65070503464-midterm-data-model

February 19, 2024

1 Ratchanon Tarawan 65070503464

[119]:
```python
import pandas as pd
import matplotlib.pyplot as plt
```

1. Display information of the data: size, shape, number of dimensions, and overview information.

[120]:
```python
df = data = pd.read_csv('melb_data.csv')
```

[121]:
```python
df.size
```

[121]:
```
285180
```

[122]:
```python
df.shape
```

[122]:
```
(13580, 21)
```

[123]:
```python
df.ndim
```

[123]:
```
2
```

[124]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13580 entries, 0 to 13579
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Suburb         13580 non-null  object
 1   Address        13580 non-null  object
 2   Rooms          13580 non-null  int64
 3   Type           13580 non-null  object
 4   Price          13580 non-null  float64
 5   Method         13580 non-null  object
 6   SellerG        13580 non-null  object
 7   Date           13580 non-null  object
 8   Distance       13580 non-null  float64
 9   Postcode       13580 non-null  float64
```

```
 10  Bedroom2       13580 non-null  float64
 11  Bathroom       13580 non-null  float64
 12  Car            13518 non-null  float64
 13  Landsize       13580 non-null  float64
 14  BuildingArea   7130 non-null   float64
 15  YearBuilt      8205 non-null   float64
 16  CouncilArea    12211 non-null  object
 17  Lattitude      13580 non-null  float64
 18  Longtitude     13580 non-null  float64
 19  Regionname     13580 non-null  object
 20  Propertycount  13580 non-null  float64
dtypes: float64(12), int64(1), object(8)
memory usage: 2.2+ MB
```

2. Display statistics (min, max, average, S.D.) of the following:

2.1. [3 points] All attributes

[125]:
```
df.describe()
```

[125]:
```
              Rooms         Price      Distance      Postcode      Bedroom2  \
count  13580.000000  1.358000e+04  13580.000000  13580.000000  13580.000000
mean       2.937997  1.075684e+06     10.137776   3105.301915      2.914728
std        0.955748  6.393107e+05      5.868725     90.676964      0.965921
min        1.000000  8.500000e+04      0.000000   3000.000000      0.000000
25%        2.000000  6.500000e+05      6.100000   3044.000000      2.000000
50%        3.000000  9.030000e+05      9.200000   3084.000000      3.000000
75%        3.000000  1.330000e+06     13.000000   3148.000000      3.000000
max       10.000000  9.000000e+06     48.100000   3977.000000     20.000000

           Bathroom           Car       Landsize  BuildingArea    YearBuilt  \
count  13580.000000  13518.000000   13580.000000   7130.000000  8205.000000
mean       1.534242      1.610075     558.416127    151.967650  1964.684217
std        0.691712      0.962634    3990.669241    541.014538    37.273762
min        0.000000      0.000000       0.000000      0.000000  1196.000000
25%        1.000000      1.000000     177.000000     93.000000  1940.000000
50%        1.000000      2.000000     440.000000    126.000000  1970.000000
75%        2.000000      2.000000     651.000000    174.000000  1999.000000
max        8.000000     10.000000  433014.000000  44515.000000  2018.000000

          Lattitude    Longtitude  Propertycount
count  13580.000000  13580.000000   13580.000000
mean     -37.809203    144.995216    7454.417378
std        0.079260      0.103916    4378.581772
min      -38.182550    144.431810     249.000000
25%      -37.856822    144.929600    4380.000000
50%      -37.802355    145.000100    6555.000000
75%      -37.756400    145.058305   10331.000000
```

