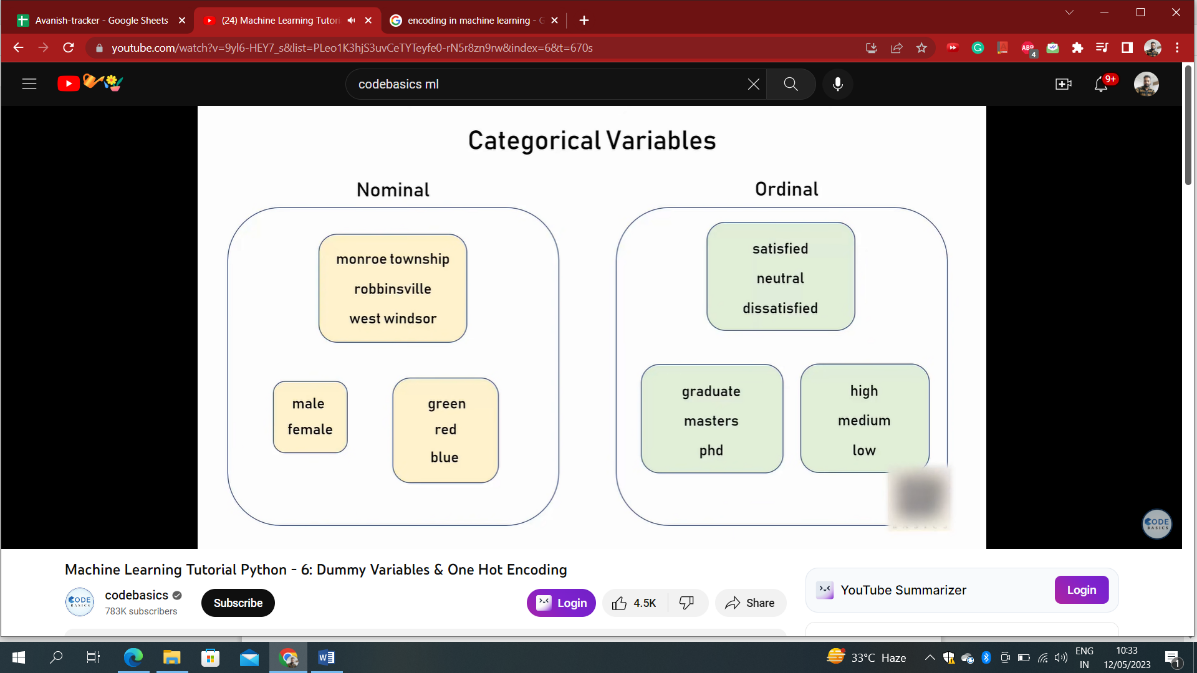
**Encoding**

The performance of a machine learning model not only depends on the model and the hyper-parameters but also on how we process and feed different types of variables to the model. Since most machine learning models only accept numerical variables, preprocessing the categorical variables becomes a necessary step. We need to convert these categorical variables to numbers such that the model is able to understand and extract valuable information.



we can see there are two kinds of categorical data-

* **Ordinal Data:** The categories have an inherent order
* **Nominal Data:** The categories do not have an inherent order

