AIML CA1
Regression

01

Title:

PC Price Predictions

BY KALEB NIM P2100829 DAAA/2A/02

Table of Contents

PROJECT OBJECTIVE

Using ML regression model to predict a continuous output variable.

We are given a dataset containing PC sale prices from a PC website and other attributes of each PC sale.

The prediction task is to create a predictive regression model to **predict the PC prices** based on the PC attribute given like CPU, Weight, ScreenSize Etc.

OUTPUT VARIABLE

The output variable is **price**, referring to the price of the PC sold in dollars ($).

Price ($) is defined as the **price paid upon purchase** of a single PC by a customer

The variable is a **numerical-continuous** variable, as such the prediction task requires a regression model.

DATA ATTRIBUTES

Number of Instances: 15320
Number of Features: 12
Zero Missing Values.

Method 2 no scaling

DATA DICTIONARY

Column	Description
Product ID	unique identifier ranging from 0 to 15319
Brand	Brand of the PC
Type	PC type, such as notebook, ultrabook
Screen Size	Size of the PC screen
Screen Specs	PC screen specs with resolution
CPU	CPU infomation of the PC
RAM	RAM information of the PC
Hard Disk	Hard Disk information of the PC
GPU:	Graphic card of the PC
Operating System	Operating system, such as Windows 10, macOS
Weight:	Weight of the PC
Price	**PC price ($)**

Target variable →

02

Exploratory Data Analysis(EDA)

note: For each step in EDA process, I'll be giving my Observations and Conclusions

1. Identifying Null Values

2. Cardinality of categorical Features

3.Distribution of Target Variable Price

4.Bi-Variate Analysis

EDA
Identifying Null Values

	percent_missing
Product ID	0.0
Brand	0.0
Type	0.0
Screen Size	0.0
Screen Specs	0.0
CPU	0.0
RAM	0.0
Hard Disk	0.0
GPU	0.0
Operating System	0.0
Weight	0.0
Price ($)	0.0

- Observations: No Null values present in the dataset
- Conclusion: No need for Imputation of data

EDA

Cardinality of categorical Features

Cardinality values

```
[7]  # Extracting only categorical features to find Unique values
     df_eda[['Brand', 'Type', 'Screen Specs', 'CPU', 'RAM', 'Hard Disk', 'GPU', 'Operating System', 'Weight']].nunique().sort_values()
```

```
Type                6
RAM                 9
Operating System    9
Brand              19
Hard Disk          39
Screen Specs       40
GPU               110
CPU               118
Weight            179
dtype: int64
```

Observations :

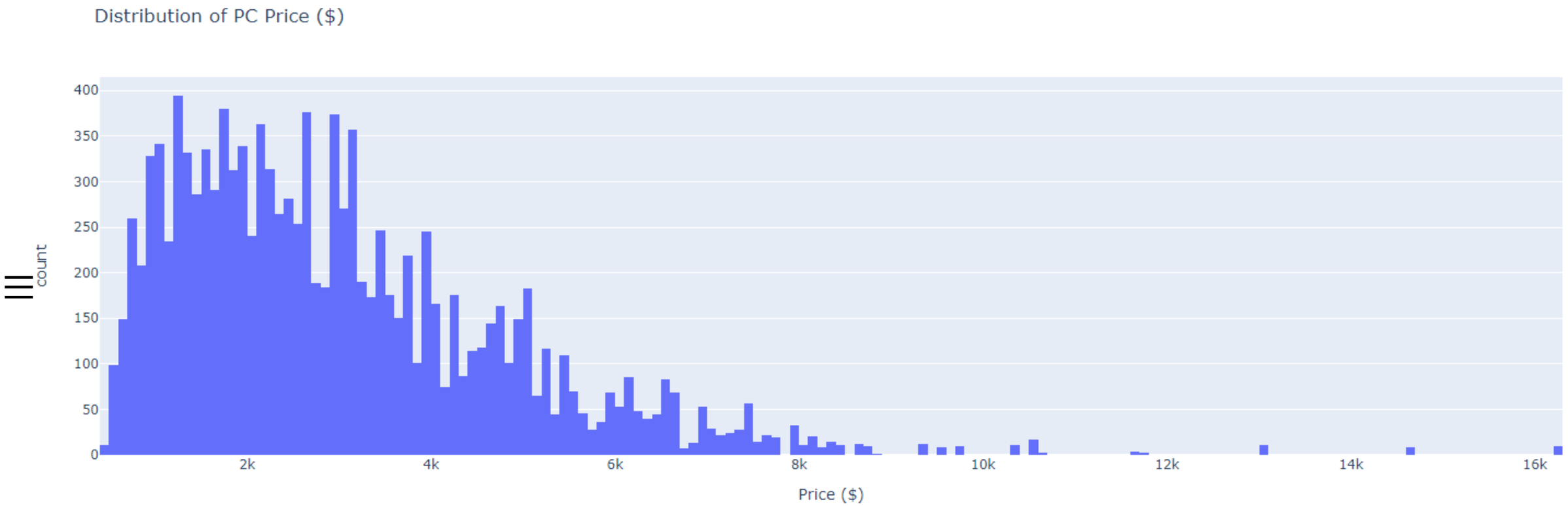
- No single value for any features
- CPU has the highest cardinality, with 118 unique values, extremely similar to GPU , with 110 unique values

Conclusions:

- Since nominal valued, One-hot-encoding not optimal as might lead to High dimension data
- Attempt further preprocessing before encoding/attempt other encoding methods

EDA

Distribution of Target Variable Price



Observations :

- Highly Positively Skewed
- Presence of outliers

Conclusions:

- Might want to attempt to Log-scale Price

EDA

Distribution of Target Variable Price

Observations :

- Distribution relatively more normal
- Presence of outliers

Conclusions:

- Price follows a log-normal distribution
- Attempt to apply a log transformation to check for model performance improvement

