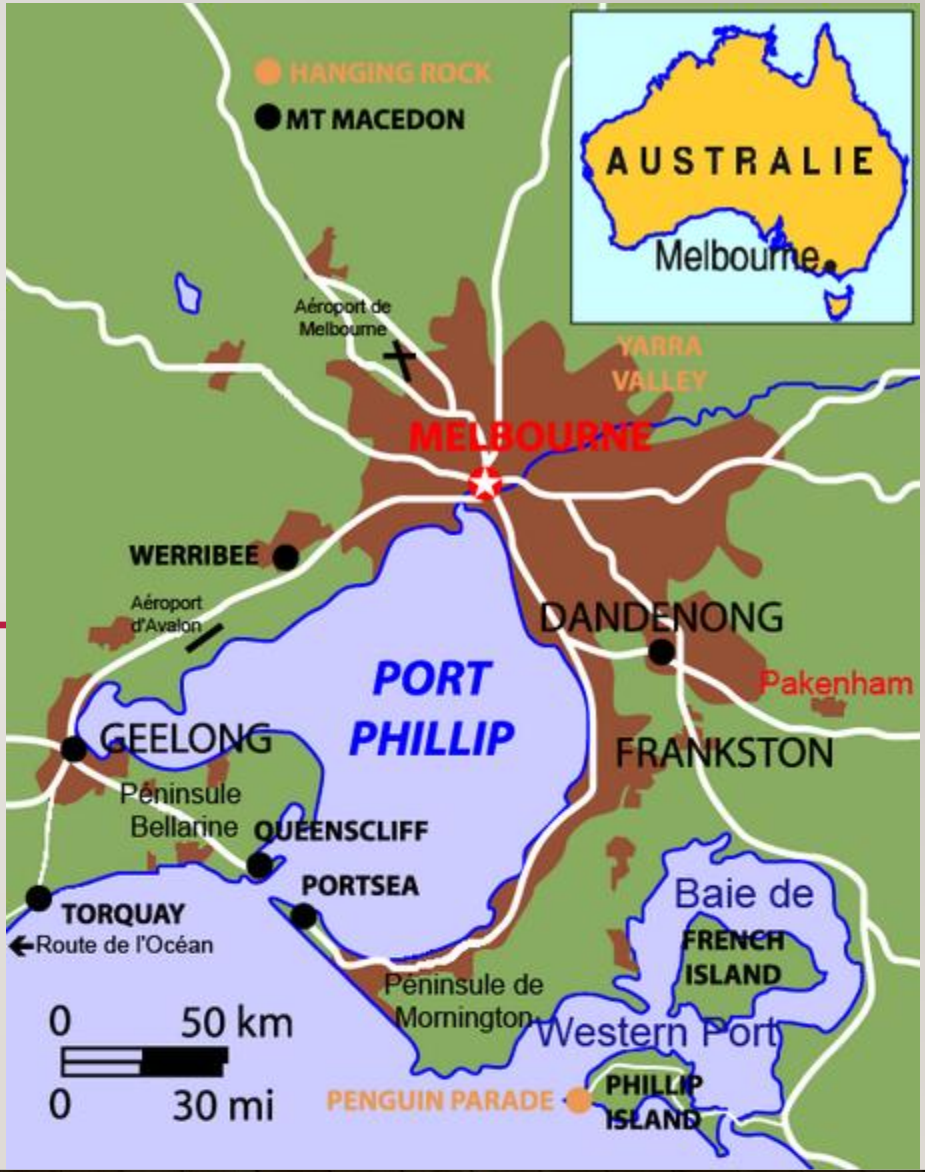
MELBOURNE HOUSING DATA



Objective:

We are going to use the Melbourne housing data and try to predict the >>price<< of a house using Supervised Machine Learning algorithms.

Source file:

the data is taken from Kaggle.com and has values for three years: 2016,2017 and 2018

SUMMARY OF THE ANALYSIS

- 1. A first look to the dataset
- 2.Data Cleaning
- 2.1.Missing Values
- 2.2.Outliers
- 2.3.Feature Engineering
- 3. Visualisations
- 3.1 Boxplot and histogram
- 3.2 Time Series
- 3.3 Scatterplot
- 3.4 Map
- 4. Machine Learning
- 4.1 Model Comparison
- 4.2 Fine Tuning - hyperparameters
- a. *Random Search*
- b. *Grid Search*
- 4.3 Test set evaluation

1. A FIRST LOOK TO THE DATASET

- There are 21 columns and 34857 rows

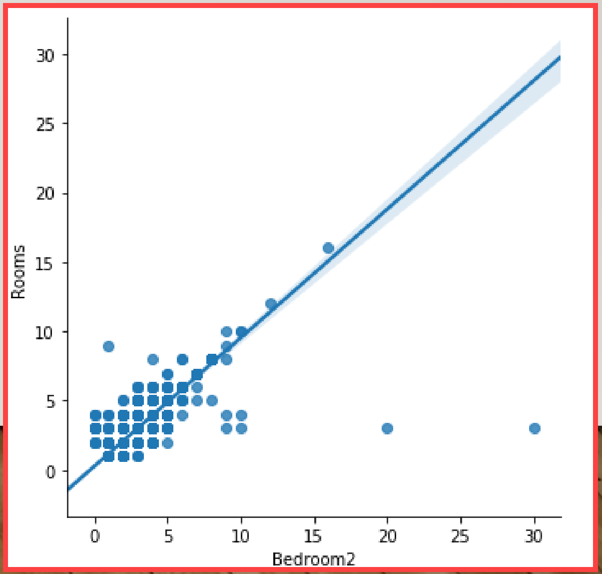
- We have insights about:
- - Rooms: 1 to 16
- - Price: 85K to 11,2M AUD
- - Distance to city center (CBD)
  - 0 – 48 km
- - Land Size & Building Area
- - Type of a house:
  - h - house, cottage, villa
  - t – townhouse
  - u – unit, duplex

- Region Name - General Region (West, North West, North, North east)
- Year Built: 1965 - 2018
- Latitude and Longitude
- Bathrooms and Car spots
  - 0-12 / 0-26
- Suburb
- Property count: number of properties in the Suburb

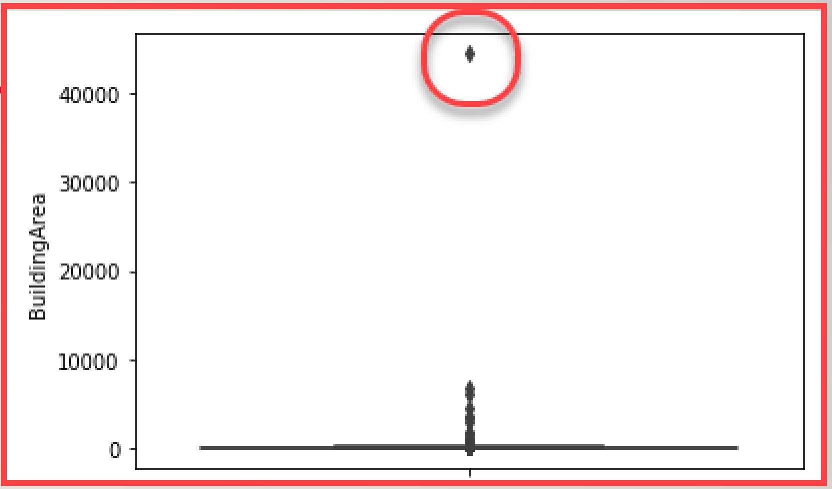
1. A FIRST LOOK TO THE DATASET

- As we read the description of each column we see that:
  - Bathroom, Car are floats but should be integer
    - Because there are NULL values we'll wait after fixing that
  - Postcode is float but we should change it to a category type
  - Yearbuilt should be an integer
  - Propertycount should be an integer
  - Rooms and Bedroom2 seem to represent the same data
    - Out of 34854 records, only 9162 have different values
    - From those 9162, 8217 are NULLs which means approximately 90%.

```
ValueCounts:Bathroom
----------------
 1.0      12969
 2.0      11064
NaN        8226
 3.0       2181
 4.0        269
 5.0         77
 0.0         46
 6.0         16
 7.0          4
 8.0          3
 12.0         1
 9.0          1
Name: Bathroom, dtype: int64
```



- Analyze 'Building Area' column
  - The boxplot shows an outstanding outlier, with a building area of more than 40000 squared meters.



