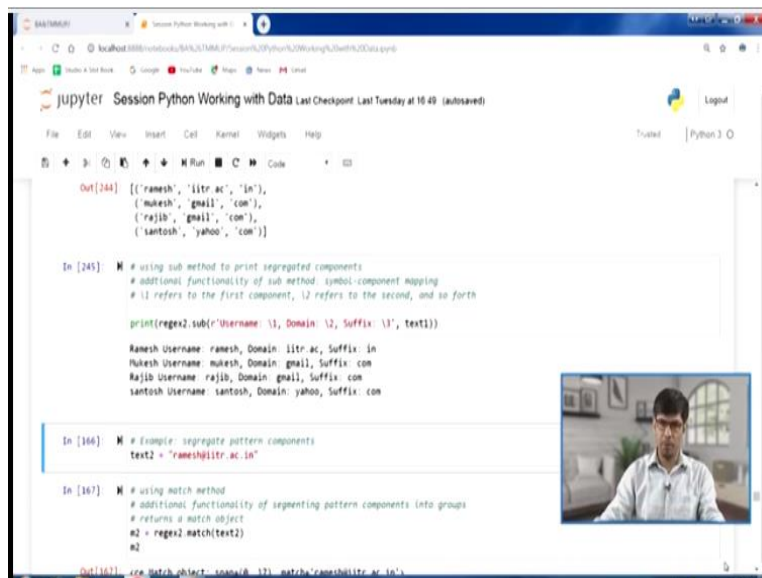


Business Analytics and Text Mining Modeling Using python
Prof. Gaurav Dixit
Department of Management Studies
Indian Institute of Technology Roorkee

Lecture-34
String and Text Processing-Part II

Welcome to the course business analytics and text mining modeling using python. So, in previous lecture we were discussing regular expression and we talked about certain important methods and certain examples of regular expression. And how they can be really useful in terms of you know processing text where you know slightly complex text manipulation is involved.

(Refer Slide Time: 00:48)



```
Out[244]: [('ramesh', 'iitr.ac', 'in'),
          ('mahesh', 'gmail', 'com'),
          ('rajib', 'gmail', 'com'),
          ('santosh', 'yahoo', 'com')]

In [245]: # using sub method to print segregated components
# additional functionality of sub method: symbol-component mapping
# \1 refers to the first component, \2 refers to the second, and so forth

print(regex2.sub("Username: \1, Domain: \2, Suffix: \3", text1))

Ramesh Username: ramesh, Domain: iitr.ac, Suffix: in
Mahesh Username: mahesh, Domain: gmail, Suffix: com
Rajib Username: rajib, Domain: gmail, Suffix: com
Santosh Username: santosh, Domain: yahoo, Suffix: com

In [166]: # Example: segregate pattern components
text2 = "ramesh@iitr.ac.in"

In [167]: # using match method
# additional functionality of segmenting pattern components into groups
# returns a match object
m2 = regex2.match(text2)
m2

Out[167]: <re.Match object; span=(0, 13), match='ramesh@iitr.ac.in'>
```

So, we will continue our discussion from the point where we stopped in the previous lecture, so in the previous lecture we were using a sub method for another kind of functionality where we are able to you know segregate different components of email addresses. And also create a mapping with you know various symbols /1, /2, /3 and mapping with the different components of the email addresses, so we produce that output.

So, let us move forward now we will take another approach, so in this we are using this another example where again we are doing similar kind of things segregate pattern components but the text is different smaller text.

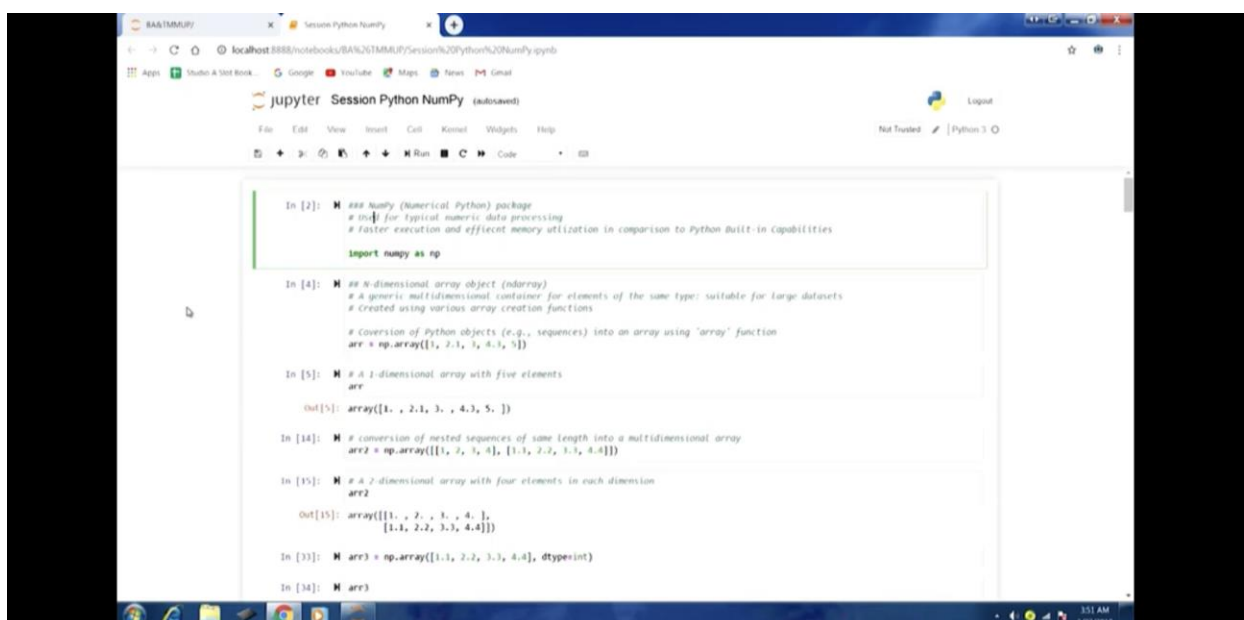
(Videos Starts: 01:31)

This is we are using to demonstrate the match method and how match method can actually be used to achieve the same thing segregation here of pattern components. So, this is for match method also it has this additional functionality of segmenting pattern component into groups. And again this will return a match object and that match object is to be you know processed to actually achieve this segregation.