EDA

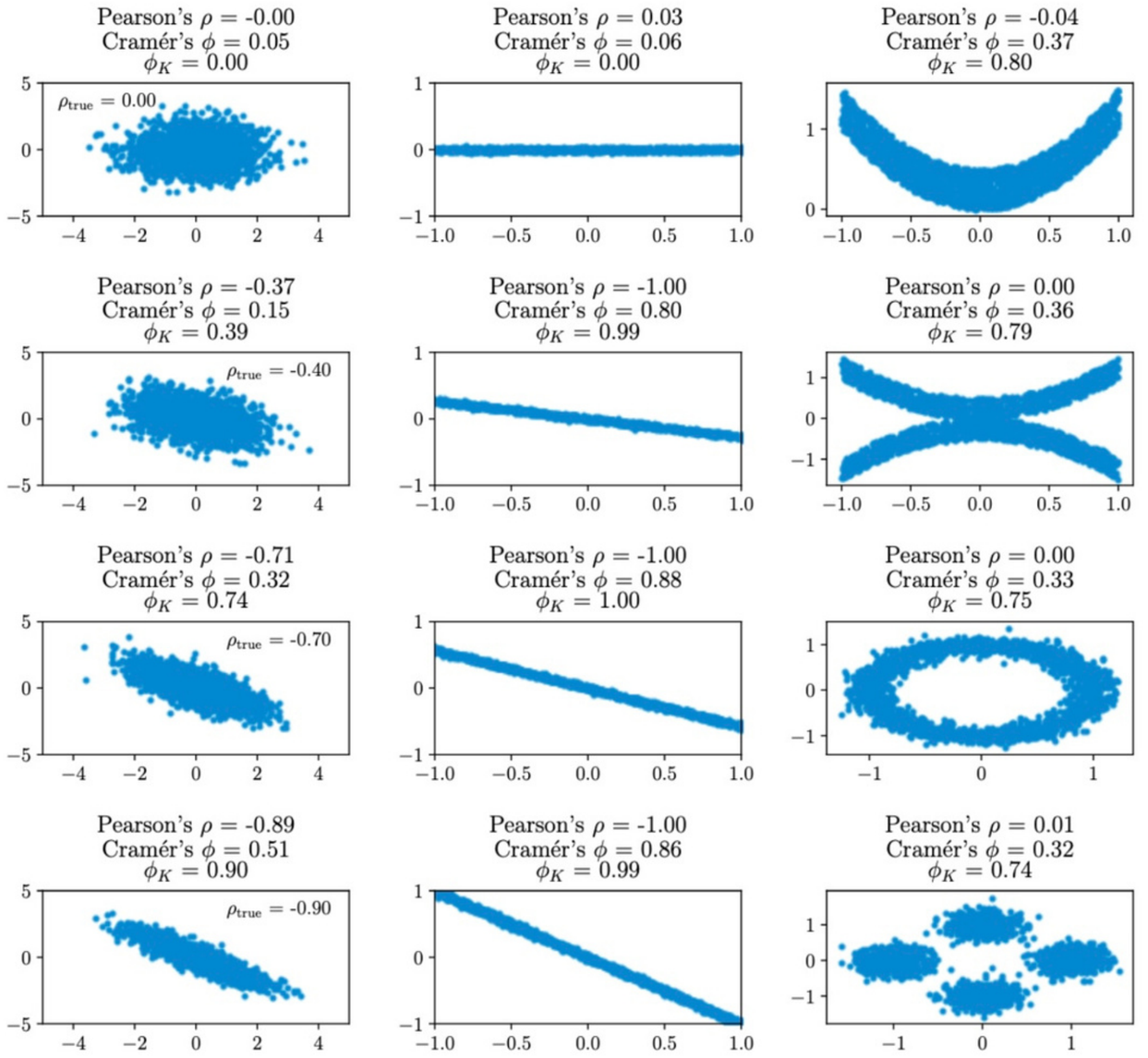
Phik correlation metric:

(same as classification)

Berif intro to Phik

Phik correlation (ϕk) is the latest relatively new correlation metric that is based on several refinements to Pearson's $\chi 2$ (chi-squared) contingency test

Unfortunately there's isn't a closed forum formula . Therefore let see a visual representation of how Phik correlation works



source

Usage in EDA

We will be using Phik correlation in Bi-variation analysis:

- Check high correlation between two variable
- Check highly correlated feature to target feature Price

EDA (raw dataset)

Bi-variate Analysis

- **Correlation Metric**

  Phik correlation (ϕk)

- **Greatest Correlation with Price**

1. GPU ----------> 0.9ϕk

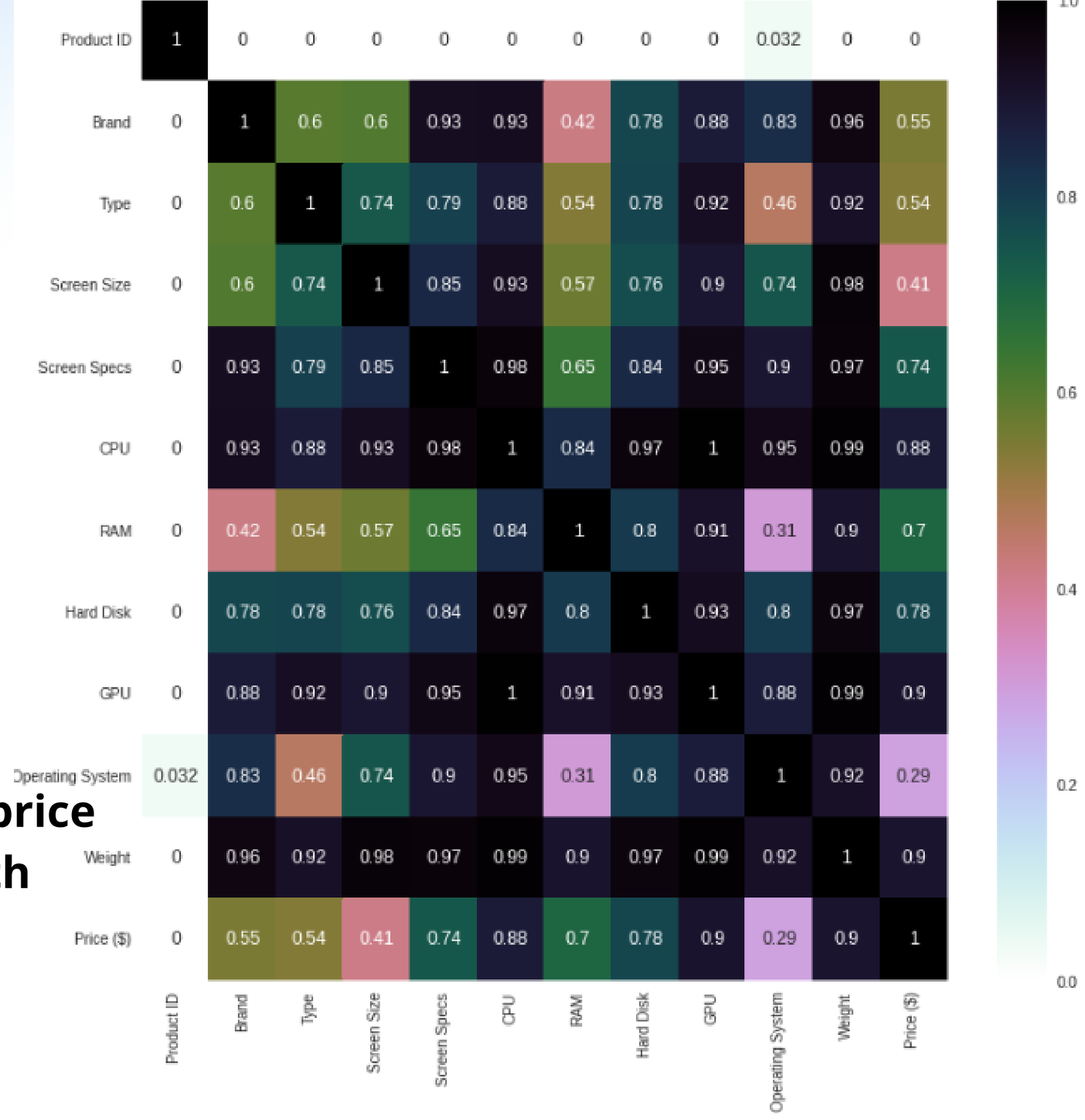
≡ 2. Weight --> 0.9ϕk

3. CPU---------->0.88ϕk

- **Correlated Features**

1. The top 3 Greatest correlation with price CPU, GPU,price is Highly correlated with every other feature

Lets take a deeper look.



