

Module: B9BA102_Applied Statistics & Machine Learning

CA-2 Group Assignment



Regression Analytical Report

By

Manisha - 20025385

Abhishek Sen - 20012367

Kashmira Ghag - 20023193

Lecturer: Assem ABDELHAK

Submission Date: April 25th,2024

Table of Contents

Introduction.....	3
1. Dataset summary.....	3
2. Data Preparation	4
Data Cleaning	4
Library import and reading dataset.....	5
Encoding for binary features	5
Data Splitting	6
Feature reduction and Target encoding	7
I. Manual drop.....	7
II. Target encoding.....	7
III. Heatmap	7
IV. Important features	8
Data Scaling/Transformation.....	9
3. Regularization Techniques.....	9
LASSO - L1 Regularization	10
RIDGE – L2 regularization	10
Elastic Net Regularization	10
4. Linear Regression (LR) and Impact of Regularizations	11
Linear Regression (LR) with “None” Penalty	11
Linear Regression (LR) with L1 Regularization.....	12
Linear Regression (LR) with L2 Regularization.....	14
Linear Regression (LR) with Elastic Net Regularization	15
Comparison between Different Regularization Method in LR.....	16
Impact of L2 (C) Regularization on SVR	16
Impact of L2 (C) regularization on SVR performance and interpretability.....	17
Result and Impact Analysis of L2 Regularization on SVR Model	18
5. Comparison with Random Forest Regression.....	18
Random Forest Regression	18
Linear Regression with (Elastic Net)	19
Support Vector Regression (SVR)	19
6. Prediction -New data	19
Conclusion	20
Appendix – Team contribution	21
Coding	21

Introduction

This report includes analysis and the findings about the predictive analytics and regression modelling. After all the research and analysis, we decided to work on the dataset focused on laptop information which had various features and specifications. Our first objective was to apply statistical regression techniques as required for better understanding of the dataset. We build the code using Google Collab, to know the performance and results of various regression models like Linear regression, Random Forest regression, and Support vector regression (SVR) while using different regularization methods and their tuning. During working on the dataset, we tried to understand how all these different attributes in the dataset influence the laptop prices and, how the specific laptop features or specifications contribute to higher and lower prices in the market.

The main objective of this analysis was to use data analytics and machine learning algorithms while understanding the dataset and using its best features to find the best regression model which would help in predicting the prices of the laptops which could be used in real world for data driven decision making or similar purposes.

In this report, we are going to discuss all the steps and practices used to build the predictive model. We started with data selection and moved further steps of “Data preprocessing” like data cleaning, data encoding, feature selection, feature deduction, and data scaling. After data pre-processing, we build many regressors or models to find the model with best results. We tried to cover and discuss all the approaches and steps in details with the results in this analytical report.

1. Dataset summary

This dataset was carefully reviewed before considering for training and predictive analysis and it includes below information:

Number of columns: 12 input features/variables and 1 output/target variable.

Number of rows: 1523

➤ Input variable information:

Index	Columns	Description
1	Company	Laptop manufacturer or the brand of the laptop e.g. "Dell" , "HP", and "Lenovo"
2	TypeName	Type (Notebook, Workstation, Gaming, etc.
3	Screen_size	The size of laptop screen e.g. "Large", "Small" etc.
4	IPS_panel	If the laptop has IPS display or not
5	PPI	Pixel per inch (PPI)
6	Touchscreen	If the laptop has touchscreen or not
7	CPU_brand	the manufacture or brand of CPU e.g. "Intel" , "AMD" etc.
8	Ram	RAM of the laptop (in GB)
9	Memory_type	Type of the memory e.g. "SSD", "HHD" etc.
10	GPU_brand	The manufacturer or brand of GPU e.g. "Intel", "Nvidia" etc.
11	OpSys	Operating system (OS) e.g. "Windows", "MAC" etc.
12	Weight	The weight of the laptop (in KG)

➤ Target/output variable:

Price (13 column): The price of the laptop in thousands (Indian rupees)

We removed the unwanted columns from the raw dataset e.g. column – “Index” and kept the relevant data and which was again reviewed and cleaned in further process.

2. Data Preparation

Data preparation is the first and foremost important step in machine learning. It includes refining, cleaning, and converting raw data into structured and interpretable data for running any machine learning model. We followed various and different steps to perform this process and get the structured dataset for our machine learning model.

Data Cleaning

Once we decided the dataset, it was required to review the dataset thoroughly and make sure to follow few steps like removing duplicates, handling missing values, removing any irrelevant features, giving a better structure to the dataset before we started any coding in Google Colab.