So, let us call this regex2.match on text 2 and we will have this match object you can see here. Now we can call groups method for this match object and that will actually give us the component. So, we can call m2.groups here and you can see in the output we have got different components in a tuple. So, this in this fashion even match method, so we looked at some method sub and find all method to achieve this.

Now in this particular lecture we saw that how match can also be used to segregate different pattern components. Now we will move forward now we will talk about vectorized string operations, so will focus on sting operation on series and data frame objects. So, let us take example, so what we are going to do here we will create a dict object here, that is comprising of email addresses.

(Refer Slide Time: 02:45)



```
In [2]: # import numpy (Numerical Python) package
# ndarray for typical numeric data processing
# faster execution and efficient memory utilization in comparison to Python built-in capabilities
import numpy as np

In [4]: # # N-dimensional array object (ndarray)
# A generic multidimensional container for elements of the same type; suitable for large datasets
# Created using various array creation functions

# Conversion of Python objects (e.g., sequences) into an array using 'array' function
arr = np.array([1, 2, 3, 4, 5])

In [5]: # # A 1-dimensional array with five elements
arr

Out[5]: array([1, 2, 3, 4, 5])

In [14]: # # conversion of nested sequences of same length into a multidimensional array
arr2 = np.array([[1, 2, 3, 4], [1, 1, 2, 2, 3, 3, 4, 4]])

In [15]: # # A 2-dimensional array with four elements in each dimension
arr2

Out[15]: array([[1, 2, 3, 4],
               [1, 1, 2, 2, 3, 3, 4, 4]])

In [17]: # # arr3 = np.array([1, 1, 2, 2, 3, 3, 4, 4], dtype=int)

In [18]: # # arr3
```

So, we are using key value pair combination here ramesh for example name is key and then email ID is actually the you know value here. So, we can create a dict object like this and let us you know see the output you can see, so this is the dict object now you can use this one to create a series and this is the series that we wanted. Now we will talk about you know vectorized of certain vectorized operation that we can perform.

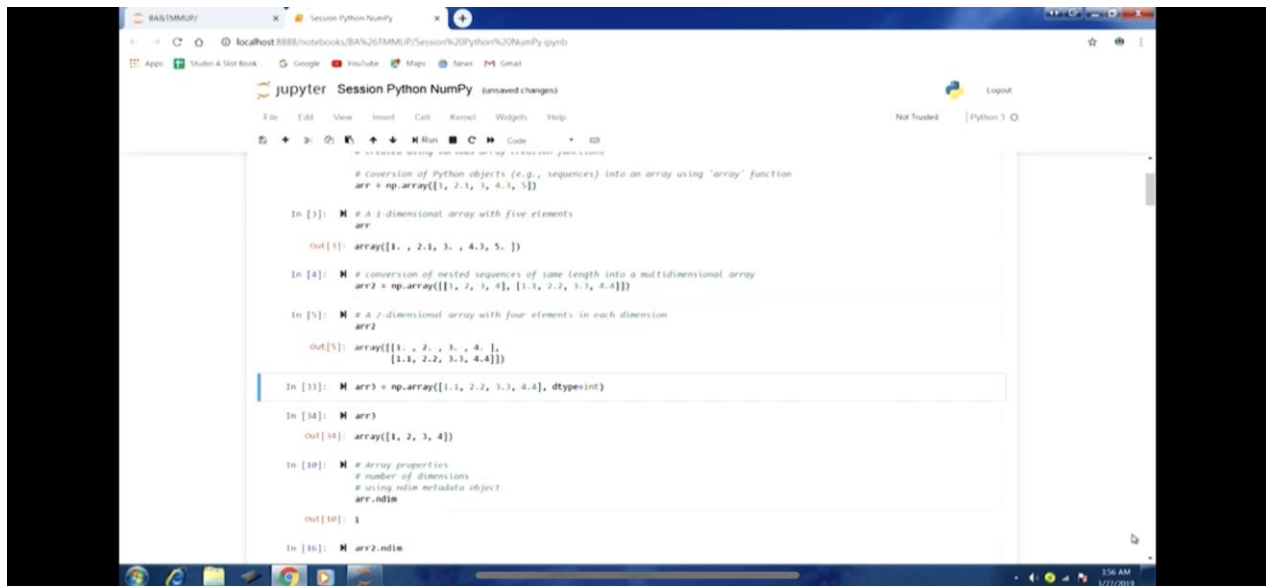
So, on this series if we want to find out whether there are any missing values or not, so in one go in essence a vectorized operation we can call is null you know method here. And in one go for each of the rows that are there in the series object will get the Boolean output whether missing values are present or not. So, if I call this is 3. is null and you can see for each of the row and we will get this output.

So, for one of the rows we have got true, so for one of the rows email addresses missing, in this fashion in a vectorized fashion in a vectorized approach we can actually produce in one go we can find out where the missing values are. Now let us focus on some other aspect which are also related to vectorized operations. So, now we will use the str attribute of series objects to you know call certain methods related to a string operations.

So, advantage of these method is that they skip na, so you know string and regular expression the method if there are nas they might fail. So, which is in the analytics context it might be the case that we might be dealing with a number of nas we might have a number of nas in our data. So, in those situation the methods you know with the str attribute they could be really useful because they do not you know they skip an nas.

So, let us take example we can check whether strings have a particular you know term. So, for this we can use this contains method, it will return a Boolean array, so for all the you know rows that we have in the series object. In one go we can find out using the str.contains method whether a particular term let us say gmail is present in all of the email addresses or not.