EDA (processed dataset)

Bi-variate Analysis

- **Correlation Metric**

  Phik correlation (ϕk)

- **Greatest Correlation with Price**

1. GPU_type ----------> 0.88ϕk

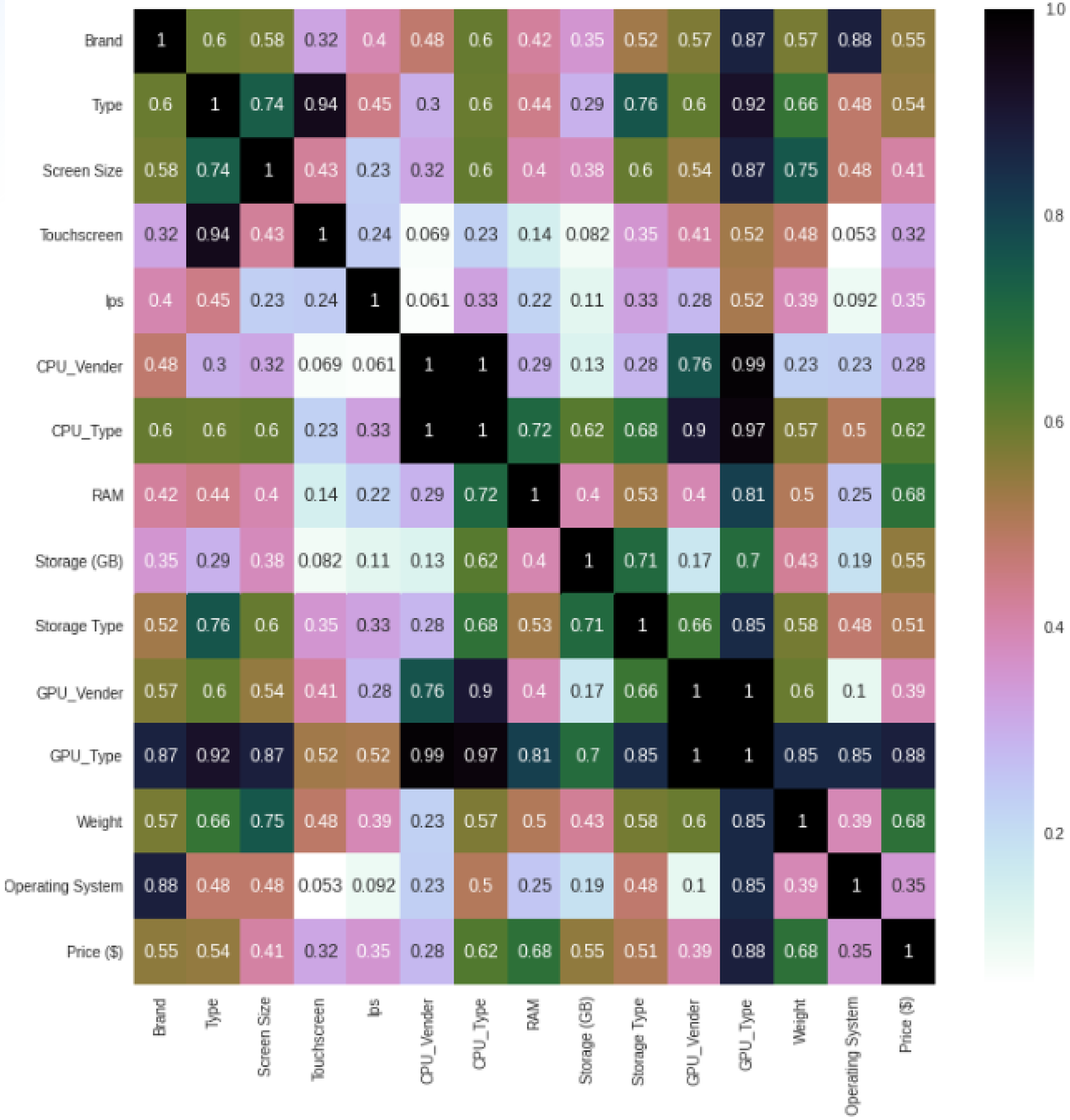
≡ 2. RAM --> 0.68ϕk

3. Weight---------->0.68ϕk

- **Correlated Features**

1. GPU type still is highly correlated to most features

2.



Note: how i derived this dataset found in Data processing

EDA

Bi-variate Analysis

- **Correlation Metric**

  Phik correlation (ϕk)

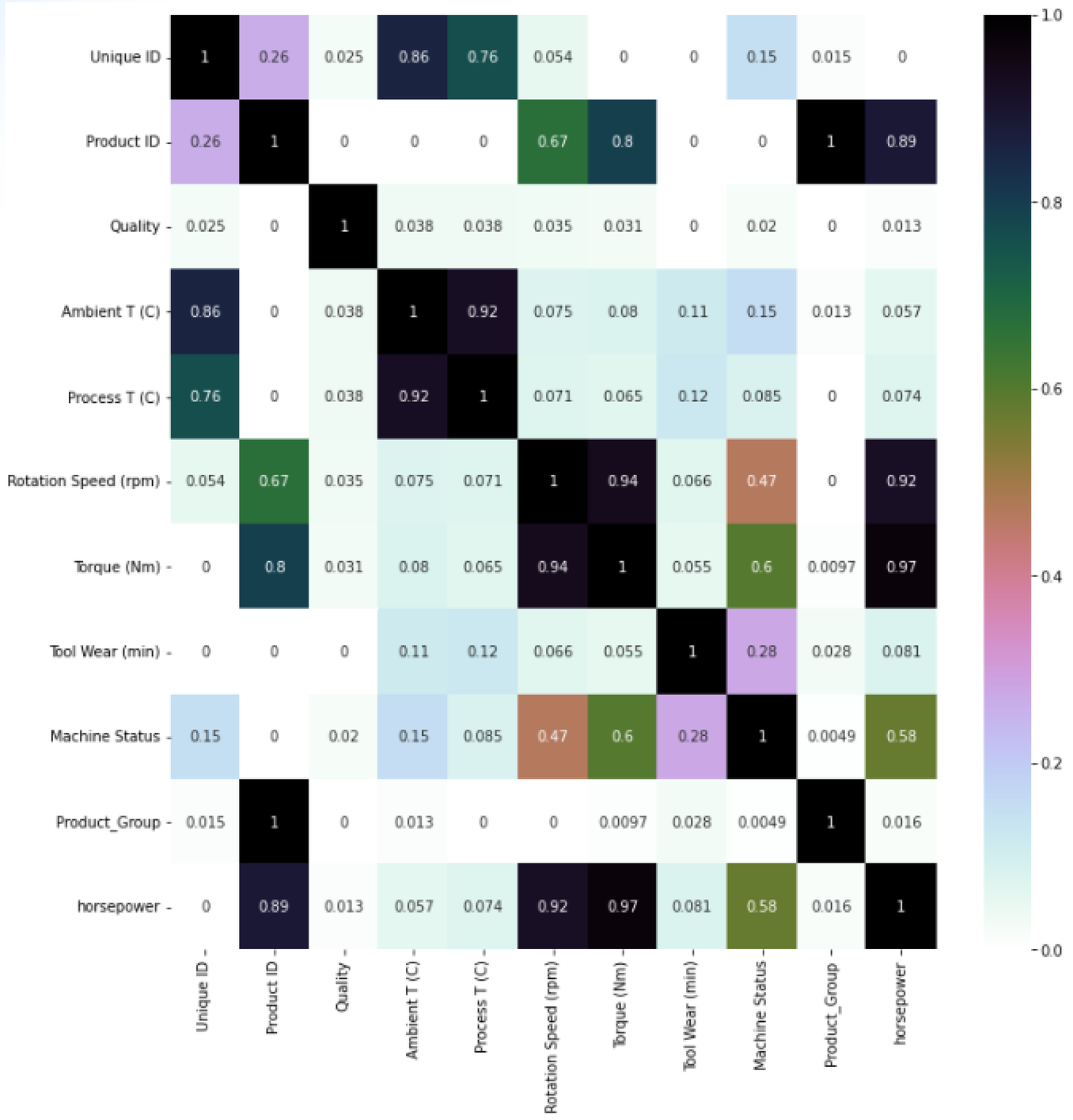
- **Greatest Correlation with Machine Status**

