PREDICT LAPTOP PRICE

DATA DESCRIPTION:

	Company	TypeName	Inches	Ram	OS	Weight	Price_euros	Screen	ScreenW	ScreenH	...	RetinaDisplay	CPU_company	CPU_freq	CPU_model	PrimaryStorage	SecondaryStorage	Primar
0	Apple	Ultrabook	13.3	8	macOS	1.37	1339.69	Standard	2560	1600	...	Yes	Intel	2.3	Core i5	128	0	
1	Apple	Ultrabook	13.3	8	macOS	1.34	898.94	Standard	1440	900	...	No	Intel	1.8	Core i5	128	0	
2	HP	Notebook	15.6	8	No OS	1.86	575.00	Full HD	1920	1080	...	No	Intel	2.5	Core i5 7200U	256	0	
3	Apple	Ultrabook	15.4	16	macOS	1.83	2537.45	Standard	2880	1800	...	Yes	Intel	2.7	Core i7	512	0	
4	Apple	Ultrabook	13.3	8	macOS	1.37	1803.60	Standard	2560	1600	...	Yes	Intel	3.1	Core i5	256	0	

5 rows × 22 columns

As shown in the bottom left, we can see that the data has 21 predictors, ignoring the first column which is just labels. Out of those 21 predictors I will take Pirce_euros as the response variable, leaving 20 predictors for data processing and predicting the price for the laptops. The dataset contains numerical, and categorical data.

```
df.shape[0] # number of samples
```

```
1275
```

```
[7] df.shape[1] # find number of columns
```

```
22
```

The image above shows the lines of code used to print the number of samples, and number of columns in the dataset.

DATA PREPROCESSING:

```
# remove unwanted features
data = df.drop(['Company', 'TypeName', 'OS', 'Screen', 'Touchscreen','IPSpanel', 'RetinaDisplay', 'CPU_company', 'CPU_model', 'PrimaryStorageType', 'SecondaryStorageType', 'GPU_
data.head()
```

	Inches	Ram	Weight	Price_euros	ScreenW	ScreenH	CPU_freq	PrimaryStorage	SecondaryStorage
0	13.3	8	1.37	1339.69	2560	1600	2.3	128	0
1	13.3	8	1.34	898.94	1440	900	1.8	128	0
2	15.6	8	1.86	575.00	1920	1080	2.5	256	0
3	15.4	16	1.83	2537.45	2880	1800	2.7	512	0
4	13.3	8	1.37	1803.60	2560	1600	3.1	256	0

After loading the data, I removed all unwanted features from the dataset. In this case, all of the categorical features were removed leaving only the numerical features to be able to perform a linear regression analysis in the dataset.

```
# clean up NaN values
data.isnull().sum()
```

