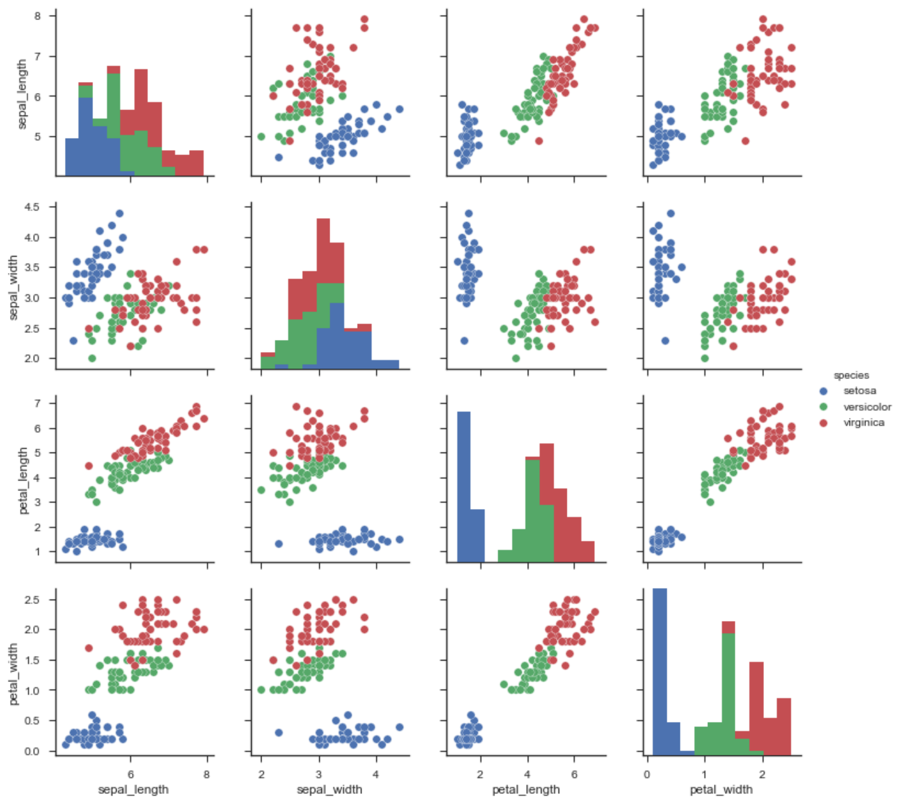


> ***Caveat***: Over the last few years, visualization in Python has made major strides forward. It is easy to find some high powered and beautiful visualization libraries for Python on the Internet. We will not look at all of them YET. We need to focus our efforts on learning the basics and the most common data visualization libraries (Matplotlib as the base and Seaborn as the next level up). In time, we will cover many more visualization libraries and techniques.

1 Data Visualization - Part 2 - Seaborn

In our last meet-up, we introduced the Matplotlib library for plotting. While Matplotlib library is robust and allows us to do quite a lot, the plots are not refined and would not be anyone's first choice for publishing. This is where Seaborn comes into the picture.

> *Seaborn is a Python data visualization library based upon Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics* - Seaborn Documentation

Additional Reading Resources:

1. https://towardsdatascience.com/seaborn-lets-make-plotting-fun-4951b89a0c07 (https://towardsdatascience.com/seaborn-lets-make-plotting-fun-4951b89a0c07)
2. https://medium.com/@neuralnets/statistical-data-visualization-series-with-python-and-seaborn-for-data-science-5a73b128851d (https://medium.com/@neuralnets/statistical-data-visualization-series-with-python-and-seaborn-for-data-science-5a73b128851d)

If you are not already a subscriber or follower of ***Medium e-Magazine*** [https://medium.com/] (https://medium.com/%5D), you will want to not only subscribe, but also receive their alerts. They have several sections on

1. data science [https://towardsdatascience.com/] (https://towardsdatascience.com/%5D),
2. data visualization [https://towardsdatascience.com/data-visualization/home] (https://towardsdatascience.com/data-visualization/home%5D), and
3. machine learning [https://towardsdatascience.com/machine-learning/home] (https://towardsdatascience.com/machine-learning/home%5D)

They also cover

1. Python programming [https://towardsdatascience.com/search?q=Python] (https://towardsdatascience.com/search?q=Python%5D)
2. Juypter Notebooks [https://towardsdatascience.com/search?q=Jupyter%20Notebook] (https://towardsdatascience.com/search?q=Jupyter%20Notebook%5D)

as well as other related topics.

1.1 Import Data

For our exercises in this notebook, we are going to use the **California Housing Dataset** from Kaggle [https://www.kaggle.com/camnugent/california-housing-prices#] (https://www.kaggle.com/camnugent/california-housing-prices#%5D), which is located in the `data/` sub-directory.

Your first step is to

1. import the dataset into a DataFrame, which we will call 'df'
2. examine the shape of the DataFrame
3. examine the column names and the first five (5) rows of the DataFrame

In [1]:
```
import pandas as pd

df = pd.read_csv('data/housing.csv')
print('DataFrame Shape: {}'.format(df.shape), '\n')
print('Columns: {}'.format(df.columns))
display(df.head())
```
executed in 523ms, finished 17:36:44 2019-12-21

```
DataFrame Shape: (20640, 10)

Columns: Index(['longitude', 'latitude', 'housing_median_age', 'total_roo
ms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value', 'ocean_proximity'],
      dtype='object')
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	m
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	

2 Seaborn

2.1 Load the necessary libraries

Since Seaborn is based on Matplotlib, we first need to import Matplotlib, then we need to import Seaborn, and because we are working in Jupyter Notebooks, we finish off the libraries load with `%matplotlib inline` so we can see our work.

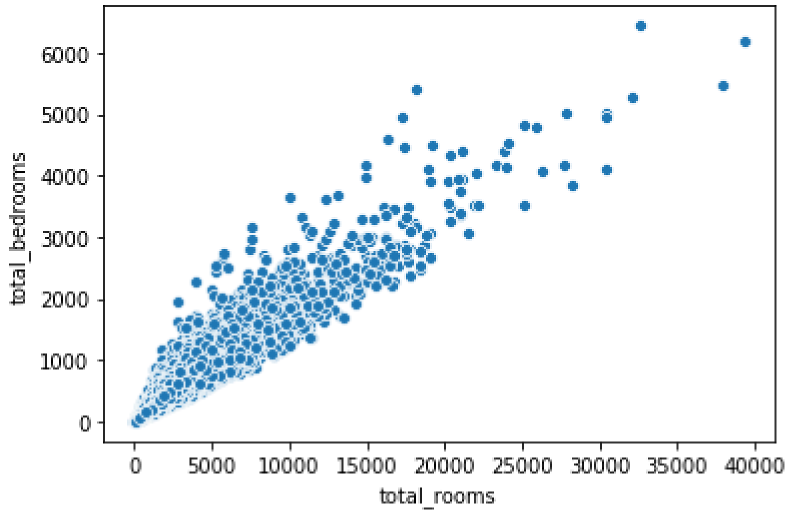
(**Note:** If when you loaded Seaborn you received a pink box with warnings, you can run the warnings block to suppress those warnings in the future of this notebook. Receiving the warnings will not affect anything more than the appearance of your notebook.)

In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns


# Suppress the warnings we may receive in this example about an update
import warnings
warnings.filterwarnings("ignore")


%matplotlib inline
```
executed in 381ms, finished 17:36:44 2019-12-21

2.2 Scatter Plot

The scatter plot is used when we want to show the relationship between two features or a feature and a label. The *basic* command for Seaborn is simple.

In [3]:
```
sns.scatterplot(data=df, x='total_rooms', y='total_bedrooms')
plt.show()
```
executed in 253ms, finished 17:36:44 2019-12-21



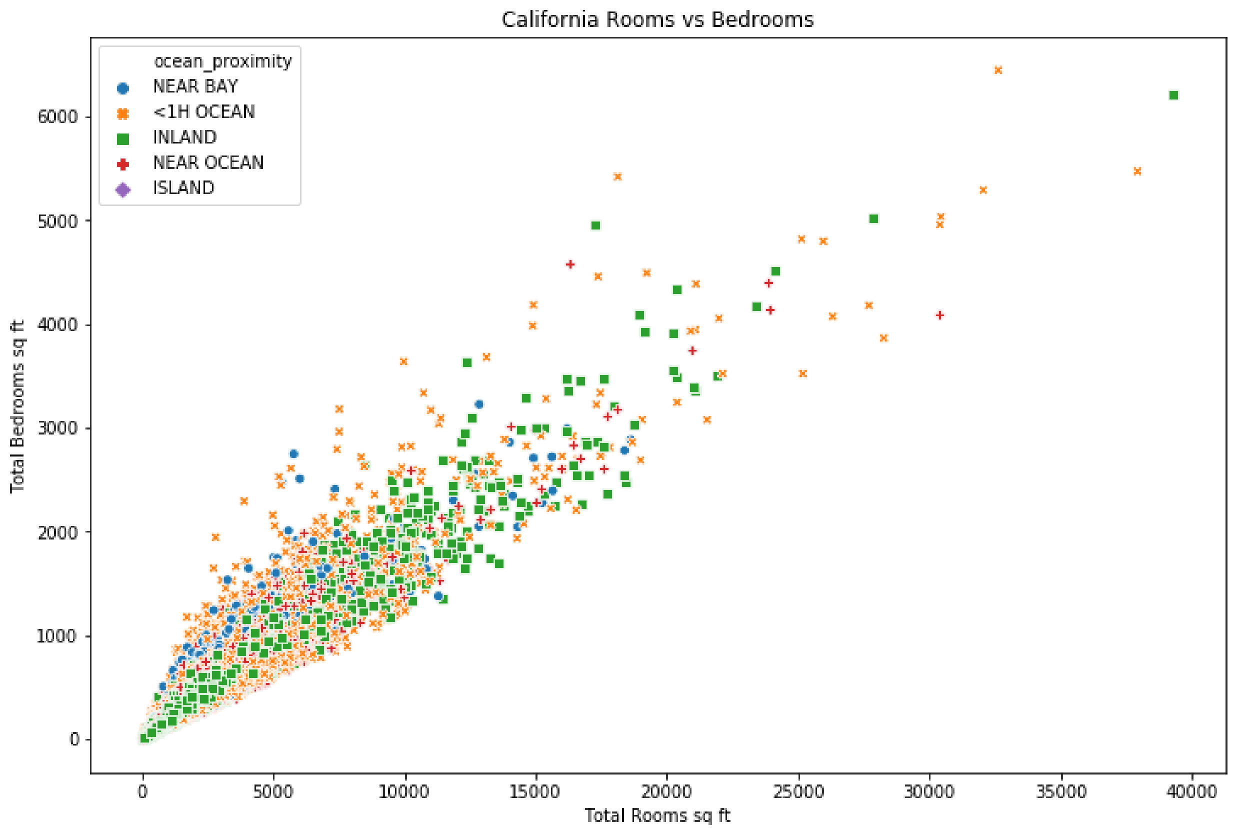
2.2.1 A Few Aesthetics

Plotting a basic scatter plot is simple enough, but let's add in some additional aestheics and information to our plot in order to really bring out the power of Seaborn.