1. Torque (Nm) ----------> 0.6ϕk
2. Rotation Speed (rpm) --> 0.47ϕk
3. Tool Wear (min) ---------->0.28ϕk

- **Correlated Features**

1. Ambient T (C) and Process T (C) is at 0.92ϕk

2. Torque (Nm) and Rotation Speed (rpm) at 0.93ϕk



03

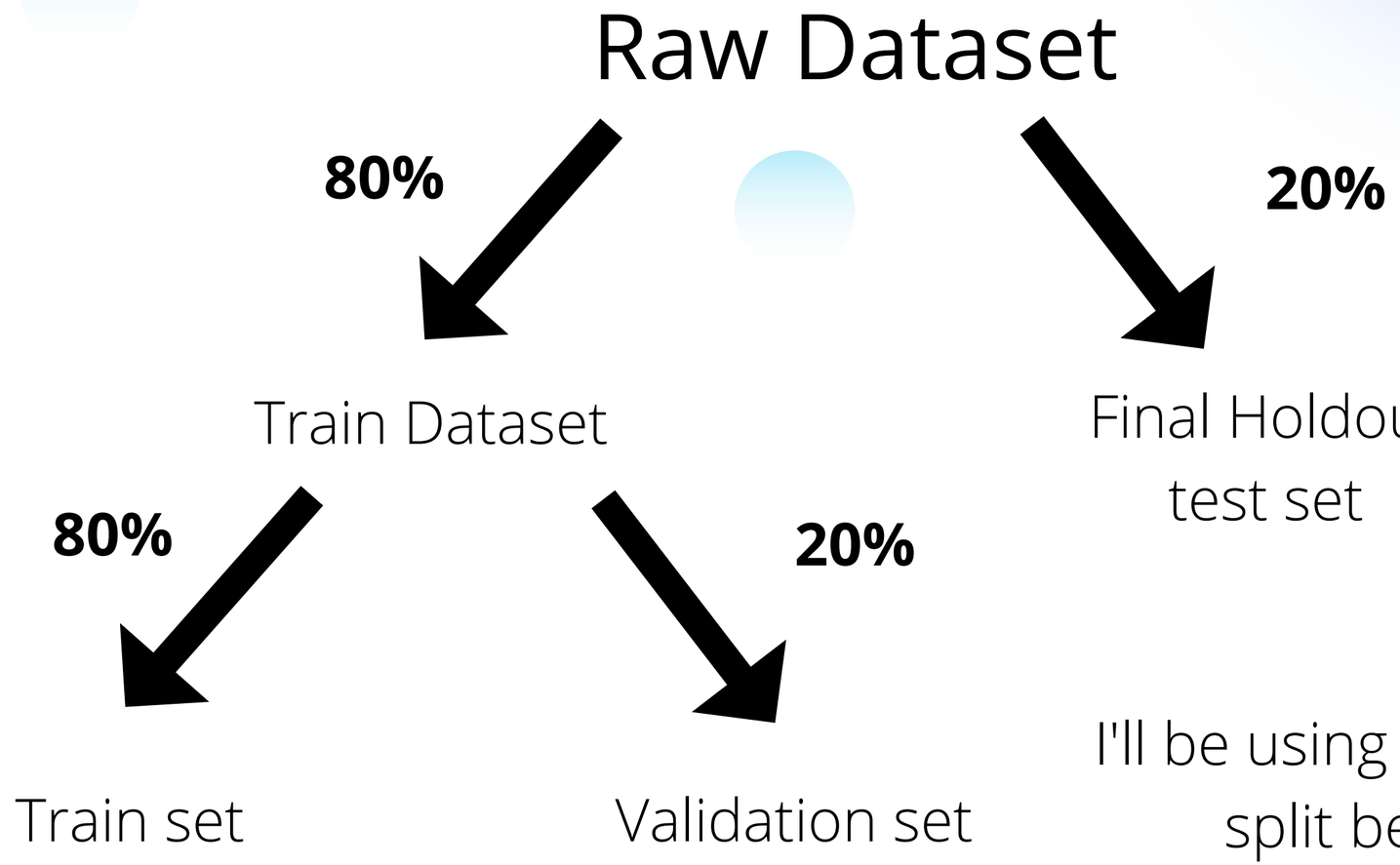
Feature Processing

List of techniques applied to dataset

- **Data Partition**
- **Feature Scaling**
- **Categorical Encoding**
  - **One-Hot-Encoding**
  - **Frequency Encoding**
- **Split feature based on real-life context**

Feature Processing

Data Partition



I'll be using the conventional **64-16-20** split between the 3 datasets

Feature Processing

Feature Scaling

Standardscalar

Translating the all numerical features: mean value of zero and a standard deviation of one.

$$z = \frac{x - \mu}{\sigma}$$

Example:

RAM	Storage (GB)	Weight
4.0	128.0	2.20
8.0	256.0	1.91
16.0	256.0	2.43
8.0	512.0	1.37
4.0	32.0	1.50
...	...	...
16.0	128.0	2.72
8.0	128.0	3.00
16.0	512.0	1.95
8.0	256.0	2.02
4.0	16.0	1.50

	RAM	Storage (GB)	Weight
	-0.900451	-0.859605	0.256850
	-0.063996	-0.509937	-0.177007
	1.608913	-0.509937	0.600943
	-0.063996	0.189399	-0.984878
	-0.900451	-1.121857	-0.790390
	...	...	...
	1.608913	-0.859605	1.034800
	-0.063996	-0.859605	1.453696
	1.608913	0.189399	-0.117164
	-0.063996	-0.509937	-0.012440
	-0.900451	-1.165565	-0.790390

Note: We will be normalizing all features that were not one-hot-encoded as there is no need to "normalize" what is already categorical.

Feature Processing

OK SO THIS IS WHERE THE HEADACHE STARTS

From our EDA, we understand the following about our Categorical features:

Features CPU, GPU have high cardinility (100+ unique values respectively)
All features are Nominal valued

Dealing with high cardinality features can lead to having 300++ features if we blindly one-hot-encode all categorical features then we which leads to problem of Curse of dimensionality.

SO to tackle this I've come up with 3 different Methods.
Method 1, Method 2 and Method 3. Details will be in the following slide

Feature Processing

Method 1: one-hot-encode all categorical variables

Since categorical features are norminal it logical to perform one-hot-encoding. However, We will end up with data with extremely high dimensions which will pose problems (curse of dimensionality)

Although we could further process the data by reducing dimensonality using PCA.While it is technically possible to use PCA on categorical variables (after one-hot-encode) we should not as variables don't belong on a coordinate plane.

Other methods to include Recursive Feature Elimination to extract only relevant features

Method 2: Split High Cardinality features based on real-life context groups followed by one-hot-encoding.

The groups ill be splitting will be as follows:

1. Extract CPU Vender, CPU Type and CPU Speed in Different Columns
2. Extract Memory type from Memory Column
3. Extract GPU Vender, GPU Type in Different Columns
4. Extract IPS and Touchscreen Feature form ScreenResolution Column

note: I'm personally am not a avid PC enthusist. Thus i will be using the grouping derived from PC context from EVERYDAYCODINGS

Method 3: Frequency Encode + One-hot-Encoing

Frequency endoing refers to replacing the categories by their frequencies.

