Machine Learning Project - Laptop Price Prediction

Arijit Samal, Hieu Nguyen Minh (BDMA)

May 28, 2024

1 Overview

We decide to do a regression problem which is laptop price prediction. In this task, we are given information related to laptops from which we will try to predict their price. We choose a dataset provided from Kaggle[1].

This dataset has 1303 instances with 12 features. The dataset dataframe is shown in Figure 1.

	Unnamed: 0	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price
0	0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8GB	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37kg	71378.6832
1	1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8GB	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34kg	47895.5232
2	2	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8GB	256GB SSD	Intel HD Graphics 620	No OS	1.86kg	30636.0000
3	3	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16GB	512GB SSD	AMD Radeon Pro 455	macOS	1.83kg	135195.3360
4	4	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8GB	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37kg	96095.8080
...	...	...	...	...	...	...	...	...	...	...	...	...
1298	1298	Lenovo	2 in 1 Convertible	14.0	IPS Panel Full HD / Touchscreen 1920x1080	Intel Core i7 6500U 2.5GHz	4GB	128GB SSD	Intel HD Graphics 520	Windows 10	1.8kg	33992.6400
1299	1299	Lenovo	2 in 1 Convertible	13.3	IPS Panel Quad HD+ / Touchscreen 3200x1800	Intel Core i7 6500U 2.5GHz	16GB	512GB SSD	Intel HD Graphics 520	Windows 10	1.3kg	79866.7200
1300	1300	Lenovo	Notebook	14.0	1366x768	Intel Celeron Dual Core N3050 1.6GHz	2GB	64GB Flash Storage	Intel HD Graphics	Windows 10	1.5kg	12201.1200
1301	1301	HP	Notebook	15.6	1366x768	Intel Core i7 6500U 2.5GHz	6GB	1TB HDD	AMD Radeon R5 M330	Windows 10	2.19kg	40705.9200
1302	1302	Asus	Notebook	15.6	1366x768	Intel Celeron Dual Core N3050 1.6GHz	4GB	500GB HDD	Intel HD Graphics	Windows 10	2.2kg	19660.3200

Figure 1: Dataframe of the dataset.

2 Data Exploration

After loading the dataset, we check and confirm that it has no duplicate instances nor null values. The dataframe has an redudant column "Unnamed: 0" which is originally the index column, so we drop it. Then we make the following modifications to columns:

- Create four new features "Touchscreen" (boolean), "IPS" (boolean), "PPI" (float), and "Retina" (boolean) from the feature "ScreenResolution", then drop that column.
- Create two new features "Cpu_name" (string) and "Cpu_clock_speed" (float) from the feature "Cpu", then drop that column. We also observe that there is only one laptop with Samsung CPU which does not help so we remove it.
- Remove the string "GB" in the column "Ram" and set the value type to integer.
- Create four new features "HDD" (boolean), "SSD" (boolean), "Hybrid" (boolean), and "Flash_storage" (boolean) from the feature "Memory", then drop that column.
- Create a new feature "Gpu_brand" from the feature "Gpu", then drop that column.
- From the feature "OpSys", we keep "Windows" and "Mac", while grouping others (Linux, Android, Chrome Os, No OS) as one group. Then we drop the column "OpSys".
- Remove the string "kg" in the column "Weight" and set the value type to float.