1. Let's increase the size of the plot so it is a bit easier to see.
2. Let's make the size of each dot based upon 'median_house_value' (bringing in another data value)
3. Color the dots based upon 'ocean_proximity'
4. And base the style of the markers based upon 'ocean_proximity'
5. Add main and axes titles

In [4]:
```
plt.figure(figsize=(12,8)) #Sets a new figure size away from the defaul

sns.scatterplot(data = df, # Calls the data source
                x = 'total_rooms', # Declares x
                y = 'total_bedrooms',# Declares y
                hue = 'ocean_proximity',# Declares another variable by c
                style = 'ocean_proximity')# Declares another variable by

# Add some clarifying features
plt.title('California Rooms vs Bedrooms')
plt.xlabel('Total Rooms sq ft')
plt.ylabel('Total Bedrooms sq ft')
plt.show()
```

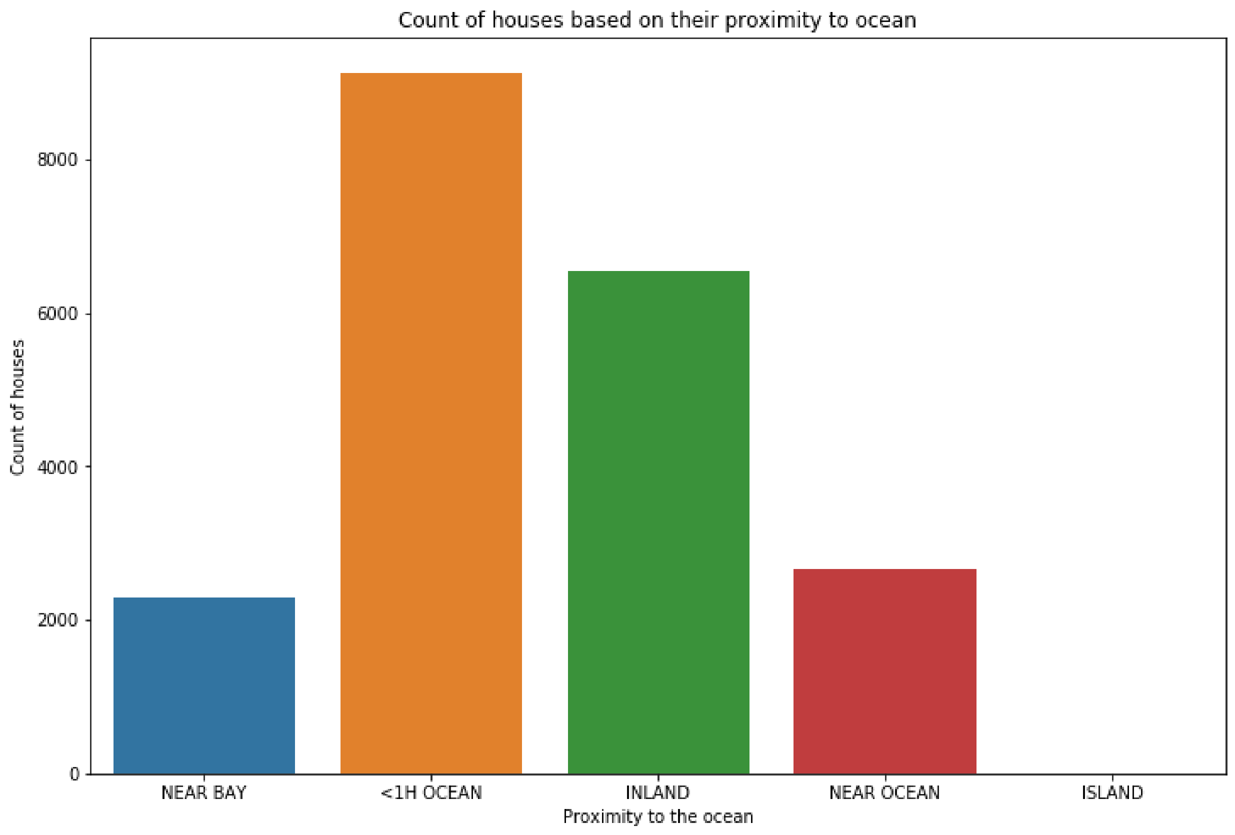
executed in 811ms, finished 17:36:45 2019-12-21



2.3 Bar Plot (or Count Plot)

Bar plots, sometimes called a Count plot, automatically count the data points based upon a certain categorical column and present the data as a bar plot. Bar or Count plots are used when you are totalling the count of observations within a category or condition.

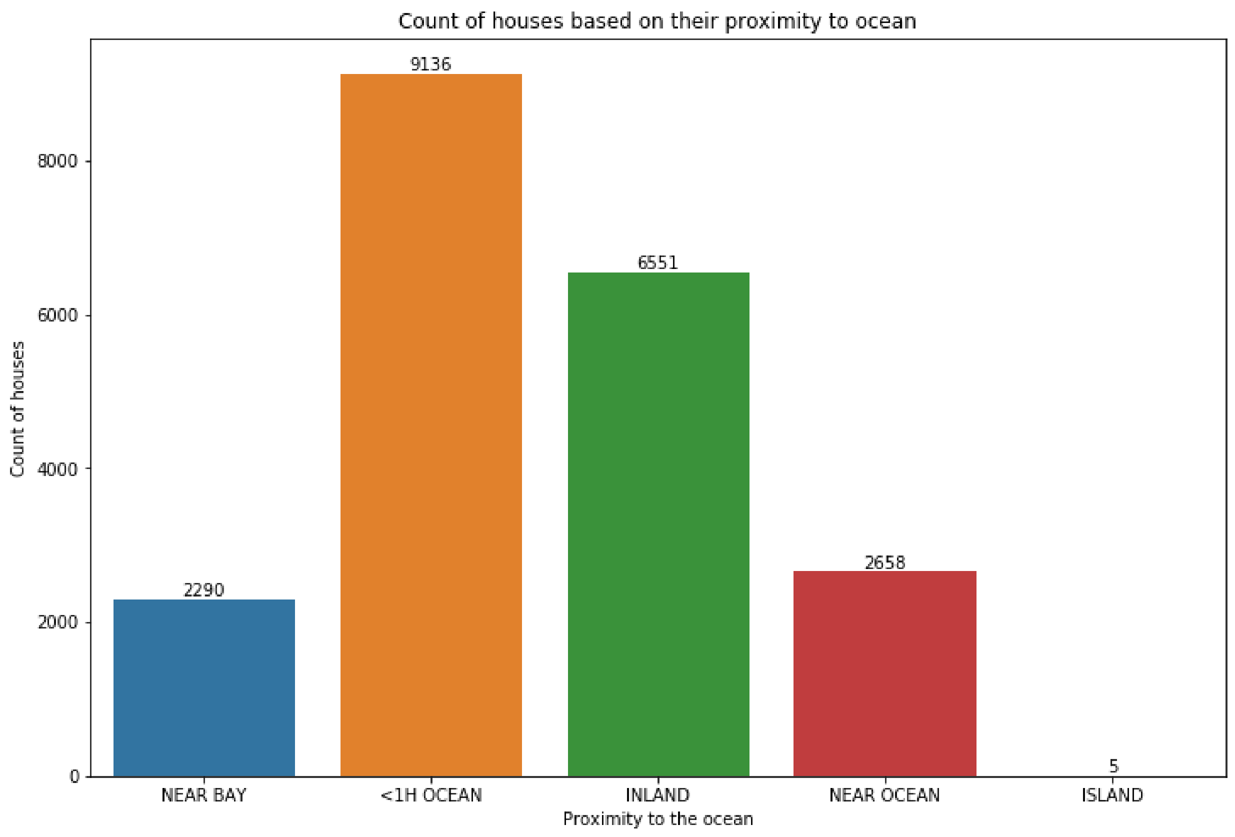
In [5]:
```
#Basic count plot (bar chart) with external labels

plt.figure(figsize = (12, 8)) #Establish the size of the graphic

sns.countplot(data = df, x = 'ocean_proximity')# Actual line that draws


#Adding some clarifying elements
plt.title("Count of houses based on their proximity to ocean")
plt.xlabel("Proximity to the ocean")
plt.ylabel("Count of houses")
plt.show()
```
executed in 135ms, finished 17:36:45 2019-12-21



Simple to do, as you can see. But what if we want to add the count to the ends of each of the bars (that is always a good idea for clarity in our graphics). In the next block of code, we will do that.

In [6]:
```
plt.figure(figsize = (12, 8))# Adjust the figure size

#Draw the actual plot and assign that action to the variable 'ocean_plo
ocean_plot = sns.countplot(data = df, x = 'ocean_proximity')

'''
The next FOR loop establishes the location for placing the counts, then
the end of each bar respectively.

'''

for p in ocean_plot.patches:# patches are a library of commands for aes
    ocean_plot.annotate(p.get_height(),
                        (p.get_x() + p.get_width() / 2.0,
                         p.get_height()), #Setting width of bars so the
                        ha = 'center', #horizontal axis
                        va = 'center', # vertical axis
                        xytext = (0, 5), # placing the text
                        textcoords = 'offset points')


    # And, of course, we add our clarifying features (labels)
plt.title("Count of houses based on their proximity to ocean")
plt.xlabel("Proximity to the ocean")
plt.ylabel("Count of houses")
plt.show()
```

executed in 145ms, finished 17:36:46 2019-12-21



So we don't rush over the code to add the count to the ends of each column, let's take a moment and run just the FOR loop, printing out the 'p' variable to see what is 'p'. Remember that 'p' will be used in the following annotation set-ip

> ocean_plot.annotate(p.get_height(), (p.get_x() + p.get_width() / 2.0, p.get_height()), ha = 'center', va = 'center', xytext = (0, 5), textcoords = 'offset points')

In [7]:
```
for p in ocean_plot.patches:
    print(p)
```
executed in 3ms, finished 17:36:46 2019-12-21

```
Rectangle(xy=(-0.4, 0), width=0.8, height=2290, angle=0)
Rectangle(xy=(0.6, 0), width=0.8, height=9136, angle=0)
Rectangle(xy=(1.6, 0), width=0.8, height=6551, angle=0)
Rectangle(xy=(2.6, 0), width=0.8, height=2658, angle=0)
Rectangle(xy=(3.6, 0), width=0.8, height=5, angle=0)
```

