In [1]:
```
from IPython.display import Image
Image(filename='Logo.PNG', height=340, width=900)
```

Out[1]:

EXPLORATORY DATA ANALYSIS

1. **EDA** – plays a critical role in understanding the what, why, and how of the problem statement. It's first in the order of operations that a data analyst will perform when handed a new data source and problem statement.
2. Exploratory Data Analysis is an approach to analyzing data sets by summarizing their main characteristics with visualizations. The EDA process is a crucial step prior to building a model in order to unravel various insights that later become important in developing a robust algorithmic model.
3. Let's understand different operations where EDA comes into play:

```
        a. First and foremost, EDA provides a stage for b
reaking down problem statements into smaller experiments w
hich can help understand the dataset
```

```
        b. EDA provides relevant insights which help anal
ysts make key business decisions
        c. The EDA step provides a platform to run all th
ought experiments and ultimately guides us towards making
 a critical decision
```

Suppose we want to predict the house prices depending on some of the variables, we might need to build a regression model for prediction of house prices!!! But before jumping to model building its preferred to study and understand the data we have.

What is Data Exploration

1. If we wish to build an impeccable predictive model, neither any programming language nor any machine learning algorithm can award it to you unless you perform data exploration
2. Data Exploration not only uncovers the hidden trends and insights, but also allows you to take the first steps towards building a highly accurate model
3. Major time needs to be spent on data exploration, cleaning and preparation as this would take major part of the project time
4. Data cleaning can support better analytics as well as all-round business intelligence which can facilitate better decision making and execution

Importing Libraries & Data

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [3]:
```
eda = pd.read_excel('EDA_Dataset.xlsx')
```

In [4]:
```
eda.head()
```

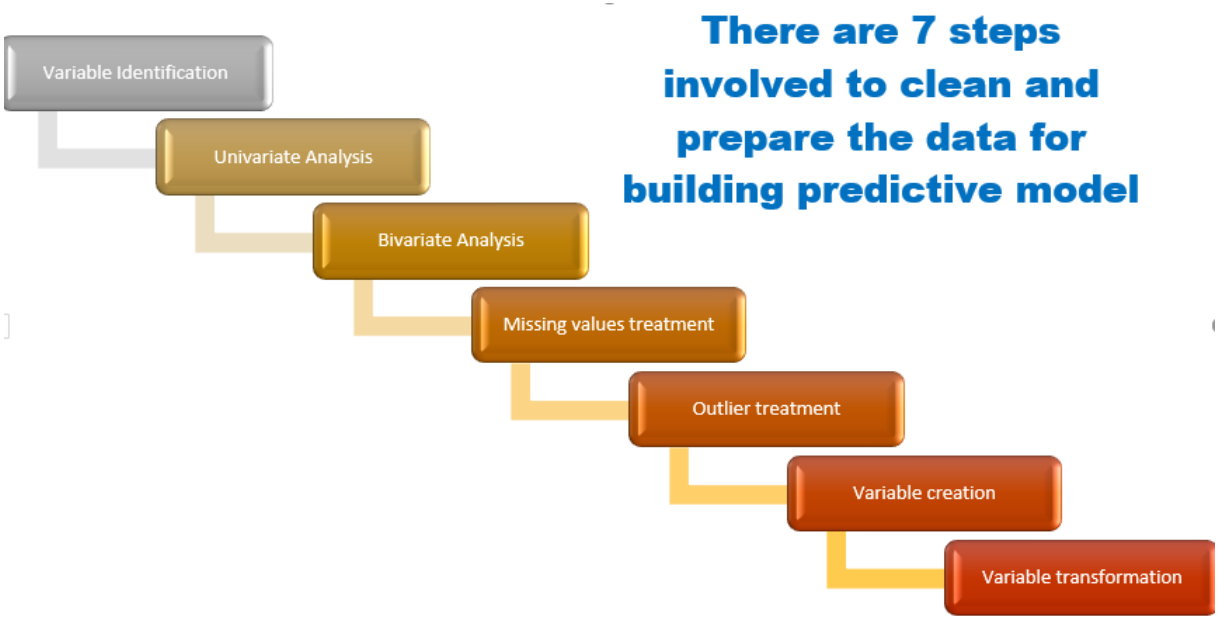
Out[4]:

	Name	Age	Gender	Education	Salary	AppraisedValue	Location	Landacres	House
0	Tony	25	M	Grad	50	700.0	GlenCove	0.2297	2448
1	Harret	52	F	PostGrad	95	364.0	GlenCove	0.2192	1942
2	Jane	26	F	PostGrad	65	600.0	GlenCove	0.1630	2073
3	Rose	45	F	Grad	99	548.4	LongBeach	0.4608	2707
4	John	42	M	Grad	77	405.9	LongBeach	0.2549	2042

Steps For Cleaning

In [5]:
```
from IPython.display import Image
Image(filename='Steps.PNG')
```

Out[5]:



In [6]:
```
eda.head()
```

Out[6]:

	Name	Age	Gender	Education	Salary	AppraisedValue	Location	Landacres	House
0	Tony	25	M	Grad	50	700.0	GlenCove	0.2297	2448
1	Harret	52	F	PostGrad	95	364.0	GlenCove	0.2192	1942
2	Jane	26	F	PostGrad	65	600.0	GlenCove	0.1630	2073
3	Rose	45	F	Grad	99	548.4	LongBeach	0.4608	2707
4	John	42	M	Grad	77	405.9	LongBeach	0.2549	2042

Step 1: Variable Identification

1. Understand the variables and the type of data for each variable

2. We need to identify predictor variables, target variable, data type of variables and category of variables

Dependent Variable: AppraisedValue

Independent Variables/Predictor Variables:

1. Age
2. Gender
3. Education
4. Salary
5. Location
6. Landacres
7. HouseSizesqrft
8. Rooms
9. Baths
10. Garage

In [7]:
```
eda.head()
```

Out[7]:

	Name	Age	Gender	Education	Salary	AppraisedValue	Location	Landacres	House
0	Tony	25	M	Grad	50	700.0	GlenCove	0.2297	2448
1	Harret	52	F	PostGrad	95	364.0	GlenCove	0.2192	1942
2	Jane	26	F	PostGrad	65	600.0	GlenCove	0.1630	2073
3	Rose	45	F	Grad	99	548.4	LongBeach	0.4608	2707
4	John	42	M	Grad	77	405.9	LongBeach	0.2549	2042

In [8]:
```
eda.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 15 entries, 0 to 14
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Name            15 non-null     object
 1   Age             15 non-null     int64
 2   Gender          15 non-null     object
 3   Education       15 non-null     object
 4   Salary          15 non-null     int64
 5   AppraisedValue  15 non-null     float64
 6   Location        15 non-null     object
 7   Landacres       15 non-null     float64
 8   HouseSizesqrft  15 non-null     int64
 9   Rooms           12 non-null     float64
 10  Baths           15 non-null     float64
 11  Garage          15 non-null     int64
dtypes: float64(4), int64(4), object(4)
memory usage: 1.5+ KB
```

Data Types:

String:

1. Name
2. Gender
3. Education
4. Location

Numeric:

1. Age
2. Salary
3. Rooms
4. Baths
5. Garage
6. Landacres

7. HouseSizesqrft
8. Appraised_value

Variable Category:

Categorical:

1. Gender
2. Education
3. Location

Continuous:

1. Age
2. Salary
3. Landacres
4. HouseSizesqrft
5. Appraised_value

Discreet:

1. Rooms
2. Baths
3. Garage

In [9]:
```
eda.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15 entries, 0 to 14
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Name            15 non-null     object
 1   Age             15 non-null     int64
 2   Gender          15 non-null     object
```

```
 3   Education       15 non-null     object
 4   Salary          15 non-null     int64
 5   AppraisedValue  15 non-null     float64
 6   Location        15 non-null     object
 7   Landacres       15 non-null     float64
 8   HouseSizesqrft  15 non-null     int64
 9   Rooms           12 non-null     float64
 10  Baths           15 non-null     float64
 11  Garage          15 non-null     int64
dtypes: float64(4), int64(4), object(4)
memory usage: 1.5+ KB
```

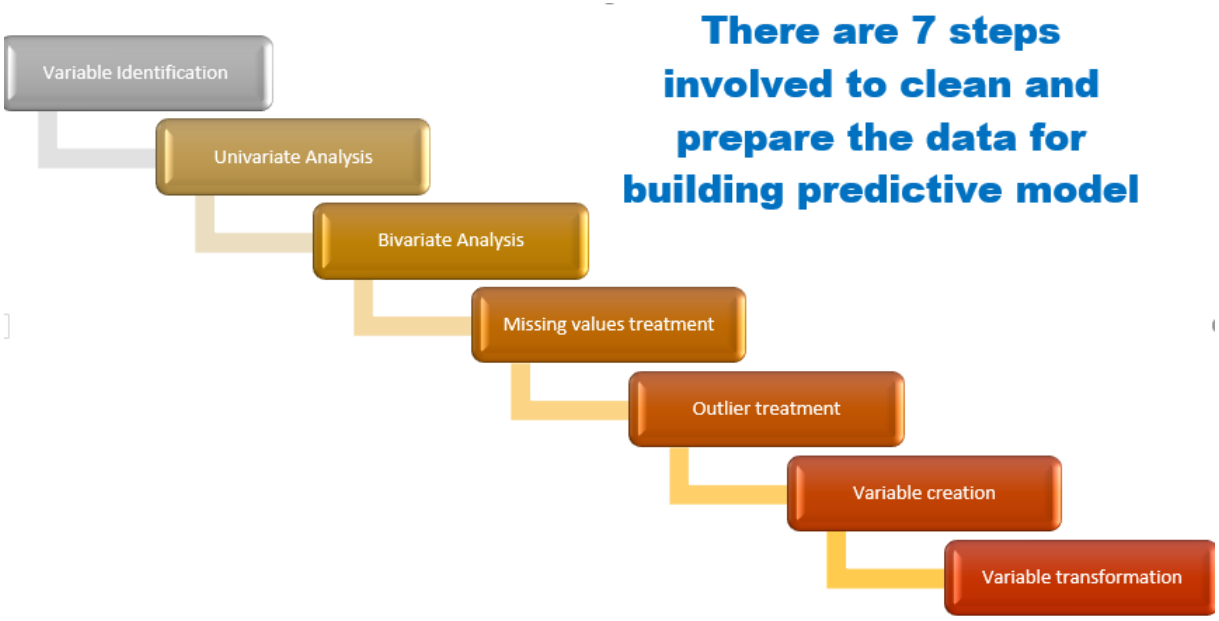
In [10]:
```
eda.describe()
```

Out[10]:

