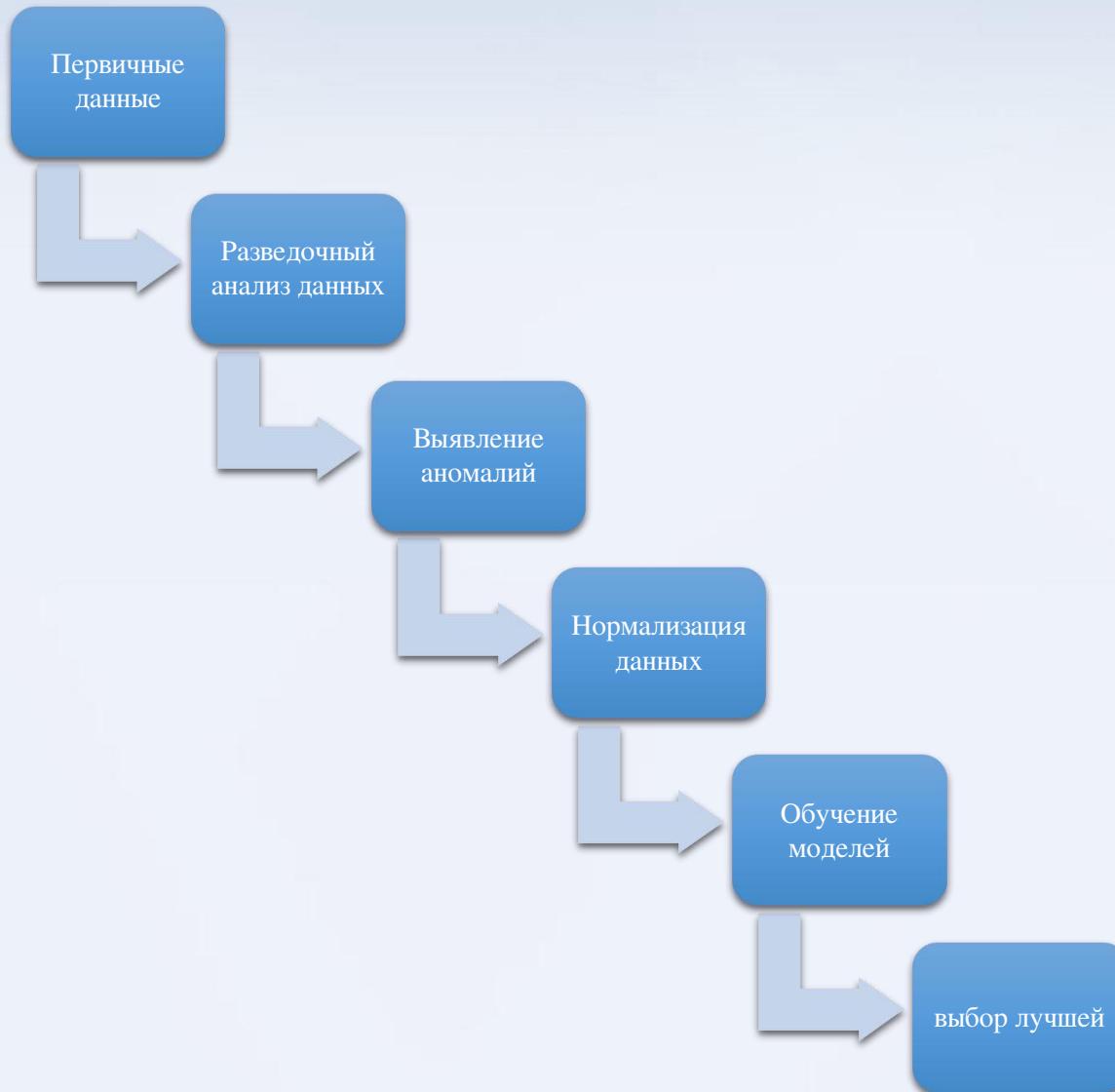




# ML-Обработка цен на ноутбуки



# Исходные данные

- **984 строки данных**
  - **98 параметров, из них:**
    - **без пропусков всего 24**
    - **3 float64**
    - **1 int64**
    - **все остальные object**