Library import and reading dataset

First, we created Google Colab file for entire coding process and the below code was used to install the `Category_encoders` for target encoding because we already had 12 input variables and with `get_dummies` it was creating up to 45 variables after one hot encoding. Hence, to avoid multiple variable/feature creation and as a feature reduction step we used **target encoder** for this dataset.

```
[ ] pip install category_encoders
```

Afterwards, we imported all the required python modules from library for the model building into Colab notebook and then we imported dataset as CSV file to using `read_CSV` function for data analysis.

```
# Library imports
from pandas import read_csv, Series, DataFrame, get_dummies
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDRegressor
from category_encoders import TargetEncoder
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from plotly import graph_objs, figure_factory
from sklearn import linear_model

# Data reading

data1 = read_csv("/content/drive/MyDrive/Colab/Laptop Price Prediction.csv")
```

Encoding for binary features

Encoding is a very important part of data preparation where the categorical data gets converted to numerical data as the system can only understand the binary language (0-1). We used various codes to analyse the data such as `data.info()` to get information shown in below figure where we can check the features (variables) name and count, data size and data type which helps us with information gain and the features to be encoded.

```
[2] <class 'pandas.core.frame.DataFrame'>
RangeIndex: 1522 entries, 0 to 1521
Data columns (total 13 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Company         1522 non-null   object
1   TypeName        1522 non-null   object
2   Screen_size     1522 non-null   object
3   IPS_panel       1522 non-null   int64
4   PPI             1522 non-null   int64
5   Touchscreen     1522 non-null   int64
6   CPU_brand       1522 non-null   object
7   Ram             1522 non-null   int64
8   Memory_type     1521 non-null   object
9   GPU_brand       1522 non-null   object
10  OpSys           1522 non-null   object
11  Weight          1522 non-null   float64
12  Price           1522 non-null   int64
dtypes: float64(1), int64(5), object(7)
memory usage: 154.7+ KB
```

Once the data analysis was done. There were 2 features we could find for encoding from categorical values of the feature to numerical values through **.map** code which is used for binary values (0-1-2).

```
# Encoding using .map for binary features
data1['Screen_size'] = data1['Screen_size'].map({'small':2,'medium':1,'large':0})
data1['CPU_brand'] = data1['CPU_brand'].map({'Intel':2,'AMD':1,'Samsung':0})
```

Data Splitting

Feature splitting is mainly important for model's training set to get it trained on only independent variables (Input) and hide the testing set (target variable) from the model during its training. We separated the input variables from the target variable, and it was stored in **X_F** called **Features** which was used for further model's training. We split the data into 2nd variable (target variable) as **Y** called **Label** which was later used for models testing and evaluation.

```
#Feature Selection/splitting
X_F = data1.drop(['Price'], axis = 1) # Features
Y = data1['Price'] # Label
print(X_F.shape)
print(Y.shape)
```

Feature reduction and Target encoding

Feature Reduction is one of the critical steps in machine learning, but it can be very useful for big dataset or in case of datasets with irrelevant features. It helps in finding the irrelevant features and making the dataset smaller for avoiding overfitting which consequences to better performance, less training time, and better interpretability. We used 3 feature reduction steps for this dataset and avoided PCA(data wasn't continuous) and other feature reduction steps as the below 3 steps were enough to get a good model.

I. Manual drop

Before the data was imported to Google Colab, during the data analysis and cleaning we reduced the irrelevant features (e.g. "index column", "colour" etc.) manually in order to make sure the uploaded file has relevant features only. It's important to be careful while dropping any feature manually because we may lose a highly correlated feature for the good model.

II. Target encoding

Usually, we use **get_dummies** for one hot encoding for the columns with many categorical attributes. We couldn't use get dummies as it creates so many columns(features) for each attribute and for this dataset with 12 input features, get dummies was creating up to 45 features after encoding which could make the data complex, considering everything we used "**Target encoding**" for the features with more 3 attributes. Target encoding take the average for all the categorical attributes of a feature(column) and encodes it to one feature with average value(float). Moreover, we had to use data splitting before target encoding but in case of get dummies, data splitting is done after one hot encoding.

```
#Target Encoding
Target_Encode = TargetEncoder(cols=['Company', 'TypeName', 'Memory_type', 'GPU_brand', 'OpSys']).fit(X_F,Y)
X = Target_Encode.transform(X_F)
print(X.info())
X.head()
```