	Age	Salary	AppraisedValue	Landacres	HouseSizesqrft	Rooms	
count	15.000000	15.000000	15.000000	15.000000	15.000000	12.000000	15
mean	40.333333	88.466667	547.240000	0.242487	2140.800000	7.166667	2.3
std	11.842217	22.752917	217.331829	0.093602	754.829517	1.114641	0.7
min	24.000000	50.000000	299.000000	0.137700	1120.000000	5.000000	1.0
25%	29.000000	72.500000	390.350000	0.173200	1707.000000	6.750000	2.0
50%	42.000000	90.000000	517.700000	0.229000	2042.000000	7.000000	2.0
75%	50.000000	104.500000	600.000000	0.252300	2472.000000	8.000000	2.7
max	62.000000	122.000000	1200.000000	0.460800	4067.000000	9.000000	4.0

In [11]:
```
from IPython.display import Image
Image(filename='Steps.PNG')
```

Out[11]:



Step 2: Univariate Analysis

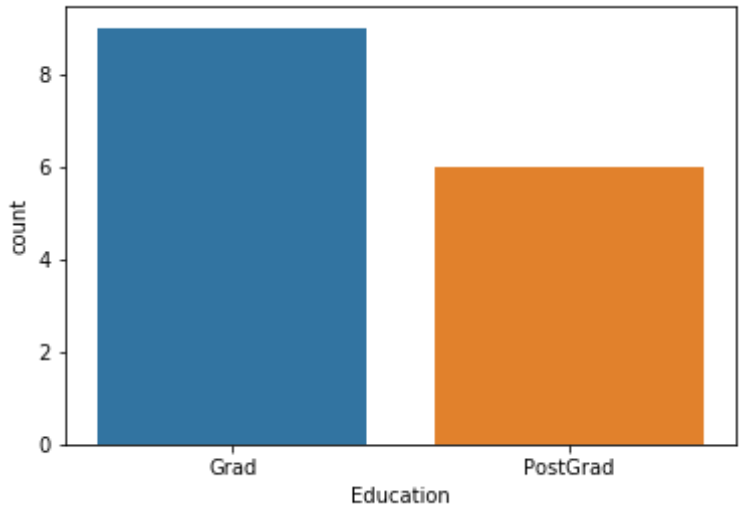
1. Univariate analysis is the simplest form of analyzing data. **“Uni” means “one”**, so we analyze one variable at one time
2. It doesn’t deal with causes or relationships among variables but mostly to describe and summarize and find patterns in the data
3. Used to highlight missing and outlier values
4. Method to perform univariate analysis depends on whether the variable type is categorical or continuous

Categorical Variables - UNIVARIATE ANALYSIS

Visualization of the Categorical variable is shown through the BarPlot.

In [12]:
```
sns.countplot(x='Education', data=eda)
```

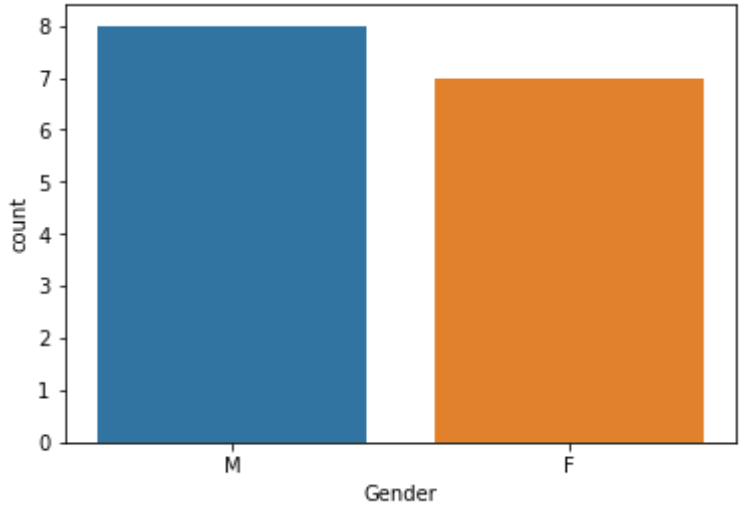
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x1bc8814d0b8>



As we can see the Barplot above, we observe that our data has more number of graduates than postgraduates.

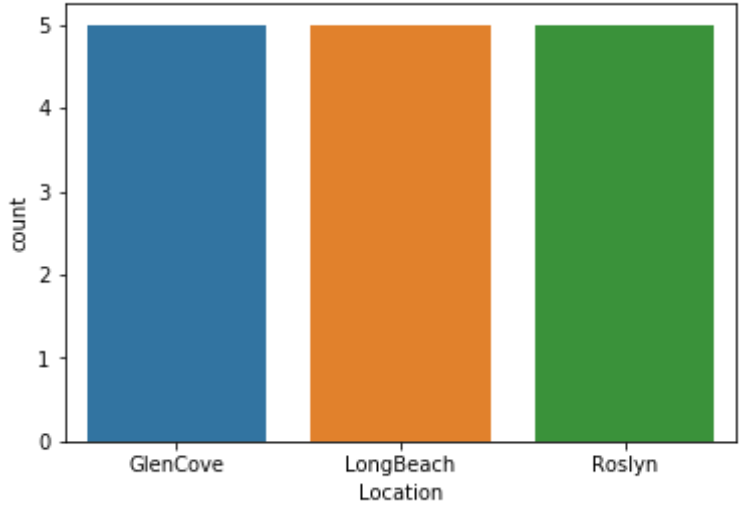
In [13]:
```
sns.countplot(x='Gender', data=eda)
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1bc8a1e2ef0>



In [14]:
```
sns.countplot(x='Location', data=eda)
```

Out[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bc8a21f2b0>
```

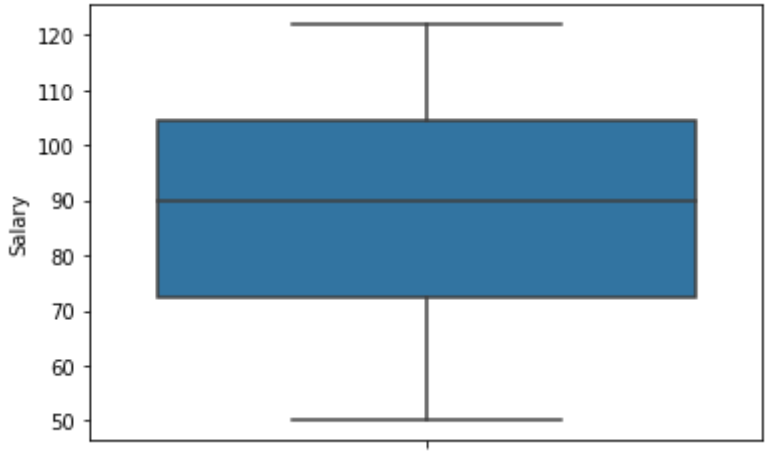


Continuous Variables - UNIVARIATE ANALYSIS

We can view graphical output through Box Plot and Histogram for Continuous variable

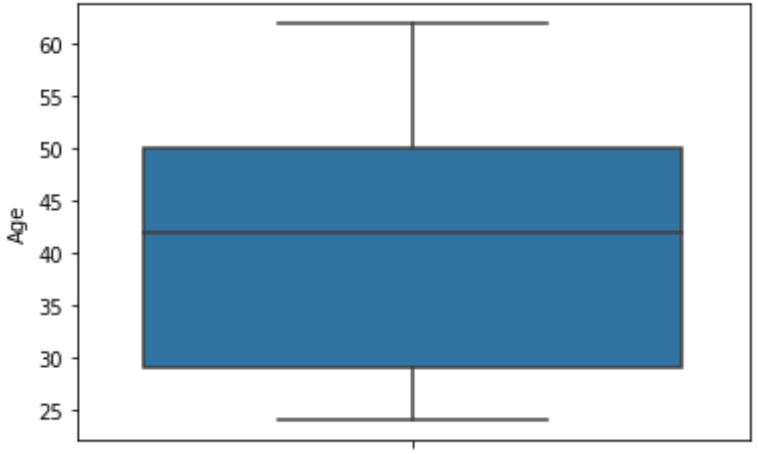
In [15]:
```
sns.boxplot(y='Salary', data=eda)
```

Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bc8a1a8fd0>
```



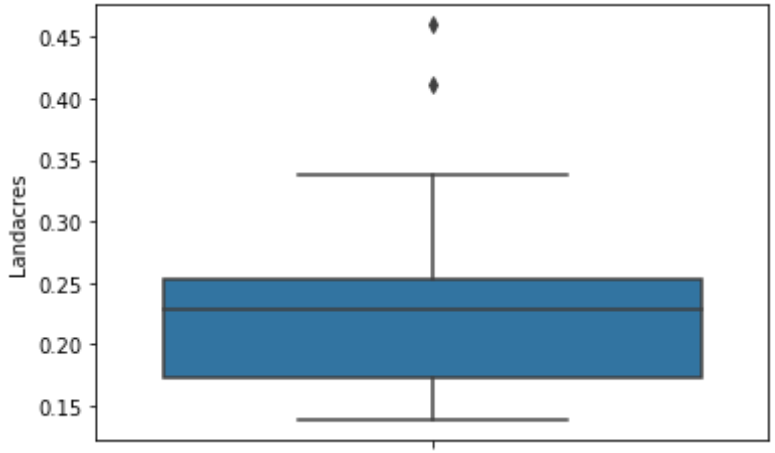
In [16]:
```
sns.boxplot(y='Age', data=eda)
```

Out[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bc8a301da0>
```



In [17]:
```
sns.boxplot(y='Landacres', data=eda)
```

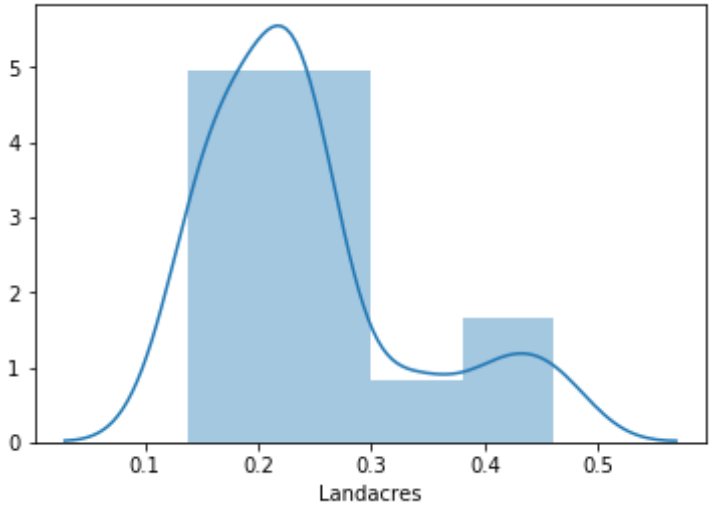
Out[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bc8a34e588>
```



In [18]:
```
sns.distplot(eda['Landacres'], bins=4)
```

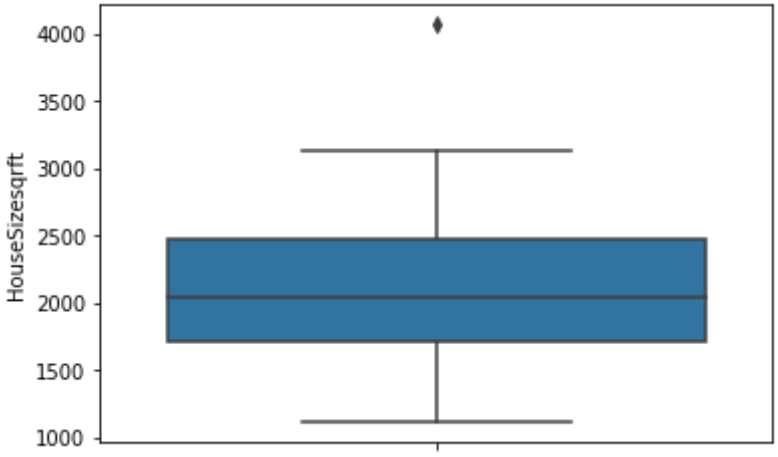
```
C:\Users\Shivani\Anaconda3\lib\site-packages\scipy\stats\stats.py:1706:
FutureWarning: Using a non-tuple sequence for multidimensional indexing
is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the futu
re this will be interpreted as an array index, `arr[np.array(seq)]`, wh
ich will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bc8a3c95c0>
```



In [19]:
```
sns.boxplot(y='HouseSizesqrft', data=eda)
```

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1bc8a497fd0>

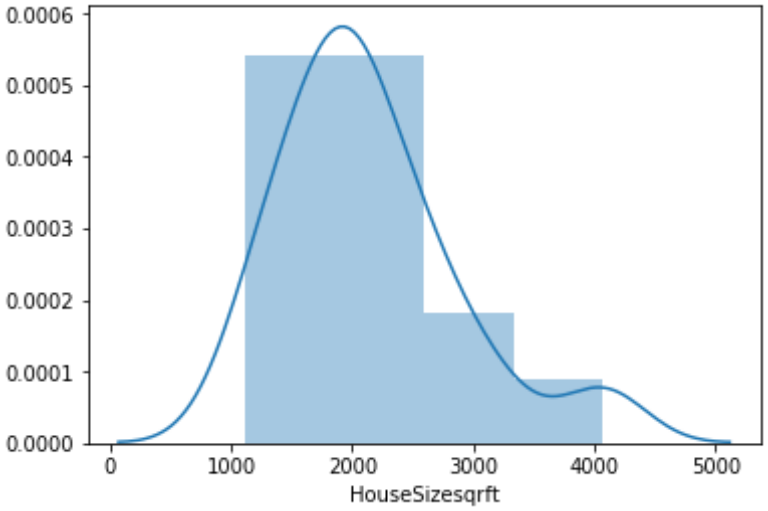


In [20]:
```
sns.distplot(eda['HouseSizesqrft'], bins=4)
```

```
C:\Users\Shivani\Anaconda3\lib\site-packages\scipy\stats\stats.py:1706:
```

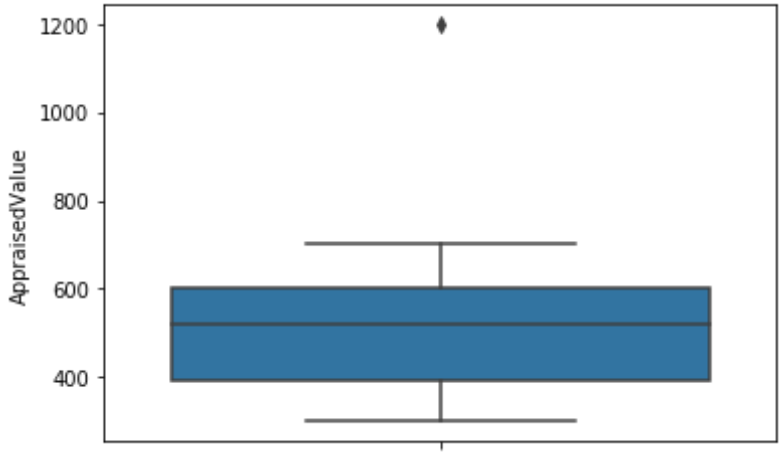
```
FutureWarning: Using a non-tuple sequence for multidimensional indexing
is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the futu
re this will be interpreted as an array index, `arr[np.array(seq)]`, wh
ich will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1bc8a4bf898>



In [21]:
```
sns.boxplot(y='AppraisedValue', data=eda)
```

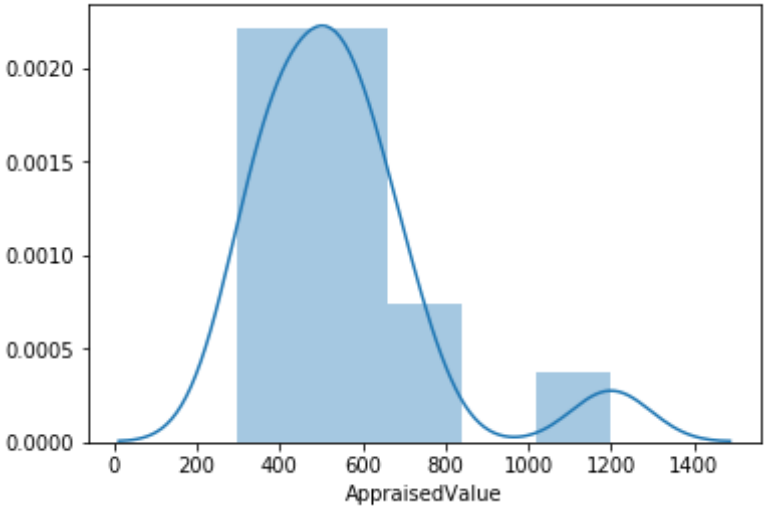
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x1bc8a524a90>



In [22]:
```
sns.distplot(eda['AppraisedValue'], bins=5)
```

```
C:\Users\Shivani\Anaconda3\lib\site-packages\scipy\stats\stats.py:1706:
FutureWarning: Using a non-tuple sequence for multidimensional indexing
is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the futu
re this will be interpreted as an array index, `arr[np.array(seq)]`, wh
ich will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bc8a5caf28>
```



Step 3: Bivariate Analysis

1. In Univariate Analysis, we study one variable at a time, like we did in earlier slides, but if we want to find if there is any relation between two variables we need to perform bivariate analysis.
2. Bivariate analysis, can be performed Categorical & Categoricalfor any combination of categorical and continuous variables
3. Different methods are used to tackle different combinations during analysis process.
4. Possible Combinations are:

    a. Continuous & Continuous
    b. Continuous & Categorical
    c. Categorical & Categorical