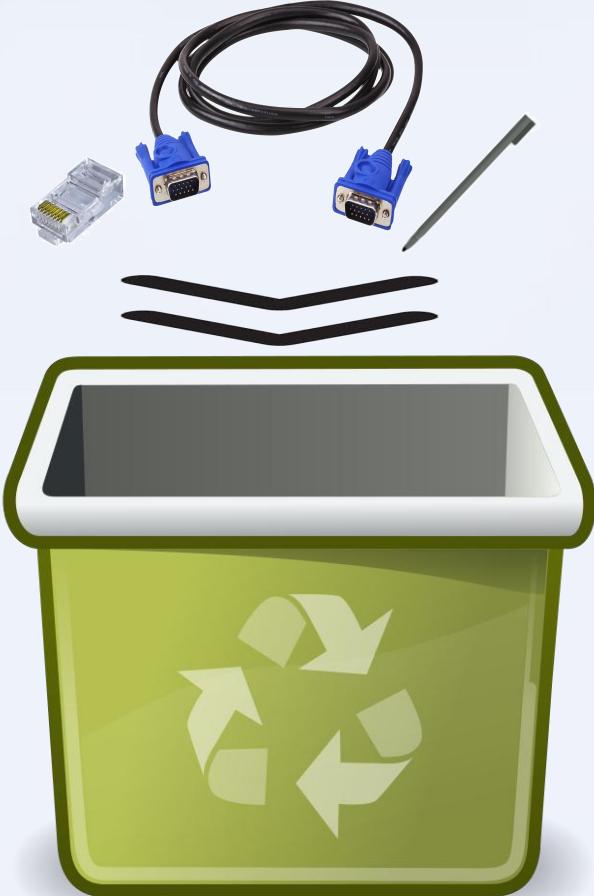
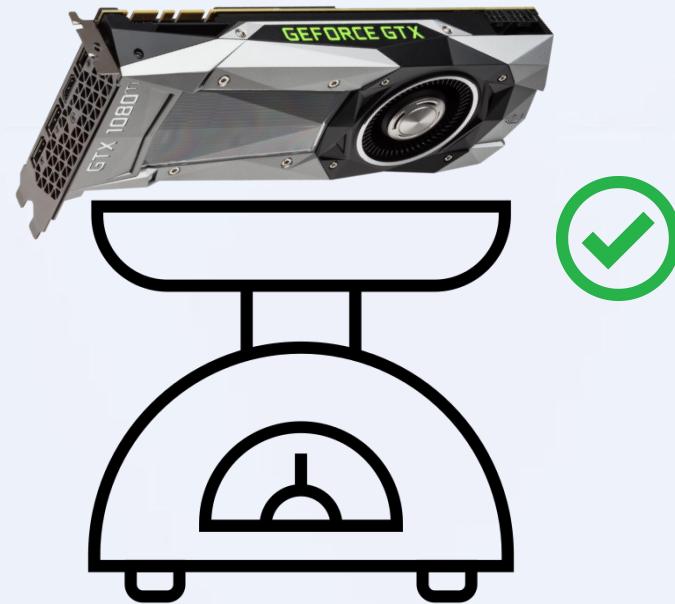
# Разведочный анализ данных(EDA)

- Визуально было обнаружено сразу много столбцов с сильно уникальными значениями( «Web Camera», «Sound Chip», etc. )



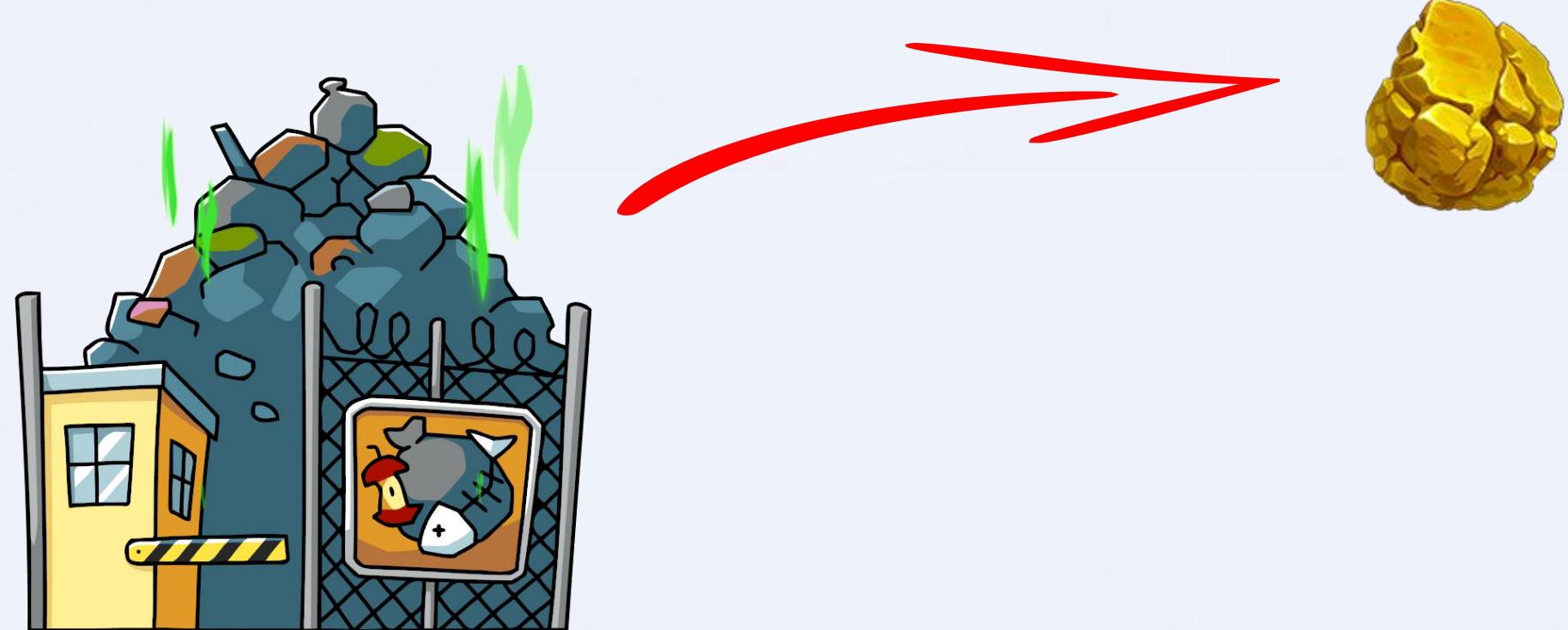
# Разведочный анализ данных(EDA)

- Были удалены колонки с большим количеством пропусков( к примеру «rj 11», «rj 45» ) с сохранением сильно значимых колонок (к примеру ( «GPU Type» )



