AIML CA1

Tan Yu Hoe	P2026309
Diploma in Applied AI and Analytics	DAAA/FT/2A/04
School of Computing	Singapore Polytechnic
tanyh.20@ichat.sp.edu.sg	

AIML	ST1511
CA1 Assignment	

Part A: Mushroom Classification

Prediction Task

The prediction task is to create a machine learning model to **predict a mushroom's edibility**, either edible or poisonous, based on the mushroom's attributes.

Output Variable

The output variable is a **two-class** label - **edible** or **poisonous**. Edible (e) is defined the mushroom would cause no hard when consumed; whereas poisonous (p) refers to the mushroom will cause harmful effects when eaten.

The Data

This data set includes descriptions corresponding to 23 species of gilled mushrooms. Each species is identified to be either edible or poisonous.

Number of Instance: 8,124
Number of Attributes: 22 (all nominal valued)
Missing values are denoted as '?'

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population	habitat
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s	u
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n	g
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n	m
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	s	u
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	a	g
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8119	e	k	s	n	f	n	a	c	b	y	...	s	o	o	p	o	o	p	b	c	l
8120	e	x	s	n	f	n	a	c	b	y	...	s	o	o	p	n	o	p	b	v	l
8121	e	f	s	n	f	n	a	c	b	n	...	s	o	o	p	o	o	p	b	c	l
8122	p	k	y	n	f	y	f	c	n	b	...	k	w	w	p	w	o	e	w	v	l
8123	e	x	s	n	f	n	a	c	b	y	...	s	o	o	p	o	o	p	o	c	l

8124 rows × 23 columns

Attribute Information

1. cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
2. cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s
3. cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r, pink=p,purple=u,red=e,white=w,yellow=y
4. bruises?: bruises=t,no=f
5. odor: almond=a,anise=l,creosote=c,fishy=y,foul=f, musty=m,none=n,pungent=p,spicy=s
6. gill-attachment: attached=a,descending=d,free=f,notched=n
7. gill-spacing: close=c,crowded=w,distant=d
8. gill-size: broad=b,narrow=n
9. gill-color: black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e, white=w,yellow=y
10. stalk-shape: enlarging=e,tapering=t
11. stalk-root: bulbous=b,club=c,cup=u,equal=e, rhizomorphs=z,rooted=r,missing=?
12. stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s
13. stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s
14. stalk-color-above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
15. stalk-color-below-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
16. veil-type: partial=p,universal=u
17. veil-color: brown=n,orange=o,white=w,yellow=y
18. ring-number: none=n,one=o,two=t
19. ring-type: cobwebby=c,evanescent=e,flaring=f,large=l, none=n,pendant=p,sheathing=s,zone=z
20. spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r, orange=o,purple=u,white=w,yellow=y
21. population: abundant=a,clustered=c,numerous=n, scattered=s,several=v,solitary=y
22. habitat: grasses=g,leaves=l,meadows=m,paths=p, urban=u,waste=w,woods=d

Exploratory Data Analysis

Topics I want to cover:

- Data Profile
- Warnings
- Feature Selection

I have hosted a copy of my data profile on GitHub so feel free to scan the QR or access the link below.

URL:
https://lekekbot.github.io/mushroom-profile/

Credits: this is not my GitHub account, special thanks to Bryan from DIT

Scan Here!

Data Profile

Overview | Warnings 47 | Reproduction

Dataset statistics

Number of variables	23
Number of observations	8124
Missing cells	2480
Missing cells (%)	1.3%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	1.4 MiB
Average record size in memory	184.0 B

Variable types

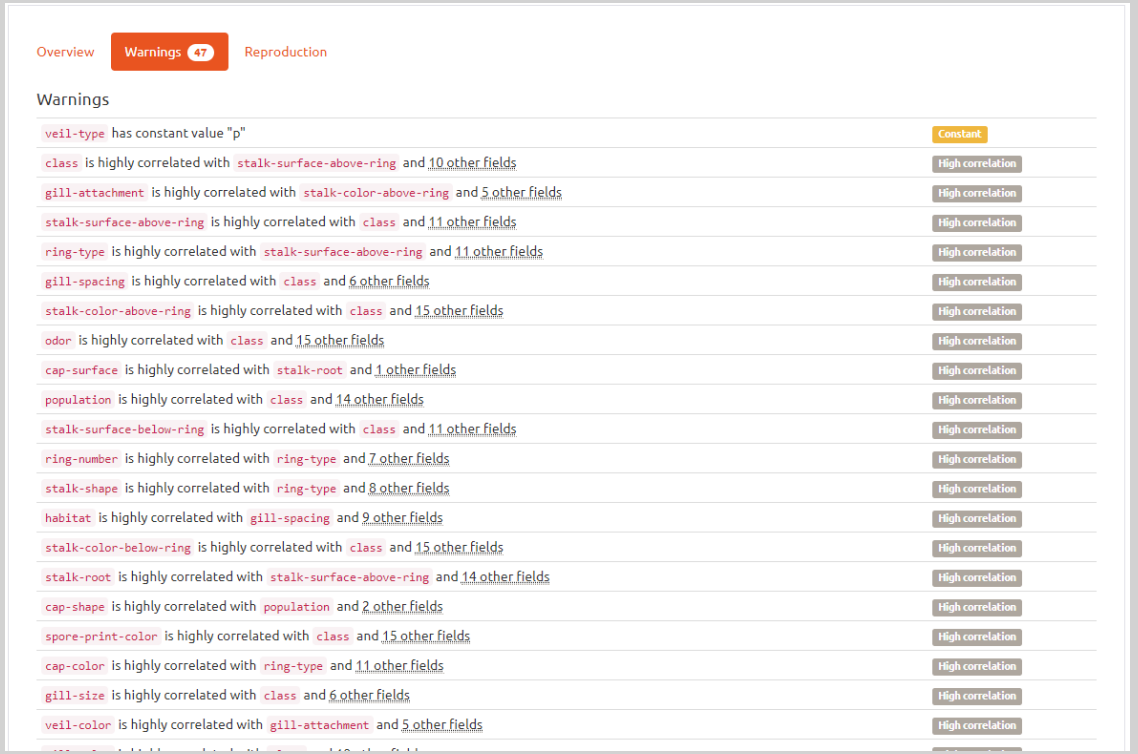
Categorical	22
Boolean	1

Warnings

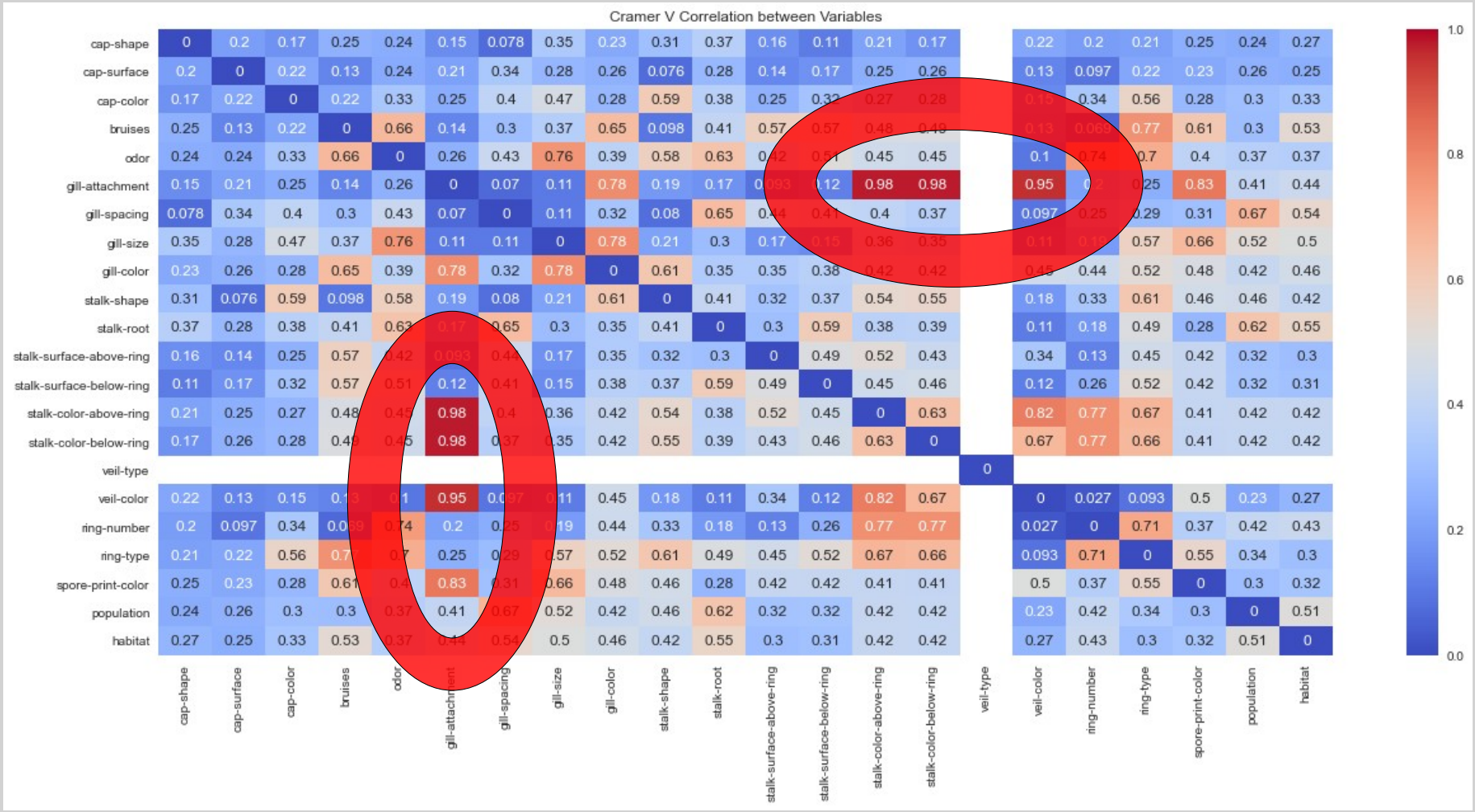
Profile Reports shows **47 Warnings**

Key points:

- All of the columns are categorical data
  - requires encoding
- “veil-type” contains only one class
  - drop column since there is no variance
- “stalk-root” has 30.5% missing data
  - either impute or drop the entire column
- high correlation between features
  - have to check for multicollinearity



Multicollinearity Feature Selection



Multicollinearity Feature Selection

Key Points to take down:

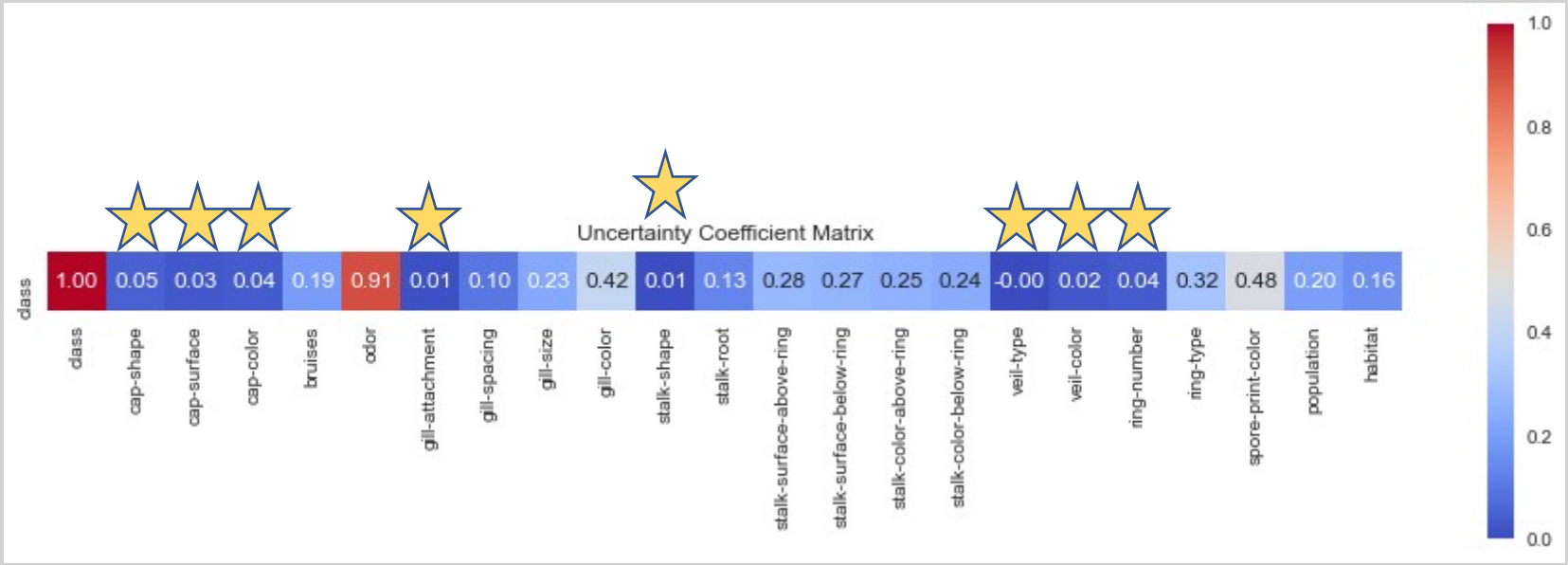
- **“gill-attachment”** has very high correlation with other attributes, where V > 0.9
- these attributes are color-related attributes

Action Plan

- drop column as it might reduce statistical significance on other attributes

	Correlations	Features
0	0.977312	[gill-attachment, stalk-color-above-ring]
1	0.977312	[gill-attachment, stalk-color-below-ring]
2	0.954963	[gill-attachment, veil-color]
3	0.977312	[stalk-color-above-ring, gill-attachment]
4	0.977312	[stalk-color-below-ring, gill-attachment]
5	0.954963	[veil-color, gill-attachment]

Theil's U – Univariate Feature Selection



Theil’s U – Univariate Feature Selection

Key Points to take down:

- These columns have low correlation with edibility, where U < 0.05
- cap-shape, cap-surface, cap-color, gill-attachment, stalk-shape, veil-type, veil-color, ring-number

Action Plan

- Drop all these columns show below

	cap-shape	cap-surface	cap-color	gill-attachment	stalk-shape	veil-type	veil-color	ring-number
class	0.048842	0.028617	0.036083	0.014178	0.007524	-1.603208e-16	0.023839	0.038489

Displaying Features

After dropping many columns, I think it is safe to say we can proceed on to feature engineering.

[21]

```
df.drop('class', axis=1).head()
```

	bruises	odor	gill-spacing	gill-size	gill-color	stalk-root	stalk-surface-above-ring	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	ring-type	spore-print-color	population	habitat
0	t	p	c	n	k	e	s	s	w	w	p	k	s	u
1	t	a	c	b	k	c	s	s	w	w	p	n	n	g
2	t	l	c	b	n	c	s	s	w	w	p	n	n	m
3	t	p	c	n	n	e	s	s	w	w	p	k	s	u
4	f	n	w	b	k	e	s	s	w	w	e	n	a	g

Feature Engineering / Preprocessing

For Feature Engineering, I will be covering these technique applied to this dataset:

- Data Partition
- Imputation
- Categorical Encoding

Nothing fancy to do on this dataset :/

Features and Target variable

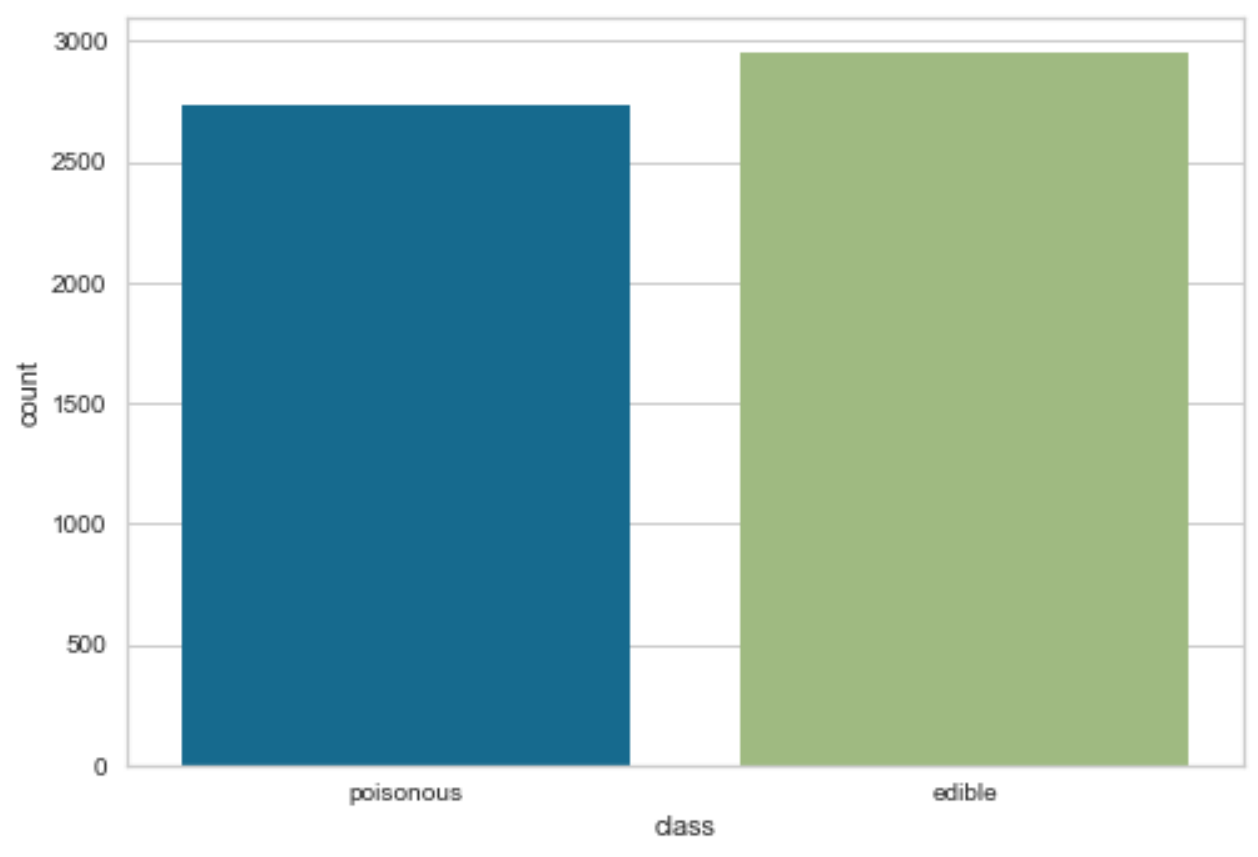
```
# defining features, X, and target variable, y
X = df.drop(columns='class')
y = df['class']
print(X.shape, y.shape)

>> (8124, 14) (8124,)
```

Data Partition

```
from sklearn.model_selection import train_test_split
# using sklearn method to split the data into training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3)
# getting the order
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

Data Partition

The two class count are quite close to each other, so no class balancing is needed.



Imputation

I will be using `SimpleImputer` from `sklearn.impute` to impute missing values with the mode value.

```
pd.DataFrame(imp.fit_transform(X_train), columns=X_train.columns).isna().sum()
```

```
bruises                   0
odor                      0
gill-spacing              0
gill-size                 0
gill-color                0
stalk-root                0
stalk-surface-above-ring  0
stalk-surface-below-ring  0
stalk-color-above-ring    0
stalk-color-below-ring    0
ring-type                 0
spore-print-color         0
population                0
habitat                   0
dtype: int64
```

One Hot Encoding

Probably the most magical thing we learnt during Week 2.

Why One Hot Encoding?
The metadata states that all variables are nominally valued, as such we have to assume all variables to be nominal data. Since there is no order, I have to use sklearn.preprocessing.OneHotEncoder method to encode the data.

	bruises_t	odor_c	odor_f	odor_l	odor_m	odor_n	odor_p	odor_s	odor_y	gill-spacing_w	...	population_n	population_s	population_v	population_y	habitat_g	habitat_l	habitat_m	habitat_p	habitat_u	habitat_w
0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
1	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows x 70 columns

Label Encoder

Label encoding the target variable, 0 = ‘edible’, 1 = ‘poisonous’.