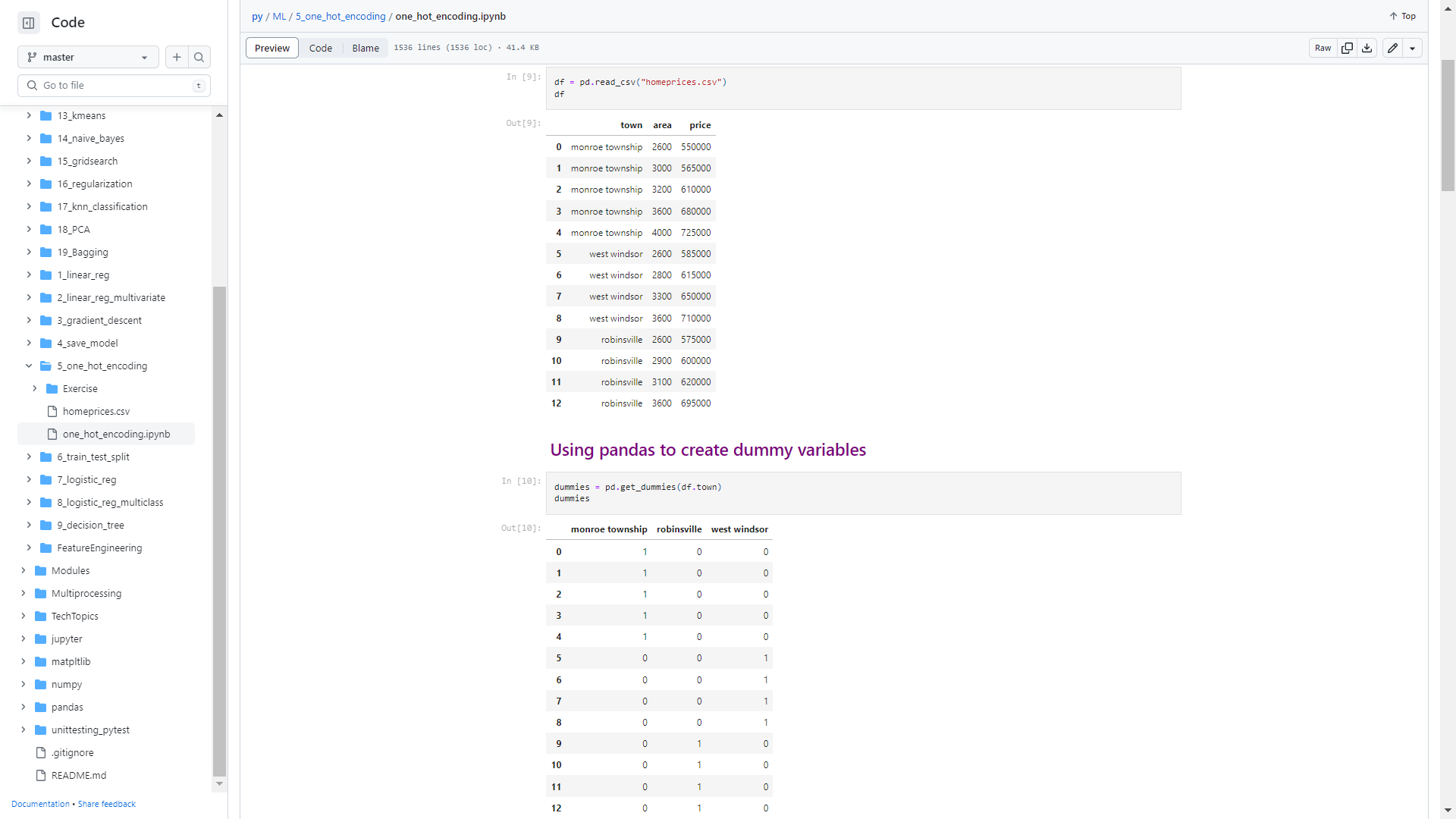
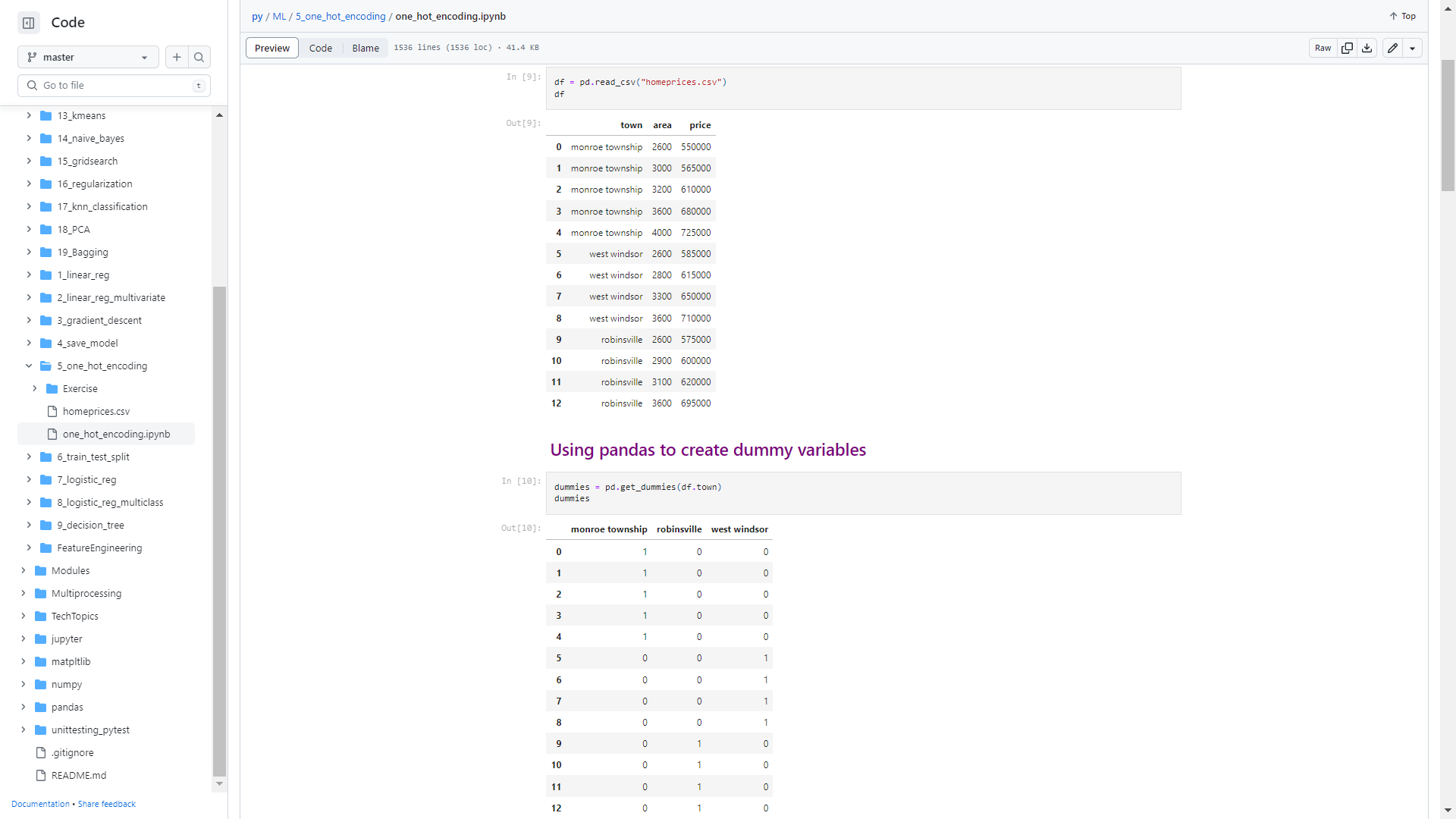
In ordinal data, while encoding, one should retain the information regarding the order in which the category is provided. Like in the above example the highest degree a person possesses, gives vital information about his qualification. The degree is an important feature to decide whether a person is suitable for a post or not.