III. Heatmap

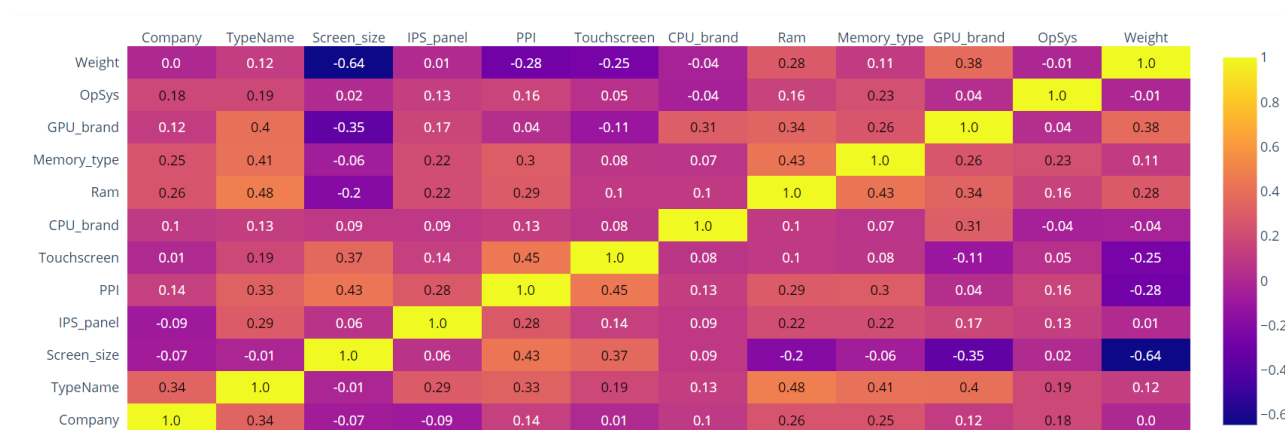
Heatmap is one of the best techniques to find out the **correlation** and **causation** between the 2 features. It's a data visualization technique which helps you to see the most correlated featured with higher % to analyse and decide if one of them can be dropped or not. It's important to keep "causation" in mind while dropping any feature because it can show high correlation, but we need to decide based on our understanding if it has high causation.

Examples:

- **Garage area** and **Number of cars** can be 2 correlated features with high causation as well. Hence, one of them can be dropped.
- **Basement living area** and **Number of rooms** can be marked as correlated features with no causation. Hence, none of them can be dropped.

```
[ ] #Heatmap
correlation = X.corr()
# print (correlation)
Heatmap = figure_factory.create_annotated_heatmap(correlation.values,list(correlation.columns),list(correlation.columns),correlation.round(2).values,
Heatmap.show()
```

We used figure_factory to create the heatmap for our dataset and as shown in below figure there is not high correlation between any of these features. Therefore, we didn't remove any of these features.



IV. Important features

We didn't use important features for feature reduction under Random Forest Regressor because first it's the weakest method among all of them and second, we build the regressor with important features and concluded that the results from Random Forest Regressor with and without important features gave the same score (r2). Hence, Random Forest regressor kept as a model with good result and we deleted the 2nd regressor with important features.


```
# Selecting features with higher significance and redefining feature set
data = X[['Ram', 'TypeName', 'Weight', 'PPI', 'Memory_type', 'Company']]
RF_Regressor2 = RandomForestRegressor(criterion='squared_error', max_features='sqrt', random_state=1)
no_Trees = {'n_estimators': [350, 400, 450, 500, 550, 600]}
grid_search6 = GridSearchCV(estimator=RF_Regressor2, param_grid=no_Trees, scoring='r2', cv=5)
grid_search6.fit(X_, Y)

best_parameters = grid_search6.best_params_
print("Best parameters: ", best_parameters)
best_result = grid_search6.best_score_
print("r2: ", best_result)
modified_r2 = 1-(1-best_result)*((4/5-1)/5*r-1)/((4/5-1)/5*r-c-1)
print("modified_r2: ", modified_r2)
```

```
Best parameters: {'n_estimators': 450}
r2: 0.8482672289033225
modified_r2: 0.8729125084534055
```

Figure: RF Regressor with Imp features (Deleted this regressor from the coding.)

