead to a better performance.

● Parameter grid

'max_leaf_nodes': [10, 15, 20, 30, 40],
'n_estimators': [50, 100, 150, 200],
'n_jobs': [-1],
'random_state': [42]

● Result obtained for the best configuration

The best estimators for Random Forest:
RandomForestRegressor(max_leaf_nodes=40, n_estimators=200, n_jobs=-1, random_state=42)

The best R2 score for Random Forest: 0.8634273335519678

● Result for the 25% testing dataset

R2 score of 25% testing data: 0.8758176638887478

5.4 AdaBoost

● Explanation of the hyperparameters tuned

learning_rate shrinks the contribution of each classifier.

estimator__max_depth limits the maximum depth of the individual estimators (typically decision trees).

estimator__min_samples_leafis the minimum number of samples required for a leaf node to be created.

● Parameter grid

'learning_rate': [0.01, 0.1, 0.5, 1],
'estimator__max_depth': [5, 10, 15],
'estimator__min_samples_leaf': [1, 5, 10]

● Result obtained for the best configuration

The best estimators for AdaBoost:
AdaBoostRegressor(estimator=DecisionTreeRegressor(max_depth=10, min_samples_leaf=5), learning_rate=0.01, random_state=42)

The best R2 score for AdaBoost:0.8607966495491809

● Result for the 25% testing dataset

R2 score of 25% testing data: 0.8857901626630696

5.5 Gradient Boosting

● Explanation of the hyperparameters tuned

learning_rate shrinks the contribution of each classifier.

n_estimators in the case of gradient boosting is the number of boosting stages to be run.

max_leaf_nodes limits the number of leaf nodes in the tree.

● Parameter grid

'learning_rate': [0.01, 0.1, 0.5, 1],
'n_estimators': [50, 100, 150, 200, 250],
'max_leaf_nodes': [10, 15, 20]

● Result obtained for the best configuration

The best estimators for GradientBoost:
GradientBoostingRegressor(max_leaf_nodes=10, n_estimators=200, random_state=42)

The best R2 score for GradientBoost: 0.8759263356768365

● Result for the 25% testing dataset

R2 score of 25% testing data: 0.848568403782159

6. Results after hyperparameter tuning

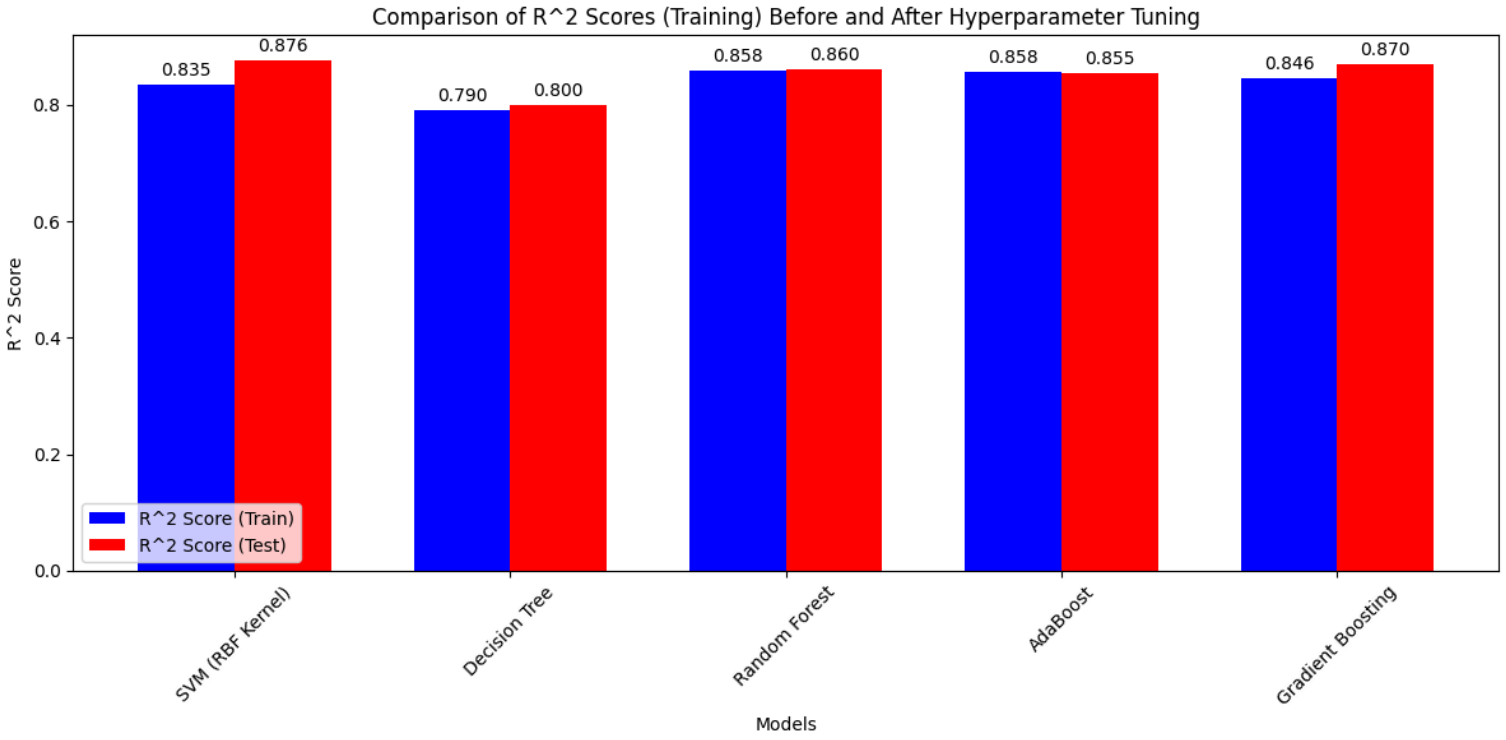
- R^2 cross validated scores of all models (training data - model accuracy) on the standardized dataset