	Company	TypeName	Inches	Ram	Weight	Price	Touchscreen	IPS	PPI	Retina	Cpu_name	Cpu_clock_speed	HDD	SSD	Hybrid	Flash_storage	Gpu_brand	OS
0	Apple	Ultrabook	13.3	8	1.4	71378.6832	0	1	227	1	Intel Core i5	2.3	0	128	0	0	Intel	Mac
1	Apple	Ultrabook	13.3	8	1.3	47895.5232	0	0	128	0	Intel Core i5	1.8	0	0	0	128	Intel	Mac
2	HP	Notebook	15.6	8	1.9	30636.0000	0	0	141	0	Intel Core i5	2.5	0	256	0	0	Intel	Linux/Android/Chrome OS/No OS
3	Apple	Ultrabook	15.4	16	1.8	135195.3360	0	1	221	1	Intel Core i7	2.7	0	512	0	0	AMD	Mac
4	Apple	Ultrabook	13.3	8	1.4	96095.8080	0	1	227	1	Intel Core i5	3.1	0	256	0	0	Intel	Mac
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1298	Lenovo	2 in 1 Convertible	14.0	4	1.8	33992.6400	1	1	157	0	Intel Core i7	2.5	0	128	0	0	Intel	Windows
1299	Lenovo	2 in 1 Convertible	13.3	16	1.3	79866.7200	1	1	276	0	Intel Core i7	2.5	0	512	0	0	Intel	Windows
1300	Lenovo	Notebook	14.0	2	1.5	12201.1200	0	0	112	0	other Intel processors	1.6	0	0	0	64	Intel	Windows
1301	HP	Notebook	15.6	6	2.2	40705.9200	0	0	100	0	Intel Core i7	2.5	1024	0	0	0	AMD	Windows
1302	Asus	Notebook	15.6	4	2.2	19660.3200	0	0	100	0	other Intel processors	1.6	500	0	0	0	Intel	Windows

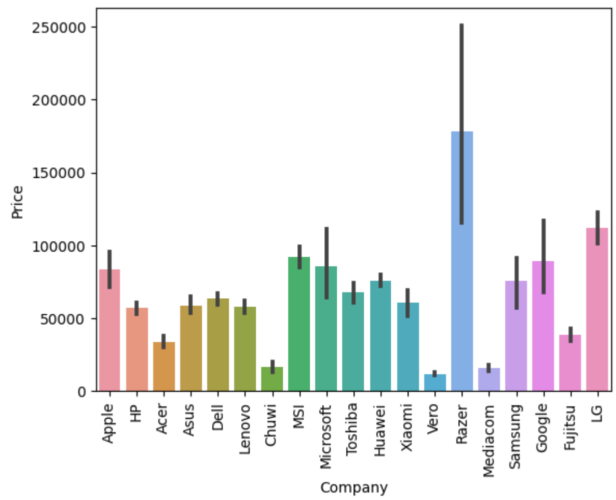
Figure 2: Dataframe of the dataset after preprocessing.

Now the new dataframe has 1302 instances with 18 features as shown in Figure 2. Next, we draw bar charts to see how each feature is related to the price. We have the following observations:

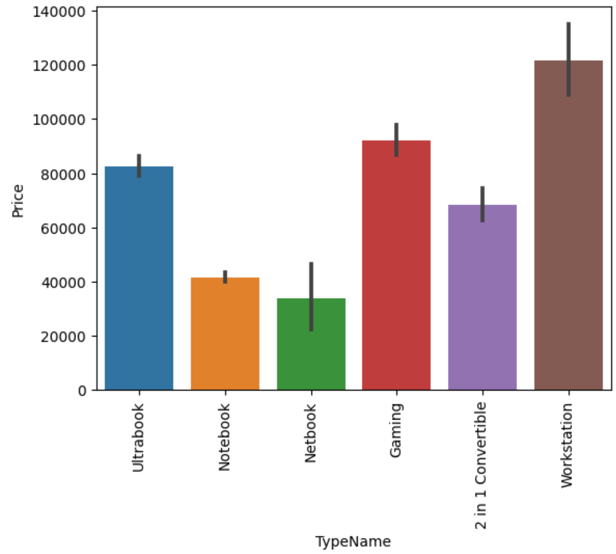
- Laptop prices among companies do not vary much, except "Chuwi", "Vero" and "Mediacom" where the price is very low, and "Razer" where the price is very high.
- High-end laptop types (Ultrabook, Gaming, 2in1 Convertible, Workstation) have higher price than basic ones (Notebook, Netbook).
- Laptops with larger screen tend to have higher price than ones with smaller screen.
- Laptops with higher RAM capacity have higher price than ones with smaller RAM capacity.
- Heavier laptops tend to have higher price.
- Apart from some special cases, high-resolution laptops (higher PPI) tend to have higher price than low-resolution ones.
- Intel-CPU laptops tend to have higher price than AMD-and-other-CPU laptops.
- Laptops with faster CPU tend to have higher price thand ones with slower CPU.
- Laptops with higher SSD/HDD/Hybrid/Flash_storage capacity tend to have higher price than ones with lower disk capacity.
- NVIDIA-GPU laptops have higher price thand Intel and AMD ones.
- Mac and Windows laptops have higher price and ones with other operating systems.
- Laptops with touch screen, IPS screen and Retina screen have higher price than ones without these screen types.