```
# Initiate label encoder
label_encoder = LabelEncoder()
# Fitting to Training Data and Apply Transfomation
y_train = label_encoder.fit_transform(y_train_p)
y_test = label_encoder.transform(y_test_p)
label_encoder.classes_

>> array(['edible', 'poisonous'], dtype=object)
```

Model Selection

My plan for model selection is to fit through a collection of models at default hyperparameters with a pipeline. Then I will make my decision based on the K-Fold Cross Validati

```
scoring = ['accuracy', 'balanced_accuracy', 'f1', 'roc_auc']
for name, model in models:
  pipeline = Pipeline(
    steps=[
      ('SimpleImputer', imp),
      ('OneHotEncoder', onehot),
      (name, model)
    ]
  )
  cross_validate(pipeline, X_train, y_train,
                 scoring=scoring, n_jobs=4, verbose=4)
```

Model Selection

Most of the models I have preselected has 1.0 accuracy.

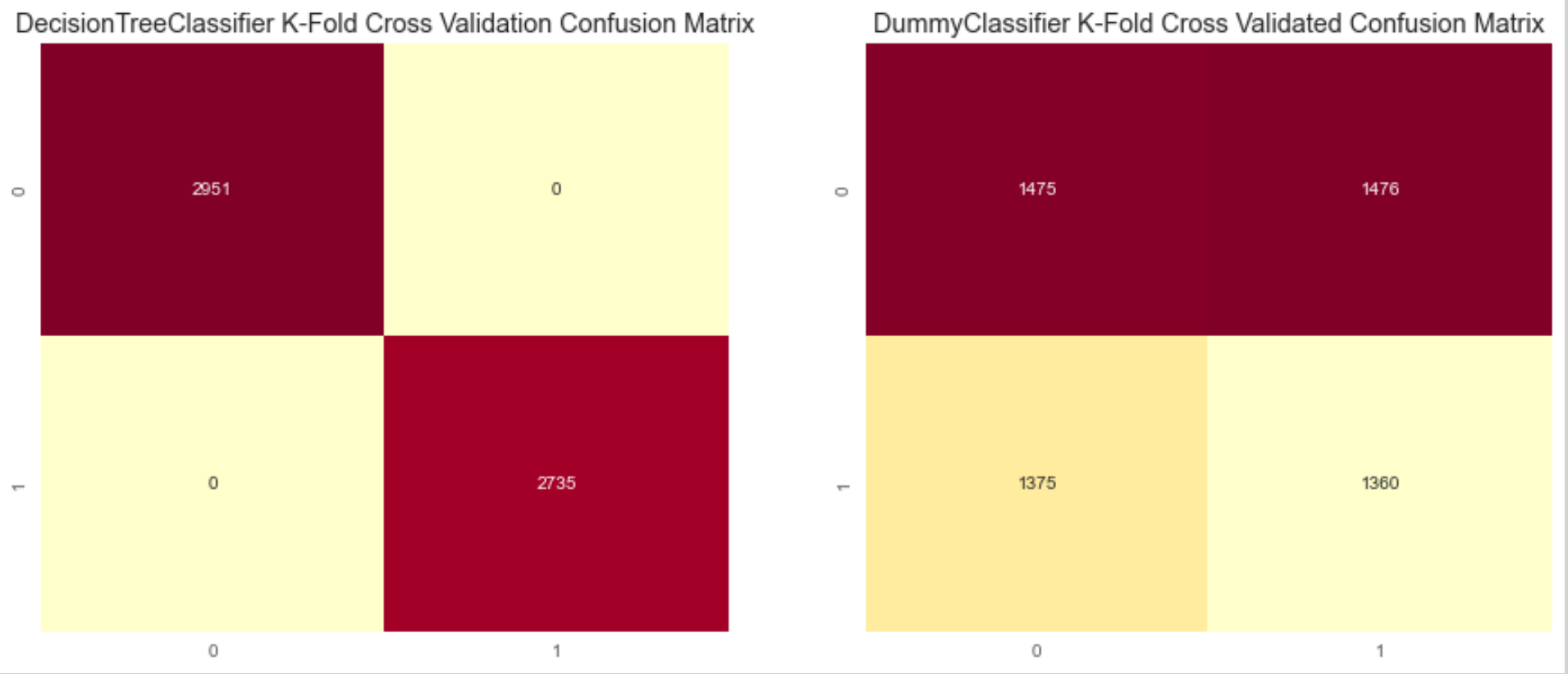
Since there is a dilemma in choosing learning algorithms, I decided to go with the simplest model with the best explain ability, Decision Tree Classifier.

	fit_time	score_time	test_accuracy	test_balanced_accuracy	test_f1	test_roc_auc
DecisionTreeClassifier	0.036299	0.015959	1.000000	1.000000	1.000000	1.000000
RandomForestClassifier	0.294108	0.050652	1.000000	1.000000	1.000000	1.000000
AdaBoostClassifier	0.549287	0.079787	1.000000	1.000000	1.000000	1.000000
GradientBoostingClassifier	1.052442	0.024467	1.000000	1.000000	1.000000	1.000000
Perceptron	0.043483	0.032352	1.000000	1.000000	1.000000	1.000000
Linear SVC	0.139350	0.035562	1.000000	1.000000	1.000000	1.000000
Polynomial SVC	0.359826	0.079387	1.000000	1.000000	1.000000	1.000000
ExtraTreesClassifier	0.372700	0.067840	1.000000	1.000000	1.000000	1.000000
MLPClassifier	1.763953	0.037759	1.000000	1.000000	1.000000	1.000000
CalibratedClassifierCV	0.203737	0.043484	1.000000	1.000000	1.000000	1.000000
SGDClassifier	0.048475	0.029321	1.000000	1.000000	1.000000	1.000000
KNeighborsClassifier	0.045479	0.701378	0.999472	0.999452	0.999450	1.000000
RBF SVC	0.421568	0.208354	0.999472	0.999452	0.999450	1.000000
RidgeClassifier	0.050712	0.033710	0.999120	0.999086	0.999084	1.000000
RidgeClassifierCV	0.109710	0.035540	0.999120	0.999086	0.999084	1.000000
LogisticRegression	0.112658	0.034307	0.999120	0.999086	0.999084	0.999987
GaussianNB	0.034707	0.025941	0.974147	0.975027	0.973790	0.996524
Sigmoid SVC	0.334529	0.079947	0.964649	0.964138	0.962766	0.985698

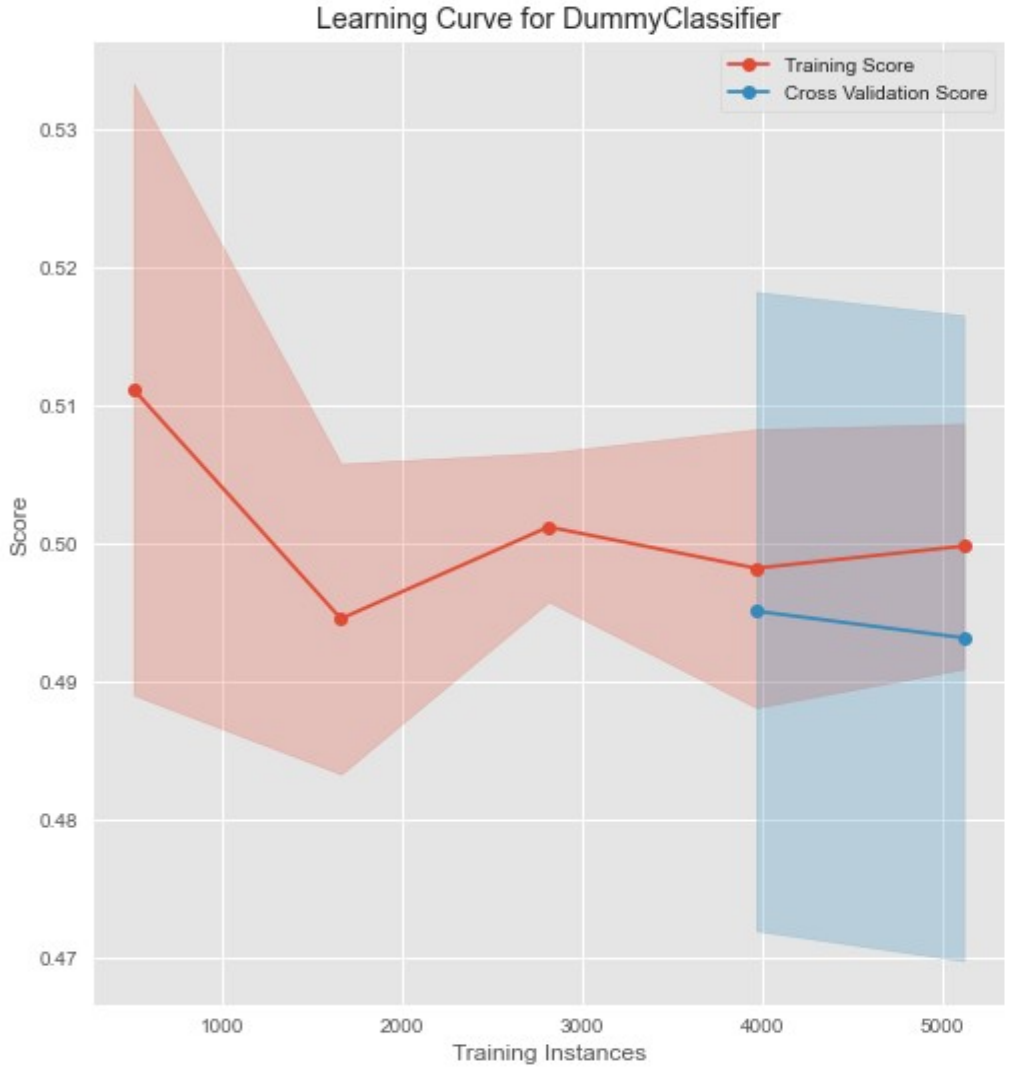
Model Selection

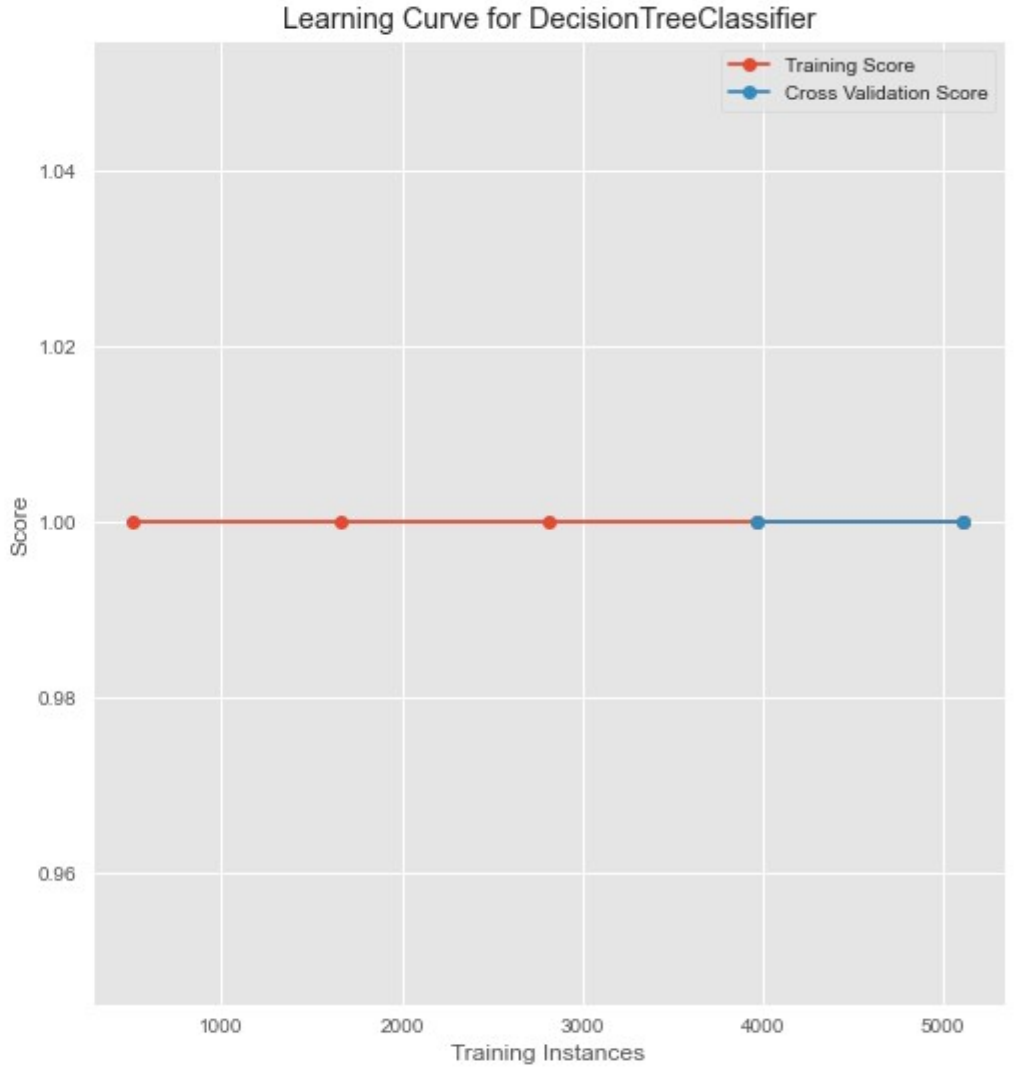
```
CART = Pipeline(
    steps=[
      ('Imputation', imp),
      ('OneHot Encoding', onehot),
      ('DecisionTreeClassifier', DecisionTreeClassifier())
    ]
)
CART.fit(X_train, y_train)
```

Comparing to Baseline – Confusion Matrix



Comparing to Baseline – Learning Curve





Hyperparameter Tuning

Since the dataset is relatively small and this is a two-class classification prediction task, we can afford to use GridSearchCV to perform an exhaustive search for the best hyperparameters. Since we are using DecisionTreeClassifier, I will be listing down the parameters that I will be tuning.

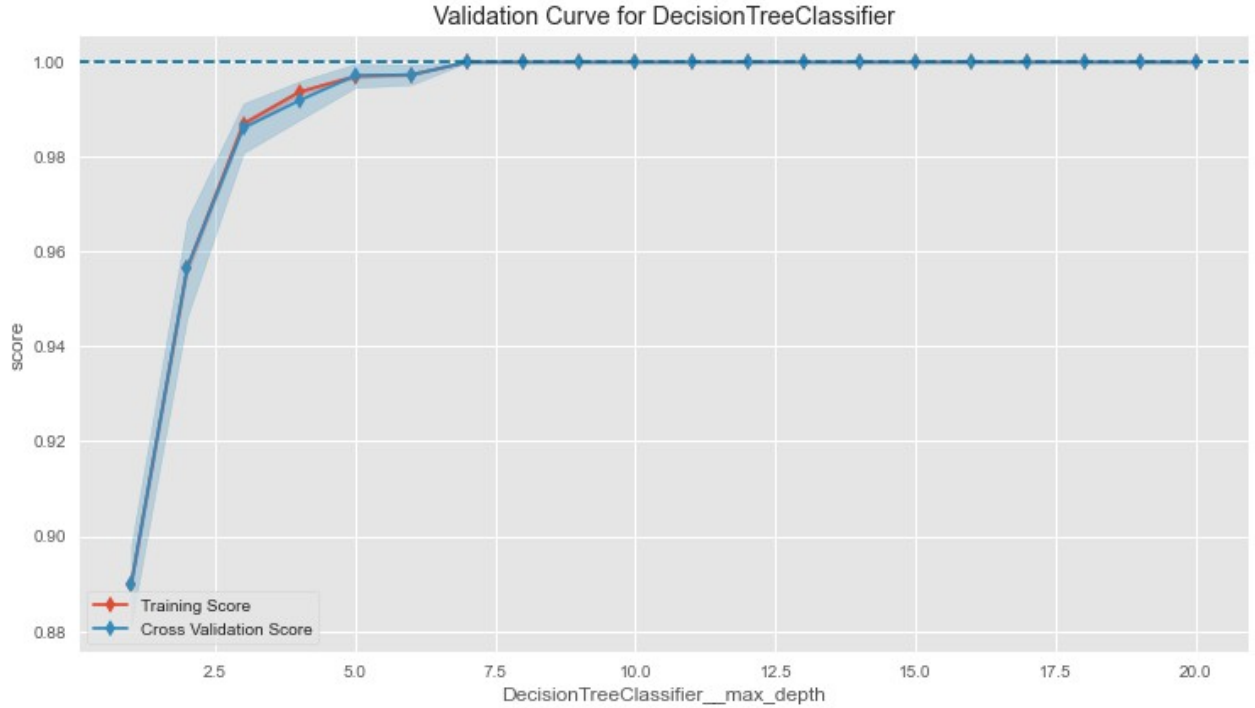
- criterion - measure the quality of the split
- max_depth - maximum depth of the tree, basically how big can the tree grow
- max_leaf_nodes - stopping criteria for number of leaves

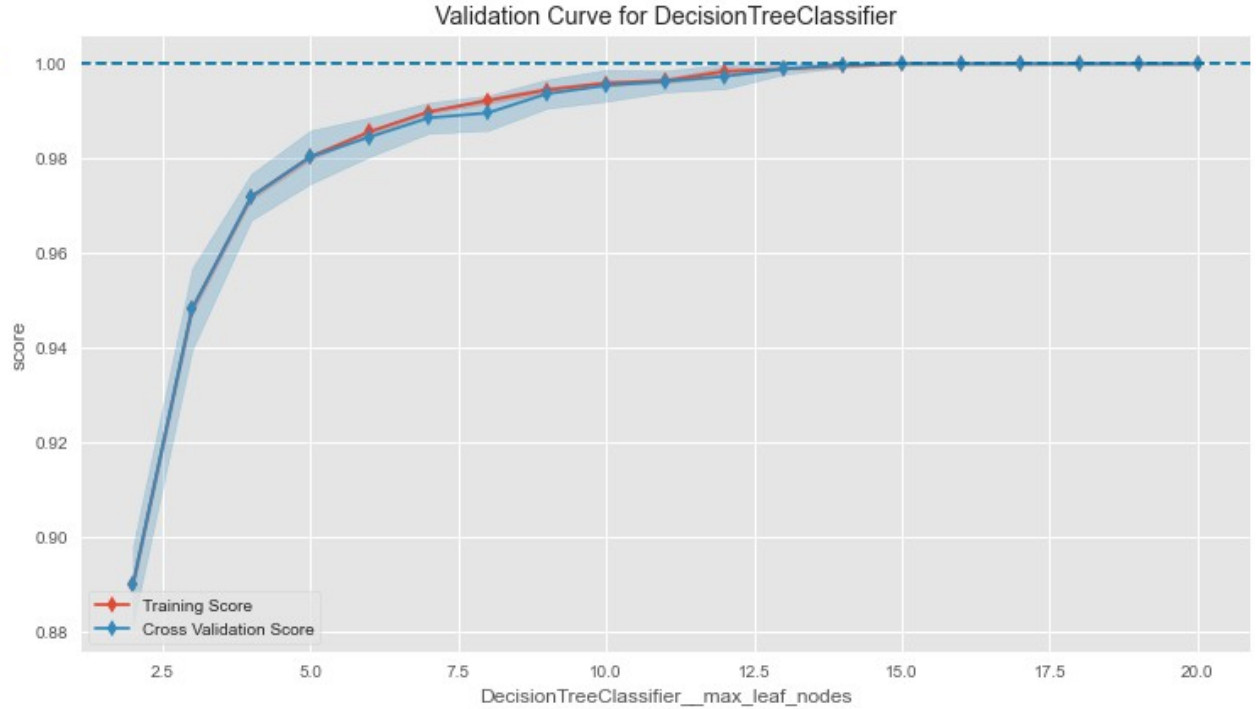
Hyperparameter Tuning

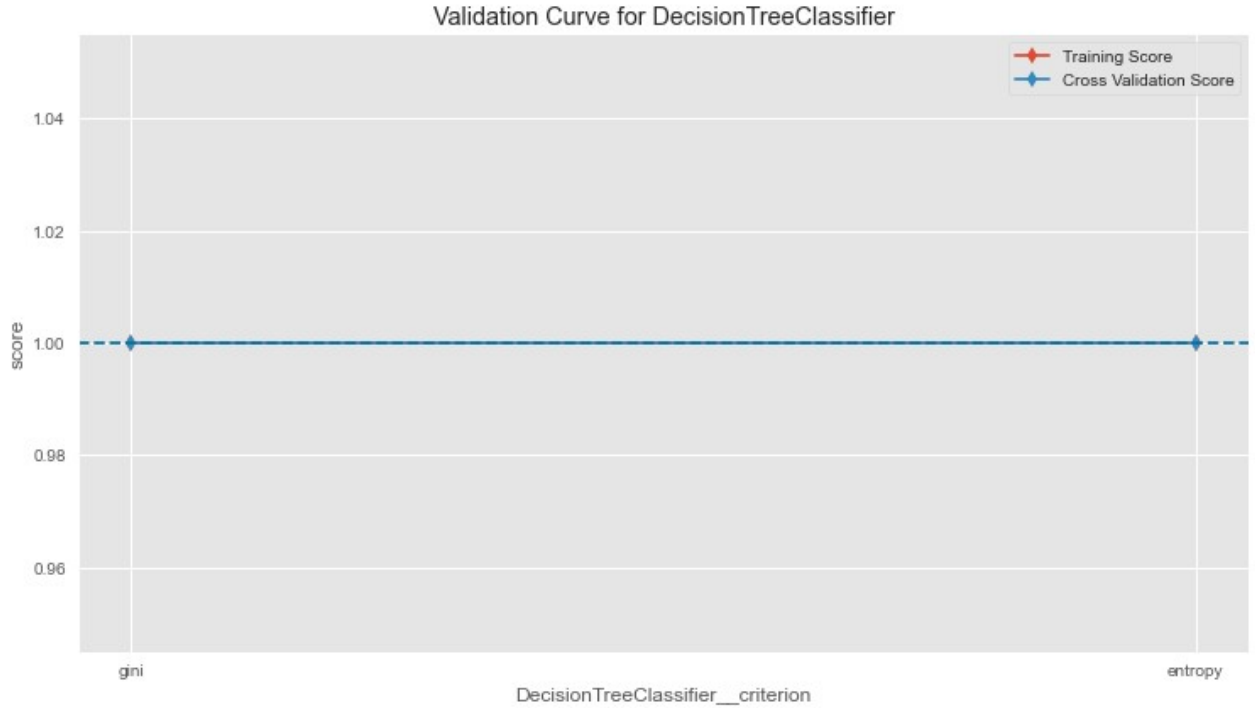
[35]

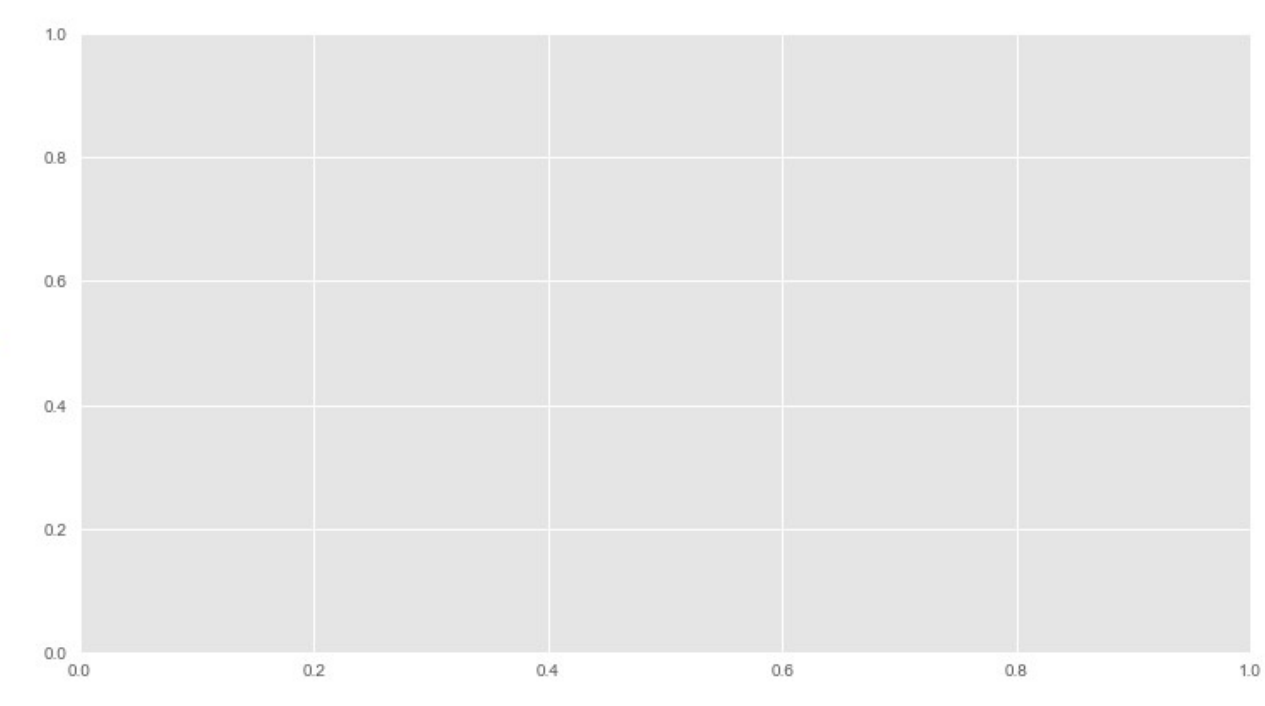
```
filterwarnings('ignore')
# Create the parameter grid
params_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': np.arange(5, 15),
    'max_leaf_nodes': np.arange(10, 16)
}
# Creating a model based on the pipeline
grid_search = Pipeline(
    steps=[
        ('SimpleImputer', imp),
        ("OneHotEncoder", onehot),
        ("GridSearchCV", GridSearchCV(
            DecisionTreeClassifier(min_samples_split=2, min_samples_leaf=1),
            params_grid,
            cv=5,
            verbose=2,
            n_jobs=4,
            scoring='accuracy'
            )
        )
    ]
)
# Fitting Model
grid_search.fit(X_train, y_train)
print(grid_search.named_steps['GridSearchCV'].best_estimator_)
print(grid_search.named_steps['GridSearchCV'].best_params_)
print(grid_search.named_steps['GridSearchCV'].best_score_)
```