We will perform frequency encoing to features with high cardinility and one-hot-encode the rest of the categorical features

This way we won't have a high dimension space dataset while still retaining infomation about the categories

Modelling

- Cross validate using KFold each model on training set, and keeps a history of the model performance.
- Scoring on Validation set
- Plot learning curve of each model

Selected Metric for eval: Mean Absolute Error

```python
## Quick Eval function to fit and score all models at once
def quick_evalutaion(models, X_train, X_test, y_train, y_test, metrics=["neg_mean_squared_error","neg_mean_absolute_error","neg_mean_absolute_percentage_error"],curve = True,
    hist = {}

    for idx, model in (enumerate(models)):
        try:
            clf = model(random_state=42) # Setting random_state for certain model
        except:
            clf = model()
        clf.fit(X_train, y_train)
        test_prediction = clf.predict(X_test) # Testing on validation dataset

        MAPE_test = mean_absolute_percentage_error(y_test, test_prediction)
        MAE_test = mean_absolute_error(y_test, test_prediction)
        MSE_test = mean_squared_error(y_test,test_prediction)

        # 5-Fold CV
        cv_hist = cross_validate(clf, X_train, y_train, scoring=metrics,verbose=1)

        # Record down the performance
        hist[model.__name__] = dict(
            # train_acc = acc_train,
            fit_time = cv_hist['fit_time'].mean(),
            score_time = cv_hist['score_time'].mean(),
            cv_MAE = cv_hist['test_neg_mean_absolute_error'].mean(),
            MAE_test_score = MAE_test,
            cv_MAPE = cv_hist['test_neg_mean_absolute_percentage_error'].mean(),
            MAPE_test_score = MAPE_test,
            cv_MSE = cv_hist['test_neg_mean_squared_error'].mean(),
            MSE_test_score = MSE_test
        )

        # Plotting the learning Curve of each Model using: neg_mean_squared_log_error
        if curve:
          fig, ax = plt.subplots(figsize=(10, 8))
          train_sizes = np.linspace(.1, 1.0, 10)
          train_sizes, train_scores, test_scores = learning_curve(clf, X_train, y_train, cv = cv, n_jobs = -1, train_sizes = train_sizes, scoring="neg_mean_absolute_error")
          scores = pd.DataFrame({
          "Train Sizes" : np.tile(train_sizes, train_scores.shape[1]),
          "Train Scores" : train_scores.flatten(),
          "Test Scores" : test_scores.flatten()
          }).melt(value_vars=["Train Scores", "Test Scores"], var_name="Score Type", value_name="Scores", id_vars=["Train Sizes"])
          # print(f"THis is train_sizes:\n{train_sizes}\n This is train_scores:\n{train_scores}\n This is test_scores:{test_scores}")
          sns.lineplot(data=scores, x="Train Sizes", y="Scores", hue="Score Type", ax = ax ,palette=['#3DD5E2', '#A045B5'])
          ax.set_title(f"Learning Curve of {model}")
          ax.set_ylabel("neg_mean_absolute_error")
          ax.set_xlabel("Train Sizes")
          plt.show()

    # plt.tight_layout()
    # display(pd.DataFrame(hist).T)

    results = pd.DataFrame(hist).T
    return results
    # plt.show()
```

04

Model Selection

Selecting Final model based on:

- **Variance/bias**
- **Interpretability**
- **Evaluation Metric performance**
  - **Chosen metric: Mean Absolute Error**
- **Fit/score time**

Model Selection

Results: Method 2 (slide 22)

	fit_time	score_time	cv_MAE	MAE_test_score	cv_MAPE	MAPE_test_score	cv_MSE	MSE_test_score
DummyRegressor	0.012689	0.001093	-1401.582539	1433.038544	-0.712372	0.708108	-3.314905e+06	3.607604e+06
BayesianRidge	0.130146	0.006611	-533.208994	538.878808	-0.211143	0.207031	-5.430868e+05	5.584676e+05
LinearRegression	0.163341	0.009714	-532.323535	538.535588	-0.211038	0.206757	-5.431484e+05	5.597880e+05
GradientBoostingRegressor	1.617326	0.008739	-449.327215	458.023683	-0.176616	0.176319	-3.746642e+05	3.874060e+05
DecisionTreeRegressor	0.062951	0.005477	-32.348349	413.530342	-0.011376	0.117547	-1.396345e+04	1.612307e+06
HistGradientBoostingRegressor	1.379355	0.016415	-236.683014	290.783661	-0.089118	0.107015	-1.123693e+05	1.721866e+05
RandomForestRegressor	3.484714	0.058740	-38.116978	252.457666	-0.013208	0.072264	-1.479048e+04	5.795514e+05
ExtraTreesRegressor	4.221316	0.059551	-32.394002	76.896497	-0.011417	0.024233	-1.398590e+04	2.158772e+04
KNeighborsRegressor	0.014398	0.373668	-72.356492	38.705877	-0.024780	0.014407	-3.482381e+04	1.541381e+04

Model Selection

Results: Method 3 (slide 22)

	fit_time	score_time	cv_MAE	MAE_test_score	cv_MAPE	MAPE_test_score	cv_MSE	MSE_test_score
DummyRegressor	0.003197	0.000934	-1435.093284	1406.315145	-0.720243	0.710594	-3.494597e+06	3.420915e+06
LinearRegression	0.017340	0.004421	-696.121128	690.760428	-0.265919	0.262341	-9.627924e+05	9.188949e+05
BayesianRidge	0.024633	0.004450	-696.276401	690.062320	-0.265744	0.261115	-9.630402e+05	9.185257e+05
GradientBoostingRegressor	0.875449	0.007326	-465.609562	514.729818	-0.181720	0.200277	-4.078765e+05	5.166373e+05
HistGradientBoostingRegressor	0.520477	0.012966	-223.334579	308.683190	-0.089470	0.118608	-9.612266e+04	2.007577e+05
DecisionTreeRegressor	0.030593	0.003694	-15.424262	199.865705	-0.006375	0.063727	-3.292218e+03	4.778257e+05
RandomForestRegressor	1.922895	0.051928	-21.412606	170.340685	-0.008396	0.054547	-4.264259e+03	2.464049e+05
ExtraTreesRegressor	1.839498	0.054272	-15.338349	97.955342	-0.006324	0.035140	-3.242712e+03	2.435977e+04
KNeighborsRegressor	0.005522	0.319333	-64.781139	27.727516	-0.023383	0.009908	-3.056977e+04	9.496011e+03

Model Selection Analysis

Looking at the results for both methods

We can observe the following for different family of models:

Final selected Model: ExtraTreesRegressor

- Linear Models(Linear Regression,BayesianRidge):

Low Variance, may test scores and train cv scores little difference
Does not seems to suffers from major overfitting
Might be suffering from underfitting due to inductive biases.
Potential improvement: Try out polynomial regression and evaluate the model performance

- Distance Based models(KNeighborsRegressor):

Best performing model in terms of lowest MAE test scores. However,
Extremely High variance as seen from the large difference in CV test scores and validation test scores.
Severly overfitted to data
Might be suffering from slight underfitting.
Potential improvement: Increase the number of neighbours to reduce overfitting

- DecisionTree:

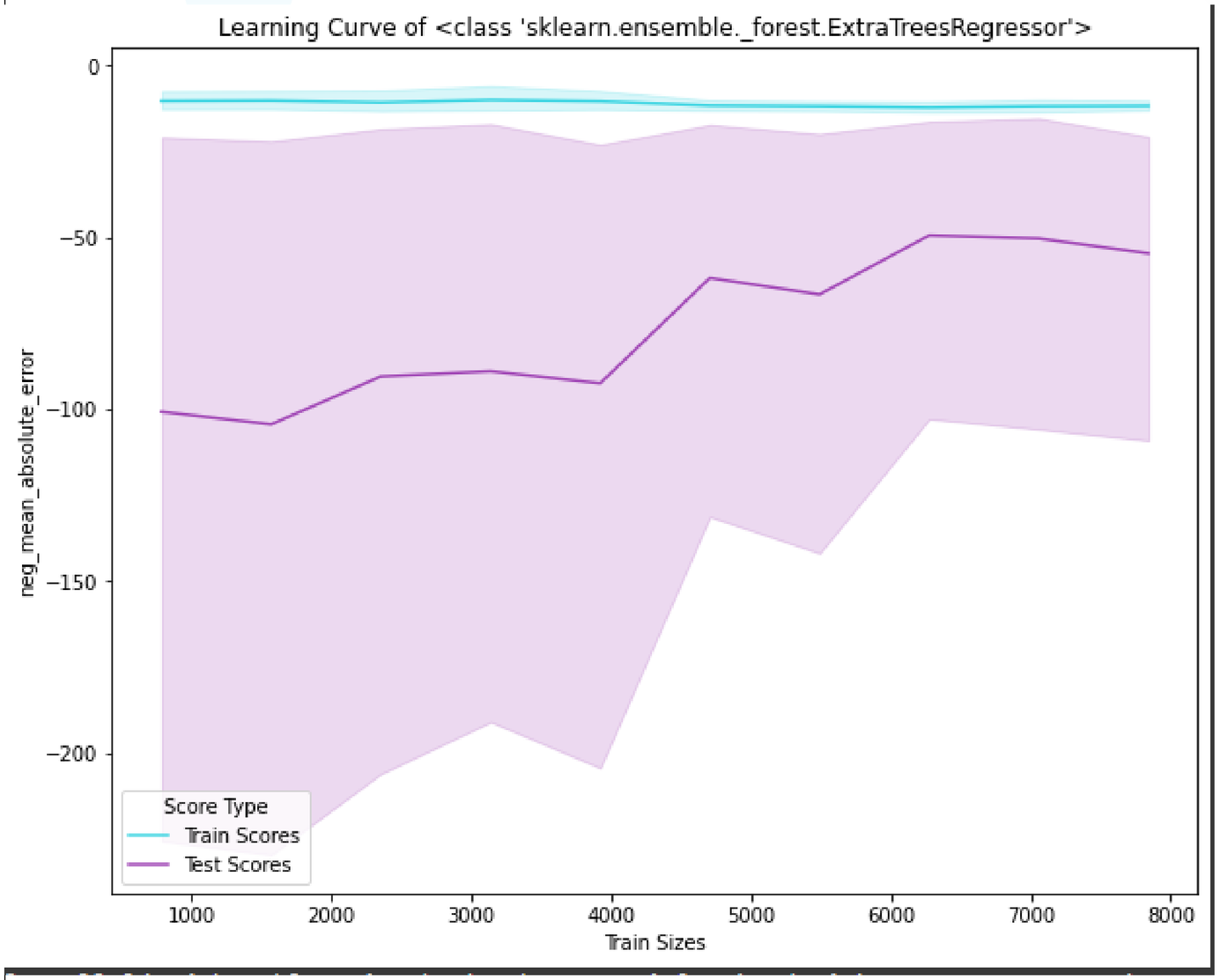
Extremely High variance as seen from the large difference in CV test scores and validation test scores.
Low biases and able to obtain relatively lower MAE test score
Potential improvement: HyperparamTune model better to reduce overfitting by limiting the model complexity

- Ensemble Tree Models (RandomForest, GradientBoosting,ExtraTreesRegressor)

Relatively high variance
Does not suffer as much overfitting than decision tree, however, overfitting still occurs
Low biases and able to obtain lowest MAE test score (out of all the family of models)
Potential improvement: HyperparamTune model to reduce overfitting

Model Selection

Analysis



Final selected Model: ExtraTreesRegressor

Use Method 3 for Feature processing

06

Model Improvement

Attempted list of methods

- **Log scaling to Target variable Price**
- **Feature selection**
- **Compare Feature Scaling Methods**
- **Hyperparameter tuning**

Model improvement

Log scaling to Target variable Price

From our EDA process, price has a very right skewed distribution. Skewness, the distortion from normal distribution, could affect linear models performances.

Furtermore, A distribution that is symmetric or nearly so is often easier to handle and interpret than a skewed distribution.

To counteract this problem, we will be applying LogTransformation on the target variable.

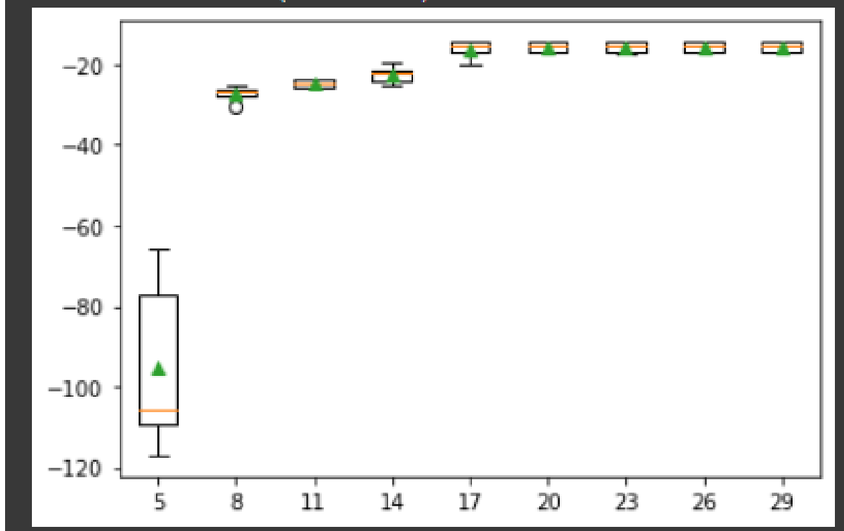
Since we will be using the 'FunctionTransformer' with parameters set to 'np.log1p' which applies log transformation for us

Model improvement

Feature selection

Method: Recursive Feature Elimination

```
>5 -95.055538 (19.994072)
>8 -27.220050 (1.824161)
>11 -24.753712 (0.932031)
>14 -22.476489 (1.976829)
>17 -16.241855 (2.123120)
>20 -15.630641 (1.112562)
>23 -15.661684 (1.154047)
>26 -15.605555 (1.080480)
>29 -15.645327 (1.131953)
```



We can see that when eliminated to just 23 features, the MAE score is the lowest. Hence will choose to RFE till there's 23 features remaining

Model Improvement
Hyper-parameter tuning

Method: **RandomSearchCV**
I chose RandomSearchCV instead of GridseachCV as it is more efficient through computing power and time.