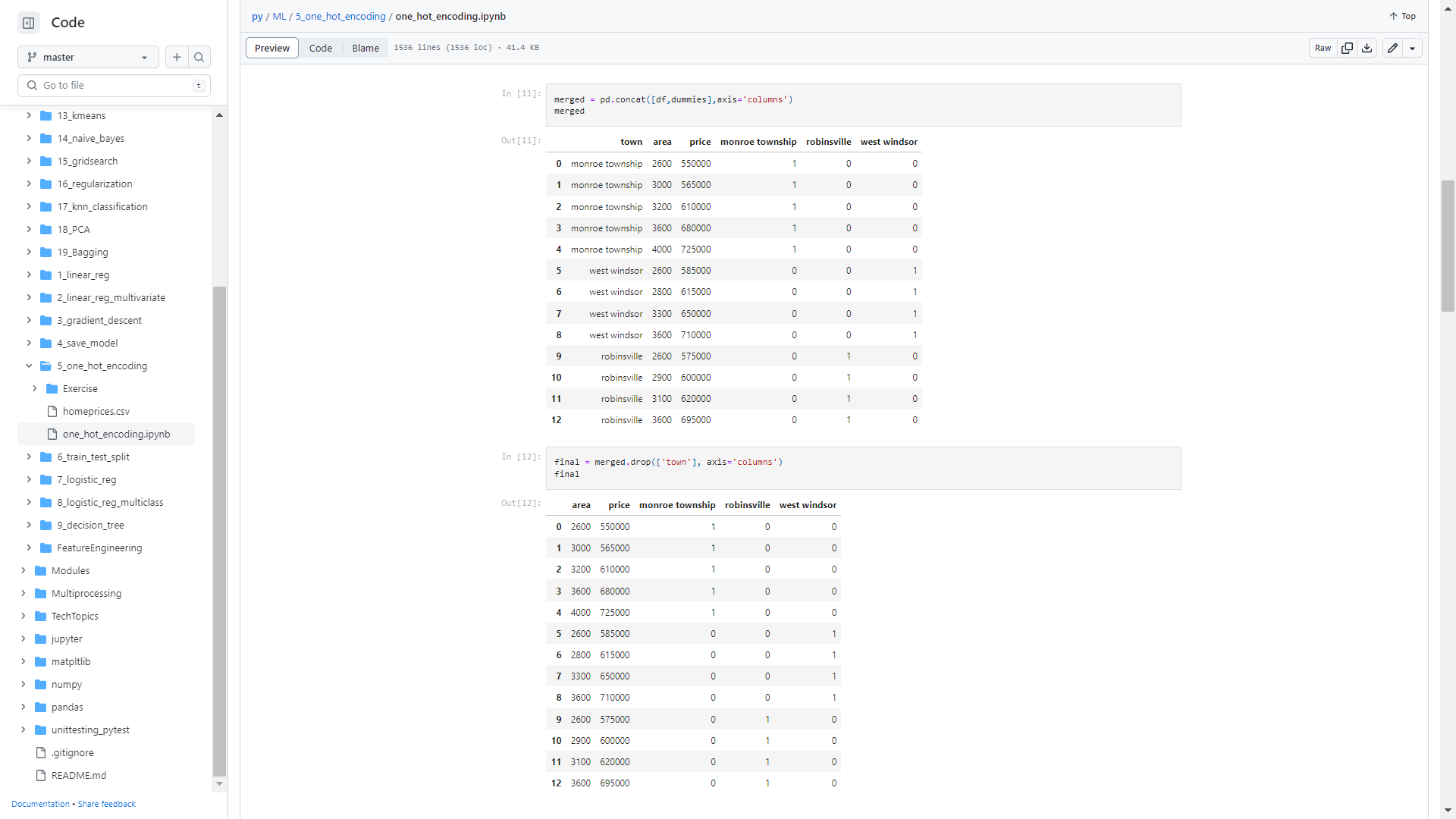
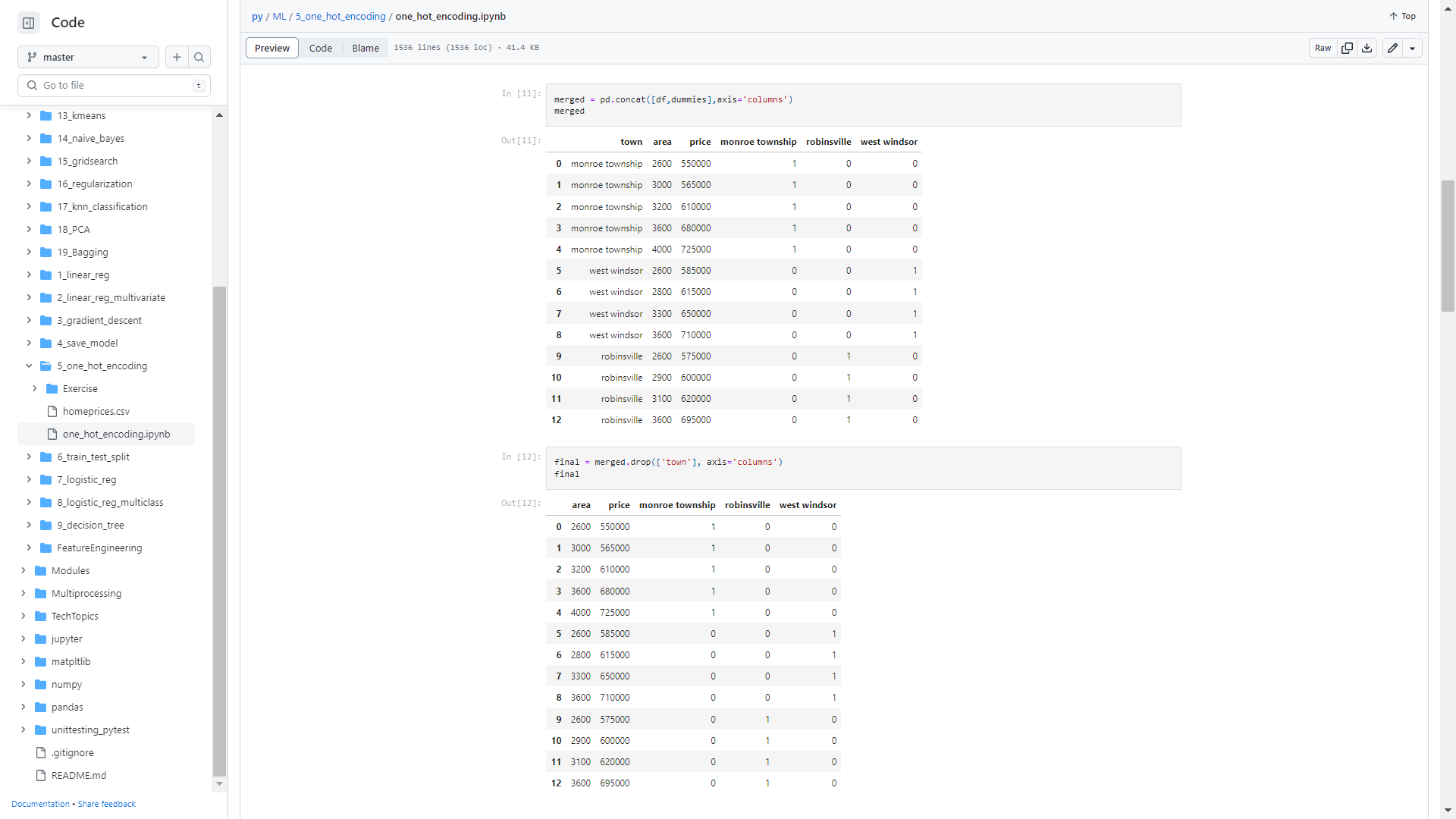
While encoding nominal data, we have to consider the presence or absence of a feature. In such a case, no notion of order is present. For example, the city a person lives in. For the data, it is important to retain where a person lives. Here, no need to have any order or sequence. It is equal if a person lives in Delhi or Bangalore.