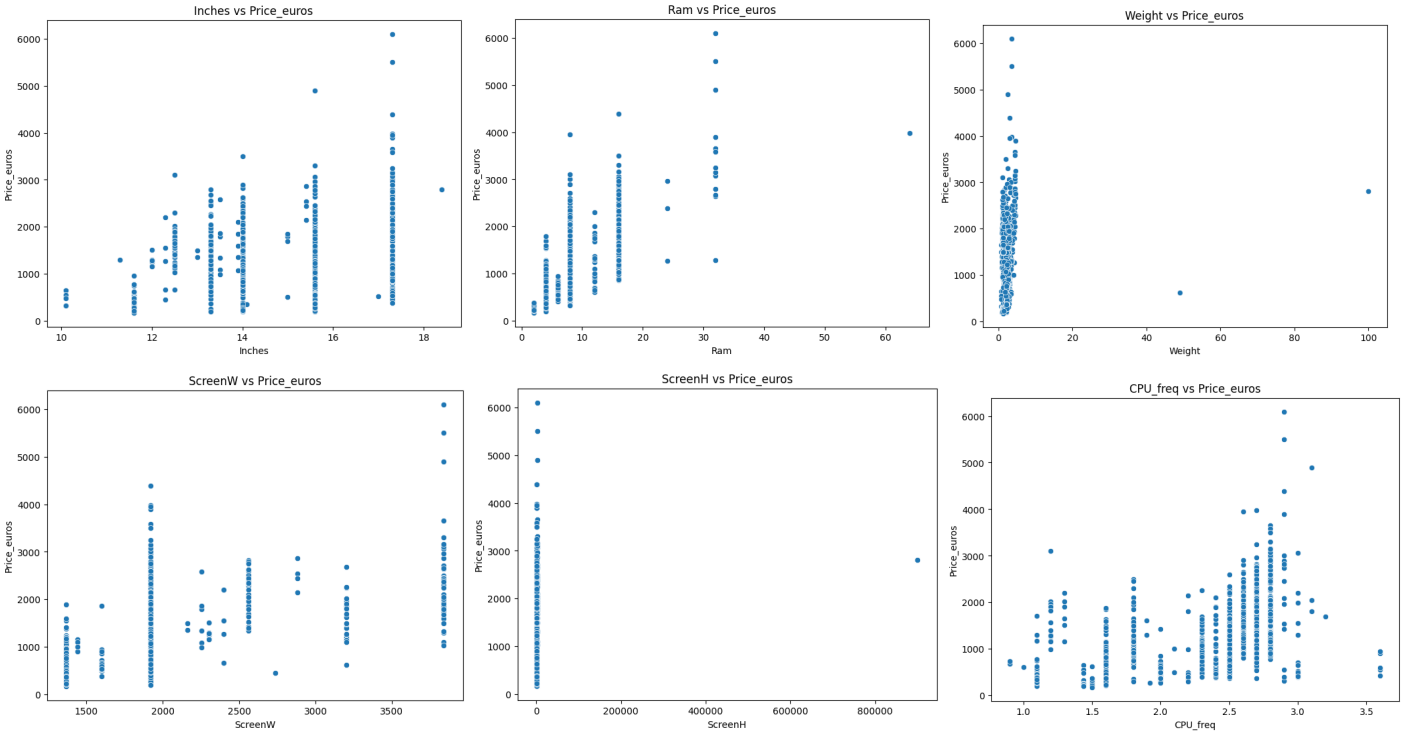
	0
Inches	0
Ram	0
Weight	0
Price_euros	4
ScreenW	0
ScreenH	0
CPU_freq	1
PrimaryStorage	0
SecondaryStorage	0

```
#Drop and check for null values
data.dropna(inplace = True)
data.isnull().sum()
```

	0
Inches	0
Ram	0
Weight	0
Price_euros	0
ScreenW	0
ScreenH	0
CPU_freq	0
PrimaryStorage	0
SecondaryStorage	0

Checking for null values, I found 4 null values for Price_euros, and 1 for CPU_freq. I deleted all the rows with null values to properly process the data to predict the laptop's prices.

EXPLORATORY DATA ANALYSIS:



Based on the images above, we can say that Ram, Weight, and CPU_freq have a positive correlation; indicating that as these features increase, the price tends to increase as well. On the other hand, Inches shows a very weak negative or negligible correlation, implying that the screen size may not have a substantial influence on the price. Other graphs show a non-linear relationship, like the screenH and screenW.

MODEL DEVELOPMENT & PERFORMANCE EVALUATION:

SIMPLE LINEAR REGRESSION:

Price_euros ~ Inches

OLS Regression Results

Dep. Variable:	Price_euros	R-squared:	0.005
Model:	OLS	Adj. R-squared:	0.004
Method:	Least Squares	F-statistic:	5.796
Date:	Wed, 27 Nov 2024	Prob (F-statistic):	0.0162
Time:	17:07:02	Log-Likelihood:	-10121.
No. Observations:	1270	AIC:	2.025e+04
Df Residuals:	1268	BIC:	2.026e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	639.8732	207.130	3.089	0.002	233.518	1046.228
Inches	33.0496	13.728	2.408	0.016	6.118	59.981

Omnibus:	377.038	Durbin-Watson:	2.018
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1263.310
Skew:	1.444	Prob(JB):	4.74e-275
Kurtosis:	6.942	Cond. No.	160.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
MSE: 489486.0433165707

Price_euros ~ Ram

OLS Regression Results

Dep. Variable:	Price_euros	R-squared:	0.549
Model:	OLS	Adj. R-squared:	0.549
Method:	Least Squares	F-statistic:	1546.
Date:	Wed, 27 Nov 2024	Prob (F-statistic):	1.02e-221
Time:	17:15:47	Log-Likelihood:	-9618.0
No. Observations:	1270	AIC:	1.924e+04
Df Residuals:	1268	BIC:	1.925e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	275.7976	25.569	10.786	0.000	225.635	325.960
Ram	101.9628	2.593	39.316	0.000	96.875	107.051

Omnibus:	261.131	Durbin-Watson:	2.030
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1106.802
Skew:	0.917	Prob(JB):	4.58e-241
Kurtosis:	7.190	Cond. No.	19.2

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
MSE: 221590.2842380382

When comparing the two previous simple linear regression models, I noticed that a simple linear model between price_euros and Ram has a better mean squared error than price_euro and Inches. I assume this is because the positive correlation between price and Ram is stronger than the negative or negligible correlation between price

and inches. None of the values in the table are random because the threshold 0.05 is met and the coefficient falls in the confidence interval.

