Introduction to Machine Learning in Python – Workshop 4

Presented by Jiajia Li

14.08.2024

Australian National University

Agenda

Recap – MAE or RMSE ?

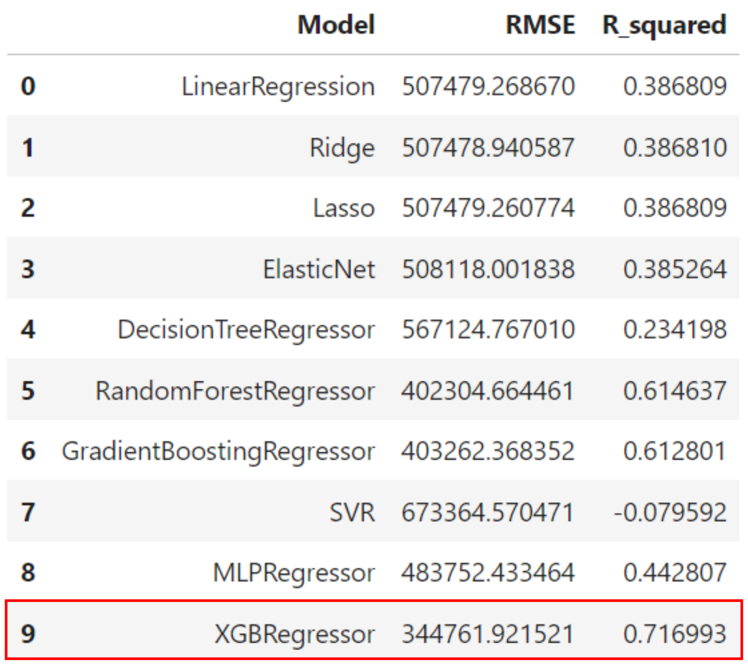
Choosing between Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) depends on what aspects of the error distribution are more important for your specific application.

MAE is less sensitive to outliers because it does not square the errors, treating all errors equally. Use MAE when you want a metric that is easy to interpret and not overly influenced by large errors.

RMSE is differentiable, making it more suitable for optimisation in many machine learning algorithms. Use RMSE when larger errors are particularly undesirable, and you want to penalise them more heavily.

Recap – Which model performs the best?

Calculate RMSE and R-squared values for all models.

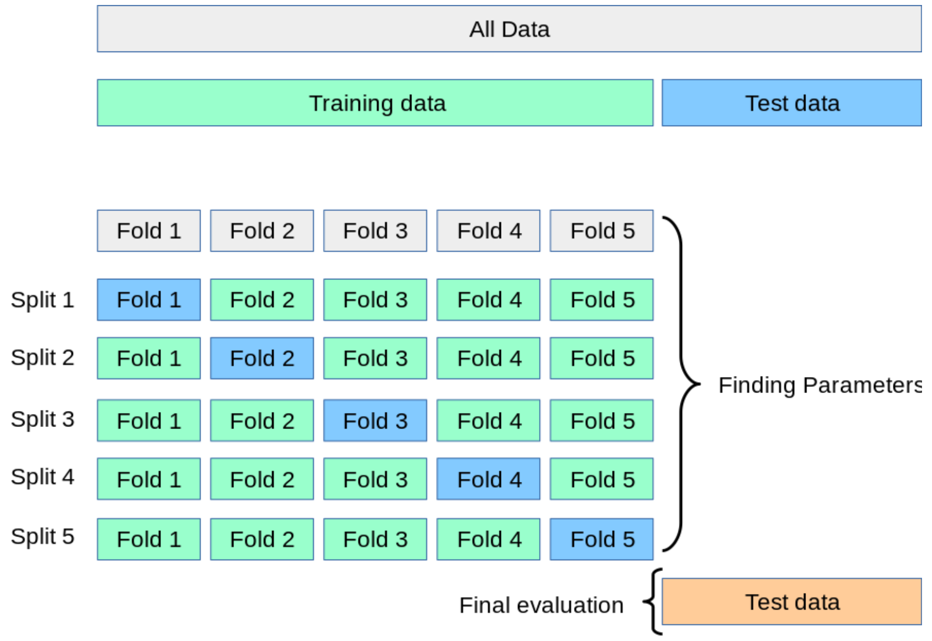


	Model	RMSE	R_squared
0	LinearRegression	507479.268670	0.386809
1	Ridge	507478.940587	0.386810
2	Lasso	507479.260774	0.386809
3	ElasticNet	508118.001838	0.385264
4	DecisionTreeRegressor	567124.767010	0.234198
5	RandomForestRegressor	402304.664461	0.614637
6	GradientBoostingRegressor	403262.368352	0.612801
7	SVR	673364.570471	-0.079592
8	MLPRegressor	483752.433464	0.442807
9	XGBRegressor	344761.921521	0.716993

From the table we can see that 'XGBRegressor' performs the best.

Cross Validation

Cross-validation is a statistical method used in machine learning to assess the performance of a model. It helps ensure that the model generalises well to unseen data by systematically splitting the data into training and validation sets.



Cross-validation

There are several ways of cross-validation:

- K-fold cross-validation
- Leave-one-out cross-validation
- Stratified k-fold cross-validation

Benefits of cross-validation:

- **Improved accuracy estimate**: by averaging the results over multiple folds, cross-validation gives a more accurate estimate of model performance.
- **Better generalisation**: it helps in identifying how well the model generalises to unseen data.
- **Reduces overfitting**: by testing the model on different subsets, it reduces the risk of overfitting to the training data.

Cross-validation

There are also a few draw-backs:

- **Computationally intensive**: cross-validation, especially with many folds, can be computationally expensive.
- **Complexity in interpretation**: while it provides a good estimate, it can sometimes be difficult to interpret the results when the variance between folds is high.

When to use cross-validation:

- **Model selection**: when you're comparing multiple models and want to choose the one that performs best on unseen data.
- **Parameter tuning**: to fine-tune the hyperparameters of a model by evaluating how changes impact performance across different folds.
- **Assessing model stability**: to determine if a model's performance is consistent across different subsets of data.

K-fold cross-validation

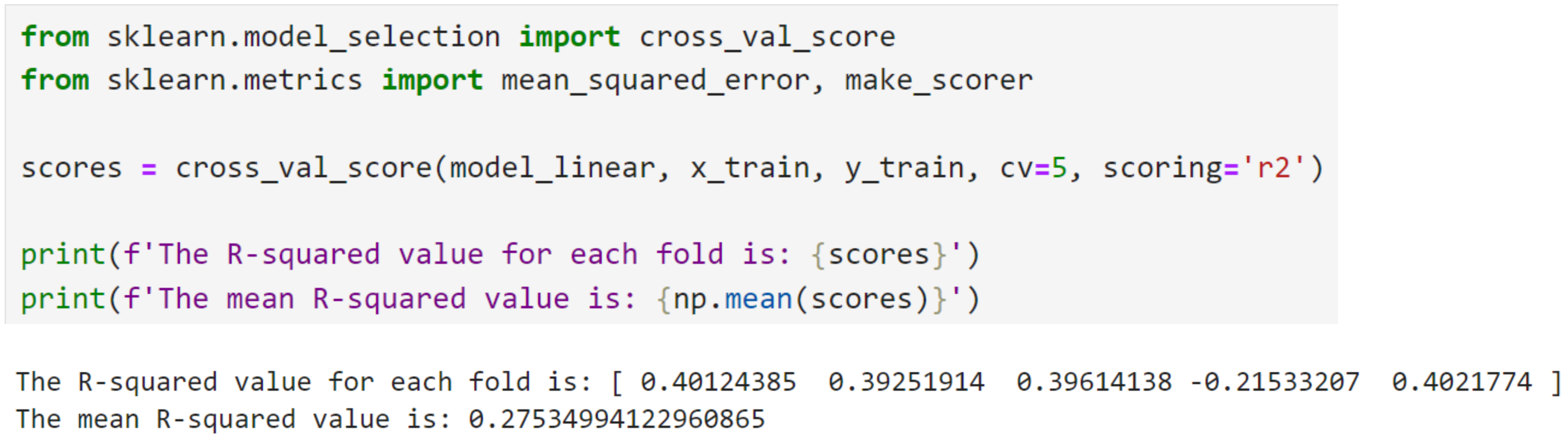
The dataset is randomly divided into **k** equally sized folds.

The model is trained on **k−1** folds and validated on the remaining fold. This process is repeated **k** times, with each fold being used as the validation set once.

After **k** iterations, the performance metrics (e.g., accuracy, MSE, R^2) from each iteration are averaged to provide a more robust estimate of the model’s performance.