Data Scaling/Transformation

After data splitting, we used StandardScaler function to scale the data (only X_ training set). This step makes sure that all the features have a similar scale. First it calculates the mean and standard deviation and then the fit_transform() applies the mean and standard deviation to the training data to standardize it. It's good for better performance and stability in machine learning models.

```
# Data Scaling/Transformation
X_ = StandardScaler().fit_transform(X)
DataFrame(X_)
```

3. Regularization Techniques

During training of the model, if the model can memorize the noise and doesn't learn from the data, in that case it cannot be good at predicting the unseen data. Regularization Techniques can be used to avoid the overfitting and improves the performance of the model by adding penalty. It penalizes the model for being complex and having big β_j . Sometimes it's hard to rely on feature reduction because we remove any feature without complete confidence it might impact the performance of the model and that's why regularization should be used as it's not completely dependent on the features and it controls the overfitting. We have used the below 3 regularization techniques to avoid overfitting in regression model.

LASSO - L1 Regularization

L1 (Lasso) doesn't allow models to be complex by adding a penalty equal to the absolute α value of size of coefficients. Lasso regularization technique can lead to sparser model which means some of the coefficients become zero and as a result some of the features are avoided which helps with feature selection. It's useful in case of clean data and to find the key features. It is described by the below equation.

$$L = \frac{1}{2m} \left[\sum_{i=1}^m (f(x^{(i)}) - y^{(i)})^2 + \alpha \sum_{j=1}^n |\beta_j| \right]$$

RIDGE – L2 regularization

L2 (Ridge) is another regularization technique in machine learning to reduce overfitting in regression model. It doesn't allow the model to become too complex or memorizing the patterns which leads to bad prediction on new data. Like L1 the cost function compares the predicted values with actual values.

L2 squares the coefficients to maintain penalty always positive, shrinking all coefficients towards zero by penalizing the positive and negative coefficients but it doesn't always make them to zero completely. It helps with simpler model but doesn't go with feature selection (meaning it retains all the features). L2 is calculated with below equation.

$$L = \frac{1}{2m} \left[\sum_{i=1}^m (f(x^{(i)}) - y^{(i)})^2 + \alpha \sum_{j=1}^n (\beta_j)^2 \right]$$

Elastic Net Regularization

Elastic net is a combination of the two regularized variants of linear regression ridge and lasso. Ridge utilizes an L2 penalty and lasso uses an L1 penalty. The elastic net procedure provides the inclusion of "n" number of variables until saturation. If the variables are highly correlated groups, lasso tends to choose one variable from such groups and ignore the rest entirely. We use 4 parameters in Elastic net to get accurate model and avoid overfitting.

To summarize, the elastic net technique is most appropriate where the dimensional data is greater than the number of samples used. Also, Groupings and variable selection are the key roles of the elastic net technique. Below is the equation how Elastic net is calculated.

$$L = \frac{1}{2m} \left[\sum_{i=1}^m (f(x^{(i)}) - y^{(i)})^2 + \alpha \left\{ (1 - \lambda) \sum_{j=1}^n (\beta_j)^2 + \lambda \sum_{j=1}^n |\beta_j| \right\} \right]$$

4. Linear Regression (LR) and Impact of Regularizations

The linear regression is the common technique of a machine learning that is used to model the relationship between the dependent variable (what is going to be predicted) and one or more independent variables (the features that influence the prediction). In practice, it fits a simple straight line to a set of data points to make the predictions.

The main problem in linear regression is overfitting. The model overfits the training data if it pays too much attention to getting a good fit to both the noise and the random variations which results in a model with low performance on unseen data. This is where we use **regularization** to avoid overfitting.

We have 12 input features for this dataset after scaling process and we have used 3 regularizations to analyse impact of each regularization (L1,L2 and Elastic Net) on linear regressor.

Linear Regression (LR) with “None” Penalty

We run the linear regressor with penalty “None” which mean the regularization was not used in this model hence, It can lead to overfitting.

- We have only used the hyperparameters **eta0** (learning rate) and **max_iter** (Number of iteration for training data) and tunned them with various attempts in running the model.
- We used GridsearchCV for this regressor while using scoring = 'r2' (proportion of variance explained) and **CV** (cross validation or 5 -folds to find the best train and test set).
- **Gridsearch1.fit** is used for training, testing, evaluation and ranking of the model.
- At last, we used modified r2 (limitation of r² by penalizing the model for complexity).

```
# Linear Regression (LR) with 'None' penalty

LinearRegression1 = SGDRegressor(random_state = 1, penalty = None) # Building Regressor
Hparameter1 = {'eta0': [.00001, .0001, .001, .01, 1], 'max_iter': [500, 800, 1100, 1400, 1700]}
grid_search1 = GridSearchCV(estimator=LinearRegression1, param_grid=Hparameter1, scoring='r2', cv=5)
grid_search1.fit(X_, Y) #Traning, Testing, Evaluation, Ranking

best_parameters = grid_search1.best_params_
print("Best parameters: ", best_parameters)
best_result = grid_search1.best_score_
print("Best result: ", best_result)
best_model = grid_search1.best_estimator_
print("Intercept  $\beta_0$ : ", best_model.intercept_)
print(DataFrame(zip(X.columns, best_model.coef_), columns=['Features', 'Coefficients']).sort_values(by=['Coefficients'], ascending=False))

# Modified mean square error
r, c = X_.shape
modified_r2 = 1 - (1 - best_result) * ((4/5 - 1) / 5 * r - 1) / ((4/5 - 1) / 5 * r - c - 1) # 4/5*r is number of rows in training set, c is number of columns
print("modified_r2: ", modified_r2)
```