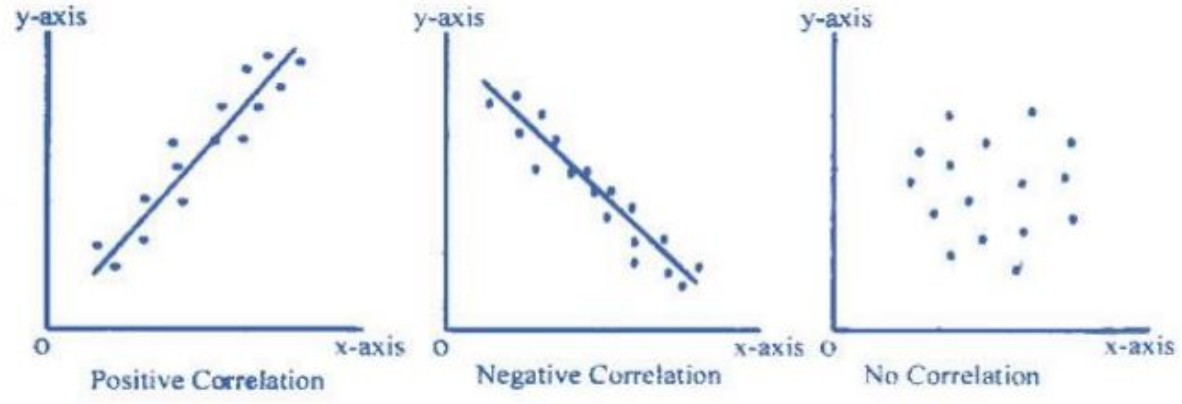
Continuous & Continuous

Scatter plot

1. find out the relationship between two variables
2. The pattern of scatter plot indicates the relationship between variables, but does not indicates the strength of relationship amongst them
3. The relationship can be linear or non-linear
4. To find the strength of the relationship, we use Correlation(-1 negative linear correlation to +1 positive linear correlation and 0 is no correlation).
5. We get an idea of some relation and pattern among 2 variables in the dataset

In [23]:
```
from IPython.display import Image
Image(filename='Correlation.jpg')
```

Out[23]:



Let us check scatter plot for few continuous variables. We have the following Continuous Variables:

Continuous:

1. Age
2. Salary
3. Landacres
4. HouseSizesqrft
5. Appraisedvalue

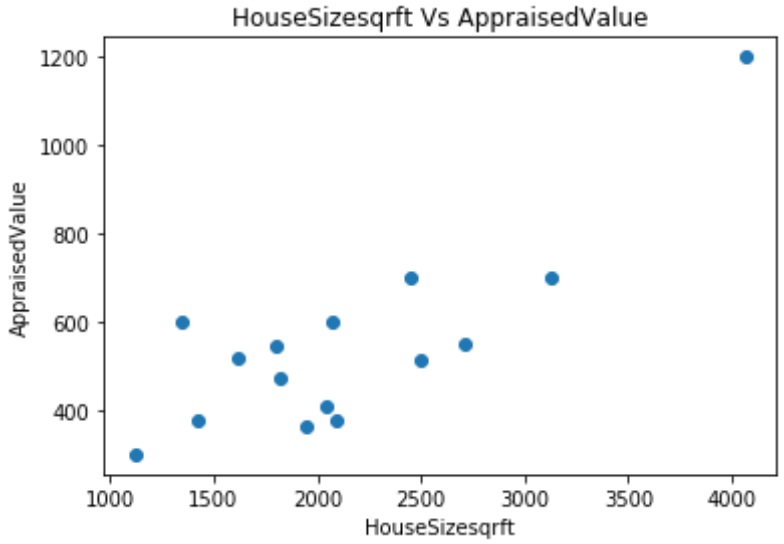
In [24]:
```
plt.scatter(eda['Age'], eda['Salary'])
plt.title('Age Vs Salary')
plt.xlabel('Age')
plt.ylabel('Salary')
```

Out[24]: Text(0,0.5,'Salary')

This scatter plot tells us that there is positive linear relationship between Age and Salary, these two are continuous variables

In [25]:
```
plt.scatter(eda['HouseSizesqrft'], eda['AppraisedValue'])
plt.title('HouseSizesqrft Vs AppraisedValue')
plt.xlabel('HouseSizesqrft')
plt.ylabel('AppraisedValue')
```

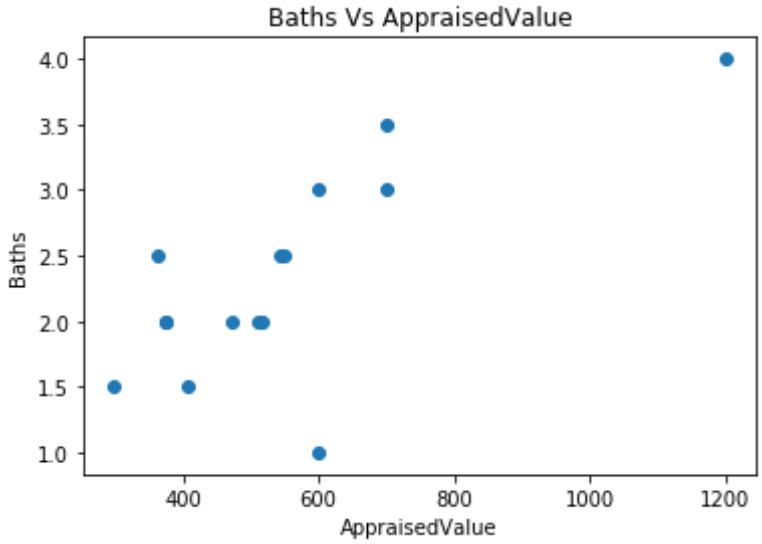
Out[25]: Text(0,0.5,'AppraisedValue')



This scatter plot tells us that there is positive linear relationship between HouseSizesqrft and AppraisedValue, these two are continuous variables. Also, please note the existence of a possible outlier on top right corner

In [26]:
```
plt.scatter(eda['AppraisedValue'], eda['Baths'])
plt.title('Baths Vs AppraisedValue')
plt.xlabel('AppraisedValue')
plt.ylabel('Baths')
```

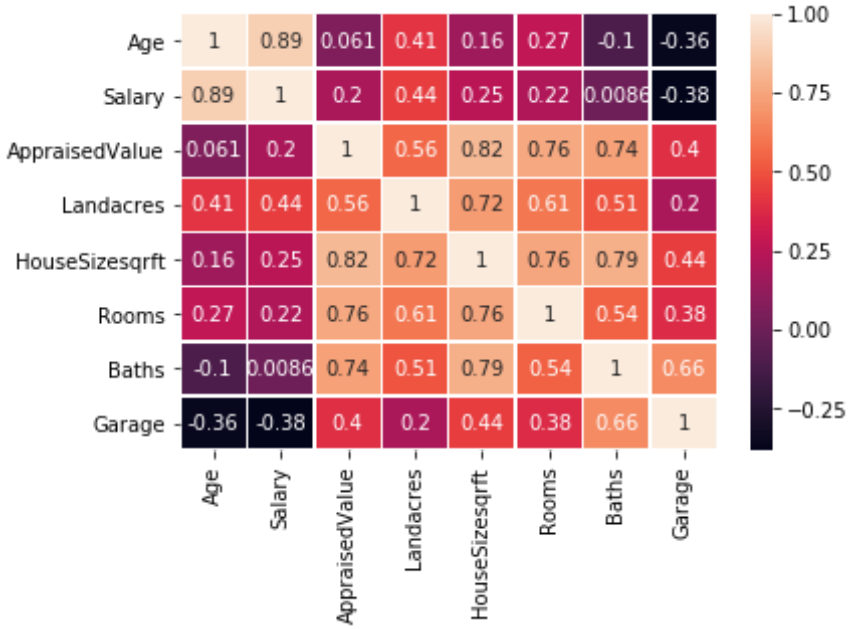
Out[26]:
```
Text(0,0.5,'Baths')
```



Let's build a heatmap. For exploring correlations between features, a heatmap is among the best visual tools. The individual values in the data matrix are represented by different colors helping quickly see what features have the most and the least dependencies.