## **Dummy Encoding**

We use this categorical data encoding technique when the features are nominal(do not have any order). In dummy encoding, for each level of a categorical feature, we create a new variable. Each category is mapped with a binary variable containing either 0 or 1. Here, 0 represents the absence, and 1 represents the presence of that category.

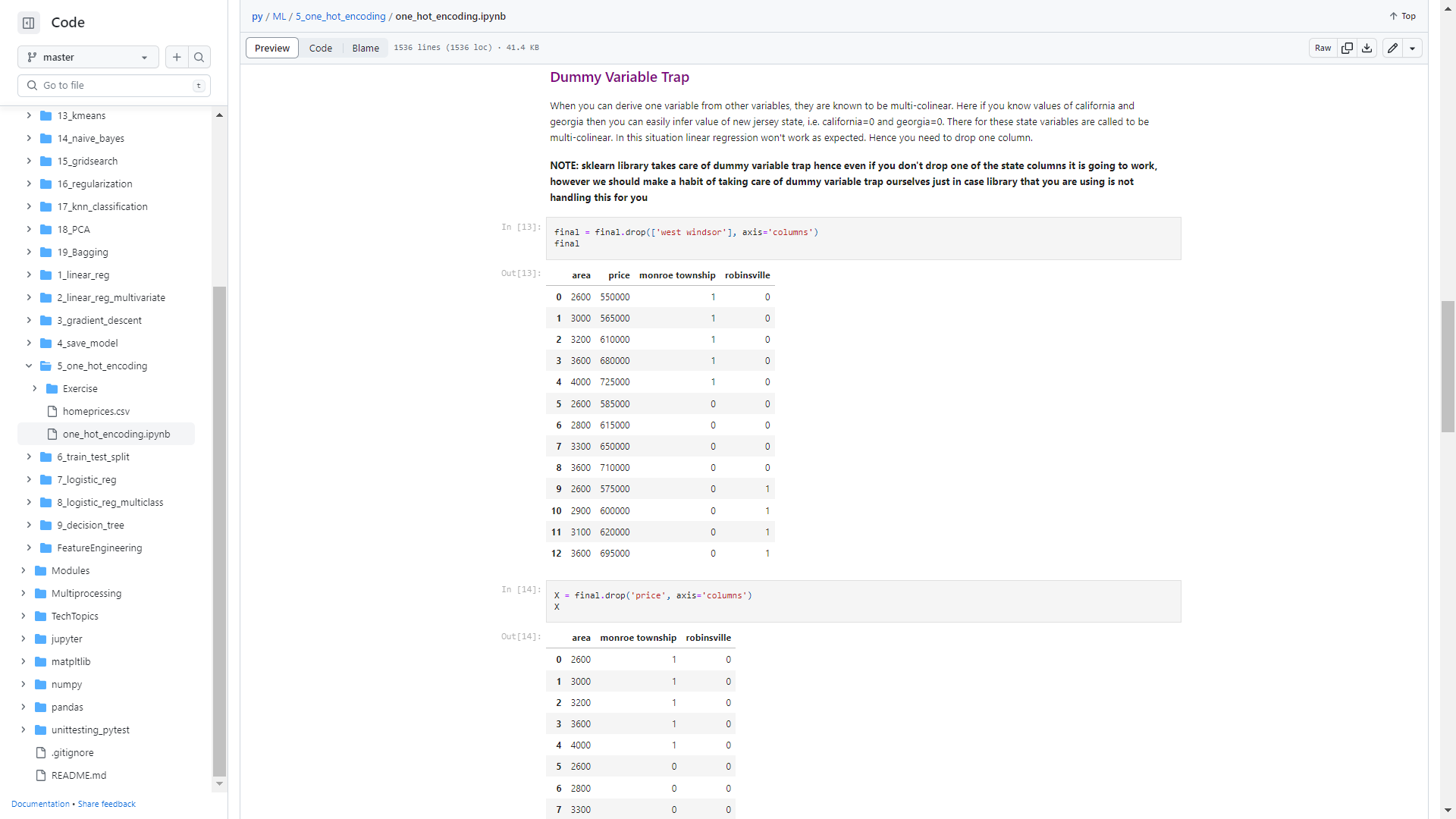
These newly created binary features are known as**Dummy variables.** The number of dummy variables depends on the levels present in the categorical variable. Dummy encoding uses N-1 features to represent N labels/categories. We will see it further. Let us implement it in python.