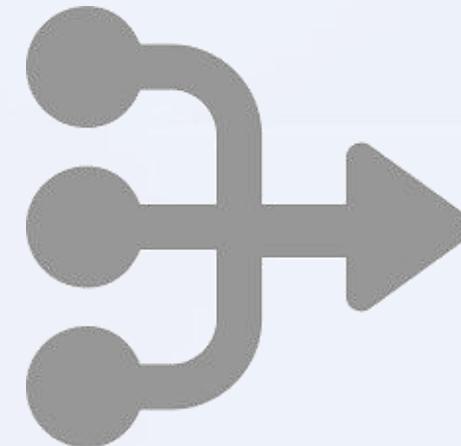
# Разведочный анализ данных(EDA)

Из некоторых сильно уникальных параметров были извлечены данные, которые могут послужить уточнению в данных, исходные данные удаляются. ( ex. «Name» -> «Vendor» )



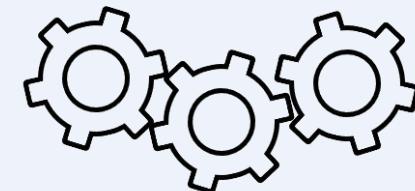
# Разведочный анализ данных(EDA)

В процессе - не обходилось и без ошибок. В погоне за наполняемостью данных, которые казались ключевыми были объединены неправильные столбцы, что привело к большому количеству уникальных значений.(Processor Name,Brand,Variant,Generation)



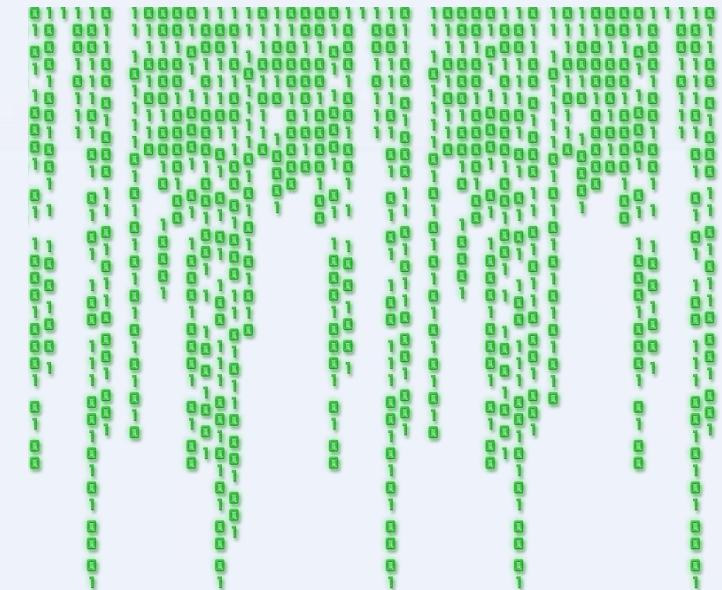
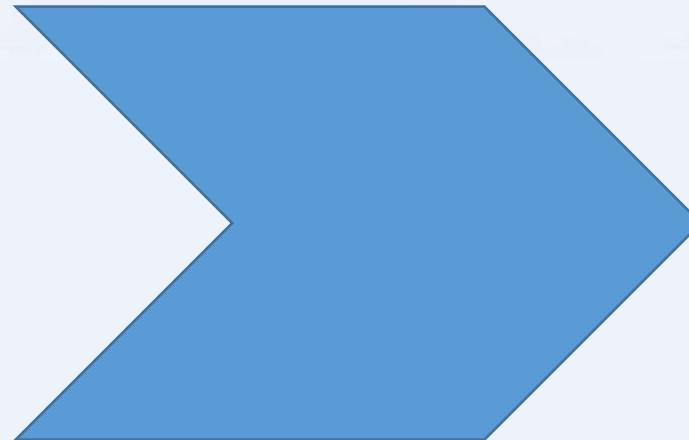
500

Unexpected Error :(



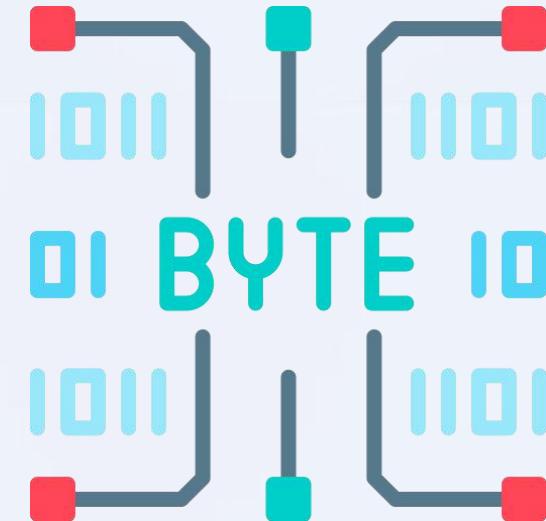
# Разведочный анализ данных(EDA)

Все колонки подобные бинарным были превращены в 0 и 1. (к примеру Finger Print Sensor - array(['No', nan, 'Yes']) ). Преимущественно nan заполнялся как отсутствие для таких колонок. Сильно уникальные колонки, где можно было сказать тоже, что есть наличие и нет - так же приводились к двоичному виду(к примеру Multi Card Slot)



# Разведочный анализ данных(EDA)

Все байтовые единицы приводились к единому стилю в числовом формате (минимальная единица в таблице MB, к ней все и приводилось)  $2\text{GB} = 2*1024$  ( к примеру SSD Capacity), А так же все остальные параметры, которые могли пройти подобную обработку, к примеру время, или количество ячеек (минимальная единица - год)  $12 \text{ мес} = 1 \text{ год}$  (к примеру warranty).