(Refer Slide Time: 07:48)



```

# Conversion of Python objects (e.g., sequences) into an array using 'array' function
arr = np.array([1, 2, 3, 4, 5])

In [1]: # A 1-dimensional array with five elements
arr
Out[1]: array([1, 2, 3, 4, 5])

In [4]: # Conversion of nested sequences of same length into a multidimensional array
arr2 = np.array([[1, 2, 3, 4], [1, 2, 3, 4]])

In [5]: # A 2-dimensional array with four elements in each dimension
arr2
Out[5]: array([[1, 2, 3, 4],
               [1, 2, 3, 4]])

In [11]: # arr = np.array([1, 2, 3, 4], dtype=int)

In [14]: # arr
Out[14]: array([1, 2, 3, 4])

In [10]: # Array properties
# number of dimensions
# using ndim metadata object
arr.ndim
Out[10]: 1

In [16]: # arr2.ndim

```

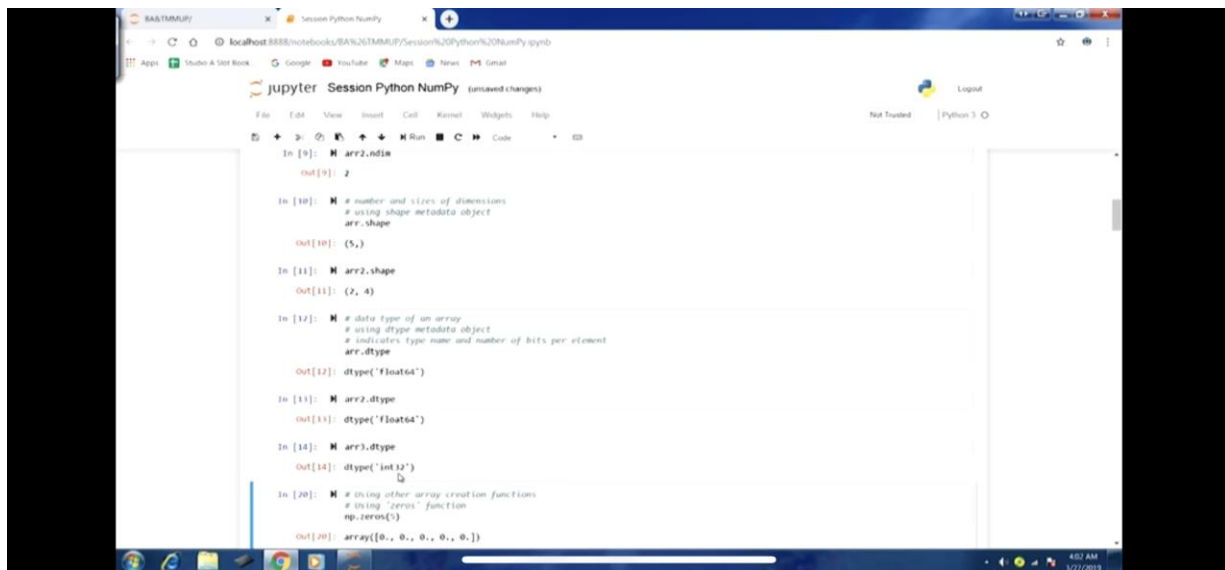
So, I can call like this series3.str.contains and within the parenthesis the argument the **the** term that I want to you know I want to find out whether it is present or not gmail here. So, if I run this you can see first in a row we did not have gmail second, third we had, fourth is was nan, so you can see even with the na values it just worked fine. Now let us again use a regex expression here to segregate components.

Here we are using string literal in this regular expression, so let me define this and here we are calling str.findall methods. So, just like the string method the you know just like their regular expression method we had these methods findall method we had there also for here also the str attribute method we have findall as well. So, we can use this to drip all pattern occurrences in each string, so we can call this method like this series3.str.findall.

So, we can pass on the pattern here in which we justify, so in similar fashion here, so you can see in one go for all the rows in a string we are able to you know segregate we are able to detect all pattern occurrences segregated pattern occurrences here as you can see in the output. Now let us take another example vectorized element access, so whether we would be able to access you know each you know element there, so let us take this example match method.

So, whether we can call this method to find whether easy string matches the pattern, so a boolean output is going to be return here. So, for each of the each of the row that we have any in the series object will you know use this pattern and this method to match whether you know email address whether that is matching the pattern or not, so we can call like this series3.str.match first argument pattern.

(Refer Slide Time: 13:09)

A screenshot of a Jupyter Notebook interface. The notebook is titled "Session Python NumPy" and shows a series of code cells and their outputs. The code cells include: 1) Creating a 2D NumPy array 'arr2' with shape (2, 4) and dtype float64. 2) Printing the shape of 'arr2' as (2, 4). 3) Printing the dtype of 'arr2' as dtype('float64'). 4) Printing the dtype of 'arr' as dtype('int32'). 5) Using the np.zeros() function to create a zero array of shape (0, 0, 0, 0). The outputs are: 2, (2, 4), dtype('float64'), dtype('int32'), and array([0., 0., 0., 0.]).

```
In [9]: arr2.ndim
Out[9]: 2

In [10]: arr2.shape
Out[10]: (2, 4)

In [11]: arr2.dtype
Out[11]: dtype('float64')

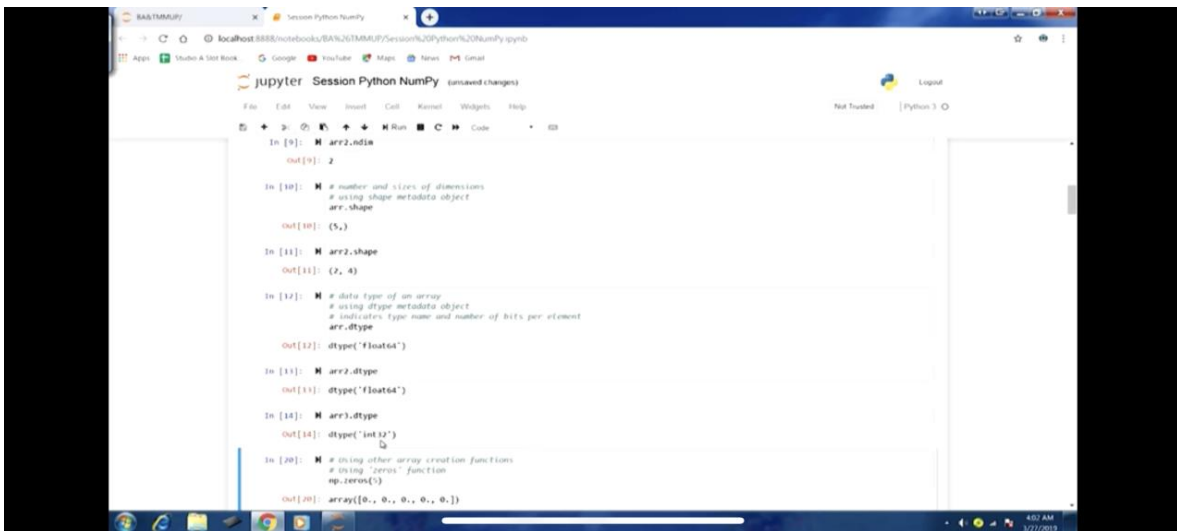
In [12]: arr.dtype
Out[12]: dtype('int32')

In [13]: np.zeros(0)
Out[13]: array([0., 0., 0., 0.])
```