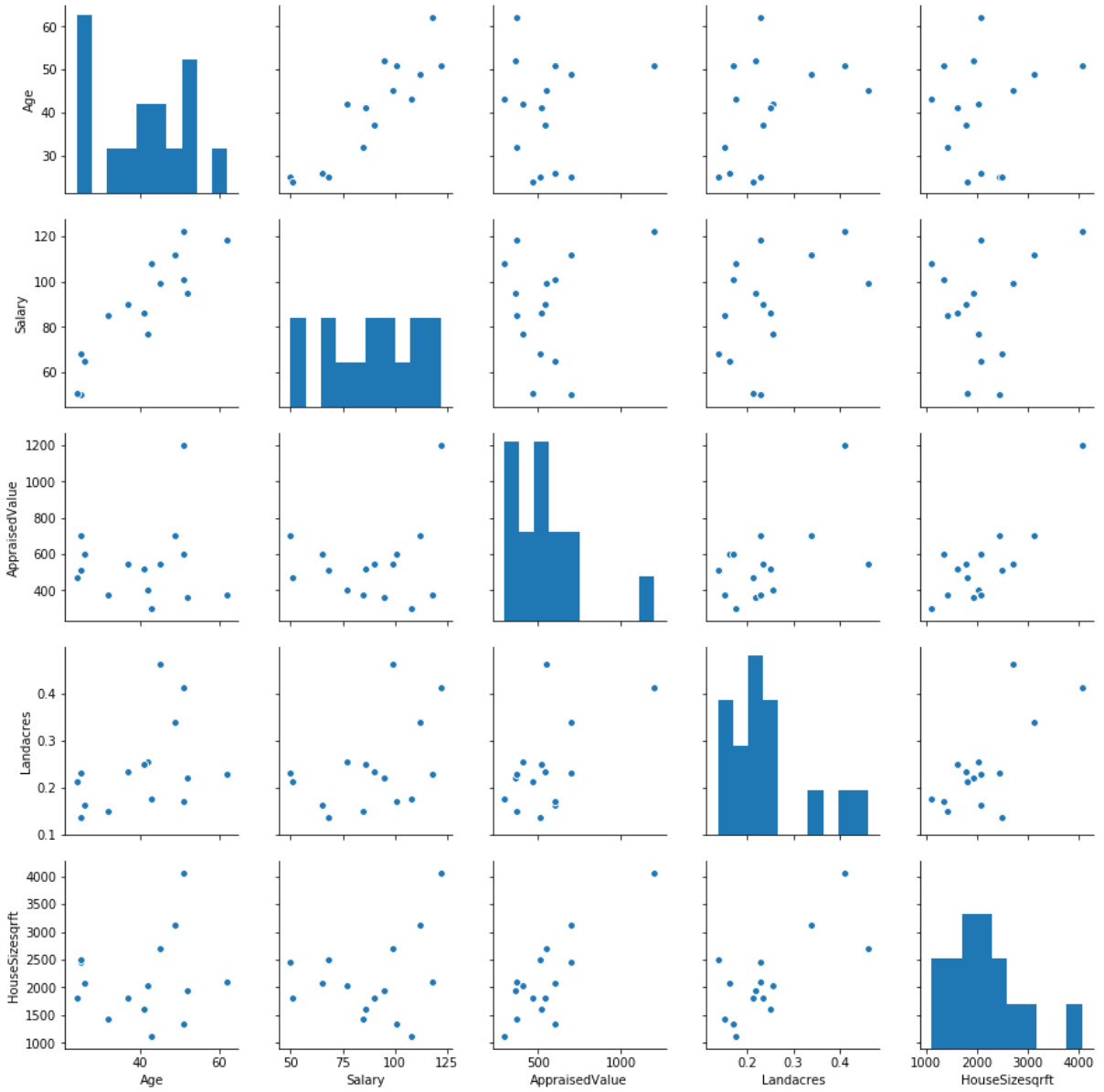
In [27]:
```
sns.heatmap(eda.corr(), annot=True, linewidth=0.5)
```

Out[27]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bc8b7c9828>
```



In [28]:
```
eda_cont = eda.iloc[:,:-3]
eda_cont
sns.pairplot(eda_cont)
```

Out[28]: <seaborn.axisgrid.PairGrid at 0x1bc8a653940>



Categorical & Categorical

Methods to identify the relationship between two categorical variables:

Two-Way Table: In this method by creating a two-way table of count and count%. Both row and column represents category of their respected variable.

Stacked Column Chart: This method is one of the most visual form of Two-way table

Categorical:

1. Gender
2. Education
3. Location

Continuous:

1. Age
2. Salary
3. Landacres
4. HouseSizesqrft
5. Appraised_value

In [29]:
```
counts = eda.groupby(['Gender', 'Education'], axis= 0)
counts.size()
```

Out[29]:
```
Gender  Education
F       Grad         3
        PostGrad     4
M       Grad         6
        PostGrad     2
dtype: int64
```

Through the two-way table for Education and Gender, we see that more Males are Grads whereas more Females are Post Grads.

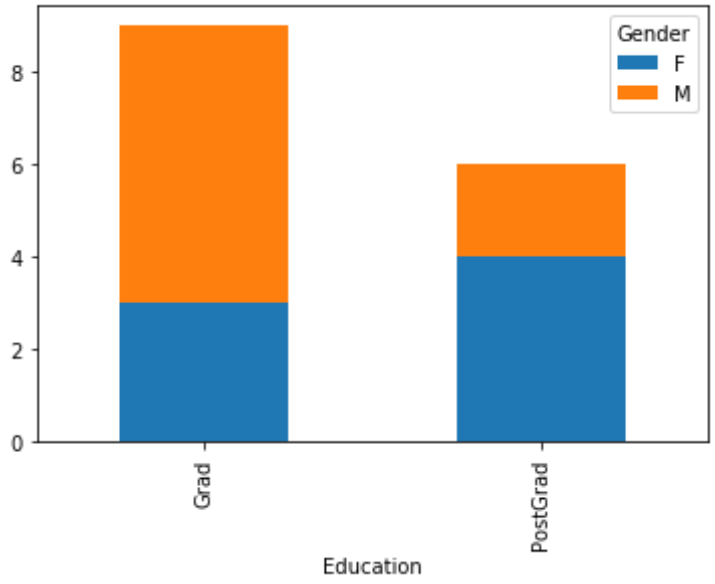
In [30]:
```
counts1 = eda.groupby(['Education', 'Location'], axis= 0)
counts1.size()
```

Out[30]:
```
Education  Location
Grad       GlenCove     2
           LongBeach    5
           Roslyn       2
PostGrad   GlenCove     3
           Roslyn       3
dtype: int64
```

Through the two-way table for Education and Location, we see that more Longbeach has more Grads as compared to other locations whereas there is equal distribution of Males & Female as Post Grads.

In [31]:
```
pd.crosstab(eda['Education'],eda['Gender']).plot(kind='bar',stacked=True)
```

Out[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bc8c93f160>
```



Step 4: Missing Values Treatment

1. There may be situations where there could be missing values in your data.
2. Missing Data will not make any impact on the result if its percentage is less 1%, if missing data's range within the range of 1-5% then it is somehow manageable; however in case of 5-15% complex techniques are used for handling the problems of missing data but if it exceeds from 15% then it will surely hinder the result achieved after applying data mining techniques
3. Handling such values is very important as this could lead to wrong results
4. Missing values could occur due to several reasons like,

```
a. – During data extraction i.e. while fetching the data
required for the analysis
b. – During data collection itself there could be some fi
elds for which the values may not have been collected.
```

5. But there are ways to handle these problems

In [32]:
```
eda.isnull()
```

Out[32]:

	Name	Age	Gender	Education	Salary	AppraisedValue	Location	Landacres	Hous
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False
8	False	False	False	False	False	False	False	False	False

	Name	Age	Gender	Education	Salary	AppraisedValue	Location	Landacres	Hous
9	False	False	False	False	False	False	False	False	False
10	False	False	False	False	False	False	False	False	False
11	False	False	False	False	False	False	False	False	False
12	False	False	False	False	False	False	False	False	False
13	False	False	False	False	False	False	False	False	False
14	False	False	False	False	False	False	False	False	False

In [33]:
```
#To check if there are any null values in the dataset:
eda.isnull().values.any()
```

Out[33]: True

In [34]:
```
eda.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15 entries, 0 to 14
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Name            15 non-null     object 
 1   Age             15 non-null     int64  
 2   Gender          15 non-null     object 
 3   Education       15 non-null     object 
 4   Salary          15 non-null     int64  
 5   AppraisedValue  15 non-null     float64
 6   Location        15 non-null     object 
 7   Landacres       15 non-null     float64
 8   HouseSizesqrft  15 non-null     int64  
 9   Rooms           12 non-null     float64
 10  Baths           15 non-null     float64
 11  Garage          15 non-null     int64  
```

```
dtypes: float64(4), int64(4), object(4)
memory usage: 1.5+ KB
```

In [35]:
```
eda.describe()
```

Out[35]:

	Age	Salary	AppraisedValue	Landacres	HouseSizesqrft	Rooms	
count	15.000000	15.000000	15.000000	15.000000	15.000000	12.000000	15
mean	40.333333	88.466667	547.240000	0.242487	2140.800000	7.166667	2.3
std	11.842217	22.752917	217.331829	0.093602	754.829517	1.114641	0.7
min	24.000000	50.000000	299.000000	0.137700	1120.000000	5.000000	1.0
25%	29.000000	72.500000	390.350000	0.173200	1707.000000	6.750000	2.0
50%	42.000000	90.000000	517.700000	0.229000	2042.000000	7.000000	2.0
75%	50.000000	104.500000	600.000000	0.252300	2472.000000	8.000000	2.7
max	62.000000	122.000000	1200.000000	0.460800	4067.000000	9.000000	4.0

In [36]:
```
eda.isnull().sum()
```

Out[36]:
```
Name              0
Age               0
Gender            0
Education         0
Salary            0
AppraisedValue    0
Location          0
Landacres         0
HouseSizesqrft    0
Rooms             3
Baths             0
Garage            0
dtype: int64
```

In [37]:
```
eda.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15 entries, 0 to 14
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Name            15 non-null     object 
 1   Age             15 non-null     int64  
 2   Gender          15 non-null     object 
 3   Education       15 non-null     object 
 4   Salary          15 non-null     int64  
 5   AppraisedValue  15 non-null     float64
 6   Location        15 non-null     object 
 7   Landacres       15 non-null     float64
 8   HouseSizesqrft  15 non-null     int64  
 9   Rooms           12 non-null     float64
 10  Baths           15 non-null     float64
 11  Garage          15 non-null     int64  
dtypes: float64(4), int64(4), object(4)
memory usage: 1.5+ KB
```

