In [1]:

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


path = './melb_data.csv'
data = pd.read_csv(path)
data.describe()
```

Out[1]:

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000
mean	2.937997	1.075684e+06	10.137776	3105.301915	2.914728	1.534242
std	0.955748	6.393107e+05	5.868725	90.676964	0.965921	0.691712
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000
25%	2.000000	6.500000e+05	6.100000	3044.000000	2.000000	1.000000
50%	3.000000	9.030000e+05	9.200000	3084.000000	3.000000	1.000000
75%	3.000000	1.330000e+06	13.000000	3148.000000	3.000000	2.000000
max	10.000000	9.000000e+06	48.100000	3977.000000	20.000000	8.000000

In [2]:

```
print(data.columns)
```

```
Index(['Suburb', 'Address', 'Rooms', 'Type', 'Price', 'Method', 'SellerG',
       'Date', 'Distance', 'Postcode', 'Bedroom2', 'Bathroom', 'Car',
       'Landsize', 'BuildingArea', 'YearBuilt', 'CouncilArea', 'Lattitud
e',
       'Longtitude', 'Regionname', 'Propertycount'],
      dtype='object')
```

In [3]:

```
print(data.info())
```

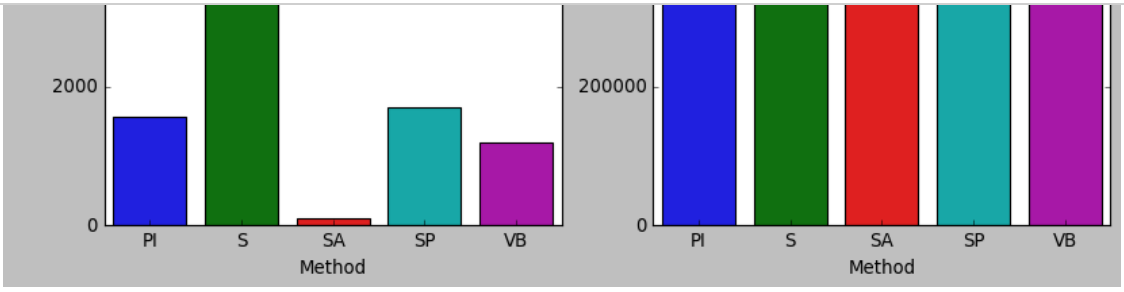
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13580 entries, 0 to 13579
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Suburb         13580 non-null  object 
 1   Address        13580 non-null  object 
 2   Rooms          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
None
```

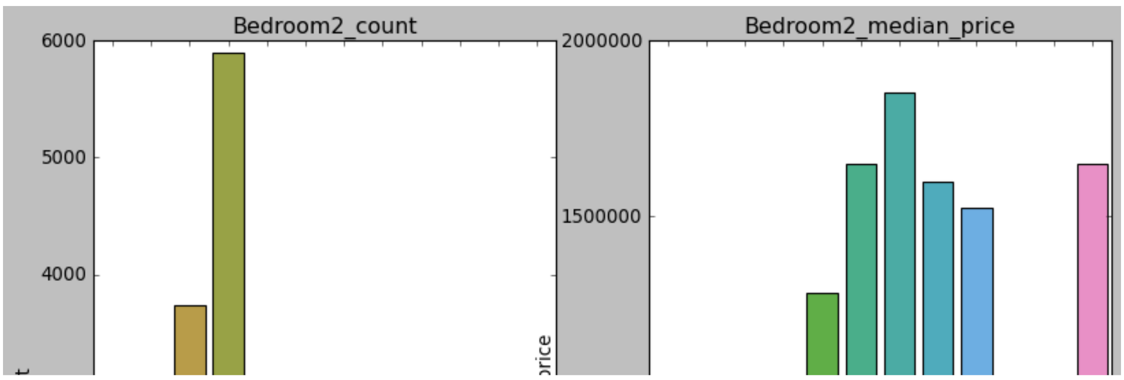
In [4]:

```
# 'Landsize','BuildingArea','YearBuilt'有遺失值，依變項為 'Price'
data[['day','month','year']] = data.Date.str.split('/',expand=True)
y = data.Price
dataa = data.drop(['Price','Lattitude','Longtitude'],axis=1)

def plot_data(dat,y_dat,colnam):
    plt.style.use('classic')
    dat = dat.join(y_dat)
    if len(set(dat[colnam])) >50:
        return print(colnam + 'over')
    # 處理遺失值
    if dat[[colnam]].isnull().any().values == True:
        dat = dat.dropna(axis=0,subset=colnam)

    # 處理浮點數
    if dat[[colnam]].dtypes.values == 'float64' :
        dat[[colnam]] = dat[[colnam]].round().astype(int)

    dat1 = dat.groupby(colnam)[y_dat.name].agg(['median','count'])
    fig,ax = plt.subplots(1,2,figsize=(12,8))

    ax1 = ax[0]
    ax2 = ax[1]
    sns.barplot(ax=ax1,x=dat1.index,y=dat1['count'])
    ax1.set_title(colnam +'_count')
    ax1.set_xlabel(colnam)
    ax1.set_ylabel('count')

    sns.barplot(ax=ax2,x=dat1.index,y=dat1['median'])
    ax2.set_title(colnam +'_median_price')
    ax2.set_xlabel(colnam)
    ax2.set_ylabel('median_price')
```