```
Fitting 5 folds for each of 120 candidates, totalling 600 fits
DecisionTreeClassifier(max_depth=7, max_leaf_nodes=14)
{'criterion': 'gini', 'max_depth': 7, 'max_leaf_nodes': 14}
1.0
```

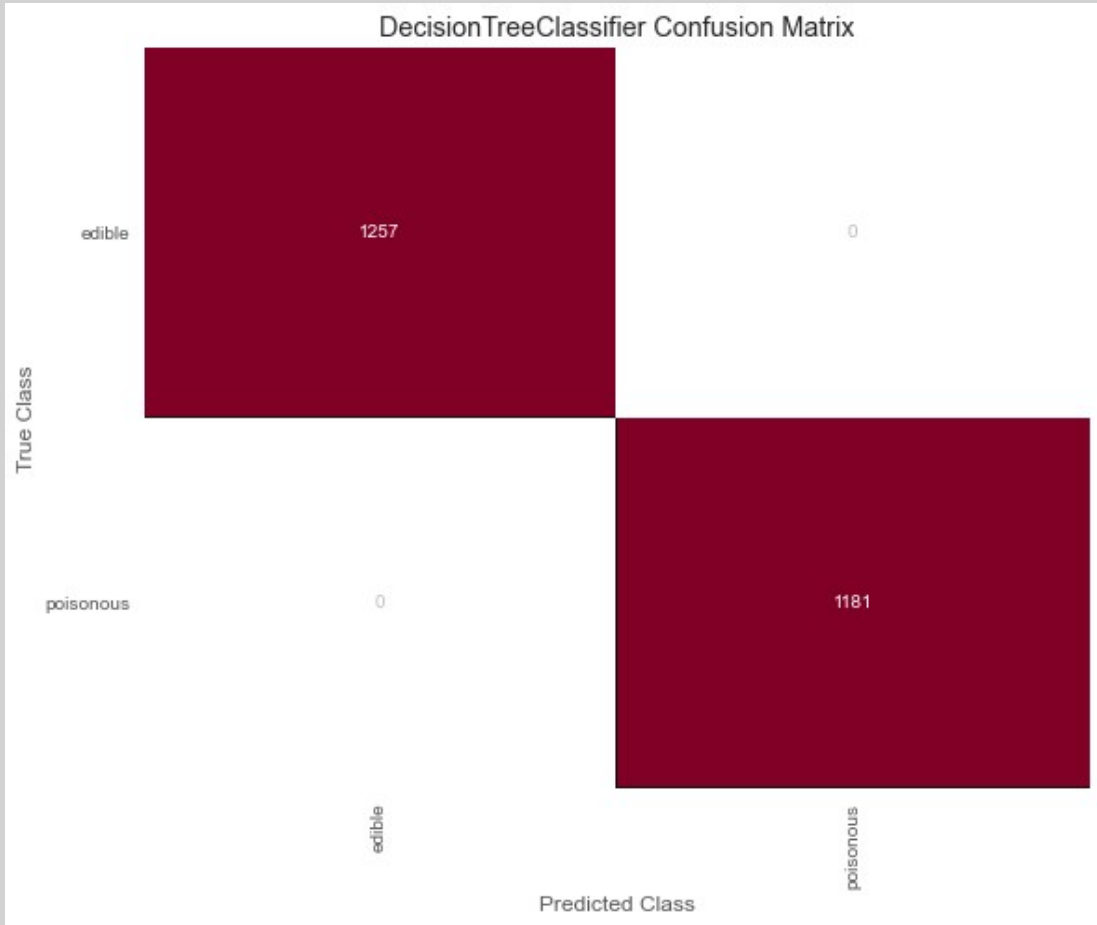

Validation Curve for DecisionTreeClassifier
score
1.00
0.98
0.96
0.94
0.92
0.90
0.88
Training Score
Cross Validation Score
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
DecisionTreeClassifier__max_depth

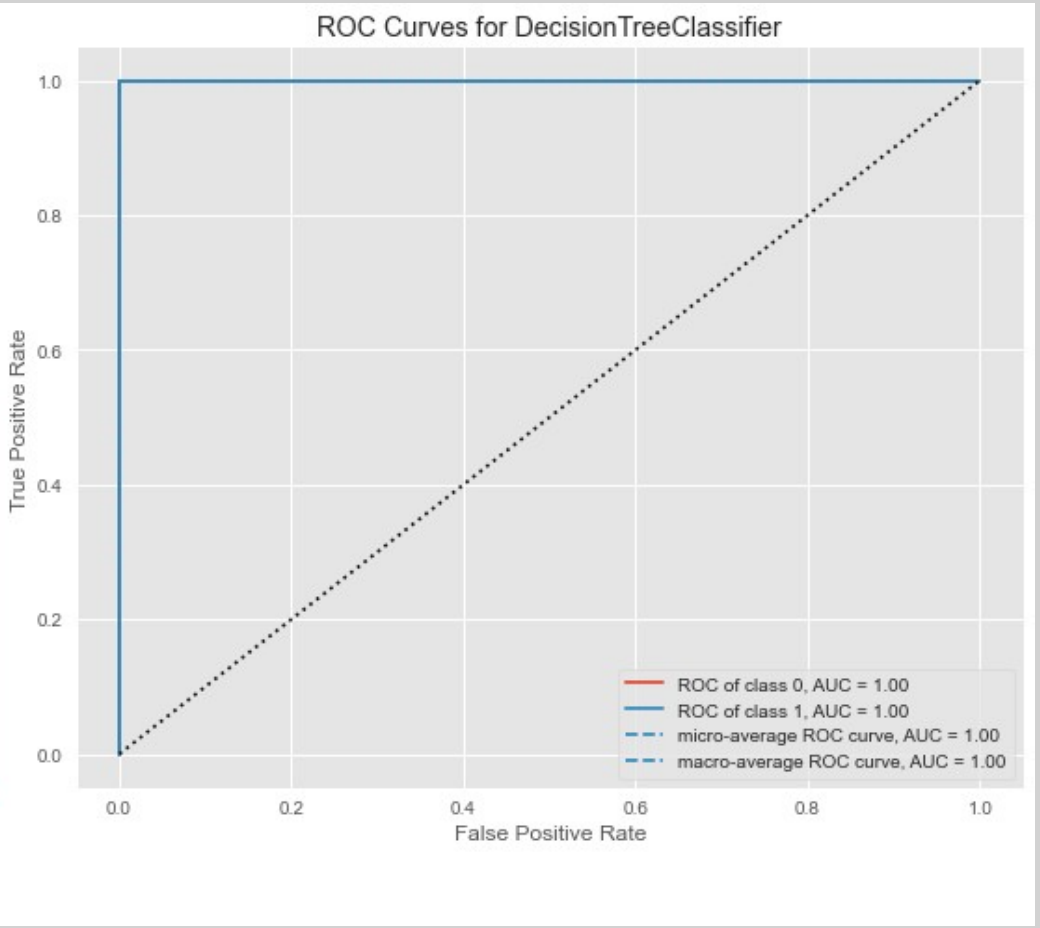

Validation Curve for DecisionTreeClassifier
score
1.00
0.98
0.96
0.94
0.92
0.90
0.88
Training Score
Cross Validation Score
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
DecisionTreeClassifier__max_leaf_nodes


Validation Curve for DecisionTreeClassifier
Training Score
Cross Validation Score
score
1.04
1.02
1.00
0.98
0.96
gini
entropy
DecisionTreeClassifier__criterion

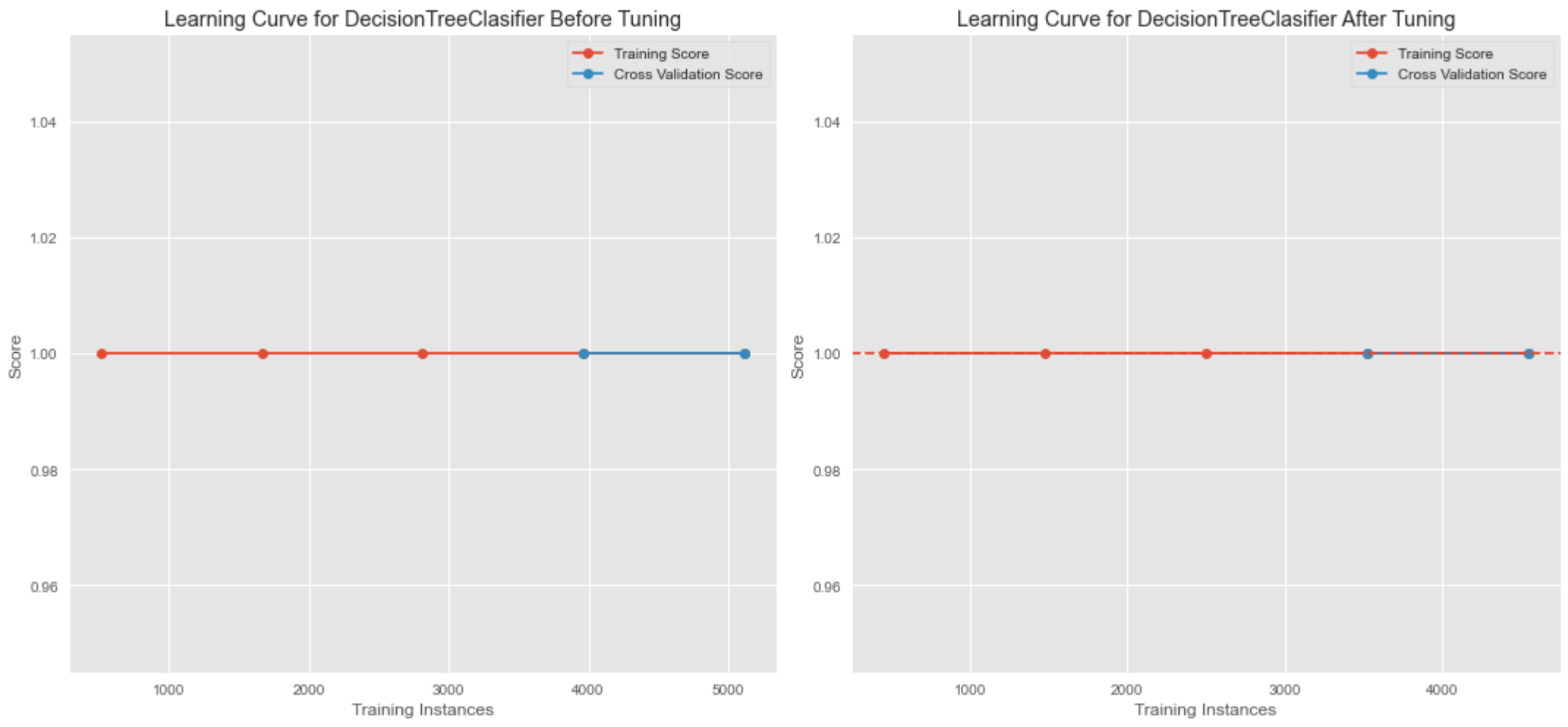

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Model Evaluation





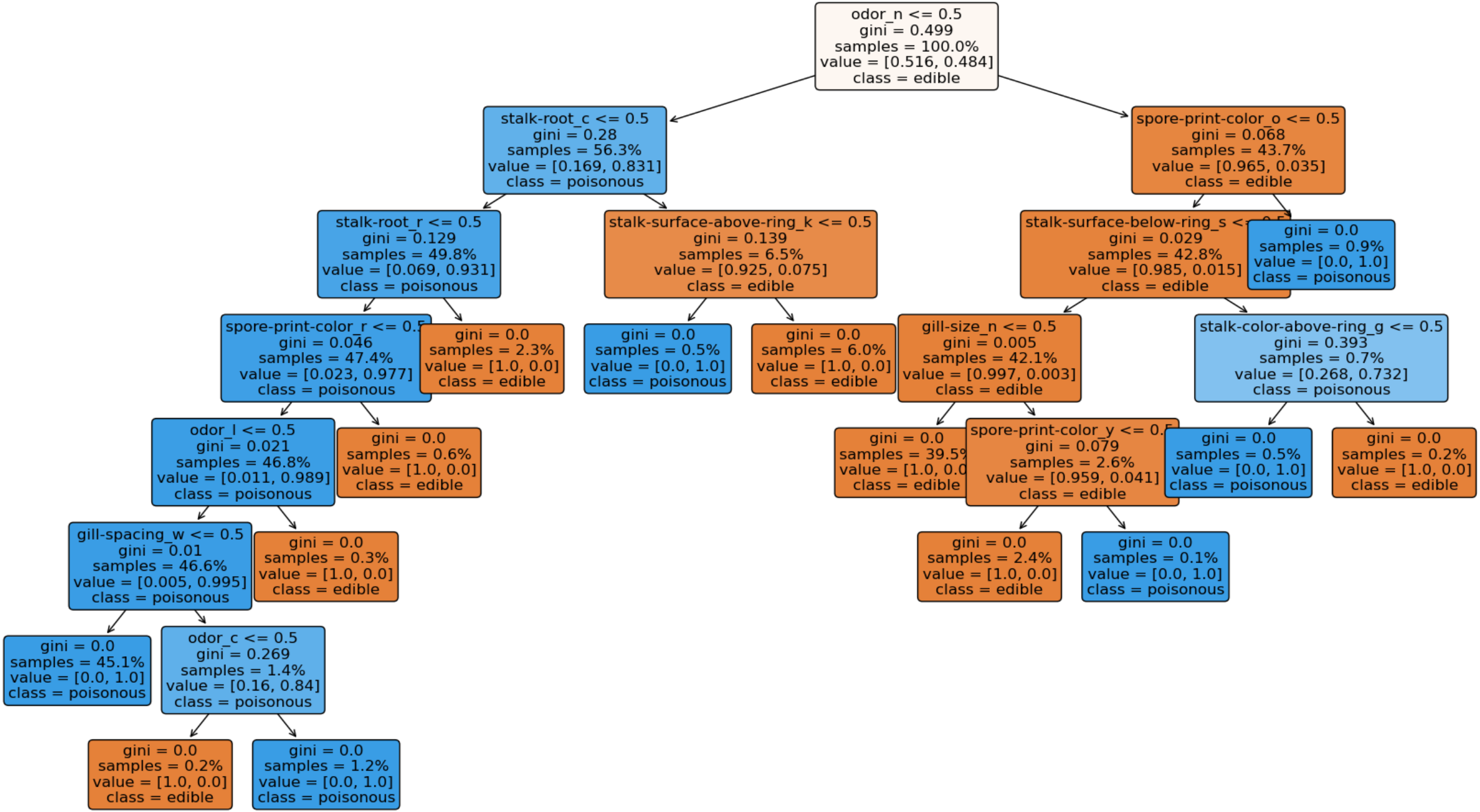
Model Evaluation – Learning Curve



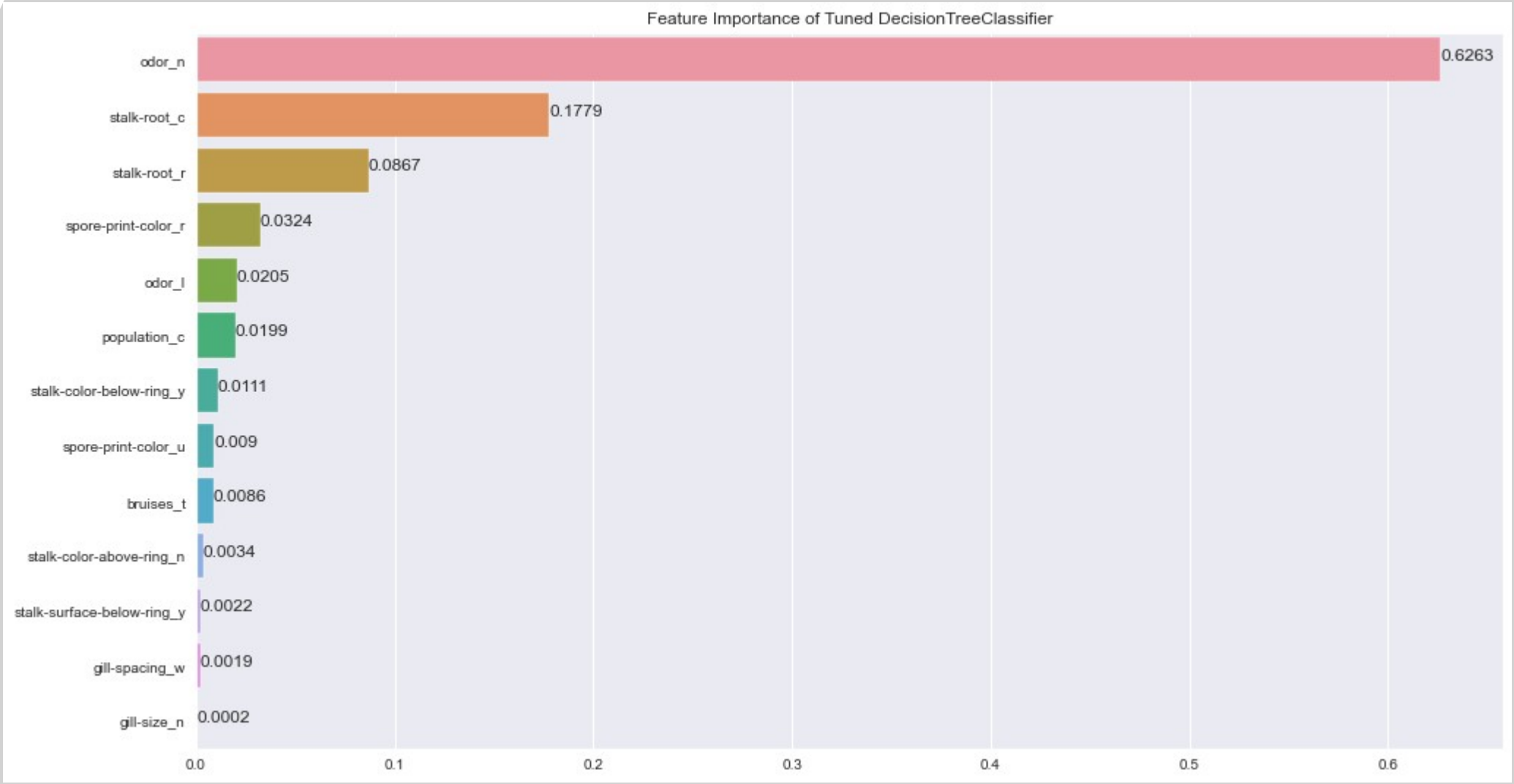
Model Evaluation

	precision	recall	f1-score	support
0	1.000	1.000	1.000	1257
1	1.000	1.000	1.000	1181
accuracy			1.000	2438
macro avg	1.000	1.000	1.000	2438
weighted avg	1.000	1.000	1.000	2438