2.4 Histograms

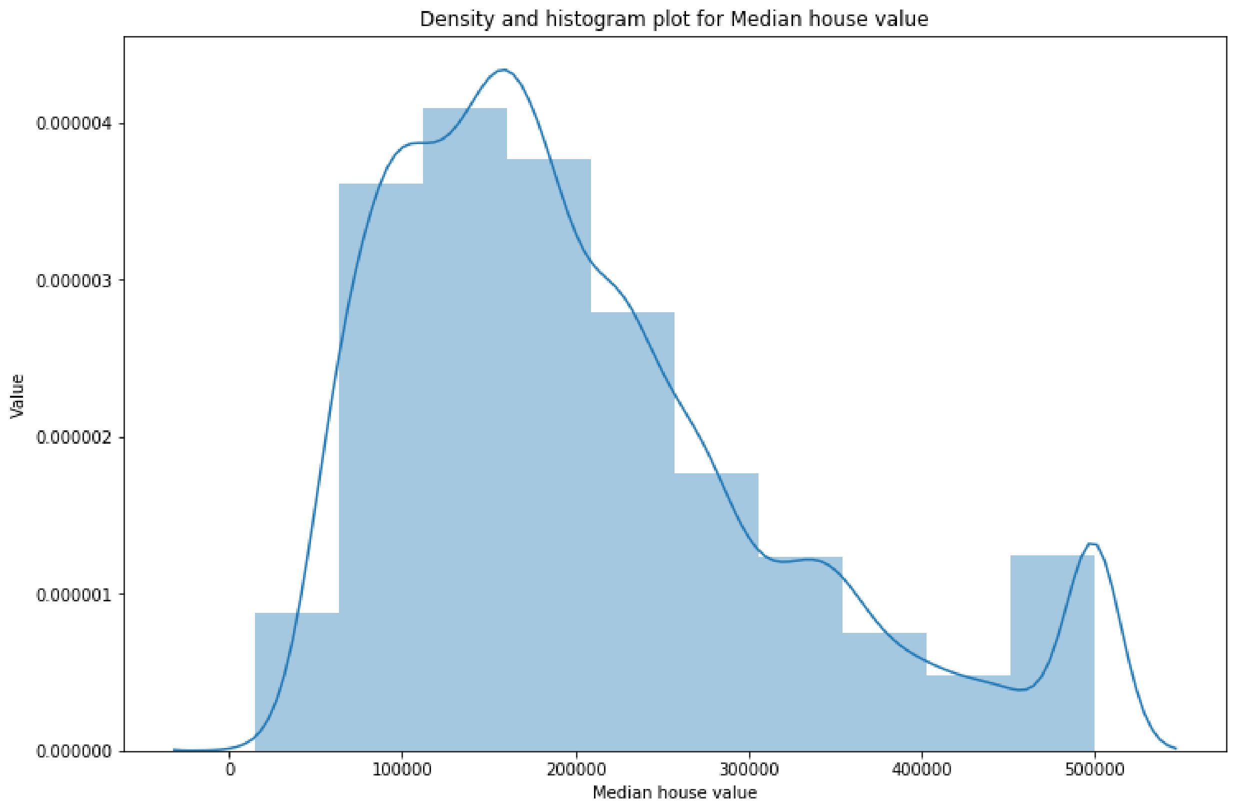
Histograms are an effective way to show continuous points of data and see how they are distributed. The `dist` plot in Seaborn produces both a histogram as well as a density line on that plot based on the data.

Since Histograms require us to define the number of total sorting bins, we defined a total number of 10 bins for `median_house_value` .

In [9]:

```python
#Set the figsize to a 12x8 so it is clearly visible
plt.figure(figsize = (12, 8))

#The actual line of code that draws the historgram
sns.distplot(a = df['median_house_value'],
             bins = 10,
             hist = True,
             kde = True)

#Adding some clarifying features (labels)
plt.title("Density and histogram plot for Median house value")
plt.xlabel("Median house value")
plt.ylabel("Value")
plt.show()
```

executed in 184ms, finished 17:36:58 2019-12-21

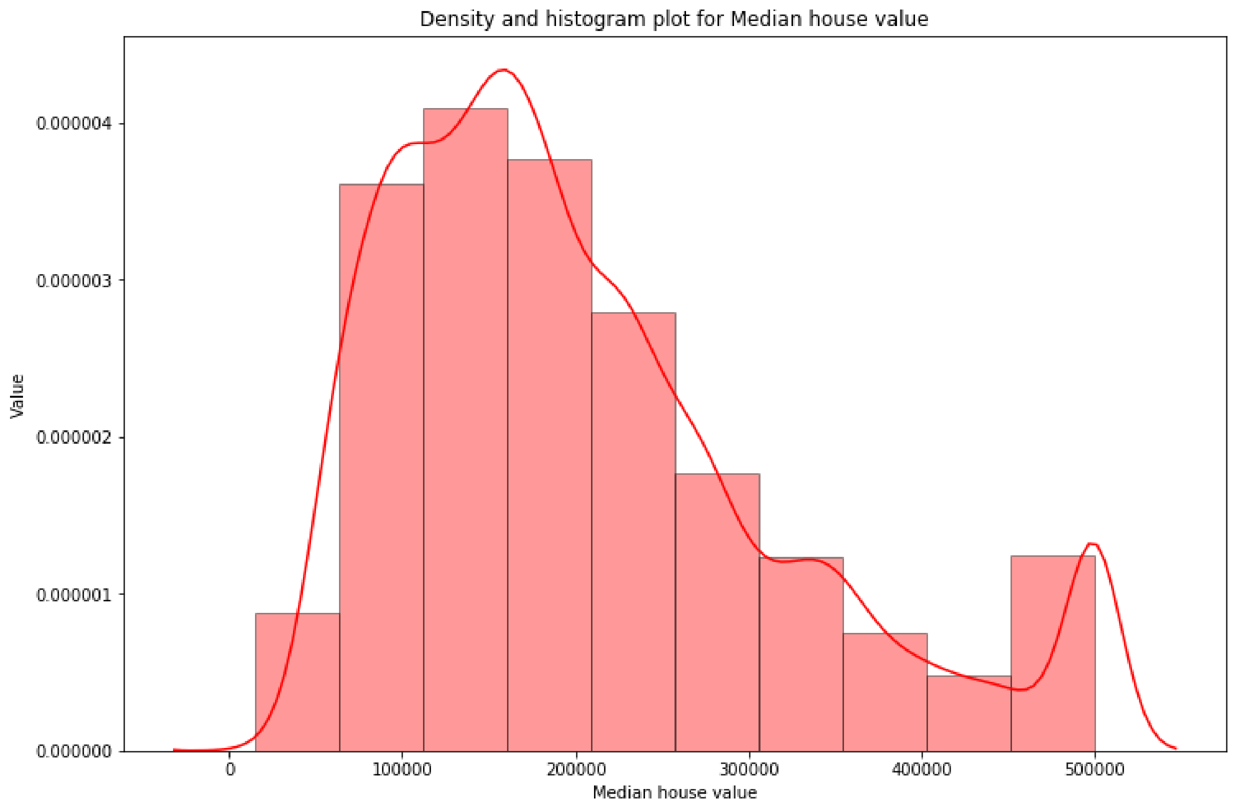


Quite simple. But let's add a little color and pleasing aestheics to our Histogram.

1. Change the color of our Histgram to 'red'
2. Add borders to each bin

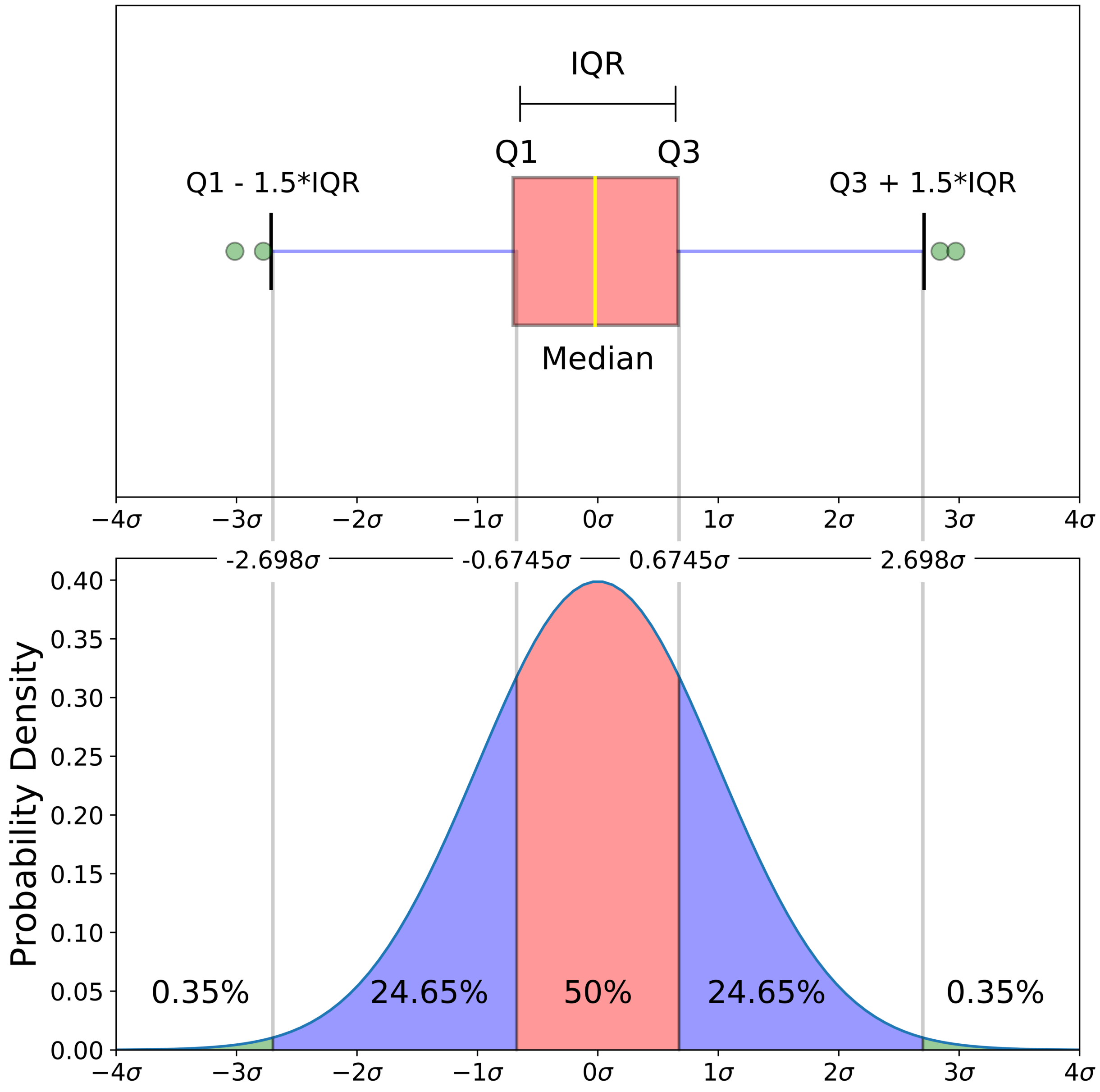
In [58]:
```python
#Set the figsize to a 12x8 so it is clearly visible
plt.figure(figsize = (12, 8))

#The actual line of code that draws the historgram
sns.distplot(a = df['median_house_value'],
             bins = 10,
             hist = True,
             color='r',
             hist_kws=dict(edgecolor='k', linewidth=1))

#Adding some clarifying features (labels)
plt.title("Density and histogram plot for Median house value")
plt.xlabel("Median house value")
plt.ylabel("Value")
plt.show()
```