- **Choice of k**: A common choice is k=5 or k=10. Larger k values give a more accurate estimate but are computationally expensive.
- **Shuffling**: it's a good practice to shuffle the data before splitting into folds to ensure randomness.

K-fold cross-validation



```python
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, make_scorer

scores = cross_val_score(model_linear, x_train, y_train, cv=5, scoring='r2')

print(f'The R-squared value for each fold is: {scores}')
print(f'The mean R-squared value is: {np.mean(scores)}')
```

```
The R-squared value for each fold is: [ 0.40124385  0.39251914  0.39614138 -0.21533207  0.4021774 ]
The mean R-squared value is: 0.27534994122960865
```

The evaluation scores for each fold are varied. This indicates our model did not generalise well. There also may be other reasons we got this result, e.g., outliers.

Leave-one-out cross-validation

Leave-One-Out Cross-Validation (LOO-CV) is a special case of K-fold cross-validation where the number of folds k is equal to the number of observations in the dataset.

Each fold consists of a single observation used for validation, while the remaining $n-1$ observations are used for training.

When to Use LOO-CV:

- LOO-CV is particularly useful when you have a **very small dataset**, where splitting the data into multiple folds would result in too small training sets.
- If your concern is **overfitting**, LOO-CV might help by using as much data as possible for training but be cautious of the high variance in the performance estimate.

Stratified k-fold cross-validation

Stratified K-Fold Cross-Validation is particularly useful when dealing with imbalanced datasets.

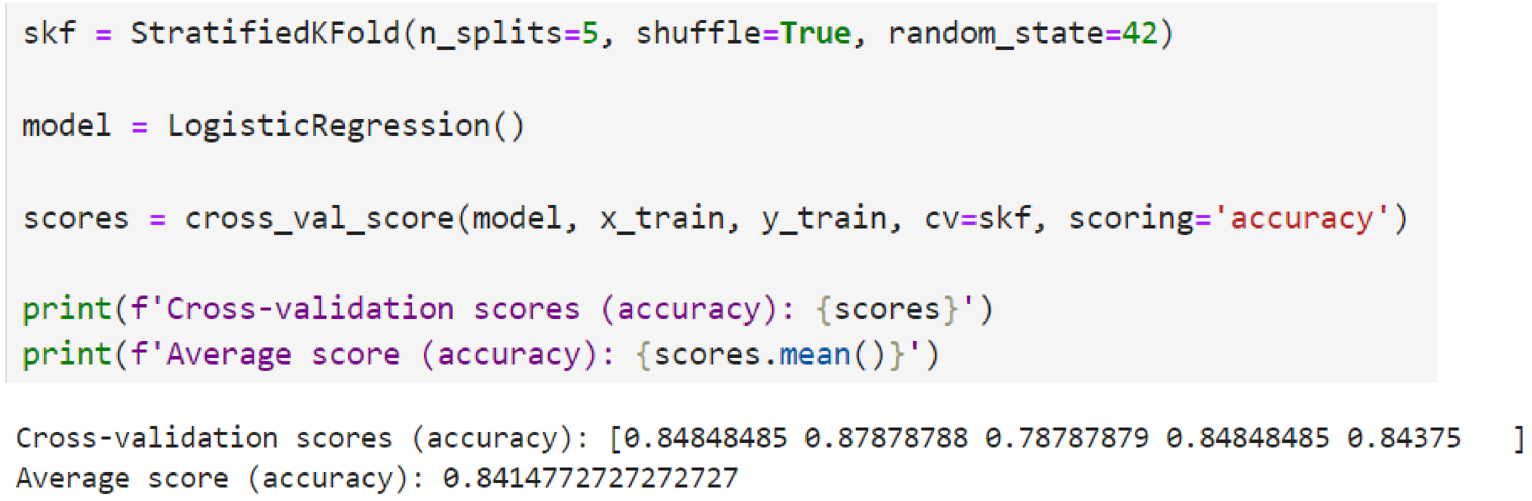
Each fold has approximately the same proportion of classes as the original dataset.

This ensures that each fold is representative of the entire dataset, which is crucial for classification problems where some classes may be underrepresented.

When to Use:

Use Stratified K-Fold Cross-Validation whenever you're working with a **classification problem**, especially if the classes in your dataset are **imbalanced**. It's an essential technique for ensuring that your model evaluation is fair and reliable.

Stratified k-fold cross-validation



```python
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

model = LogisticRegression()

scores = cross_val_score(model, x_train, y_train, cv=skf, scoring='accuracy')

print(f'Cross-validation scores (accuracy): {scores}')
print(f'Average score (accuracy): {scores.mean()}')
```

```
Cross-validation scores (accuracy): [0.84848485 0.87878788 0.78787879 0.84848485 0.84375    ]
Average score (accuracy): 0.8414772727272727
```

For stratified k-fold cross-validation, we can only apply it to classification models. So here, I have applied it to the eICU dataset with a logistic regression model.

You can find the list of available scoring metrics here.

Hyperparameter tuning

Hyperparameter tuning is the process of finding the most optimal set of hyperparameters for a machine learning model.

Unlike **model parameters** (slope, y-intercept), which are learned during the training process, hyperparameters are set before the training begins and control the behaviour of the training algorithm.

Common hyperparameters:

- **Learning rate**: control the step size in optimisation.
- **Batch size**: number of samples processed before the model is updated.
- **Regularisation parameters** (e.g., L1, L2 regularisation).
- **Number of trees** in a random forest.
- **Number of layers** in a neural network.

Grid search

Grid search is a hyperparameter optimisation technique used in machine learning to find the best combination of hyperparameters for a model.

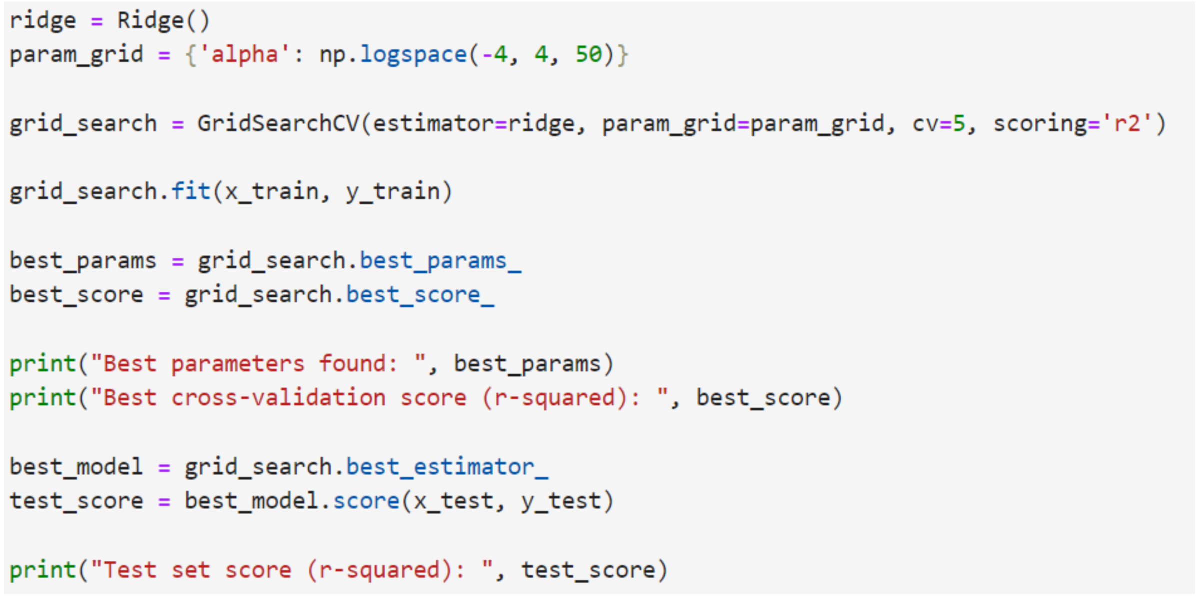
Scikit-learn has a built-in function of ‘GridSeachCV()’ to apply grid search.

Let’s try to use grid search with **Ridge Regression**.

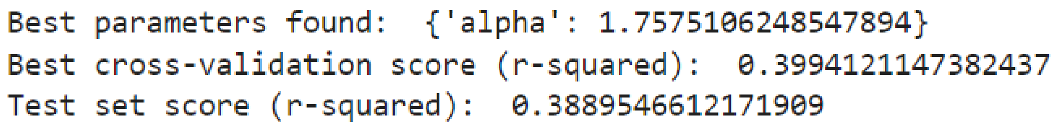
For Ridge Regression, there is only one hyperparameter we can tune, which is **alpha**.

Alpha is a constant that multiplies the L2 term, controlling regularisation strength. Alpha must be a non-negative float.