To get insight into these relations, we make a correlation analysis shown in Figure 6. More or lesss all features contribute to the laptop price.

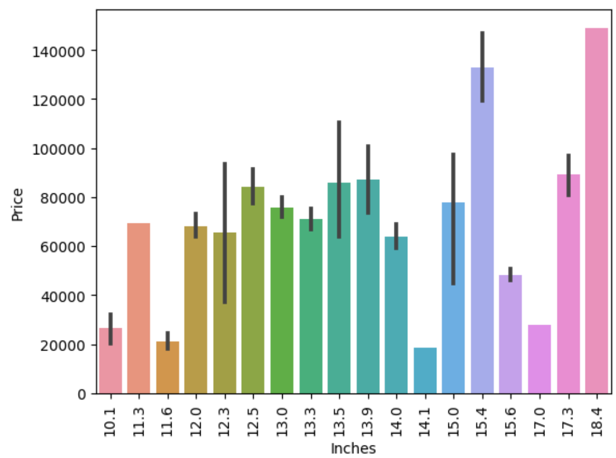
[1] https://www.kaggle.com/datasets/ayush12nagar/laptop-data-price-predition/data



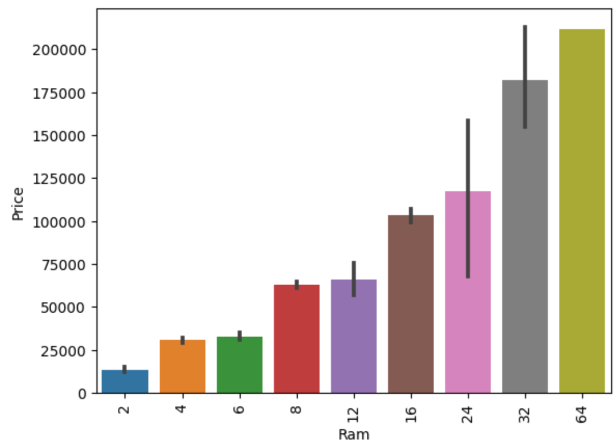
(a) Company-Price.



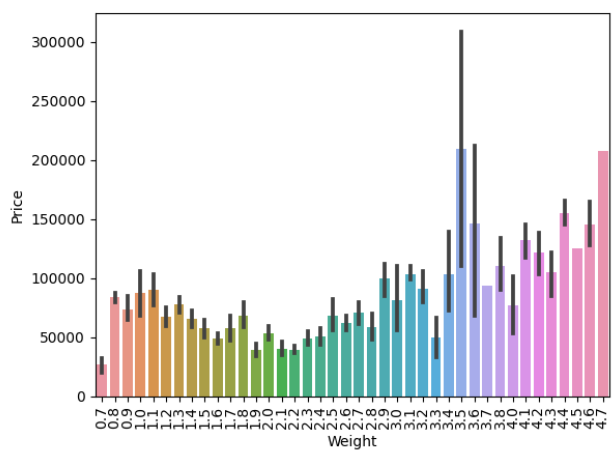
(b) TypeName-Price.