executed in 171ms, finished 12:43:18 2019-12-16



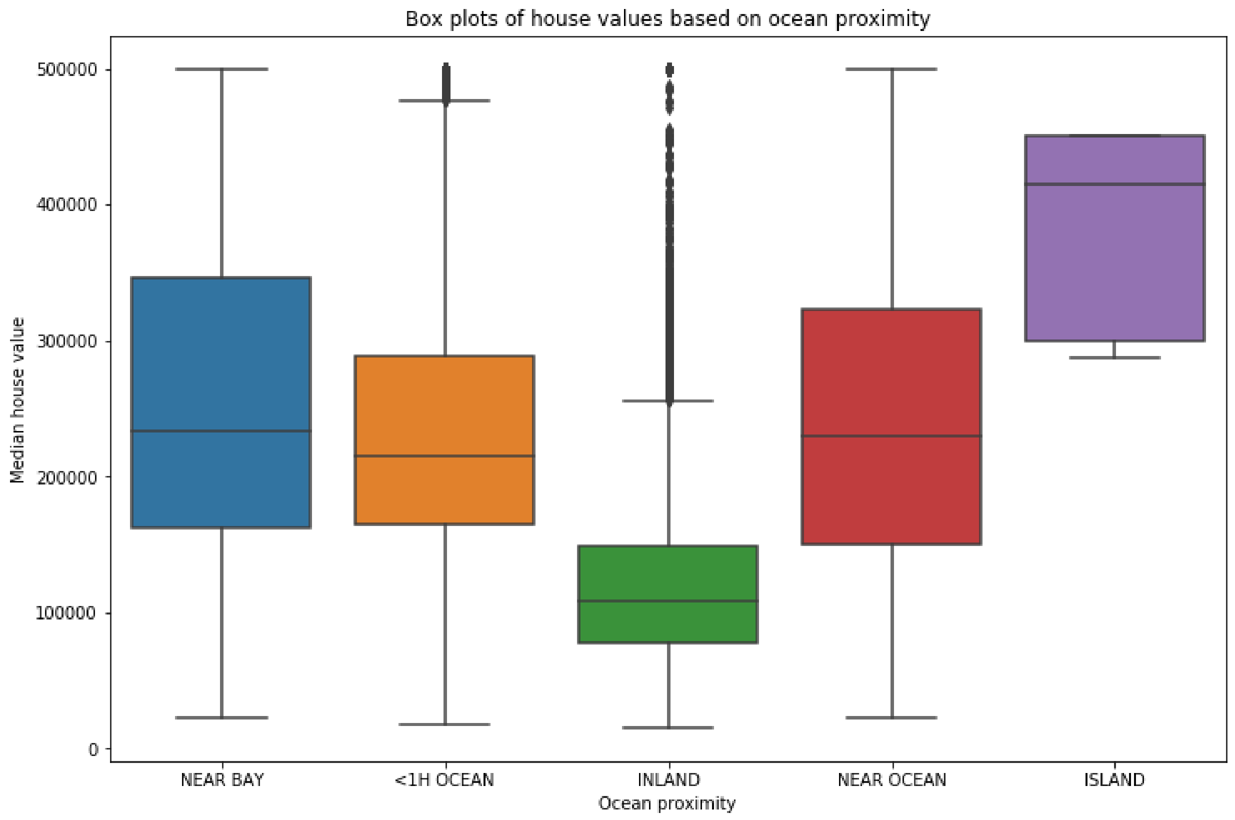
2.5 Boxplot

A boxplot is a standardized way of displaying the distribution of data based on a five number summary (“minimum”, first quartile (Q1), median, third quartile (Q3), and “maximum”). It can tell you about your outliers and what their values are. It can also tell you if your data is symmetrical, how tightly your data is grouped, and if and how your data is skewed.



In [59]:
```
plt.figure(figsize = (12, 8))


sns.boxplot(x = 'ocean_proximity',
            y = 'median_house_value',
            data = df,
              )


plt.title("Box plots of house values based on ocean proximity")
plt.xlabel("Ocean proximity")
plt.ylabel("Median house value")
plt.show()
```

executed in 208ms, finished 12:43:18 2019-12-16



2.6 Violin Plot

For most people, a Violin Plot is an obscure, strange looking, never used plot. I believe this would not be the case if Excel had violin plots as a part of the standard selection. But because of this, and their strange look, we rarely see these plots outside of more serious Data Science.

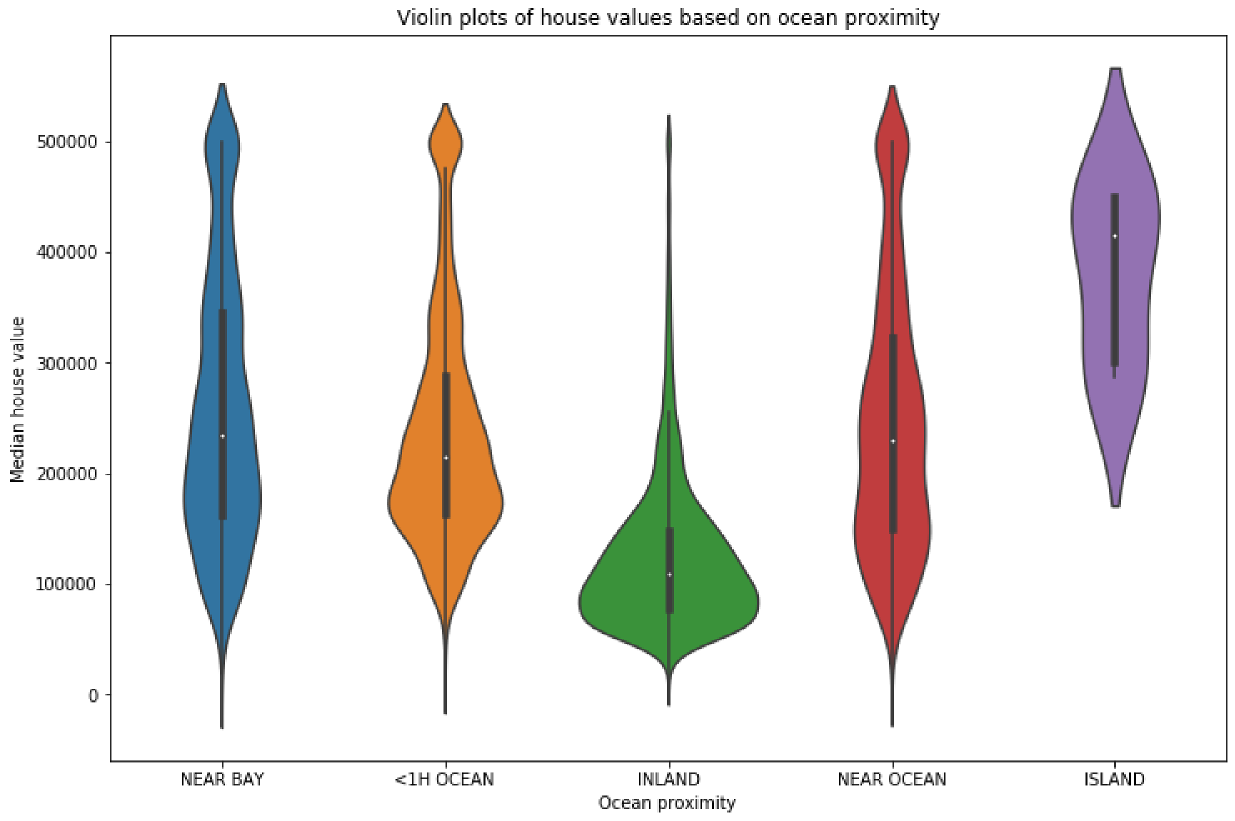
Violin plots are another representation of the distribution of a set of observations.

Let's look at how to draw a violin plot first, then we will discuss how to read it (which is quite simple). After that, we will discuss how easy it makes comparing values.

In [60]:

```python

plt.figure(figsize = (12, 8))# Set the figure size


sns.violinplot(data = df, #Set the data source
               x = 'ocean_proximity', #Set the x variable
               y = 'median_house_value', #Set the y variable
              )

#Add clarifying features
plt.title("Violin plots of house values based on ocean proximity")
plt.xlabel("Ocean proximity")
plt.ylabel("Median house value")
plt.show()
```

executed in 229ms, finished 12:43:18 2019-12-16



Before we move ahead, let’s talk about how to understand these plots.

Consider the green plot `INLAND` .

1. The black line that extends from zero to approximately 250000 is the 95% confidence interval.
2. The thick black block inside is the interquartile range meaning approximately 50% of all data lies in this range.
3. The width of the plot is based on the density of the data.

We can understand it as the histogram of this specific dataset with the black line as the x-axis that is completely smoothed out and turned 90 degrees.

2.7 Heatmap or Correlation Matrix

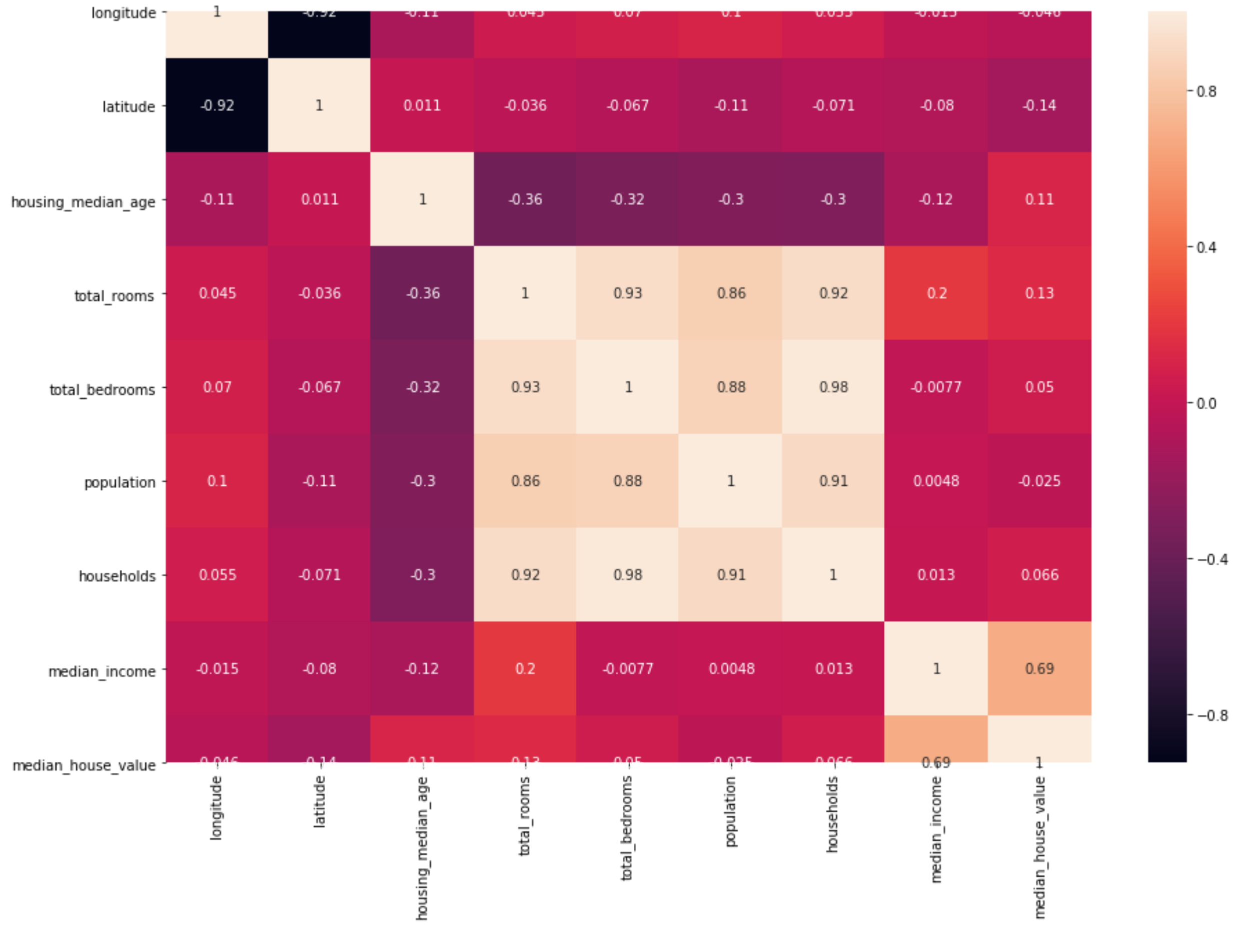
I have to admit that a Heatmap (or, more appropriately, a Correlation Matrix) is one of my favorite graphics. It is also my first go-to graphics for conducting an Exploratory Data Analysis (EDA) on data sets with a considerable number of quantitative categories.