If I run this you can see the Boolean output you know here and you can see that for first 3 address for first 3 rows we have the email address pattern is matching the email addresses. Now another important method get method is there that we can use to actually you know access elements of this string. So, all the string values that we have in the different rows of the series they can actually be accessed using this get method.

So, we can call like this series3.str.get and we can specify 0 to indicate which element in this string we can access, so 0 that means first element. So, if I run this in the output you can see you know the values the email addresses that we had the first character of that in those email addresses has been produced in the output. Similarly we can pass on another index here series3.str.get1, so that means second element in those strings for email addresses should be the output.

(Refer Slide Time: 19:20)



```

In [9]: arr2.ndim
Out[9]: 2

In [10]: # a number and sizes of dimensions
# using shape metadata object
arr.shape
Out[10]: (5,)

In [11]: arr2.shape
Out[11]: (2, 4)

In [12]: # a data type of an array
# using dtype metadata object
# indicates type name and number of bits per element
arr.dtype
Out[12]: dtype('float64')

In [13]: arr2.dtype
Out[13]: dtype('float64')

In [14]: arr3.dtype
Out[14]: dtype('int32')

In [20]: # a using other array creation functions
# using 'zeros' function
np.zeros(5)
Out[20]: array([0., 0., 0., 0., 0.])

```

So, if I run this you can see in the output we have got the second element, similarly we can also use the index you know directly here instead of the get method. So, we can call like series3.str and we can specify the index let us say 2. So, third element in those string values will be the output, so this is how in vectorized accessing elements in a vectorized fashion can be achieved here for each of the rows we are able to produce the output.

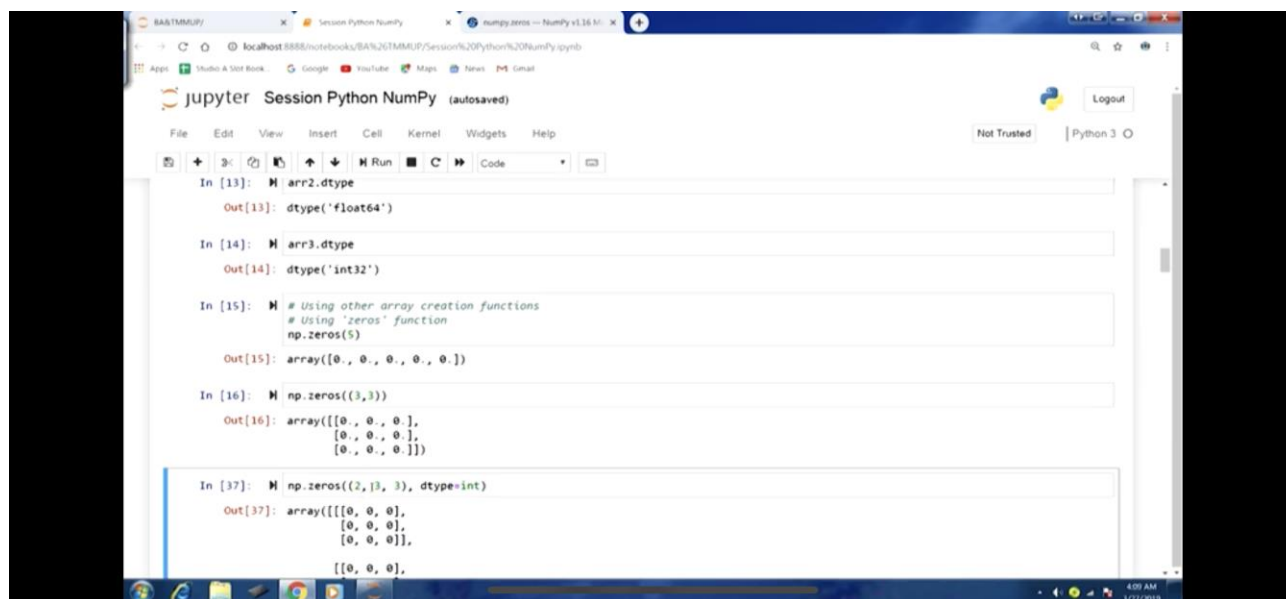
Slicing a string, so again here also we can use the you know earlier syntax that we have been using with other data structures here. Here also a series3.str and within brackets we can specify column3 and only those only up to those characters are going to be kept here. So, if I run this you can see in the output 261 that we are left with just you know 3 characters in each of the row, now that was about a string and text processing.

Now we move to the next aspect that is categorical type, so just like in our other courses business analytics data mining modeling using R where we talk about the categorical type. And we talked about how R platform can actually you know facilitates conversion of you know variables which are typically you know which would typically have values either in the string form or the numeric value type or the string value type to actually convert them into a categorical variable.

So, just like R platform in python also we have this categorical type and it has certain benefits

also. So, it can yield significant performance improvements for analytics context, so typically use for holding data that uses the integer based categorical representation, so that data can actually be used. Binning function that we have previously discussed cut and qcut, so those functions also written categorical objects.