Mean Squared Error (MSE): MSE values varied depending on the predictor used (Inches, CPU_freq, Ram).

MULTIPLE LINEAR REGRESSION:

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            Price_euros   R-squared:                       0.668
Model:                            OLS   Adj. R-squared:                  0.666
Method:                 Least Squares   F-statistic:                     317.5
Date:                Wed, 27 Nov 2024   Prob (F-statistic):          9.45e-296
Time:                        17:01:47   Log-Likelihood:                -9423.5
No. Observations:                1270   AIC:                         1.887e+04
Df Residuals:                    1261   BIC:                         1.891e+04
Df Model:                           8
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const           -167.0325    152.136     -1.098      0.272    -465.501     131.436
Inches           -41.6633     10.246     -4.066      0.000     -61.764     -21.563
Ram               80.3432      2.794     28.753      0.000      74.861      85.825
Weight            -1.8895      8.403     -0.225      0.822     -18.376      14.597
ScreenW            0.3631      0.026     13.896      0.000       0.312       0.414
ScreenH            0.0012      0.001      1.187      0.235      -0.001       0.003
CPU_freq         285.2308     25.083     11.371      0.000     236.021     334.441
PrimaryStorage    -0.2024      0.035     -5.759      0.000      -0.271      -0.133
SecondaryStorage  -0.0228      0.034     -0.664      0.507      -0.090       0.045
==============================================================================
Omnibus:                      330.944   Durbin-Watson:                   2.054
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1768.997
Skew:                           1.102   Prob(JB):                         0.00
Kurtosis:                       8.345   Cond. No.                     3.39e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.39e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
MSE: 163131.2889055351
```

For multiple linear regression I used all of the existing predictors that I left after cleaning the data. None of the values in the table are random and I met all of the confidence intervals. The MSE is better than the simple linear regression MSEs.

Mean Squared Error (MSE): MSE generally improved compared to simple linear regression when using multiple features.

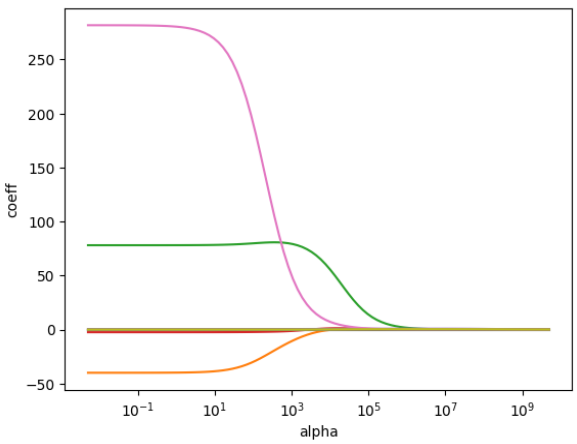
RIDGE REGRESSION:

```python
# the model
linear = LinearRegression().fit(X_train, y_train) # fit the model
y_hat = linear.predict(X_test) # predict the model
print(f"MSE: {mean_squared_error(y_hat, y_test)}") # print mse
print(pd.Series(linear.coef_, index = X.columns)) # ???
```

```
MSE: 148729.4418529383
const                0.000000
Inches             -40.026059
Ram                 78.014058
Weight              -2.489271
ScreenW              0.385276
ScreenH              0.001279
CPU_freq           281.612369
PrimaryStorage      -0.227613
SecondaryStorage    -0.024384
dtype: float64
```

Ridge():

```
MSE:148742.26832382078
const                0.000000
Inches             -39.880825
Ram                 78.042195
Weight              -2.486382
ScreenW              0.385422
ScreenH              0.001279
CPU_freq           280.268222
PrimaryStorage      -0.227623
SecondaryStorage    -0.024396
dtype: float64
```

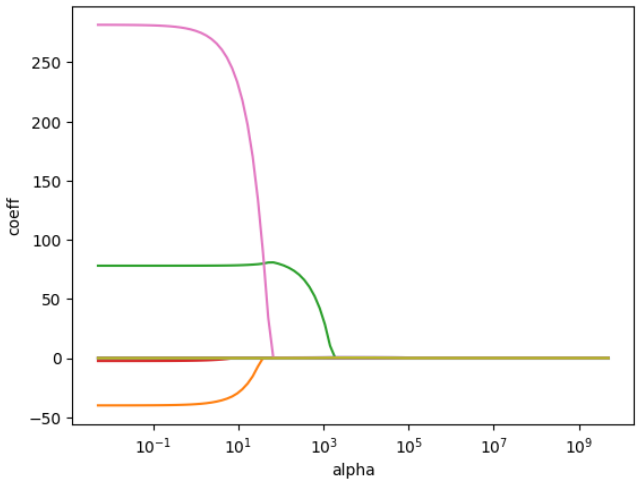



