**END-TO-END MACHINE LEARNING PROJECT**

**Working With Real Data**

* **Kaggle** is one of the most popular websites to download **datasets**.
* Imagine that you are an **intern** in a **real estate company**.
* We will be using dataset for **California housing prices**.

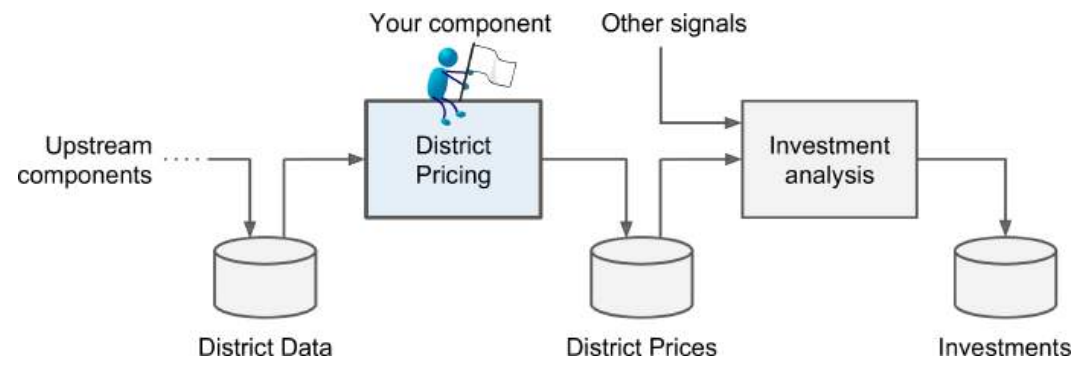
**Creating the Model for Solution (In a Nutshell)**

* Frame the problem
* Select a performance measure
* Check the assumptions

**Creating the Model for Solution (Brief)**

Frame the problem:-

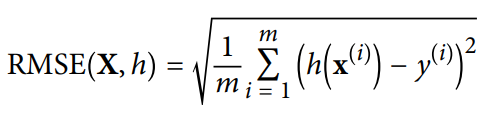
* First thing to know is **what’s the objective** of creating the model.
* This will help you figure out **what kind of model to make**, with **what properties**.
* **Signal:** Piece of information fed to a ML system.
* With our chosen dataset, our goal is to determine the **median housing price** & **feed** that to another model.
* This is because the company wants to know **how it must invest**.



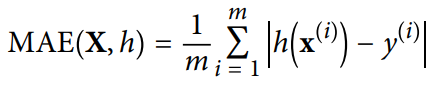
* **Pipelining:** The part of the ML system with many **components**; responsible for **processing** our data to a desired manner.
* These components run **asynchronously**.
* Between the **passings** of data from one component to another, the data is stored at ***data store***.
* **Multiple components** than **one large component** are helpful as different members in team can develop & monitor different components (**division of labour**).
* Plus, if any component is **broken**, then we can **easily detect** it **without disintegrating** other components.
* Proper monitoring is required to know if a component is broken.
* Next, we think **how the solution must look like**.
* For datasets like ours, a team of experts decide the statistics & if **not** being able to find the **median**, they use **complex formulae** to find it.
* But these calculations are often **inaccurate** (there comes time to train a model).
* Now decide **what kind of model is to be used**.
* You might ask the following questions while deciding:
  + Supervised or unsupervised?
  + Regression or classification?
  + Batch or online? Etc.
* Features that our model must include here:
  + Supervised learning model
  + Regression model (***multiple regression*** + ***univariate regression***)
  + Batch learning (as data is **small enough** to easily fit in the memory & it’s **one time job**).
* **Multiple regression:** Multiple attributes help in determining the target value.
* **Univariate regression:** We have to determine only **one target value**.
* **Multivariate regression:** ***\*Now you know\****

Select a performance measure:-

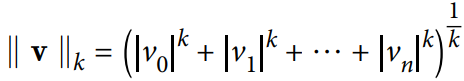
* We use **error measurement formulae** to calculate the possible error that could be generated in our predictions.
* One of it is ***RMSE*** *(****root mean square error****)*.



* **m** = Number of **rows/instances**
* **x(i)** = Vector (vertical matrix) of all **feature values** at **instance i**, except **label values**
* **y(i)** = Label value at **instance i**
* **X** = Vector of all **transposed** **instances** **x(i)** [**(x(i))T**]
* **h** = Prediction function called **hypothesis** [**ŷ(i) = h(x(i))**]
* For example, **median housing price** in first instance is **$158,400**.
* Then, **ŷ(1) = h(x(1)) = $158,400**
* **Prediction error = ŷ(1) – y(1) = 2,000**
* **RMSE(X,h)** is **cost function** measured on **whole dataset** using **hypothesis h**.
* We use **lowercase italic font** to represent **scalar values** & **function names**.
* And, we use **lowercase bold font** to **represent vectors**.
* Also, we use **uppercase bold font** to **represent matrices**.
* These are general convention to write them but **not** necessary.
* In case of many **outlier** instances, we prefer using **Mean Absolute Error (MAE)** instead.
* It is also known as ***Average Absolute Deviation (AAD)***.



* When we talk about **error** or **performance measurement**, we are basically referring to **calculating distance between two vectors** (prediction & actual value).
* **Norms:** Distance measures
* **RMSE** follows ***Euclidean norm*** (familiar distance), also called **ℓ2** or **||.||** or **||.||2** **norm**.
* **MAE** is also known as ***Manhattan norm*** because it is used to measure **orthogonal distances** between two points in a city.
* And it is also called **ℓ1** or **||.||1 norm**.
* Or simply speaking, a **ℓk** norm with vector **v** & **n** elements is defined as:



* **ℓ0**gives us the number of **non-zero elements** in the **vector**.
* Whereas, **ℓ∞**gives us the **maximum absolute value** in the **vector**.
* **Norm index:** Value of **k** in above equation.
* The **higher** the **norm index**, the **more** it **focuses** on **larger values** & **neglects small values**.
* That is why **RMSE** are **more sensitive** to **outliers** than **MAE**.

Check the assumption:-

* We **don’t** want to find out that the **assumptions** we made **mustn’t** been made, after months of work on model.
* In case of our chosen dataset, we have to pass the results from our model to another model, on which other members are working.
* What if that model tags the house prices as **cheap**, **medium** or **expensive**?
* Then it’s a **classification task** & we need **not** to get the prices perfect.
* So, let’s say that after confirmation, we found out that another model is **not** classifying the results passed from our model.
* Thus, we need to perform **regression** & **not classification**.