(Refer Slide Time: 23:43)



```
In [13]: arr2.dtype
Out[13]: dtype('float64')

In [14]: arr3.dtype
Out[14]: dtype('int32')

In [15]: # Using other array creation functions
# Using 'zeros' function
np.zeros(5)
Out[15]: array([0., 0., 0., 0., 0.])

In [16]: np.zeros((3,3))
Out[16]: array([[0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.]])

In [37]: np.zeros((2,3,3), dtype=int)
Out[37]: array([[[0, 0, 0],
                [0, 0, 0],
                [0, 0, 0]],
                [[0, 0, 0],
                [0, 0, 0],
                [0, 0, 0]])]
```

But at that time we did not focus on those you know a categorical type aspect of those functions. Now in this particular section we will talk about the categorical type. So, let us take an example, let us take a basket of you know fruits where we have this variable list variable fruits list of you know different fruits are mentioned in the values part apple orange, apple apple, we are multiplying by 2.

So, we will have more number of elements here and we are recording a length of this fruits variable this list very variable here also. And then we are defining a data frames, if you look at the data frame in the first you know column is fruit and there we are using this fruit list variable for values then basket id, np arrange then per count you know we will taking random numbers and then uniform distribution for weight.

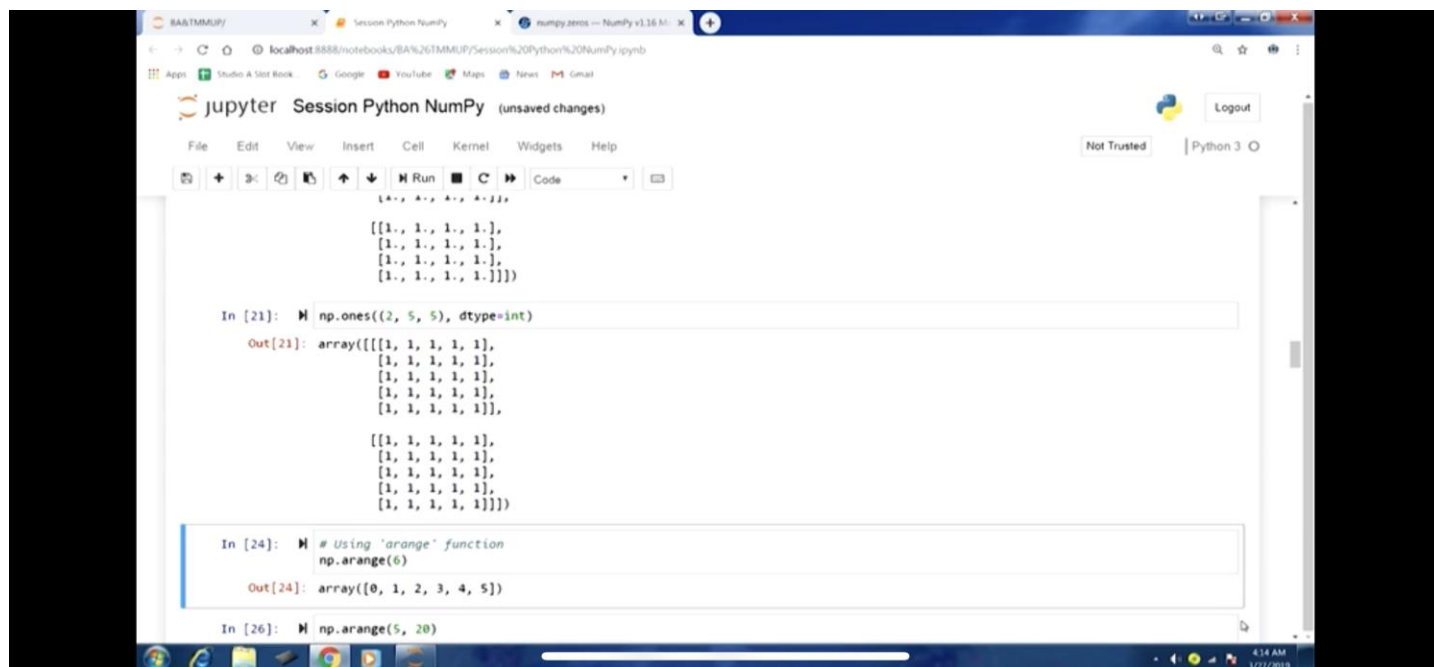
So, we will have and column we are also specifying the column names here using the columns argument. So, if I run this in the output we will have this data frame you can see basket id and

fruit count and weight. So, we will focus on the fruit variable that fruit column that we have because we will focus on how we can actually if you look at the fruit column it is filled with string values.

And we look at the unique or distinct values that are present in the fruit column, so these are only 2 apple and orange. So, this fruit variable can be treated as a categorical variable, so for more understanding on categorical variable you can refer to my previous course business analytics and data mining modeling using R. However here our focus is on how we can convert this fruit variable which is actually having a string values here into a categorical type.

So, converting fruit column into a categorical type, we can use astype method and there we can specify the type. So, fruit_cat this new variable that will create d15 and the column fruit and we are calling this method .astype and passing on the first argument the type category. So, this will actually convert this column into a categorical type of higher if I run this will have fruit_cat and if you look at the output you will see that name is there fruit dtype data type is category you can see that has changed and categories are also mentioned.

(Refer Slide Time: 28:18)



```
[[1., 1., 1., 1.],
 [1., 1., 1., 1.],
 [1., 1., 1., 1.],
 [1., 1., 1., 1.]])

In [21]: M np.ones((2, 5, 5), dtype=int)
Out[21]: array([[[[1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1]],
 [[1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1]]]])

In [24]: M # Using 'arange' function
np.arange(6)
Out[24]: array([0, 1, 2, 3, 4, 5])

In [26]: M np.arange(5, 20)
```

So, 2 categories are there apple, orange, so you can see this you know fruit you know column

that we had with the string values. Now it has been converted into a categorical type with 2 categories apple and orange. Now as we know that we have values attribute you know, so in this case this values attribute of this variable is actually a categorical object. So, `fruits_cat.values`, so if you look at this you can see this list of these labels apple orange, apple apple, apple.

And so all these labels are there and you can see the 2 categories there, so this is a is values attribute is a you know categorical object. You can look at you can use the `type` function to find out you can pass on the `cat1` and find out the type of object and you can see output 266 it is a categorical one. Now let us talk about the attributes of a categorical objects, so there are 2 important attributes that will discuss categories and codes.