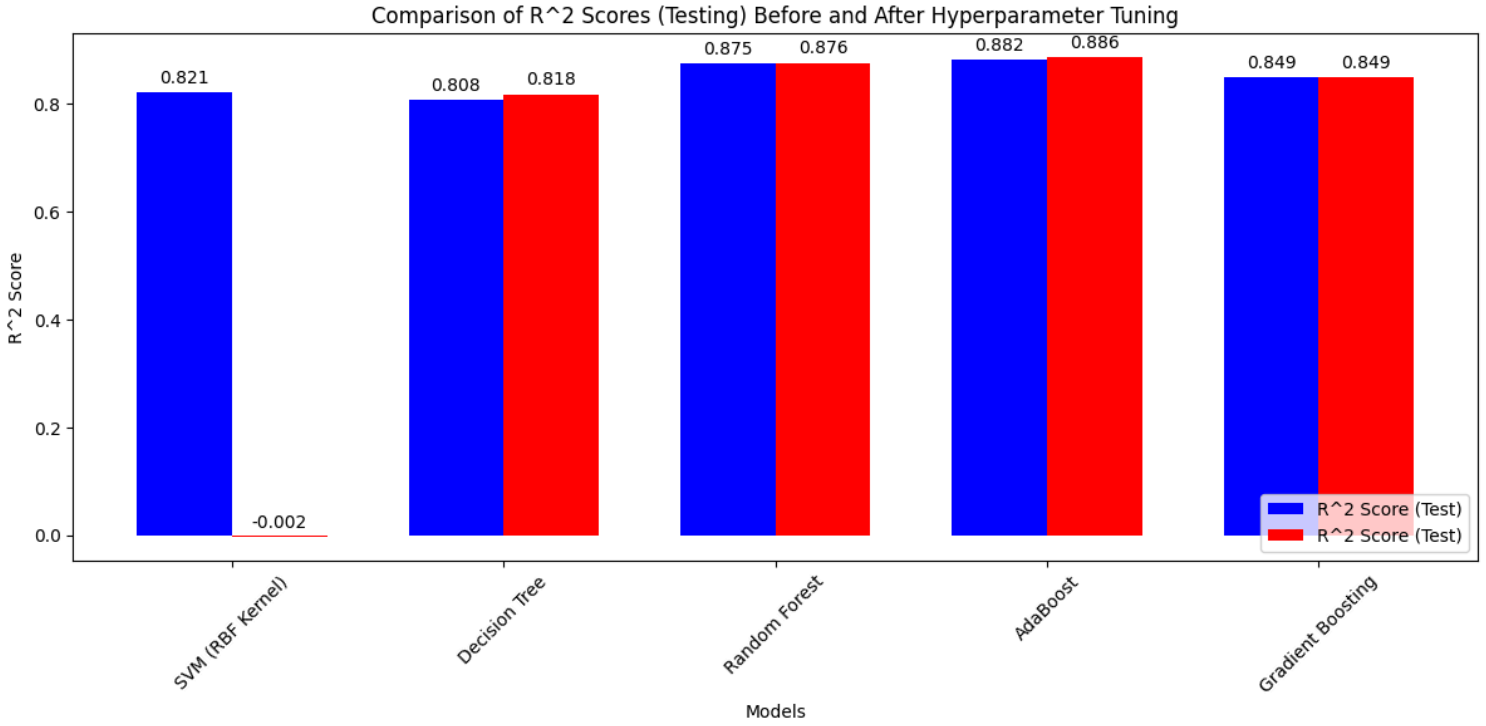
(Note: highest scores are italicized per column and the best model is underlined)