**Getting the Data (In a Nutshell)**

* Create the workspace
* Download the data
* Create a test set

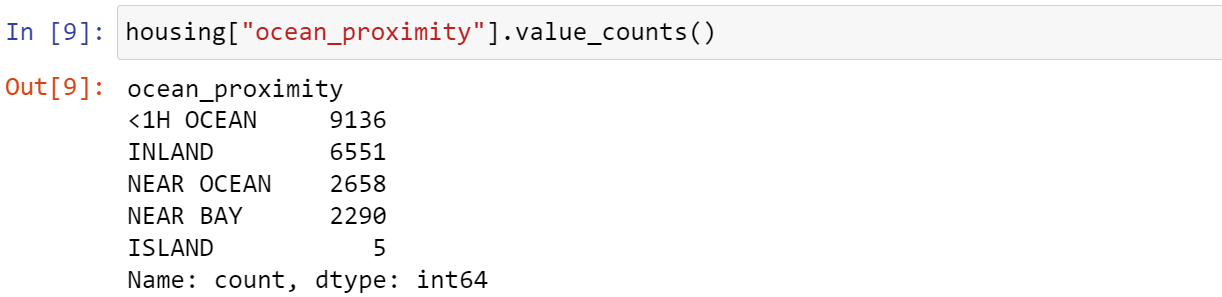
**Getting the Data (Brief)**

Create the workspace:-

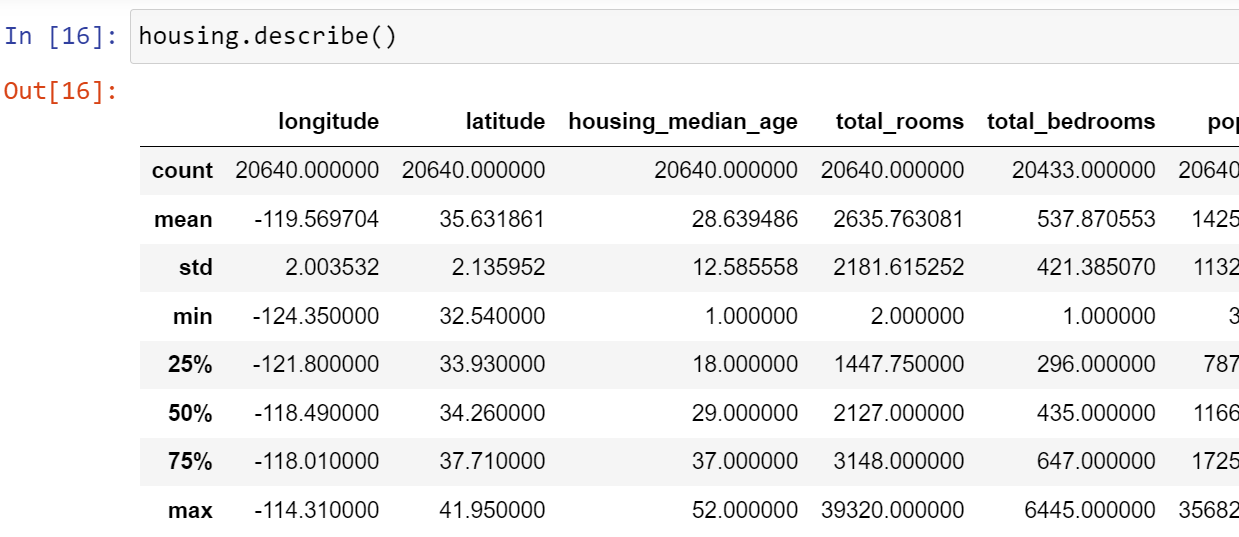
* Its **compulsory** to install **Python** from its official website.
* Then we can download **Anaconda navigator** to access all **scientific tools**.
* It is recommended to use create a **virtual environment**.

Download the data:-

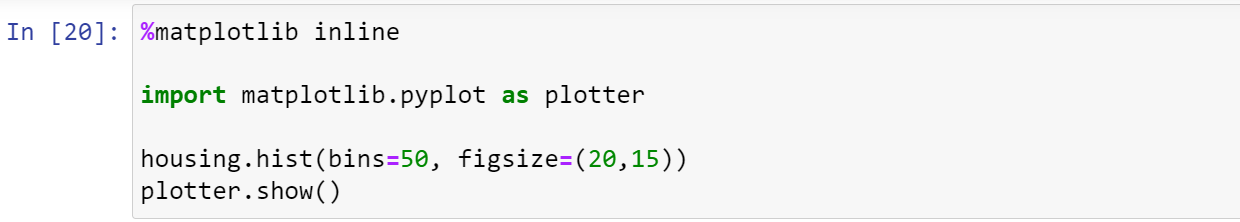
* We can now load the data to **Jupyter Notebook**.
* Notice that out of **20,640** **total\_bedrooms** attribute, only **20,433** are **non-null** values.
* We also notice that **ocean\_proximity** is an **object type** & repetitive.
* Repetitive values can mean that the attribute is **categorical**.
* To check **category distribution** statistics, write code as below.

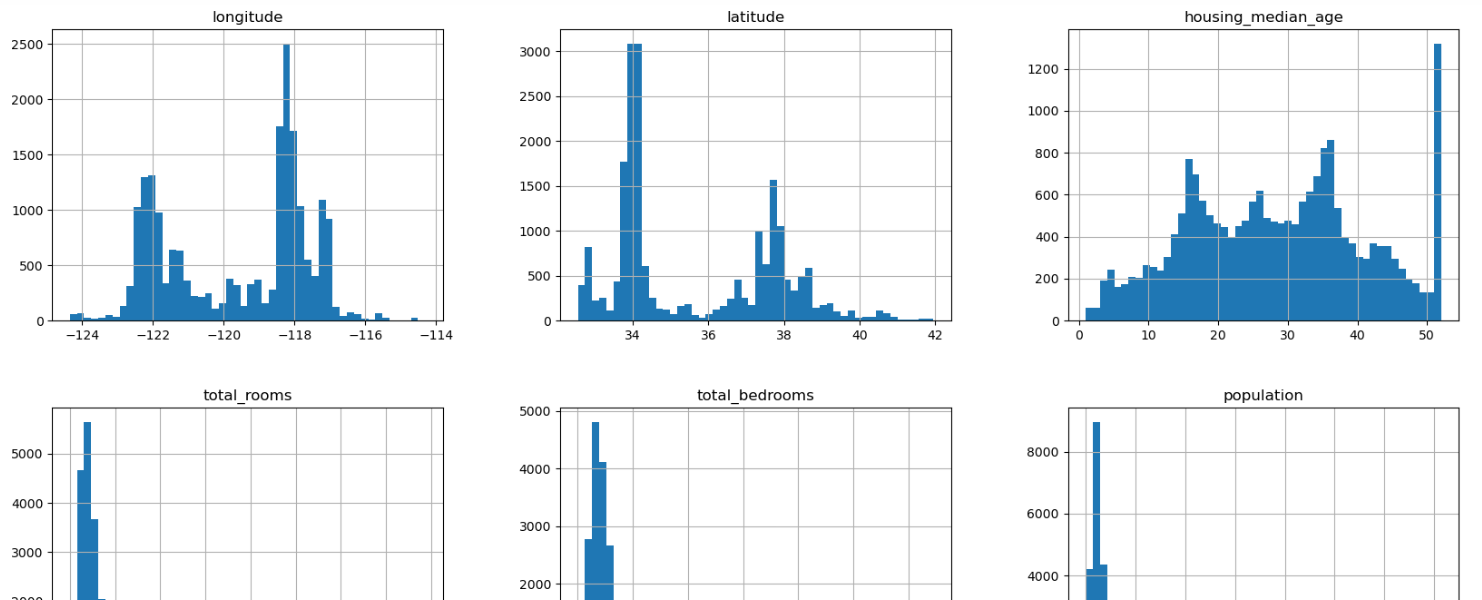


* Another thing to note is that in **Jupyter Notebook** the data are output **without** using **print** statements, unlike **IDLE**.
* The **describe()** method provides summary for numerical attributes.