**Dummy Variable Trap**

When you can derive one variable from other variables, they are known to be multi-collinear. Here if you know values of ‘West Windsor’ = 0 and ‘Robinsville’ = 0 then you can easily infer value of ‘Monroe Township’ = 1 and likewise. Therefore these state variables are called to be multi-collinear. In this situation linear regression won’t work as expected. Hence, you need to drop one column.

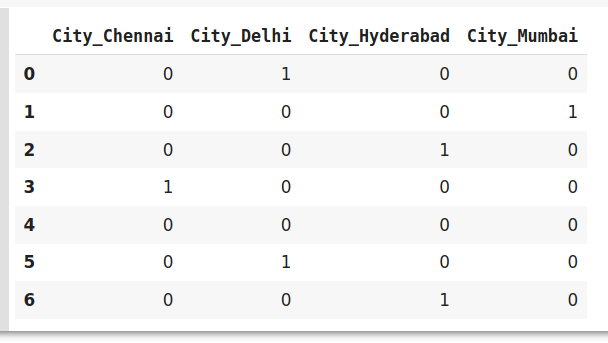


**Note:** One of the column can be dropped using the argument ‘’**drop\_first**’’ during creation of the dummy variables.

#encode the data

dummies = pd.get\_dummies(data.city, drop\_first=True)

dummies

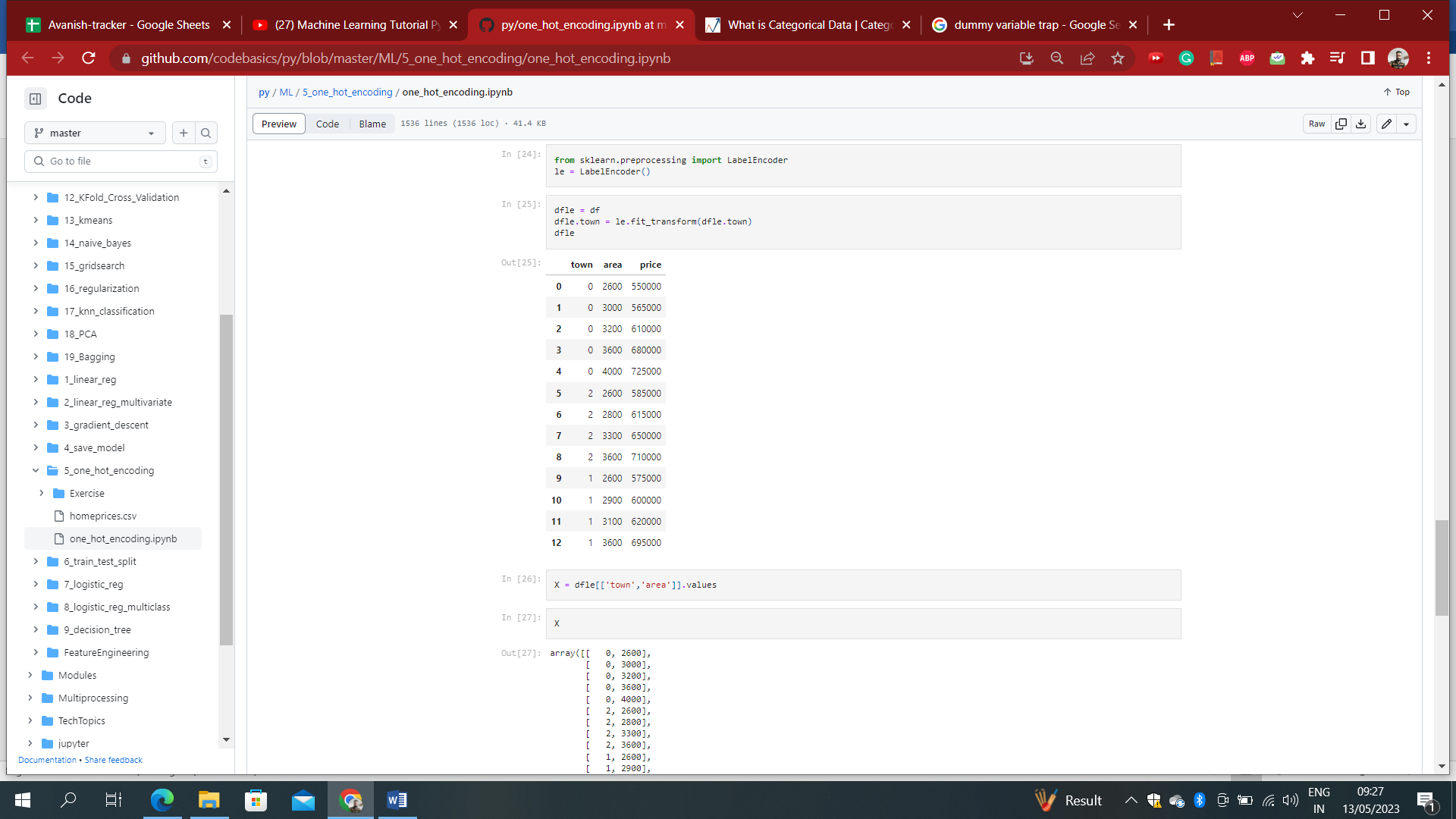
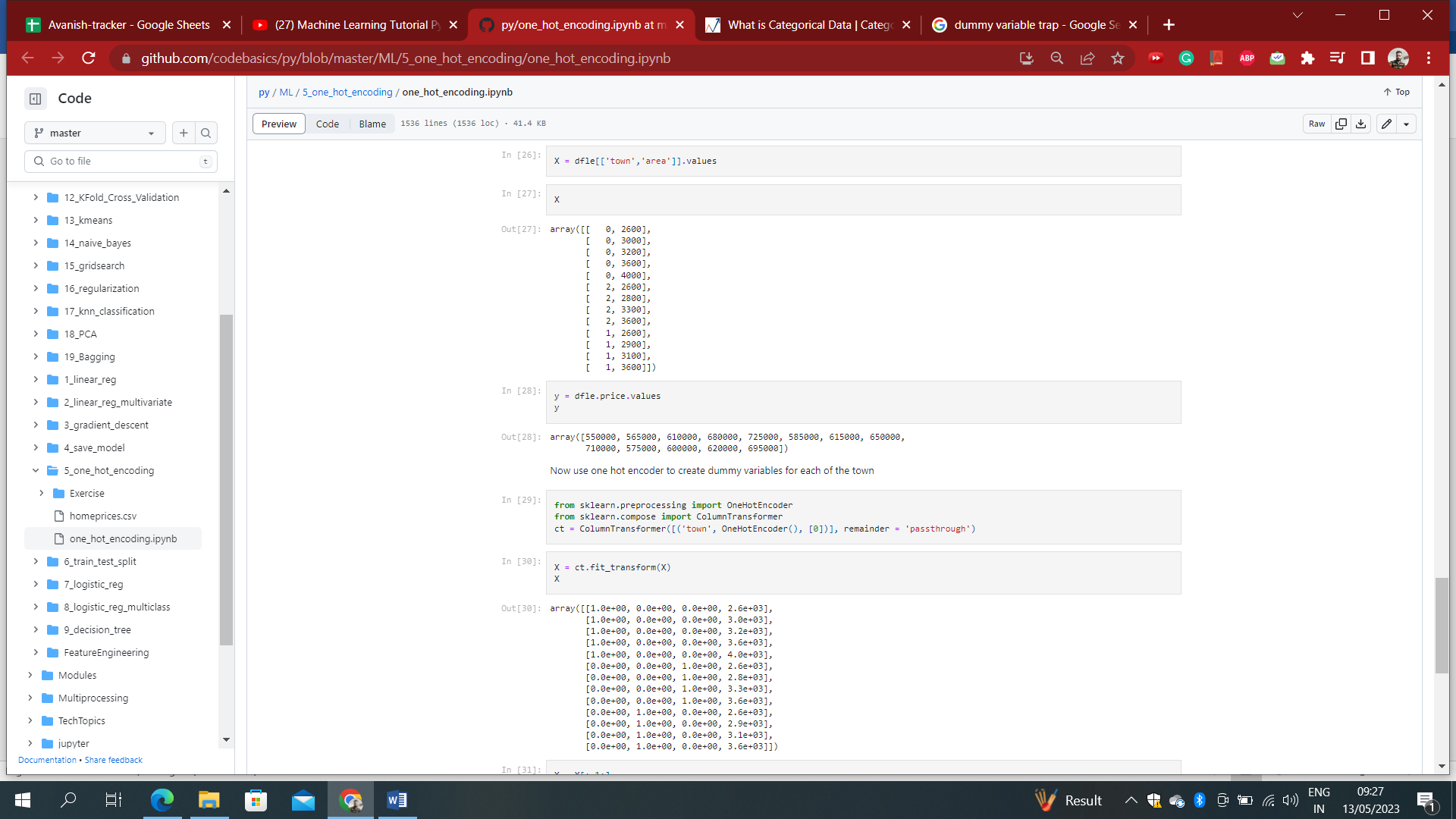
 