Model's Result	
Best parameter - eta0	0.0001
Best parameter – max_iter	1100
r ²	0.66571315
Intercept [β_0]	60194.87537
Modified r2	0.72000988

Linear Regression (LR) with L1 Regularization

The below regressor was run with regularization L1 (LASSO) as explained previously L1 adds a penalty equal to the absolute α value of size of coefficients to avoid overfitting and its simplicity (feature selection). In this regressor we used 3 parameters unlike above regressor with 2 parameters. We added α (alpha controls the overall strength of the penalty term) to the parameter list and L1 in SGDRegressor for regularization.

```
[29] # Linear Regression (LR) using 'L1' Regularization

LinearRegression2 = linear_model.SGDRegressor(random_state = 1, penalty = 'l1') # Building Regressor
Hparameter2 = {'eta0': [.00001, .0001, .001, .01, 1], 'max_iter': [500, 800, 1100, 1400, 1700], 'alpha': [.0001, .001, .01, .1, 1]}
grid_search2 = GridSearchCV(estimator=LinearRegression2, param_grid=Hparameter2, scoring='r2', cv=5)
grid_search2.fit(X_, Y) #Traning, Testing, Evaluation, Ranking

best_parameters = grid_search2.best_params_
print("Best parameters: ", best_parameters)
best_result = grid_search2.best_score_
print("Best result: ", best_result)
best_model = grid_search2.best_estimator_
print("Intercept  $\beta_0$ : ", best_model.intercept_)
print(DataFrame(zip(X.columns, best_model.coef_), columns=['Features', 'Coefficients']).sort_values(by=['Coefficients'], ascending=False))

#Modified mean square error
r, c = X_.shape
modified_r2 = 1 - (1 - best_result) * ((4/5 - 1) / 5 * r - 1) / ((4/5 - 1) / 5 * r - c - 1)
print("modified_r2: ", modified_r2)
```

Impacts:

- L1 regularization use **feature selection** by pushing coefficients of irrelevant features towards zero which helps to keep only important features and reducing complexity. We can examine in the coefficients figures below that there is a slight difference in the size of coefficients of L1 from the ones used with None penalty.
- Lasso leads to sparse model by making some **coefficients exactly to zero**. This sparsity is good when it's a complex and huge dataset. In this case, it didn't remove any feature but there was slightly difference in all the coefficients values.
- Lasso penalizes the large sizes of coefficients while pushing them towards zero which is helpful in overfitting reduction to make sure the model doesn't rely too much on any specific feature.
- Lasso helps in Increasing Bias, and reducing Variance which is good for unseen data prediction.
- LASSO uses alpha to controls the strength of penalty. The tuning and selecting best value of alpha is important as big alpha value can push more coefficients to zero which may impact the models performance.

```
Intercept  $\beta_0$ : [60194.87527688]
      Features Coefficients
7      Ram    13375.818116
4      PPI     9853.861768
1     TypeName  9842.294001
8   Memory_type  4556.881253
10    OpSys    3671.317621
6   CPU_brand  3029.022590
0    Company   2117.024456
5   Touchscreen 1095.374888
11    Weight    966.200974
3   IPS_panel   227.397466
9   GPU_brand  -362.137113
2  Screen_size -5306.221243
modified_r2:  0.720018670926921
```

Model's Result	
Best parameter - eta0	0.0001
Best parameter – max_iter	1100
Best parameter – α (alpha)	1
r ²	0.66572365
Intercept [β_0]	60194.87
Modified r2	0.72001867