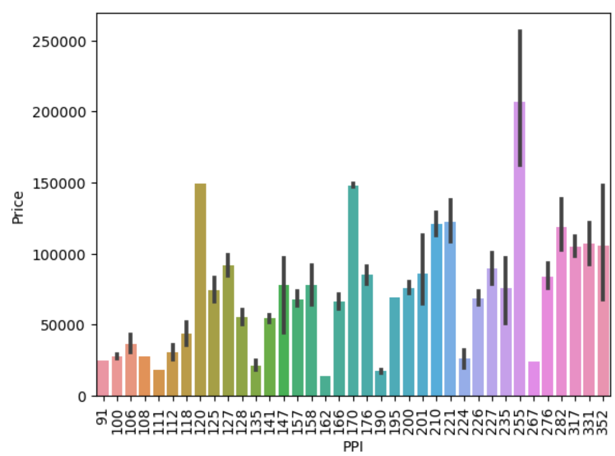
(c) Inches-Price.



(d) Ram-Price.

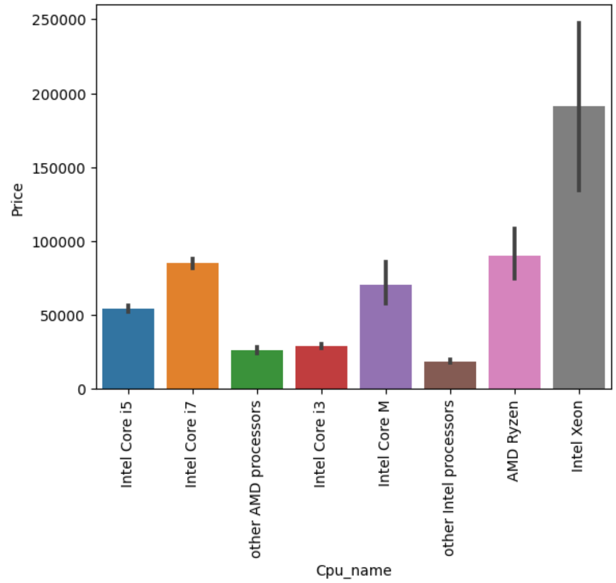


(e) Weight-Price.

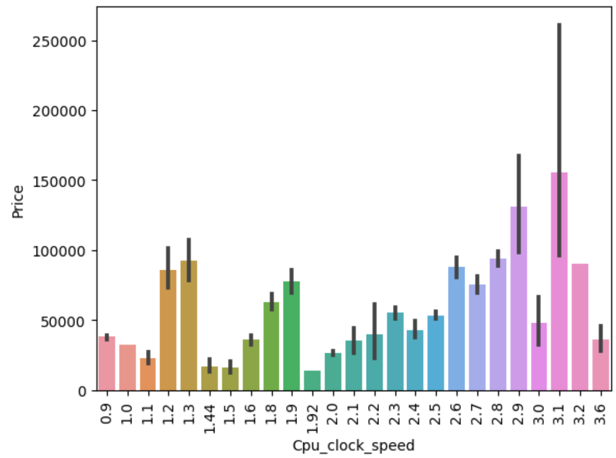


(f) PPI-Price.

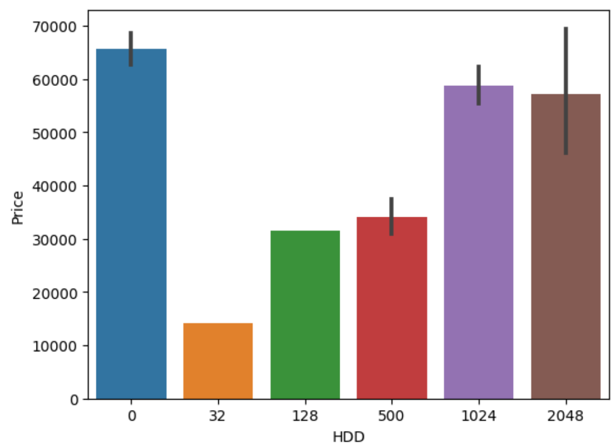
Figure 3: Feature-Price relations (part I).