* **25%** (1st quartile), **50%** (median) and **75%** (3rd quartile) are percentile for a given value.
* For example, **housing\_median\_age** of **25%** instances are below **18**.
* And, **housing\_median\_age** of **50%** instances are below **29** etc.

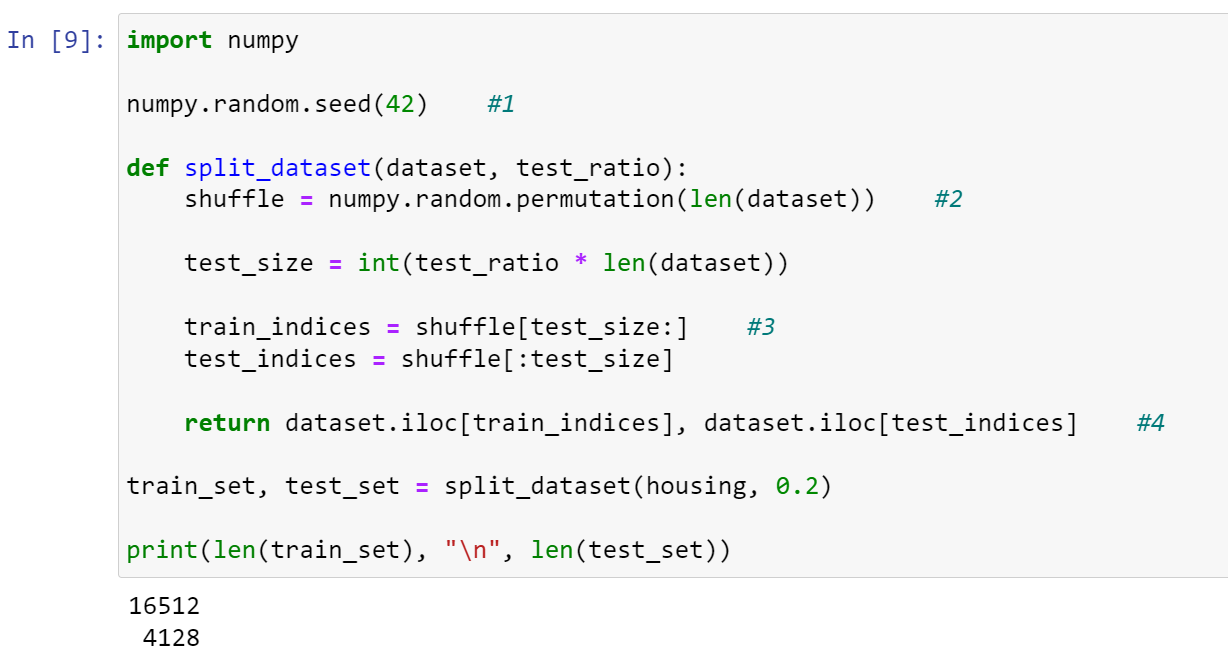




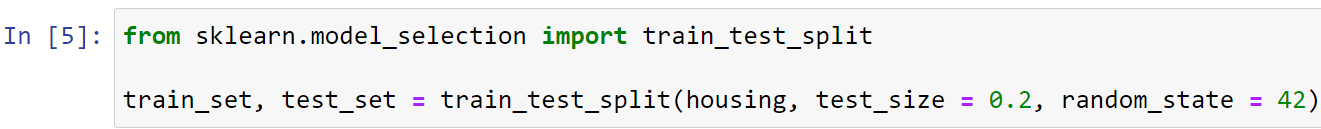
* **Inline magic commands** are used to render graphs on **Notebook** itself.
* With new version of **Jupyter**, writing **inline commands** are **not** necessary (but a good practice to write them).
* Parameter **bin** sets the **accuracy** of the plot (**high** **bin** value = **high** accuracy).
* **show()** method is option in **Jupyter**.
* **Cap:** Approximate
* Some attributes like **median\_income** have been **scaled** & **capped** in order to reduce the range of values.
* **Capped** **values** are actually result of **scaling**.
* **High medium incomes** are capped around **15.0001**.
* And **lower** ones are capped around **0.4999**.
* In this **scaled value**, a small figure like **3** might represent **$30,000**!
* This could create a problem while training our model, as they are **not** the original but **scaled values**, which **affects** how the model is trained.
* Many **histograms** as we see, are ***tail heavy*** (emptier right side than left).
* This makes detecting patterns a **little difficult** for the ML algorithm.
* Thus, we will try to transform them to a more **bell-shaped** distribution.

Create a Test Set:-

* **Data snooping bias:** Unknowingly considering all the data, including **test set** to create a suitable algorithm, which shows **good test results** but **fails** on the production.
* We need to only consider the **training set** for now & **test** the **test set** later to check performance.
* If we consider **testing set** too while picking up a good algorithm, it will **pass** the **tests** with **good** **performance**, but will **not** guarantee **good results** with production.
* So now we will **split** the dataset.



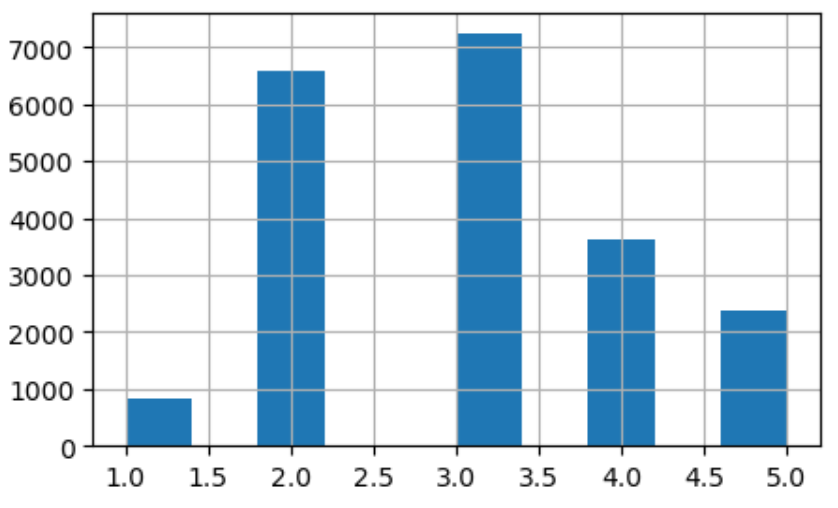
* At **#1**, we set a **random seed** in order to get **same** **random sets** each time.
* At **#2**, we created a list with **shuffled indexes** of our dataset.
* At **#3**, we created another list containing **training indices**.
* At **#4**, we **returned the rows** whose indices are mentioned in the respective lists.
* But the distribution pattern will **change** when a row is **added** or **deleted** from our dataset.
* So, one way to ensure that instances in **training set** & **testing set** don’t get exchanged is by using or setting a **unique id** to each instance for verification.
* Some of the options are to **add a column** with **unique hash codes**, or to use an attribute containing **unique values** for each instance.
* Best option is to use **Scikit-learn** for **splitting** up the dataset.