In [5]:

```
for col in dataa.columns:
    plot_data(dataa,y,col)
```



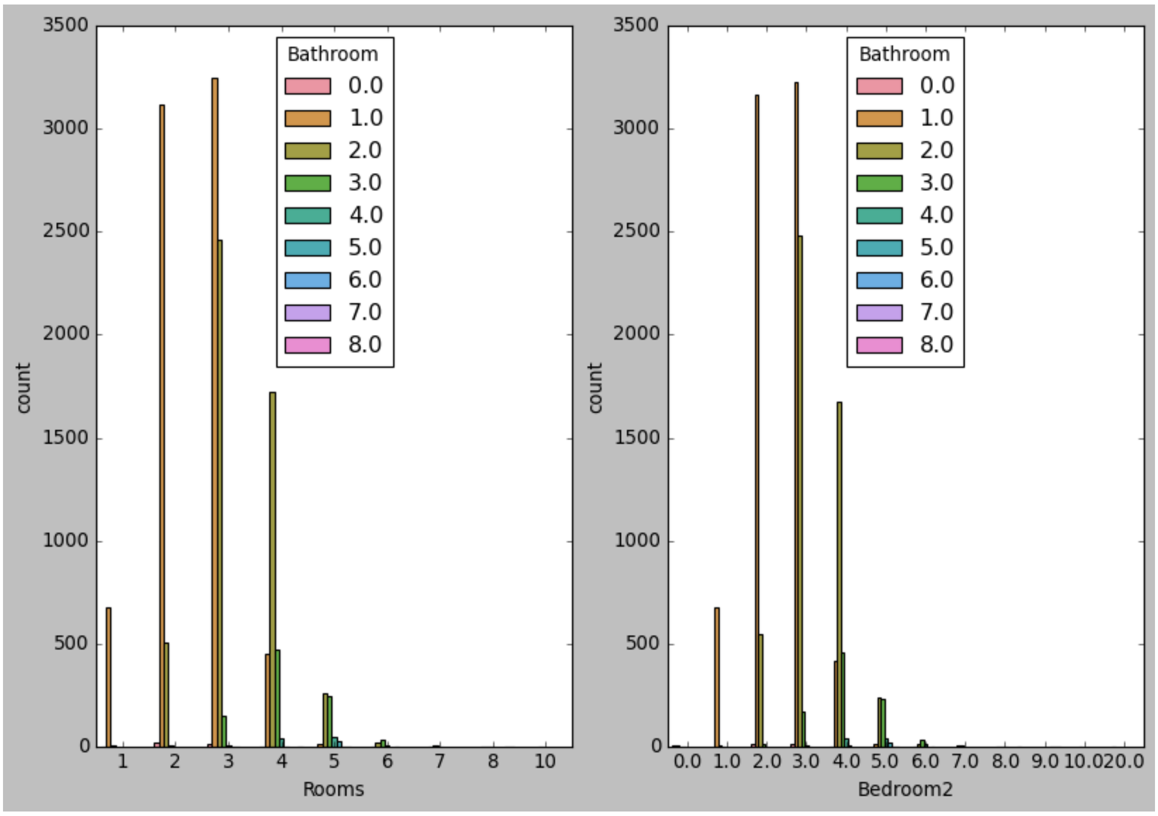


In [6]:

```
check_room1 = pd.crosstab(dataa.Rooms,dataa.Bedroom2,margins=True)
check_room2 = pd.crosstab(dataa.Rooms,dataa.Bathroom,margins=True)
check_room3 = pd.crosstab(dataa.Bedroom2,dataa.Bathroom,margins=True)

fig,ax=plt.subplots(1,2,figsize=(12,8))
sns.countplot(x='Rooms',hue='Bathroom',data=dataa,ax=ax[0])
sns.countplot(x='Bedroom2',hue='Bathroom',data=dataa,ax=ax[1])

# 'Rooms' 'Bedroom2' 'Bathroom'
# 房間可能存在正相關且集中
# 刪除 'Rooms'和'Bedroom2':7以下 'Bathroom':6以下
```