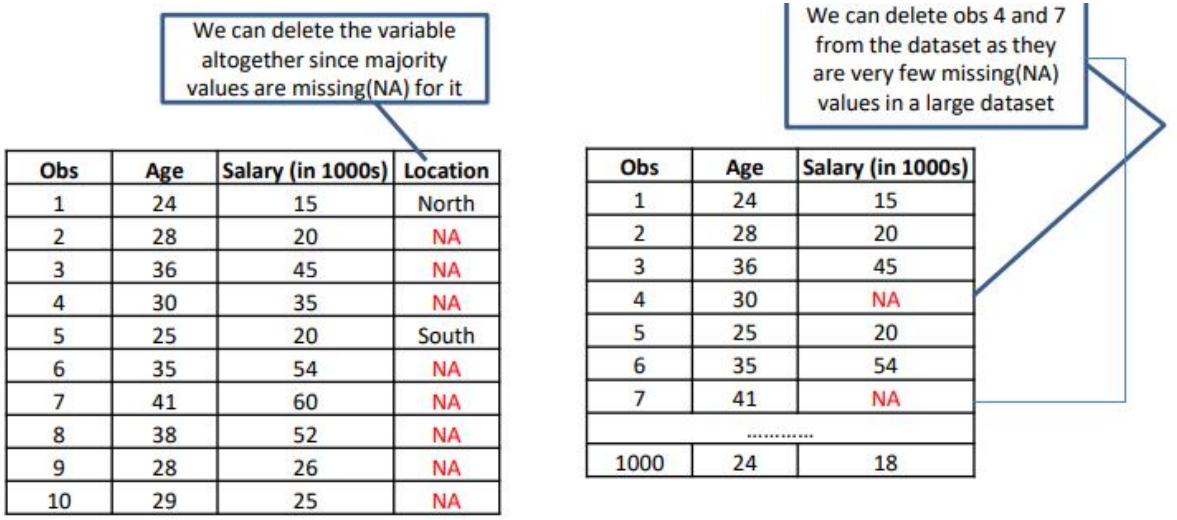
1. Deletion: Deleting observations or variables.

If a particular variable is having more missing values than rest of the variables in the dataset, then we are better off without that variable unless it is a really important predictor that makes a lot of business sense.

Also, if in a huge dataset we have very minute number of observations missing, then we can delete the whole of observations altogether.

In [38]:
```
from IPython.display import Image
Image(filename='DeleteMissing.jpg')
```

Out[38]:



Obs	Age	Salary (in 1000s)	Location
1	24	15	North
2	28	20	NA
3	36	45	NA
4	30	35	NA
5	25	20	South
6	35	54	NA
7	41	60	NA
8	38	52	NA
9	28	26	NA
10	29	25	NA

Obs	Age	Salary (in 1000s)
1	24	15
2	28	20
3	36	45
4	30	NA
5	25	20
6	35	54
7	41	NA
.....		
1000	24	18

2. Single Imputation: In single imputation, we use mean, median or mode.

If the variable is numeric then replace the missing values with either mean, median or mode.

If the variable is otherwise generally normally distributed (and in particular does not have any skewness), we would choose mean.

If the data skewed, median imputation is suggested.

If the variable is categorical then we could replace the missing values with the most frequent occurring value in that variable, i.e the mode.

We use the fillna function of pandas to replace na values with the value of our interest and inplace=True command makes the permanently changes the value in that dataframe

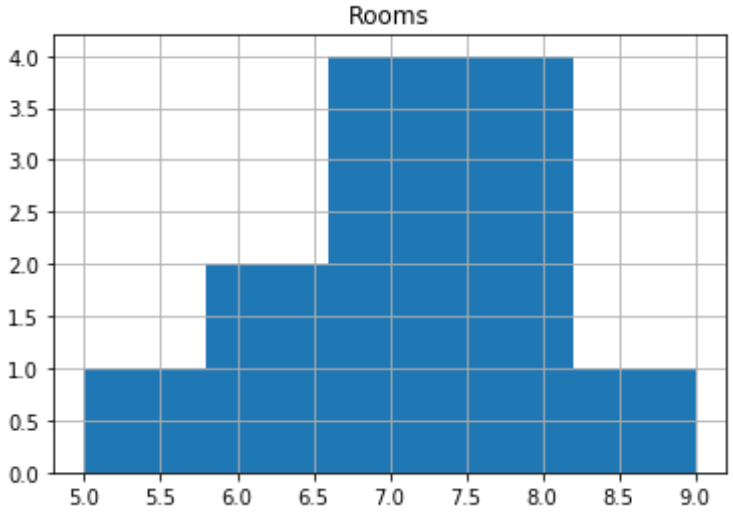
dataframe.['col_name'].fillna(0, inplace=True)

We can see that the variable Rooms has 3 missing values, we need to find a way to replace the missing values

Looking at the histogram of the variable Rooms (non missing value, we see that it is normally distributed. Hence we can impute missing values with Mean of nonmissing data

In [39]:
```
eda.hist(column=['Rooms'], bins=5)
```

Out[39]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001BC8C991B70>]],
      dtype=object)
```



Looking at the histogram of the variable Rooms (non missing value, we see that it is normally distributed. Hence we can impute missing values with Mean of nonmissing data

In [40]:
```
eda['Rooms']=eda['Rooms'].fillna(value=eda['Rooms'].median())
```

In [42]:
```
eda
```

Out[42]:

	Name	Age	Gender	Education	Salary	AppraisedValue	Location	Landacres	Hou
0	Tony	25	M	Grad	50	700.0	GlenCove	0.2297	244

	Name	Age	Gender	Education	Salary	AppraisedValue	Location	Landacres	Hou
1	Harret	52	F	PostGrad	95	364.0	GlenCove	0.2192	194
2	Jane	26	F	PostGrad	65	600.0	GlenCove	0.1630	207
3	Rose	45	F	Grad	99	548.4	LongBeach	0.4608	270
4	John	42	M	Grad	77	405.9	LongBeach	0.2549	204
5	Mark	62	M	PostGrad	118	374.1	GlenCove	0.2290	208
6	Bruce	51	M	Grad	101	600.0	GlenCove	0.1714	134
7	Steve	43	M	Grad	108	299.0	Roslyn	0.1750	112
8	Carol	24	F	PostGrad	51	471.0	Roslyn	0.2130	181
9	Henry	25	M	PostGrad	68	510.7	Roslyn	0.1377	249
10	Donald	41	M	Grad	86	517.7	LongBeach	0.2497	161
11	Maria	51	F	Grad	122	1200.0	LongBeach	0.4116	406
12	Janet	49	F	PostGrad	112	700.0	Roslyn	0.3372	313
13	Sophia	32	F	Grad	85	374.8	Roslyn	0.1503	142
14	Jeffery	37	M	Grad	90	543.0	LongBeach	0.2348	179

Constant:

This choice allows us to provide our own default value to fill in the gaps. This might be an integer or real number for numeric variables, or else a special marker or the choice of something other than the majority category for Categorical variables.

Closest fit:

The closest fit algorithm depends upon exchanging absent values with present value of the similar attribute of other likewise cases. Main notion is to find out from dataset likewise scenarios

and select the likewise case to the case in discussion with missing attribute values. This method is more useful for a small dataset

Step 5: Outliers

What is an Outlier?

1. Outlier is an observation that appears far away and diverges from an overall pattern in a sample.
2. Outliers can drastically change the results of the data analysis and statistical modeling. There are numerous unfavorable impacts of outliers in the data set:

```
    a. It increases the error variance and reduces the power
    of statistical tests
    b. If the outliers are non-randomly distributed, they can
    decrease normality
    c. They can bias or influence estimates that may be of su
   bstantive interest
```

Causes of Outliers:

1. Data Entry Errors - Human errors such as errors caused during data collection, recording, or entry can cause outliers in data.
2. Measurement Error - When the measurement instrument used turns out to be faulty.
3. Intentional Error - This is commonly found in self-reported measures that involves sensitive data.
4. Data Processing Error - When data is collected from different sources
5. Sampling Error - Data considered which is not part of the sample
6. Natural Outlier - When an outlier is not artificial (due to error), it is a natural outlier.

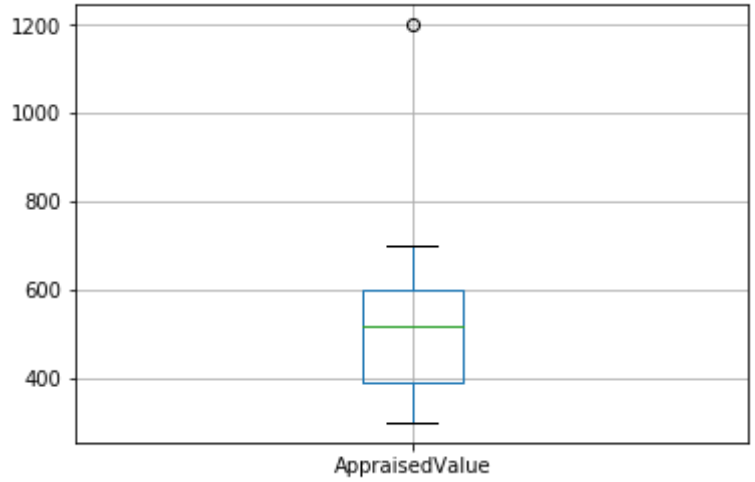
Outlier Detection

1. Outliers can be detected using boxplots and scatter plots

1. In our data, we plot a scatter plot for Appraised_value and Baths(bivariate analysis) and also a boxplot for Appraised_value(Univariate analysis)

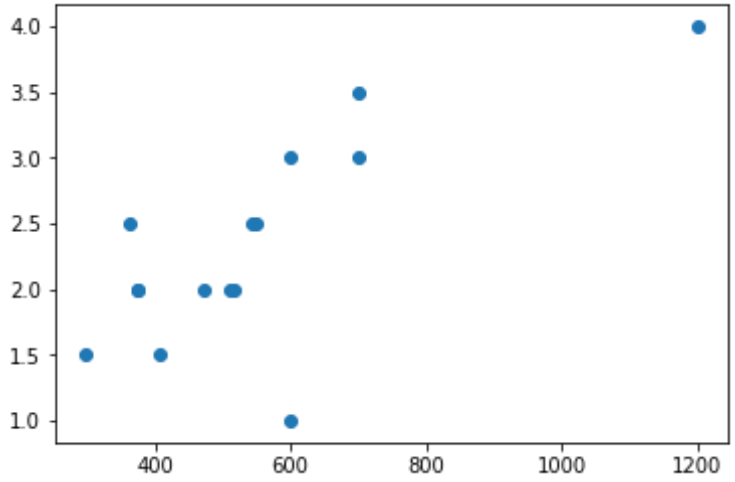
In [43]:
```
eda.boxplot(column = ['AppraisedValue'])
```

Out[43]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bc8c9d3ac8>
```