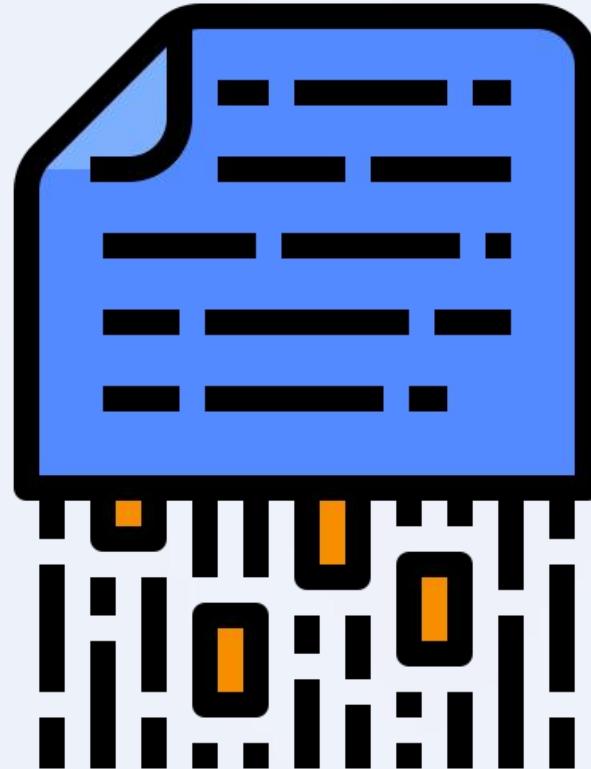
# Разведочный анализ данных(EDA)

Для пропусков выбирались оптимальные для корреляции с ценой параметры из 0, min(), max(), mean(), median(),quantile(0.3), quantile(0.4),quantile(0.6),quantile(0.7)



# Разведочный анализ данных(EDA)

Все оставшиеся object данные были перекодированы с помощью ordinal encoding. Так как в принципе все варианты кодирования давали близкие результаты.



	Ordinal_encoding	Count_encoding	Frequency_encoding
StandardScaler_r2	0.730474	0.732305	0.732305
StandardScaler_mse	-12126.403755	-12463.141494	-12463.141494
StandardScaler_rmse	-18374.782710	-18342.412909	-18342.412909
RobustScaler_r2	0.730474	0.732305	0.732305
RobustScaler_mse	-12126.403755	-12463.141494	-12463.141494
RobustScaler_rmse	-18374.782710	-18342.412909	-18342.412909

# Разведочный анализ данных(EDA)

Отшкалировав данные и опробовав на них линейную регрессию - были получены уже в принципе не такие и плохие результаты. На обучении RMSE представлял из себя примерно 16 тыс рупий, что на момент слайда было чуть более чем 10 тыс рублей(при разбросе цен от 20 тыс, до 300 тыс рупий) Но к чему еще стремиться естественно было. Модель только начала зарождаться. и даже с данными еще было что сделать



```
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_train_pred = model.predict(X_train)

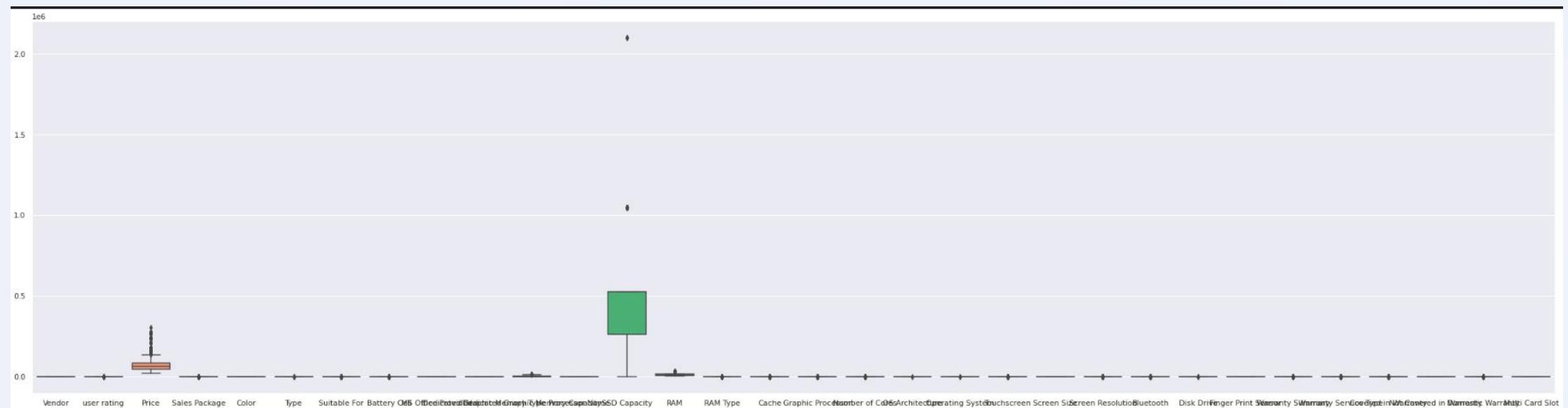
print("Test MSE = %.4f" % mean_squared_error(y_test, y_pred))
print("Train MSE = %.4f" % mean_squared_error(y_train, y_train_pred))
print("Test RMSE = %.4f" % mean_squared_error(y_test, y_pred,squared=False))
print("Train RMSE = %.4f" % mean_squared_error(y_train, y_train_pred,squared=False))

✓ 0.4s

Test MSE = 399741486.8442
Train MSE = 251365314.7239
Test RMSE = 19993.5361
Train RMSE = 15854.5046
```