- We notice the building has only 5 rooms,

3 bathrooms and the 'BuildingArea' is bigger

than the 'LandSize' it might me an error.

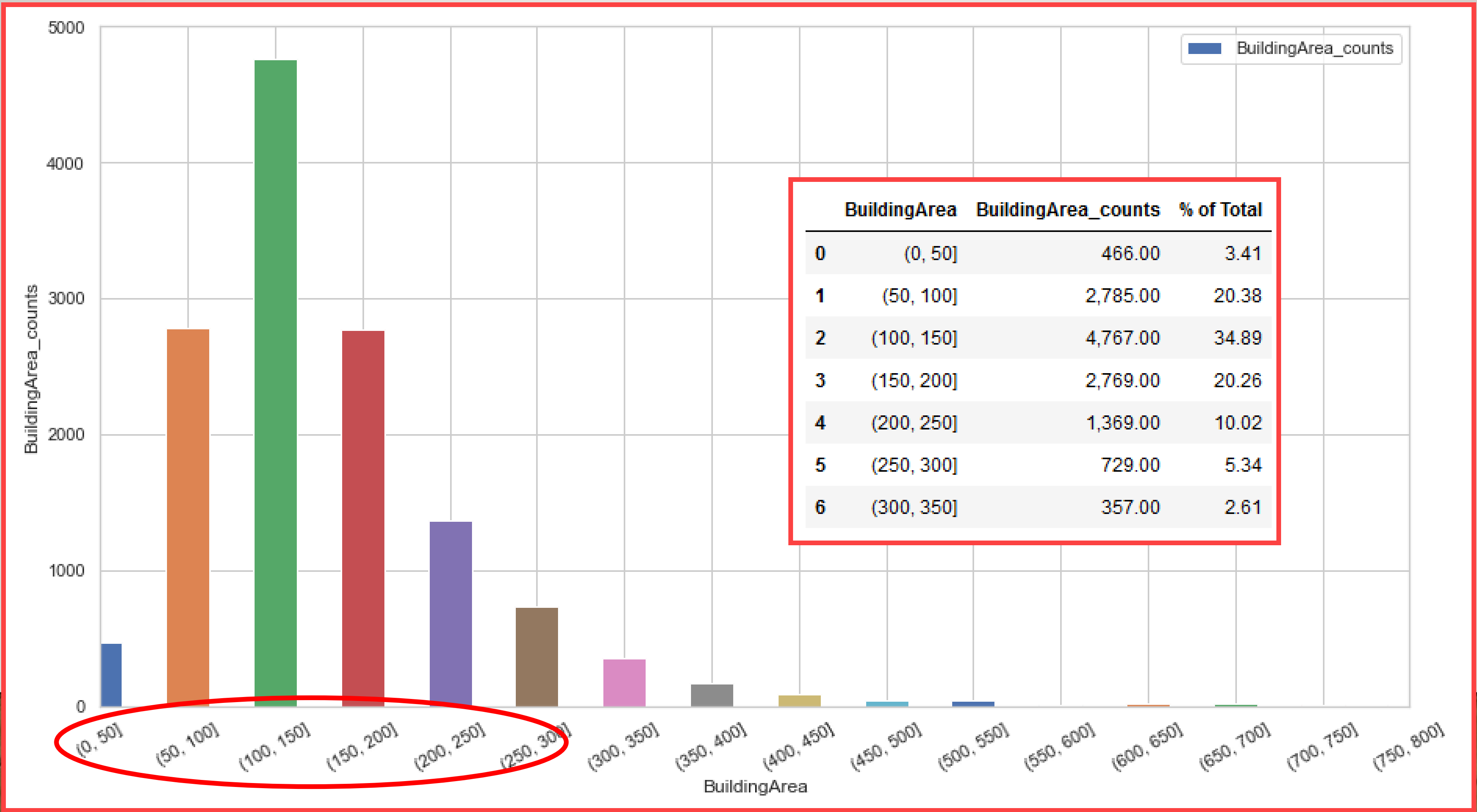
The YearBuilt is also missing.

However this property could be an old factory with lots of land.

We can safely drop this row.

	index	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	Bathroom	Car	Landsize	BuildingArea	YearBuilt
22614	22632	New Gisborne	71 Hamilton Rd	5	h	1355000.0	S	Raine	23/09/2017	48.1	3438.0	3.0	5.0	44500.0	44515.0	NaN

- After dropping the most extreme outlier, we can group BuildingArea into bins of ~50 sqm to see on a plot bar how are the houses distributed.
- Most of the houses are between 100 and 200 sqm (~55%)



	BuildingArea	BuildingArea_counts	% of Total
0	(0, 50]	466.00	3.41
1	(50, 100]	2,785.00	20.38
2	(100, 150]	4,767.00	34.89
3	(150, 200]	2,769.00	20.26
4	(200, 250]	1,369.00	10.02
5	(250, 300]	729.00	5.34
6	(300, 350]	357.00	2.61

2. DATA CLEANING

- 2.1. Missing values
  - Out of a total of ~34750 rows some features have a lot of NULLs

Price	7594
Distance	1
Postcode	1
Bathroom	8226
Car	8726
Landsize	11790
BuildingArea	21115
YearBuilt	19303
CouncilArea	3
Lattitude	7976
Longtitude	7976
Regionname	3
Propertycount	3

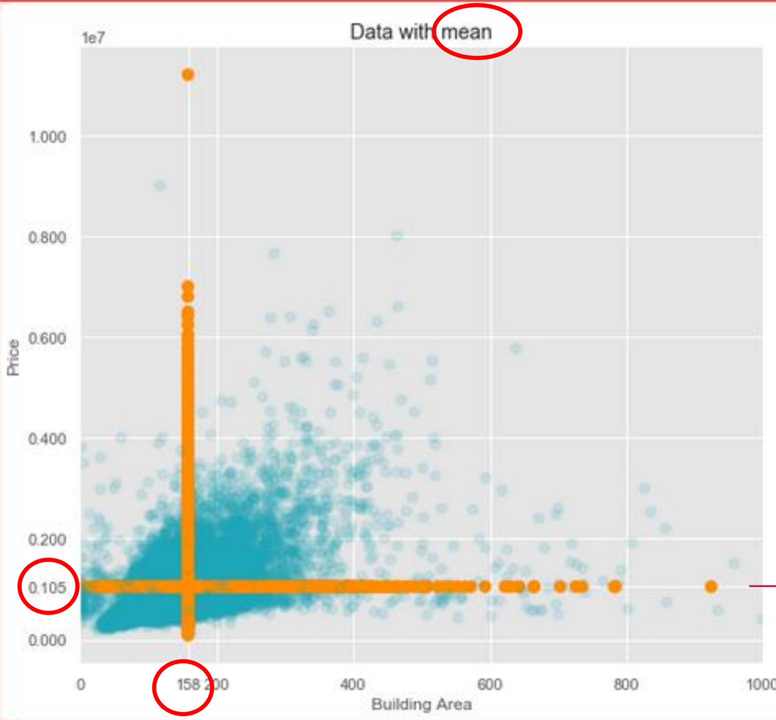
- We'd evaluate whether it is better to impute the missing values with their mean or median.

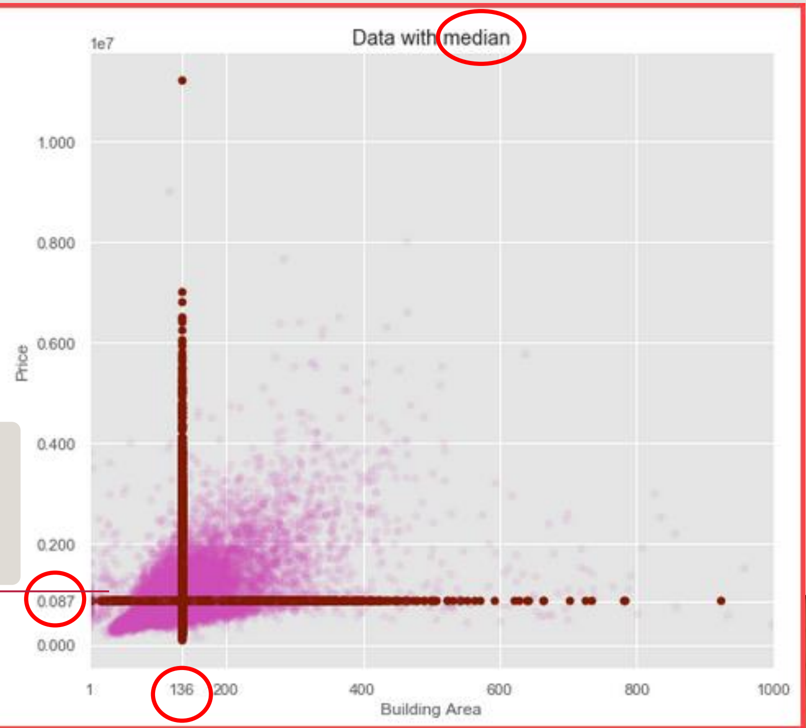
In both graphs we can see where the imputed values would be.