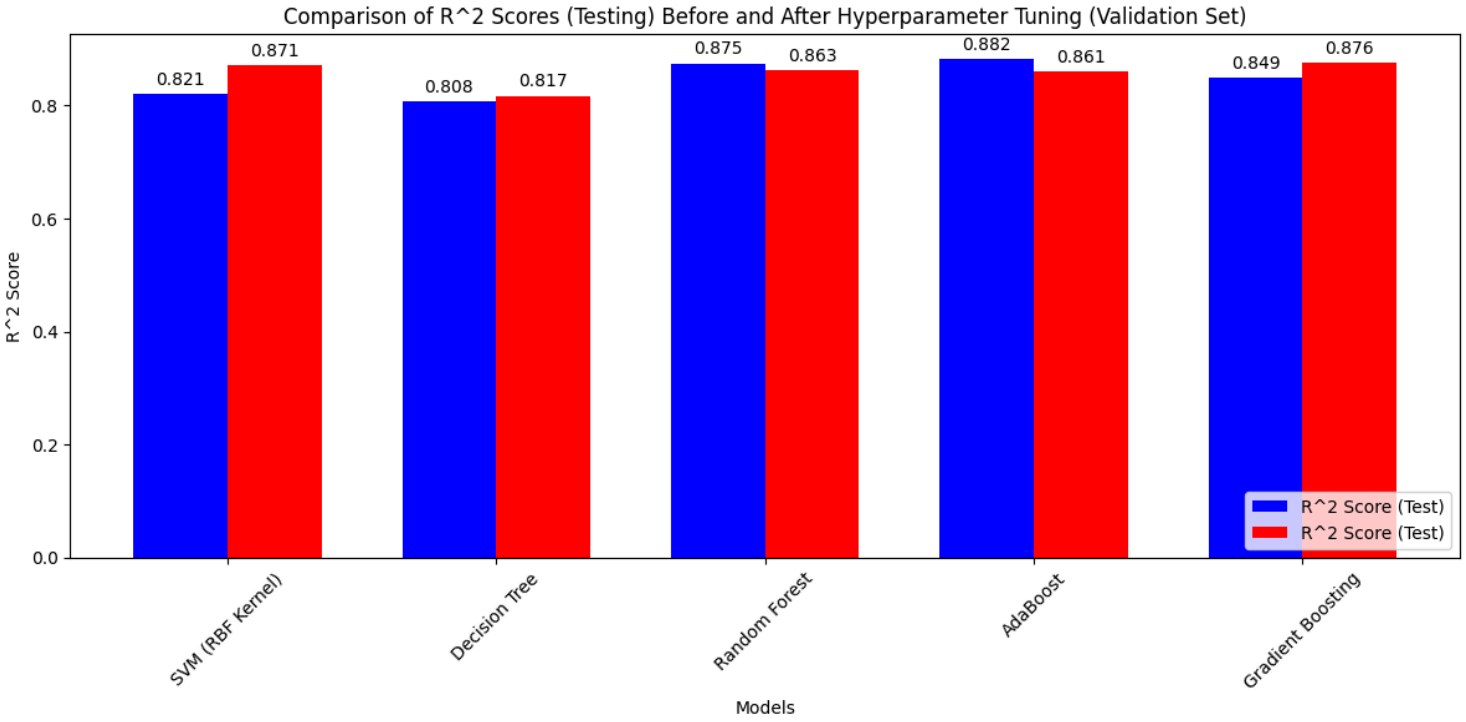
Model	R^2 Score (Train[5])	R^2 Score (Test[6])	R^2 Score (Train) after Hyperparameter Tuning	R^2 Score (Test) after Hyperparameter Tuning	Best R^2 Score for model generalization (performance on unseen data)
SVM (RBF Kernel)	0.8347493182260568 ± 0.05345007295891266	0.820993512053968 5	*0.8760897727335604 ± 0.0386644448857064*	-0.002159972721777592	0.8708278284655335
Decision Tree	0.7902699045449595 ± 0.07133062238701596	0.807749389860078 5	0.7999140794774294 ± 0.05897125728648034 6	0.8178987872769241	0.8172823526866762
Random Forest	*0.8584581709134035 ± 0.04474131567228862*	0.874550040539434 2	0.8604754349748308 ± 0.04484463084272774 5	0.8758176638887478	0.8634273335519678
AdaBoost	0.8576495751634475 ± 0.04729878741978559	*0.881987536007458*	0.8554151880557728 ± 0.04427979655004686	*0.8857901626630696*	0.8607966495491809
Gradient Boosting	0.8456366858408841 ± 0.04970654127465231 5	0.848568403782159	0.8701785537685509 ± 0.0427743411454075	0.848568403782159	*0.8759263356768365*