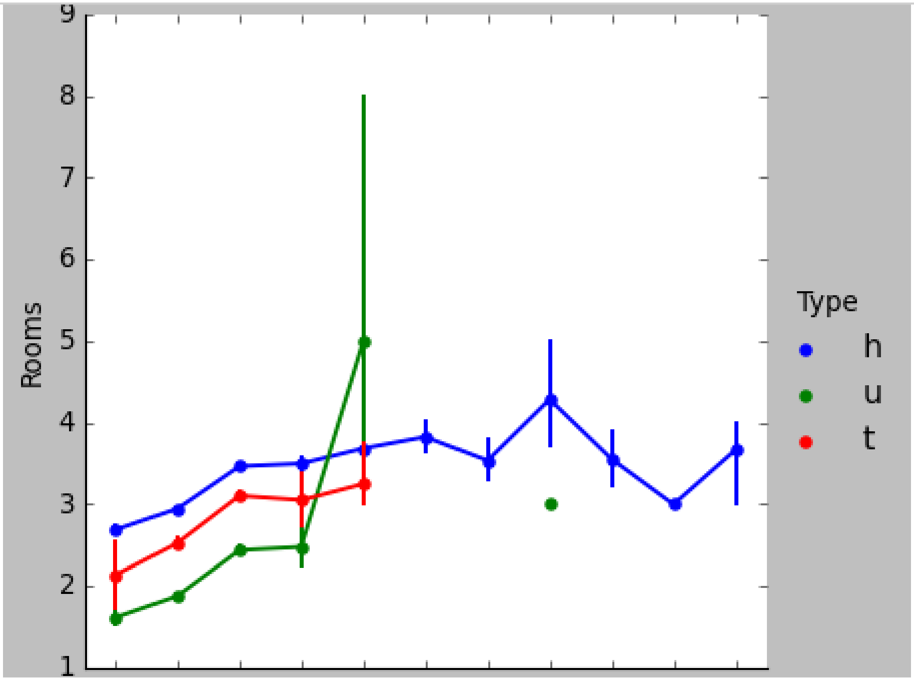
Out[6]:

```
<AxesSubplot:xlabel='Bedroom2', ylabel='count'>
```



In [7]:

```
check_type = pd.crosstab([dataa.Rooms,dataa.Car],dataa.Type,margins=True)
sns.factorplot('Car','Rooms',hue='Type',data=dataa,ax=ax)
check_type
# type U和T車位在5以下
# 房間數在5以上為type U
# Car 6以上刪除
```



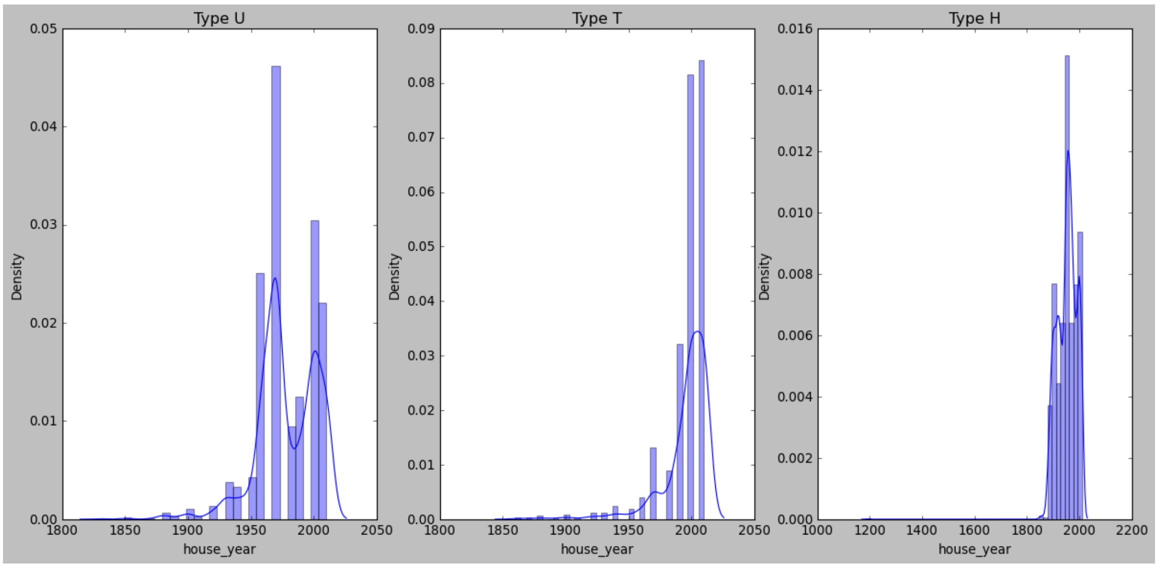
In [8]:

```
dataa['house_year'] = dataa['YearBuilt'].apply(lambda x: (x//10)*10 if x!=None else None
dataa1 = dataa[dataa.house_year !=None]
fig,ax=plt.subplots(1,3,figsize=(18,8))
sns.distplot(dataa1[dataa1['Type']=='u'].house_year,ax=ax[0] )
ax[0].set_title('Type U')
sns.distplot(dataa1[dataa1['Type']=='t'].house_year,ax=ax[1] )
ax[1].set_title('Type T')
sns.distplot(dataa1[dataa1['Type']=='h'].house_year,ax=ax[2] )
ax[2].set_title('Type H')
```

```
C:\Users\jerry\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\jerry\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\jerry\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[8]:

```
Text(0.5, 1.0, 'Type H')
```



In [9]:

```
dataa['house_year'] = dataa['house_year'].apply(lambda x: 1870 if x<1870 else x )
dataa['house_year'].fillna(value=dataa.groupby('Regionname')['house_year'].transform('mec
# yearbuilt 區間合併
# 使用 Regionname 進行插補
# type H主要在 1800年之後建造
```

In [10]:

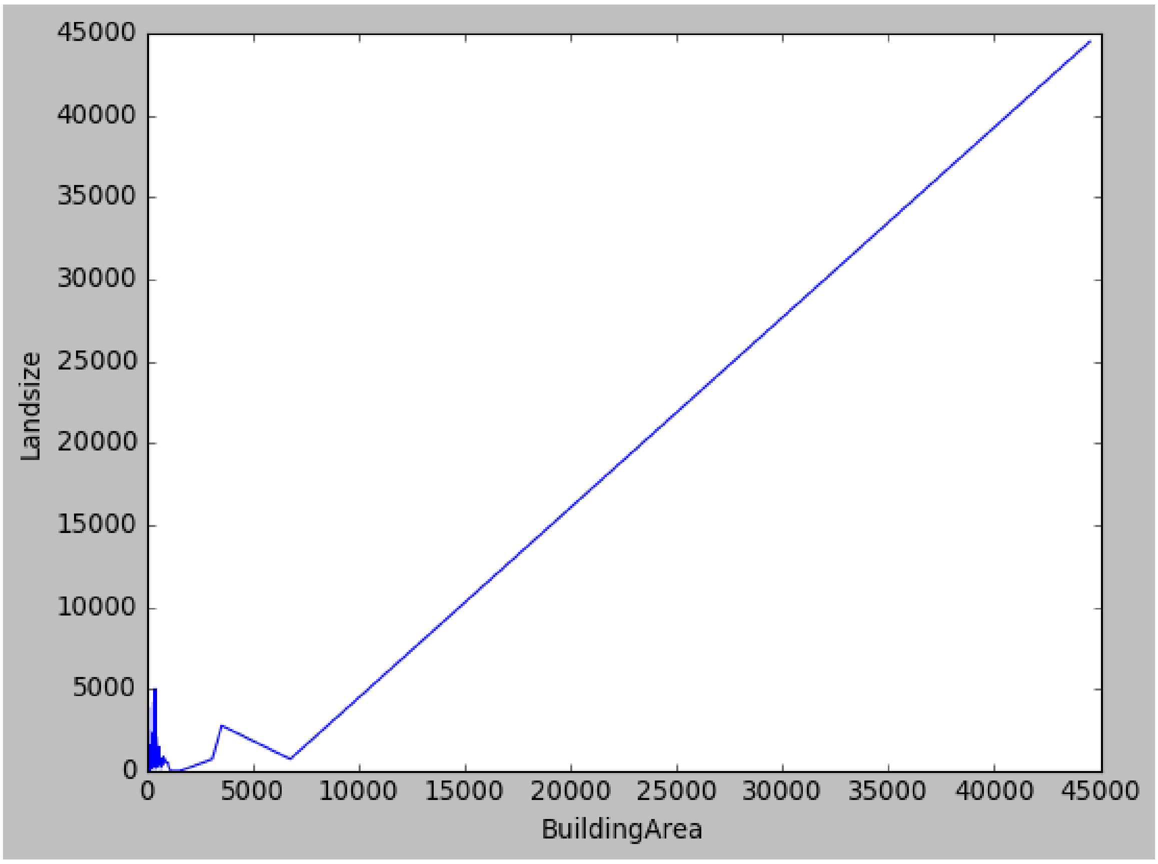
```
# distance 區間合併 27,32,35,38
dataa['Distance'] = dataa['Distance'].apply(lambda x: (x//3)*3 if x>=27 else x )
```

In [12]:

```
# 'BuildingArea'  'Landsize'  刪除
ddta = dataa.dropna(axis=0,subset=('BuildingArea','Landsize'))
sns.lineplot(x='BuildingArea',y='Landsize',data=ddta)
```

Out[12]:

```
<AxesSubplot:xlabel='BuildingArea', ylabel='Landsize'>
```



In [13]:

```
sns.relplot(x= 'Distance',y= 'CouncilArea',hue='month' ,col= 'Method',row='year' ,data=da
# 'CouncilArea'中'Hume'之後僅在2017年出現，且'Distance'在20之內
# 'CouncilArea'與'Distance'似乎存在正相關
```

Out[13]:

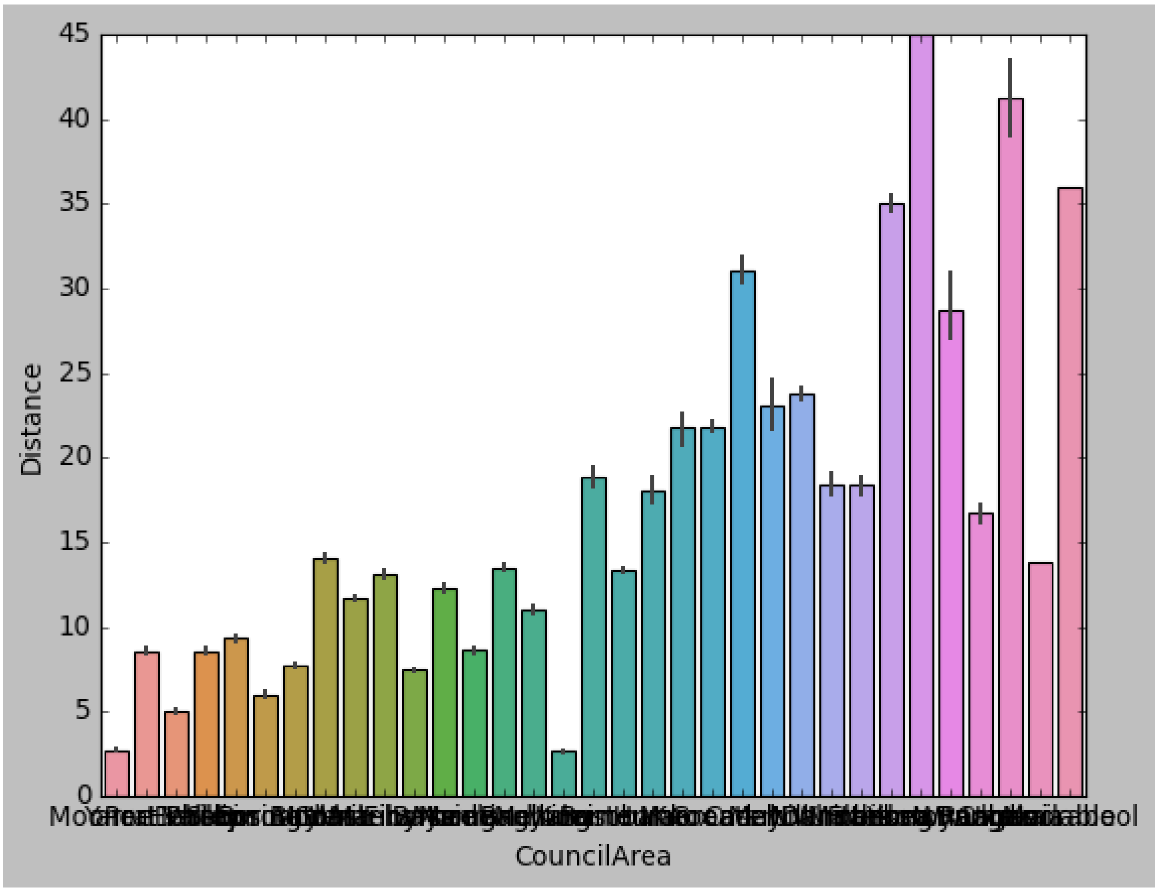
```
<seaborn.axisgrid.FacetGrid at 0x13eba705820>
```

In [14]:

```
sns.barplot(x='CouncilArea',y='Distance',data=dataa)
```

Out[14]:

```
<AxesSubplot:xlabel='CouncilArea', ylabel='Distance'>
```

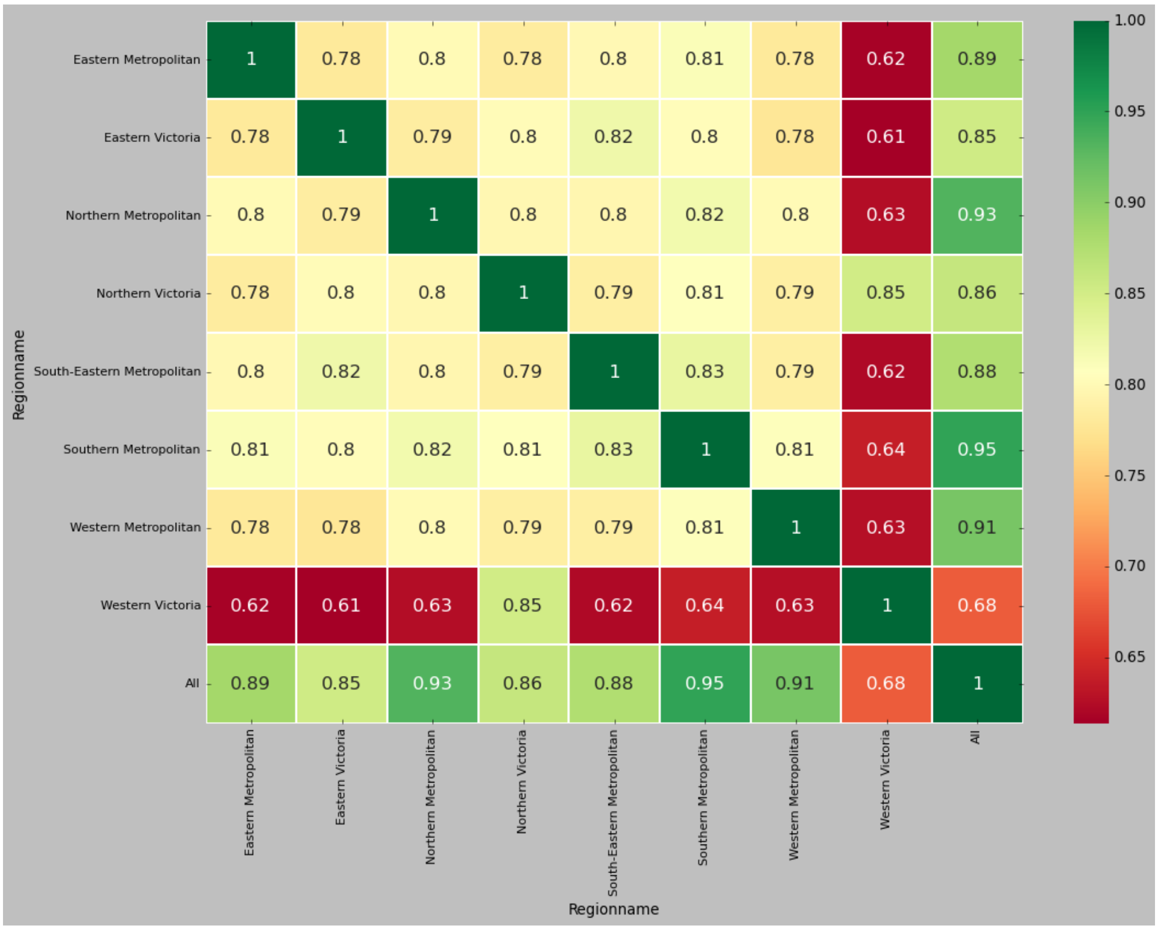


In [15]:

```
check = pd.crosstab(dataa.CouncilArea,dataa.Regionname,margins=True)
sns.heatmap(check.corr(),annot=True,cmap='RdYlGn',linewidths=0.2,annot_kws={'size':15})
fig=plt.gcf()
fig.set_size_inches(15,10)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
# 留'CouncilArea' 刪除 'Distance' 'Regionname'
```

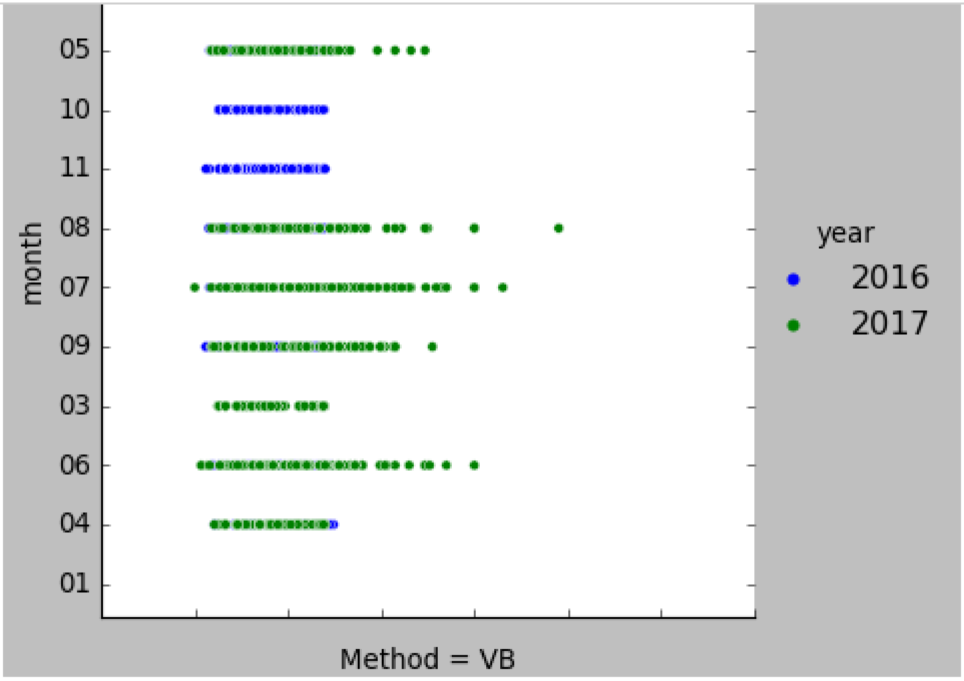
Out[15]:

```
(array([0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5]),
 [Text(0, 0.5, 'Eastern Metropolitan'),
  Text(0, 1.5, 'Eastern Victoria'),
  Text(0, 2.5, 'Northern Metropolitan'),
  Text(0, 3.5, 'Northern Victoria'),
  Text(0, 4.5, 'South-Eastern Metropolitan'),
  Text(0, 5.5, 'Southern Metropolitan'),
  Text(0, 6.5, 'Western Metropolitan'),
  Text(0, 7.5, 'Western Victoria'),
  Text(0, 8.5, 'All')])
```



In [16]:

```
sns.relplot(x= 'Distance',y= 'month',hue='year',row='Method' ,data=dataa)
# type 沒有差異
# 2017年在10 11 12 月沒有銷售
# 刪除年  年差異或許存在問題
```



In [17]:

```
dataa = pd.merge(dataa,y,right_index=True,left_index=True)
dataa = dataa[(dataa.Rooms<7)]
dataa.Price = dataa.Price /10000
data_last = dataa[['Rooms', 'Type', 'Method', 'SellerG', 'Car', 'CouncilArea', 'month', '
```

In [18]:

```
# type 使用 one hotEncoder
from sklearn.preprocessing import OneHotEncoder
onehot = OneHotEncoder(handle_unknown='ignore',sparse=False).fit_transform(data_last[['Ty
colnames = ['Type_' + s for s in sorted(list(set(data_last['Type'])))]
onehotdata = pd.DataFrame(onehot,columns=colnames,index=data_last.index)
onehotdata.head(5)
```

Out[18]:

	Type_h	Type_t	Type_u
0	1.0	0.0	0.0
1	1.0	0.0	0.0
2	1.0	0.0	0.0
3	1.0	0.0	0.0
4	1.0	0.0	0.0

In [19]:

```
# SellerG 使用 Frequency Encoding
FEcod = data_last.SellerG.value_counts()
FEcheck = { FEcod.index[i] :FEcod.values[i]  for i in range(len(FEcod))  }
SellerG = pd.DataFrame(data_last['SellerG'].map(FEcheck) )
SellerG.columns = ['seller_FE']
SellerG.head(5)
```

Out[19]:

	seller_FE
0	393
1	393
2	393
3	393
4	1563

In [20]:

```
# CouncilArea 使用 Target Encoding
TEcod = data_last.groupby('CouncilArea')['Price'].agg('median')
TEcheck = { TEcod.index[i] :TEcod.values[i]  for i in range(len(TEcod))  }
CouncilArea = pd.DataFrame(data_last['CouncilArea'].map(TEcheck) )
CouncilArea.columns = ['CouncilArea_TE']
CouncilArea.head(5)
```

Out[20]:

	CouncilArea_TE
0	109.0
1	109.0
2	109.0
3	109.0
4	109.0

In [21]:

```
# Method 使用 LabelEncoder
from sklearn.preprocessing import LabelEncoder
labelencod = LabelEncoder().fit_transform(data_last['Method'])
labelcod = pd.DataFrame(labelencod,index=data_last.index)
labelcod.columns = ['Method_recode']
labelcod.head(5)
```

Out[21]:

	Method_recode
0	1
1	1
2	3
3	0
4	4

In [22]:

```
data_last = pd.concat([data_last,onehotdata,SellerG,CouncilArea,labelcod],axis=1)
data_last1 = data_last.dropna(axis=0)
y = data_last1.Price.astype('int')
x = data_last1.drop(['Price'],axis=1)
x.month = x.month.astype('int')
x1 = x.select_dtypes(exclude='object')

from sklearn.model_selection import train_test_split
from sklearn import metrics

train_x, val_x, train_y, val_y = train_test_split(x1, y,test_size=0.3, random_state = 106
```

In [23]:

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(train_x,train_y)
pred_logis = model.predict(val_x)
metrics.accuracy_score(pred_logis,val_y)
```

```
C:\Users\jerry\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.
py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

Out[23]:

```
0.017222522597047567
```

In [24]:

```
# Support Vector Machines(Linear and radial)
from sklearn import svm
model=svm.SVC(kernel='rbf',C=1,gamma=0.1)
model.fit(train_x,train_y)
svm_rad=model.predict(val_x)
metrics.accuracy_score(svm_rad,val_y)
```

Out[24]:

```
0.015035538545653362
```

In [25]:

```
from sklearn import svm
model=svm.SVC(kernel='linear',C=0.1,gamma=0.1)
model.fit(train_x,train_y)
svm_lin=model.predict(val_x)
metrics.accuracy_score(svm_lin,val_y)
```

Out[25]:

```
0.023236741388737013
```

In [26]:

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier
model=RandomForestClassifier(n_estimators=100)
model.fit(train_x,train_y)
predRF=model.predict(val_x)
metrics.accuracy_score(predRF,val_y)
```

Out[26]:

```
0.017495899398578457
```

In [27]:

```
# K-Nearest Neighbours
from sklearn.neighbors import KNeighborsClassifier
model=KNeighborsClassifier()
model.fit(train_x,train_y)
predKN=model.predict(val_x)
metrics.accuracy_score(predKN,val_y)
```

Out[27]:

```
0.010934937124111536
```

In [28]:

```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB
model=GaussianNB()
model.fit(train_x,train_y)
predNB=model.predict(val_x)
metrics.accuracy_score(predNB,val_y)
```

Out[28]:

```
0.003553854565336249
```

In [29]:

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier
model=DecisionTreeClassifier()
model.fit(train_x,train_y)
predtree=model.predict(val_x)
metrics.accuracy_score(predtree,val_y)
```

Out[29]:

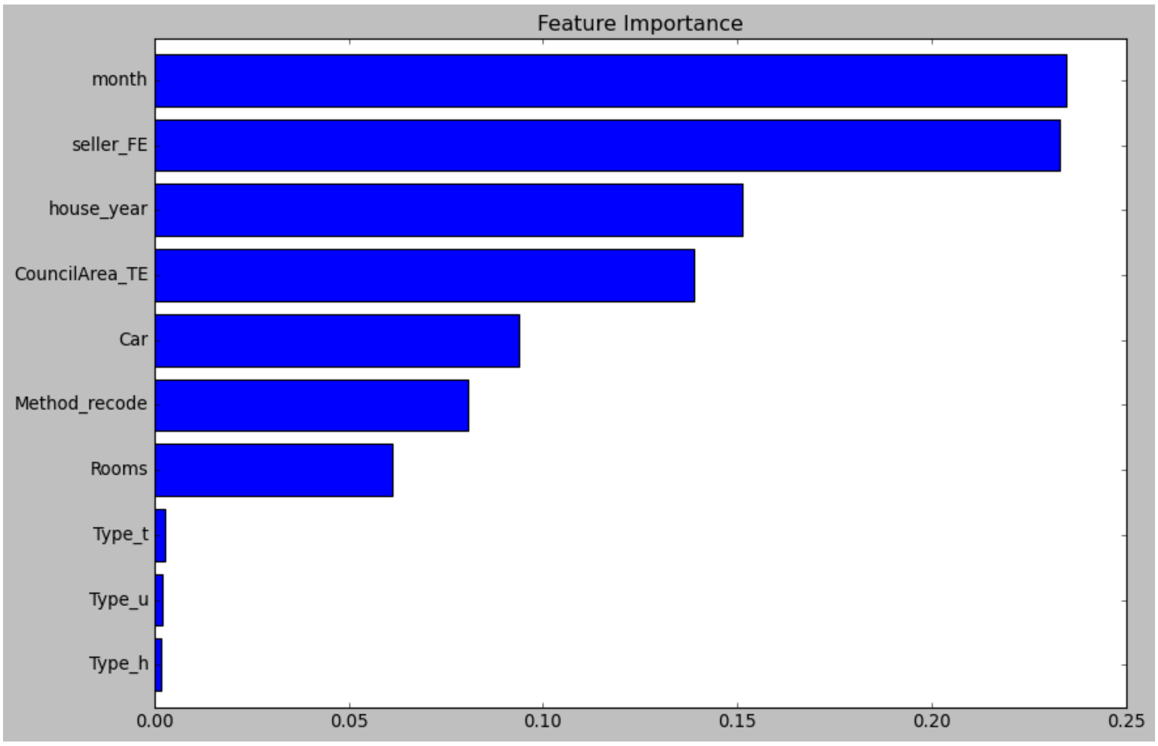
```
0.018316019682886823
```

In [30]:

```
fig, ax = plt.subplots(figsize=(12,8))
pd.Series(model.feature_importances_,train_x.columns).sort_values(ascending=True).plot.ba
ax.set_title('Feature Importance')
```

Out[30]:

```
Text(0.5, 1.0, 'Feature Importance')
```



In [31]:

```
# cross validation
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
```

In [32]:

```
kfold = KFold(n_splits=5)

classfer =['Support Vector Machines(radial)','Random Forest','K-Nearest Neighbours','Naiv
models = [svm.SVC(kernel='rbf'),RandomForestClassifier(n_estimators=100),KNeighborsClassi

cv_mean = []
cv_std = []
for i in models:
    model = i
    cv_result = cross_val_score(model,x1,y,cv=kfold,scoring='accuracy')
    cv_mean.append(cv_result.mean())
    cv_std.append(cv_result.std())

cv_resul = pd.DataFrame({'cv_mean':cv_mean,'cv_std':cv_std},index=classfer)
cv_resul
```

Out[32]:

	cv_mean	cv_std
Support Vector Machines(radial)	0.011728	0.002256
Random Forest	0.018700	0.002803
K-Nearest Neighbours	0.010744	0.001019
Naive Bayes	0.003035	0.001177
Decision Tree	0.015255	0.003032

In [35]:

```
# Bagging(KNN & Tree)
from sklearn.ensemble import BaggingClassifier

model = BaggingClassifier(base_estimator=KNeighborsClassifier(n_neighbors=5),random_state
model.fit(train_x,train_y)
predBKN=model.predict(val_x)
metrics.accuracy_score(predBKN,val_y)
```

Out[35]:

```
0.013942044833242209
```

In [36]:

```
model = BaggingClassifier(base_estimator=DecisionTreeClassifier(),random_state=1024,n_est
model.fit(train_x,train_y)
predBT=model.predict(val_x)
metrics.accuracy_score(predBT,val_y)
```

Out[36]:

```
0.02433023510114817
```