A correlation matrix helps us see how all features and labels are related to one another and the level of dependance.

The Pandas DataFrame has a function called `corr()`, which generates a correlation matrix. When we input that Pandas correlation matrix to the Seaborn heatmap, we get a beautiful heatmap.

Setting `annot` as **True** ensures that the correlations are also defined with numbers.

In [61]:
```python

plt.figure(figsize = (15,10))#Set the figure Size


sns.heatmap(df.corr(), annot = True)#Note the .corr()
plt.show()
```
executed in 421ms, finished 12:43:18 2019-12-16



	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
longitude	1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
latitude	-0.92	1	0.011	-0.036	-0.067	-0.11	-0.071	-0.08	-0.14
housing_median_age	-0.11	0.011	1	-0.36	-0.32	-0.3	-0.3	-0.12	0.11
total_rooms	0.045	-0.036	-0.36	1	0.93	0.86	0.92	0.2	0.13
total_bedrooms	0.07	-0.067	-0.32	0.93	1	0.88	0.98	-0.0077	0.05
population	0.1	-0.11	-0.3	0.86	0.88	1	0.91	0.0048	-0.025
households	0.055	-0.071	-0.3	0.92	0.98	0.91	1	0.013	0.066
median_income	-0.015	-0.08	-0.12	0.2	-0.0077	0.0048	0.013	1	0.69
median_house_value	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.69	1

NOTE: *Do you see how the top and bottom rows in the correlation matrix are cut off? This was a* `matplotlib` *regression introduced in version 3.1.1, which has been fixed in version 3.1.2 (still forthcoming). For now the fix is to downgrade* `matplotlib` *to a prior version, unless you can live with this for a few more months until the fix is available.*

2.8 Joint Plot

A Joint Plot is a combination of scatter plot along with the density plots (histograms) for both features we're trying to plot.

The Seaborn's Joint Plot allows us to even plot a linear regression all by itself using `kind` as `reg`.

We defined the square dimensions using height as 8 and color as green.

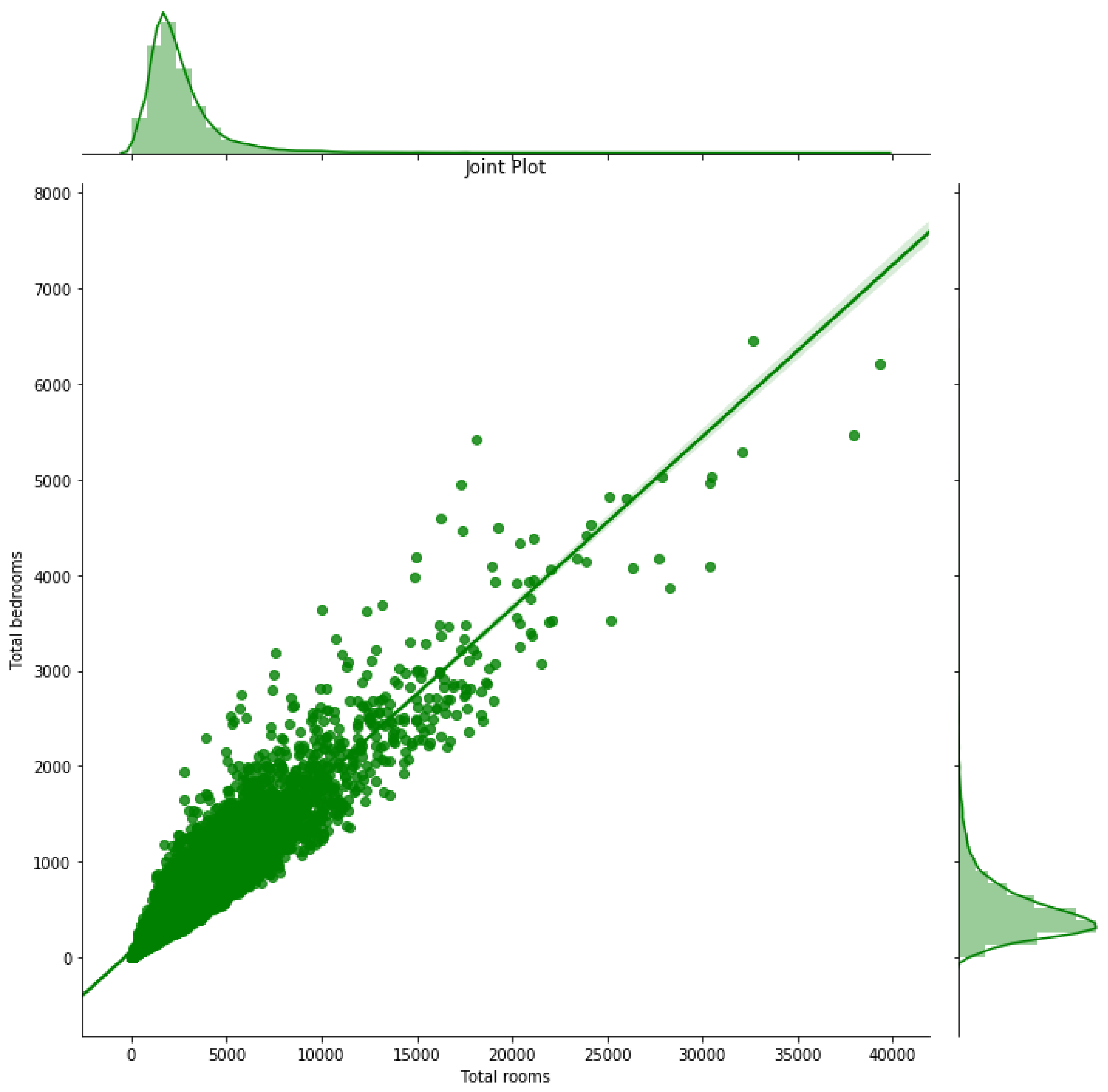
2.8.1 JointPlot Regression

In [63]:

```

plt.figure(figsize=(12,10))# Set the figure size larger than default

sns.jointplot(data=df, #Declare the dataset
              x = "total_rooms", #Declare the x value
              y = "total_bedrooms", #Declare the y value
              kind="reg", #kind = regression
              height = 10, #Set the height of the plot
              color = 'g',#Set the color as green
             )

#Add some clarifying aspects
plt.title('Joint Plot')
plt.xlabel("Total rooms")
plt.ylabel("Total bedrooms")
plt.show()
```

executed in 1.71s, finished 12:43:40 2019-12-16

```
<Figure size 864x720 with 0 Axes>
```



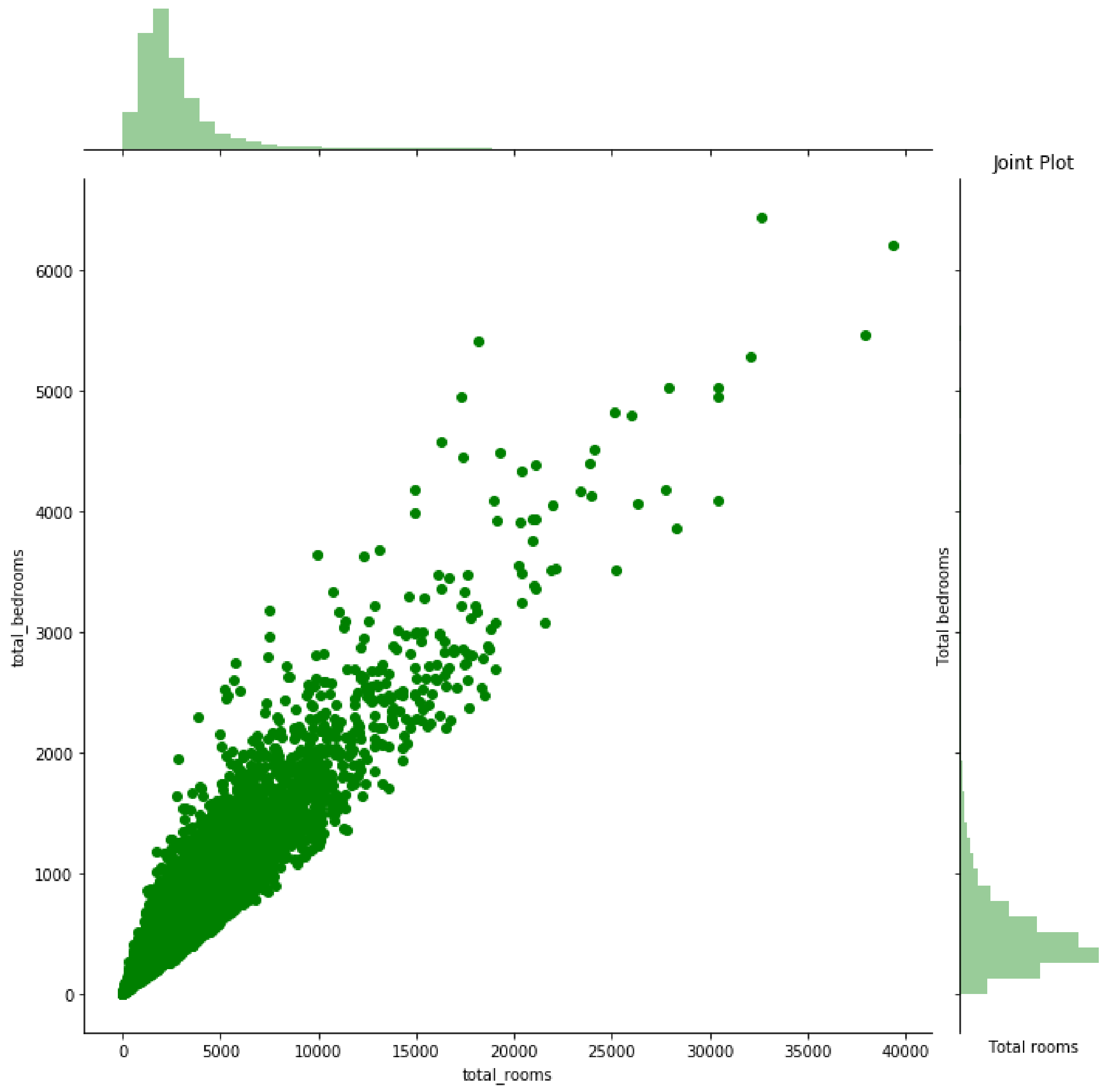
2.8.2 Joint Plot Scatter

In [64]:

```python
plt.figure(figsize=(12,10))# Set the figure size larger than default

sns.jointplot(data=df, #Declare the dataset
              x = "total_rooms", #Declare the x value
              y = "total_bedrooms", #Declare the y value
              kind="scatter", #kind = scatter
              height = 10, #Set the height of the plot
              color = 'g',#Set the color as green
             )

#Add some clarifying aspects
plt.title('Joint Plot')
plt.xlabel("Total rooms")
plt.ylabel("Total bedrooms")
plt.show()
```

executed in 923ms, finished 12:45:16 2019-12-16

```
<Figure size 864x720 with 0 Axes>
```