[5] The 75% training dataset
[6] The 25% testing dataset

- Bar-plots containing the comparisons before and after hyperparameter tuning.







7. Feature Reduction

We now train the AdaBoost model using our PCA reduced dataset, obtaining an R^2 value of approximately 0.8768. This high R^2 score suggests that PCA effectively captures the essential variance in the data, allowing the AdaBoost model to maintain strong predictive performance even with a reduced feature set.

8. Feature Selection

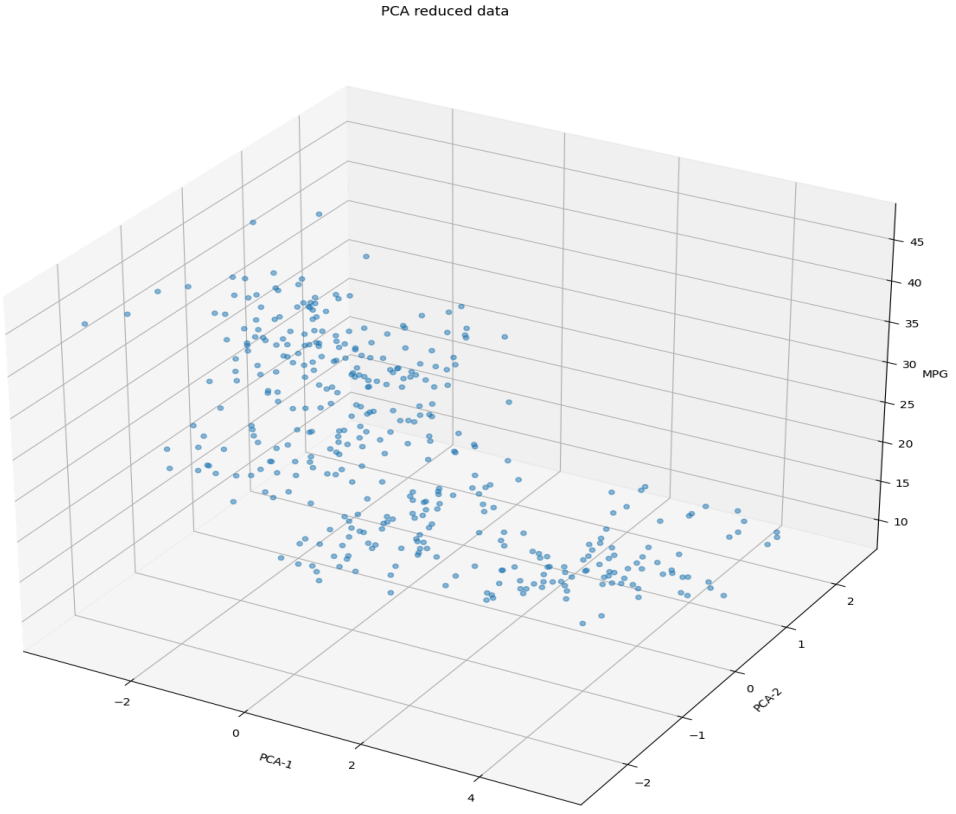
For feature selection, we use the SelectPercentile method to keep only the top user-defined percentage of features with the highest scores (in this case, the top 50%). Applying this method to our best model results in an R^2 score of approximately 0.6428. This score is lower than that of the PCA-reduced model, suggesting that while SelectPercentile identifies the most relevant features, it may not capture the complex relationships between features as effectively as