There is not a striking visual difference between the two graphs. However, after a careful look we can say that:

Price has a lot of outliers, and the median is less affected by them, as the horizontal line in the graph on the right is lower.

Chosen Solution: Impute NULLs in BuildingArea with median from ['Regionname' and 'Suburb']





The median for groups:

	Regionname	Suburb	BuildingArea
0	Eastern Metropolitan	Bayswater	134.2
1	Eastern Metropolitan	Bayswater North	192.5
2	Eastern Metropolitan	Bellfield	116.7
3	Eastern Metropolitan	Blackburn	183.8
4	Eastern Metropolitan	Blackburn North	167.7

2. DATA CLEANING

- 2.2. Outliers

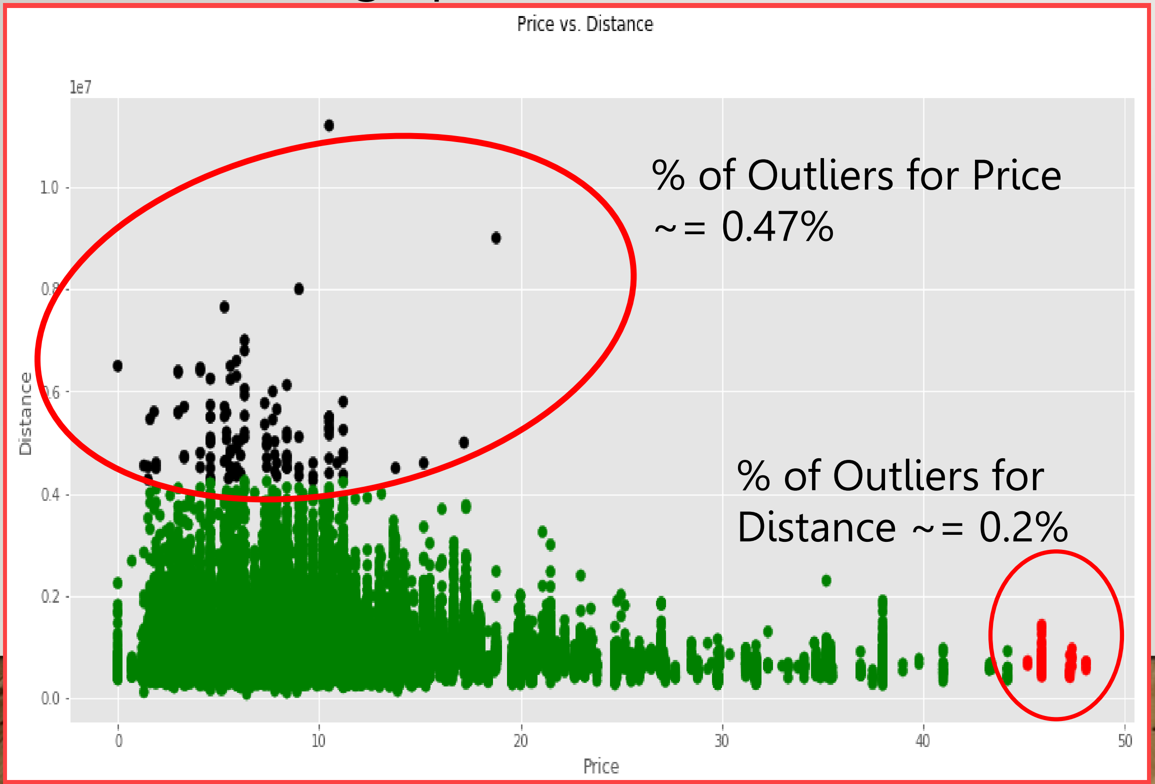
Let's visualize Price and Distance to spot outliers:

> each point is either an outlier either for price or distance

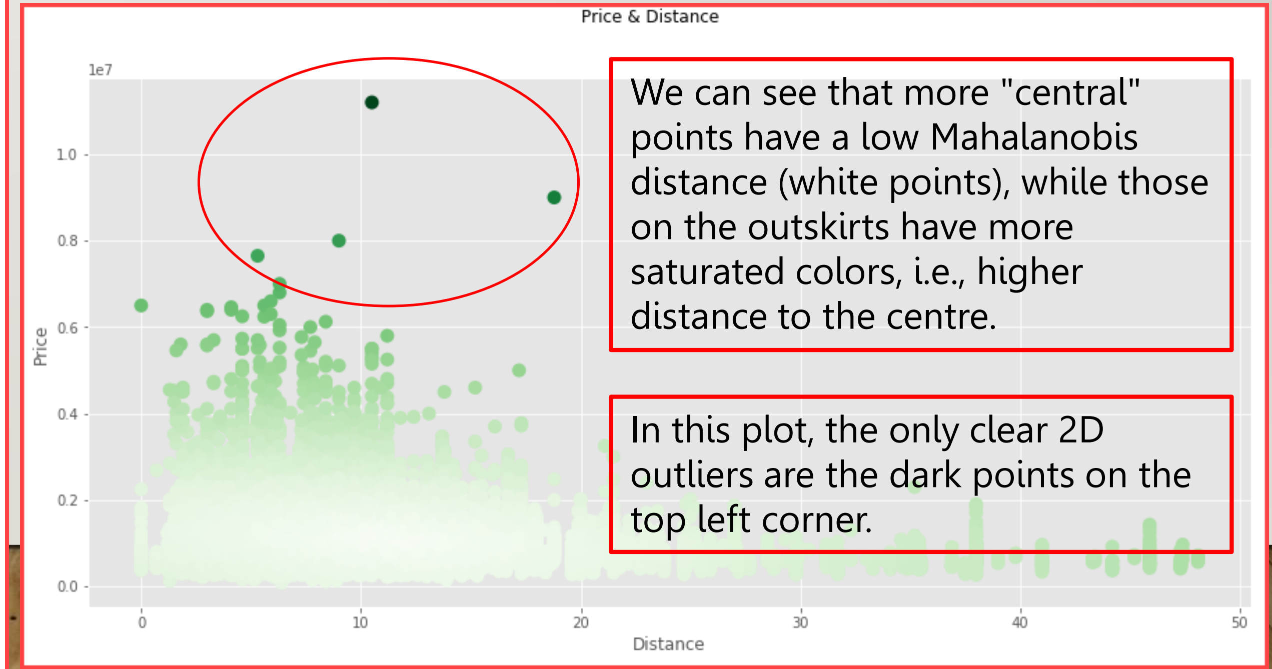
Price	7594
Distance	1
Postcode	1
Bathroom	8226
Car	8726
Landsize	11790
BuildingArea	21115
YearBuilt	19303
CouncilArea	3
Lattitude	7976
Longtitude	7976
Regionname	3
Propertycount	3

- I chose to keep an extremely high threshold to define outliers: over 5 std, where 2 or 3 would have been much more common.

Reason: the high prevalence of outliers found in this database.