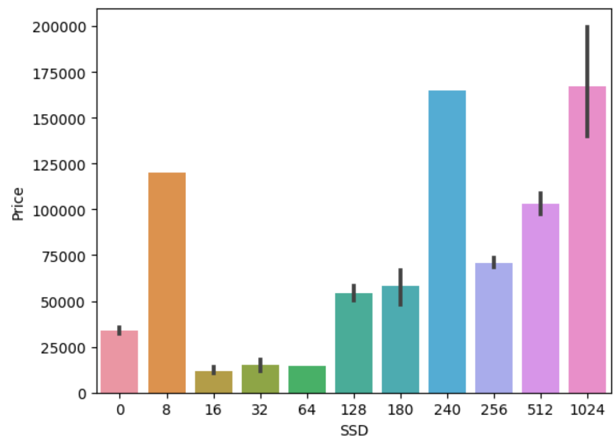
(a) Cpu_name-Price.



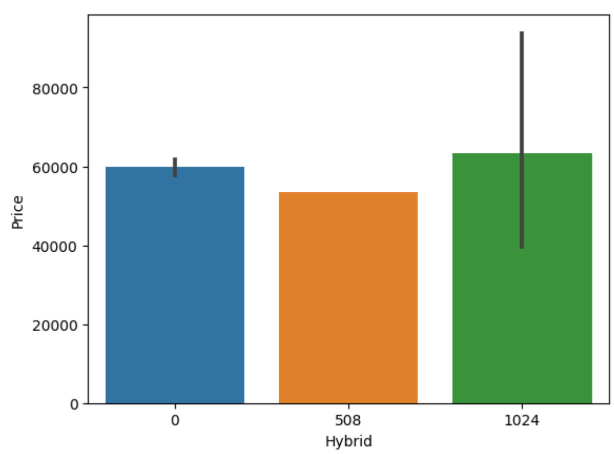
(b) Cpu_clock_speed-Price.



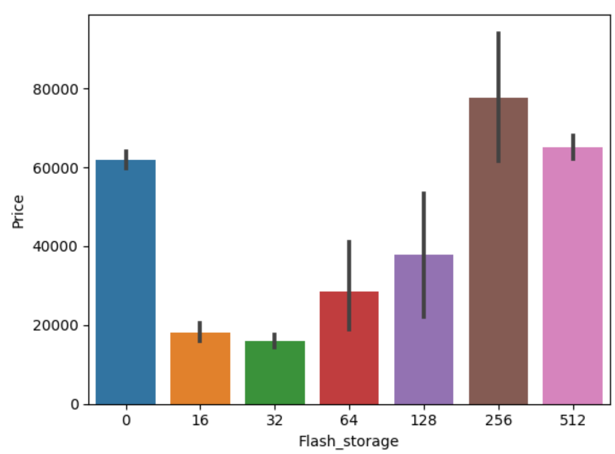
(c) HDD-Price.



(d) SSD-Price.