PCA. SelectPercentile filters features based on statistical measures, which may not always account for crucial feature interactions.

9. Data Visualization with the aid of PCA and t-SNE

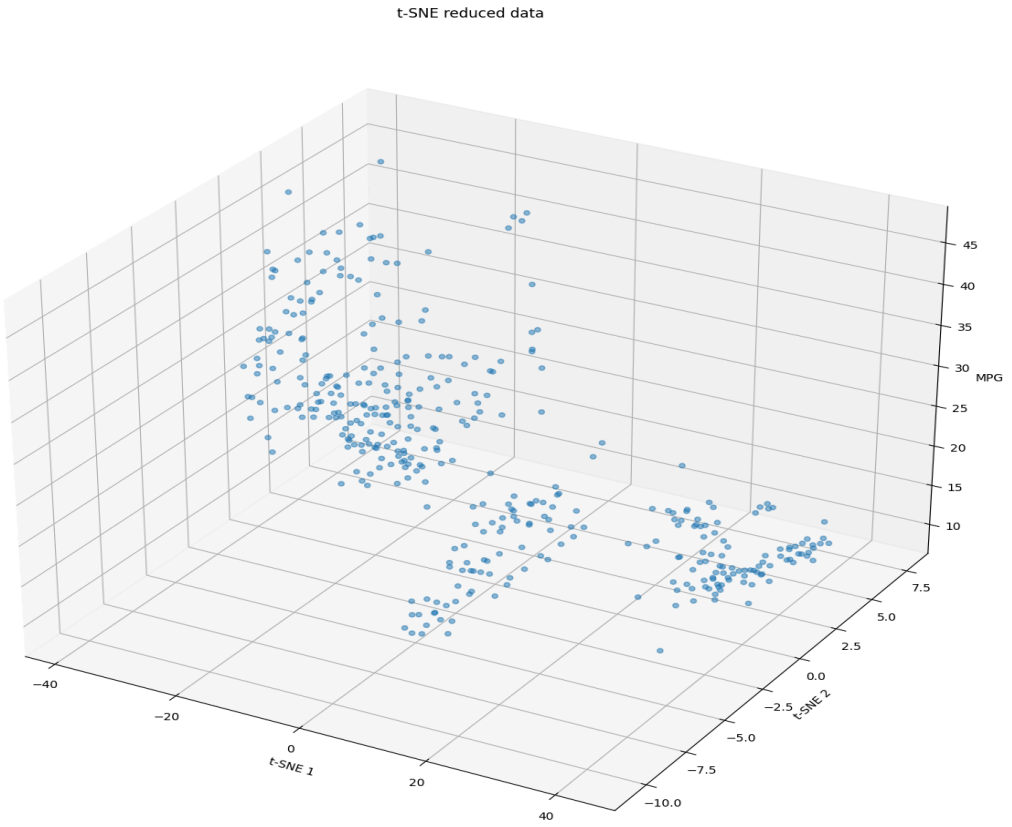
Principal Component Analysis (PCA) is a projection based approach that reduces the features of the data while retaining the greatest proportion of the original datasets variance. It is a dimensionality reduction method which results in faster training and aids in visualizing data.