```
Cross Validation Scores: [1. 1. 1. 1. 1.]
Mean Cross Validation Scores: 1.0
```


odor_n <= 0.5
gini = 0.499
samples = 100.0%
value = [0.516, 0.484]
class = edible
stalk-root_c <= 0.5
gini = 0.28
samples = 56.3%
value = [0.169, 0.831]
class = poisonous
spore-print-color_o <= 0.5
gini = 0.068
samples = 43.7%
value = [0.965, 0.035]
class = edible
stalk-root_r <= 0.5
gini = 0.129
samples = 49.8%
value = [0.069, 0.931]
class = poisonous
stalk-surface-above-ring_k <= 0.5
gini = 0.139
samples = 6.5%
value = [0.925, 0.075]
class = edible
stalk-surface-below-ring_s <=
gini = 0.029
samples = 42.8%
value = [0.985, 0.015]
class = edible
gini = 0.0
samples = 0.9%
value = [0.0, 1.0]
class = poisonous
spore-print-color_r <= 0.5
gini = 0.046
samples = 47.4%
value = [0.023, 0.977]
class = poisonous
gini = 0.0
samples = 2.3%
value = [1.0, 0.0]
class = edible
gini = 0.0
samples = 0.5%
value = [0.0, 1.0]
class = poisonous
gini = 0.0
samples = 6.0%
value = [1.0, 0.0]
class = edible
gill-size_n <= 0.5
gini = 0.005
samples = 42.1%
value = [0.997, 0.003]
class = edible
stalk-color-above-ring_g <= 0.5
gini = 0.393
samples = 0.7%
value = [0.268, 0.732]
class = poisonous
odor_l <= 0.5
gini = 0.021
samples = 46.8%
value = [0.011, 0.989]
class = poisonous
gini = 0.0
samples = 0.6%
value = [1.0, 0.0]
class = edible
gini = 0.0
samples = 39.5%
value = [1.0, 0.0]
class = edible
spore-print-color_y <= 0.5
gini = 0.079
samples = 2.6%
value = [0.959, 0.041]
class = edible
gini = 0.0
samples = 0.5%
value = [0.0, 1.0]
class = poisonous
gini = 0.0
samples = 0.2%
value = [1.0, 0.0]
class = edible
gill-spacing_w <= 0.5
gini = 0.01
samples = 46.6%
value = [0.005, 0.995]
class = poisonous
gini = 0.0
samples = 0.3%
value = [1.0, 0.0]
class = edible
gini = 0.0
samples = 2.4%
value = [1.0, 0.0]
class = edible
gini = 0.0
samples = 0.1%
value = [0.0, 1.0]
class = poisonous
gini = 0.0
samples = 45.1%
value = [0.0, 1.0]
class = poisonous
odor_c <= 0.5
gini = 0.269
samples = 1.4%
value = [0.16, 0.84]
class = poisonous
gini = 0.0
samples = 0.2%
value = [1.0, 0.0]
class = edible
gini = 0.0
samples = 1.2%
value = [0.0, 1.0]
class = poisonous

Model Evaluation – Feature Importance



Forenote

I think the most defining factor to checking a mushroom edibility is their odor and their stalk roots.

One Area I would improve is perhaps implement better feature selection techniques for practical machine learning and better deployment flexibility.

Part B: Kings County House Price Prediction

Prediction Task

The prediction task is to create a predictive regression model to predict the house prices based of the house sales' attribute given.

Output Variable

The output variable is `price`, refering to the price of the house sale. The variable is numerical-continuous variable, as such the prediction task requires a regression model.

The Data

Metadata

column	description	type
id	house id	nominal
date	date of sale	continuous
price	price of sale	continuous
bedrooms	number of bedrooms	discrete
bathrooms	number of bathrooms	discrete
sqft_living	floor area of interior living area in square feet	continuous
sqft_lot	floor area of surrounding land area in square feet	continuous
waterfront	house is overlooking a waterfront area	categorical-binary
view	an index from 0 to 4 how good the view of the property	categorical-ordinal
condition	condition of the house from 1 to 5	categorical-ordinal
grade	An index from 1 to 13, where 1-3 falls short of building construction and design, 7 has an average level of construction and design, and 11-13 have a high quality level of construction and design	discrete
sqft_above	area of interior house space above ground levels in square feet	continuous
sqft_basement	area of the interior basement in square feet	continuous
yr_built	the year the house was built	continuous
yr_renovated	the year the house was last renovated	
zipcode	a code to represent a certain area the house is located at	discrete
long	longitude	continuous
lat	latitude	continuous
sqft_living15	The square footage of interior housing living space for the nearest 15 neighbors	continuous
sqft_lot15	The square footage of the land lots of the nearest 15 neighbors	continuous

Exploratory Data Analysis

Topics I want to cover:

- Data Profile
- Warnings
- Analysis on Target Variable and Attributes
- Feature Selection

I have hosted a copy of my data profile on GitHub so feel free to scan the QR or access the link below.

URL: https://lekekbot.github.io/yh-just-why/

Scan Here!

Credits: this is not my GitHub account, special thanks to Bryan from DIT

Data Profile

Overview | Warnings 55 | Reproduction

Dataset statistics

Number of variables	21
Number of observations	21613
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	3.5 MiB
Average record size in memory	168.0 B

Variable types

Numeric	17
DateTime	1
Categorical	3

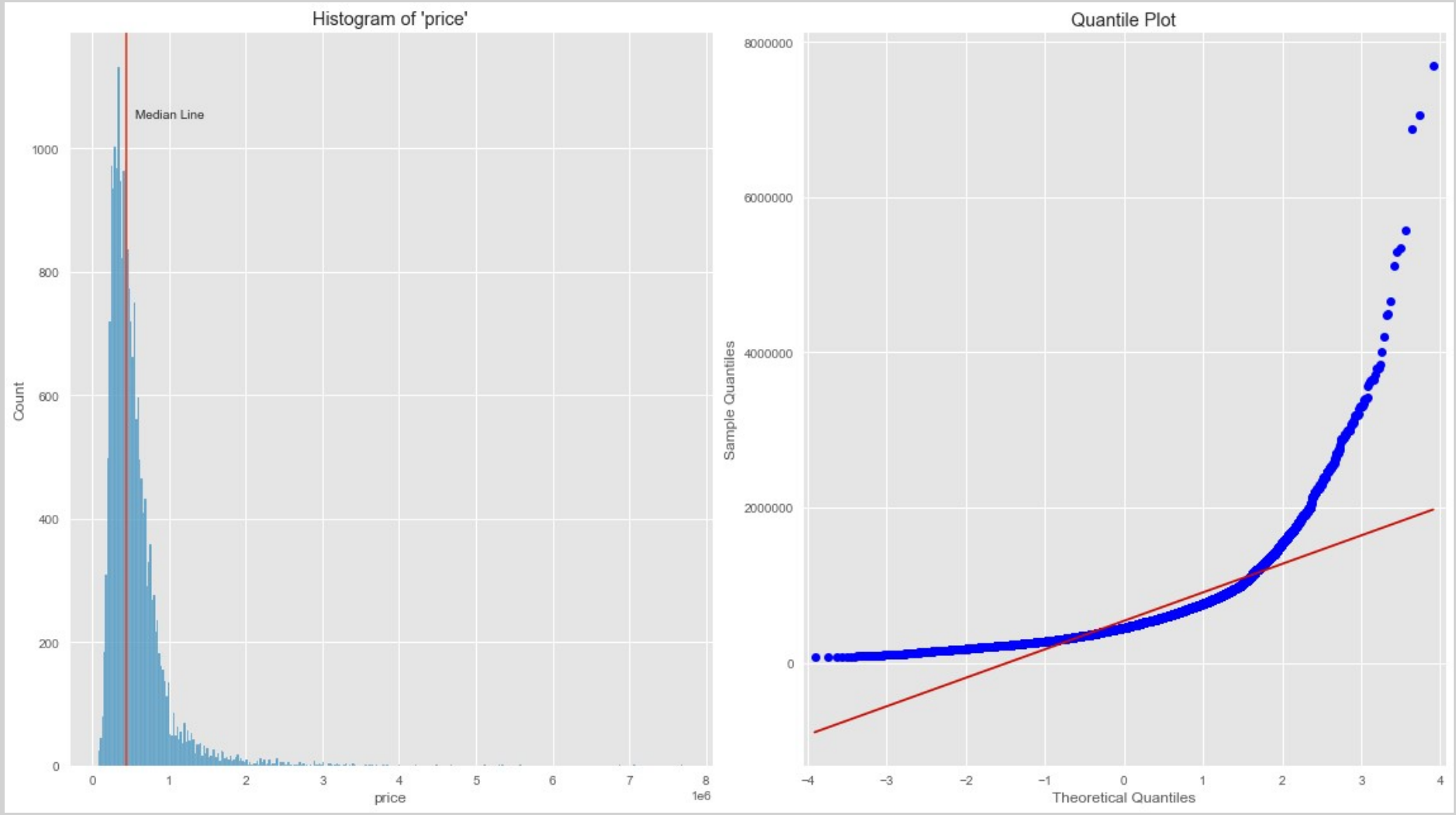
Warnings

Profile Reports shows **55 Warnings**

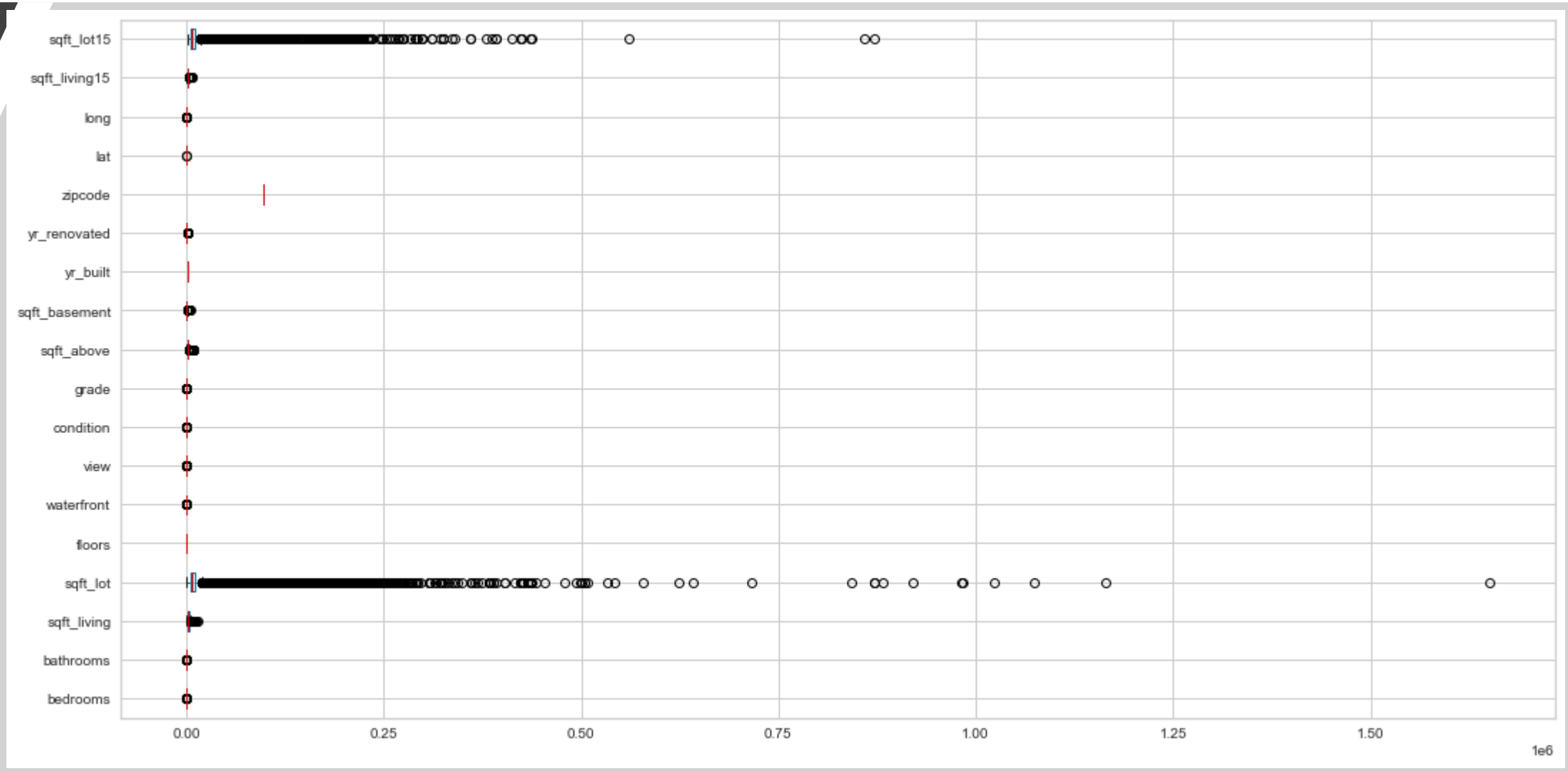
Key points:

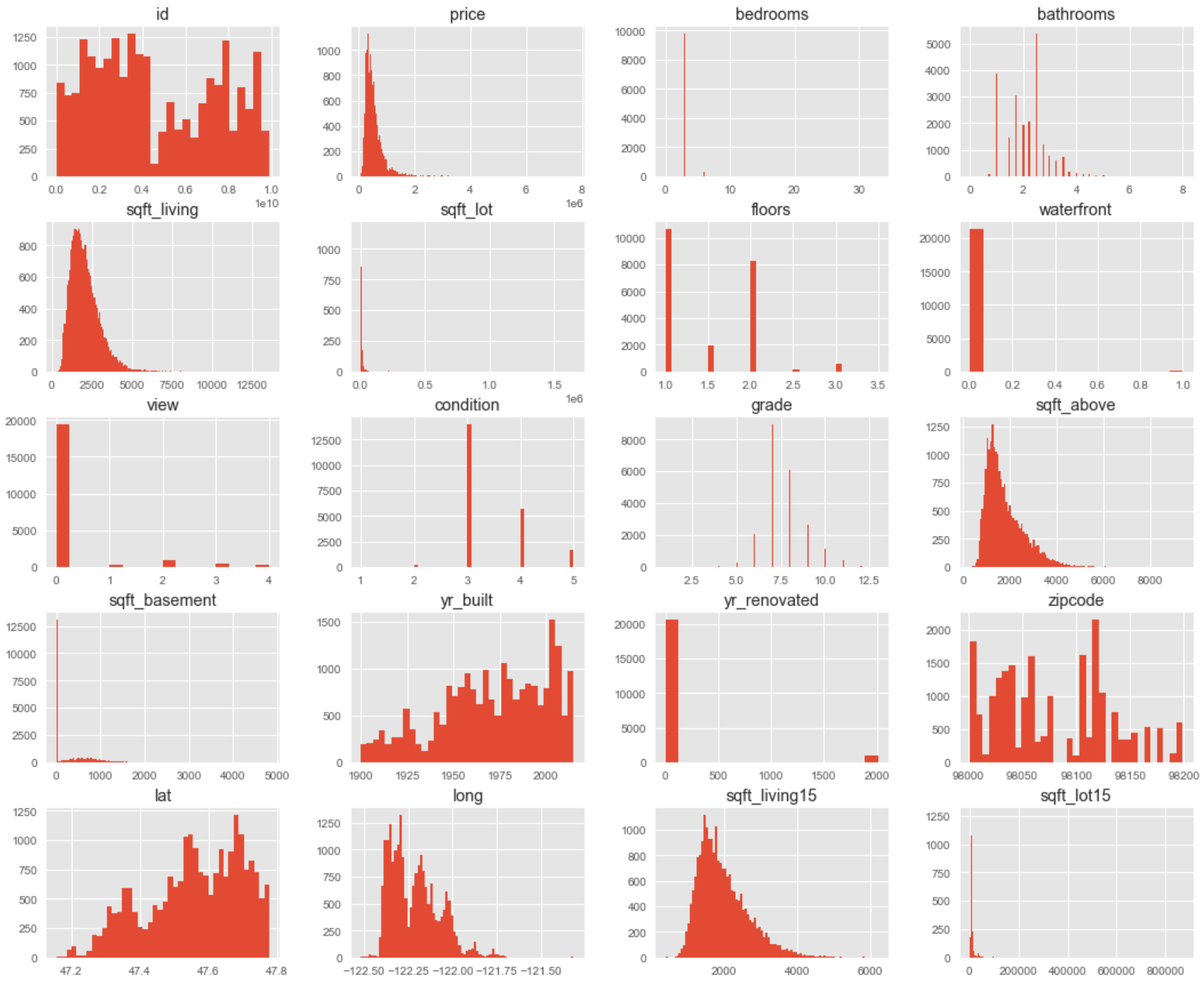
- high correlation between features
  - have to check for multicollinearity
- ‘sqft_basement’ has 60.7% zeros
- ‘yr_renovated’ has 95.8% zeros

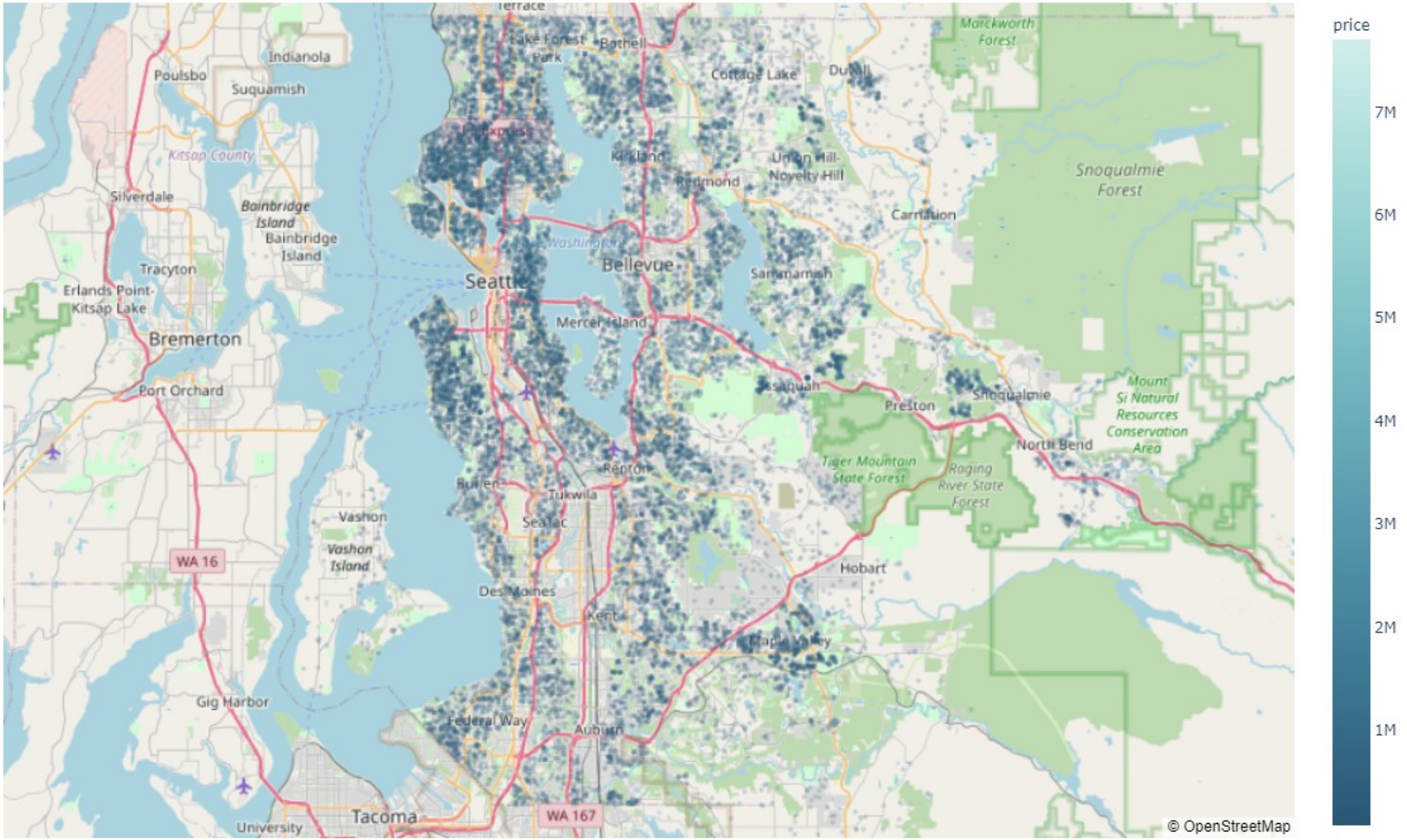
Analysis on Target Variable



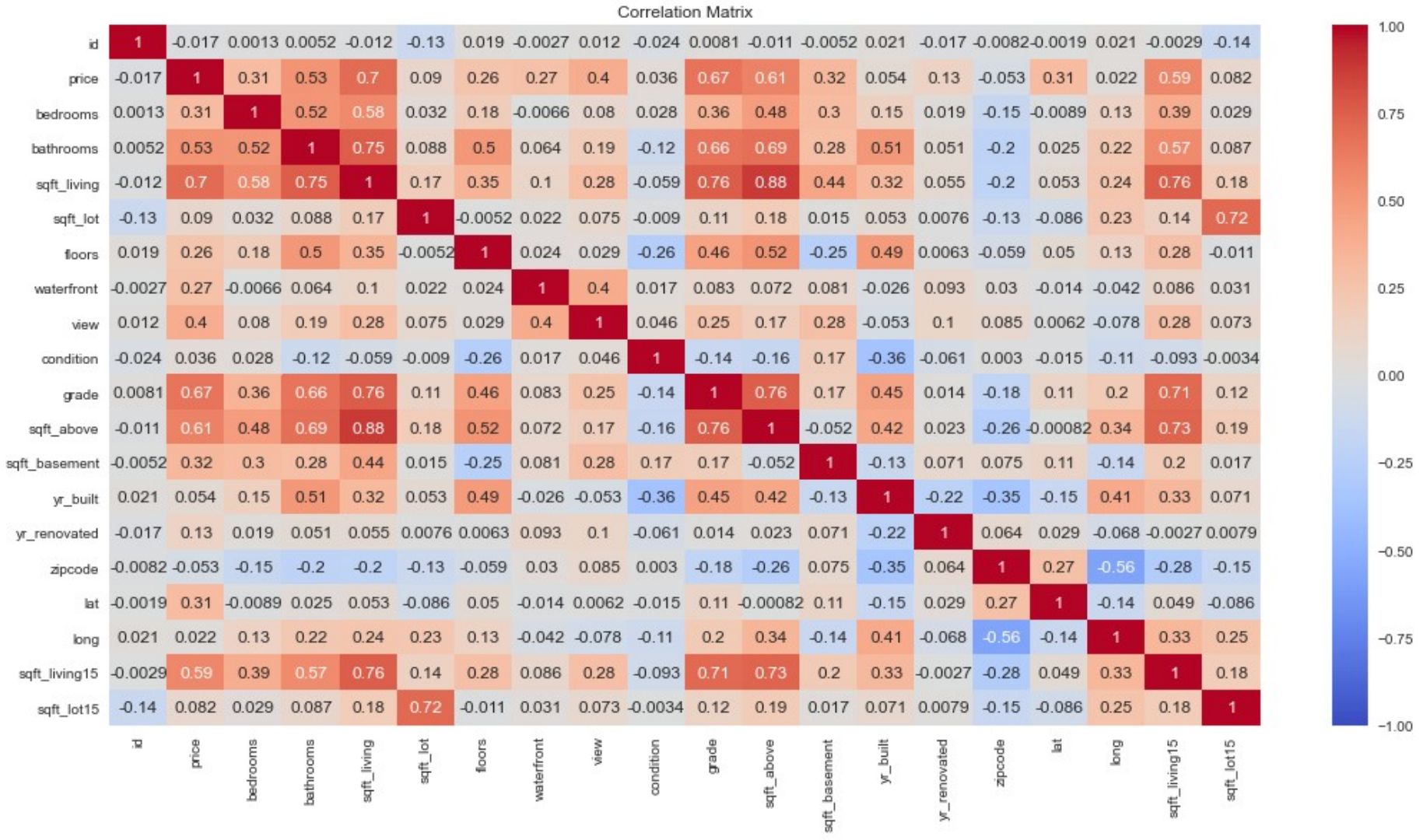
Analysis on Attributes




id
price
bedrooms
bathrooms
sqft_living
sqft_lot
floors
waterfront
view
condition
grade
sqft_above
sqft_basement
yr_built
yr_renovated
zipcode
lat
long
sqft_living15
sqft_lot15


price
7M
6M
5M
4M
3M
2M
1M
Seattle
Bellevue
Bremerton
Tacoma
Kirkland
Redmond
Renton
Kent
Auburn
Federal Way
Tukwila
Sea-Tac
Des Moines
Burien
Vashon
Vashon Island
Gig Harbor
Port Orchard
Poulsbo
Suquamish
Indianola
Silverdale
Kitsap County
Bainbridge Island
Tracyton
Erlands Point-Kitsap Lake
Mercer Island
Issaquah
Snoqualmie
Preston
North Bend
Carnation
Duvall
Cottage Lake
Union Hill-Novelty Hill
Snoqualmie Forest
Mount Si Natural Resources Conservation Area
Tiger Mountain State Forest
Raging River State Forest
Hobart
Maple Valley
WA 16
WA 167
University
© OpenStreetMap

Feature Selection: Pearson Correlation Matrix



Feature Selection – Multicollinearity

Key Points

- sqft_living and sqft_above has high collinearity
- Remove sqft_above as sqft_living has a high correlation with price

	Correlations	Features
0	0.876597	[sqft_living, sqft_above]
1	0.876597	[sqft_above, sqft_living]

Feature Selection: Weak Correlation with price

- id and long has very weak correlation with price
- Dropping both columns are they are insignificant to predicting price