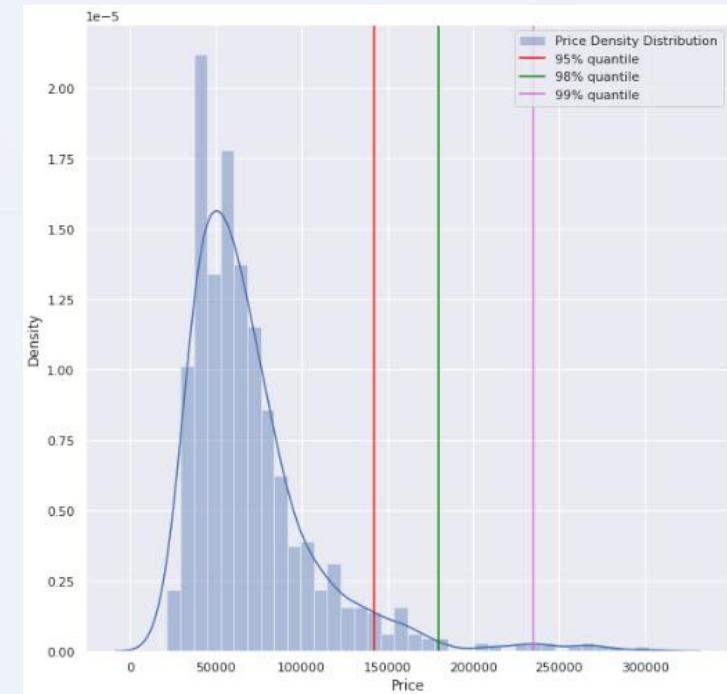
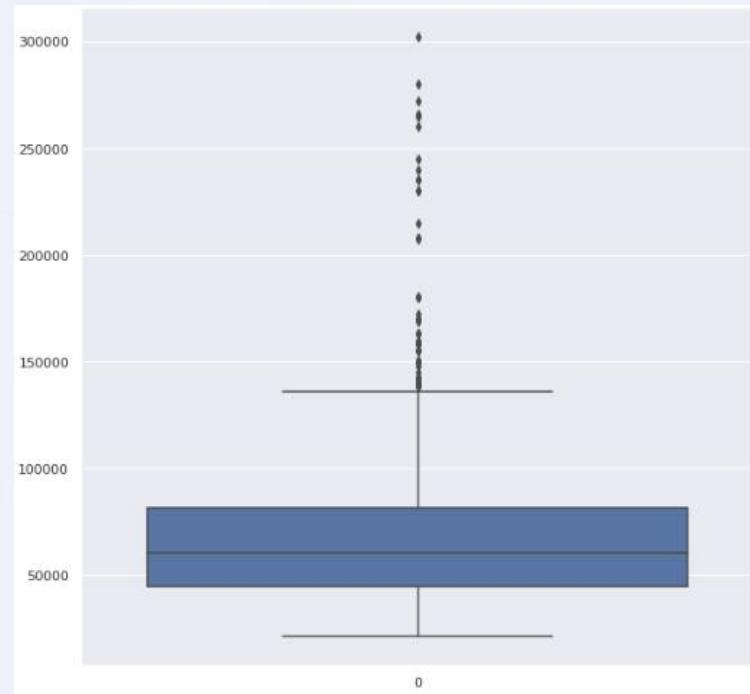
# Выявление аномалий

Для начала была построена мешаница из данных, чтобы понять где же аномалии в первую очередь зарылись. И сразу видно - есть серьзные аномалии в SSD capacity и немножко в Price. Проверим - так ли это.



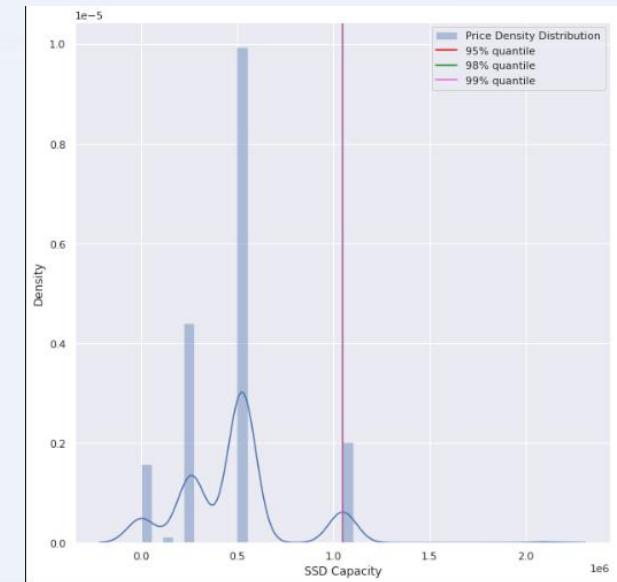
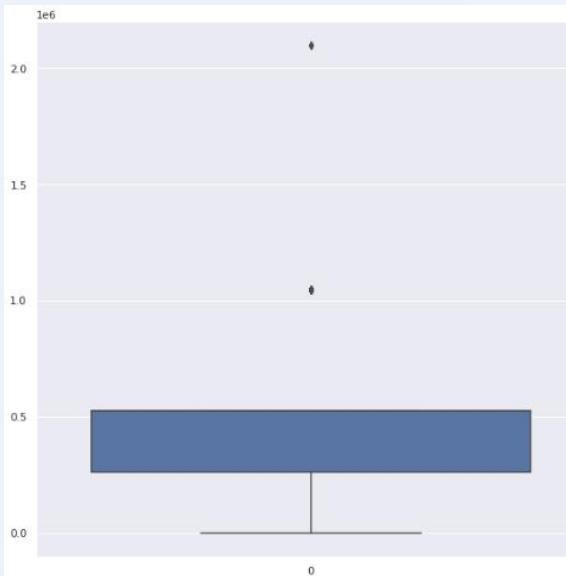
# Выявление аномалий