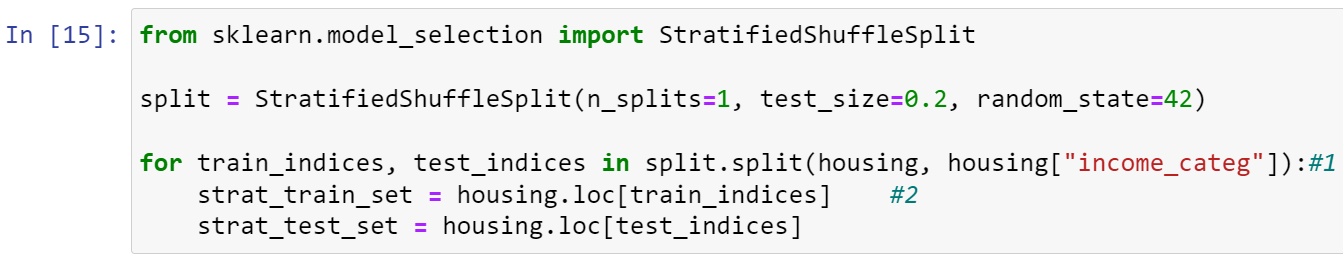
* During **sampling** or **survey**, people or objects are **not** picked **randomly** but in a **homogeneous ratio**, to avoid **sampling bias**.
* For example, **51.3%** of US population are **males** & **48.7%** are **females**.
* So, when sampling we will make sure that out of **1,000** people surveyed, **513** are **males** & **487** are **females**.
* This way of sampling is known as ***stratified sampling***.
* And the **subgroups** it introduces (like **males** & **females** above) are called ***strata***.
* In our case, the **median\_income** is a very important attribute to predict **median\_housing\_prices**.
* Most of our **median\_income** as we see in graph, are clustered around **1.5 to 6**.
* We must **not** have too much of ***stratum*** (singular for **strata**) & a good number of instances for each **stratum**, to avoid bias.
* We are going to use **Pandas’ cut()** method to **categorize** various **median\_income** ranges.



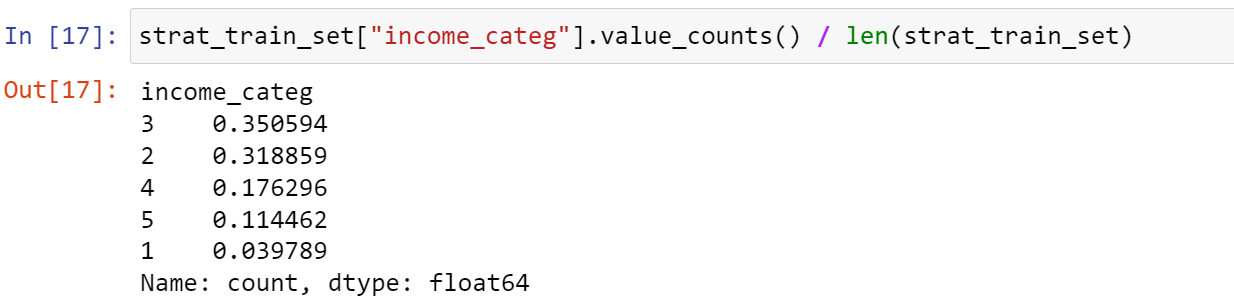
* At **#1**, the **list** called **bins** tells the group of ranges to be included in **income\_categ**.
* Means, **0 to 1.5** (**$0 to $15,000**) & **1.5 to 3.0** (**$15,000 to $30,000**) etc.
* **numpy.inf** means **positive infinity**.
* At **#2**, we are providing a **label** to each **range group**.



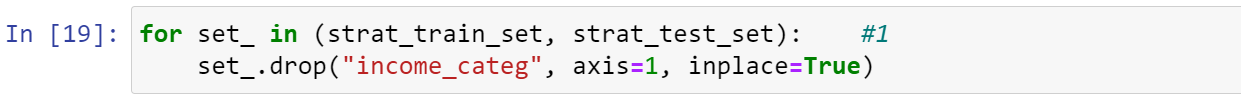
* Now we will do **stratified income sampling**, that means we have picked **different** income groups in order to avoid bias.
* We need to ensure that the **ratios** of **strata** are **same** in both **training** & **testing sets**.
* So, we will be using **Scikit-learn’s** **StratifiedShuffleSplit** class.



* At **#1**, **train\_indices** is strictly written **first** to denote **training indices** & same for **test\_indices**.
* And the parameters there tells the **dataset we are** **splitting** & the **category we are** **stratifying** on respectively.
* At **#2**, we are declaring a list **strat\_train\_set** containing all **training indices** in **homogeneous** ratio of chosen category (**income\_categ**).
* We can also check if the **stratified splitting** was successful or not by coding as below.



* Adding all these values must give us a **sum of 1** (approximately).
* Same if we check this with **strat\_test\_set**.
* Now, we can **drop** the **attribute** as our work is done now.



* At **#1**, the paranthesis tells on **which dataframes** we are making our **operations**.
* Or simply saying, instead of writing **two different** lines of codes mentioning the sets explicitly, we used **for loop** instead to **drop income\_categ** of each mentioned **argument dataframe**.
* We will later on also perform **cross validation**.

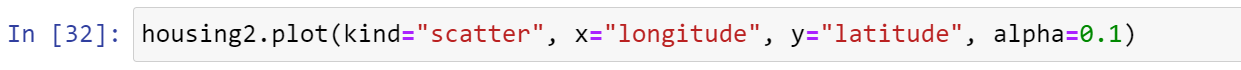
**Discover and Visualize the Data to Gain Insights (In a Nutshell)**

* Visualizing geographical data
* Looking for correlations
* Experimenting with attribute combinations