Here the column ‘city\_Bangalore’, has been dropped. Thus, Dummy encoding uses 4 features to represent 5 labels/categories.

## **Label Encoding or Ordinal Encoding**

We use this categorical data encoding technique when the categorical feature is ordinal. In this case, retaining the order is important. Hence encoding should reflect the sequence.

In Label encoding, each label is converted into an integer value. We can do the same as dummy encoding by employing combined method of label encoding and one hot encoding(discussed further). But, this time we are going to use sklearn library.

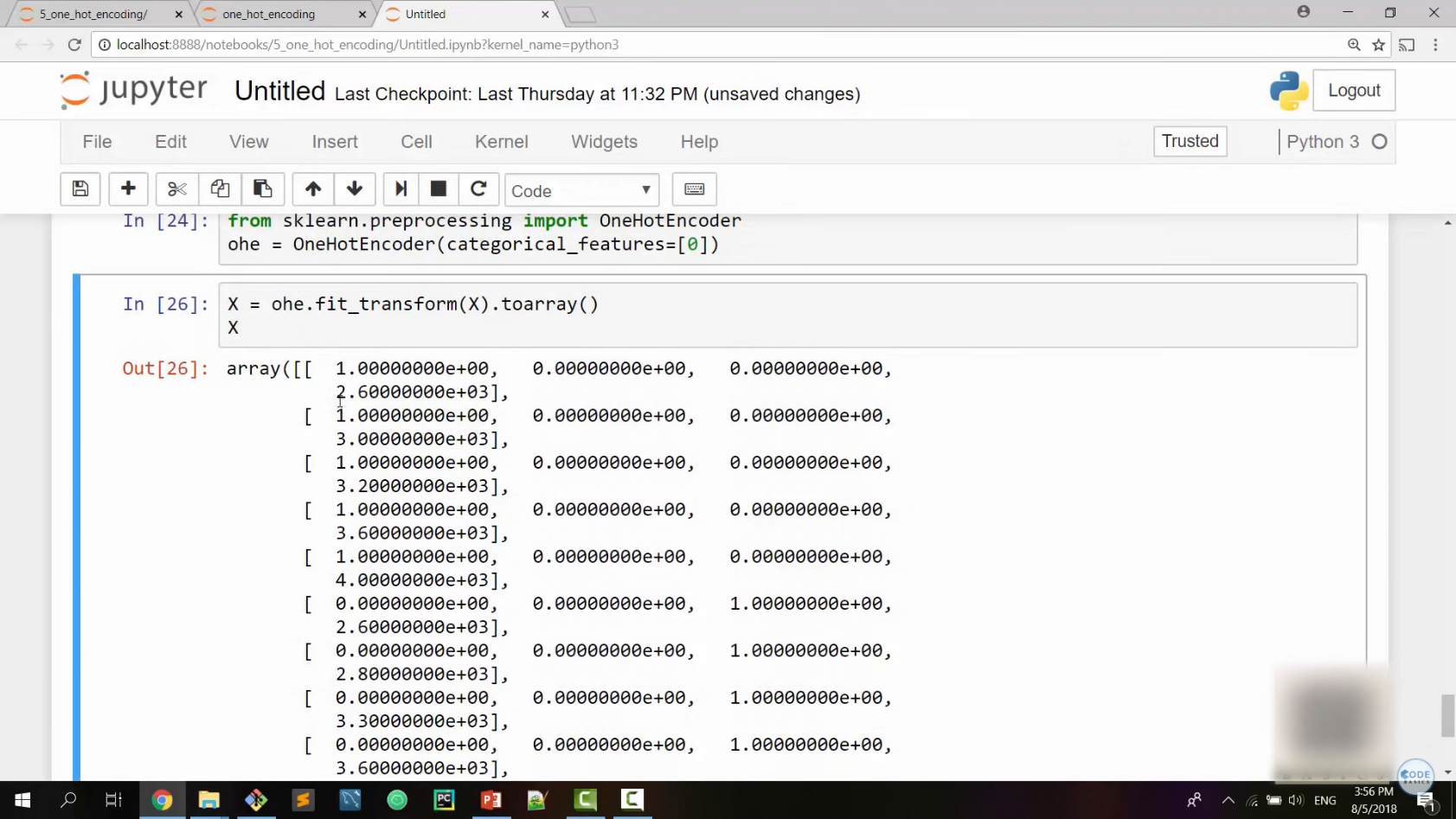
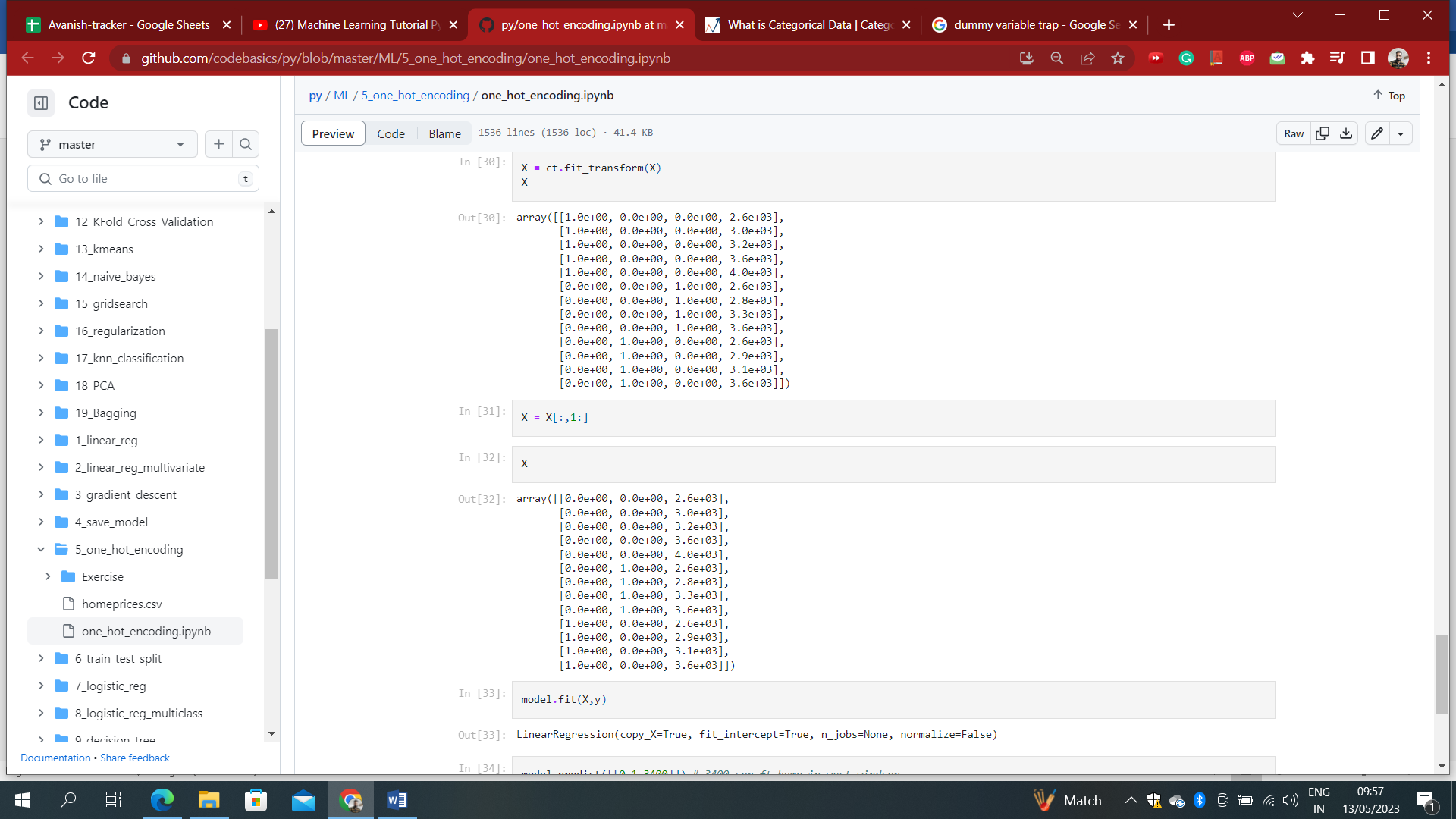
 