Grid search – Ridge Regression



```
ridge = Ridge()
param_grid = {'alpha': np.logspace(-4, 4, 50)}

grid_search = GridSearchCV(estimator=ridge, param_grid=param_grid, cv=5, scoring='r2')

grid_search.fit(x_train, y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best parameters found: ", best_params)
print("Best cross-validation score (r-squared): ", best_score)

best_model = grid_search.best_estimator_
test_score = best_model.score(x_test, y_test)

print("Test set score (r-squared): ", test_score)
```

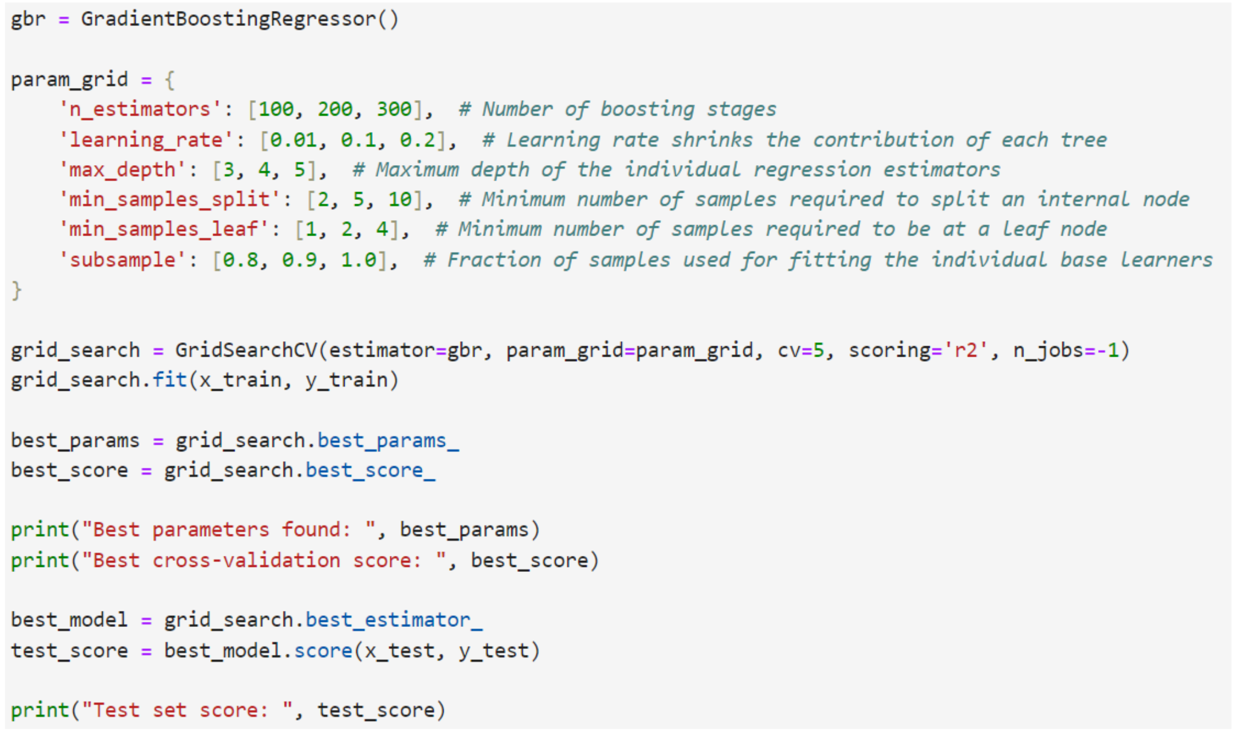


```
Best parameters found:  {'alpha': 1.7575106248547894}
Best cross-validation score (r-squared):  0.3994121147382437
Test set score (r-squared):  0.3889546612171909
```

If you compare the R-squared value we get this time with the one we get from setting **alpha=1.0**, the model with **alpha=1.757** performs slightly better.

Using **np.logspace(-4, 4, 50)** generates 50 values between 10^{-4} and 10^{4}. This covers a broad range, from very little regularisation to very strong regularisation.

Grid search – Gradient Boosting Regression



```python
gbr = GradientBoostingRegressor()

param_grid = {
    'n_estimators': [100, 200, 300],  # Number of boosting stages
    'learning_rate': [0.01, 0.1, 0.2],  # Learning rate shrinks the contribution of each tree
    'max_depth': [3, 4, 5],  # Maximum depth of the individual regression estimators
    'min_samples_split': [2, 5, 10],  # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4],  # Minimum number of samples required to be at a leaf node
    'subsample': [0.8, 0.9, 1.0],  # Fraction of samples used for fitting the individual base learners
}

grid_search = GridSearchCV(estimator=gbr, param_grid=param_grid, cv=5, scoring='r2', n_jobs=-1)
grid_search.fit(x_train, y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best parameters found: ", best_params)
print("Best cross-validation score: ", best_score)

best_model = grid_search.best_estimator_
test_score = best_model.score(x_test, y_test)

print("Test set score: ", test_score)
```

```
Best parameters found:  {'learning_rate': 0.2, 'max_depth': 5, 'min_samples_leaf': 4, 'min_samples_split': 5, 'n_estimators': 300, 'subsample': 0.9}
Best cross-validation score:  0.7265687043797385
Test set score:  0.743244305894013
```

This takes approx. 15-30mins to run because of the complexity and the number of hyperparameter combinations.

From the result we can see that the R-squared value is better than the one we did before using **default hyperparameters**.

Data preprocessing

Data preprocessing ensures that our data is clean, formatted, and ready for modelling. Proper preprocessing can significantly improve the performance of our machine learning models.

1. Data collection
    - Gather data
    - Merge datasets
2. Data cleaning
    - Handle missing values
    - Handle duplicates
    - Handle outliers
3. Data transformation
    - Scaling/Normalisation
    - Encoding categorical variables
    - Feature engineering
4. Feature selection
    - Remove irrelevant features
    - Select important features
5. Handling imbalanced data
    - Resampling techniques
    - Class weight adjustment
6. Splitting the data
    - Train-test split
    - Cross-validation

Data preprocessing – EDA

Exploratory Data Analysis (EDA) is a crucial step in the data science process, typically performed before formal modelling. The goal of EDA is to understand the structure, patterns, and relationships in your data, identify any anomalies or outliers, and generate hypotheses for further analysis.

EDA is not a onetime thing, there are several stages in the modelling process where you need to gain insights by conduction EDA. Such as:

- Before preprocessing
- Before feature engineering
- Before modelling
- Before hypothesis testing
- Throughout the project
- When communicating results

Data preprocessing – EDA

Key techniques in EDA:

- **Descriptive statistics**: mean, median, mode, standard deviation, etc., provide a summary of our data.
- **Data visualisation**: histograms, boxplots, scatter plots, pair plots, etc., help visualise distributions, relationships, and potential outliers.
- **Correlation analysis**: correlation matrices and heatmaps show relationships between variables.
- **Dimension reduction**: techniques like PCA (Principal Component Analysis) can help visualise high-dimensional data.
- **Univariate and bivariate analysis**: explore distributions of individual features (univariate) and relationships between pairs of features (bivariate).

Data preprocessing – descriptive statistics

```python
housing = pd.read_csv('./melb_data.csv')
housing.describe()
```

```
              Rooms         Price      Distance      Postcode      Bedroom2      Bathroom           Car       Landsize  BuildingArea    YearBuilt     Lattitude
count  13580.000000  1.358000e+04  13580.000000  13580.000000  13580.000000  13580.000000  13518.000000   13580.000000   7130.000000  8205.000000  13580.000000
mean       2.937997  1.075684e+06     10.137776   3105.301915      2.914728      1.534242      1.610075     558.416127    151.967650  1964.684217    -37.809203
std        0.955748  6.393107e+05      5.868725     90.676964      0.965921      0.691712      0.962634    3990.669241    541.014538    37.273762      0.079260
min        1.000000  8.500000e+04      0.000000   3000.000000      0.000000      0.000000      0.000000       0.000000      0.000000  1196.000000    -38.182550
25%        2.000000  6.500000e+05      6.100000   3044.000000      2.000000      1.000000      1.000000     177.000000     93.000000  1940.000000    -37.856822
50%        3.000000  9.030000e+05      9.200000   3084.000000      3.000000      1.000000      2.000000     440.000000    126.000000  1970.000000    -37.802355
75%        3.000000  1.330000e+06     13.000000   3148.000000      3.000000      2.000000      2.000000     651.000000    174.000000  1999.000000    -37.756400
max       10.000000  9.000000e+06     48.100000   3977.000000     20.000000      8.000000     10.000000  433014.000000  44515.000000  2018.000000    -37.408530
```

Data preprocessing – check data types and missing values

Data types:

```
Suburb            object
Address           object
Rooms              int64
Type              object
Price            float64
Method            object
SellerG           object
Date              object
Distance         float64
Postcode         float64
Bedroom2         float64
Bathroom         float64
Car              float64
Landsize         float64
BuildingArea     float64
YearBuilt        float64
CouncilArea       object
Lattitude        float64
Longtitude       float64
Regionname        object
Propertycount    float64
dtype: object
```

Missing values:

```
Suburb              0
Address             0
Rooms               0
Type                0
Price               0
Method              0
SellerG             0
Date                0
Distance            0
Postcode            0
Bedroom2            0
Bathroom            0
Car                62
Landsize            0
BuildingArea     6450
YearBuilt        5375
CouncilArea      1369
Lattitude           0
Longtitude          0
Regionname          0
Propertycount       0
dtype: int64
```

Data preprocessing – Histograms

Histograms are typically used for visualising the distribution of **continuous numerical** data.