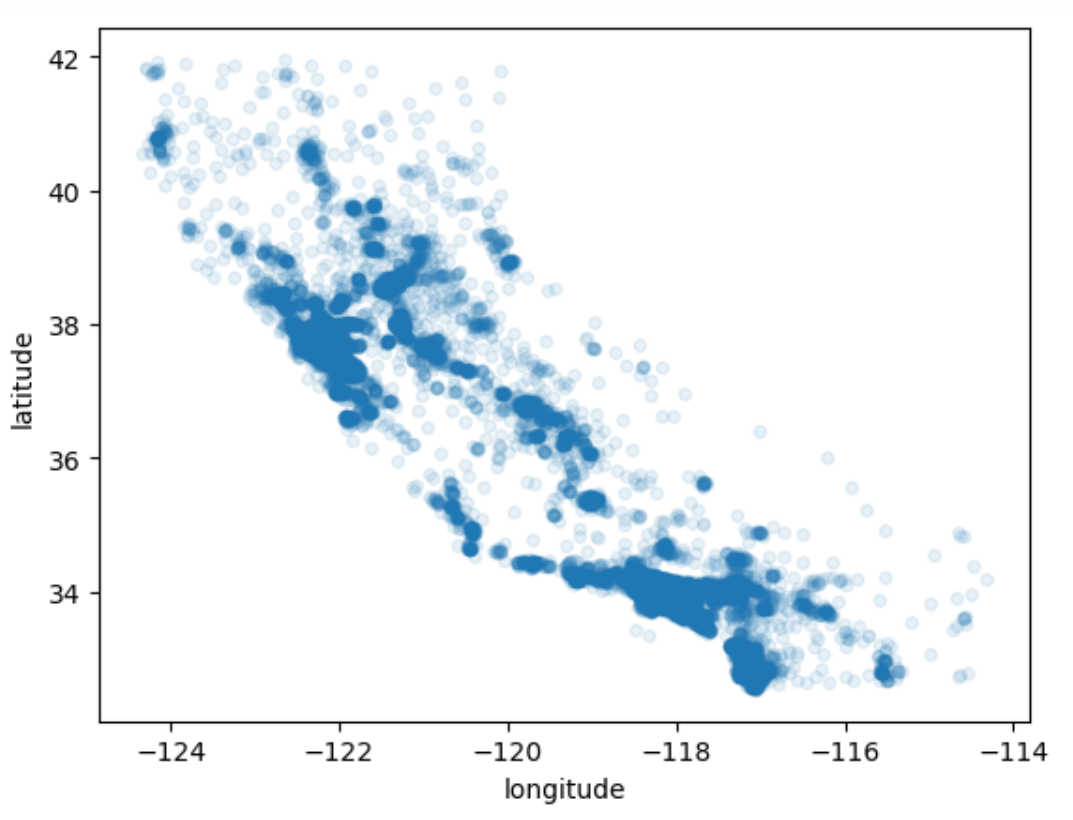
**Discover and Visualize the Data to Gain Insights (Brief)**

Visualizing geographical data:-

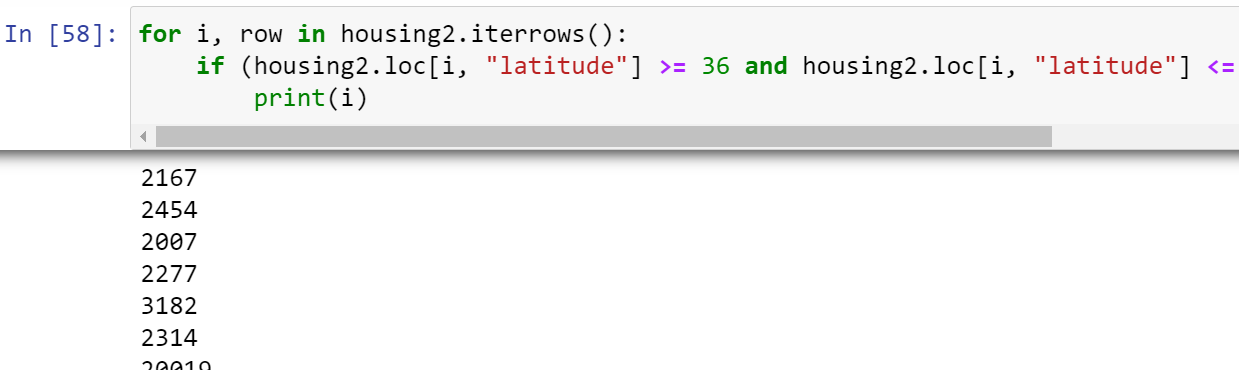
* It is **highly advised** to **copy** the **training set** when freely exploring it, to avoid making unrequired changes mistakenly.



* We can add the **figsize** argument here too if we want.
* **alpha** argument sets the **density** of datapoints (range = [0,1]).

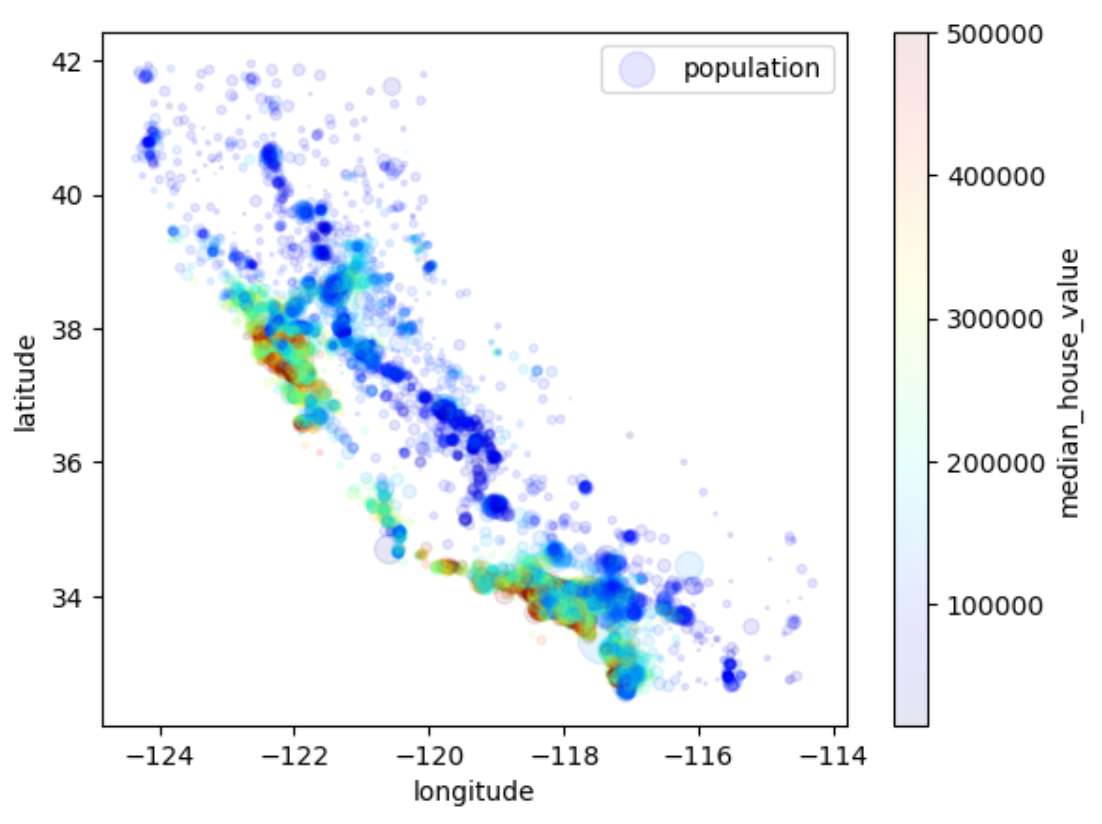


* Code as below for getting **indices** of **most dense points** in the scattered graph.