So, in the `categories` attribute as the name says will have the list of you know categories that are there in that categorical variable or object. And in the `codes` will have the you know the values that are there in that variable. So, all those would be coded using numeric values, so will have numeric codes in a sense for all the values that are present and for that for you know variable.

So, let us first execute `cat1.categories`, if I run this you can see in the output apple and orange we have these 2 categories. Similarly `cat1.codes` and you can see 0, 1, 0, 0, 0 and 1, 0, 0, so you can see that apple. Now if you can compare this output 268 with 265 and apple is you know apple is for apple we are using numeric code 0 and for orange we are using numeric code 1.

So, it is just based on the ordering because in the `categories` ordering it was alphabetical happen an orange and here the ordering in the numerical fashion 01 in that way. So, therefore apple is associated with 0 and orange is associated with 1 here. Now we can if we want to directly you know convert the data frame column instead of recording that column into another variable, so we can do that also.

So, we can type like this `d15` within brackets `fruit` and then on the right hand side `d15fruit.astype category`. So, in this fashion will be directly in the data frame itself that particular column will become a categorical type. So, if I run this you can see the similar output now but the output this particular you know change this work particular converge into categorical type is going to be

reflected in the data frame itself.

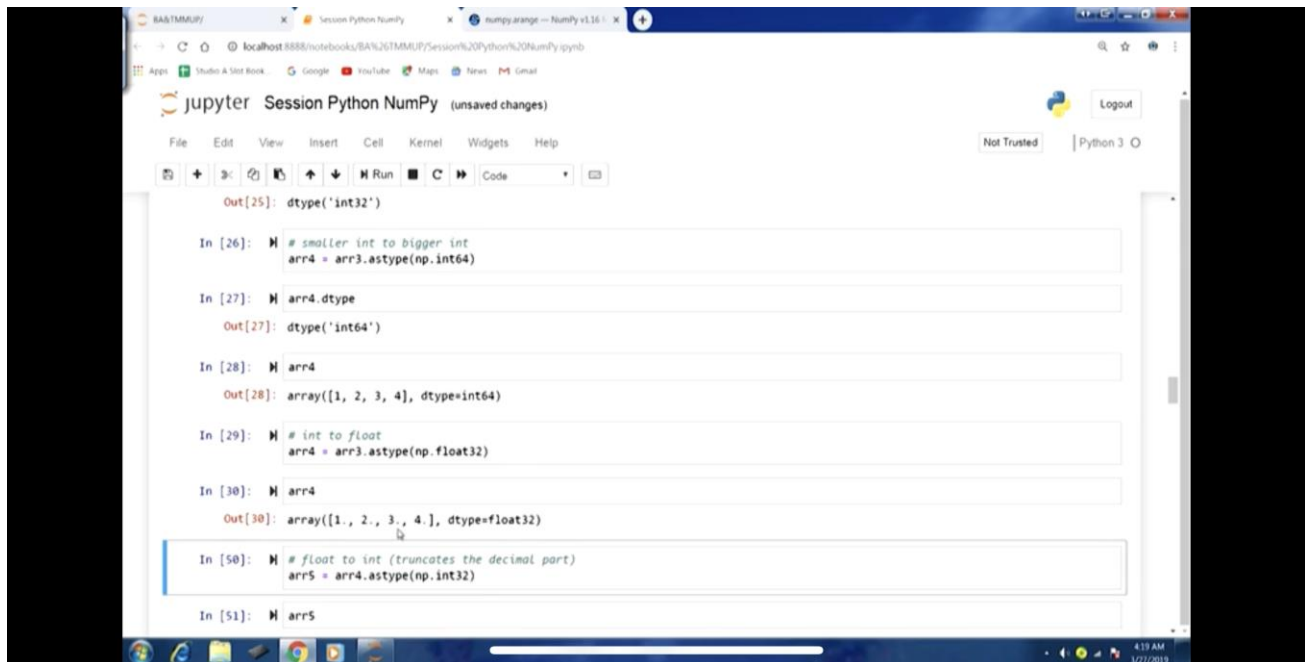
Now let us talk about how do we create a categorical object, so there are certain mechanisms which can be used to create categorical objects. So, first one is from sequences, so we can use categorical function to actually create categorical objects from sequences. So, let us take the sequence where we have a list of string values a young, young adult, mid aged young, young adult.

So, you can see there are 2, 3 you know distinct values here 3 distinct values to be precise and you know, so this is a categorical variable this with you know the categorical variable with 5 values 5 rows. We can use categorical function here to actually create this categorical object and if I access this cat2 you can see that on the values and the categories 3 mid aged young and young adult.

So, using categorical function we can create a categorical object from a sequence it could be the categorical variable with all the values that are there. Now let us take another example, so one thing is passing on the sequence of values and the categorical function would be able to you know extract the distinct values and create you know categories from there.

Then another approach could be that we can define categories and codes separately and then use another function which is categorical from codes. So, there we can pass on these 2 separately codes and categories to create a categorical object. So, in this case we are defining categories as these 3 C, B, A 3 categories and we have codes like 0, 1, 2, 0, 0, 1. So, we can use this categorical from codes function and pass on these 2 argument to create this you know categorical object.

(Refer Slide Time: 31:30)



```
Out[25]: dtype('int32')

In [26]: # smaller int to bigger int
arr4 = arr3.astype(np.int64)

In [27]: # arr4.dtype
Out[27]: dtype('int64')

In [28]: # arr4
Out[28]: array([1, 2, 3, 4], dtype=int64)

In [29]: # int to float
arr4 = arr3.astype(np.float32)

In [30]: # arr4
Out[30]: array([1., 2., 3., 4.], dtype=float32)

In [50]: # float to int (truncates the decimal part)
arr5 = arr4.astype(np.int32)

In [51]: # arr5
```

If you look at cat3 you can see that C is if you compare you know the output then C is associated with 0 then you know B is associated with the 1 and then A is associated with 2. If you look at the categories output in output 194 you can see C, B, A and the values you can see the code that we had 0, 1, 2, 0, 0, 1 those have been replaced as well. So, in this fashion you know we can use this function and create a categorical object.