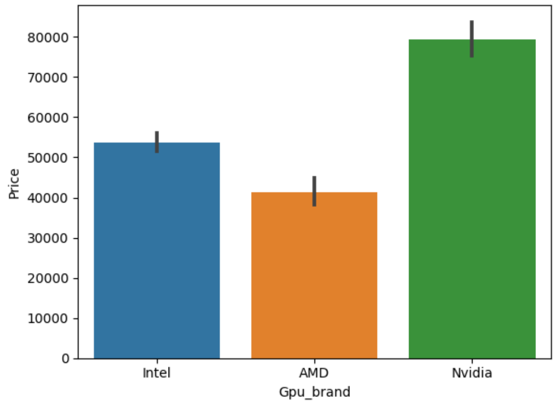


(e) Hybrid-Price

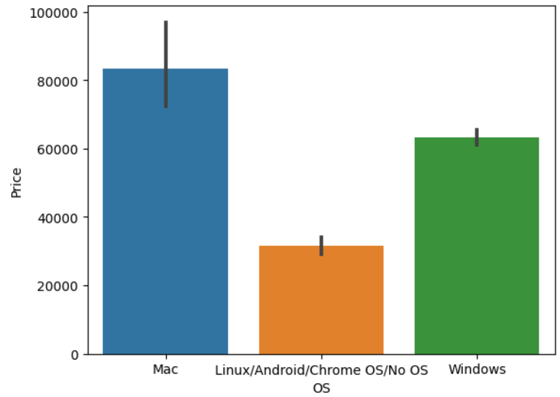


(f) Flash_storage-Price.

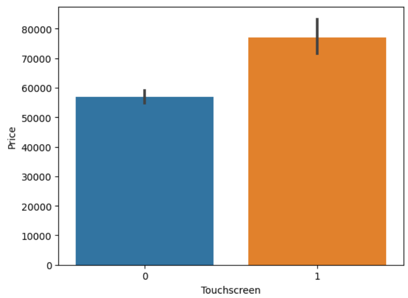
Figure 4: Feature-Price relations (part II).



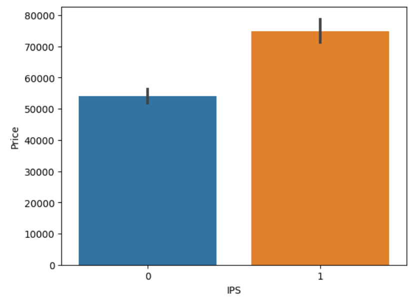
(a) Gpu_brand-Price.



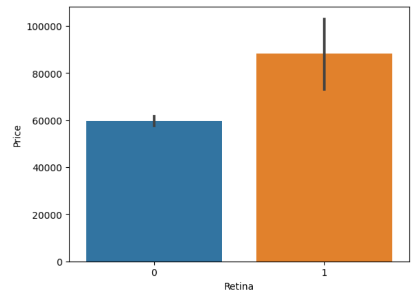
(b) OS-Price.



(c) Touchscreen-Price.



(d) IPS-Price.



(e) Retina-Price.

Figure 5: Feature-Price relations (part III).