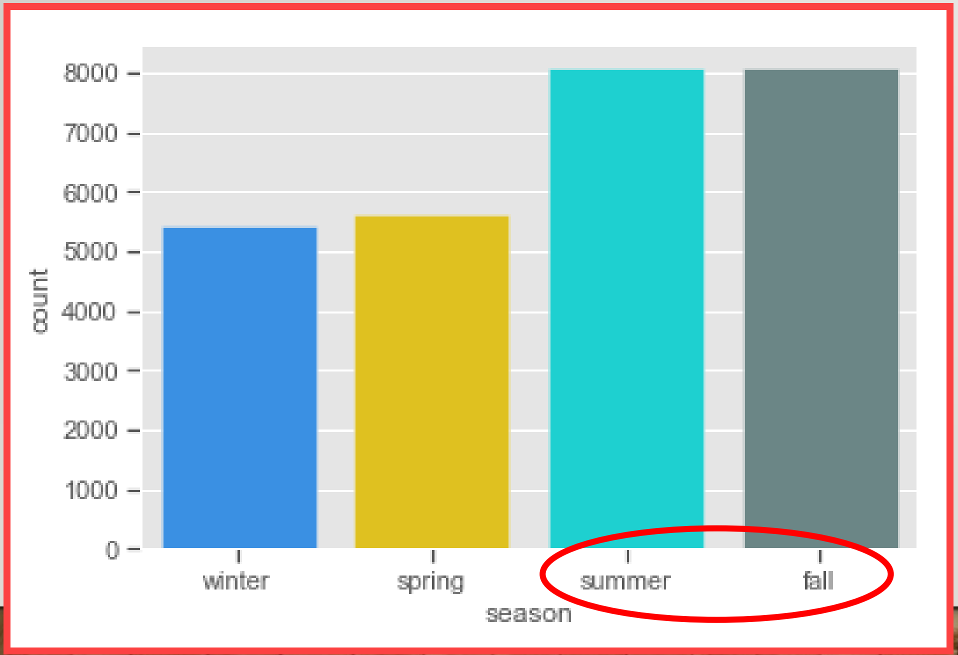
Graph with Mahalanobis distance in color:

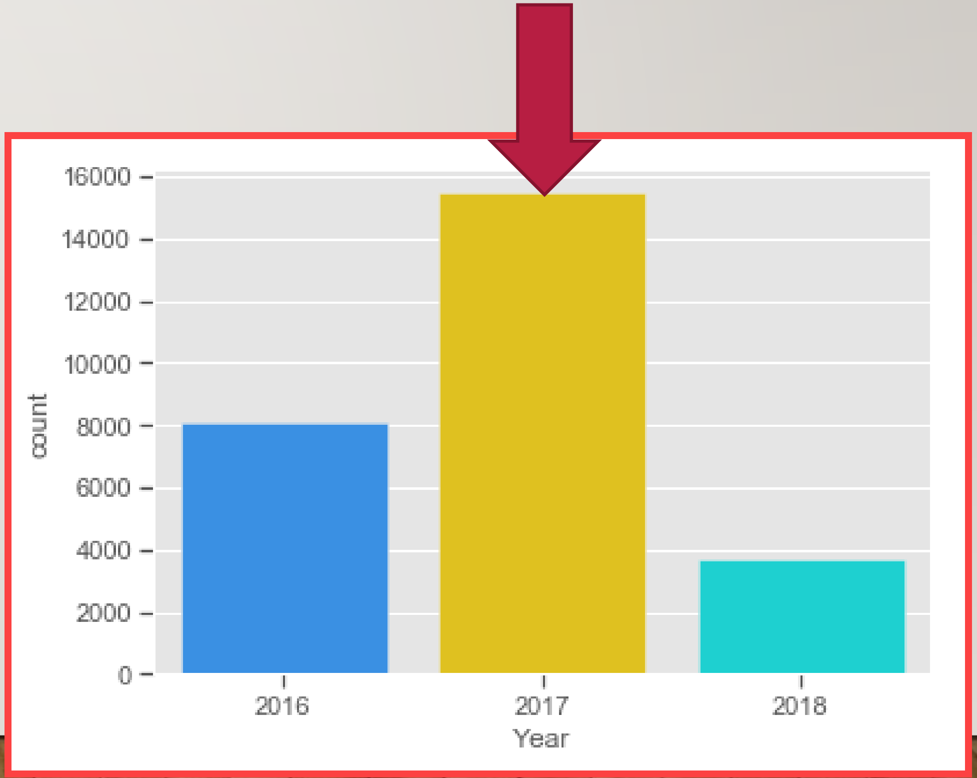


2.3. Feature engineering

- Create a new feature from DateSold as Season
and another asYear

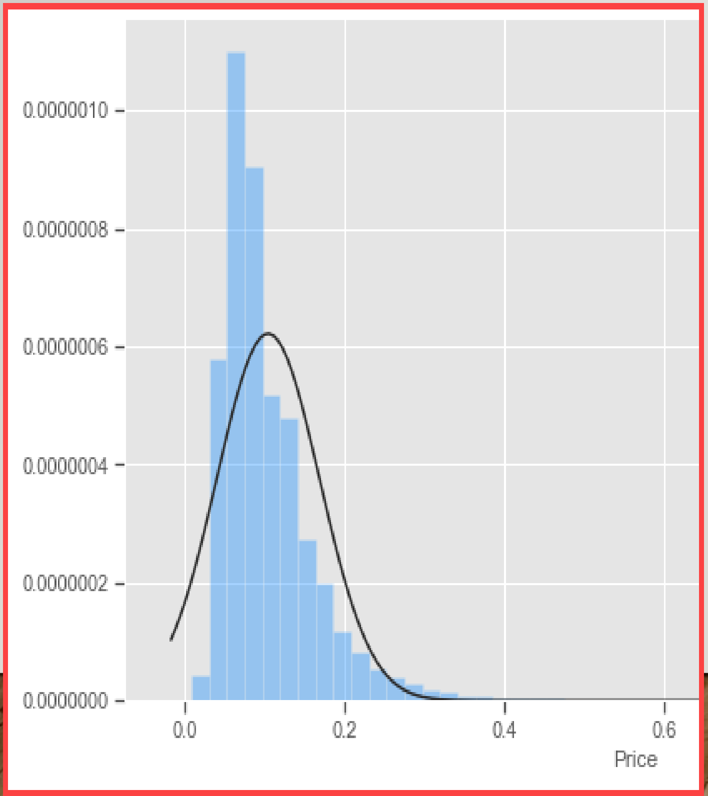
- Most properties were sold during summer and falls
- Most properties were sold in 2017