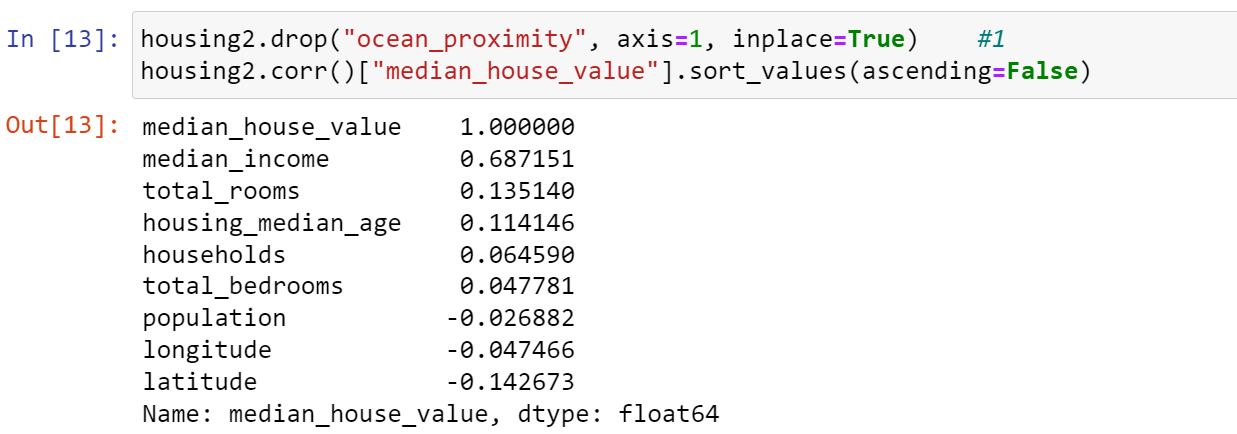
* At **#1**, label defines just a **label** which can be seen at **top right corner**, nothing else.
* At **#2**, **s** defines **radius** for each datapoint (which in our case will **vary** as per **population**).
* At **#3**, **c** defines what will affect the **colour** of our datapoints.
* At **#4**, **cmap** plots graph with **"jet"** type **colour** distribution (**blue=low** & **red=high**).
* At **#5**, **colorbar** when set to **True**, a **colour bar** appears on **right** as shown below.



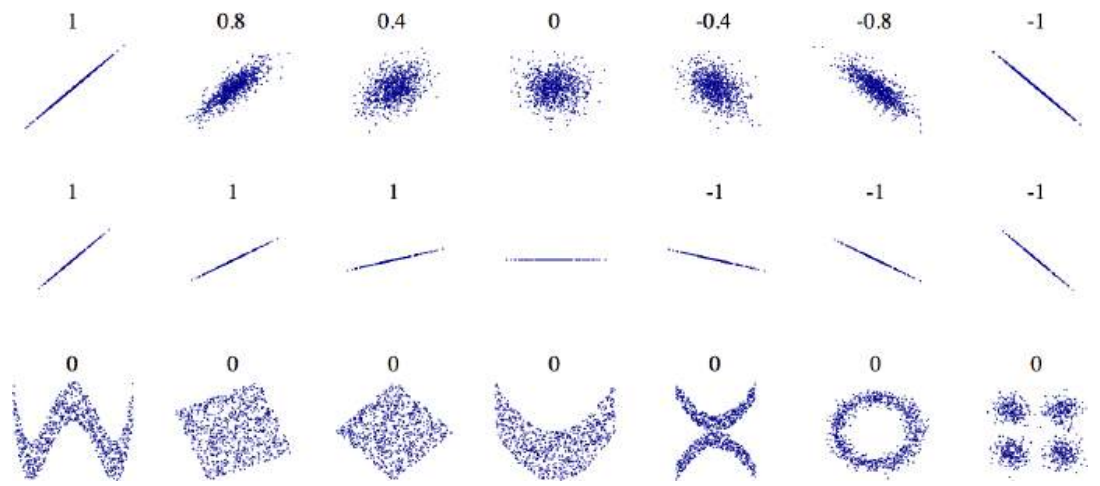
* So, we conclude that the **houses near ocean** are of **high prices** with **dense population**.
* We can perform **clustering** to cluster houses near ocean.

Looking for correlations:-

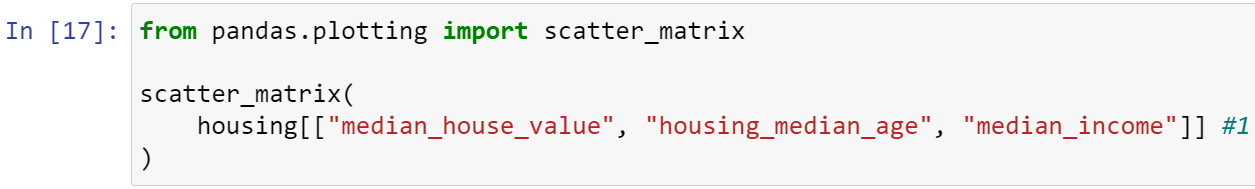
* We can compute ***standard correlation coefficient*** also known as ***Pearson’s r*** using **corr()** method.



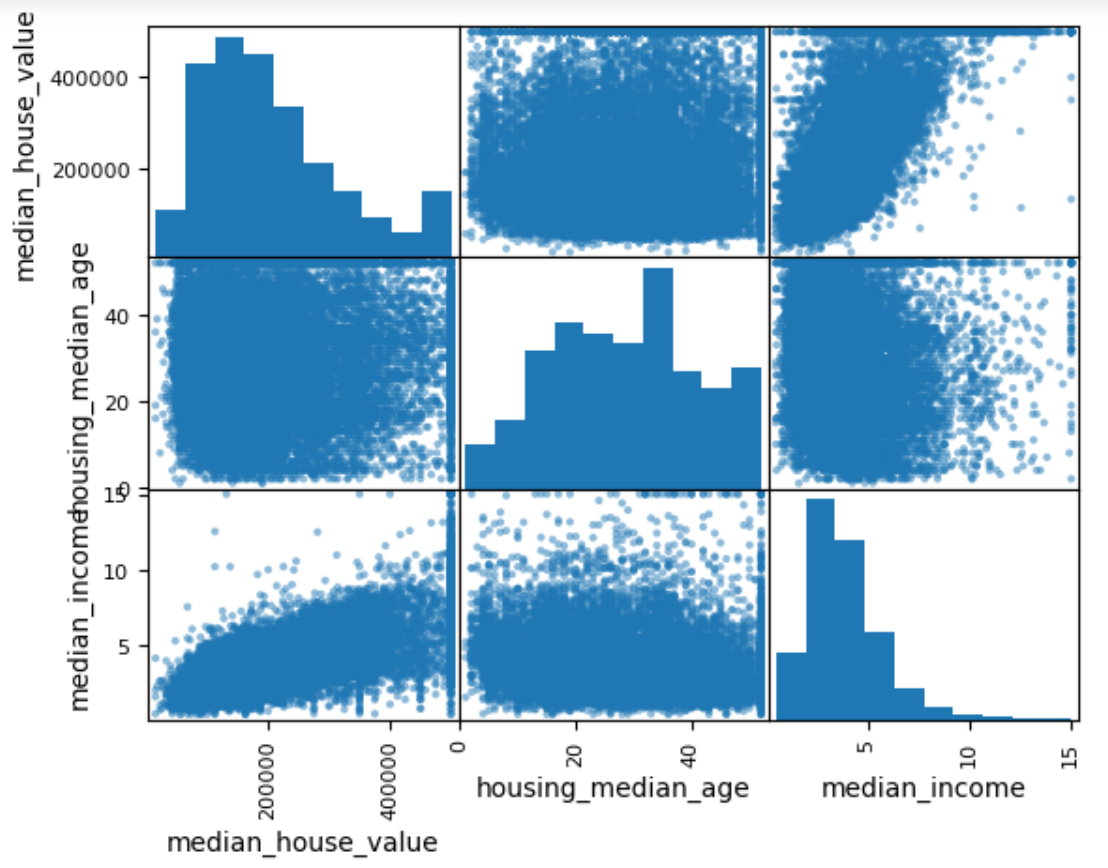
* At **#1**, we dropped the column with **string value**, because correlation is found among **numerical values** only. Not dropping it would have resulted in **error**.
* **Correlation** value **1** means **strong positive correlation**.
* Whereas value **-1** means **strong negative correlation**.
* The graphical diagram below shows how various **standard correlation coefficient** look like.