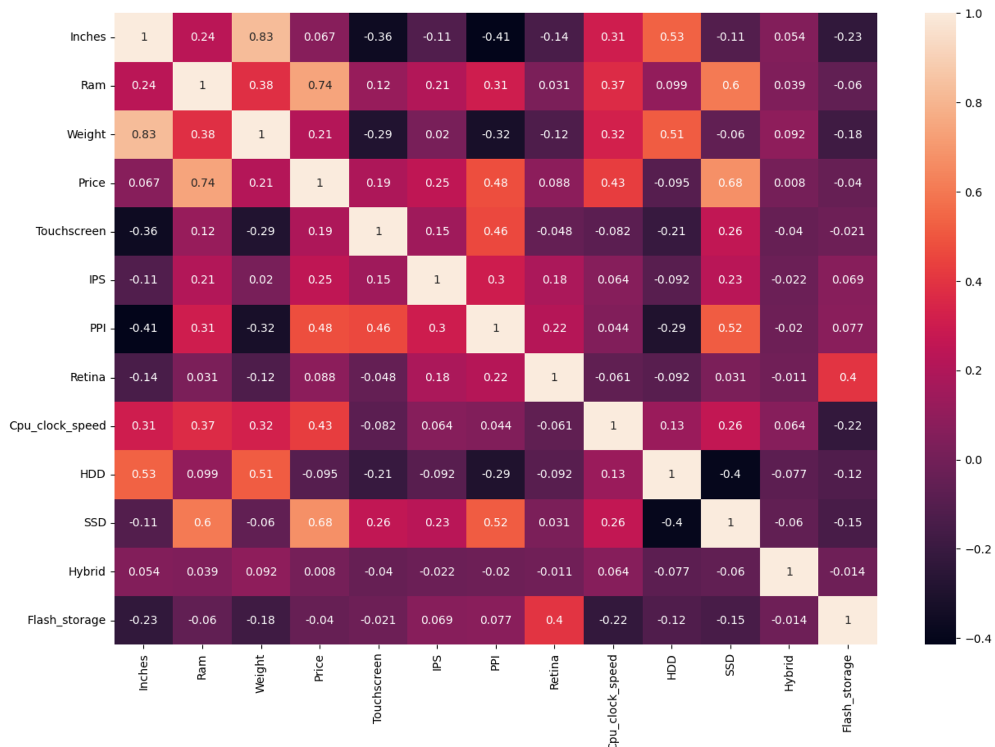


Figure 6: Correlation analysis.

3 Modeling Methods

Firstly we create input data and corresponding labels from the dataframe. For features whose type is string (Company, TypeName, Cpu_name, Gpu_brand, OS) or categorical (HDD, SSD, Hybrid, Flast_storage), we conert them to binary features with the function `pandas.get_dummies()`. For the price, we take logarithm of its value to make it easy to predict. This results in a training matrix X with shape $(1302, 82)$ and a label column matrix of length 1302. After that, we split the data into a training set (80%) and a test set (20%).

Next, we make a regression pipeline. We choose nine regression models from scikit-learn[2] library:

1. SVR (Support Vector machine for Regression)
2. LinearRegression
3. Ridge
4. Lasso
5. ElasticNet
6. KNeighborsRegressor
7. DecisionTreeRegressor
8. GradientBoostingRegressor
9. XGBRegressor

We use the MSE (mean squared error)[3] and R^2 (coefficient of determination)[4] as the metrics. In the pipeline, we include two hyperparameter tuning methods: RandomizedSearchCV[5] and GridSearchCV[6]. To this end, we need to set the search space where the hyperparameters are. Below is a code snippet of our pipeline.

```
def regression_pipeline(model, X_train, y_train, X_test, y_test, exhaustive=False,
     tuning=False):
    if(model==svr):
        grid_params={'kernel':['linear', 'poly', 'rbf', 'sigmoid'],
                     'C':[0.0001,0.001,0.01,0.1,1,10]}
    elif(model==rlr):
        grid_params={'alpha':[0.0001,0.001,0.01,0.1,1,10]}
    elif(model==llr):
        grid_params={'alpha':[0.0001,0.001,0.01,0.1,1,10]}
    elif(model==elr):
        grid_params={'alpha':[0.0001,0.001,0.01,0.1,1,10],
                     'l1_ratio':[0.25,0.50,0.75]}
    elif(model==knr):
        grid_params={'n_neighbors':np.arange(1,100,2),
                     'leaf_size':np.arange(2,501)}
    elif(model==dtr):
```