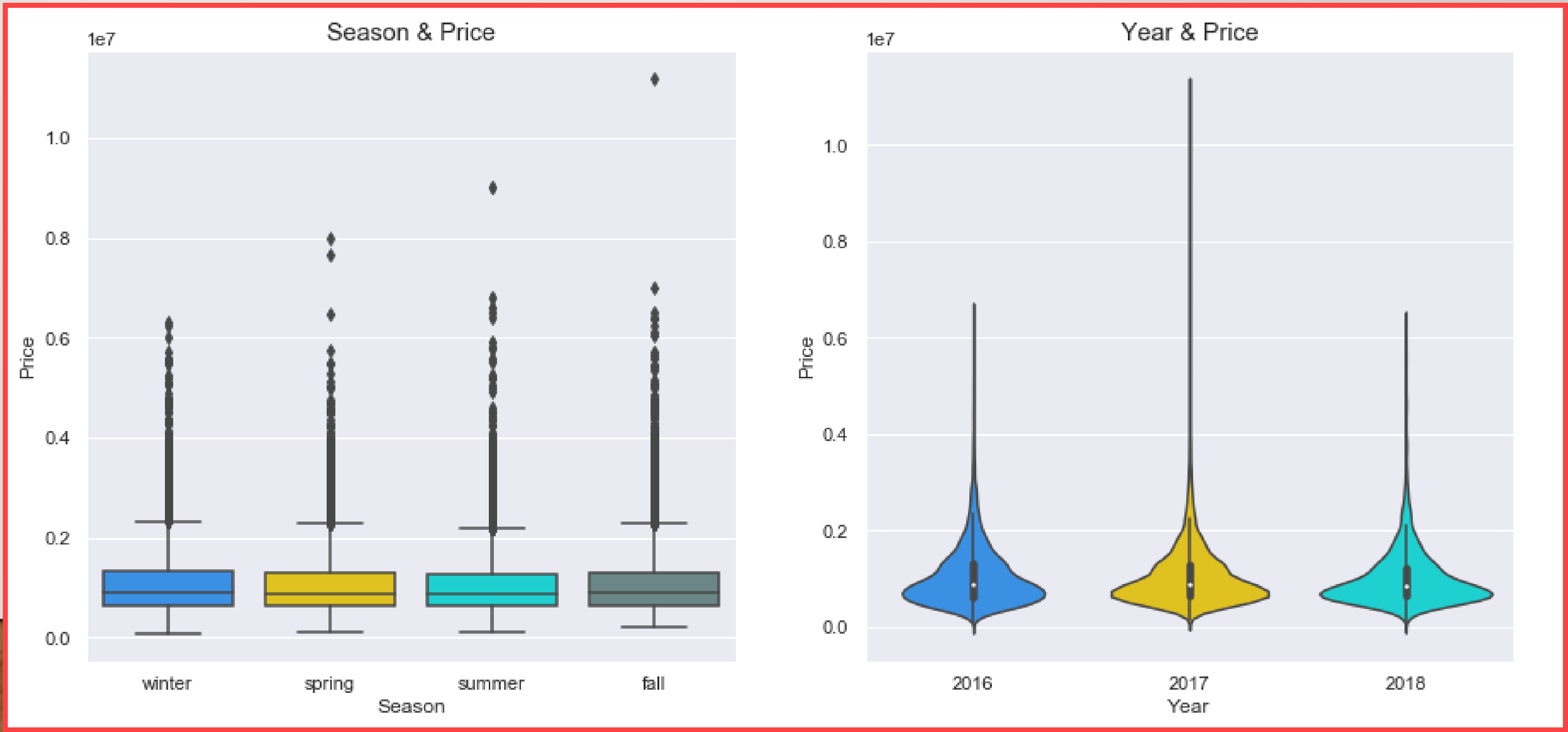




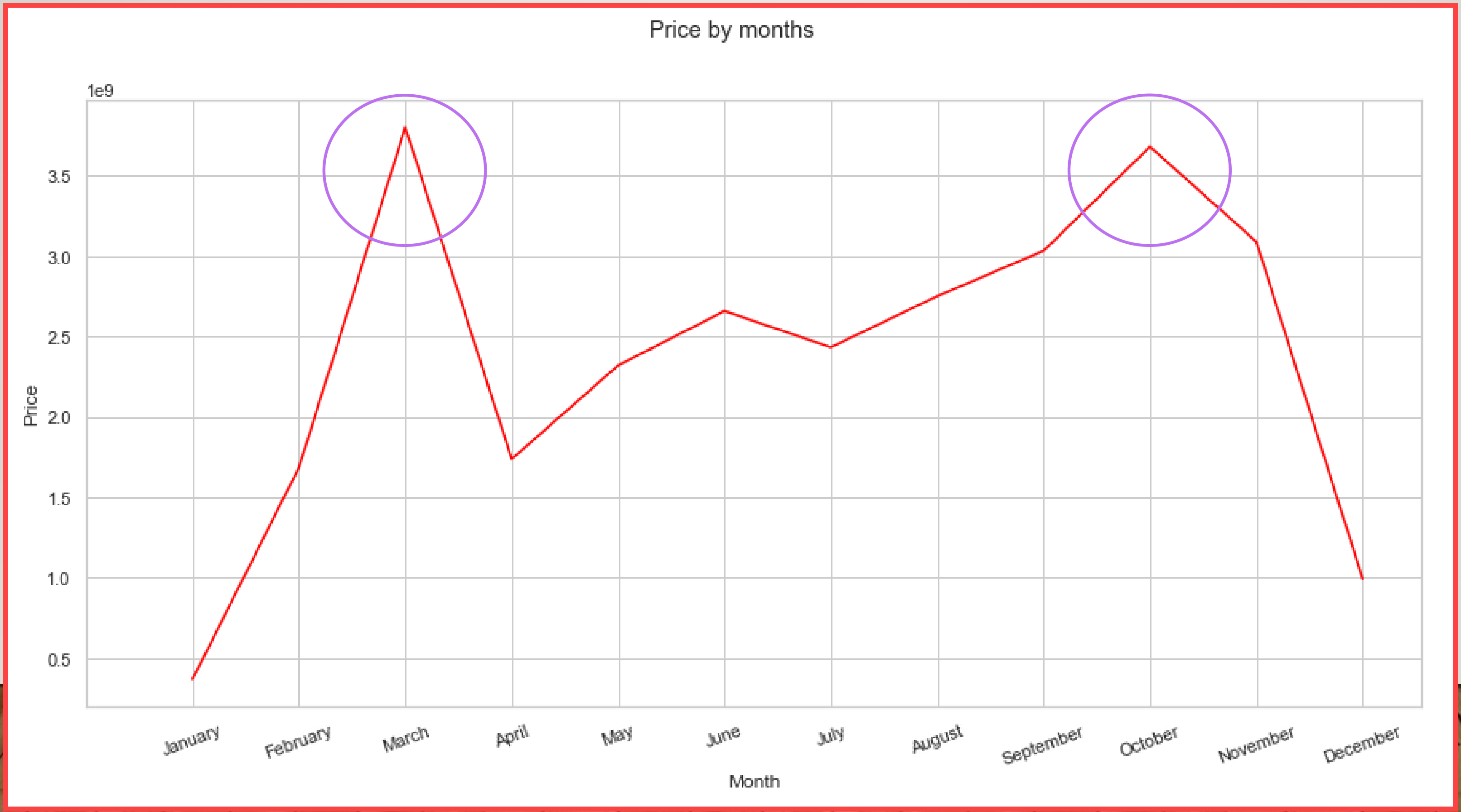
- 3. Visualizations
- 3.1. Boxplots and Histograms
  - The Price histograms shows right skewness towards big values.

  - The sold price during season is similar with some outliers
  - 2017 was indeed a better year with some outliers