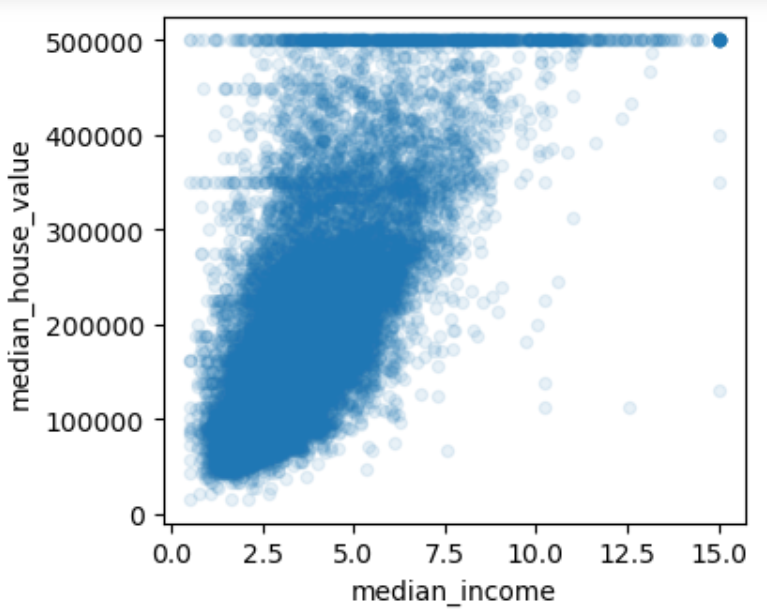
* **Pearson’s r** method is just for finding out **linear correlations** & not **non-linear correlations**.
* For example, see how last row shows various **symmetric shapes** (**dependent axes**). Which means that these datapoints are **non-linearly** correlated somehow.
* **Pandas’ scatter\_matrix** function plots correlation of each attribute **with one another**.
* And we have **9 attributes** with us right now in **housing2**, meaning **92 or 81 plots**.
* So, we would plot only few of the plots which are correlated with **median\_house\_value**.



* At **#1**, notice that attributes are written in **double list** form (**double square brackets**).
* One can also apply **shortcuts** like assigning them to a list & then passing the list as an argument.



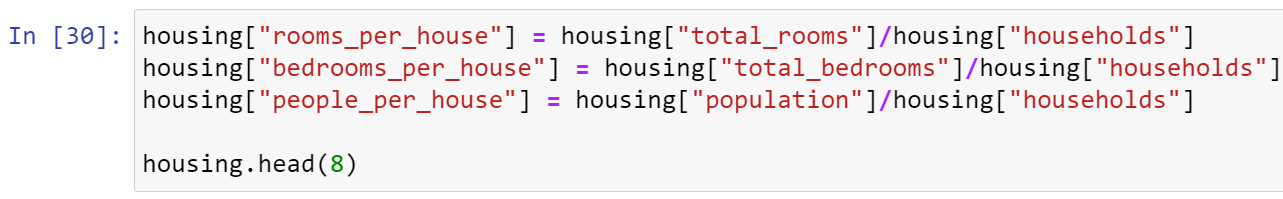
* The **top left** to **bottom right** **diagonal** shows **histogram** instead of straight line because we know it will be a **straight line**.
* So, **Pandas** has made it so to display **histogram** instead which, as it might provide **insights** we are unaware of.
* And these plots are nothing other than **simple scatter plots** between various attributes, it is just represented in form of **box matrix** as a collection.



* In this scatter plot between **median\_house\_value** & **median\_income**, we see few **horizontal lines**, the most visible one of them is at **$50,000**.
* These lines represent **outliers**, as they are forming pattern **different** from majority of the datapoints which **disrupts** the possible **regression line**.

Experimenting with attribute combinations:-

* There are some attributes like **total\_rooms**, **total\_bedrooms** & **population** etc, which are **not** useful in its raw form.
* For example, what will we do after knowing **total number of rooms** in whole district?
* We need to find **average number of rooms per household** to get an idea about **type of households** in a given district.



* When checking the **correlation** with **median\_house\_value**, we find our recently created attributes as **more correlated** than the ones we said are **not** much useful.

**Prepare the Data for Machine Learning Algorithms (In a Nutshell)**

* Data cleaning
* General note on Scikit-learn’s features
* Handling test & categorical attributes

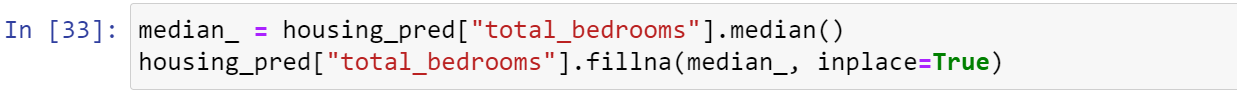
**Prepare the Data for Machine Learning Algorithms (Brief)**

* It is **highly recommended** to use **functions** for **preparing data**, in order to save time than writing same code again and again.
* One way is to **write library** with various ways to **transform data** & use it in future projects.
* So, we will start with a **clean dataset** in order to gain better clarity.
* And also, we would be keeping our **predictors** & **labels** (**median\_house\_value**) separately, because we might prefer **different** transformation methods on both.