Till now these 2 examples that we have taken where we have created categorical objects, these are categorical variables of one particular type which is referred as nominal variable again nominal variable for more details you can refer to my previous course. And so here till now we talked about nominal my level variable, now it is also possible to create a ordinal variable where the categories that will have, they will have meaningful order among themselves.

So, creating a categorical object with a specific ordering of the categories, so essentially an ordinal variable. So, for that we can use the ordered argument, so in the function categorical form course which we just now in the previous line of code we had used. There we can pass on third you know argument ordered and we can specify true. So, whatever categories are there a particular specific ordering is going to be you know implemented there.

So, if I run this you can see in the output 275 categories 3 and you can see $C < V < A$, so you can see in this fashion a particular ordering among categories is also there. So, we have created essentially an ordinal variable using this particular argument ordered. Now for other function also pd categorical for this also we can other example that we had there also if we can specify the ordered argument then using this also we would be able to create a ordinal object ordinal variable.

Refer Slide Time: 33:35)

```

Out[34]: array(['1.1', '-5.5', '9'], dtype='<U4')

In [35]: # string to float
# NumPy aliases the Python types to its own equivalent data types
arr6.astype(float)

Out[35]: array([ 1.1, -5.5,  9. ])

In [36]: # arr7 = np.array(['1.123456789123456789', '-5.5', '9'])
arr7

Out[36]: array(['1.123456789123456789', '-5.5', '9'], dtype='<U20')

In [75]: # string to float (truncates value)
arr7.astype(np.float)

Out[75]: array([ 1.12345679, -5.5,  9. ])

In [77]: # arr7.astype(np.float).dtype
Out[77]: dtype('float64')

In [71]: # arr7 = np.array([1.123456789123456789, -5.5, 9])

In [72]: # arr7.astype(np.string_)

```

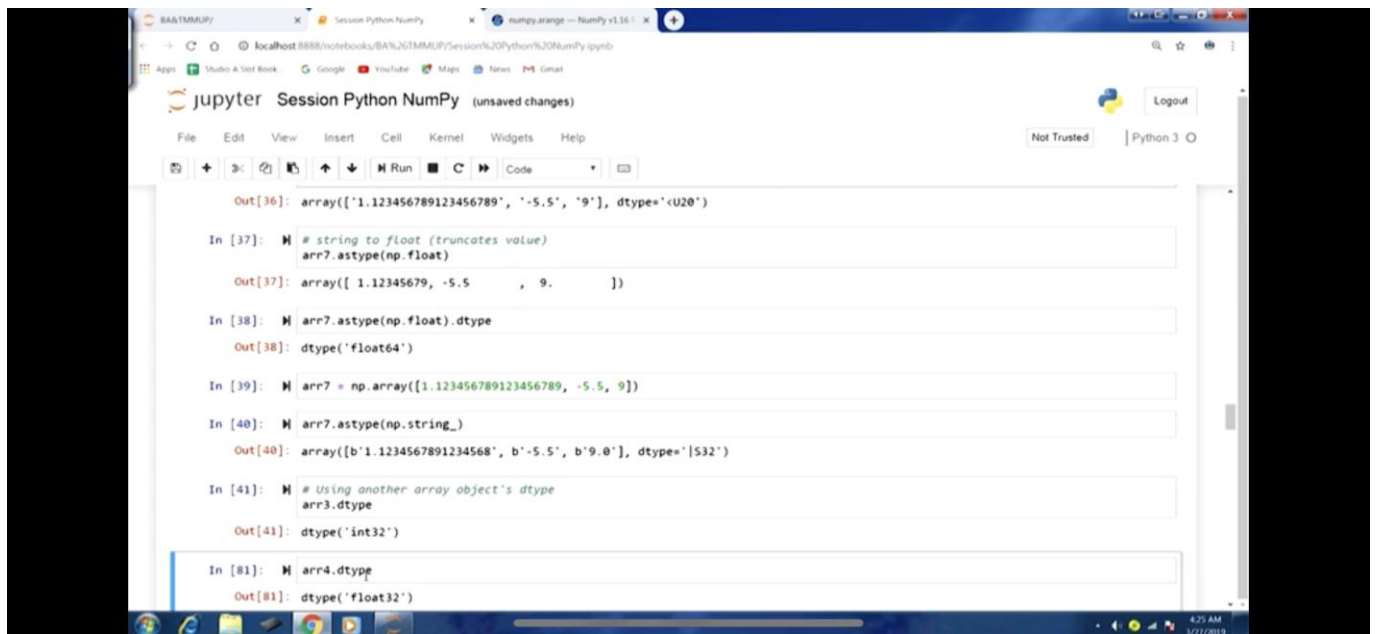
So, cat5 you can have a look, here also you can see mid aged then young then young adult. So, you can see this in this fashion you know a particular specific ordering has been implemented here. Now let us talk about you know converting so many times we will have the categorical variable the nominal variable unordered categorical variable which is nominal variable. And you would like to convert it into an ordered categorical variable which is ordinal variable, so for that we can use `as_ordered` method here.

So, we can call `cat2.as_order`, so `cat2` we saw previously we had created that was an unordered categorical variable. So, we can call this method and convert into 1 as ordered in ordinal variable, so I run this and you can see in the output 278 and you can look at the categories. And you can see the `<` sign is there mid aged less than young then yes young adult, so and a specific order has been implemented there.

Now till now if you have a look at typically we have been using strings for to a specify the indicate the categorical data. However this is non necessity we can use any immutable value type to actually create a categorical object. So, we will take another example and we are calling categorical function and we are passing a list of values which are actually numeric values.

So, these values also we can use to create a categorical variable here, so let me run this and if you look at the output you can see the values are like -1, 0, 0, 1, -1 and you can look at the categories. So, unique distinct values -1, 0, 1, so it is not necessary that data categorical data has to be in the string format it can be numerical also it should be immutable value type. Now this was about the categorical type and there are certain associated methods also categorical methods also.