```
max      -37.408530    145.526350    21650.000000
```

2.2. [6 points] Selected attributes: Price, Landsize, Propertycount

[126]:
```python
df[['Price', 'Landsize', 'Propertycount']].describe()
```

[126]:

	Price	Landsize	Propertycount
count	1.358000e+04	13580.000000	13580.000000
mean	1.075684e+06	558.416127	7454.417378
std	6.393107e+05	3990.669241	4378.581772
min	8.500000e+04	0.000000	249.000000
25%	6.500000e+05	177.000000	4380.000000
50%	9.030000e+05	440.000000	6555.000000
75%	1.330000e+06	651.000000	10331.000000
max	9.000000e+06	433014.000000	21650.000000

2.3. [6 points] Selected attributes with a specific condition: Landsize < 500, Bedroom2 =2 and Bathroom =1 and Car =1

[127]:
```python
filtered_df = df[ (df['Landsize'] < 500) & (df['Bedroom2'] == 2) &␣
  ↪(df['Bathroom'] == 1) & (df['Car'] == 1)]
filtered_df.describe()
```

[127]:

	Rooms	Price	Distance	Postcode	Bedroom2	\
count	1749.000000	1.749000e+03	1749.000000	1749.000000	1749.0	
mean	2.026872	6.693860e+05	8.018754	3098.251001	2.0	
std	0.202582	2.784590e+05	4.310617	69.396890	0.0	
min	1.000000	1.450000e+05	0.000000	3000.000000	2.0	
25%	2.000000	4.825000e+05	5.100000	3046.000000	2.0	
50%	2.000000	6.100000e+05	7.700000	3078.000000	2.0	
75%	2.000000	7.800000e+05	11.200000	3146.000000	2.0	
max	4.000000	2.905000e+06	41.000000	3910.000000	2.0	

	Bathroom	Car	Landsize	BuildingArea	YearBuilt	Lattitude	\
count	1749.0	1749.0	1749.000000	956.000000	1176.000000	1749.000000	
mean	1.0	1.0	106.315609	84.622228	1969.389456	-37.809896	
std	0.0	0.0	122.028427	44.034837	32.794572	0.063897	
min	1.0	1.0	0.000000	0.000000	1880.000000	-38.164390	
25%	1.0	1.0	0.000000	69.000000	1960.000000	-37.852580	
50%	1.0	1.0	85.000000	80.000000	1970.000000	-37.806350	
75%	1.0	1.0	177.000000	94.000000	1996.000000	-37.764300	
max	1.0	1.0	499.000000	1143.000000	2016.000000	-37.570630	

	Longtitude	Propertycount
count	1749.000000	1749.000000
mean	144.988234	7874.824471
std	0.072346	4619.813212
min	144.571590	438.000000

```
25%      144.940000    4675.000000
50%      144.993300    6938.000000
75%      145.034800   10412.000000
max      145.292840   21650.000000
```

3. Inspect if there are any missing values; and If there are, perform the following cases:

3.1 [5 points] Use the original data, remove rows that contain missing values. Display data shape and overview information after the removal.

[135]:
```python
missing_values = df.isnull().sum()
remove_df = df.dropna()
# remove_df.isnull().sum()
print('Shape ', remove_df.shape)
print(remove_df.info())
```

```
Shape  (6196, 21)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6196 entries, 1 to 12212
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Suburb         6196 non-null   object
 1   Address        6196 non-null   object
 2   Rooms          6196 non-null   int64
 3   Type           6196 non-null   object
 4   Price          6196 non-null   float64
 5   Method         6196 non-null   object
 6   SellerG        6196 non-null   object
 7   Date           6196 non-null   object
 8   Distance       6196 non-null   float64
 9   Postcode       6196 non-null   float64
 10  Bedroom2       6196 non-null   float64
 11  Bathroom       6196 non-null   float64
 12  Car            6196 non-null   float64
 13  Landsize       6196 non-null   float64
 14  BuildingArea   6196 non-null   float64
 15  YearBuilt      6196 non-null   float64
 16  CouncilArea    6196 non-null   object
 17  Lattitude      6196 non-null   float64
 18  Longtitude     6196 non-null   float64
 19  Regionname     6196 non-null   object
 20  Propertycount  6196 non-null   float64
dtypes: float64(12), int64(1), object(8)
memory usage: 1.0+ MB
None
```

3.2 [12 points] Use the original data, replace missing values with zeros. For those columns that contain missing values, compare the average value of the before versus after replacement. Based on this result, briefly discuss if this method should be used, and why.

[129]:
```
# missing = Car, BuildingArea, YearBuilt, CouncilArea

df_filled = df.copy()
df_filled[['Car', 'BuildingArea', 'YearBuilt', 'CouncilArea']] =␣
↪df_filled[['Car', 'BuildingArea', 'YearBuilt', 'CouncilArea']].fillna(0)