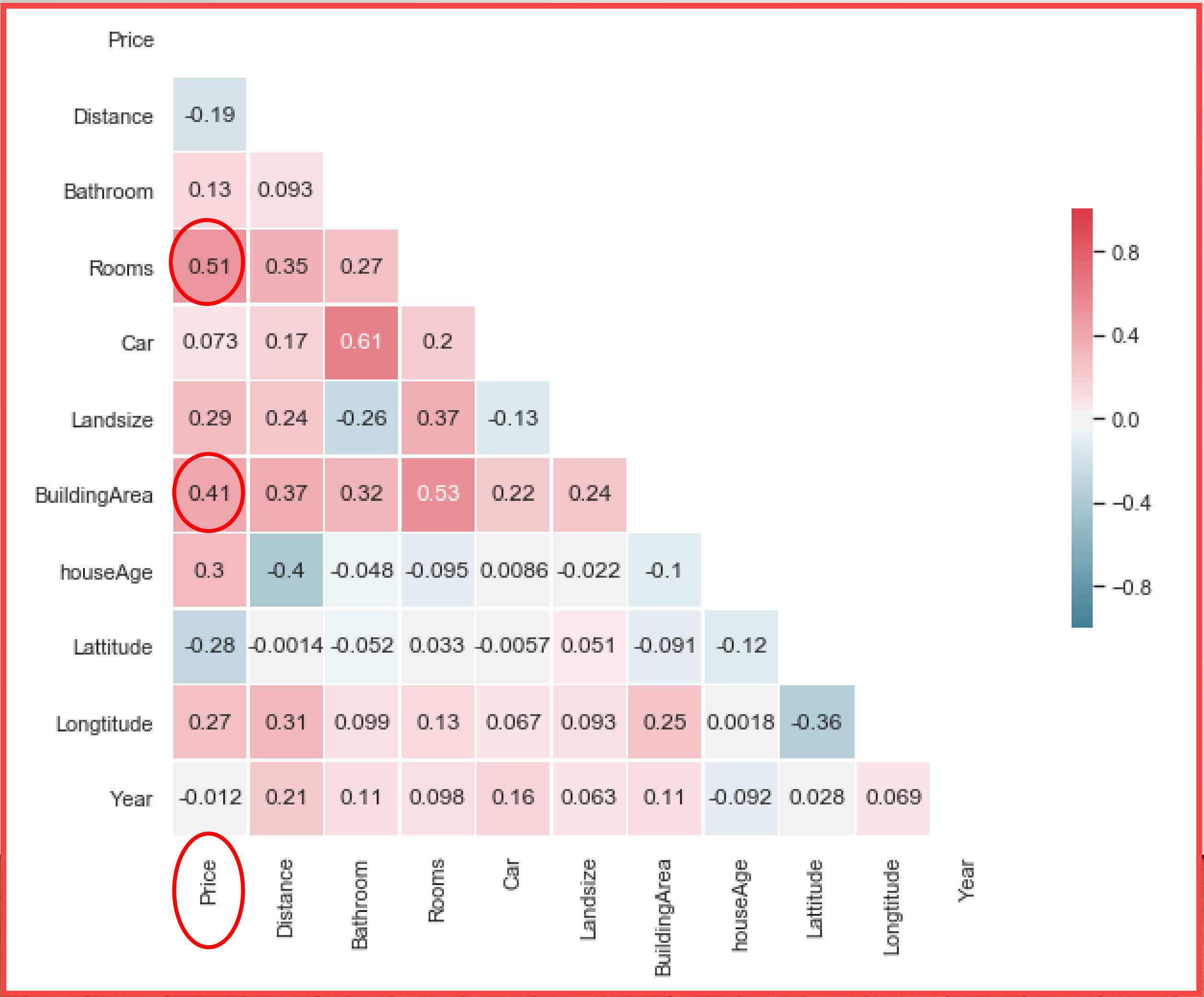


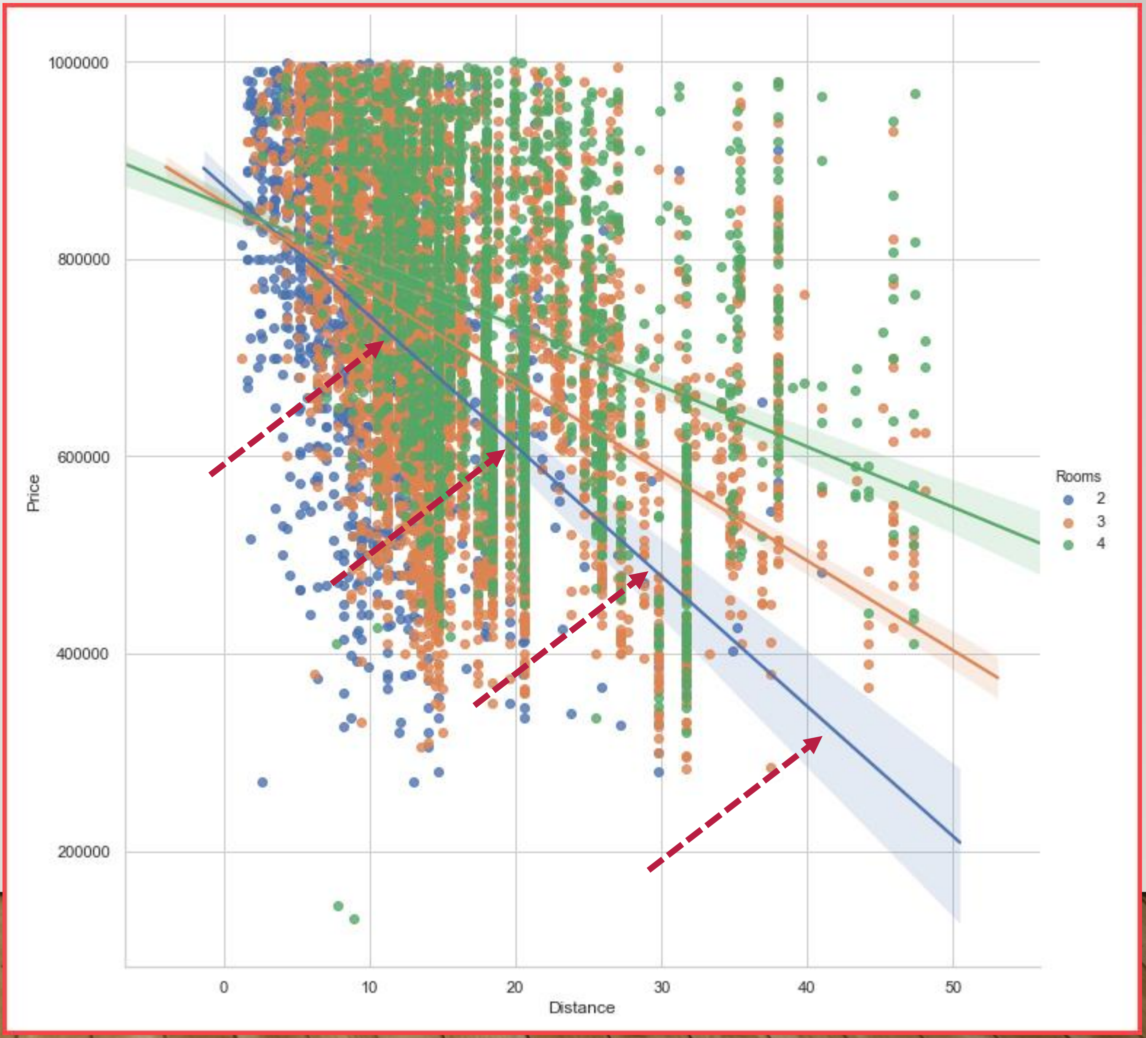


- 3. Visualizations
- 3.2. Time Series
  - Considering the size of the dataset I expected to have house sales for almost every day.
  - Unexpectedly, out of all the 2 and a half years, houses were sold only during 78 days
- We can also see how were distributed the sales M-by-M without the year value:



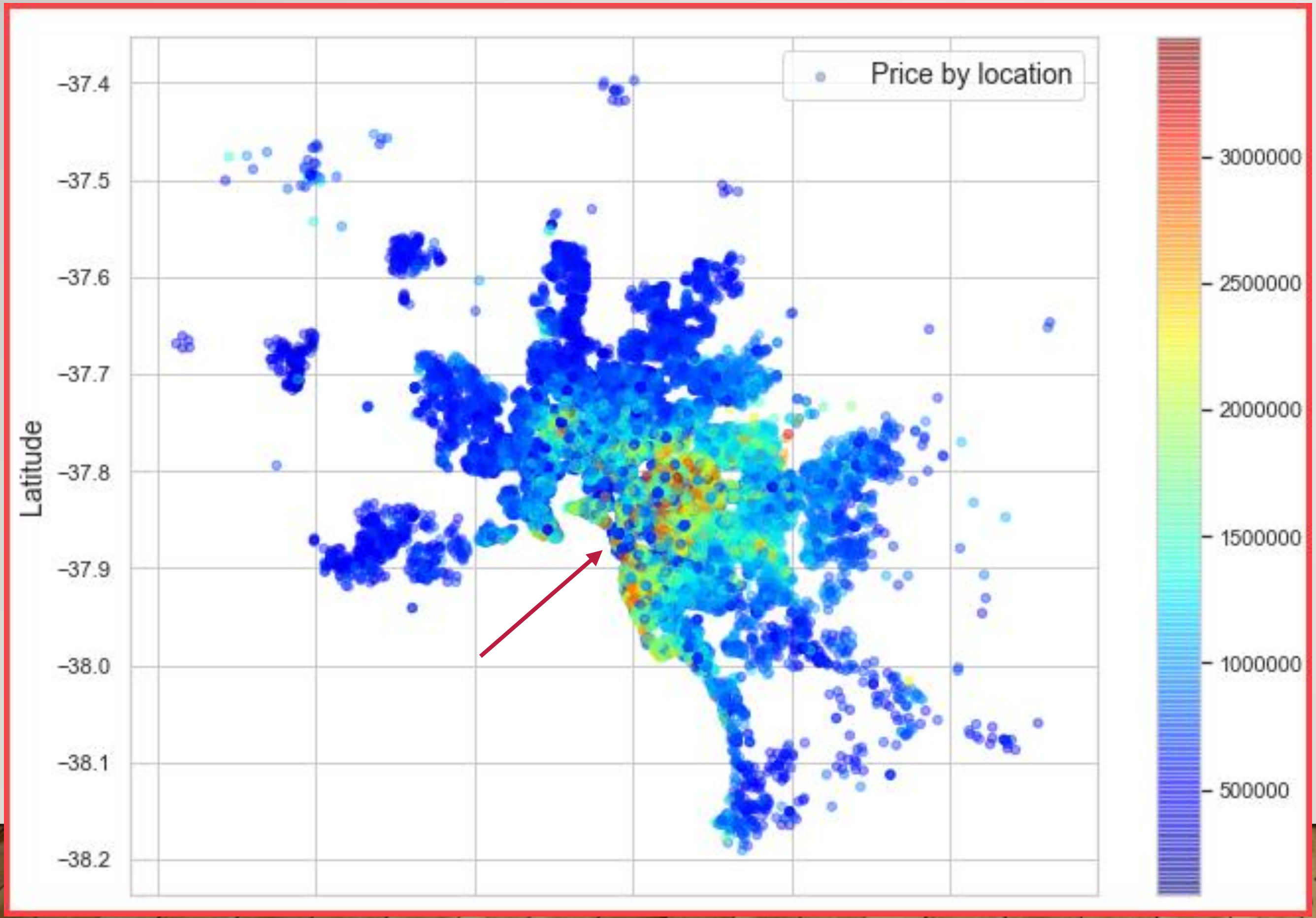
- 3.3. Correlation matrix and scatterplots
  - Let us look at the correlations between our variables, first with a correlation matrix.
  - As we move further from CBD the price decreases with different rates depending on how many rooms the house has