We will try adjusting the following set of hyperparameters:

- n_estimators = number of trees in the forest
- max_features = max number of features considered for splitting a node
- max_depth = max number of levels in each decision tree
- min_samples_split = min number of data points placed in a node before the node is split
- min_samples_leaf = min number of data points allowed in a leaf node
- bootstrap = method for sampling data points (with or without replacement)

Model Improvement

Hyper-parameter tuning

```
# Number of trees in ExtraTrees
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
```

Model Improvement

Hyper-parameter tuning

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 75, cv = 3, verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
rf_random.fit(over_X_train, over_y_train)

rf_random.best_params_
```

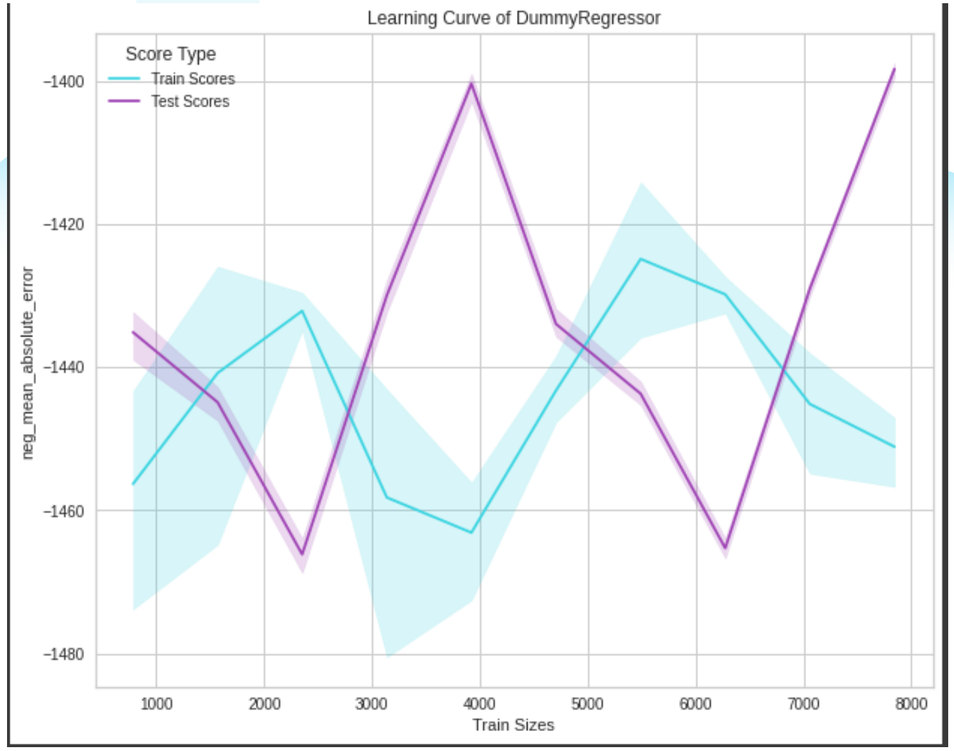
```
Fitting 3 folds for each of 75 candidates, totalling 225 fits
{'bootstrap': False,
 'max_depth': None,
 'max_features': 'sqrt',
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 400}
```

06

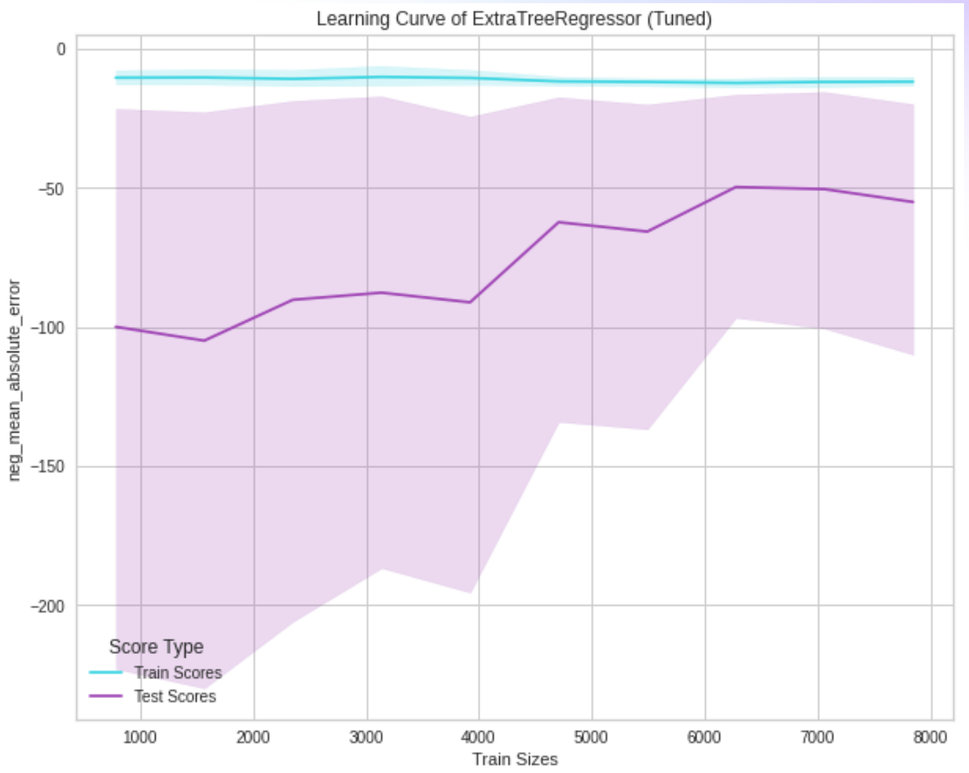
Model Evaluation

Model Evaluation

Stupid baseline: DummyRegressor



Final Model: ExtraTreeRegressor (Tuned)

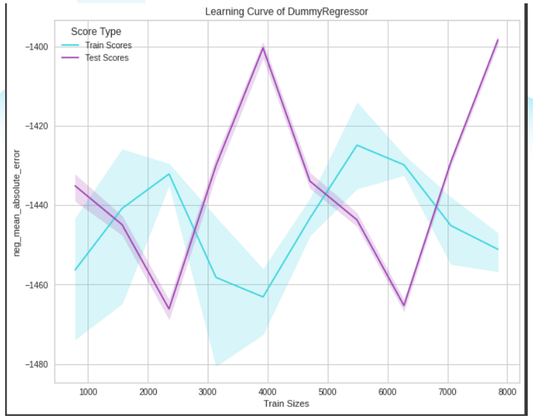


	MAE_test_score	MAPE_test_score	MSE_test_score	cv_MAE	cv_MAPE	cv_MSE	fit_time	score_time
DummyRegressor	1406.315145	0.710594	3.420915e+06	-1435.093284	-0.720243	-3.494597e+06	0.003427	0.001067

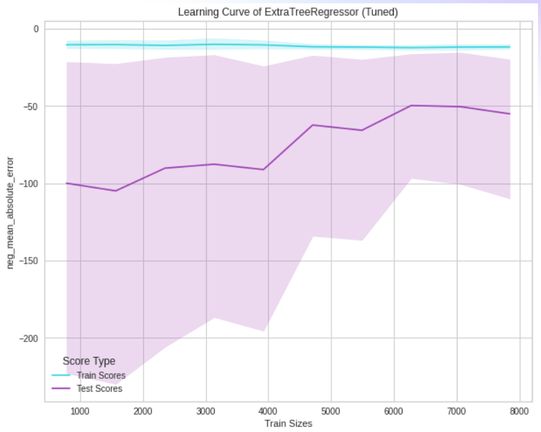
	MAE_test_score	MAPE_test_score	MSE_test_score	cv_MAE	cv_MAPE	cv_MSE	fit_time	score_time
ExtraTreeRegressor (Tuned)	17.525948	0.006677	3893.498788	-15.351024	-0.006332	-3246.983216	54.187011	0.684427

Model Evaluation

Stupid baseline: DummyRegressor



Final Model: ExtraTreeRegressor (Tuned)



	MAE_test_score	MAPE_test_score	MSE_test_score	cv_MAE	cv_MAPE	cv_MSE	fit_time	score_time