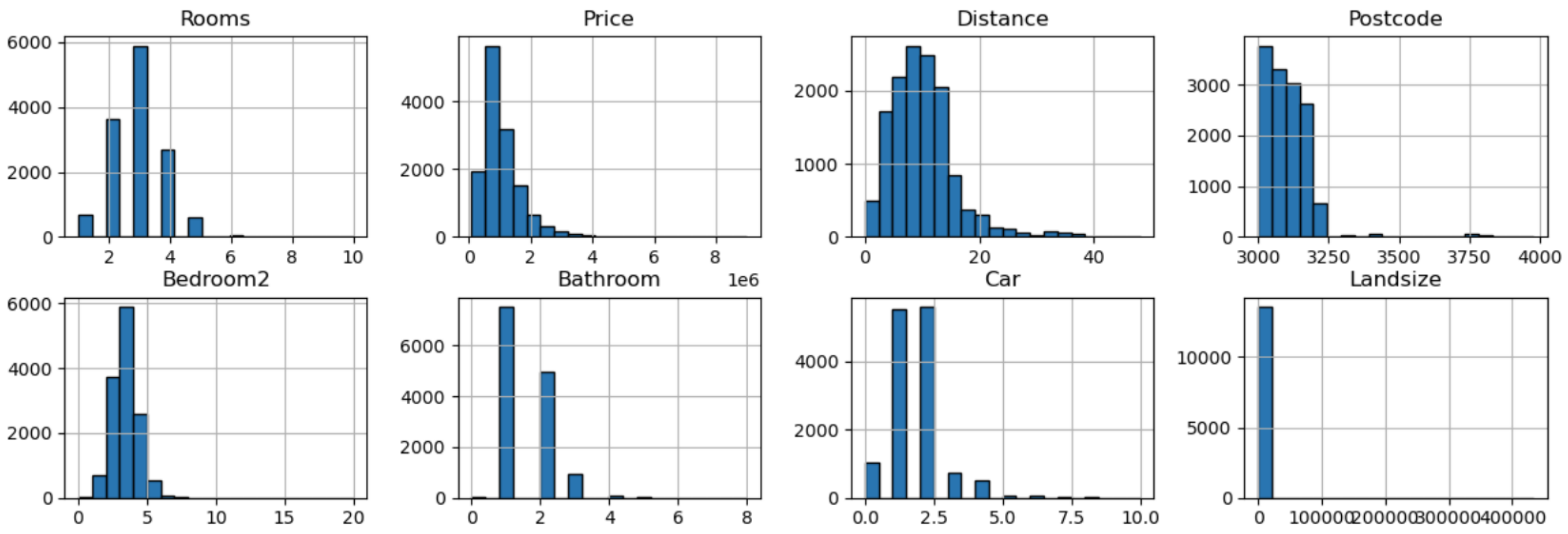
For **integer** values, we can use histogram or bar plot. When we have a large range of values or the values are numerous and span a board range, we can use histogram.

For columns with integers for a **small range of values**, we can use bar plot. Or use histogram but adjust the bins.

Here, we will use the pandas dataframe built-in function to plot histograms. It automatically plots histograms for numeric columns.

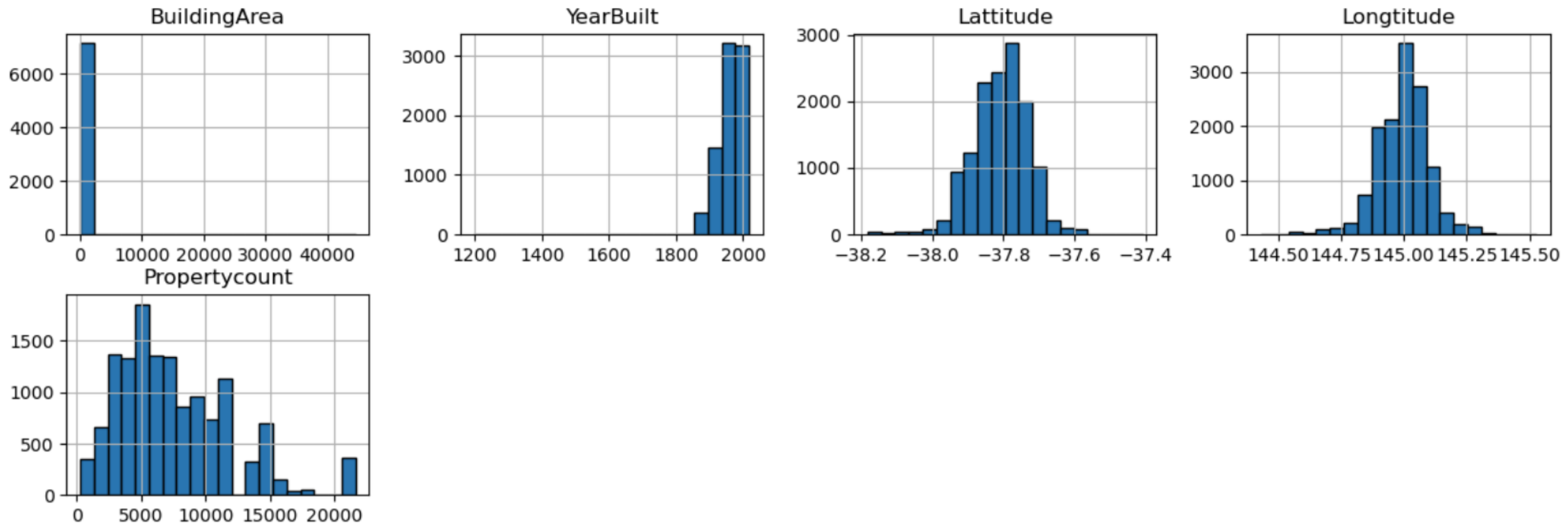
```
housing.hist(bins=20, edgecolor='black', figsize=(15, 10))
plt.show()
```

Data preprocessing – Histograms



Mostly normal distribution with right skew.
Landsize and Bedroom2 has extreme outliers.

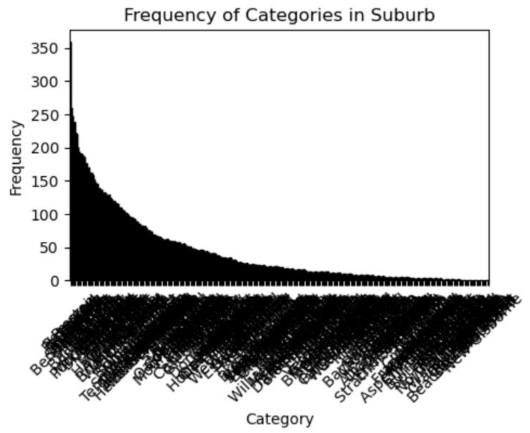
Data preprocessing – Histograms

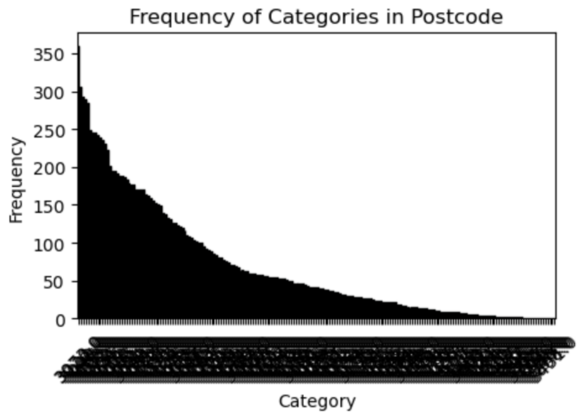


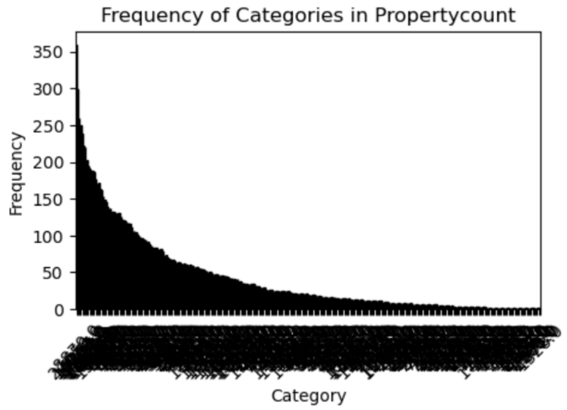
BuildingArea and YearBuilt has extreme outliers.
Propertycount might be bimodal distribution.
We can also change the bin width to capture the right level of detail.