```
[197] ridge_cv = RidgeCV(alphas = alphas) # create model for ridge cross-validation
      ridge_cv.fit(X, y) # fit the model
      best_alpha = ridge_cv.alpha_ # Best alpha value from the best model
      print(best_alpha) # print the best alpha value

→ 5000000000.0

[198] np.log(12) # logarithmic function???

→ 2.4849066497880004

[199] ridge = Ridge(alpha = best_alpha).fit(X_train, y_train) # fit ridge model with best alpha value
      y_hat = ridge.predict(X_test) # predict the model
      mean_square_error = mean_squared_error(y_hat, y_test) # calculate mse
      print(f'MSE: {mean_square_error}') # print mse
      print(pd.Series(ridge.coef_, index = X.columns)) # ???

→ MSE: 446542.8501110951
  const              0.000000
  Inches             0.000015
  Ram                0.000538
  Weight             0.000014
  ScreenW            0.037113
  ScreenH            0.001947
  CPU_freq           0.000030
  PrimaryStorage    -0.006485
  SecondaryStorage   0.016970
  dtype: float64
```

Hyperparameters: alpha. This hyperparameter was selected using RidgeCV which automatically finds the best alpha within a range using cross-validation. When defining and creating the model for ridge regression I first created a model using LinearRegression() which give the best MSE out of all the models.

Mean Squared Error (MSE): MSE was better before using hyperparameter tuning with RidgeCV.

LASSO REGRESSION:




```
lasso = Lasso(alpha = best_alp).fit(X_train, y_train)
y_hat = lasso.predict(X_test)
mean_square_error = mean_squared_error(y_hat, y_test)
print(f'MSE: {mean_square_error}')
print(pd.Series(lasso.coef_, index = X.columns))
```

```
MSE: 214859.0897581228
const                0.000000
Inches               0.000000
Ram                 41.992742
Weight               0.000000
ScreenW              0.582977
ScreenH              0.001322
CPU_freq             0.000000
PrimaryStorage      -0.165429
SecondaryStorage     0.136563
dtype: float64
```

Hyperparameters: alpha. Similar to Ridge, the regularization strength. Also selected using LassoCV with cross-validation.

Mean Squared Error (MSE): MSE improved with hyperparameter tuning using LassoCV.

INTERPRETATION:

MODEL COMPARISON:

- Multiple Linear Regression outperformed Simple Linear Regression: This shows that combining features/predictors helps in capturing the overall relationship with laptop price better. I assume because when pricing a laptop you take into account all of the features involved, instead of just having 1 nice feature and all of the other ones are pretty bad.
- Ridge and Lasso, I selected these two models because they performed really well when learning about them in class. I know Ridge and Lasso help reduce overfitting, which leads to better generalization of new data.

SIGNIFICANT FEATURES:

I think the most significant features were Ram, CPU_freq, and Inches. Ram and CPU were very influential features when predicting for the response values; because when people go to the store to get a computer these are one of the first things I look at in the specs.

Inches were not as significant to the dataset as it was for me. I thought inches would have had a more influential prediction when it came to prices for laptops because I think the bigger the computer, the more expensive it is. So it was very interesting seeing how Inches was not a significant predictor of price.

RELATIONSHIP TO THE DATASET AND PROBLEM:

All these models aim to understand how different features in a laptop influence the price. The results can help predict prices for new laptops by giving specifications or in identifying the primary factors contributing to price differences in the market.

SUMMARY:

In this particular case the best model was LinearRegression() which was used for training and testing data of Ridge regression. After LR the best second model was Ridge before using alphas. Ridge could have performed better because it reduces the influence of less important features while still retaining all predictors.

Overall Multiple Linear regression and Ridge Regression before alpha performed better than Simple Linear Regression and Lasso Regression.

CONCLUSIONS:

In this project, I explored several linear regression models for predicting the laptop prices. Feature engineering and selection play a crucial role in model performance, this is why one of the first things is to clean the data and remove unnecessary features, while also making sure to not have null values in the dataset.

Regularized models like Ridge and Lasso can improve prediction accuracy by reducing overfitting. Although in this case, having all of the features was necessary to be able to achieve the lowest mse, because as mentioned in the previous section, to predict the price for a laptop we have to take into consideration many features.