avg_before = df[['Car', 'BuildingArea', 'YearBuilt', 'CouncilArea']].mean()
avg_after = df_filled[['Car', 'BuildingArea', 'YearBuilt', 'CouncilArea']].
↪mean()

pd.DataFrame({'Average Before': avg_before, 'Average After': avg_after})
```

```
<ipython-input-129-bd4d19a3dd49>:6: FutureWarning: The default value of
numeric_only in DataFrame.mean is deprecated. In a future version, it will
default to False. In addition, specifying 'numeric_only=None' is deprecated.
Select only valid columns or specify the value of numeric_only to silence this
warning.
  avg_before = df[['Car', 'BuildingArea', 'YearBuilt', 'CouncilArea']].mean()
<ipython-input-129-bd4d19a3dd49>:7: FutureWarning: The default value of
numeric_only in DataFrame.mean is deprecated. In a future version, it will
default to False. In addition, specifying 'numeric_only=None' is deprecated.
Select only valid columns or specify the value of numeric_only to silence this
warning.
  avg_after = df_filled[['Car', 'BuildingArea', 'YearBuilt',
'CouncilArea']].mean()
```

[129]:

	Average Before	Average After
Car	1.610075	1.602725
BuildingArea	151.967650	79.788611
YearBuilt	1964.684217	1187.056996

By replacing missing values with zeros, the mean of the column is likely to be significantly affected, especially if there are many missing values. This can result in misinterpretation of the data. So, it should not be this method used.

3.3 [12 points] Use the original data, apply data imputation for numeric columns (use average), and remove rows that contain missing values for non-numeric column(s). Compare the average value of the before versus after replacement for those imputed numeric columns. Based on this result, briefly discuss if this method should be used, and why.

[144]:
```
numeric_cols = df.select_dtypes(include=['number']).columns
df_imputed = df.copy()
df_imputed[numeric_cols] = df_imputed[numeric_cols].
↪fillna(df_imputed[numeric_cols].mean())

non_numeric_cols = df.select_dtypes(exclude=['number']).columns
```

```
df_cleaned = df_imputed.dropna(subset=non_numeric_cols)

avg_before = df[numeric_cols].mean()
avg_after = df_imputed[numeric_cols].mean()

pd.DataFrame({'Average Before': avg_before, 'Average After': avg_after})
```

[144]:

	Average Before	Average After
Rooms	2.937997e+00	2.937997e+00
Price	1.075684e+06	1.075684e+06
Distance	1.013778e+01	1.013778e+01
Postcode	3.105302e+03	3.105302e+03
Bedroom2	2.914728e+00	2.914728e+00
Bathroom	1.534242e+00	1.534242e+00
Car	1.610075e+00	1.610075e+00
Landsize	5.584161e+02	5.584161e+02
BuildingArea	1.519676e+02	1.519676e+02
YearBuilt	1.964684e+03	1.964684e+03
Lattitude	-3.780920e+01	-3.780920e+01
Longtitude	1.449952e+02	1.449952e+02
Propertycount	7.454417e+03	7.454417e+03

By using the average for imputation and removing rows with missing values for non-numeric columns can be a reasonable approach. This method helps retain the overall distribution of the data while handling missing values appropriately. AVG before = AVG after

4.[3 points] Format the datetime of the attribute Date to YYYY/MM/DD

[145]:
```
df_cleaned['Date'] = pd.to_datetime(df_cleaned['Date'],  format='%d/%m/%Y')
df_cleaned['Date'] = df_cleaned['Date'].dt.strftime('%Y/%m/%d')
df_cleaned['Date']
```