[2]https://scikit-learn.org/stable/
[3]https://en.wikipedia.org/wiki/Mean_squared_error
[4]https://en.wikipedia.org/wiki/Coefficient_of_determination
[5]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
[6]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

```
        grid_params={'criterion': ['squared_error','friedman_mse','absolute_error','poisson'],
                     'splitter': ['best', 'random'],
                     'max_depth': np.arange(1, 21),
                     'min_samples_split': np.arange(2, 21),
                     'min_samples_leaf': np.arange(1, 11),
                     'max_features': ['sqrt', 'log2', None],
                     'max_leaf_nodes': np.arange(2, 21),
                     'min_impurity_decrease': [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]}
    elif(model==gbr):
        grid_params={'loss': ['squared_error', 'absolute_error', 'huber', 'quantile'],
                     'learning_rate':[0.0001,0.001,0.01,0.1],
                     'n_estimators':np.arange(50,501,50),
                     'max_depth':np.arange(3,11,1),
                     'min_samples_split': np.arange(2,20),
                     'min_samples_leaf':np.arange(1,11),
                     'subsample':np.arange(0.1,1,0.2),
                     'max_features':np.arange(0.1,1,0.2),
                     'alpha':[0.0001,0.001,0.01,0.1,0.2,0.5,0.9]}
    elif(model==xgbr):
        if not exhaustive:
            grid_params ={'n_estimators': [100, 200, 300, 500],
                        'learning_rate': [0.001, 0.01, 0.05, 0.1],
                        'max_depth': [3, 5, 7, 9, 11, 15, 19],
                        'min_child_weight': [1, 5, 10],
                        'subsample': [0.5, 0.7, 0.9],
                        'colsample_bytree': [0.5, 0.7, 0.9],
                        'gamma': [0, 0.1, 0.2],
                        'reg_alpha': [0, 0.1, 0.5, 1],
                        'reg_lambda': [0, 0.1, 0.5, 1],
                        'objective': ['reg:squarederror', 'reg:squaredlogerror']}
        else:
            grid_params ={'n_estimators': [100, 200, 300, 500],
                        'learning_rate': [0.01, 0.05, 0.1],
                        'max_depth': [3, 5, 7, 9],
                        'subsample': [0.5, 0.7, 0.9],
                        'reg_alpha': [0.1, 0.5, 1],
                        'reg_lambda': [0.1, 0.5, 1],
                        'objective': ['reg:squarederror', 'reg:squaredlogerror']}
    else:
        grid_params={}

    if tuning==True and exhaustive==True:
        model= GridSearchCV(model, grid_params, scoring='r2', verbose=10, cv=5, n_jobs=-1)
    elif tuning==True and exhaustive==False:
        model= RandomizedSearchCV(model, grid_params, scoring='r2', verbose=10, cv=5, n_jobs=-1)
    elif tuning==False:
        model= GridSearchCV(model, param_grid={}, scoring='r2', verbose=10, cv=5, n_jobs=-1)

    pipe= Pipeline([('step1', model)])
    pipe.fit(X_train,y_train)
```

Listing 1: Regression pipeline.

4 Results and Discussion

The results of all models and methods are presented in Table 1. From the table, we have some observations:

- Without the hyperparameter tuning, the SVM, Ridge and ElasticNet models do not work well. This is because those models have important parameters like kernel, regularization parameter or mixing parameter that need to be appropriately set. Other models' performances are similar, among which the XGBRegressor outperforms the others.
- With RandomSearch hyperparameter tuning, the issues of SVR and ElasticNet have been resolved as those models achieve good performance. Other models' performances do not vary much, but there is a decrease in the performance of DecisionTreeRegressor. This can be explained as this hyperparameter tuning method is unable to find the optimal solution in the parameter space.
- With GridSearch parameter tuning, all models obtain good performance. The best one is still XGBRegressor.

Table 1: Comparison of results.

Model	without hyperparam tuning		with RandomSearch		with GridSearch	
	R^2	MSE	R^2	MSE	R^2	MSE
SVR	0.3945	0.2333	0.8349	0.0636	0.8349	0.0637
LinearRegression	0.8332	0.0643	0.8333	0.0642	0.8332	0.0643
Ridge	0.8363	0.0631	0.8364	0.0631	0.8363	0.0631
Lasso	0.263	0.2839	0.8369	0.0628	0.8369	0.0629
ElasticNet	0.2645	0.2834	0.8379	0.0625	0.8379	0.0625
KNeighborsRegressor	0.8189	0.0697	0.8167	0.0706	0.8267	0.0668
DecisionTreeRegressor	0.7735	0.0873	0.7486	0.0969	0.7842	0.0832
GradientBoostingRegressor	0.8734	0.0488	0.8818	0.0466	0.0.8983	0.0387
XGBRegressor	0.889	0.0428	0.8862	0.0435	0.9014	0.038

5 Conclusion

In this project, we apply data processing and data mining techniques to the task of laptop price prediction. We performs data preprocessing and data exploration to get a good training dataset. We conduct experiments with nine different regression models and two parameter tuning methods. The GridSearch tuning method prove effectively to find optimal set of hyperparameters for all models, while the model XGBRegressor consistently outerperforms others in all settings.