Действительно по цене видны отклонения, а если посмотреть на квантили - то можно увидеть что по сути справа от 98 квантили - не так и много ведь данных



# Выявление аномалий

А вот по SSD capacity таких возможностей не оказалось



# Выявление аномалий

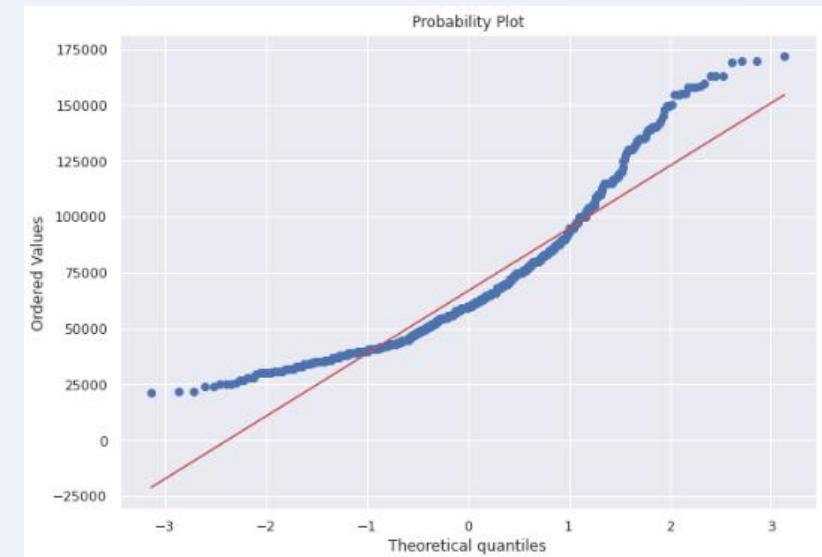
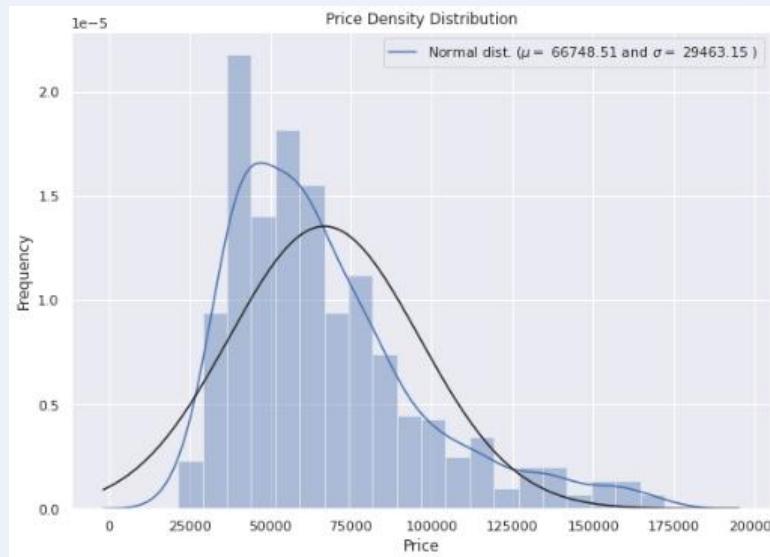
Посмотрим, на сколько улучшилась модель, после подчистки.  
А она улучшилась весьма-весьма. Почти на 30% линейная регрессия стала давать более точный ответ. Модель начинает набирать обороты.

```
check_linear()
✓ 0.7s
Test MSE = 132474421.9358
Train MSE = 159614291.1977
Test RMSE = 11509.7533
Train RMSE = 12633.8550
```