```
<ipython-input-145-e37da810dab8>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['Date'] = pd.to_datetime(df_cleaned['Date'],  format='%d/%m/%Y')
<ipython-input-145-e37da810dab8>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['Date'] = df_cleaned['Date'].dt.strftime('%Y/%m/%d')
```

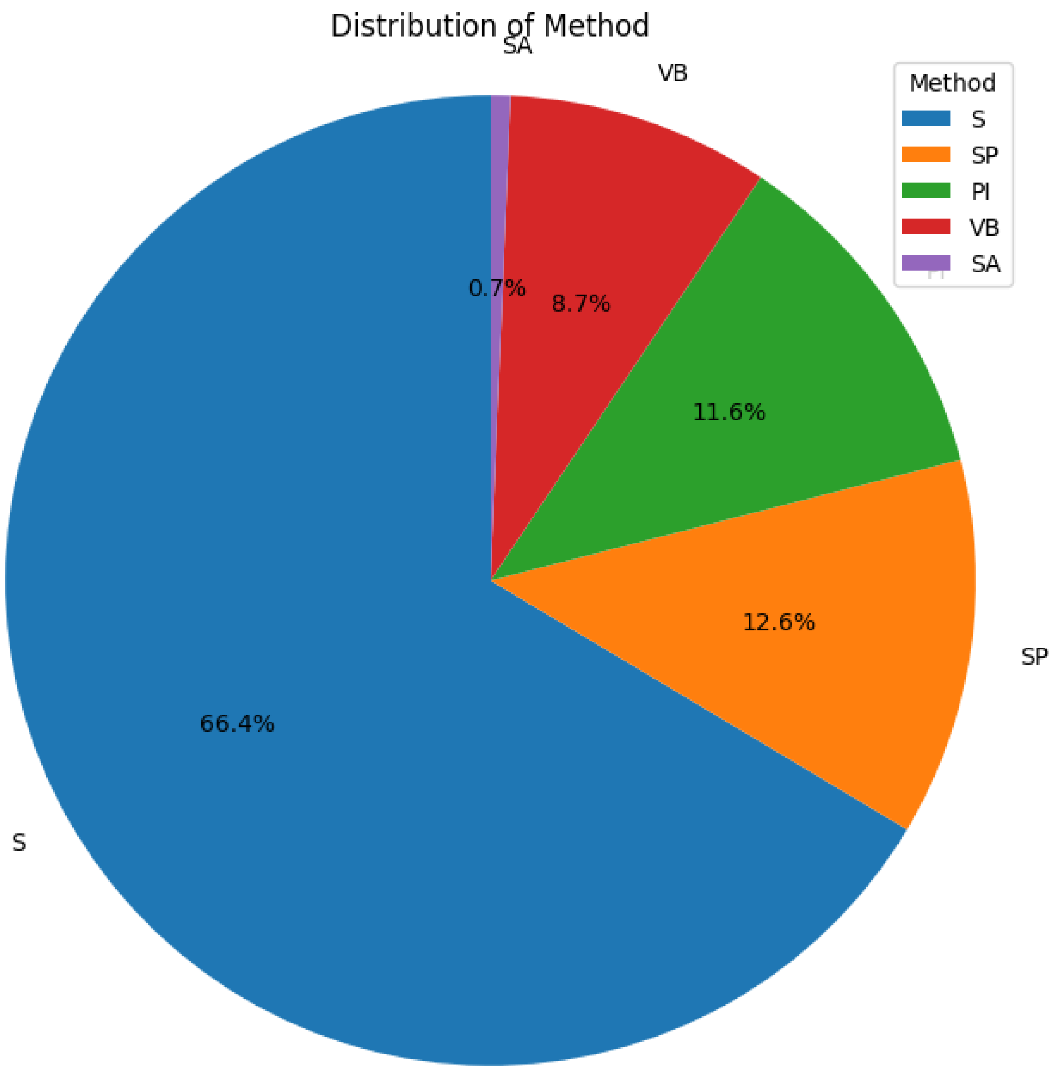
[145]:

```
0        2016/12/03
1        2016/02/04
2        2017/03/04
3        2017/03/04
4        2016/06/04
            …
12208    2017/07/29
12209    2017/07/29
12210    2017/07/29
12211    2017/07/29
12212    2017/07/29
Name: Date, Length: 12211, dtype: object
```

5. [15 points] Create a pie chart to demonstrate unique values of the attribute Method. In your visualization, also display chart title, percentage of distribution, and a legend.

[132]:

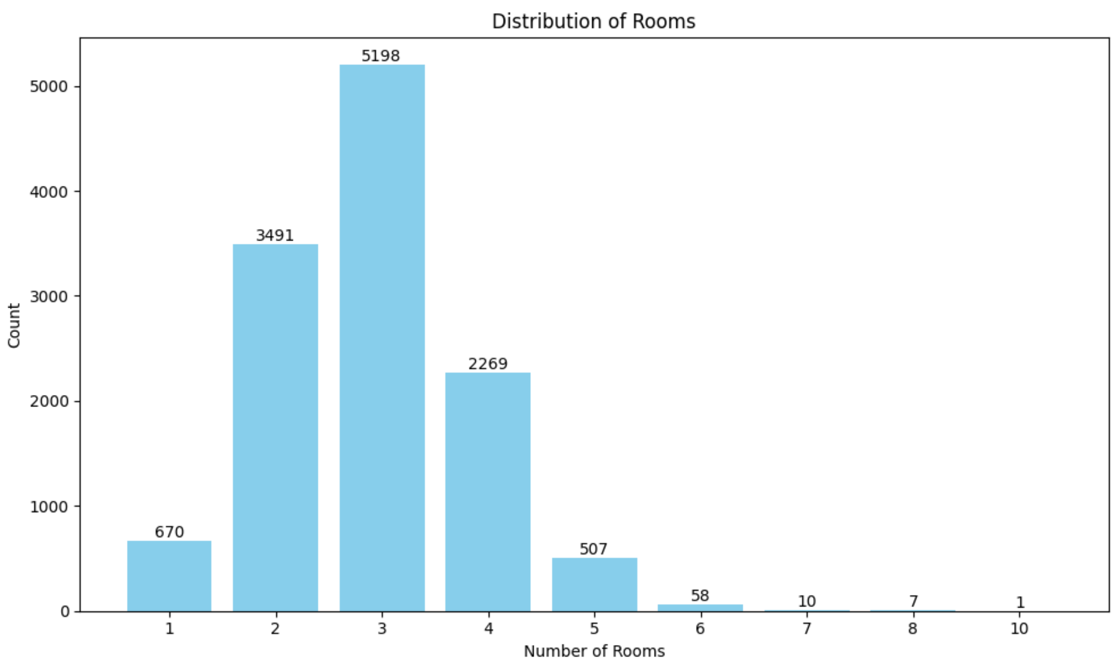
```
method_counts = df_cleaned['Method'].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(method_counts, labels=method_counts.index, autopct='%1.1f%%',␣
 ↪startangle=90)
plt.title('Distribution of Method')
plt.legend(title='Method', loc='upper right')
plt.axis('equal')
plt.show()
```

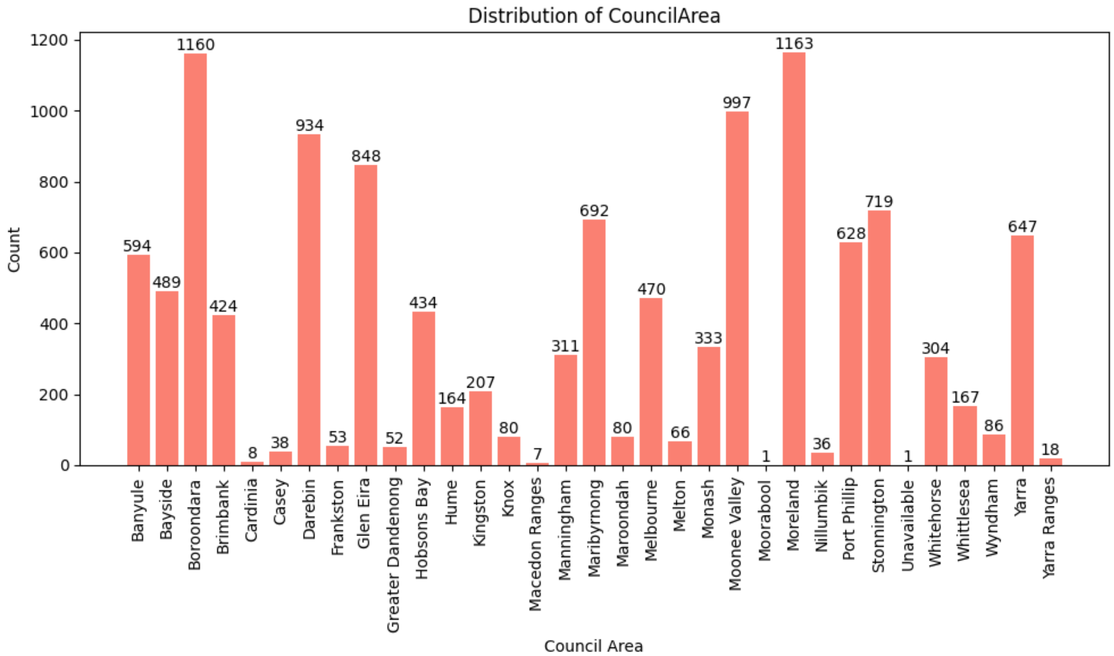


[15 points] Create two bar charts to present these attributes: Rooms and CouncilArea individually. For each visualization, also display the chart title, a legend both axes, and data labels. Make sure that data labels do not overlap one another.

[133]:
```
plt.figure(figsize=(10, 6))
room_counts = df_cleaned['Rooms'].value_counts().sort_index()
bars = plt.bar(room_counts.index.astype(str), room_counts.values,␣
  ↪color='skyblue')