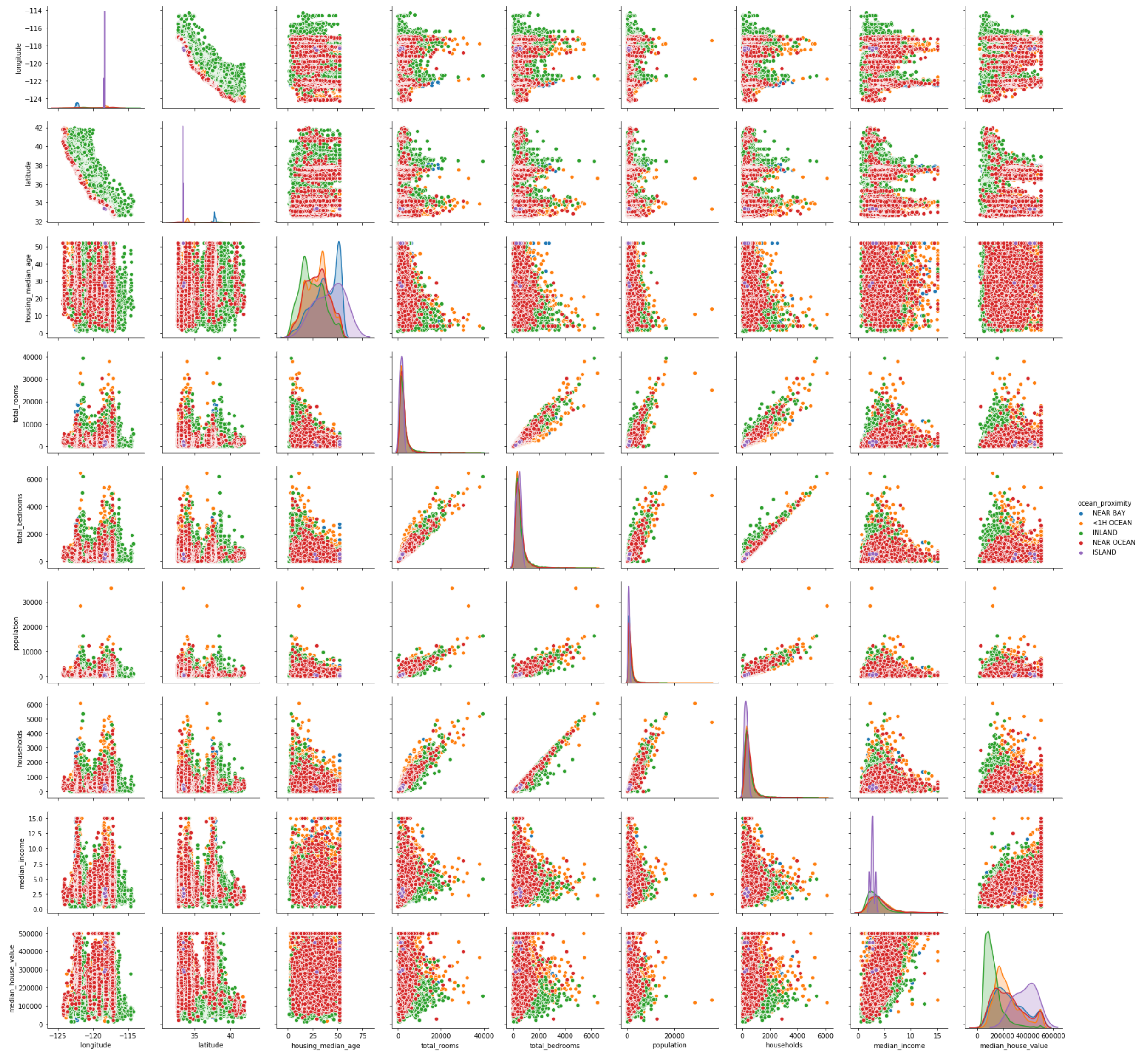
2.9 Pairplot

A Pairplot is another one of my immediate go-to plots for EDAs (Exploratory Data Analyses). A pair plot takes every numeric category in our dataset and plots that value against every other numeric value.

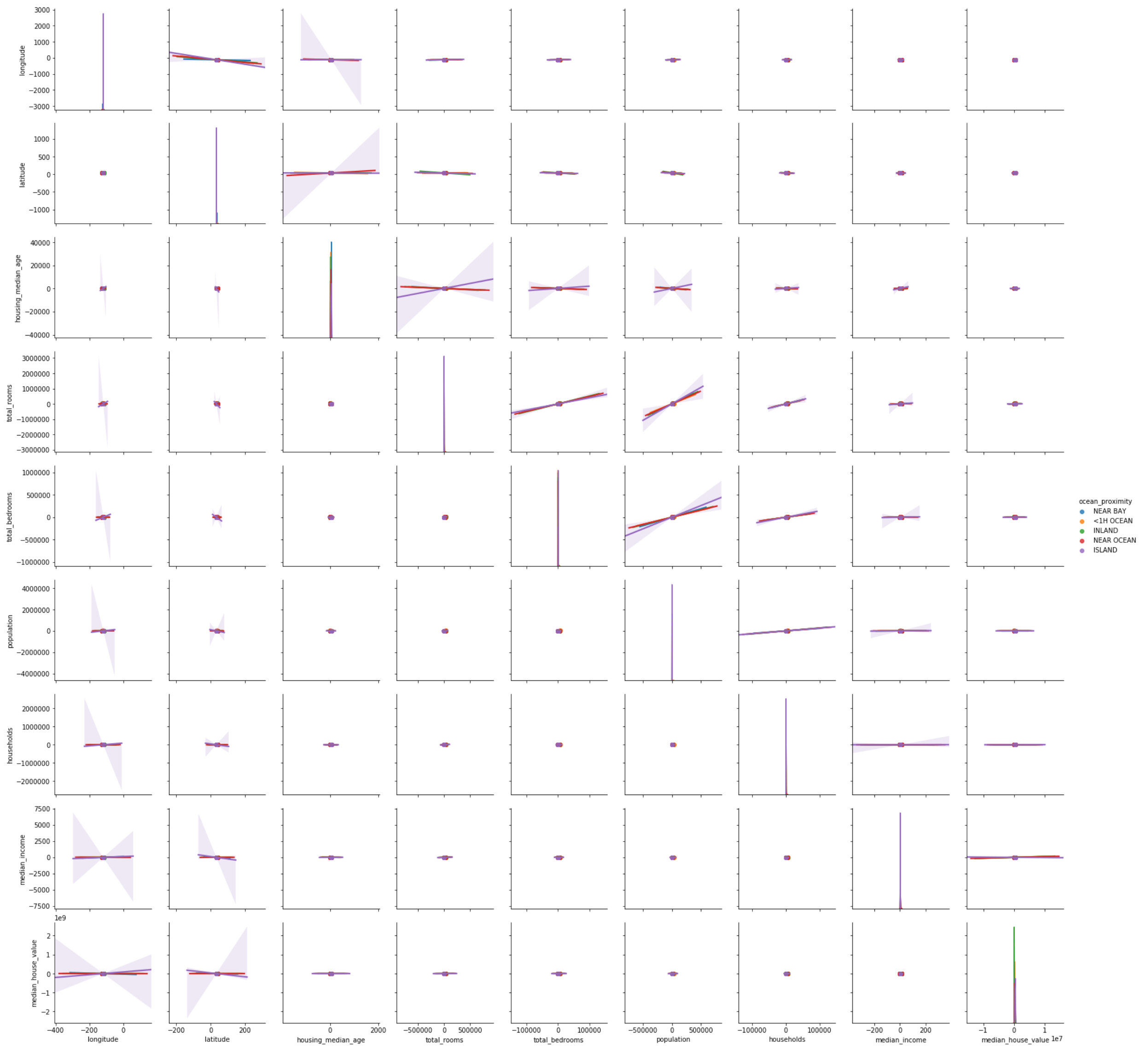
In one plot, we see trends and patterns quite easily.

In [40]:
```
sns.pairplot(df,
             hue = 'ocean_proximity',# Declares another variable by colo
             kind = 'scatter', #Scatter representation is the default. Y
             )
plt.show()
```

executed in 34.6s, finished 12:10:42 2019-12-16



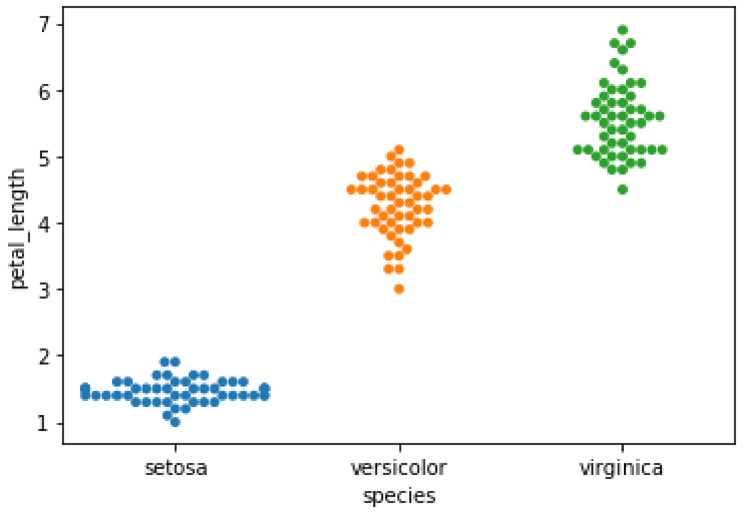
In [41]:
```
sns.pairplot(df,
             hue = 'ocean_proximity',# Declares another variable by colo
             kind = 'reg', #reg representation gives you the regression
             )
plt.show()
```
executed in 2m 20s, finished 12:13:03 2019-12-16



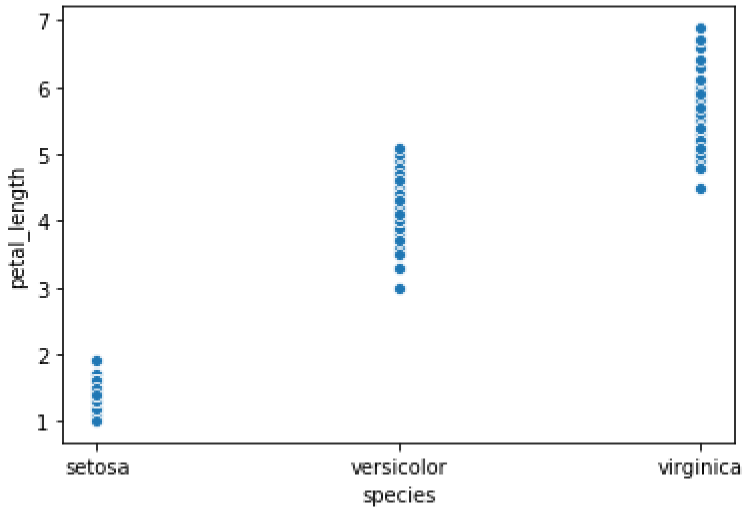
2.10 Swarmplot

A Swarmplot is a scatter plot where the *x* axis is a categorical value.