Data preprocessing – Bar plot

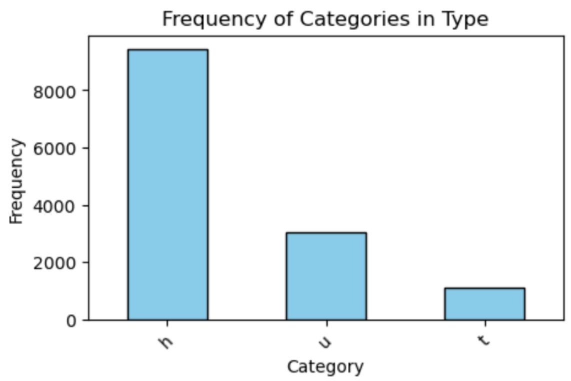
```python
cat_cols = ['Suburb', 'Postcode', 'Propertycount', 'Type', 'Method', 'SellerG', 'CouncilArea', 'Regionname']

for column in cat_cols:
    plt.figure(figsize=(5, 3))
    housing[column].value_counts().plot(kind='bar', color='skyblue', edgecolor='black')
    plt.title(f'Frequency of Categories in {column}')
    plt.xlabel('Category')
    plt.ylabel('Frequency')
    plt.xticks(rotation=45)
    plt.show()
```

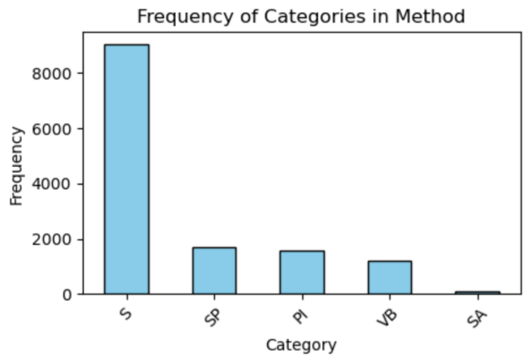


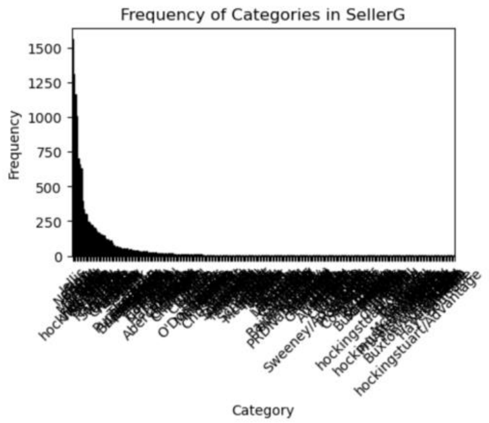


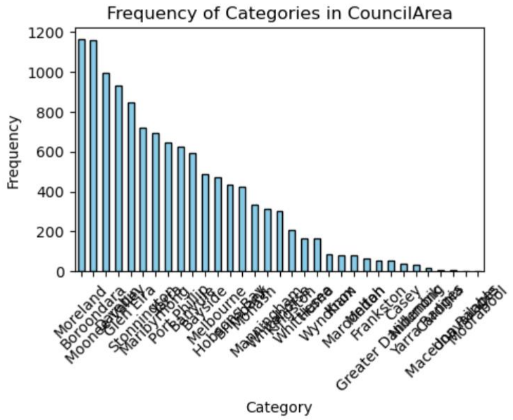


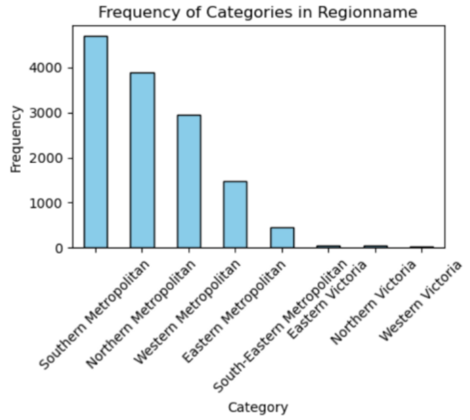
Data preprocessing – Bar plot











Data preprocessing – Bar plot

From the result we can see that some features have **too many categories with low observations**. It can lead to issues in machine learning model, such as **overfitting** and **increased computational complexity**.

Handling such features requires a balance between reducing dimensionality and preserving useful information. Here are some strategies:

- Grouping rare categories
- Feature engineering: create higher level categories, target encoding
- Frequency or count encoding
- Dimensionality reduction techniques
- One-hot encoding with feature selection
- Use of tree-based models
- Remove insignificant categories

Data preprocessing – Pair plot

A **pair plot** (also known as a scatterplot matrix) is a powerful visualisation tool in EDA. It allows us to visualise the pairwise relationships between multiple features in a dataset.

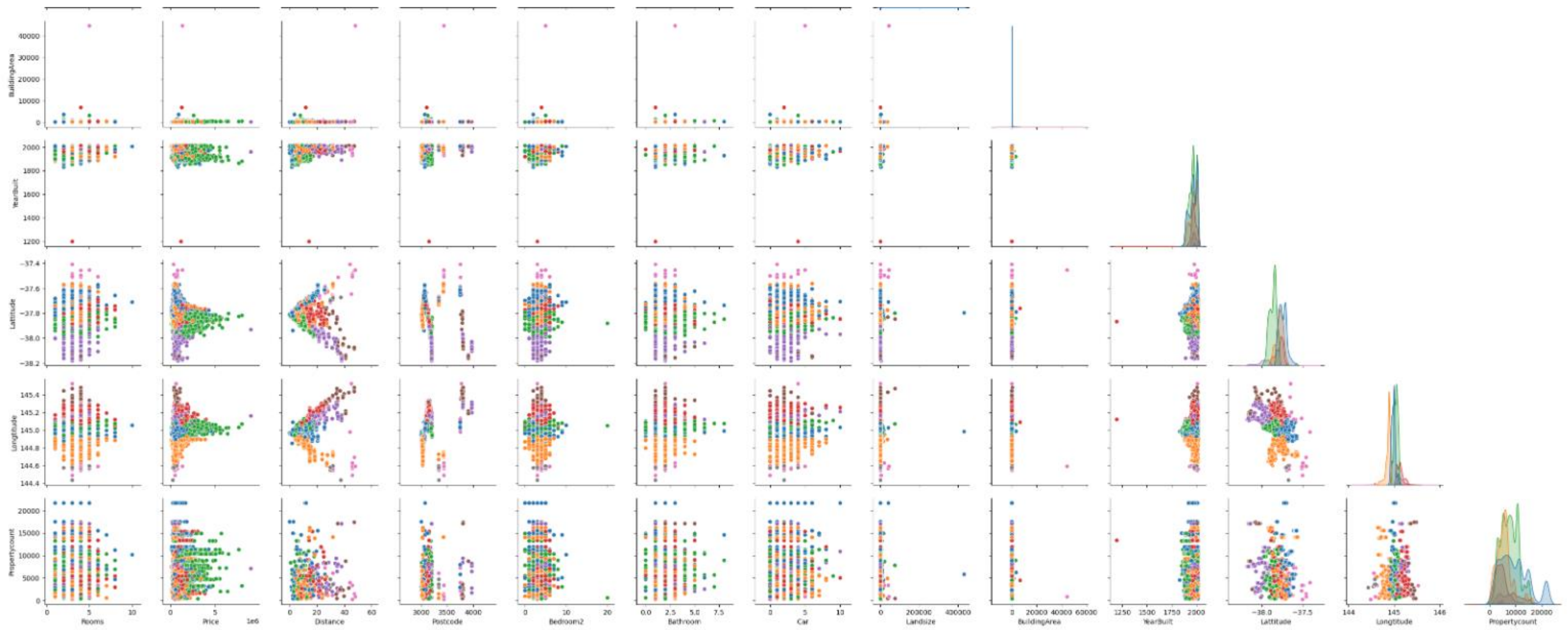
It's especially useful when we have **a mix of continuous and categorical variables** and want to explore the potential relationships and distributions across those features.