Linear Regression (LR) with L2 Regularization

Ridge (L2) penalizes the squared the size of coefficients, L2 regularization is preferred to be used when feature selection is not the main priority but avoiding overfitting and multicollinearity are concerned. In this regressor we used penalty as L2, and it had used a different number of alpha (0.01) after tuning. We can see the r2 and modified r2 slightly higher in result than L1.

```
# Linear Regression (LR) using 'L2' Regularization

LinearRegression3 = linear_model.SGDRegressor(random_state = 1, penalty = 'l2') # Building Regressor
Hparameter3 = {'eta0': [.00001,.0001, .001, .01,1], 'max_iter':[500,800,1100,1400,1700], 'alpha': [.0001,.001, .01, .1, 1]}
grid_search3 = GridSearchCV(estimator=LinearRegression3, param_grid=Hparameter3, scoring='r2', cv=5)
grid_search3.fit(X,Y) #Traning, Testing, Evaluation, Ranking

best_parameters = grid_search3.best_params_
print("Best parameters: ", best_parameters)
best_result = grid_search3.best_score_
print("Best result: ", best_result)
best_model = grid_search3.best_estimator_
print("Intercept  $\beta_0$ : ", best_model.intercept_)
print(DataFrame(zip(X.columns, best_model.coef_), columns=['Features', 'Coefficients']).sort_values(by=['Coefficients'],ascending=False))

# Modified mean square error
r,c=X_.shape
modified_r2 = 1-(1-best_result)*((4/5-1)/5*r-1)/((4/5-1)/5*r-c-1)
print("modified_r2: ", modified_r2)
```

Impacts:

- L2 penalizes the squared of complete size of coefficients which results in shrinkage to them. Ridge doesn't push coefficients to zero but tries to reduce their impact while making them smaller or closer to zero which helps in avoiding overfitting.
- It can help in **reducing variance** while shirking the coefficients. This may not help with higher accuracy for training set, but it can helps with prediction on unseen data.
- It doesn't perform **feature selection** but shirks the size of coefficients. Sometimes all the features can contribute to good predictions.
- L2 does use tuning of **alpha parameter** like L1 to control the strength of penalty.

Model's Result	
Best parameter - η_0	0.0001
Best parameter – max_iter	1100
Best parameter – α (alpha)	0.01
r ²	0.66580369
Intercept [β_0]	60194.8764
Modified r2	0.72008571

Linear Regression (LR) with Elastic Net Regularization

Elastic Net regularization is a combination of L1 and L2 and it considers their strengths while ignoring their individual limitations. It becomes important to use Elastic net as regularization where both feature selection and dealing with multicollinearity are crucial. We used l1_ratio ' λ ' (**lambda**) (controls the balance between the L1 and L2 penalties.) along with other 3 parameter like eta0 max_iter, and alpha.

```
# Linear Regression using 'Elasticnet' Regularization

LinearRegression4 = SGDRegressor(random_state = 1, penalty = 'elasticnet')
Hparameter4 = {'eta0': [.0001, .001, .01, .1, 1], 'max_iter': [200, 500, 800, 1100, 1400], 'alpha': [.0001, .001, .01, .1, 1], 'l1_ratio': [0, 0.25, 0.5, 0.75, 1]}
grid_search4 = GridSearchCV(estimator=LinearRegression4, param_grid=Hparameter4, scoring='r2', cv=5)
grid_search4.fit(X_, Y) #Traning, Testing, Evaluation, Ranking

best_parameters = grid_search4.best_params_
print("Best parameters: ", best_parameters)
best_result4 = grid_search4.best_score_
print("Best result: ", best_result)
best_model = grid_search4.best_estimator_
print("Intercept  $\beta_0$ : ", best_model.intercept_)
print(DataFrame(zip(X.columns, best_model.coef_), columns=['Features', 'Coefficients']).sort_values(by=['Coefficients'], ascending=False))

# Modified mean square error
r, c=X_.shape
modified_r2 = 1-(1-best_result)*((4/5-1)/5*r-1)/((4/5-1)/5*r-c-1)
print("modified_r2: ", modified_r2)
```

Impacts:

- Elastic net performs both feature selection and shrinkage. Like LASSO, it can push the coefficients to exactly zero and leads to feature reduction at the same time it acts like RIDGE and it can also shrink the coefficients while penalizing to make it small coefficients to avoid overfitting.
- Elastic net balance between LASSO and RIDGE and mitigates the limitation of each individual technique.
- Elastic net manages the trade-off between L1 and L2.
- It useful for high dimensional data (data with more features) and where there is risk of having many correlated features.
- Elastic net involves 2 parameters - alpha (α) and l1_ratio along with Eta0 and max_iter.