```
[16]
# checking for very weak correlation with price, abs(r) < .05
weak_corr = pd.DataFrame(house_data.corr().loc['price'][abs(house_data.corr().loc['price']) < 0.03])
list(weak_corr.index)

['id', 'long']
```

Feature Engineering / Preprocessing

For Feature Engineering, I will be covering these technique applied to this dataset:

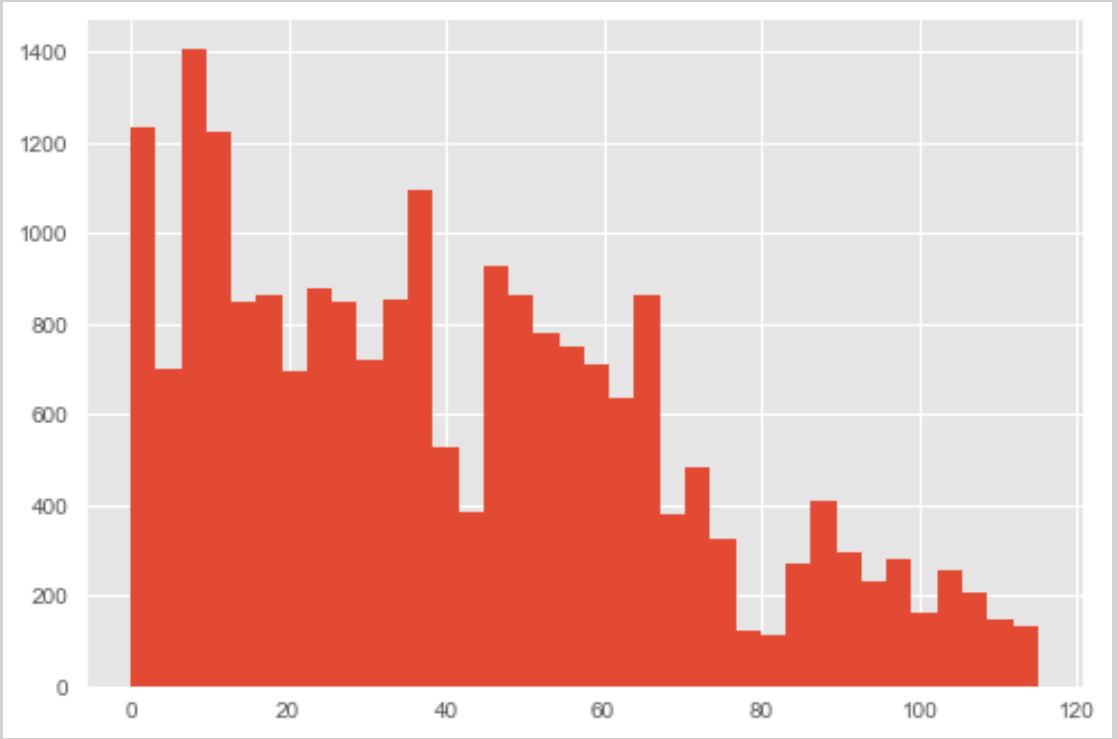
- House Age
- Data Partition
- Log Transformation
- Normalisation
- Target Transformation

House Age

Age referring to the time difference between date and yr_renovated/yr_built

	date	yr_renovated	yr_built	age
0	2014-10-13	0	1955	59.0
1	2014-12-09	1991	1951	23.0
2	2015-02-25	0	1933	82.0
3	2014-12-09	0	1965	49.0
4	2015-02-18	0	1987	28.0
...	...	...	...	...
21608	2014-05-21	0	2009	5.0
21609	2015-02-23	0	2014	1.0
21610	2014-06-23	0	2009	5.0
21611	2015-01-16	0	2004	11.0
21612	2014-10-15	0	2008	6.0

21613 rows × 4 columns



Data Partition

I dropped 6 columns in total so I will be using these columns below as features.

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	zipcode	lat	long	sqft_living15	sqft_lot15	age
0	3	1.00	1180	5650	1.0	0	0	3	7	98178	47.5112	-122.257	1340	5650	59.0
1	3	2.25	2570	7242	2.0	0	0	3	7	98125	47.7210	-122.319	1690	7639	23.0
2	2	1.00	770	10000	1.0	0	0	3	6	98028	47.7379	-122.233	2720	8062	82.0
3	4	3.00	1960	5000	1.0	0	0	5	7	98136	47.5208	-122.393	1360	5000	49.0
4	3	2.00	1680	8080	1.0	0	0	3	8	98074	47.6168	-122.045	1800	7503	28.0

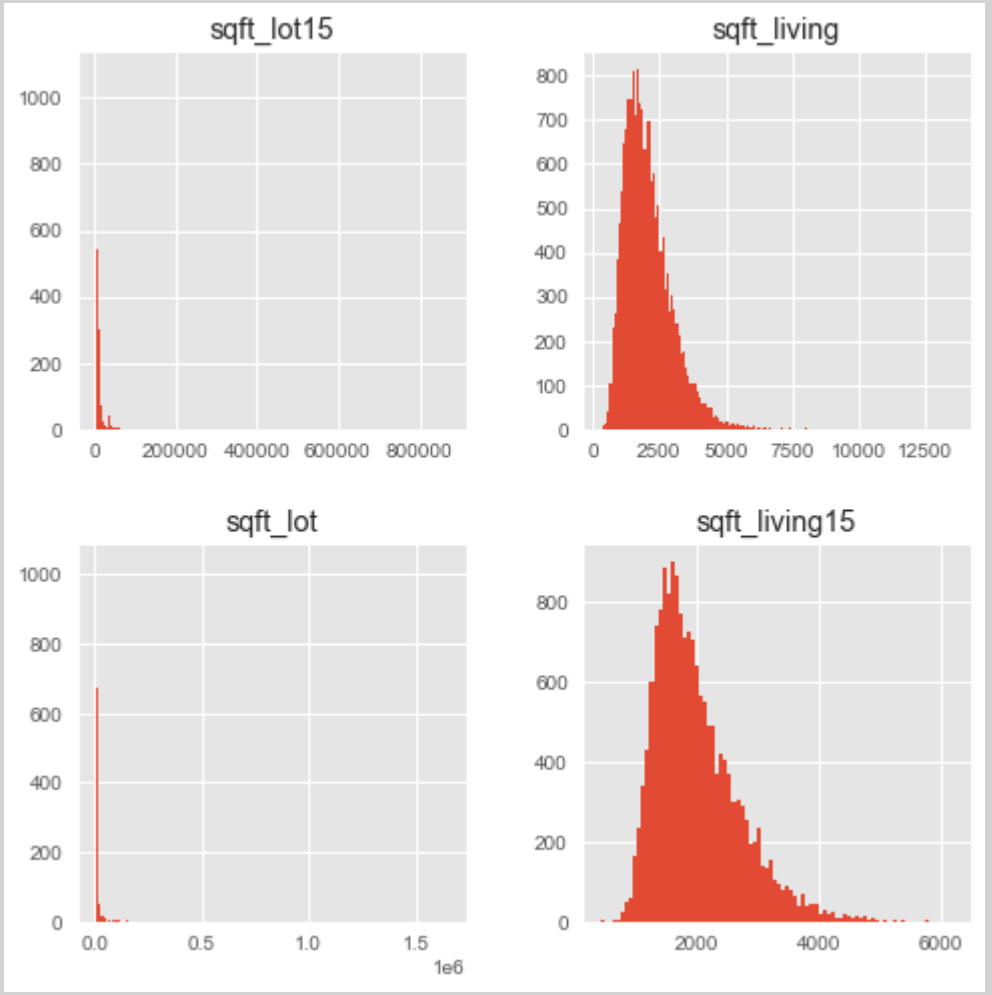
X: (21613, 15) y: (21613,)

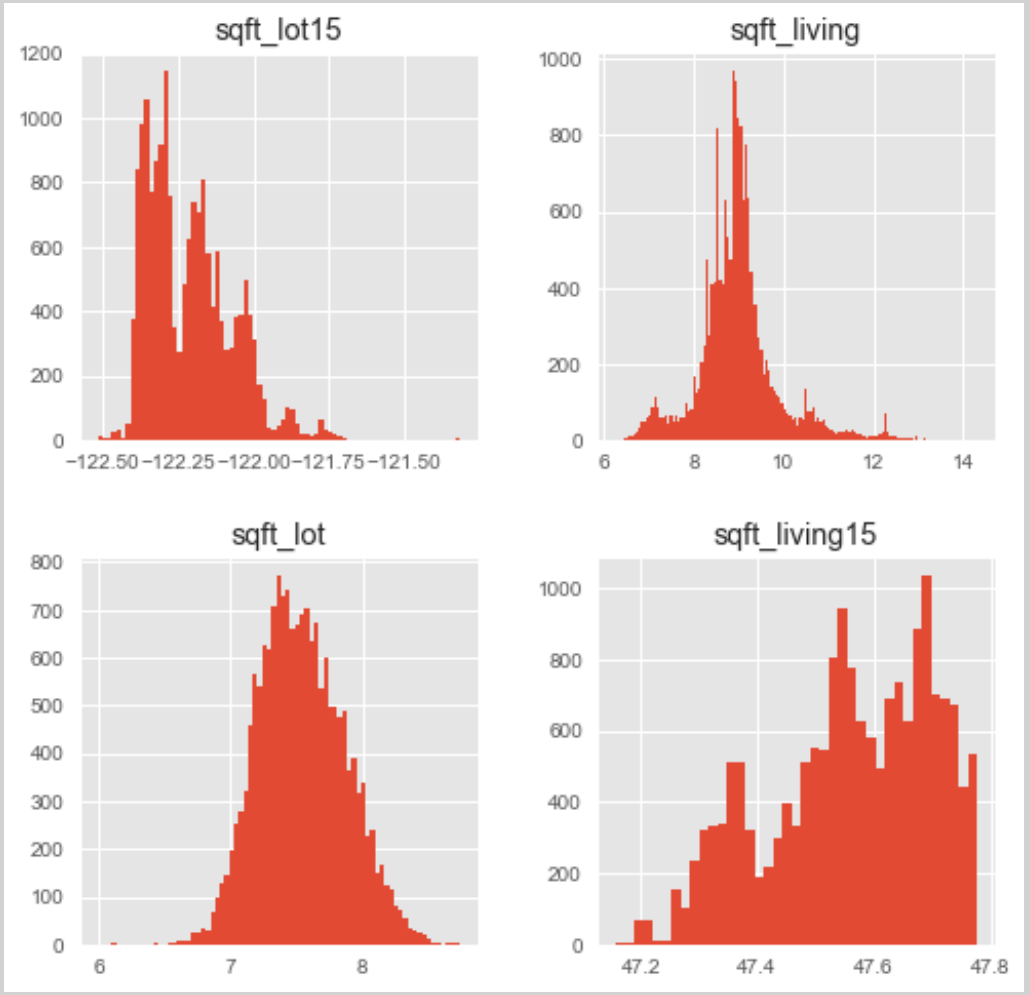
Data Partition

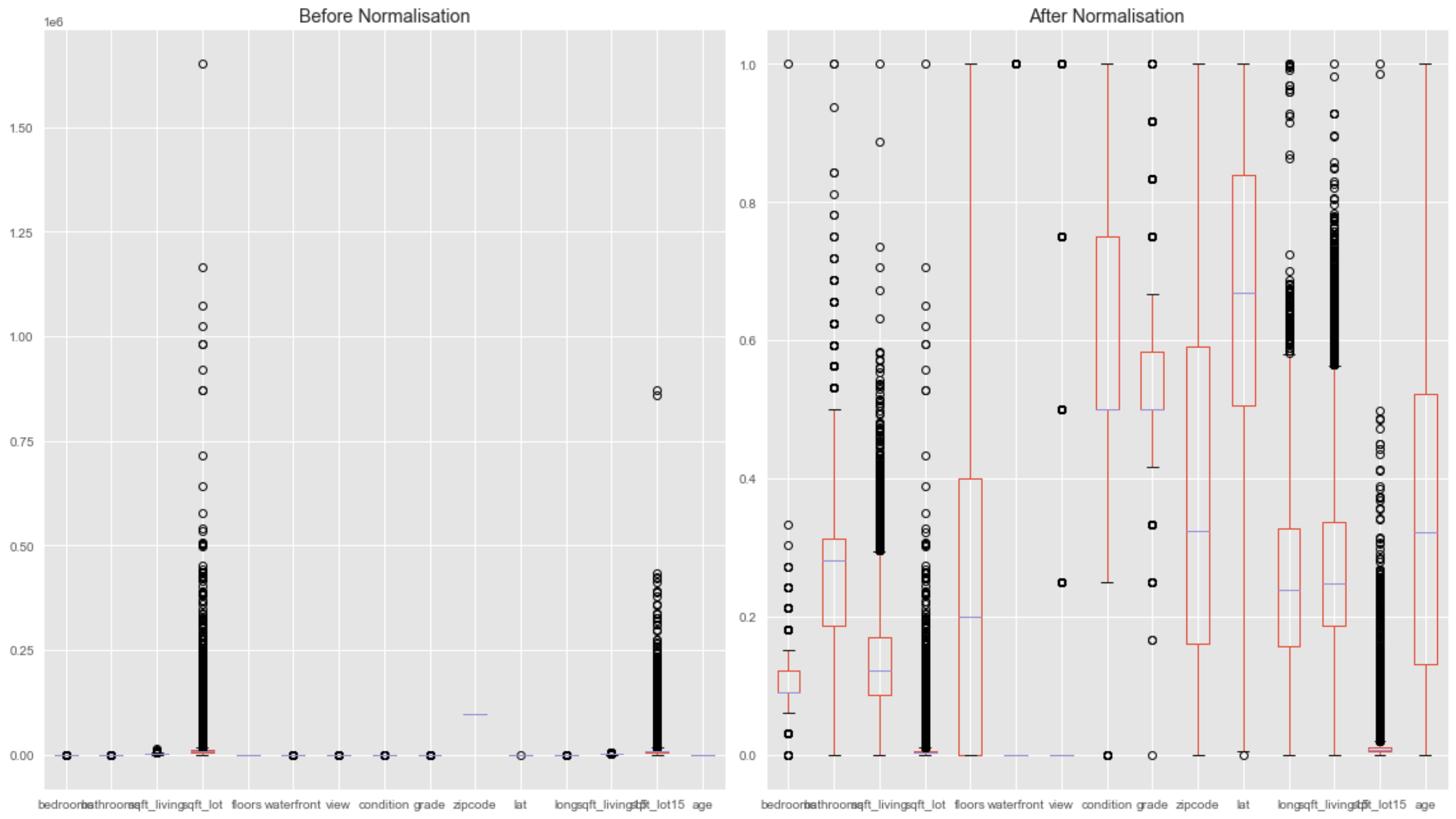
```
# splitting into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2)
print('X_train:', X_train.shape, '\ty_train', y_train.shape)
print('X_test:', X_test.shape, '\ty_test', y_test.shape)
```

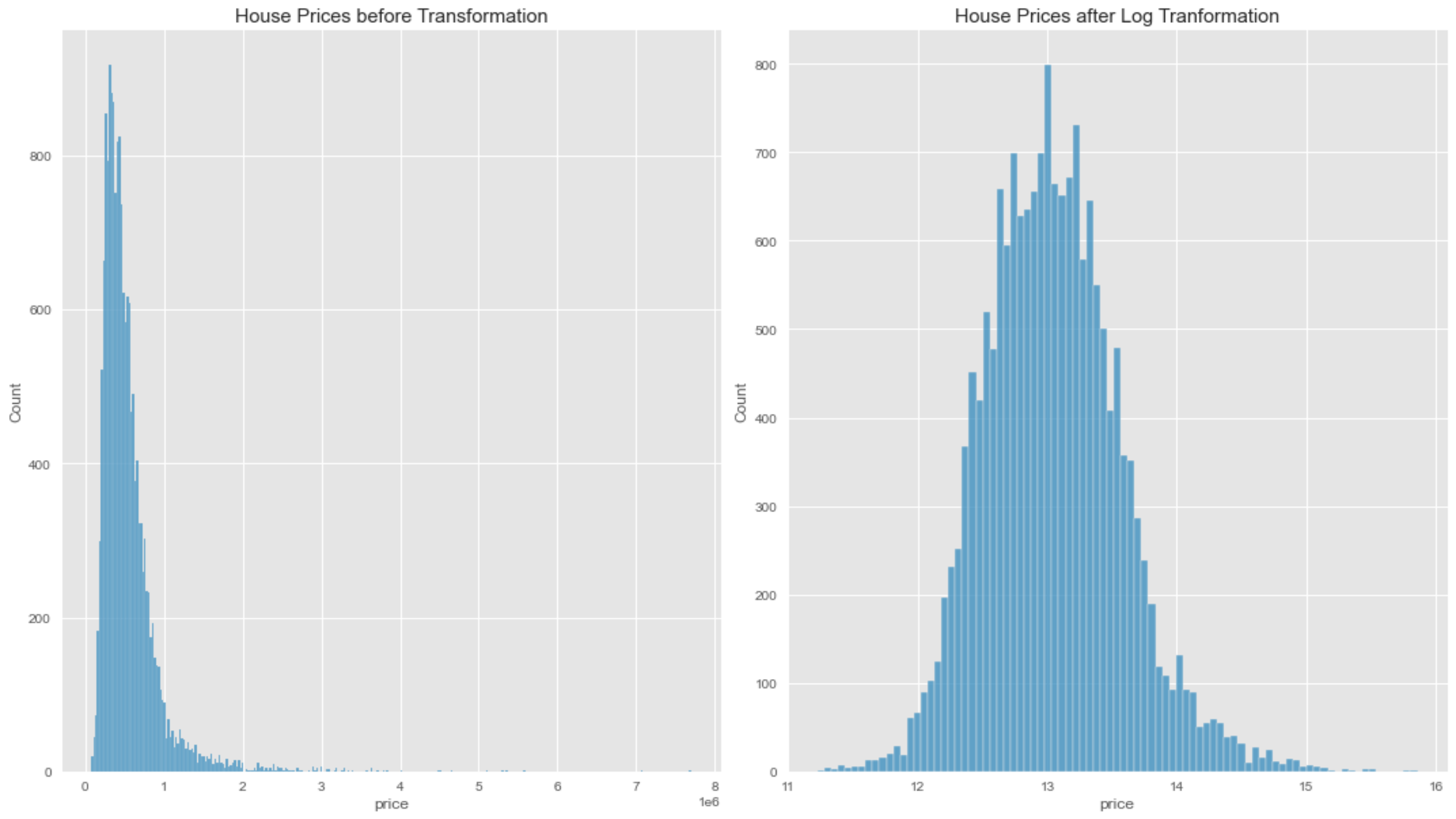
```
X_train: (17290, 15)    y_train (17290,)
X_test: (4323, 15)      y_test (4323,)
```

Log Transformation






Before Normalisation
1e6
1.50
1.25
1.00
0.75
0.50
0.25
0.00
bedrooms
bathrooms
sqft_living
sqft_lot
floors
waterfront
view
condition
grade
zipcode
lat
long
sqft_living15
sqft_lot15
age
After Normalisation
1.0
0.8
0.6
0.4
0.2
0.0
bedrooms
bathrooms
sqft_living
sqft_lot
floors
waterfront
view
condition
grade
zipcode
lat
long
sqft_living15
sqft_lot15
age


House Prices before Transformation
Count
price
1e6
House Prices after Log Tranformation
Count
price

Model Selection

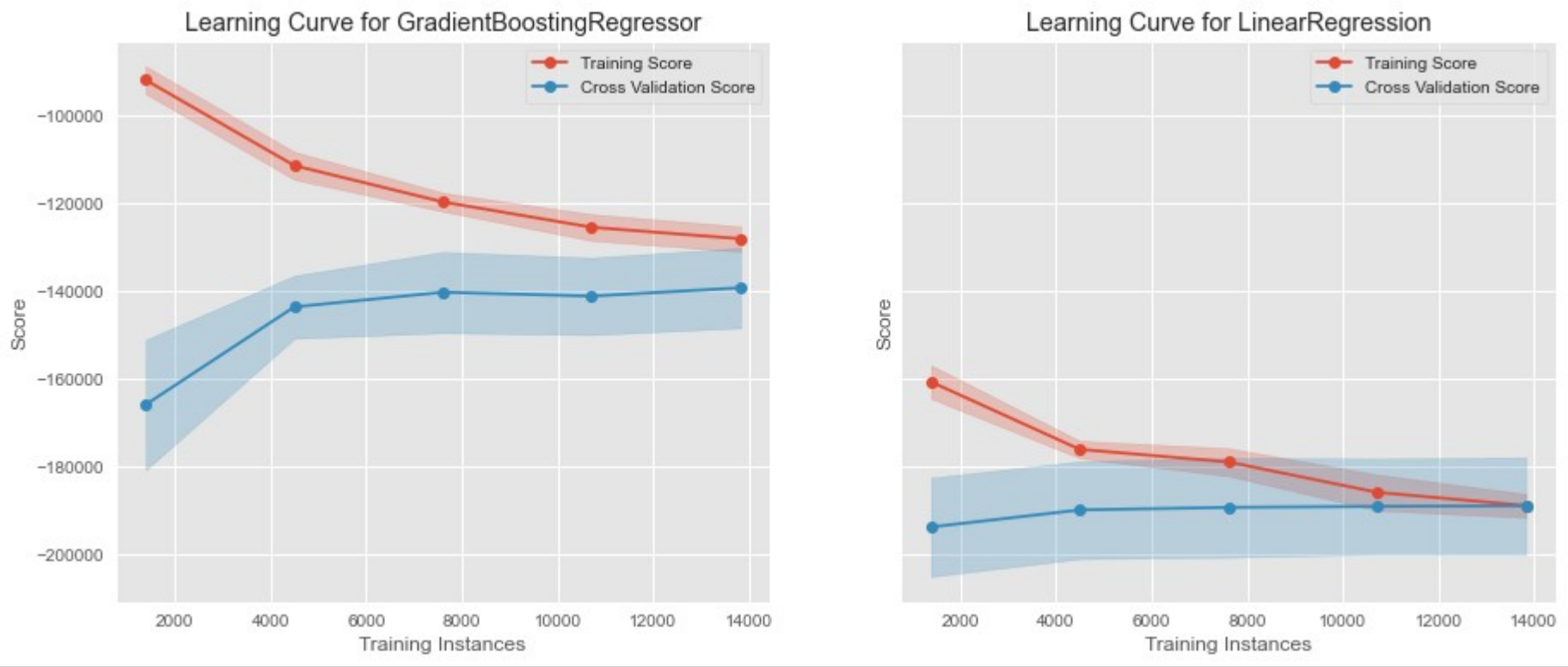
```
for name, models in models:
    pipeline = Pipeline(
        steps=[
          ('LogTransform', LogTransform),
          ('Normalisation', scaler),
          (name, TransformedTargetRegressor(regressor=model, func=np.log1p, inverse_func=np.expm1))
        ]
    )
    cross_validate(pipeline, X_train, y_train, cv=5, scoring=['RMSE', 'MSE', 'MAE', 'MAPE', 'R2'])
```

Model Selection

	fit_time	score_time	test_rmse	train_rmse	test_mse	train_mse	test_mae	train_mae	test_mape	train_mape	test_r2	train_r2
ExtraTreesRegressor	7.135958	0.200463	-133390.500654	-3068.189155	-17936571347.077385	-9513161.568014	-69219.727082	-123.724826	-0.126745	-0.000425	0.869151	0.999930
RandomForestRegressor	12.393565	0.149601	-135627.651945	-55290.737436	-18525893661.364635	-3058882980.024376	-70347.560765	-26816.512434	-0.128178	-0.047119	0.864615	0.977534