However, pair plot is best suited for datasets with a manageable number of features and observations, as too many can lead to clutter and overplotting.

```python
import seaborn as sns
import matplotlib.pyplot as plt

sns.pairplot(housing, corner=True, hue='Regionname')
plt.show()
```

Data preprocessing – Pair plot

By specifying hue='Regionname', we can add colours to the dots for different regions.

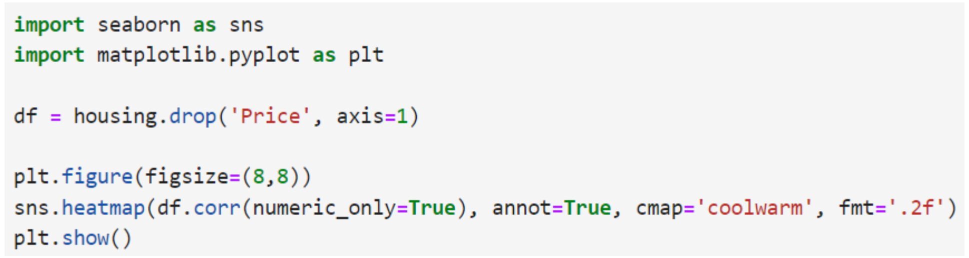


Data preprocessing – Correlation analysis

Correlation analysis is a statistical technique used to measure and analyse the strength and direction of the relationship **between two quantitative variables**.

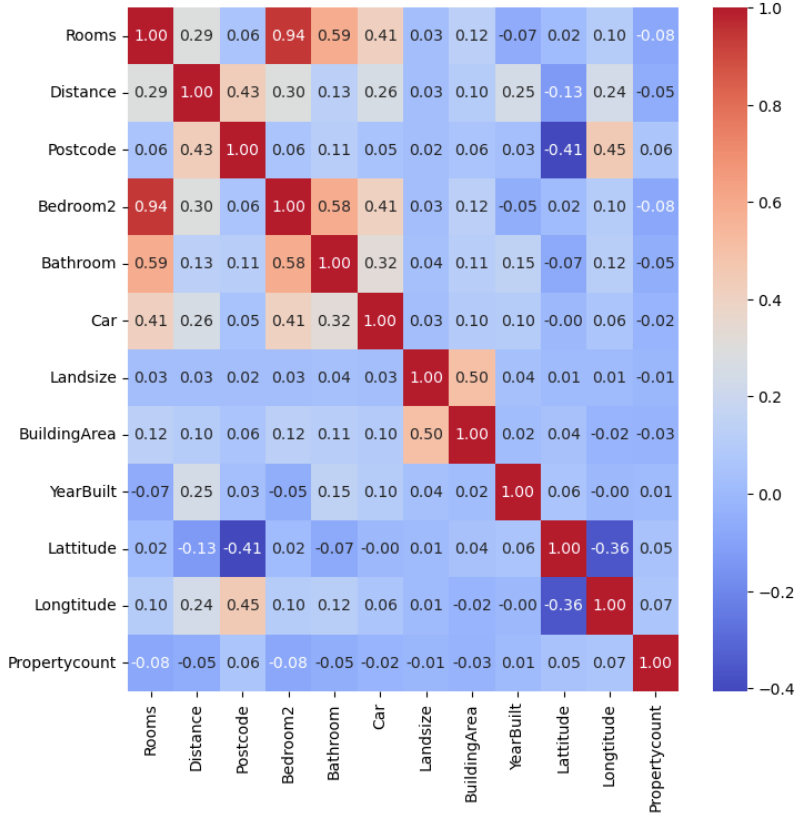
- **Strong correlations**: correlation coefficient (r) is close to -1 or +1.
- **Weak or no correlations**: r is close to 0.
- **Moderate correlation**: r is between +0.3 and +0.7 or -0.3 and -0.7

Variables with high correlation might be **redundant**, so we might choose to drop or combine them.



```python
import seaborn as sns
import matplotlib.pyplot as plt

df = housing.drop('Price', axis=1)

plt.figure(figsize=(8,8))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm', fmt='.2f')
plt.show()
```

Data preprocessing – Correlation analysis



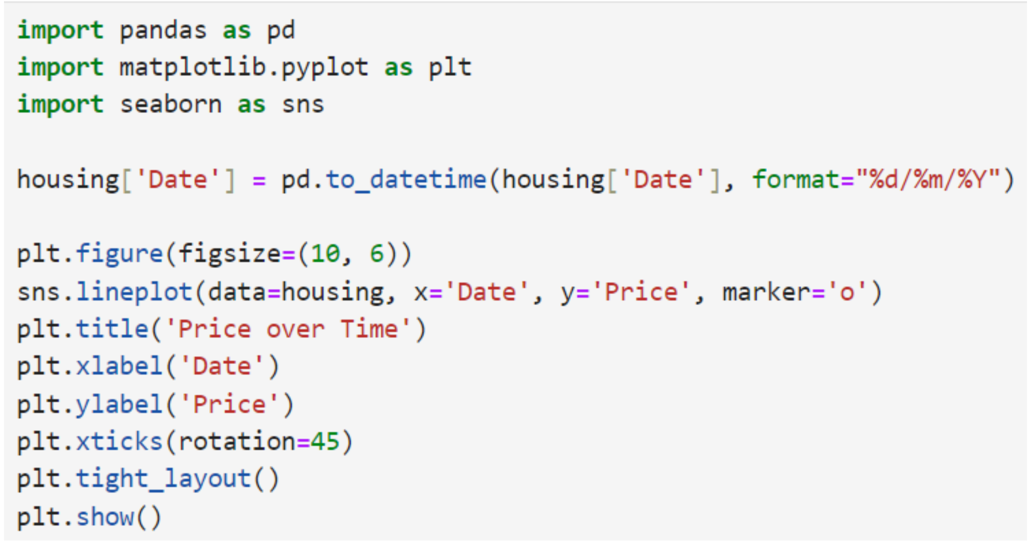
	Rooms	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt	Lattitude	Longtitude	Propertycount
Rooms	1.00	0.29	0.06	0.94	0.59	0.41	0.03	0.12	-0.07	0.02	0.10	-0.08
Distance	0.29	1.00	0.43	0.30	0.13	0.26	0.03	0.10	0.25	-0.13	0.24	-0.05
Postcode	0.06	0.43	1.00	0.06	0.11	0.05	0.02	0.06	0.03	-0.41	0.45	0.06
Bedroom2	0.94	0.30	0.06	1.00	0.58	0.41	0.03	0.12	-0.05	0.02	0.10	-0.08
Bathroom	0.59	0.13	0.11	0.58	1.00	0.32	0.04	0.11	0.15	-0.07	0.12	-0.05
Car	0.41	0.26	0.05	0.41	0.32	1.00	0.03	0.10	0.10	-0.00	0.06	-0.02
Landsize	0.03	0.03	0.02	0.03	0.04	0.03	1.00	0.50	0.04	0.01	0.01	-0.01
BuildingArea	0.12	0.10	0.06	0.12	0.11	0.10	0.50	1.00	0.02	0.04	-0.02	-0.03
YearBuilt	-0.07	0.25	0.03	-0.05	0.15	0.10	0.04	0.02	1.00	0.06	-0.00	0.01
Lattitude	0.02	-0.13	-0.41	0.02	-0.07	-0.00	0.01	0.04	0.06	1.00	-0.36	0.05
Longtitude	0.10	0.24	0.45	0.10	0.12	0.06	0.01	-0.02	-0.00	-0.36	1.00	0.07
Propertycount	-0.08	-0.05	0.06	-0.08	-0.05	-0.02	-0.01	-0.03	0.01	0.05	0.07	1.00

‘Rooms’ and ‘Bedroom2’ are highly correlated.