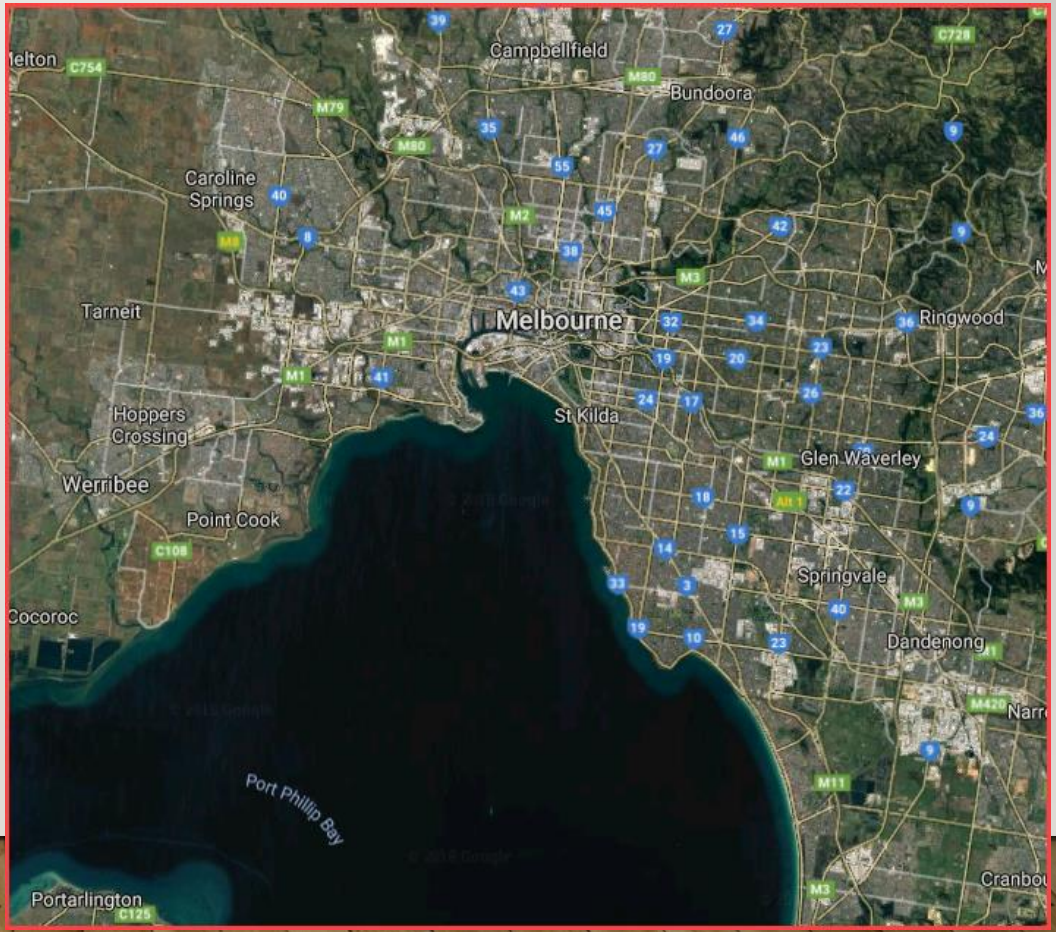




3.4. Geographical Data

- After removing some outliers that were turning the data to blue only, we can see that closer to the CBD are the most expensive houses.





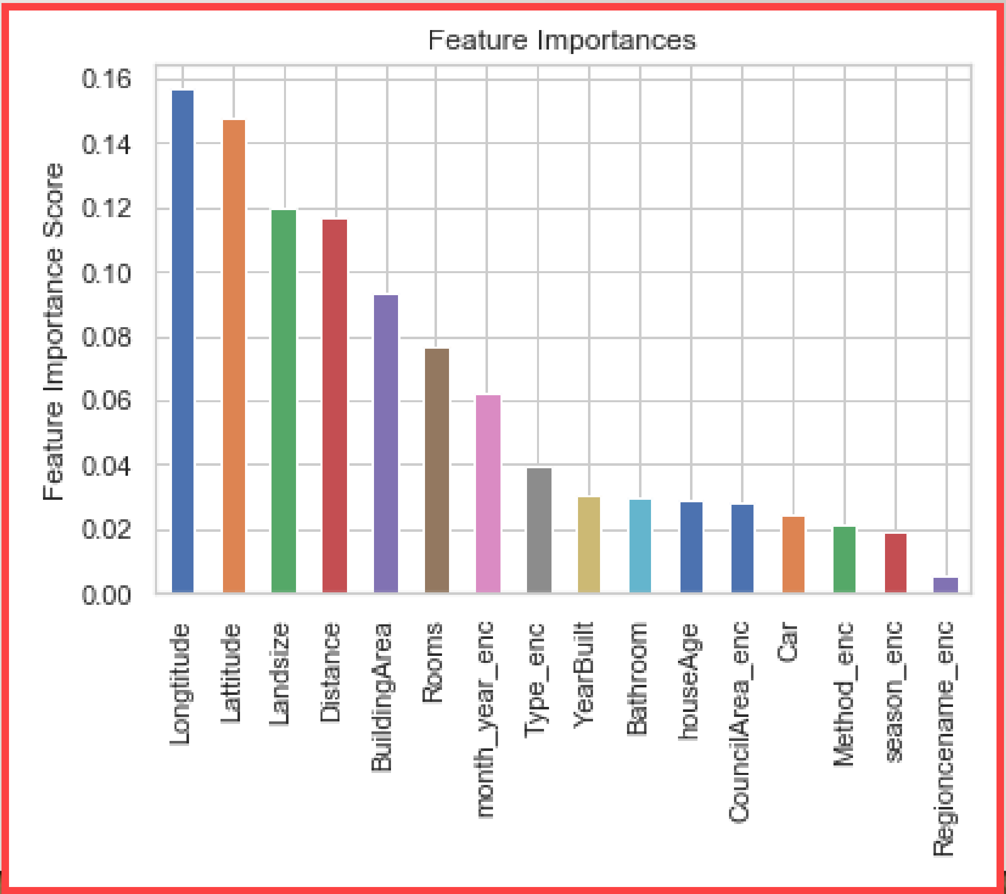
MACHINE LEARNING

- 4. Machine Learning
- 4.1 Model Comparison
- 4.2 Fine Tuning - hyperparameters
- a. *Random Search*
- b. *Grid Search*
- 4.3. Evaluate best model on the test set