GradientBoostingRegressor	3.347374	0.010773	-139285.123516	-128246.897591	-19486473186.692482	-16455351352.906652	-75522.579562	-71986.147128	-0.137311	-0.131361	0.856904	0.879157
AdaBoostRegressor	1.306047	0.024340	-140606.259676	-64731.613580	-19908085622.554283	-4203027839.285998	-74223.345925	-30492.814420	-0.135643	-0.053883	0.854434	0.969156
BaggingRegressor	1.309004	0.028725	-140874.161656	-63135.779721	-19951648787.887047	-3996438359.867338	-74411.359193	-30475.733821	-0.136075	-0.053843	0.853591	0.970632
SVR	13.303492	4.843579	-145188.994993	-131623.979113	-21304507975.040260	-17330699587.146801	-76776.802559	-72849.505850	-0.139376	-0.134285	0.844945	0.872686
MLPRegressor	15.115033	0.012760	-149213.947738	-141366.393039	-22478985541.641655	-20019688178.613991	-79865.868099	-78126.303800	-0.142699	-0.140868	0.835492	0.852696
KNeighborsRegressor	0.189692	0.985876	-168784.280528	-140807.627813	-28818222820.154930	-19848924340.629433	-84608.095344	-68939.526920	-0.150401	-0.121234	0.790136	0.854273
DecisionTreeRegressor	0.213815	0.008178	-188055.943134	-3068.189155	-35613540755.972511	-9513161.568014	-100716.780634	-123.724826	-0.185233	-0.000425	0.739723	0.999930
LinearRegression	0.624596	0.075797	-188994.705372	-188944.478452	-35841165312.482529	-35707235447.812317	-110814.325837	-110672.289058	-0.201534	-0.201302	0.736630	0.737656
Ridge	0.368858	0.006184	-189350.393436	-189301.885531	-35977847205.701393	-35842467227.082199	-110861.627598	-110719.515040	-0.201544	-0.201316	0.735637	0.736663
SGDRegressor	0.138829	0.007580	-210493.017840	-210375.920353	-44405094031.160751	-44267019889.859184	-125276.401612	-125086.255084	-0.221620	-0.221336	0.672652	0.674762
ElasticNet	0.051257	0.007977	-376119.759146	-376659.424464	-141907449864.942780	-141897913482.805664	-221876.211038	-221875.234014	-0.438628	-0.438620	-0.042641	-0.042515
Lasso	0.021941	0.007579	-376119.759146	-376659.424464	-141907449864.942780	-141897913482.805664	-221876.211038	-221875.234014	-0.438628	-0.438620	-0.042641	-0.042515

Model Selection: Comparing with Baseline

	fit_time	score_time	test_mae	test_mape	test_mse	test_r2	test_rmse
GradientBoostingRegressor	3.382798	0.009982	-75506.759853	-0.137306	-1.943598e+10	0.857226	-139112.218952
LinearRegression	0.021336	0.007188	-110814.325837	-0.201534	-3.584117e+10	0.736630	-188994.705372



Hyperparameter Tuning

Earlier this year, sklearn 0.24 update came with HalvingGridSearchCV, a new way to do Hyperparameter Tuning. HalvingGridSearchCV works by giving every combination a small number of resources and see how well it fare, those who've did well will go through the next iteration. Online articles have shown that HalvingGridSearchCV is apparently 11 times faster than GridSearchCV. As such, I will be using this method to run Hyperparameter Tuning.

Hyperparameters to be tuned:

- n_estimators - how many trees we want in the model
- max_depth - how large each tree should be, generally for gradient boosting we want to keep this Hyperparameter low
- learning_rate - how fast should each tree learn, generally there is a trade off between learning rate and n_estimators
- subsample - percentage of learners used for fitting, used to reduce variance (overfitting)

Hyperparameter Tuning – HalvingGridSearchCV