(Refer Slide Time: 34:00)



```
Out[36]: array(['1.123456789123456789', '-5.5', '9'], dtype='<U20')

In [37]: # string to float (truncates value)
arr7.astype(np.float)

Out[37]: array([ 1.12345679, -5.5, 9.])

In [38]: # arr7.astype(np.float).dtype
Out[38]: dtype('float64')

In [39]: # arr7 = np.array([1.123456789123456789, -5.5, 9])

In [40]: # arr7.astype(np.string_)
Out[40]: array([b'1.1234567891234568', b'-5.5', b'9.0'], dtype='|S32')

In [41]: # Using another array object's dtype
arr3.dtype

Out[41]: dtype('int32')

In [81]: # arr4.dtype
Out[81]: dtype('float32')
```

So, sometimes they are going to be really useful, so let us take an example you know for example if we want to reset the categories that are there or we would like to add a category. So, for that we can use set_categories method, so let us say cat3, so this we had created earlier and you can see in the output to weight 281, we have category, 3 categories CBA and we can add 1 or more category here using set categories methods.

So, we can call `cat3.set_categories` and you can see we have change the ordering also ABC and then added another category d here. So, if I run this in the next output to a 283 you can see we have 4 categories ABCD in this fashion you can see we can modify or add categories using this particular `set_categories` method. Now if you want to look at the counts, so of course because we have added a category but the associated values are not there in the variable.

So, if we call this method `value_count`, so the frequencies for the fourth category that we have just now added D, it is going to be 0. So, A is present one time, B is twice, C thrice, D is not present, C is not present any you know in once, so this is how we can add few more categories. Now let us talk about trimming you know unobserved categories, so when we are typically when we are dealing a larger data sets and there are you know a variable with a number of categories and you know some of those categories if they are not present.

Then certain situation we would like to trim those unobserved categories. So, for that we have another method `_remove_unused_categories` method, so we can call this. So, in the previous example we had added 1 category d, so which was unused because there were no values present you know associated with that particular category. So, we can remove that one, so `cat.7 remove` and `respond use categories`, so it will find any such categories and trim it off.

So, if you look at the output 285 again we are left with just 3 categories ABC. Now let us move forward, now the next thing is about creating dummy variables. So, this part we have discussed before also but in the context of categorical type at categorical objects that we have just discussed, we will go through this again. So, let us take `cat3`, so in this particular categorical object we had these values these 6 values and 3 categories CBA.

Now we can call the `pandas.get_dummies` function here and we can pass on this `cat3` this particular object here, categorical object here. And it will create so it has 3 categories there, so using `get_dummies` you know method function here will get 3 dummy variables in a data frame. And I am using prefix argument as well, so all those categories would be prefixed with the variable name `cat3`.

So, if I run this just like we have done before you can see in the output that we have got 3 columns `cat3_C`, `cat3_B` and `cat3_A` and dummy variables has been created presence or absence of that category in the variable, so you can see that here. So, this concludes our discussion on working with data, so various aspects related to data management we have been able to cover.

So, now we will move to another important aspect that is related to you know visualization techniques R plotting in the python environment. So, 1 or 2 examples we have seen before also, so in this lecture itself we would like to briefly touch upon this more details will come in the next lecture. So, first thing is whenever we would like to create plots in the path python environment.

So, the most popular you know library package is `matplotlib`, so we will talk about that even the `pandas` has certain you know features certain functions to achieve this. So, we will talk about those also, so let us start with the `matplotlib`, so to be able to use this macro not `matplotlib` in the you know Jupyter environment. We will have to use this magic `(%)` (28:01) `matplotlib` notebook however in newer versions it might not be required.

So, let me run this and then the require library modules `matplotlib.pyplot` as `plt` and because we would be using a lot of you know `numpy` functions, so let us load that one also. Now let us take a simple example we will try to create a simple plot, so first we will generate data for that, we will take an array `Array1d np.arange` function and 10 values for this array, you can see the output.

So, we can use as we have discussed zone before also, the function is `plt.plot` we can pass on this variable array `1d`. So, this is you know if we do not specify the we do not specify the variable for the other access. So, we need to specify for both the access if it we do not specify for the other one the whatever is possibly copied over for both the access. So, if I run this you can see that we have generated one plot you can see this is a line passing through you know origin `00` and this plot has been generated in this fashion.

So, this is how the plotting will work in this using this purple library. Now we will talk about few more aspect here, for example figure object, so this particular plot which we just now generated it is actually created in the frame which is defined by this object figure object. So,

whenever we are looking to generate more number of plots more number of subplots will have to work with the figure object you know where these plots will typically reside.

So, how do we create this, so creating a figure object we can call this figure function, so `plt.figure` and will get a figure object which we can you know later on use to create more number of plots. So, let me run this and will have `fig` object and you can see once we run this we have a empty figure here without any plots, so let us move forward. Now we will have to add access to the figure object, so for that we can use `add_subplot` method.

So, in this we have you know **2** first and second argument they will actually be use to specify the grid. So, across how many rows we want and how many columns we want, so that will also define how many subplots you would be adding. So, first and second argument will actually specify that number of rows and columns in the figure object. So, let us say you know we want 4 plots we can have 2×2 configuration or it could be you know 1×4 configuration as well.

So, let us go with 2×2 here the third argument that is typically use to specify the index position in the grid. So, we can call like this `fig.fig` object that we had created.`add_subplot`, first argument 2 number of rows, second argument 2 number of column and then the index position. So, if I run this and if I go here in the output figure 2 you can see in the index position 1 a plot subplot has been added and added with respect to about this configuration to do.

So, in the same figure we can you know add another one, the next you know index and another one at the next index and we can have a look at the in all 3. So, you can see that we are added subplots here, so XC essentially XC is have been added for these 3 subplots. Let us move forward now we can go ahead and you know if we have any data we can use this `fig` object to you know create those plots.

So, if we do not specify further the by default the last subplot that we have used that will be use to generate another plot.

(Video Starts: 32:29)

So, we will stop here and in the next lecture we will use some data and we will see how we can actually generate these plots using figure object, thank you.

Keywords: Numerical python, multiplication, data mining modeling, attributes, Extraction, Error handling, String, Categorical value, Qualitative and Quantitative techniques.