Data preprocessing – Time series data

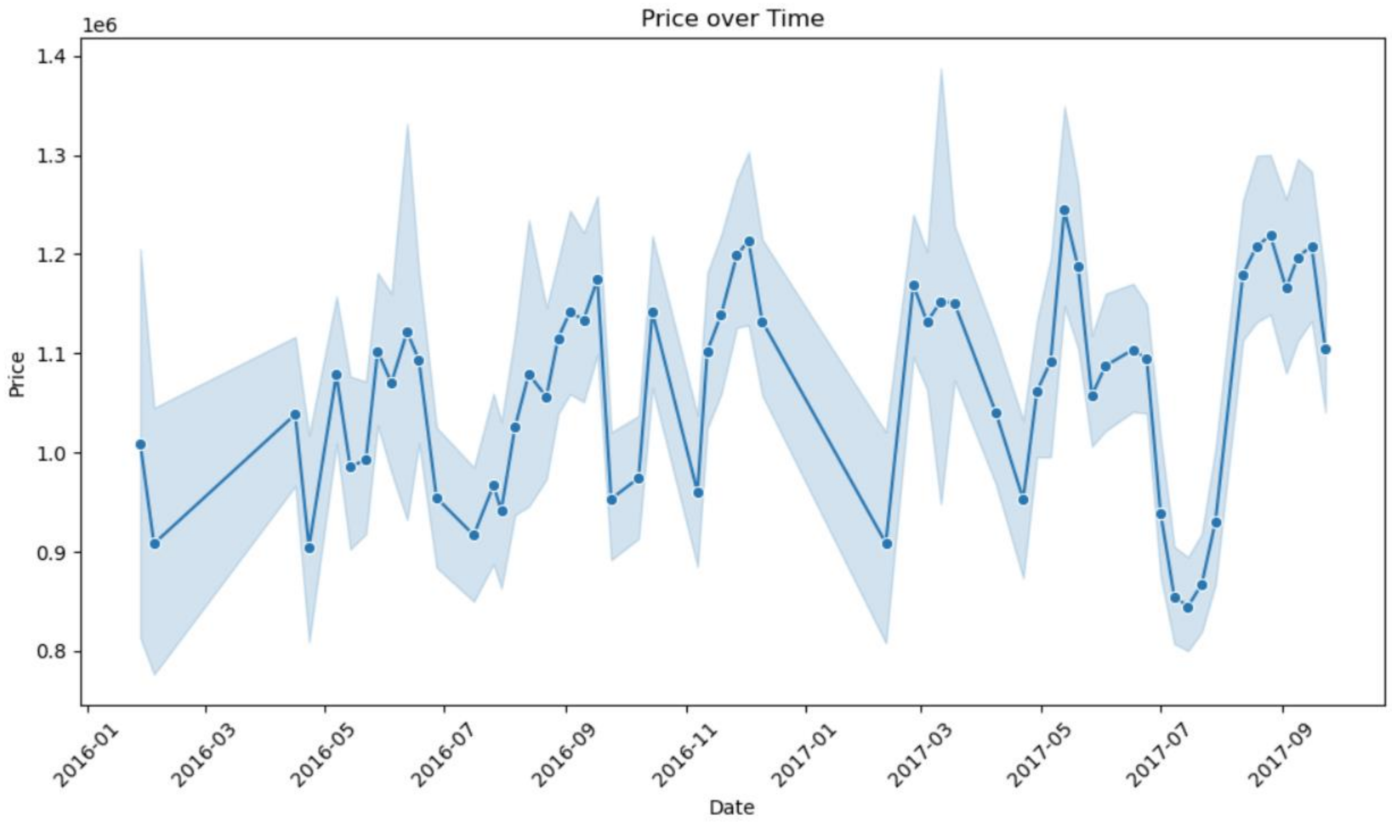
Our ‘Date’ variable is stored as object data type; to work with time series data, we can convert them to datetime format first.

Then plot them with our target variable ‘Price’.



```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

housing['Date'] = pd.to_datetime(housing['Date'], format="%d/%m/%Y")

plt.figure(figsize=(10, 6))
sns.lineplot(data=housing, x='Date', y='Price', marker='o')
plt.title('Price over Time')
plt.xlabel('Date')
plt.ylabel('Price')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Data preprocessing – Time series data



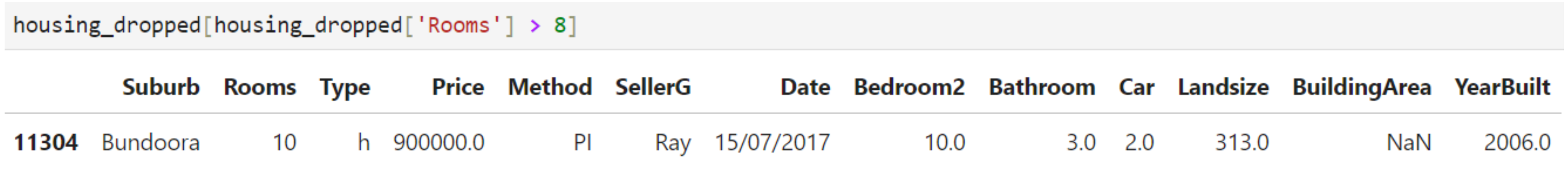
Data cleaning – remove errors

We have some extreme outliers for a few variables, such as ‘Landsize’ and ‘BuildingArea’, which has drastically impact on visualising our data. Making it difficult to see the distribution for these variables.

Let’s inspect these outliers and to decide if we can remove them.

Data cleaning – remove errors

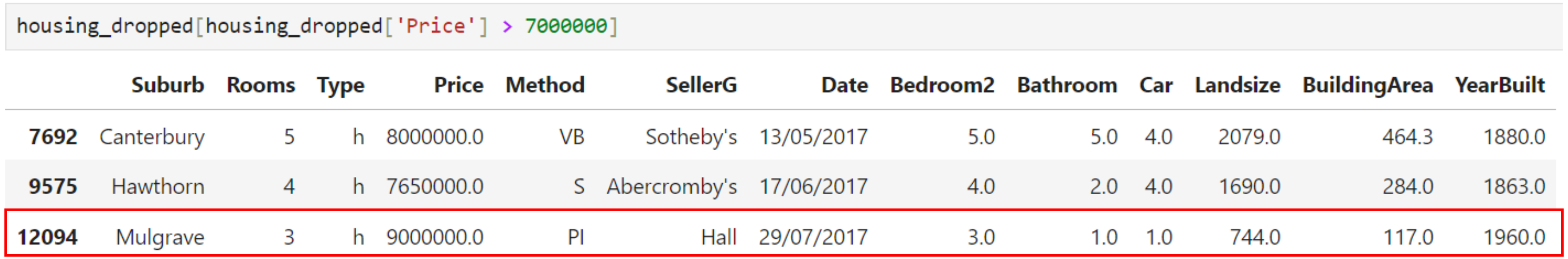
Rows where ‘Rooms’ > 8:



```
housing_dropped[housing_dropped['Rooms'] > 8]
```

	Suburb	Rooms	Type	Price	Method	SellerG	Date	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt
11304	Bundoora	10	h	900000.0	PI	Ray	15/07/2017	10.0	3.0	2.0	313.0	NaN	2006.0

Rows where ‘Price’ > 70000:

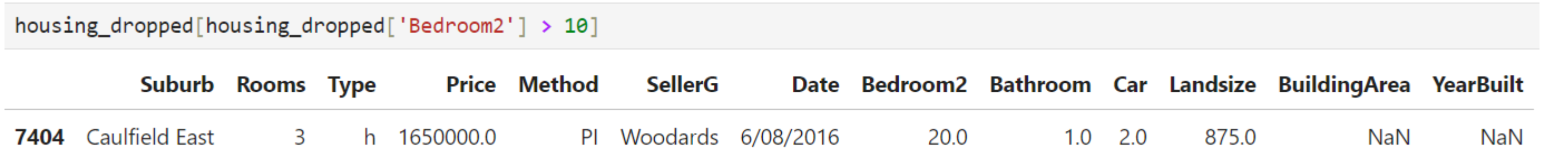


```
housing_dropped[housing_dropped['Price'] > 7000000]
```

	Suburb	Rooms	Type	Price	Method	SellerG	Date	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt
7692	Canterbury	5	h	8000000.0	VB	Sotheby's	13/05/2017	5.0	5.0	4.0	2079.0	464.3	1880.0
9575	Hawthorn	4	h	7650000.0	S	Abercromby's	17/06/2017	4.0	2.0	4.0	1690.0	284.0	1863.0
12094	Mulgrave	3	h	9000000.0	PI	Hall	29/07/2017	3.0	1.0	1.0	744.0	117.0	1960.0

Data cleaning – remove errors

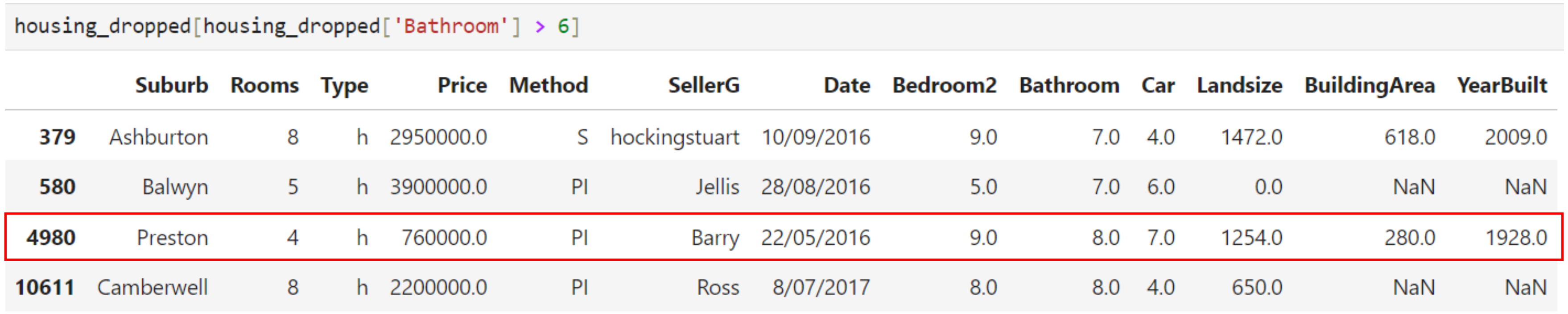
Rows where ‘Bedroom2’ > 10:



```
housing_dropped[housing_dropped['Bedroom2'] > 10]
```

	Suburb	Rooms	Type	Price	Method	SellerG	Date	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt
7404	Caulfield East	3	h	1650000.0	PI	Woodards	6/08/2016	20.0	1.0	2.0	875.0	NaN	NaN

Rows where ‘Bathroom’ > 6:

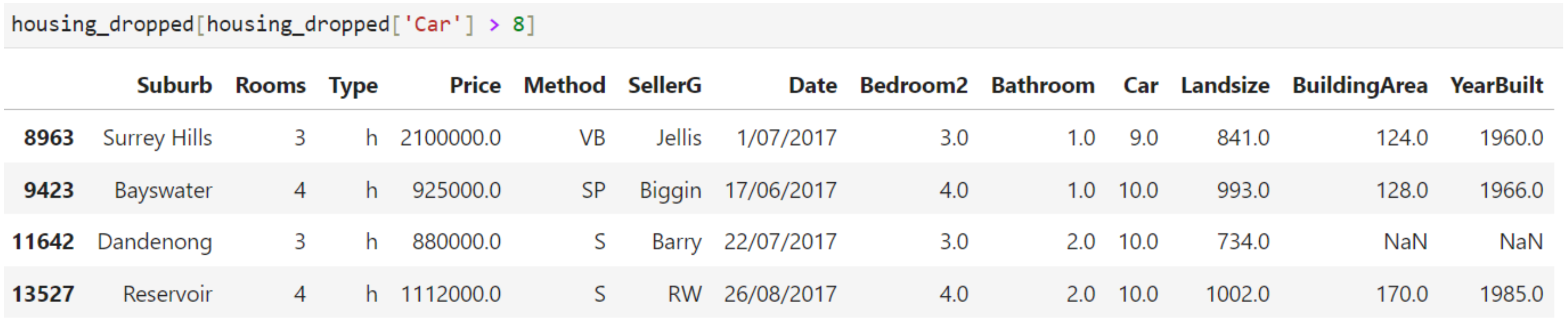


```
housing_dropped[housing_dropped['Bathroom'] > 6]
```

	Suburb	Rooms	Type	Price	Method	SellerG	Date	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt
379	Ashburton	8	h	2950000.0	S	hockingstuart	10/09/2016	9.0	7.0	4.0	1472.0	618.0	2009.0
580	Balwyn	5	h	3900000.0	PI	Jellis	28/08/2016	5.0	7.0	6.0	0.0	NaN	NaN
4980	Preston	4	h	760000.0	PI	Barry	22/05/2016	9.0	8.0	7.0	1254.0	280.0	1928.0
10611	Camberwell	8	h	2200000.0	PI	Ross	8/07/2017	8.0	8.0	4.0	650.0	NaN	NaN

Data cleaning – remove errors

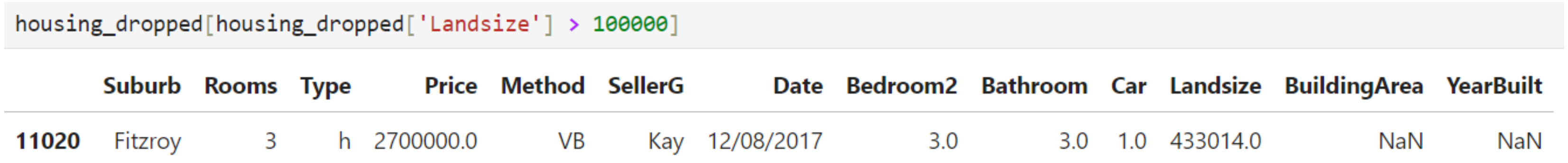
Rows where ‘Car’ > 8:



```
housing_dropped[housing_dropped['Car'] > 8]
```

	Suburb	Rooms	Type	Price	Method	SellerG	Date	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt
8963	Surrey Hills	3	h	2100000.0	VB	Jellis	1/07/2017	3.0	1.0	9.0	841.0	124.0	1960.0
9423	Bayswater	4	h	925000.0	SP	Biggin	17/06/2017	4.0	1.0	10.0	993.0	128.0	1966.0
11642	Dandenong	3	h	880000.0	S	Barry	22/07/2017	3.0	2.0	10.0	734.0	NaN	NaN
13527	Reservoir	4	h	1112000.0	S	RW	26/08/2017	4.0	2.0	10.0	1002.0	170.0	1985.0

Rows where ‘Landsize’ > 100000:

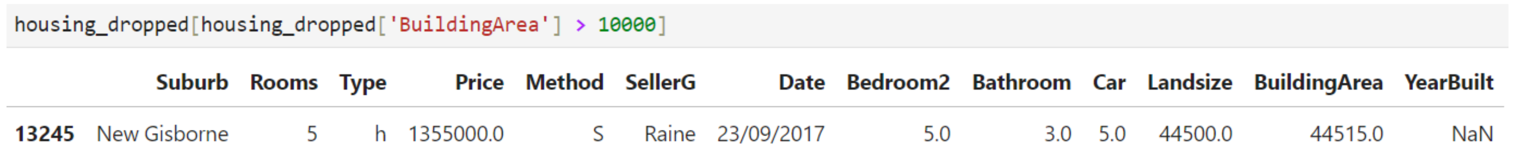


```
housing_dropped[housing_dropped['Landsize'] > 100000]
```

	Suburb	Rooms	Type	Price	Method	SellerG	Date	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt
11020	Fitzroy	3	h	2700000.0	VB	Kay	12/08/2017	3.0	3.0	1.0	433014.0	NaN	NaN

Data cleaning – remove errors

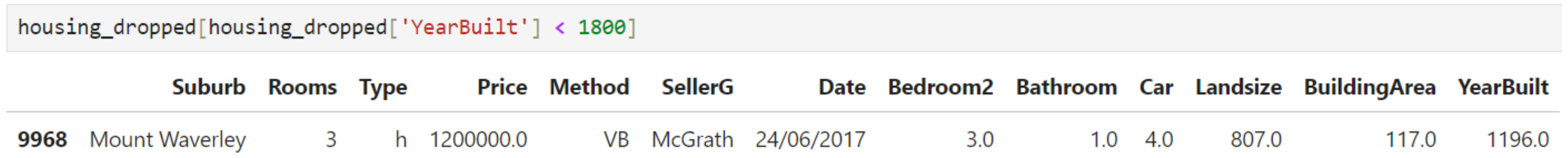
Rows where 'BuildingArea' > 10000:



```python
housing_dropped[housing_dropped['BuildingArea'] > 10000]
```

	Suburb	Rooms	Type	Price	Method	SellerG	Date	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt
13245	New Gisborne	5	h	1355000.0	S	Raine	23/09/2017	5.0	3.0	5.0	44500.0	44515.0	NaN

Rows where 'YearBuilt' < 1800:



```python
housing_dropped[housing_dropped['YearBuilt'] < 1800]
```

	Suburb	Rooms	Type	Price	Method	SellerG	Date	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt
9968	Mount Waverley	3	h	1200000.0	VB	McGrath	24/06/2017	3.0	1.0	4.0	807.0	117.0	1196.0

After investigation, we can see there are some outliers are clearly errors. We can remove them, so our data won't be influenced.

Exercise:

Repeat what I did in the previous slides.

Investigate the relationships between the target and our independent variables, by creating scatter plot with fitted lines (seaborn) for continuous variables and creating box plot (seaborn) for categorical variables.

Use online resources, to find if there are better ways to handle missing values rather than impute the median.

Use online resources to learn how to handle outliers.

THANK YOU

Contact Us

Biological Data Science Institute
ANU College of Science

RN Robertson Building
46 Sullivans Creek Rd
Acton ACT 2601

jiajia.li1@anu.edu.au

Australian
National
University