Model's Result	
Best parameter - η	0.001
Best parameter – max_iter	50
Best parameter – L1_ratio	0.75
Best parameter – α (alpha)	0.1
r^2	0.665803
Intercept [β_0]	60176.4255
Modified r^2	0.720085

Comparison between Different Regularization Method in LR

This is the comparative table with results of all the regularization. There is slightly difference between the scores of all the parameters for this dataset. All of them have good r^2 and modified r^2 scores but the model with L2 and Elastic net stands out with slightly better score. Since Elastic net is a combination of both L1 and L2 and would be good with unseen data we considered Elastic net as a good regularization for this model in terms of prediction.

The result of the model	No regularization	L1 regularization	L2 regularization	Elastic Net regularization
r^2	0.66571315	0.66572365	0.66580369	0.665803
Modified r^2	0.72000988	0.72001867	0.72008571	0.720085
η	0.0001	0.0001	0.0001	0.001
Intercept β_0	60194.87537	60194.87	60194.8764	60176.4255
max_iter	1100	1100	1100	50
Alpha	Not used	1	0.01	0.1
l1_ratio	Not used	Not used	Not used	0.75

Impact of L2 (C) Regularization on SVR

SVR handles regression problems (predicting continuous values) by finding a separating plane(hyperplane), even for curvy data. In order to reduce errors, SVR focuses on support vectors and limit it to small margin of errors (epsilon-tube) for the prediction However, we need to adjust the parameters (epsilon, kernel, and C regularization) carefully for better prediction.

To avoid overfitting, L2 (C) regularization method which is a technique in our Support Vector Regression (SVR) model was used. The regularization technique will impose discipline on the model

towards a training data influence reduction by penalizing weights values so that the model is not extremely complicated and generalizable.

Impact of L2 (C) regularization on SVR performance and interpretability

Impact on Performance

Higher C:

Advantage: C parameter helps in reducing training error by giving room to the hyperplane for being more flexible in regard to the data points. This has the result of improving the performance on the same training dataset.

Disadvantage: It increases complexity of model and is prone to get overfitting. This model may be overly obsessed with capturing the noise present in training data, too, limiting its effectiveness when applied to unseen data.

Advantage: It limits the complexity of the model by applying a penalty for too complex hyperplanes. This may allow for better generalization and cause less overfitting, thereby leading to also better results on unknown data.

Disadvantage: There is a possibility of overtraining the model with a higher training error. The model may not fit the training data accurately.

Impact on interpretability:

Higher C:

Disadvantage: It leads to a more complex model in view of which the coefficients (the weights of features) are larger in size in absolute terms.

Lower C:

Advantage: It tends to generate a smaller model which requires fewer inputs and has smaller coefficient values. With this, it would be possible to track by which feature the predictions are influenced and improve the model for better interpretability.

Result and Impact Analysis of L2 Regularization on SVR Model

We build the **SVR model without c parameter** and the r2 and modified r2 score is so low, making it a bad model for prediction on unseen data.

```
Best parameters: {'epsilon': 500, 'kernel': 'linear'}
Best result: 0.03600840798570184
modified_r2: 0.027236541366424838
```

Then, we used model with C parameter and grid search gave us these results and they showed that with C=1000000, epsilon=10000 the SVR model achieved the second best regularization parameter balance of bias and variance. Another result of this study is that the R-squared value was estimated at almost 79%, which indicates a high explanatory power of the model over the observed variance. Furthermore, the modified R-squared value not only looked into the number of factors but also showed the strong performance i.e. the validity of L2 regularization strategy increased the generalizability and robustness of SVR model.

5. Comparison with Random Forest Regression

After all the analysis and practices, we have given a comparative table below for better understanding of the results and performances of each regressor for machine learning model.

Regressor	r2	Modified r2	Hyperparameters
Random Forest Regression	0.848267	0.87291251	n_estimators': 450
Regularized LR (Elastic Net)	0.665804	0.72008572	alpha': 0.1, 'eta0': 0.001, 'l1_ratio': 0.75, 'max_iter': 50
Support Vector Regression (SVR)	0.790542	0.78863626	'C': 1000000, 'epsilon': 10000, 'kernel': 'rbf'

Random Forest Regression

We have received high r2 (0.82) and modified r2 (0.87) score for Random Forest Regressor which makes it a good model. We did not keep the random forest regressor with important features for smaller model because we could notice that the result for both the regressor was same for this dataset and as we already used modified r2 to improve the result.