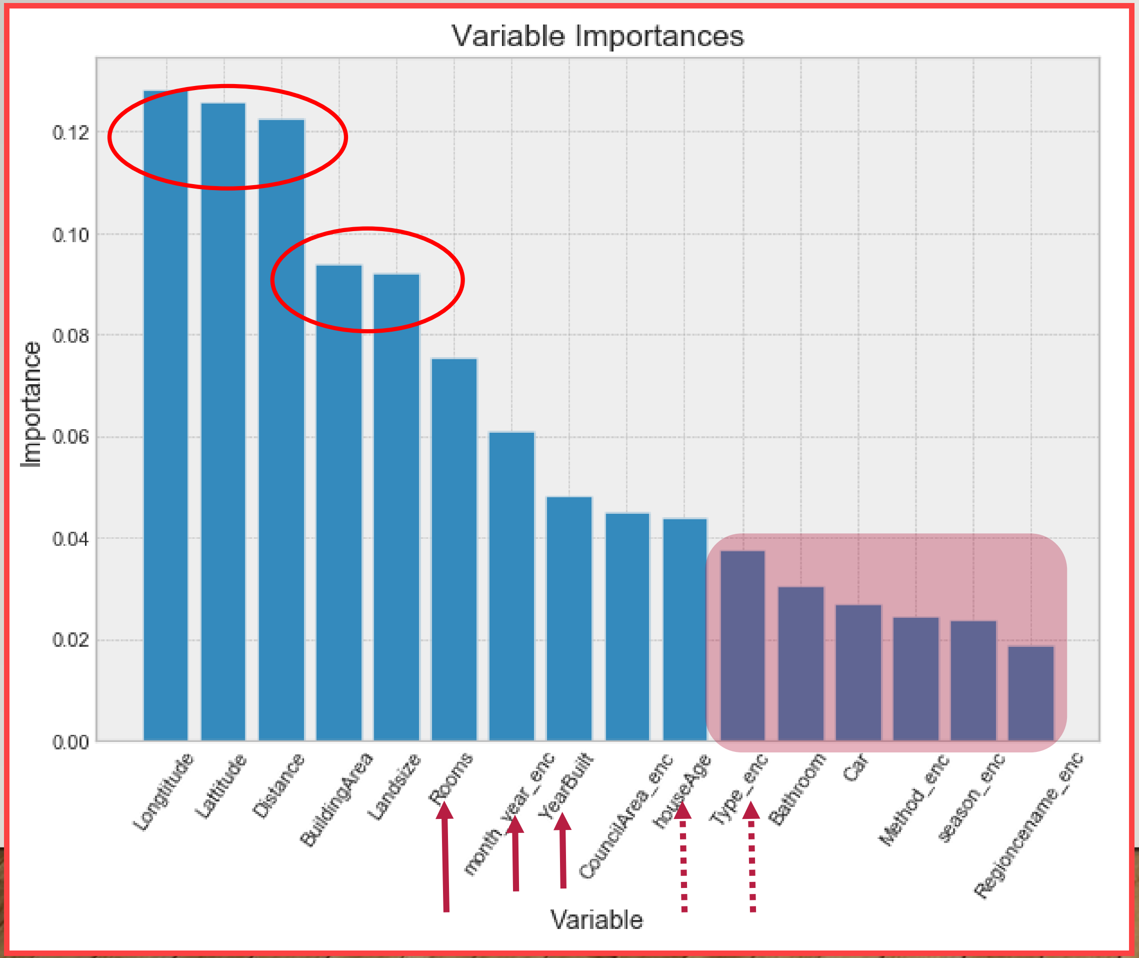
- 4. Machine Learning
- 4.1 Model Comparison

- Feature importances with Random Forest:

Models:	RMSE	R-squared



- Feature importance with GBM:
  - A future aim may be to cut the less relevant features (let's say everything after 'houseAge' in terms of importance), estimate a new model and compare it with the old ones. I reckon it would lose predictive power, but on the other hand it would gain in terms of training speed



* the best feature to reliably predict the price of a Melbourne house are actually 3: Lat&Long + Distance
* 'BuildingArea'&'Landsize' come second and are equal
>> Location and property size are weighing the most
* 'Rooms' are far ahead of 'houseAge' and 'Type'
* 'mont_year' when house was sold was also important
* Interesting is 'YearBuilt' is not as important as we may think

- 4. Machine Learning
- 4.2 Fine Tuning - Hyperparameter tuning
  - - hyperparameters are like the settings of an algorithm that can be adjusted to optimize performance

---

  - The randomized search and the grid search explore exactly the same space of parameters. The result in parameter settings is quite similar, while the run time for randomized search is drastically lower.
  - *a. Random Search* - we can define a grid of hyperparameter ranges, and randomly sample from the grid, performing K-Fold CV with each combination of values.
  - Random Forest - Hyperparameter tuning:

```
# 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
              }

random_grid
```

```
{'max_depth': [10, 20, 30, 40, 50, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [50, 287, 525, 762, 1000]}
```

```
rf_random.best_params_
```

```
{'max_depth': 30,
 'max_features': 'sqrt',
 'min_samples_leaf': 1,
 'min_samples_split': 5,
 'n_estimators': 1000}
```

- 4. Machine Learning
  - 4.2. Fine Tuning:
  - b. *Grid Search* - Now that we know where to concentrate our search we can narrow down the range for each hyperparameter.

```
param_test3 = {  'max_depth': [1,30,40,60]
                ,'max_features': ['sqrt']
                ,'min_samples_leaf': [1]
                ,'min_samples_split': [5 ]
                ,'n_estimators':[100,500,1000]}

gsearch3 = GridSearchCV(estimator = RandomForestRegressor()
```

```
print('best param: ',gsearch3.best_params_)
```

```
{'max_depth': 30,
 'max_features': 'sqrt',
 'min_samples_leaf': 1,
 'min_samples_split': 5,
 'n_estimators': 1000}
```

```
print('\nbest score R-squared: ',gsearch3.best_score_)
```

```
best score R-squared:  0.79520833
```

  - Gradient Boosting - Hyperparameter tuning
    - n_estimators = number of trees in the foreset
    - 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)

- Results with RandomizedSearchCV and with GridSearchCV

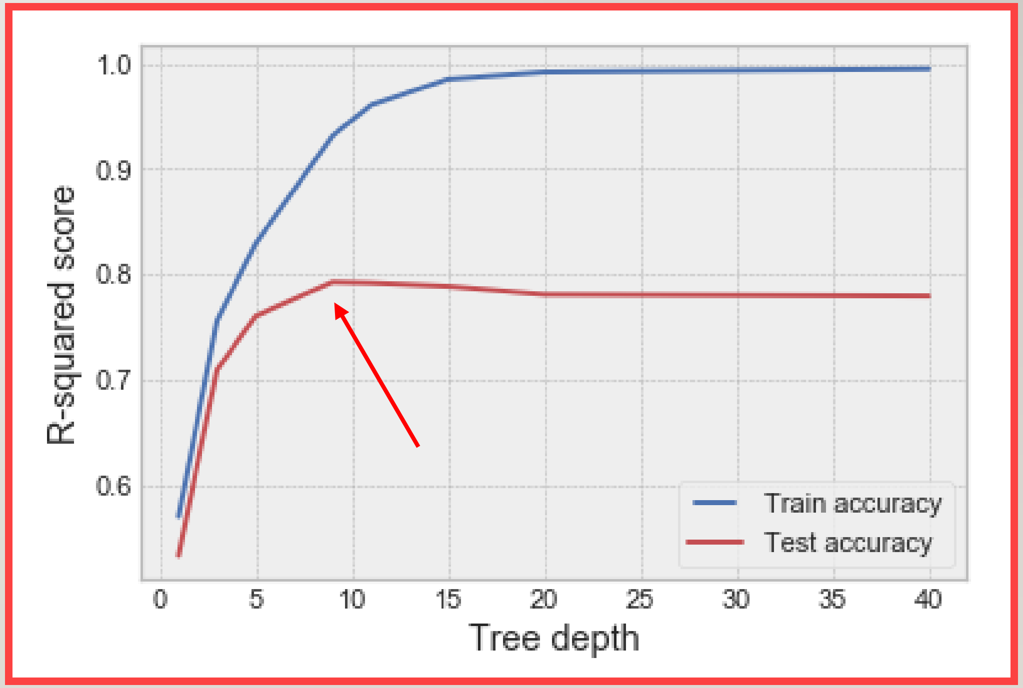
```
gb_random.best_params_
```

```
{'max_depth': 5,
 'max_features': 'sqrt',
 'min_samples_leaf': 4,
 'min_samples_split': 2,
 'n_estimators': 275}
```

```
best param:  {'n_estimators': 100, 'max_depth': 9}
best score R-squared w/ CV: 0.81047
```

- 4. Machine Learning
  - 4.3. Evaluate best model on the test set
  - As the time spent to tune parameters is significant I chose for us to see how the performance varies when we increase 'max_depth'

  - With this graph it is really easy to see where the overfitting begins: after ~9 splits the test performance does not increase anymore, while the train stops increasing after 20 splits.



- Future steps:
  - Work with categorical features to create dummies instead of using LabelEncoder;
  - Change 'Rooms' and 'Car' features to categorical type;
  - Remove some more outliers to see how the model performs;

- **THANK YOU !**