DummyRegressor	1406.315145	0.710594	3.420915e+06	-1435.093284	-0.720243	-3.494597e+06	0.003427	0.001067

	MAE_test_score	MAPE_test_score	MSE_test_score	cv_MAE	cv_MAPE	cv_MSE	fit_time	score_time
ExtraTreeRegressor (Tuned)	17.525948	0.006677	3893.498788	-15.351024	-0.006332	-3246.983216	54.187011	0.684427

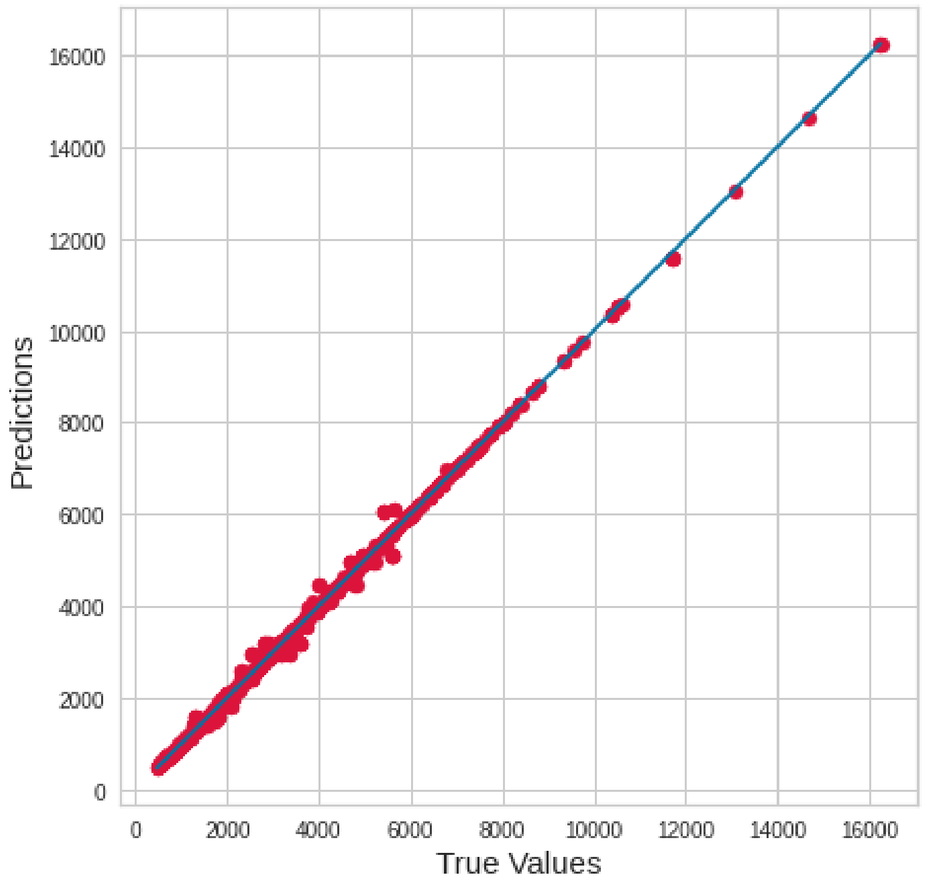
Extra trees regressor has much smaller errors compared to DummyRegressor, based on MAE by 14K. However, the fitting time for ExtraTreesRegressor is high at 54s compared to 0.003 for baseline (which makes sense)

- For ExtraTreesRegressor, we can see a gap, variance, between training score and cross-validation score. Thus can conclude the model is slightly underfitted.

- For Dummy classifier, we can see a very high bias from the learning curve score

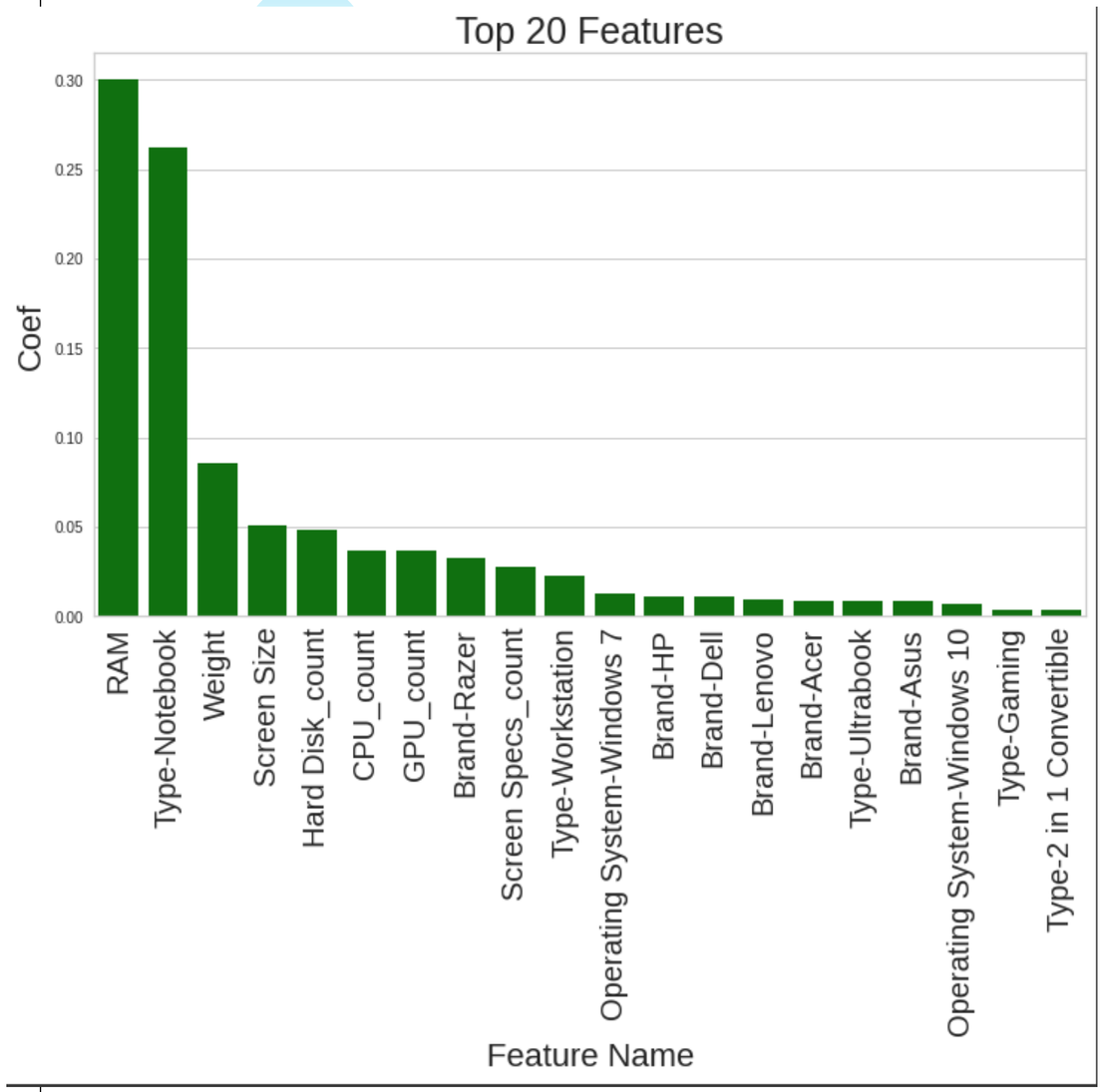
Error Plots

Prediction Error Plot



- In general, from the plot above we can clearly visualize that the prediction follows close to the True Values
- This indicates that the model performs relatively well for PC prices of almost all ranges
- Howevever, There is a slight larger margin around True values 3k
- Model not able to predict prices of PC around that 3k mark as well as other price points
- Able to predict outlier datapoint very well, no data points above the 10k mark that has a high difference margin from predictions

Feature Importance



Our Final Model ExtraTreeRegressor ranks RAM the most important feature at 0.3coef followed by Type-Notebook than Weight.

It's surprising that GPU and CPU are not the top few important features given that it was highly correlated to Price in our EDA. Maybe it could be due to the fact I frequency encoded the category.

nonetheless , our final model was able to perform very well to predict PC prices which is the main goal of the project