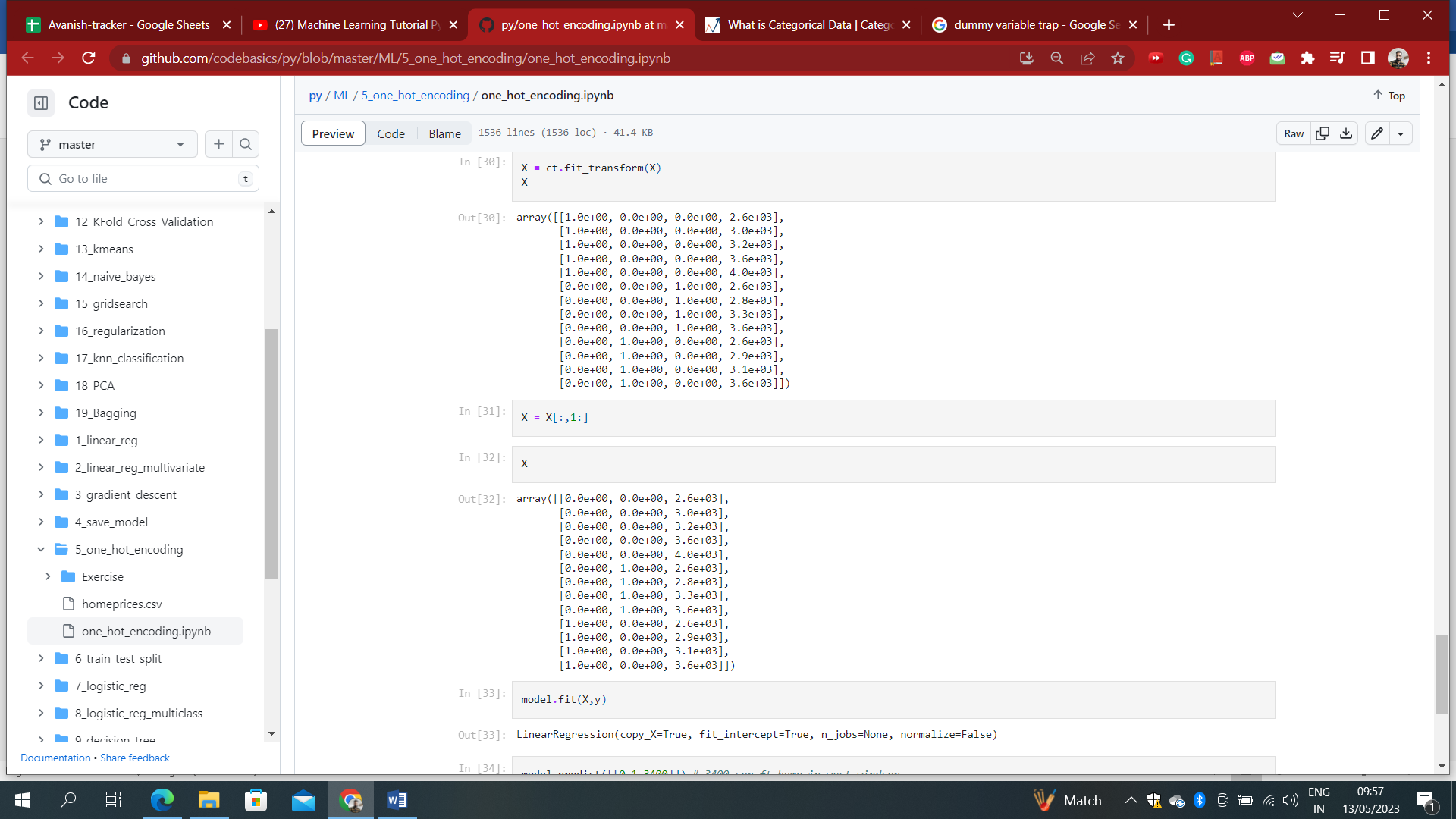
One issue with this representation is that ML algorithms will assume that two nearby values are more similar than two distant values. So, here is the need of one hot encoding to transform all of them into 1(hot) and 0(cold).

## **One Hot Encoding**

One hot encoding in sklearn works similar to dummy encoding in python.

Sklearn librabry takes care of dummy variable trap hence even if you don’t drop one of the state columns it is going to work, however we should make a habit of taking care of dummy variable trap ourselves just in case library that you are using is not handling this for you.





### **Drawbacks of  One-Hot and Dummy Encoding**

One hot encoder and dummy encoder are two powerful and effective encoding schemes. They are also very popular among the data scientists, but may not be as effective when-

1. A large number of levels are present in data. If there are multiple categories in a feature variable in such a case we need a similar number of dummy variables to encode the data. For example, a column with 30 different values will require 30 new variables for coding.
2. If we have multiple categorical features in the dataset similar situation will occur and again we will end to have several binary features each representing the categorical feature and their multiple categories e.g a dataset having 10 or more categorical columns.

In both the above cases, these two encoding schemes introduce sparsity in the dataset i.e several columns having 0s and a few of them having 1s. In other words, it creates multiple dummy features in the dataset without adding much information.

Also, they might lead to a Dummy variable trap. It is a phenomenon where features are highly correlated. That means using the other variables, we can easily predict the value of a variable.

Due to the massive increase in the dataset, coding slows down the learning of the model along with deteriorating the overall performance that ultimately makes the model computationally expensive. Further, while using tree-based models these encodings are not an optimum choice.

**Label Binarizer**

We can apply both transformations (from text categories to integer categories, then from integer categories to one-hot vectors) in one shot using the LabelBinarizer class:

>>> from sklearn.preprocessing import LabelBinarizer

>>> lb = LabelBinarizer()

>>> df\_cat\_1hot = lb.fit\_transform(df[‘town’])

>>> df\_cat\_1hot

array([[0, 1, 0, 0, 0],

[0, 1, 0, 0, 0],

[0, 0, 0, 0, 1],

...,

[0, 1, 0, 0, 0],

[1, 0, 0, 0, 0],

[0, 0, 0, 1, 0]])

Note that this returns a dense NumPy array by default. You can get a sparse matrix instead by passing sparse\_output=True to the LabelBinarizer constructor.