- **Parameters** : ['n_estimators': 450]
- **r2** : 0.8482
- **modified r2** : 0.8729

Linear Regression with (Elastic Net)

Linear Regression with Elastic net regularization may have resulted as the best method among all 3 regularization methods but its not the best regressor if we compare the modified r2 with Support vector regressor and Random Forest Regressor. However, it's a good model with r2 (0.66) and modified r2 score (0.72). It uses 4 parameters to balance the accurate predictability and complexity. Where l1_ratio helps with model's interpretability and other parameters like alpha helps in reducing the overfitting while impacting the coefficients during the process.

- **Parameters** : ['alpha': 0.1, 'eta0': 0.001, 'l1_ratio': 0.75, 'max_iter': 50]
- **r2** : 0.665
- **modified r2** : 0.720
- **Intercept (β_0)** : 60176.4255

Support Vector Regression (SVR)

Support Vector regressor in this case is second most accurate model with modified r2 (0.78) and Support vector regressor helps to reduce the margin and tries to get all the vector within the hyperplane line (tube) with the help of Epsilon (margin of tolerance) and uses C as regularization parameter. Here we tunned the C and the best number we received was 1000000 and Epsilon 10000. According to the below results SVC is the second-best model after Random forest regressor.

- **Parameters** : ['C': 1000000, 'epsilon': 10000, 'kernel': 'rbf']
- **r2** : 0.7905
- **modified r2** : 0.7886

6. Prediction -New data

Finally, we reached to the step where the prediction can be done for unseen or new data. We decided to use of the best model with high modified r2 score but sometimes, it can be time consuming to run

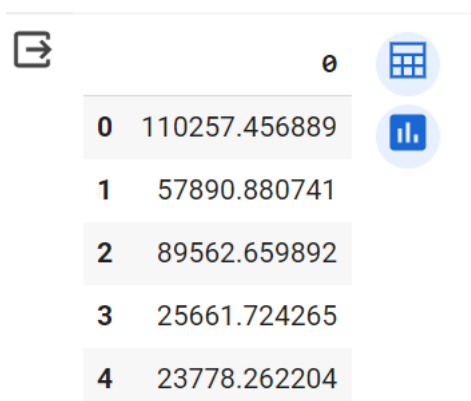
the model before making predictions. We have used **model.pkl** from joblib to save time in running of the models instead it calls the trained model automatically.

```
import joblib
print(best_model)
joblib.dump(best_model, "model.pkl")
```

Once the model.pkl is executed we can use the below code for the prediction of price for laptops using the below code.

```
price=My_model.predict(X_[0:5])
price
DataFrame(price)
```

We considered Random Forest regressor for the price prediction as this has the best modified r2 score among all the regression models. Hence, the final prediction for this dataset is shared through below figure.



	0
0	110257.456889
1	57890.880741
2	89562.659892
3	25661.724265
4	23778.262204

Conclusion

In conclusion, we started with data preparation considering all the important parameters related to data cleaning, feature reduction and training the data with all the techniques and methods. For example, we analysed why is important to use Target encoding than get_dummies or to avoid PCA for this dataset (as it did not have continuous values to be encoded).

We used Linear Regression with and without 3 regularization techniques (L1, L2 and Elastic net) and their impacts to find out the best performing method, followed by Random Forest Regressor and SVM. We also had a competitive analysis of all the regressors and concluded that Random Forest

Regressor is the best regressor for this dataset with high r^2 score and it can be used in real life for laptop price prediction or for similar analytical uses. Though, the other two regressor also have good r^2 score and can be used for prediction.

The analysis further taught us the importance of the features selection and reduction which impacts the overall performance of any model and to get the accurate model one should be mindful of each step taken to build the model.

Appendix – Team contribution

Coding

Manisha – 20024385

- Dataset review and selection
- Data analysis and preprocessing
- Prediction for new data

Abhishek Sen - 20012367

- Linear Regression with penalty “None”
- Linear Regression with LASSO (L1)
- Linear Regression with RIDGE (L2)
- Linear Regression with Elastic Net

Kashmira Ghag - 20023193

- Random Forest Regressor
- SVM without C parameter
- SVM with C Parameter

Report

Manisha – 20024385

- Introduction and Data summary
- Data preparation
- Linear Regression with penalty “None” and regularization impact
- Linear Regression with LASSO (L1) and regularization impact
- Linear Regression with RIDGE (L2) and regularization impact
- Linear Regression with Elastic Net and regularization impact

Abhishek Sen - 20012367

- Support vector regressor (SVR)
- Impact of L2 regularization on SVR
- Comparison with Random Forest Regression
- Conclusion

Kashmira Ghag - 20023193

- Regularisation Techniques Overview
- L1 (LASSO) regularization
- L2 (RIDGE) regularisation
- elastic net regularisation
- Prediction using new Data