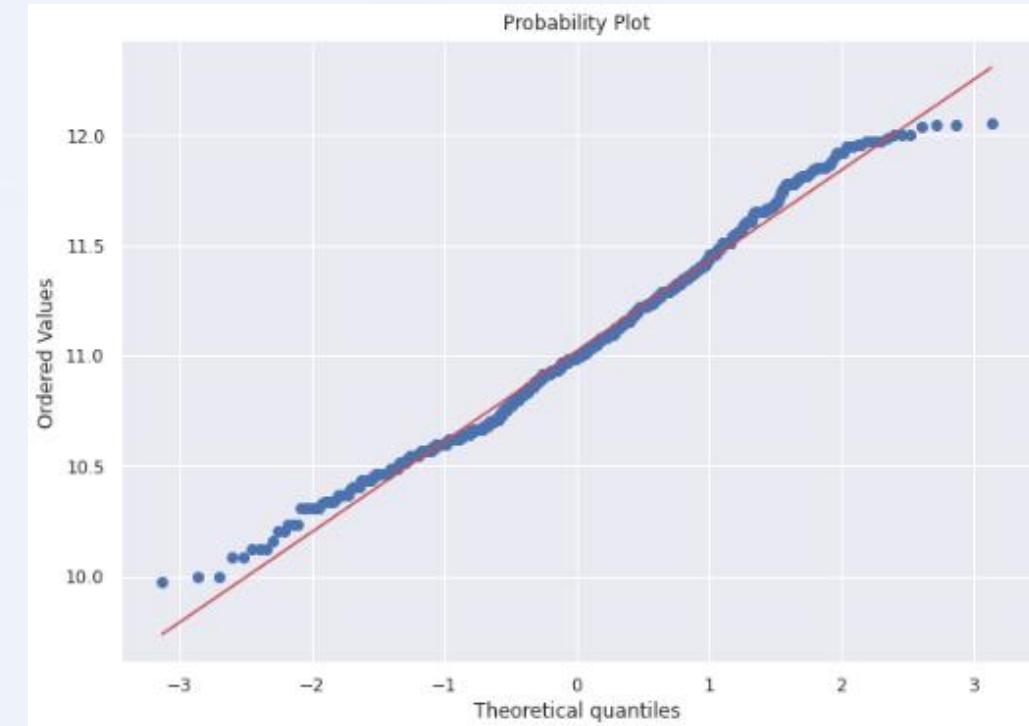
# Нормализация данных

Взглянем на распределение. И оно оказывается весьма близко к нормальному.



# Нормализация данных

Возьмем логарифм от цены и посмотрим на распределение еще раз. И увидим что оно стало заметно более близко к нормальному



# Нормализация данных

Посмотрим, на сколько улучшилась модель, после нормализации.

Ого. Почти на 50% более качественные результаты относительно прошлых.

Ошибка составляет 6000 рупий(а это уже ведь 3700 рублей примерно)

Это был бы превосходный результат... Если бы это оказалось правдой... Я забыл удалить Price =)

```
... Test MSE = 173653206.4458  
      Train MSE = 192757229.9718  
  
      Test RMSE = 13177.7542  
      Train RMSE = 13883.7038
```



```
check_linear()  
✓ 0.6s  
  
Test MSE = 36707074.6286  
Train MSE = 38385025.8720  
  
Test RMSE = 6058.6364  
Train RMSE = 6195.5650
```

# Нормализация данных

В дальнейшем при обучении будут использоваться данные логарифмированные и не логарифмированные. Для сравнения. Уж очень красиво стало выглядеть распределение после логарифмирования



# Обучение моделей

Будут обучены для сравнения следующие модели на логарифмированных и не логарифмированных данных:

- `LinearRegression`
- `Ridge`
- `Lasso`
- `ElasticNet`
- `GradientBoostingRegressor`
- `LGBMRegressor`
- `CatBoostRegressor`

# Обучение моделей

Были получены следующие результаты

	<b>LinearRegression</b>	<b>Log_LinearRegression</b>	<b>Ridge</b>	<b>Log_Ridge</b>	<b>Lasso</b>	<b>Log_Lasso</b>	<b>ElasticNet</b>	<b>Log_ElasticNet</b>	<b>GradientBoostingRegressor</b>	<b>Log_GradientBoostingRegressor</b>	<b>LGBMRegressor</b>	<b>Log_LGBMRegressor</b>	<b>CatBoostRegressor</b>	<b>Log_CatBoostRegressor</b>
Test MSE	132,474,421.94	173,653,206.45	143,202,267.38	163,787,019.48	132,359,945.00	912,177,397.72	145,782,022.15	785,220,965.69	93,519,534.42	73,384,342.84	74,325,923.02	69,709,480.73	81,090,977.37	70,933,687.41
Train MSE	159,614,291.20	192,757,229.97	168,530,605.20	191,887,487.61	160,350,009.68	895,213,650.90	171,642,896.93	766,894,963.89	17,736,313.78	20,069,323.48	39,952,549.15	40,012,341.89	38,941,891.88	41,430,720.45
Test RMSE	11,509.75	13,177.75	11,966.71	12,797.93	11,504.78	30,202.27	12,074.02	28,021.79	9,670.55	8,566.47	8,621.25	8,349.22	9,005.05	8,422.21
Train RMSE	12,633.85	13,883.70	12,981.93	13,852.35	12,662.94	29,920.12	13,101.26	27,692.87	4,211.45	4,479.88	6,320.80	6,325.53	6,240.34	6,436.67
Train MAE	8,849.05	9,006.17	8,984.02	8,834.80	8,843.99	21,881.46	9,030.76	20,012.82	3,152.42	3,193.80	4,446.64	4,216.39	4,741.89	4,576.37
Test MAE	8,826.15	9,377.40	9,119.62	9,080.72	8,817.29	21,831.03	9,186.29	19,931.00	6,405.95	5,667.92	6,235.11	5,849.46	6,410.49	6,060.95

**Самые лучшие модели - модели бустинга, они показывает среднюю абсолютную ошибку до 6к в рупиях на тестовых данных, что в принципе очень даже ничего. В рублях это бы получилось примерно 4,5 тыс**

	Min Value	Min Model	Max Value	Max Model
Test MSE	69709480.73	Log_LGBMRegressor	912177397.7	Log_Lasso
Train MSE	17736313.78	GradientBoostingRegressor	895213650.9	Log_Lasso
Test RMSE	8349.22	Log_LGBMRegressor	30202.27	Log_Lasso
Train RMSE	4211.45	GradientBoostingRegressor	29920.12	Log_Lasso
Train MAE	3152.42	GradientBoostingRegressor	21881.46	Log_Lasso
Test MAE	5667.92	Log_GradientBoostingRegressor	21831.03	Log_Lasso