In [44]:
```
plt.scatter(eda['AppraisedValue'], eda['Baths'])
```

Out[44]:
```
<matplotlib.collections.PathCollection at 0x1bc8cb0f7b8>
```



Other than the plots, Outliers can also be detected by using certain thumb rules,

1. – Any value, which is beyond the range of -1.5x IQR to 1.5 x IQR where IQR = Q3-Q1
2. – Any value which out of range of 5th and 95th percentile can be considered as outlier
3. – Data points, three or more standard deviation away from mean are considered outlier

In [45]:
```
eda['AppraisedValue'].describe()   #Q3 + 1.5IQR; Q1-1.5IQR
```

Out[45]:
```
count      15.000000
mean      547.240000
std       217.331829
min       299.000000
25%       390.350000
50%       517.700000
75%       600.000000
max      1200.000000
Name: AppraisedValue, dtype: float64
```

In [46]:
```
IQR = eda['AppraisedValue'].quantile(0.75)-eda['AppraisedValue'].quanti
le(0.25)
print(IQR)
```

```
209.64999999999998
```

In [47]:
```
import numpy as np
```

In [48]:
```
Upper_OutlierLimit = eda['AppraisedValue'].quantile(0.75) + 1.5*IQR
Lower_OutlierLimit = eda['AppraisedValue'].quantile(0.25) - 1.5*IQR

print(Upper_OutlierLimit)
print(Lower_OutlierLimit)
```

```
914.4749999999999
75.87500000000006
```

In [49]:
```
OutlierValues = eda[(eda['AppraisedValue']>=Upper_OutlierLimit) | (eda[
'AppraisedValue']<=Lower_OutlierLimit)]
```

In [50]:
```
OutlierValues
```

Out[50]:

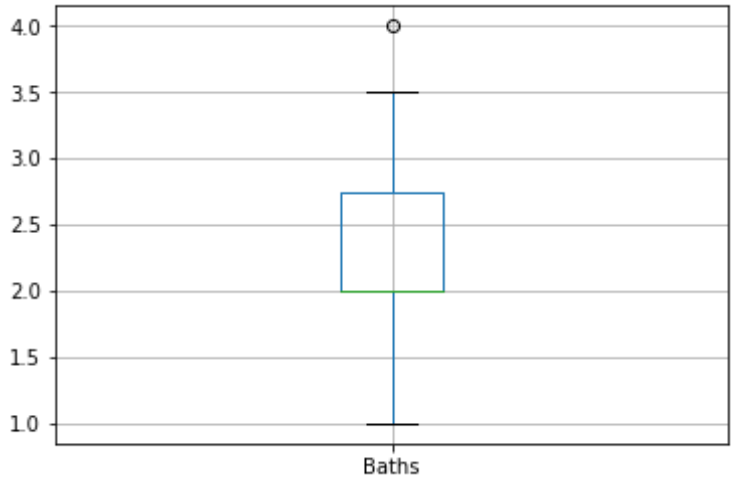
	Name	Age	Gender	Education	Salary	AppraisedValue	Location	Landacres	Hous
11	Maria	51	F	Grad	122	1200.0	LongBeach	0.4116	4067

In [51]:
```
eda['Baths'].describe()
```

Out[51]:
```
count    15.000000
mean      2.333333
std       0.794325
min       1.000000
25%       2.000000
50%       2.000000
75%       2.750000
max       4.000000
Name: Baths, dtype: float64
```

In [52]:
```
eda.boxplot(column = ['Baths'])
```

Out[52]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bc8ca08eb8>
```



In [53]:
```
IQR_Baths = eda['Baths'].quantile(0.75) - eda['Baths'].quantile(0.25)
print(IQR_Baths)
```

```
0.75
```

In [54]:
```
Upper_OutlierLimit_Baths = eda['Baths'].quantile(0.75) + 1.5*IQR_Baths
Lower_OutlierLimit_Baths = eda['Baths'].quantile(0.25) - 1.5*IQR_Baths

print(Upper_OutlierLimit_Baths)
print(Lower_OutlierLimit_Baths)
```

```
3.875
0.875
```

In [55]:
```
OutlierValues_Baths = eda[(eda['Baths']>=Upper_OutlierLimit_Baths) | (e
da['Baths']<=Lower_OutlierLimit_Baths)]
```

In [56]:
```
OutlierValues_Baths
```

Out[56]:

	Name	Age	Gender	Education	Salary	AppraisedValue	Location	Landacres	Hous

	Name	Age	Gender	Education	Salary	AppraisedValue	Location	Landacres	Hous
11	Maria	51	F	Grad	122	1200.0	LongBeach	0.4116	4067

Handle Outliers

1. We could remove the outliers from the data if they are due to data entry or data processing errors
2. Based on business understanding you could also replace the outliers with mean or median
3. If there is a pattern of interest in the outliers then they could be handled separately. For example if the outliers are like in groups then treat both groups as two different groups and build individual model for both groups and then combine the output
4. Also the outliers can be capped with 5th or 95th percentile

We will be using capping method for imputation of the outlier in our data for variable Apparaised_value

In [57]:
```
eda['AppraisedValue'][11] = eda['AppraisedValue'].quantile(0.95)
```

```
C:\Users\Shivani\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

In [58]:
```
eda['AppraisedValue'][11]
```

Out[58]:
```
849.9999999999995
```

In [59]:
```
eda['Baths'][11] = eda['Baths'].quantile(0.95)
```

```
C:\Users\Shivani\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

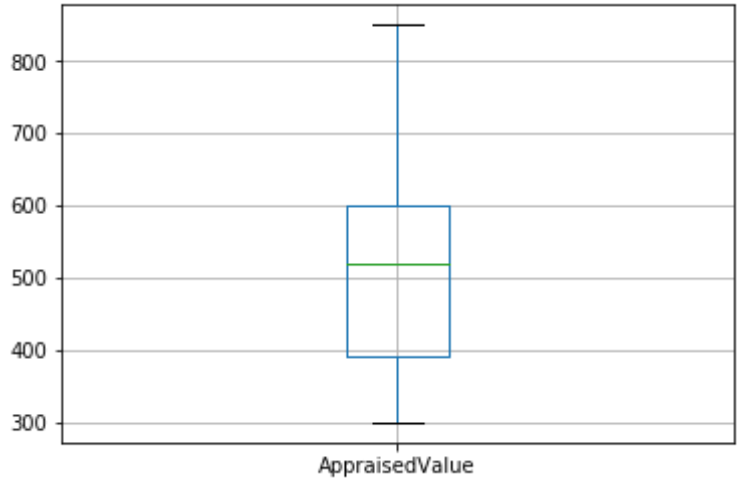
In [60]:
```
eda['Baths'][11]
```

Out[60]: 3.6499999999999995

BOX PLOTS AFTER CAPPING

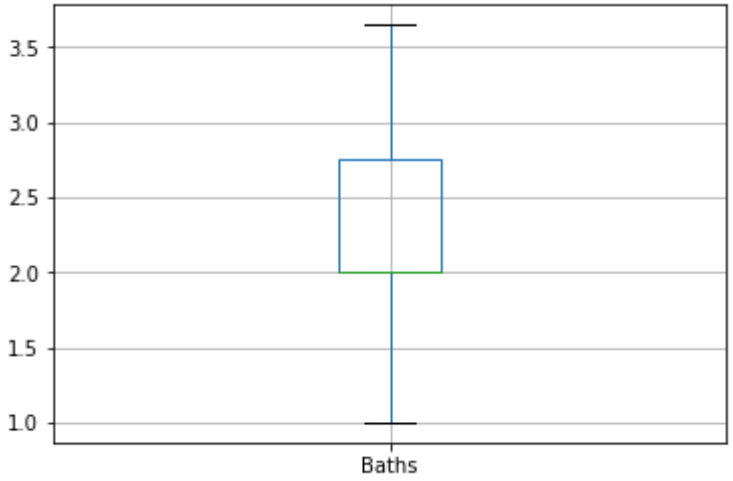
In [61]:
```
eda.boxplot(column = ['AppraisedValue'])
```

Out[61]: <matplotlib.axes._subplots.AxesSubplot at 0x1bc8cb6ef60>



In [62]:
```
eda.boxplot(column = ['Baths'])
```

Out[62]: <matplotlib.axes._subplots.AxesSubplot at 0x1bc8cb6ef98>



Step 6: Feature Engineering - Variable and Dummy Variable Creation

1. Variable creation is a process to generate a new variables / features based on existing variable(s)
2. Dummy coding provides one way of using categorical predictor variables in various kinds of estimation models (see also effect coding), such as, linear regression. Dummy coding uses only ones and zeros to convey all of the necessary information on group membership.
3. Below is an example of variable creations (Yellow columns are original variables and the columns in blue are variables created from them)

In [63]:
```
from IPython.display import Image
Image(filename='DummyVariables.jpg')
```

Out[63]:

ID	Gender	Date	Day	Month	Year	Dummy_Male	Dummy_Female
1	Male	10 May 2016	10	5	2016	1	0
2	Female	15 July 2016	15	7	2016	0	1
3	Male	01 June 2016	1	6	2016	1	0
4	Male	04 January 2016	4	1	2016	1	0
5	Female	27 March 2016	27	3	2016	0	1

Since, Location, Gender, Education are categorical variables, we need to convert them into dummy variables so that we will be able to use them as a predictor

In [64]:
```
obj = eda.dtypes == np.object
print(obj)
```