plt.title('Distribution of Rooms')
plt.xlabel('Number of Rooms')
plt.ylabel('Count')
```

```python
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height + 0.1, height,␣
↪ha='center', va='bottom')

plt.tight_layout()
plt.show()

plt.figure(figsize=(10, 6))
council_counts = df_cleaned['CouncilArea'].value_counts().sort_index()
bars = plt.bar(council_counts.index, council_counts.values, color='salmon')

plt.title('Distribution of CouncilArea')
plt.xlabel('Council Area')
plt.ylabel('Count')
plt.xticks(rotation=90)

for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height + 0.1, height,␣
↪ha='center', va='bottom')

plt.tight_layout()
plt.show()
```





7. [15 points] Group the data by Regionname and Type, then display the sum of these attributes: Price, Bedroom2, Bathroom, Car, and Landsize. The expected output looks like the following snapshot.

[149]:
```
dfgroup =  df_cleaned.groupby(['Regionname', 'Type']).agg({
    'Price': 'sum',
    'Bedroom2': 'sum',
    'Bathroom': 'sum',
    'Car': 'sum',
    'Landsize': 'sum'
})
dfgroup
```

[149]:
```
                                     Price  Bedroom2  Bathroom     Car  \
Regionname                 Type
Eastern Metropolitan       h      1.133086e+09    3261.0    1636.0  1763.0
                           t      9.422715e+07     328.0     198.0   180.0
                           u      1.095653e+08     399.0     211.0   217.0
Eastern Victoria           h      2.889698e+07     142.0      78.0    88.0
                           u      1.384000e+06       8.0       3.0     4.0
Northern Metropolitan      h      2.528641e+09    7352.0    3439.0  3789.0
                           t      2.139878e+08     720.0     461.0   375.0
                           u      4.441437e+08    1529.0     946.0   878.0
Northern Victoria          h      1.455350e+07      91.0      46.0    47.0
South-Eastern Metropolitan h      2.575009e+08     963.0     467.0   573.0
                           t      1.596875e+07      49.0      30.0    30.0
```

	u	1.952750e+07	79.0	43.0	47.0
Southern Metropolitan	h	4.317675e+09	7990.0	4366.0	4307.0
	t	4.768569e+08	1175.0	774.0	693.0
	u	1.015503e+09	2955.0	1818.0	1724.0
Western Metropolitan	h	1.943246e+09	6391.0	3072.0	3634.0
	t	1.657553e+08	661.0	421.0	350.0
	u	1.986504e+08	850.0	482.0	473.0
Western Victoria	h	9.572750e+06	83.0	38.0	49.0

		Landsize
Regionname	Type	
Eastern Metropolitan	h	675951.0
	t	28051.0
	u	53761.0
Eastern Victoria	h	147098.0
	u	886.0
Northern Metropolitan	h	1537299.0
	t	89956.0
	u	385133.0
Northern Victoria	h	62746.0
South-Eastern Metropolitan	h	177101.0
	t	3742.0
	u	11829.0
Southern Metropolitan	h	1360338.0
	t	104129.0
	u	706538.0
Western Metropolitan	h	1011765.0
	t	56754.0
	u	226520.0
Western Victoria	h	15942.0