```
grid = {
    'regressor__n_estimators': np.arange(100, 2101, 200),
    'regressor__max_depth': [3, 5, 10],
    'regressor__learning_rate': [.001, .01, .1],
    'regressor__subsample': [.5, .75, 1]
}

model_tuning = Pipeline(
    steps=[
        ('LogTransform', LogTransform),
        ('Normalisation', scaler),
        ('GridSearch', HalvingGridSearchCV(
            TransformedTargetRegressor(
                regressor=GradientBoostingRegressor(),
                func=np.log1p,
                inverse_func=np.expm1
            ),
            grid,
            scoring='neg_root_mean_squared_error',
            n_jobs=4,
            verbose=1,
            cv=5,
            aggressive_elimination=True,
            factor=5
        ))
    ]
)

model_tuning.fit(X_train, y_train)
print('Finish Tuning')
```

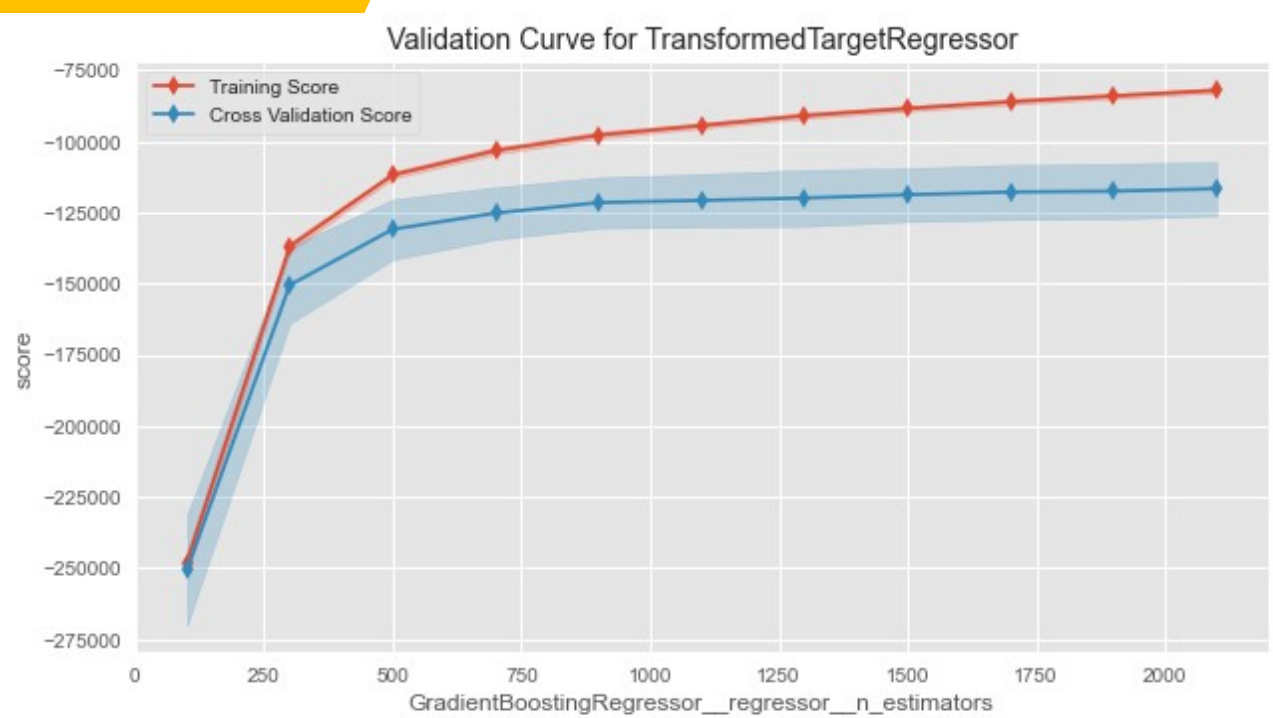
Hyperparameter Tuning – HalvingGridSearchCV

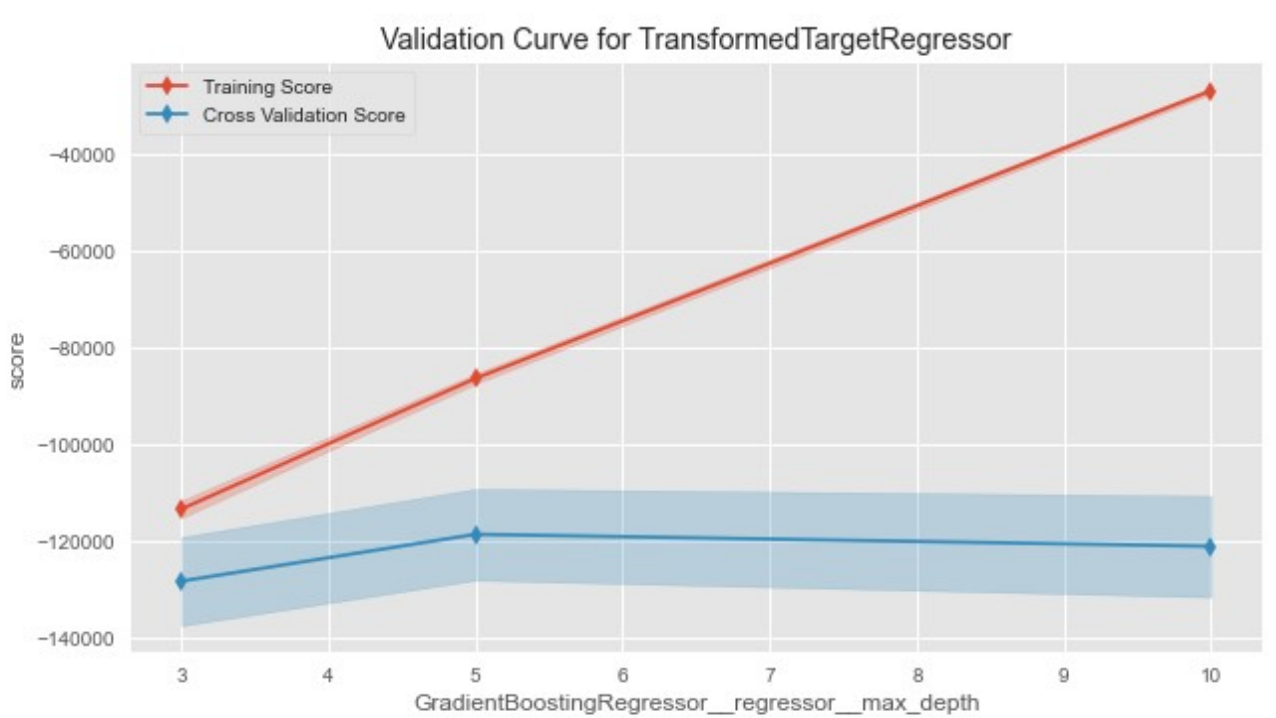
```
print(model_tuning.named_steps['GridSearch'].best_estimator_)
print(model_tuning.named_steps['GridSearch'].best_params_)
print(model_tuning.named_steps['GridSearch'].best_score_)
```

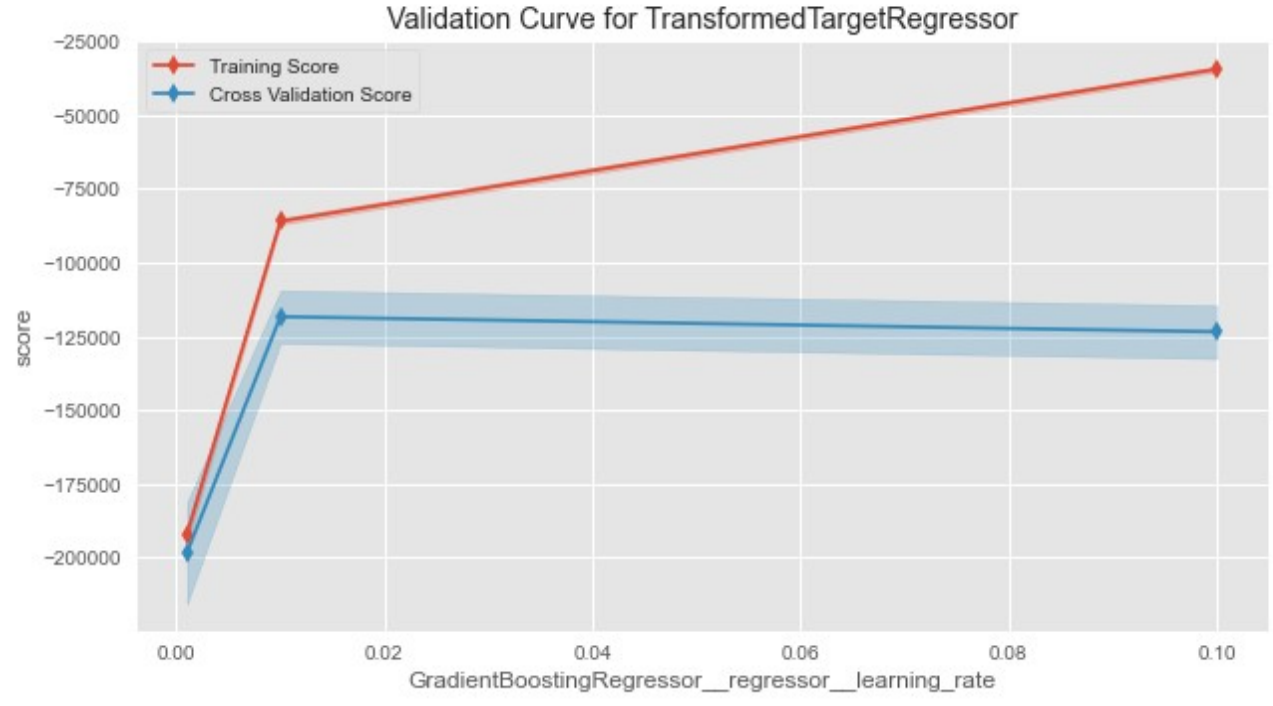
```
TransformedTargetRegressor(func=<ufunc 'log1p'>, inverse_func=<ufunc 'expm1'>,
                           regressor=GradientBoostingRegressor(learning_rate=0.01,
                                                               max_depth=5,
                                                               n_estimators=1700,
                                                               subsample=0.5))
{'regressor__learning_rate': 0.01, 'regressor__max_depth': 5, 'regressor__n_estimators': 1700, 'regressor__subsample': 0.5}
-117994.0405785848
```

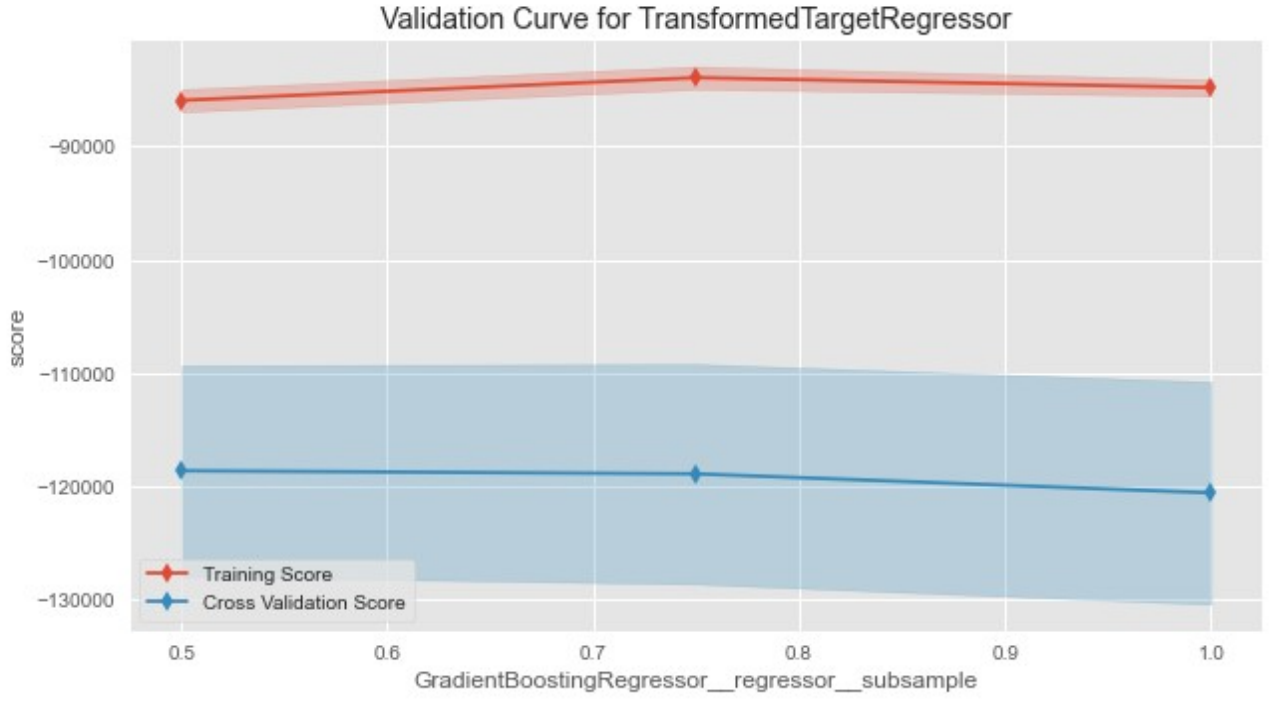
Hyperparameter Tuning – Cross Validation

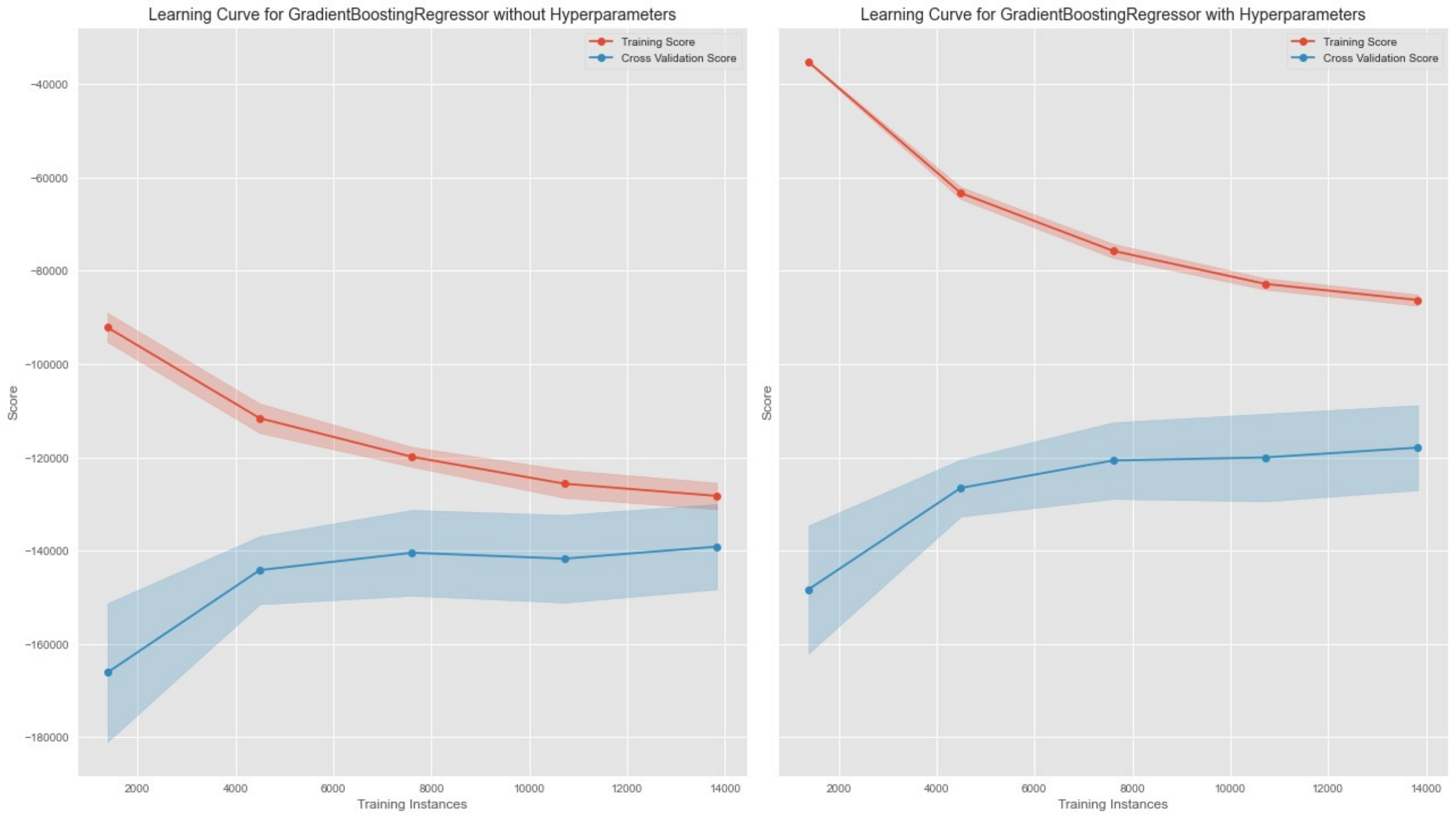
	fit_time	score_time	test_rmse	train_rmse	test_mse	train_mse	test_mae	train_mae	test_mape	train_mape	test_r2	train_r2
0	58.020123	0.086810	-110287.796549	-87873.777194	-12163398067.710070	-7721800718.262233	-61515.395022	-54170.718904	-0.118382	-0.102288	0.886237	0.944583
1	58.176704	0.087764	-117862.225627	-88627.326801	-13891504229.817755	-7854803055.889638	-64520.221175	-54287.466055	-0.117352	-0.102720	0.900445	0.942129
2	58.281422	0.093752	-98768.254392	-87822.022752	-9755168075.591537	-7712707680.333038	-60578.050622	-54082.529809	-0.119683	-0.102097	0.906017	0.944788
3	58.017130	0.084772	-116473.516912	-86506.531990	-13566080141.864235	-7483380076.901083	-66075.221596	-53647.512098	-0.125963	-0.101600	0.891767	0.945500
4	58.214603	0.090756	-124313.669620	-87320.854488	-15453888454.413439	-7624931628.540487	-67124.217803	-54041.453211	-0.126047	-0.101981	0.889574	0.943804
5	60.932899	0.089795	-124958.690351	-87206.978229	-15614674294.295811	-7605057051.759262	-66906.700463	-53994.433041	-0.116382	-0.102672	0.898286	0.943301
6	60.717511	0.085773	-118840.468054	-87202.945720	-14123056847.316528	-7604353742.278698	-66214.891106	-53877.349898	-0.116922	-0.102432	0.899703	0.943916
7	60.910029	0.091722	-144092.200695	-87013.045983	-20762562300.984791	-7571270171.312674	-66096.589442	-53847.869237	-0.120852	-0.102074	0.876179	0.942904