```
Name              True
Age              False
Gender            True
Education         True
Salary           False
AppraisedValue   False
Location          True
Landacres        False
HouseSizesqrft   False
Rooms            False
Baths            False
Garage           False
dtype: bool
```

In [65]:
```
eda.columns[obj]
```

Out[65]:
```
Index(['Name', 'Gender', 'Education', 'Location'], dtype='object')
```

In [66]:
```
del eda['Name']
```

In [67]:
```
obj = eda.dtypes == np.object
```

```
print(obj)
```

```
Age                False
Gender              True
Education           True
Salary             False
AppraisedValue     False
Location            True
Landacres          False
HouseSizesqrft     False
Rooms              False
Baths              False
Garage             False
dtype: bool
```

In [68]:
```
eda.columns[obj]
```

Out[68]:
```
Index(['Gender', 'Education', 'Location'], dtype='object')
```

In [69]:
```
dummydf = pd.DataFrame()

for i in eda.columns[obj]:
    dummy=pd.get_dummies(eda[i], drop_first=True)
    #"drop_first" drops the first category in order to avoid multicolli
nearity problem
    #prefix is used to add a certain prefix to all the dummy variables
 created for any particular categorial variable

    dummydf=pd.concat([dummydf, dummy], axis=1) # Concatenating Columns
    #"pd.concat" combines all the dummy columns for all the categorical
 variables

print(dummydf)
```

```
    M  PostGrad  LongBeach  Roslyn
0   1         0          0       0
1   0         1          0       0
2   0         1          0       0
3   0         0          1       0
4   1         0          1       0
```

```
5   1          1          0        0

6   1          0          0        0
7   1          0          0        1
8   0          1          0        1
9   1          1          0        1
10  1          0          1        0
11  0          0          1        0
12  0          1          0        1
13  0          0          0        1
14  1          0          1        0
```

In [70]:
```
eda1=pd.concat([eda,dummydf], axis=1)
```

In [71]:
```
eda1
```

Out[71]:

	Age	Gender	Education	Salary	AppraisedValue	Location	Landacres	HouseSizes
0	25	M	Grad	50	700.0	GlenCove	0.2297	2448
1	52	F	PostGrad	95	364.0	GlenCove	0.2192	1942
2	26	F	PostGrad	65	600.0	GlenCove	0.1630	2073
3	45	F	Grad	99	548.4	LongBeach	0.4608	2707
4	42	M	Grad	77	405.9	LongBeach	0.2549	2042
5	62	M	PostGrad	118	374.1	GlenCove	0.2290	2089
6	51	M	Grad	101	600.0	GlenCove	0.1714	1344
7	43	M	Grad	108	299.0	Roslyn	0.1750	1120
8	24	F	PostGrad	51	471.0	Roslyn	0.2130	1817
9	25	M	PostGrad	68	510.7	Roslyn	0.1377	2496
10	41	M	Grad	86	517.7	LongBeach	0.2497	1615
11	51	F	Grad	122	850.0	LongBeach	0.4116	4067

	Age	Gender	Education	Salary	AppraisedValue	Location	Landacres	HouseSizes
12	49	F	PostGrad	112	700.0	Roslyn	0.3372	3130
13	32	F	Grad	85	374.8	Roslyn	0.1503	1423
14	37	M	Grad	90	543.0	LongBeach	0.2348	1799

In [72]:
```
eda1.drop(['Gender', 'Education', 'Location'], axis=1, inplace=True)
```

In [73]:
```
eda2 = pd.get_dummies(eda, drop_first=True)
```

In [74]:
```
eda2
```

Out[74]:

	Age	Salary	AppraisedValue	Landacres	HouseSizesqrft	Rooms	Baths	Garage	Ger
0	25	50	700.0	0.2297	2448	8.0	3.50	2	1
1	52	95	364.0	0.2192	1942	7.0	2.50	1	0
2	26	65	600.0	0.1630	2073	7.0	3.00	2	0
3	45	99	548.4	0.4608	2707	8.0	2.50	1	0
4	42	77	405.9	0.2549	2042	7.0	1.50	1	1
5	62	118	374.1	0.2290	2089	7.0	2.00	0	1
6	51	101	600.0	0.1714	1344	8.0	1.00	0	1
7	43	108	299.0	0.1750	1120	5.0	1.50	0	1
8	24	51	471.0	0.2130	1817	6.0	2.00	0	0
9	25	68	510.7	0.1377	2496	7.0	2.00	1	1
10	41	86	517.7	0.2497	1615	7.0	2.00	1	1
11	51	122	850.0	0.4116	4067	9.0	3.65	1	0

	Age	Salary	AppraisedValue	Landacres	HouseSizesqrft	Rooms	Baths	Garage	Ger
12	49	112	700.0	0.3372	3130	8.0	3.00	1	0
13	32	85	374.8	0.1503	1423	7.0	2.00	0	0
14	37	90	543.0	0.2348	1799	6.0	2.50	1	1

In the table above, we have dropped the first column for each dummy variable as it can easily be denoted from '0'. The thumb rule of no. of dummy variables to be created is (No. of Unique Values in a Categorical Variable - 1)

Step 7: Feature Engineering - Variable Transformation

1. In data modelling, transformation refers to the replacement of a variable by a function. For instance, replacing a variable x by the square / cube root or logarithm x is a transformation
2. When do we transform?

```
 – When we want to change the scale of a variable or stand
ardize the values of a variable for better understanding.
 While this transformation is a must if you have data in d
ifferent scales

 – This transformation does not change the shape of the va
riable distribution
 – Existence of a linear relationship between variables is
 easier to comprehend compared to a non-linear or curved r
elation.
 – Variables can be transformed by applying functions like
 log, square, cube etc.
```

3. These transformations help in reducing skewness. For right skewed distribution, we take square / cube root or logarithm of variable and for left skewed, we take square.

Transforming a variable involves using a mathematical operation to change its measurement scale. In regression, a transformation to achieve linearity is a special kind of nonlinear transformation. It is a nonlinear transformation that increases the linear relationship between two variables

Methods of Transforming Variables to Achieve Linearity:

There are many ways to transform variables to achieve linearity for regression analysis. Some common methods are summarized below.

In [75]:
```
from IPython.display import Image
Image('Transformation.jpg')
```

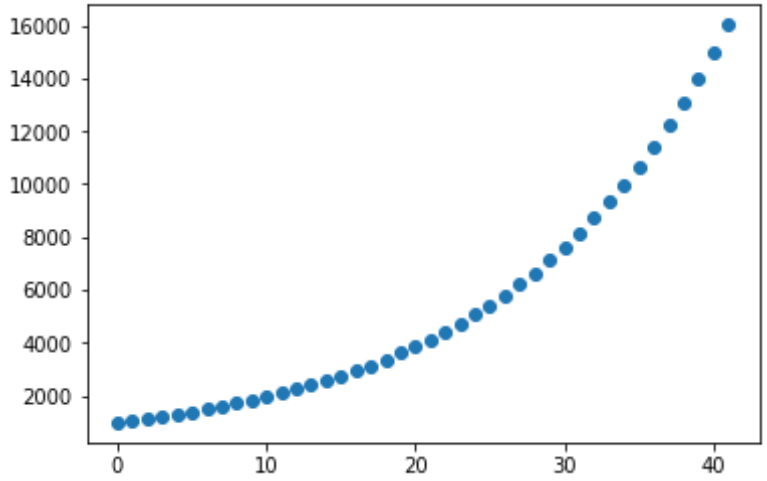
Out[75]:

Method	Transformation(s)	Regression equation	Predicted value ($\hat{y}$)
Standard linear regression	None	$y = b_0 + b_1x$	$\hat{y} = b_0 + b_1x$
Exponential model	Dependent variable = log(y)	$\log(y) = b_0 + b_1x$	$\hat{y} = 10^{b_0 + b_1x}$
Quadratic model	Dependent variable = sqrt(y)	$\text{sqrt}(y) = b_0 + b_1x$	$\hat{y} = (b_0 + b_1x)^2$
Reciprocal model	Dependent variable = 1/y	$1/y = b_0 + b_1x$	$\hat{y} = 1 / (b_0 + b_1x)$
Logarithmic model	Independent variable = log(x)	$y= b_0 + b_1\log(x)$	$\hat{y} = b_0 + b_1\log(x)$
Power model	Dependent variable = log(y) Independent variable = log(x)	$\log(y)= b_0 + b_1\log(x)$	$\hat{y} = 10^{b_0 + b_1\log(x)}$

In [76]:
```
transformation = pd.read_excel('Transformation Example.xlsx')
```

In [77]:
```
plt.scatter(transformation['Year'], transformation['Investment Value'])
```

Out[77]:
```
<matplotlib.collections.PathCollection at 0x1bc8ccb87b8>
```



We observe that Investment value is increasing exponentially. So, var X and Y are exponentially related. To make them linearly dependent on each other, we need to transform the variable(Investment Value)

In [79]:
```
transformation['Investment Value']= np.log(transformation['Investment Value'])
```