--- BONUS ---



Lenovo

ASUS®

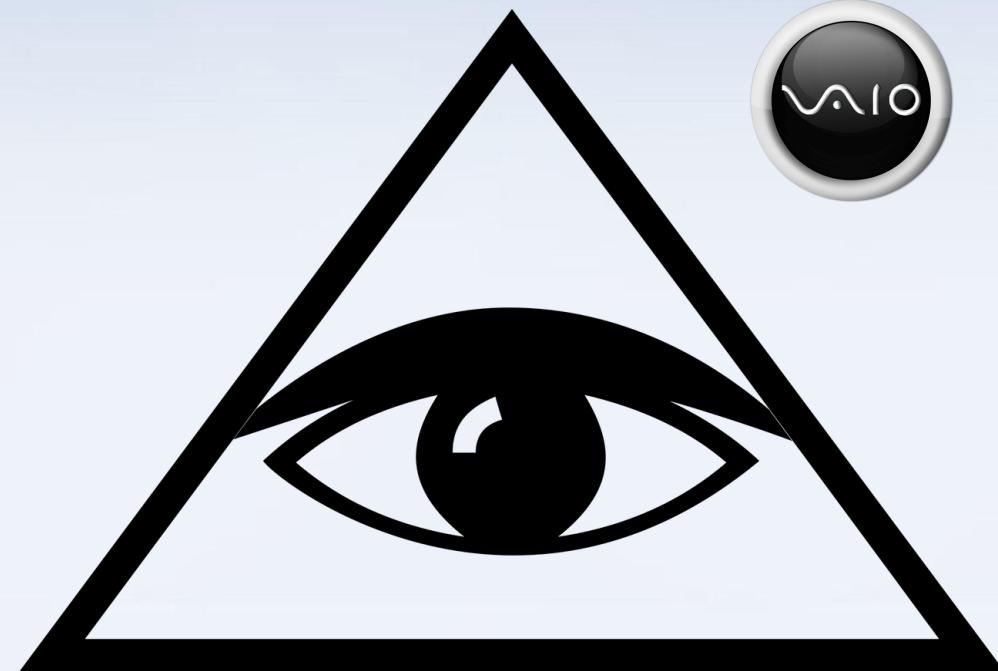
R realme



AVITA

acer

Infinix



ML-предсказание  
производится на  
ноутбуке

# --- BONUS ---

VL

Дойдя до этой части я с удивлением узнал, что некоторые вендоры оказались задвоены(Lenovo - lenovo к примеру). Пришлось вернуться в начало и исправить это, а заодно еще раз пропустить весь обсчет ради интереса

	LinearRegression	Log_LinearRegression	Ridge	Log_Ridge	Lasso	Log_Lasso	ElasticNet	Log_ElasticNet	GradientBoostingRegressor	Log_GradientBoostingRegressor	LGBMRegressor	Log_LGBMRegressor	CatBoostRegressor	Log_CatBoostRegressor
Test MSE	132,352,470.51	173,614,879.84	143,086,372.86	163,763,569.72	132,224,335.70	912,177,397.72	145,674,576.92	785,220,965.69	91,188,326.56	76,332,441.08	78,372,726.54	71,517,963.22	82,555,952.35	72,279,382.28
Train MSE	159,523,837.02	192,730,846.99	168,455,315.78	191,924,272.68	160,259,866.31	895,213,650.90	171,576,787.02	766,894,963.89	19,175,660.34	21,080,665.55	39,226,109.11	41,808,698.85	38,874,816.28	44,267,034.08
Test RMSE	11,504.45	13,176.30	11,961.87	12,797.01	11,498.88	30,202.27	12,069.57	28,021.79	9,549.26	8,736.84	8,852.84	8,456.83	9,086.03	8,501.73
Train RMSE	12,630.27	13,882.75	12,979.03	13,853.67	12,659.38	29,920.12	13,098.73	27,692.87	4,379.00	4,591.37	6,263.08	6,465.96	6,234.97	6,653.35
Train MAE	8,845.25	9,006.22	8,980.89	8,834.66	8,840.66	21,881.46	9,030.81	20,012.82	3,300.05	3,222.20	4,386.31	4,267.52	4,748.99	4,700.64
Test MAE	8,826.79	9,378.13	9,117.04	9,080.38	8,814.49	21,831.03	9,183.27	19,931.00	6,345.43	5,897.87	6,339.42	5,912.83	6,407.86	6,130.22

в целом, что любопытно - уточнение данных дало в принципе незаметный(десятие доли рупий), но тем не менее негативный эффект =)

Без какой либо дополнительной обработки данных (взяты данные до encode, чтобы сохранить Vendor, закодированы тем же способом что и для регрессии, а так же прошкалированы).

Обучал модели:

- KNeighborsClassifier
- GradientBoostingClassifier
- CatBoostClassifier

# --- BONUS ---

VL

И что мы видим =). На тестовых данных вполне себе неплохо отрабатывают модели. А значит по цене и содержимому вполне можно догадаться о производителе.

	KNeighborsClassifier	Gradient Boosting Classifier	CatBoost Classifier
f1_score_train	1	1	0.99
f1_score_test	0.79	0.89	0.85
accuracy_score_train	1	1	0.99
accuracy_score_test	0.8	0.89	0.86
recall_score_train	1	1	0.99
recall_score_test	0.8	0.89	0.86

The End