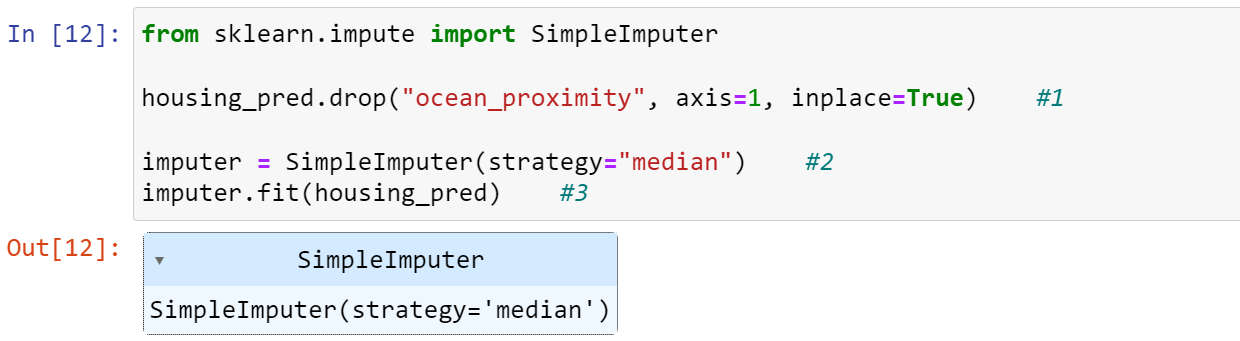


Data cleaning:-

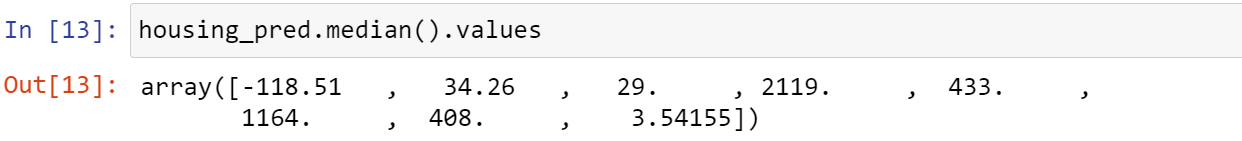
* First, we will fix the **missing values** we noticed earlier in **total\_bedrooms**.
* We have **3 options** to fix it:
  + Remove the rows with **null** **total\_bedroom** value.
  + Remove whole attribute.
  + Fill the **null total\_bedroom** with some value (zero, mean, median etc).
* We are proceeding with the **3rd option**.



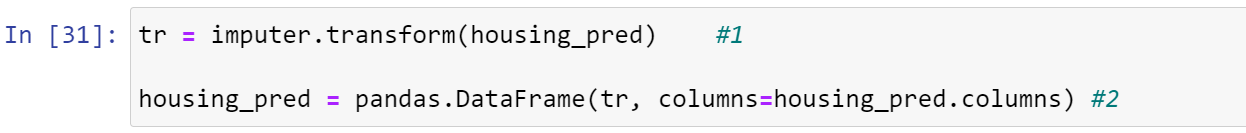
* We need to save the value of **median**, as it will be further required when filling missing values in **testing set** too, or when **adding new data** added at production.
* For making the 3rd option simpler, we will use **SimpleImputer** **class** from **Scikit-learn**.



* At **#1**, once we **drop** an attribute from a **dataframe**, running it first time will be fine, but from next time it might say that ***the attribute doesn’t exist*** (for being **dropped** in previous run).
* At **#2**, we **set the imputer** to **calculate** the median.
* At **#3**, we **calculated** the chosen value to fix (**median** here) of each attribute.
* And these attributes are in form of **NumPy** array, **not** **Pandas** DataFrame.
* **imputer.statistics\_** named instance variable now (of **Scikit-learn**) contains the **imputed values** of **all** the attributes.
* We **can’t** be sure if other attributes contain **null** values or **not**, that’s why we used **imputer** instead of **fillna**.



* This code above returns the same as statistics\_.
* Now we can fill the missing values as we have fit the imputer, as shown below.



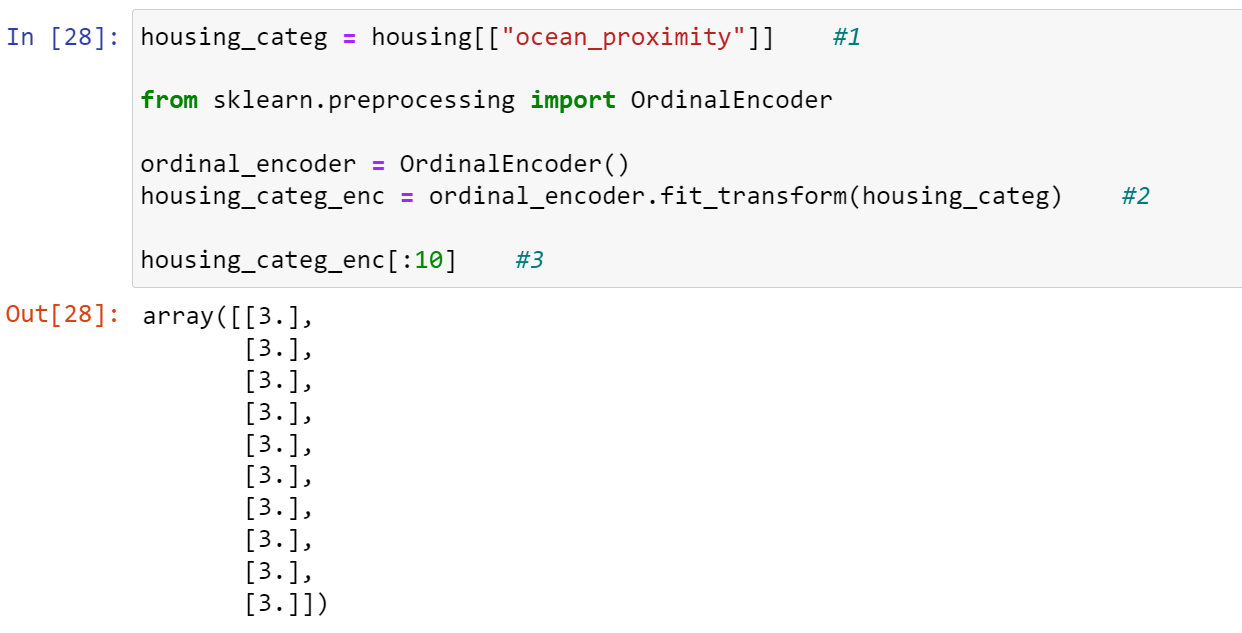
* At #1, we assigned a variable tr transformed training set in form of NumPy array.
* At #2, we fixed the transformed values in NumPy array to our Pandas dataframe.

General note on Scikit-learn’s features:-

* Estimators: Object in Scikit-learn which aid in estimating any kind of values (like imputer above).
* fit() method can take two parameters in supervised learning algorithm, second parameter containing labels.
* Transformer: An estimator which can transform a dataset.
* Predictor: An estimator which can predict.
* Inspection: Accessing parameters through public instance variables (like statistics\_).
* Nonproliferation: Representation of datasets as NumPy arrays/matrices & that of hyperparameters as simple Python string or numbers.
* Composition: Using existing building blocks in code.
* Sensible defaults: Provides reasonable number of default values, so the programmer doesn’t have to remember each parameter.

Handling test & categorical attributes:-

* Earlier we dropped the ocean\_proxomity attribute to avoid error while computing median.
* We need to now retain it back in order to proceed with our work.



* At #1, we used double square brackets for dataframe (tabular form) to be returned while accessing the dataset, instead Pandas series (plain text table).
* At #2, we are fitting & transforming in a single line using fit\_transform() method.
* At #3, this line returns a NumPy array instead rows from dataframe.
* This array contains numbers which are IDs assigned to various categories of ocean\_proximity.