8	61.028681	0.089792	-105036.102393	-88329.979208	-11032582805.893982	-7802185226.871255	-59575.494770	-54421.928143	-0.115063	-0.102637	0.902189	0.943735
9	60.828215	0.090758	-125697.542534	-86993.869298	-15799872199.030666	-7567933295.381018	-68442.750493	-53871.027080	-0.125585	-0.102213	0.906794	0.942832
mean	59.512732	0.089169	-118633.046713	-87489.733166	-14216278741.691879	-7654842264.752939	-64704.953249	-54024.228748	-0.120223	-0.102272	0.895719	0.943749


Validation Curve for TransformedTargetRegressor
Training Score
Cross Validation Score
score
−75000
−100000
−125000
−150000
−175000
−200000
−225000
−250000
−275000
0
250
500
750
1000
1250
1500
1750
2000
GradientBoostingRegressor__regressor__n_estimators


Validation Curve for TransformedTargetRegressor
Training Score
Cross Validation Score
score
−40000
−60000
−80000
−100000
−120000
−140000
3
4
5
6
7
8
9
10
GradientBoostingRegressor__regressor__max_depth

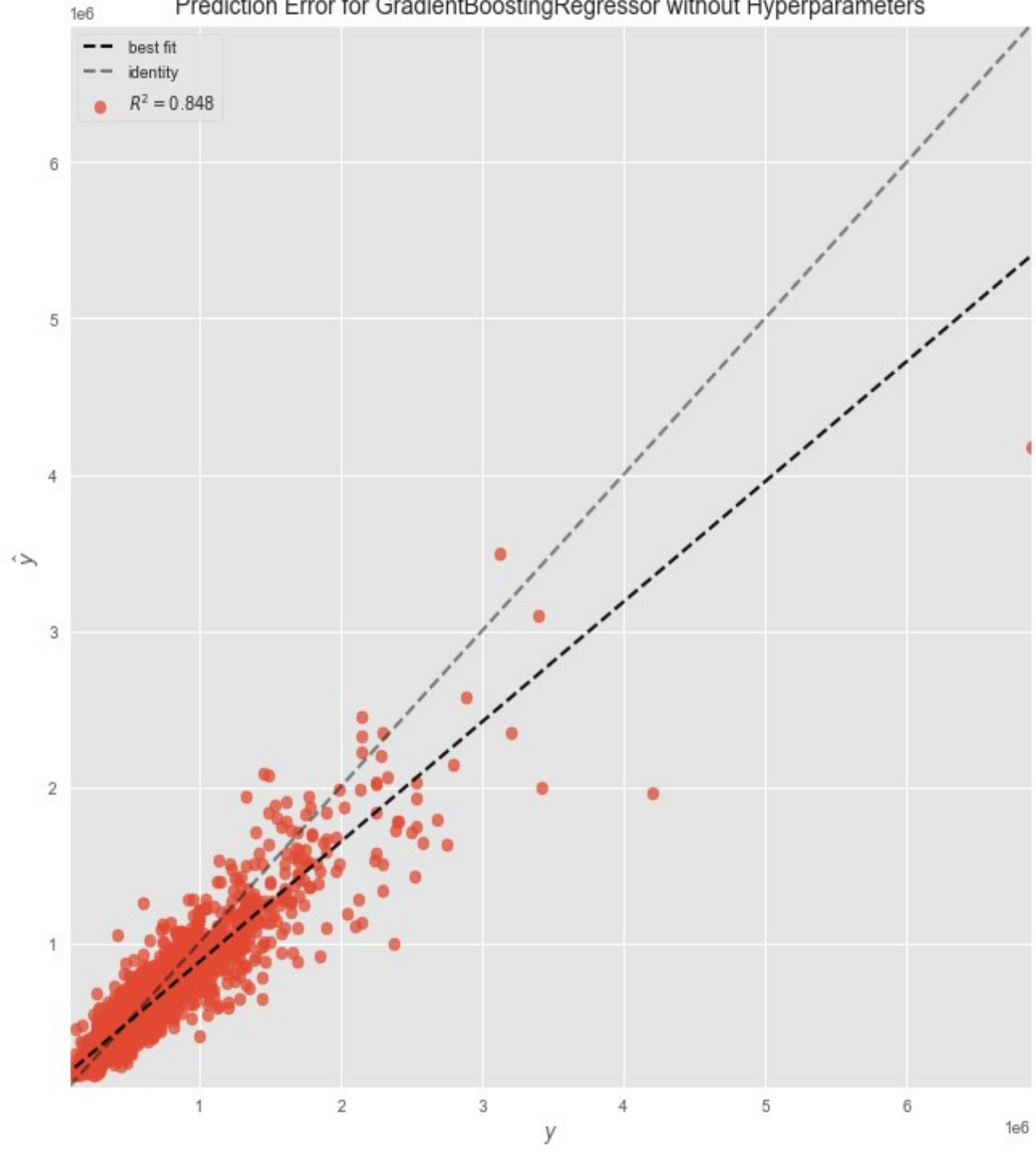

Validation Curve for TransformedTargetRegressor
Training Score
Cross Validation Score
score
−25000
−50000
−75000
−100000
−125000
−150000
−175000
−200000
0.00
0.02
0.04
0.06
0.08
0.10
GradientBoostingRegressor__regressor__learning_rate

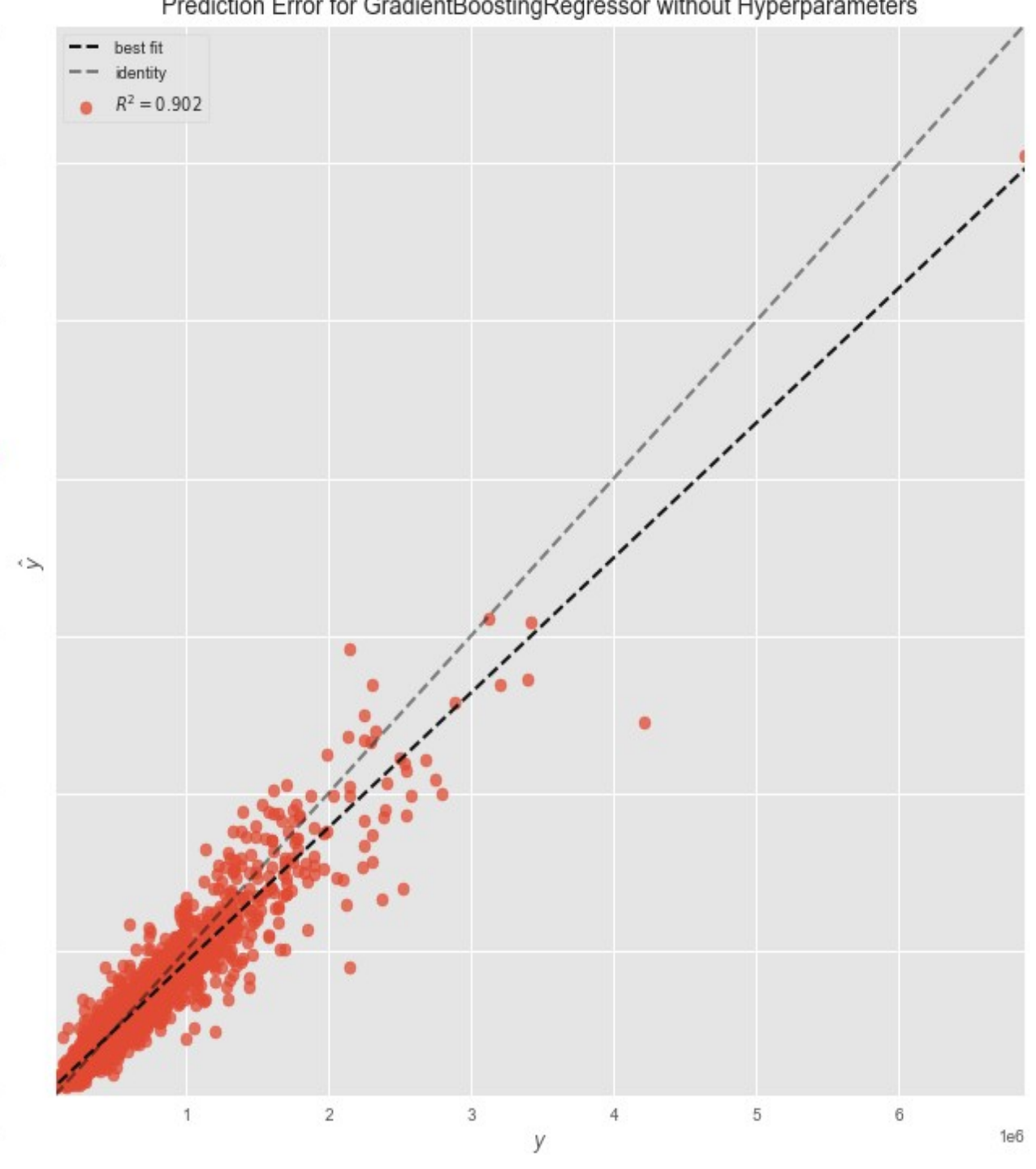

Validation Curve for TransformedTargetRegressor
Training Score
Cross Validation Score
score
−90000
−100000
−110000
−120000
−130000
0.5
0.6
0.7
0.8
0.9
1.0
GradientBoostingRegressor__regressor__subsample

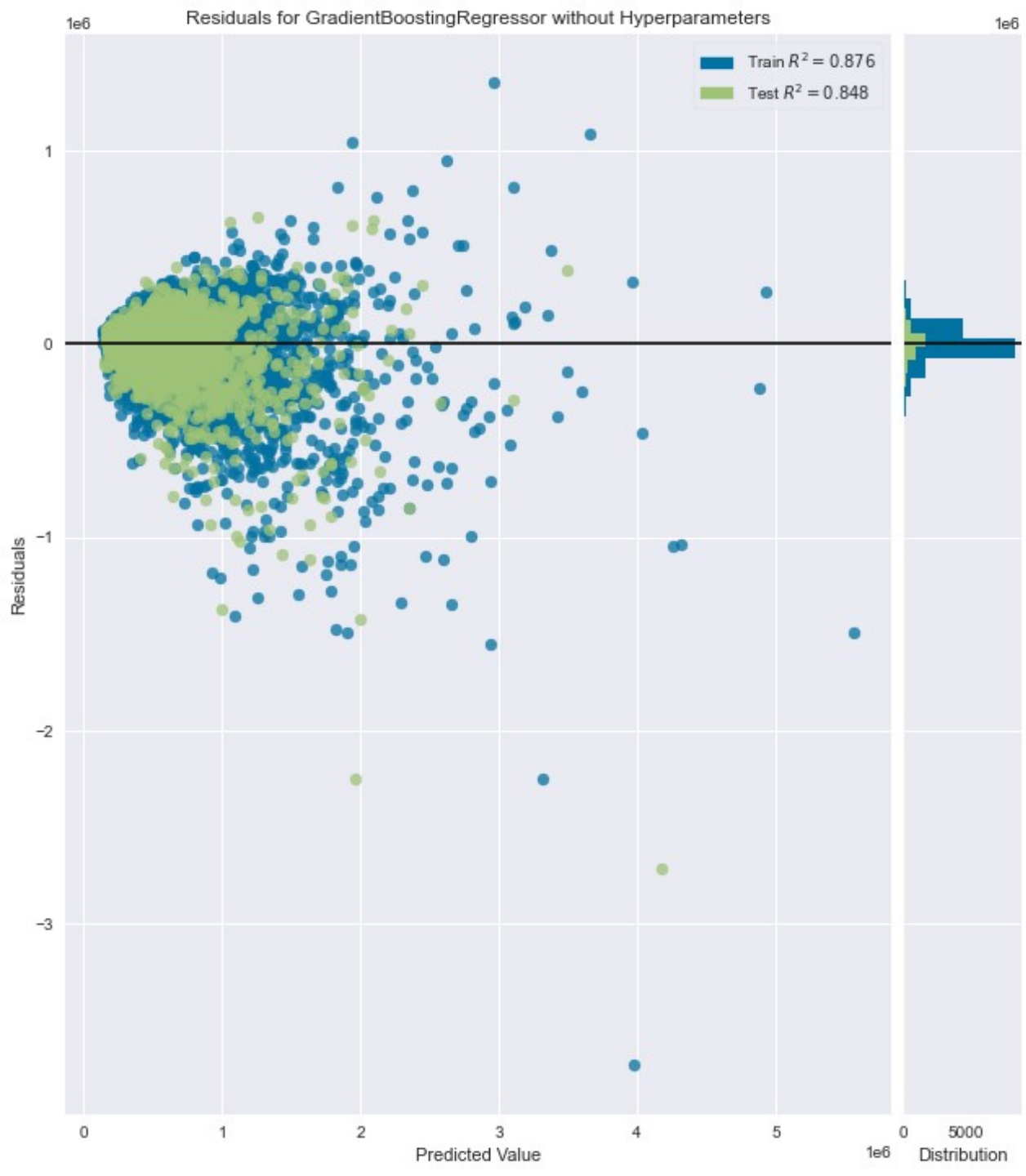

Learning Curve for GradientBoostingRegressor without Hyperparameters
Training Score
Cross Validation Score
−40000
−60000
−80000
−100000
−120000
−140000
−160000
−180000
Score
2000
4000
6000
8000
10000
12000
14000
Training Instances
Learning Curve for GradientBoostingRegressor with Hyperparameters
Training Score
Cross Validation Score
Score
2000
4000
6000
8000
10000
12000
14000
Training Instances

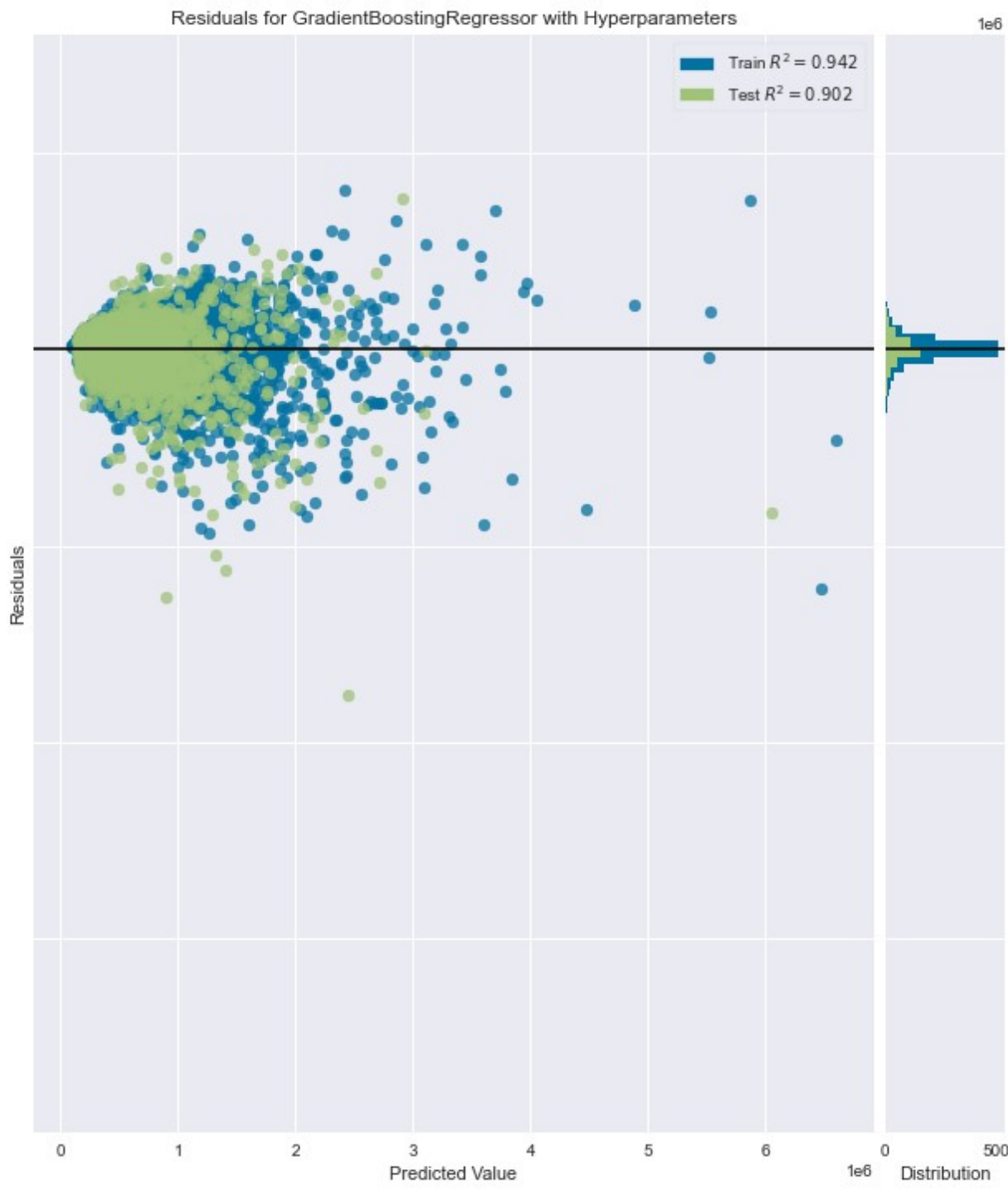
Generating Predictions

	RMSE	MSE	MAE	MAPE	R2
GradientBoostingRegressor	112580.789489	1.267443e+10	64669.690289	0.121199	0.902068

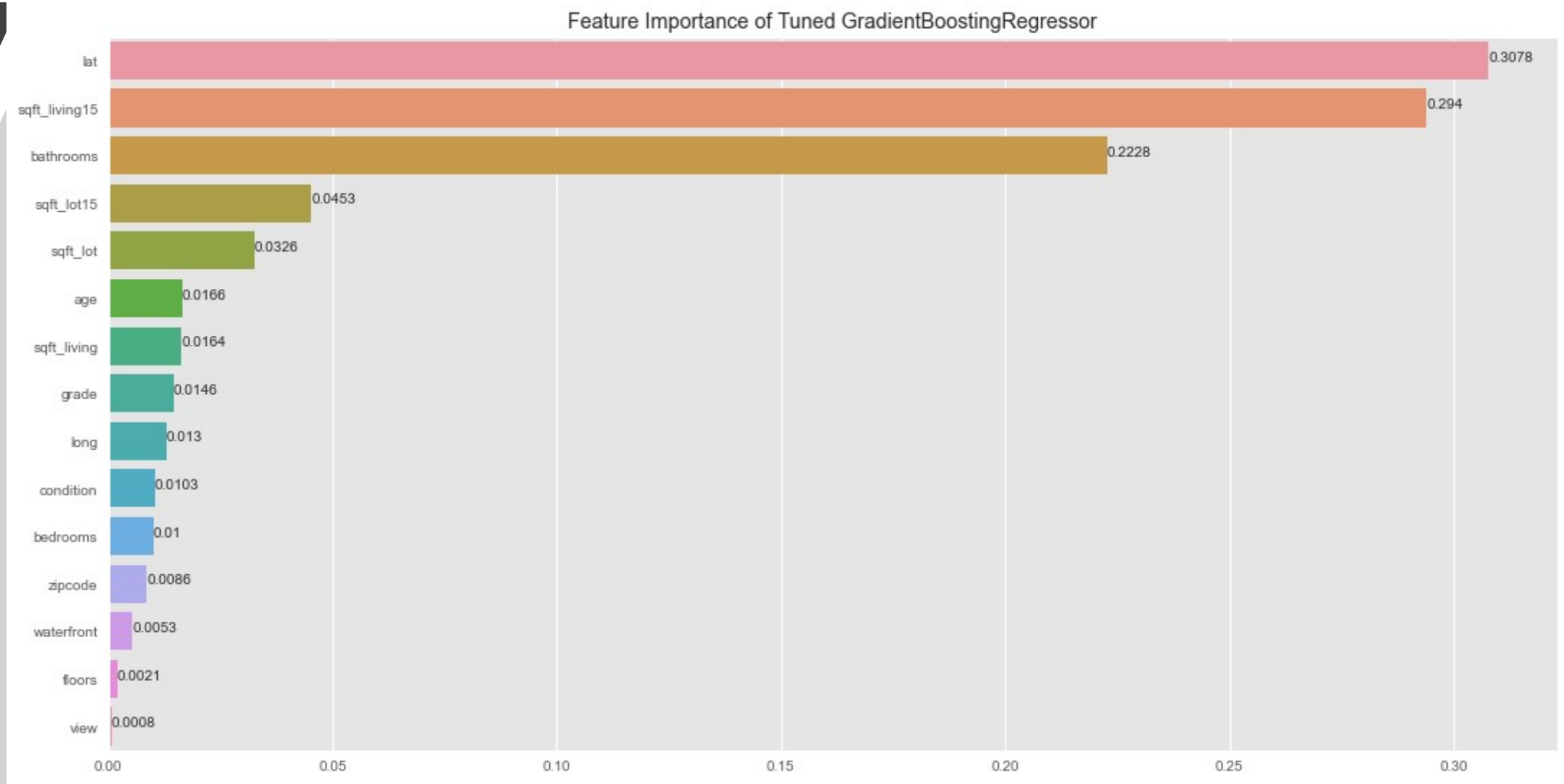

Prediction Error for GradientBoostingRegressor without Hyperparameters
1e6
best fit
identity
$R^2 = 0.848$
$\hat{y}$
y
1e6


Prediction Error for GradientBoostingRegressor without Hyperparameters
best fit
identity
$R^2 = 0.902$
$\hat{y}$
y
1e6


Residuals for GradientBoostingRegressor without Hyperparameters
Train $R^2 = 0.876$
Test $R^2 = 0.848$
1e6
Residuals
Predicted Value
Distribution
5000


Residuals for GradientBoostingRegressor with Hyperparameters
Train $R^2 = 0.942$
Test $R^2 = 0.902$
1e6
Residuals
Predicted Value
Distribution
5000

Feature Importance



Conclusion

In conclusion, I would say the model fair well, with 112580 RMSE and 90.2% R-Squared. Seeing Lat, sqft_living15 and bathrooms are the most important features, it shows that location, house sizes and amenities do come into consideration when evaluating the price of the house.

One thing I would like to improve is implementing Geospatial Mapping using Geopandas to properly represent geographical coordinates.