(The following images are visualizations of the reduced data)



t-Distributed Stochastic Neighbour Embedding (t-SNE) is a compute heavy process that is primarily used for the visualization process. It leverages pairwise similarities between data points to construct probability distributions around each point in the high-dimensional space. The constraint here is that the distributions in the original and lower dimension should be as close as possible. By doing so, it preserves the structure of each feature while it reduces

the dimensionality. However, it is important to note that t-SNE is not a convex optimization problem, and hence is not useful to reduce the complexity of the problem.



10. Conclusion II

Among the algorithms tested, AdaBoost exhibited the best performance, achieving the highest test R^2 score after hyperparameter tuning. This indicates its strong generalization capabilities. The tuned AdaBoost model achieved an R^2 score of 0.8857 on the test dataset, a slight improvement from its pre-tuning score of 0.8819. The training R^2 score remained stable, highlighting the model's robustness. It is important to consider the R^2 score on the 25% testing data for selecting the best model for future use. In this

project, prioritizing the test set over the validation set is crucial to achieving the best predictions for the auto mpg dataset. Therefore, considering said score as well as the stable R^2 training score proves that AdaBoost is indeed the best model after hyperparameter tuning. Its performance can be attributed to its ensemble technique, which combines multiple weak learners into a strong learner, effectively reducing bias and variance to improve model performance.

The SVM with RBF kernel initially showed promising results with a high training R^2 score of 0.8347 but performed poorly on unseen data with a test R^2 score of -0.0022 after tuning. This drastic drop indicates significant overfitting. Before tuning, the test R^2 score was 0.8209. The best R^2 score after hyperparameter tuning was 0.8708, showing potential during training but failing to generalize well. The SVM's sensitivity to hyperparameters and its tendency to overfit with a high-dimensional feature space made it less suitable for this dataset.

Decision Trees showed slight improvement after tuning parameters s. The training R^2 score increased from 0.7902 to 0.7999, and the test R^2 score improved from 0.8077 to 0.8178, indicating better generalization. The best R^2 score after tuning was 0.8173, a slight improvement over the pre-tuning training R^2 score, reflecting the positive impact of tuning. However, Decision Trees are prone to overfitting, and despite hyperparameter tuning, they might still not capture the complexity of the dataset as effectively as ensemble methods.

Random Forest maintained high performance and good generalization post-tuning. The training R^2 score improved slightly, and the test R^2 score improved marginally from 0.8745 to 0.8758, both reflecting stable and reliable performance. The best R^2 score after tuning was 0.8634, a slight increase over the pre-tuning training R^2 score. Random Forest benefits from averaging multiple trees to reduce overfitting and improve predictive accuracy, making it a robust choice for diverse datasets.

Gradient Boosting showed significant improvement after hyperparameter tuning. The training R^2 score increased from 0.8456 to 0.8701, and the test R^2 score remained stable at 0.8486, indicating a good balance between model

complexity and generalization. The best R^2 score after tuning was 0.8759, a notable improvement over the pre-tuning training R^2 score. Gradient Boosting works well due to its iterative boosting process, which sequentially improves the model by focusing on the errors of previous iterations.

Overall, AdaBoost's superior performance can be attributed to its ensemble technique, hyperparameter sensitivity, and ability to handle overfitting, making it the best choice for this dataset.

11. References

https://archive.ics.uci.edu/dataset/9/auto+mpg

https://medium.com/@yennhi95zz/handling-missing-data-in-data-preprocessing-and-cleaning-methods-and-examples-19a893336b2a

https://www.geeksforgeeks.org/ml-handling-missing-values/

https://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html

https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html#sklearn.feature_selection.VarianceThreshold

https://www.analyticsvidhya.com/blog/2021/10/understanding-polynomial-regression-model/

https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html#svr

https://scikit-learn.org/stable/auto_examples/model_selection/plot_randomized_search.html

https://www.geeksforgeeks.org/comparing-randomized-search-and-grid-search-for-hyperparameter-estimation-in-scikit-learn/

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html#sklearn.tree.DecisionTreeRegressor

https://scikit-learn.org/stable/auto_examples/ensemble/plot_adaboost_regression.html

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html#gradientboostingregressor

https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectPercentile.html#sklearn.feature_selection.SelectPercentile

https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectPercentile.html#sklearn.feature_selection.SelectPercentile