In [15]:
```python
# Load a built in Dataset
iris = sns.load_dataset('iris')

sns.swarmplot(data = iris,
                    x = 'species',
                    y = 'petal_length',
                   )
plt.show()
```
executed in 191ms, finished 11:03:11 2019-12-16



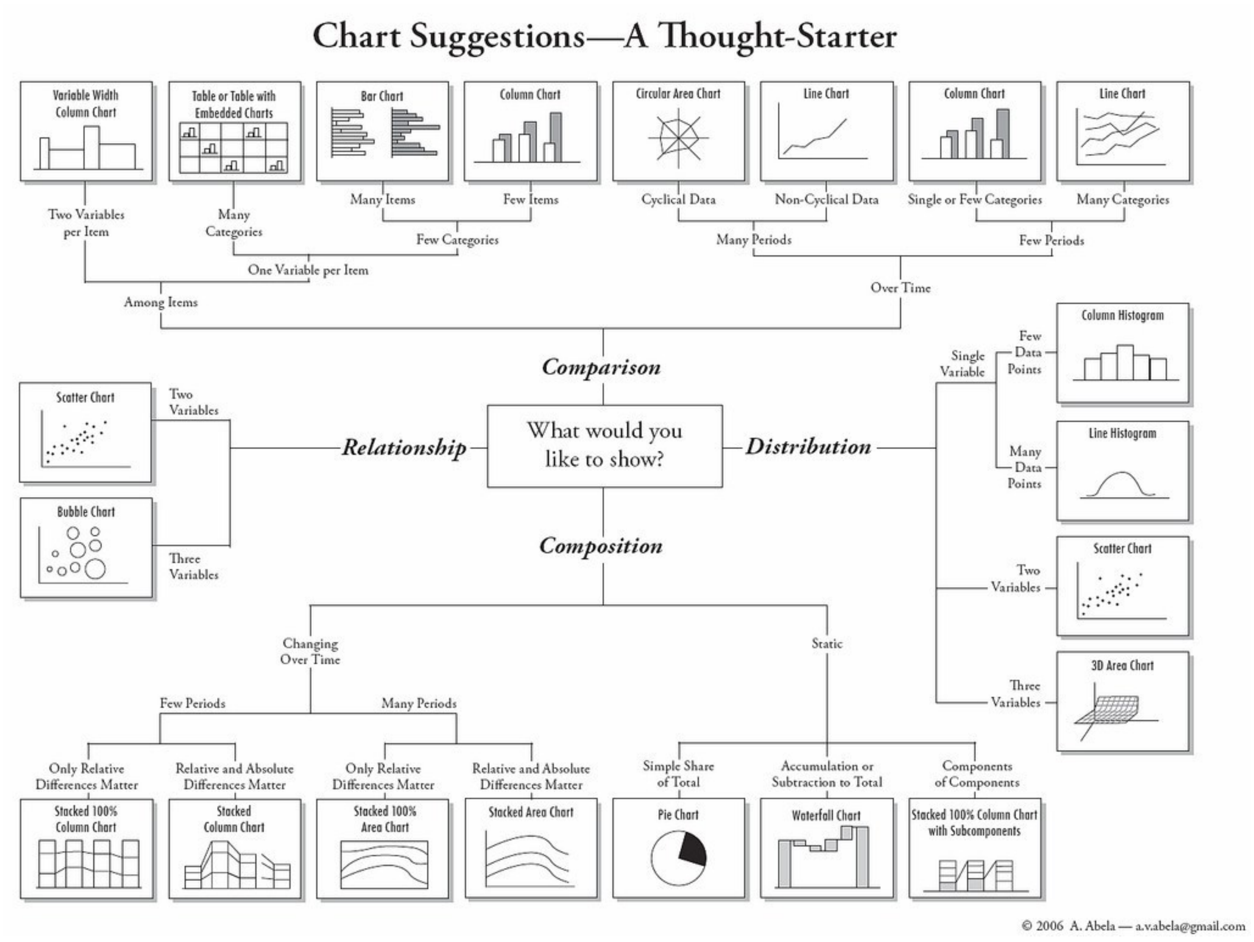
In [16]:
```python
sns.scatterplot(data = iris,
                    x = 'species',
                    y = 'petal_length',
                    )

plt.show()
```
executed in 107ms, finished 11:03:11 2019-12-16



3 Charts Starter

We have covered quite a few different plots and charts in this and the previous lesson. We have attempted to define what they are and when to use them, but we recognize this may still be confusing.

The following chart is a start in understanding what charts and plots to use given what you are trying to do. This chart comes from [https://flowingdata.com/2009/01/15/flow-chart-shows-you-what-chart-to-use/] (https://flowingdata.com/2009/01/15/flow-chart-shows-you-what-chart-to-use/%5D). It is not exhaustive, and it does not cover all of the charts/plots which we have covered in our last two meet-ups. However, it will help to organize your thoughts around charts and plots. It will also serve as a good basis for you to begin developing your own cheat sheet on charts and plots.



4 Exercises

Let's practice some of what we have learned.

4.1 Load the Data Set

First,

1. Let's load our needed libraries
2. Let's load the 'ames.csv' dataset (located in the data folder) into a DataFrame
3. Then take a look at the dataset's features to become familiar with them

In [88]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('data/ames.csv', index_col = [0])

display(df)
print(df.shape)
```
executed in 75ms, finished 13:11:13 2019-12-16

	PID	MS.SubClass	MS.Zoning	Lot.Frontage	Lot.Area	Street	Alley	Lot.Shape	Land.
Order									
1	526301100	20	RL	141.0	31770	Pave	NaN	IR1	
2	526350040	20	RH	80.0	11622	Pave	NaN	Reg	
3	526351010	20	RL	81.0	14267	Pave	NaN	IR1	
4	526353030	20	RL	93.0	11160	Pave	NaN	Reg	
5	527105010	60	RL	74.0	13830	Pave	NaN	IR1	
...	...	...	...	...	...	...	...	...	
2926	923275080	80	RL	37.0	7937	Pave	NaN	IR1	
2927	923276100	20	RL	NaN	8885	Pave	NaN	IR1	
2928	923400125	85	RL	62.0	10441	Pave	NaN	Reg	
2929	924100070	20	RL	77.0	10010	Pave	NaN	Reg	
2930	924151050	60	RL	74.0	9627	Pave	NaN	Reg	

2930 rows × 81 columns

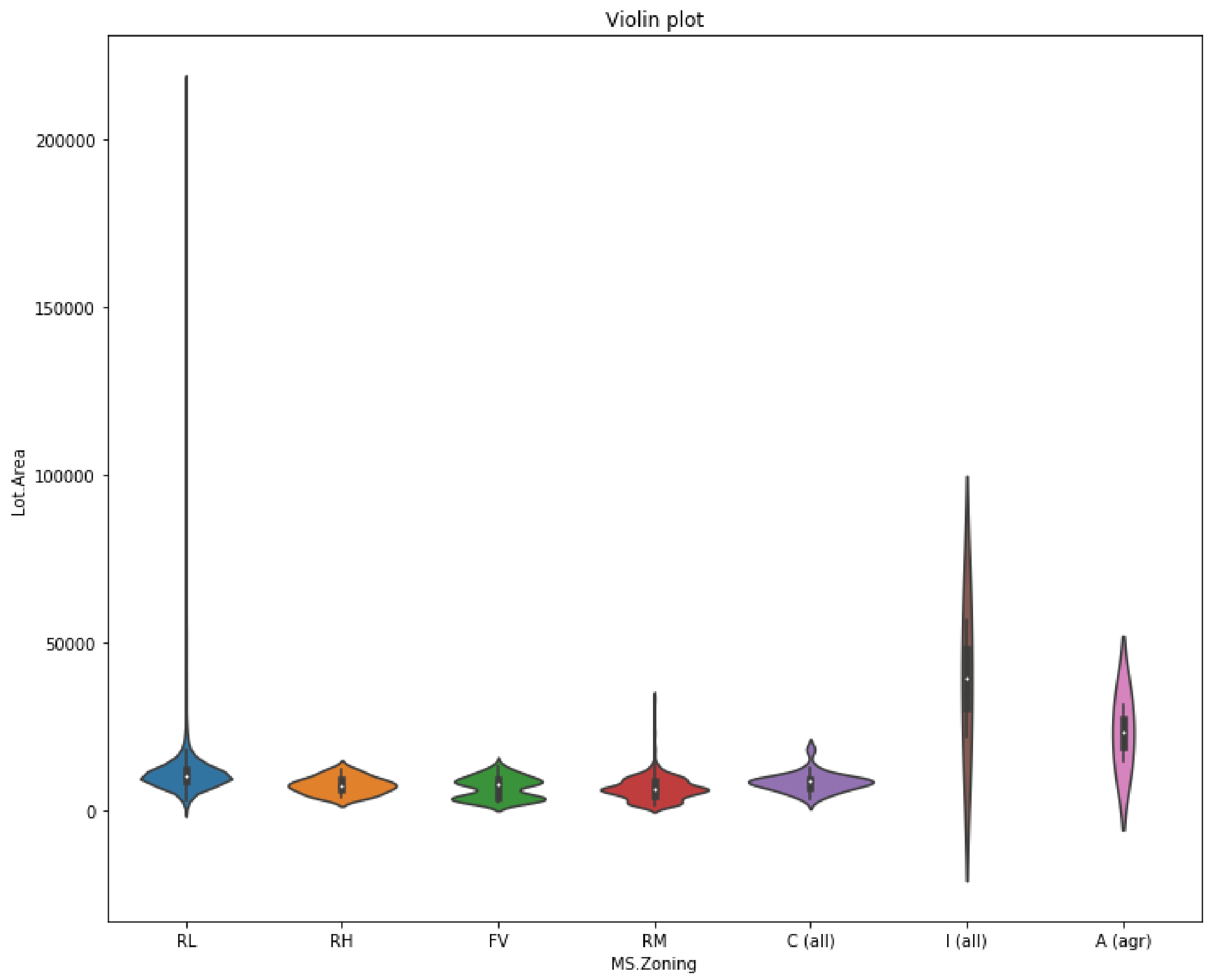
```
(2930, 81)
```

Next,

1. Plot 'Lot.Area' against 'MS.Zoning'
2. Select the plot that you believe best suits the needs for this representation

(Don't forget to add clarifying features.)

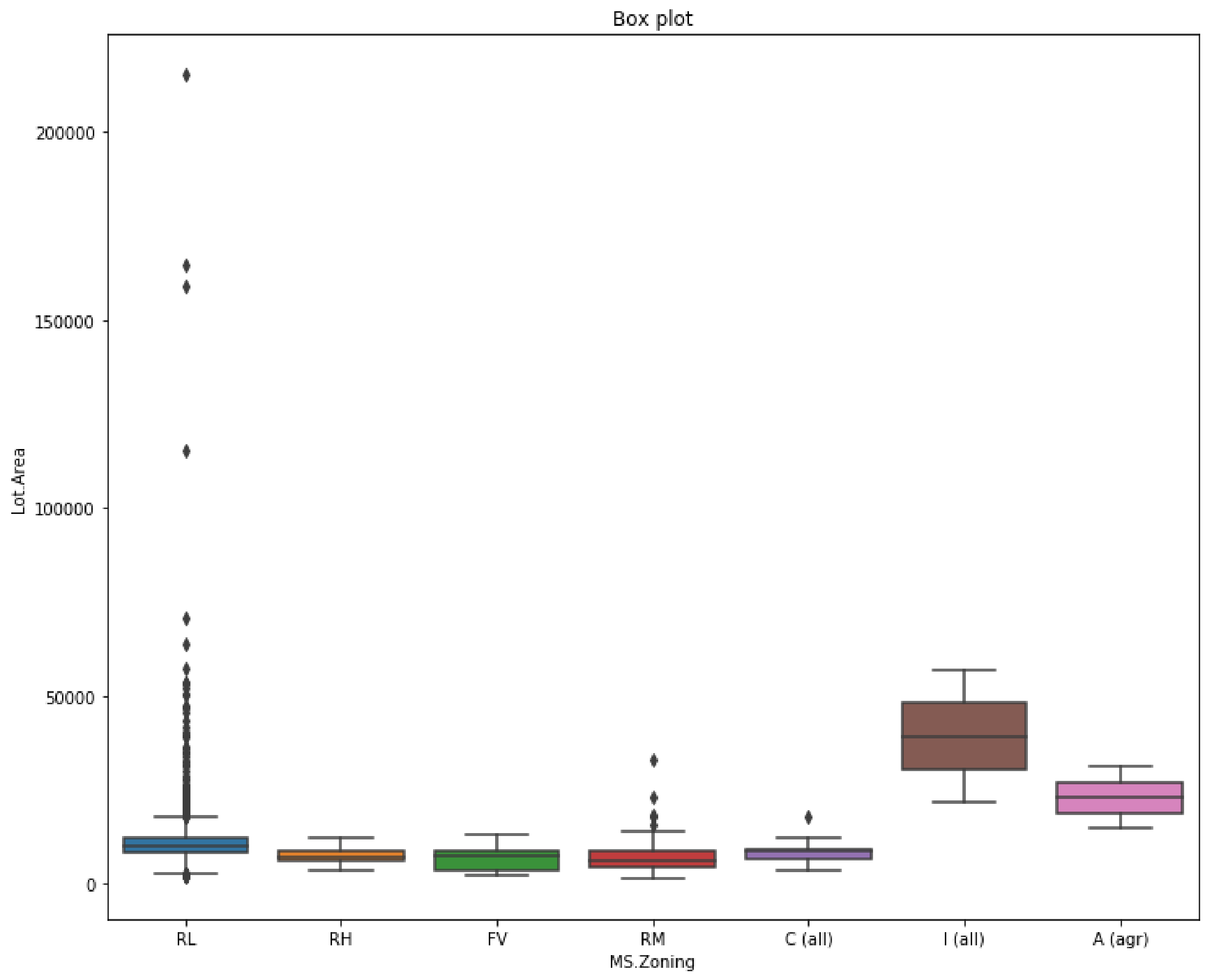
In [94]:
```
plt.figure(figsize=(12,10))

sns.violinplot(data=df,
               x = 'MS.Zoning',
               y = 'Lot.Area',
               )

plt.title('Violin plot')
plt.show()
```
executed in 221ms, finished 13:15:42 2019-12-16



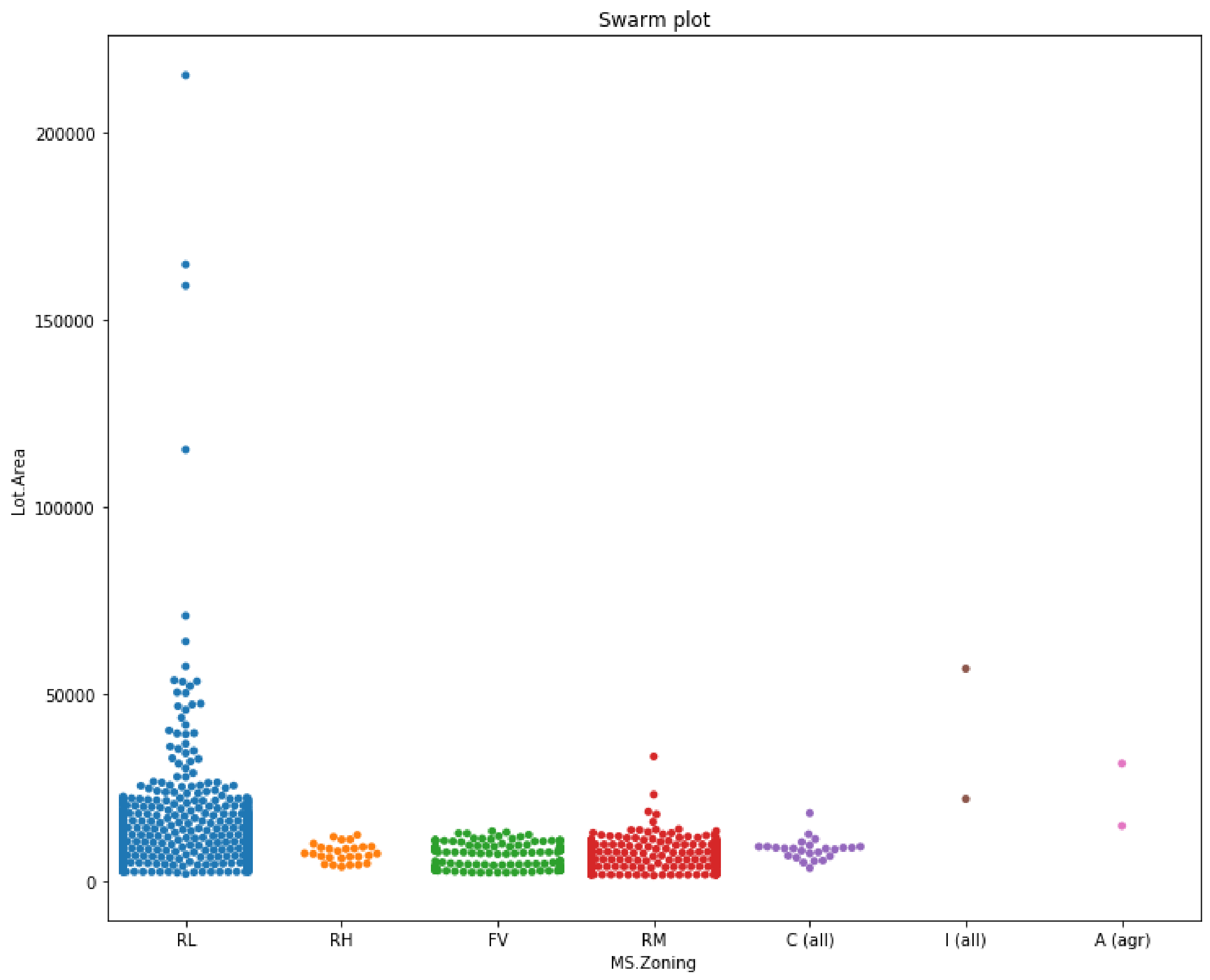
In [95]:

```
plt.figure(figsize=(12,10))

sns.boxplot(data=df,
            x = 'MS.Zoning',
            y = 'Lot.Area',
            )

plt.title('Box plot')
plt.show()
```

```
executed in 233ms, finished 13:15:42 2019-12-16
```



In [93]:
```python
plt.figure(figsize=(12,10))

sns.swarmplot(data=df,
              x = 'MS.Zoning',
              y = 'Lot.Area',
              )

plt.title('Swarm plot')
plt.show()
```
executed in 53.7s, finished 13:15:42 2019-12-16



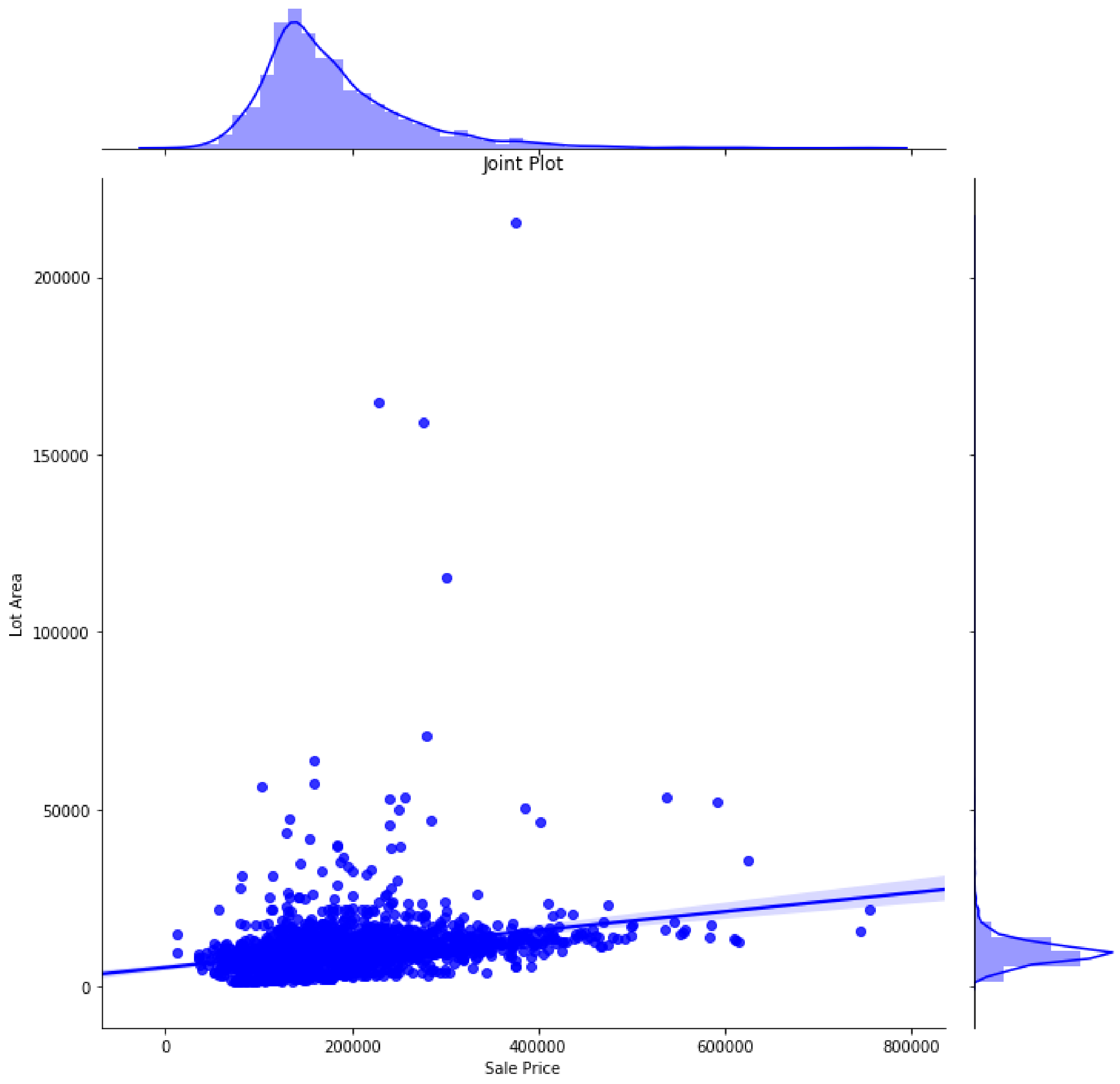
Next, we want to look at the relationship between 'SalePrice' and 'Lot.Area'. We are interested in seeing if there is a correlation between the two and what the distribution is for this relationship.

Make sure to make the plot large enough to see clearly. And make the color of the plot blue.

In [47]:
```
plt.figure(figsize=(12,10))# Set the figure size larger than default

sns.jointplot(data=df, #Declare the dataset
              x = "SalePrice", #Declare the x value
              y = "Lot.Area", #Declare the y value
              kind="reg", #Set it as regression
              height = 10, #Set the height of the plot
              color = 'blue',#Set the color as green
             )

#Add some clarifying aspects
plt.title('Joint Plot')
plt.xlabel("Sale Price")
plt.ylabel("Lot Area")
plt.show()
```

executed in 894ms, finished 12:36:42 2019-12-16

```
<Figure size 864x720 with 0 Axes>
```



In [ ]:

```

```

In [ ]:

```

```