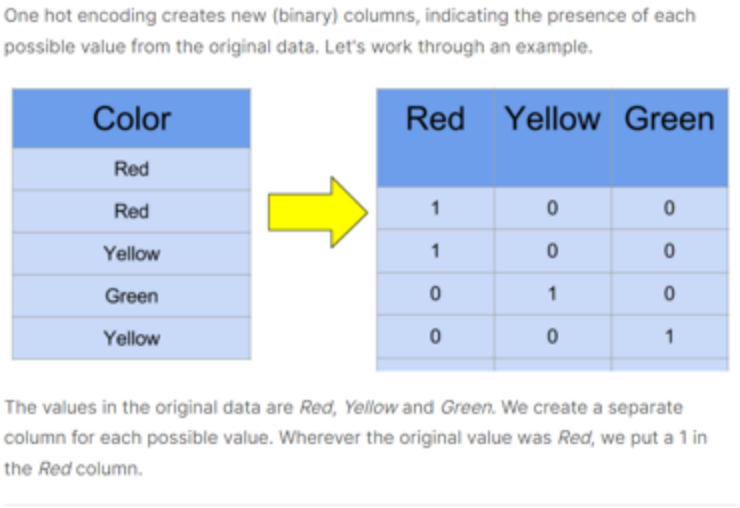
One - Hot Encoding

Categorical data is data that takes only a limited number of values.

You will get an error if you try to plug these variables into most machine learning models in Python without "encoding" them first

One-Hot Encoding : The Standard Approach for Categorical Data

One hot encoding is the most widespread approach, and it works very well unless your categorical variable takes on a large number of values (i.e. you generally won't it for variables taking more than 15 different values. It'd be a poor choice in some cases with fewer values, though that varies.)



Alternatively, you could have dropped the categoricals. To see how the approaches compare, we can calculate the mean absolute error of models built with two alternative sets of predictors:

- One-hot encoded categoricals as well as numeric predictors
- Numerical predictors, where we drop categoricals.

https://www.kaggle.com/discussions/getting-started/114797 (https://www.kaggle.com/discussions/getting-started/114797)

Categorical Encoding refers to transforming a categorical feature into one or multiple numeric features. You can use any mathematical method or logical method you wish to transform the categorical feature, the sky is the limit for this task.

Again, you usually let your favorite programming language doing the work. Do not loop through each categorical value and assign a column, because this is NOT an efficient at all. It is not difficult, right?

one hot encoding is not a good option for tree based modeling, specially for lgb or xgboost. It will create too many variables and boosting algorithms with comparatively less depth would be more confused to figure out the best splits. Although in this competition, different encoding techniques for categorical variables, like target encoding or label encoding can bring diversity

When to use Label Encoding Vs One-Hot Encoding?

This question generally depends on your dataset and the model which you wish to apply. But still, a few points to note before choosing the right encoding technique for your model:

- We apply One-Hot Encoding when :

- The Categorical Feature is not Ordinal
- The Number of Categorical features is less so One-Hot Encoding can be effectively applied

- We apply Label Encoding when :

- The Categorical feature is ordinal (like Executive , Manager, Senior Manager, CEO)
- The number of categories is quite large as One-Hot Encoding (dummy variables) can lead to high memory consumption

Also, One Hot Encoding may lead to a curse of dimensionality, it is creating complications for parallelism and multicollinearity.

Label encoder converts string to numeric and one hot encoder converts distinct values to new column Is it correct??

Label Encoding is a technique that is used to convert categorical columns into numerical ones so that they can be fitted by machine learning .

Handling Non-Numeric Categorical data

In [50]:
```
import pandas as pd
import numpy as np
data1 = pd.read_csv('suicide_data.csv')
```

In [3]:
```
1 data1.head(2)
```

Out[3]:

	country	year	sex	age	suicides_no	population	suicides/100k pop	country-year
0	Albania	1987	male	15-24 years	21	312900	6.71	Albania1987
1	Albania	1987	male	35-54 years	16	308000	5.19	Albania1987

In [4]:
```
1 data1.shape
```

Out[4]: (27820, 12)

In [7]:
```
1 data1.info(verbose = False)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27820 entries, 0 to 27819
Columns: 12 entries, country to generation
dtypes: float64(2), int64(4), object(6)
memory usage: 2.5+ MB
```

- We have 6 categorical variables

In [9]:
```
1 data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27820 entries, 0 to 27819
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   country             27820 non-null  object 
 1   year                27820 non-null  int64  
 2   sex                 27820 non-null  object 
 3   age                 27820 non-null  object 
 4   suicides_no         27820 non-null  int64  
 5   population          27820 non-null  int64  
 6   suicides/100k pop   27820 non-null  float64
 7   country-year        27820 non-null  object 
 8   HDI for year        8364 non-null   float64
 9    gdp_for_year ($)   27820 non-null  object 
 10  gdp_per_capita ($)  27820 non-null  int64  
 11  generation          27820 non-null  object 
dtypes: float64(2), int64(4), object(6)
memory usage: 2.5+ MB
```

- country , sex , age , country-year , gdp_for_year ($) , generation

In [10]:
```
1 data1.select_dtypes(include = 'object')
```

Out[10]:

	country	sex	age	country-year	gdp_for_year ($)	generation
0	Albania	male	15-24 years	Albania1987	2,156,624,900	Generation X
1	Albania	male	35-54 years	Albania1987	2,156,624,900	Silent
2	Albania	female	15-24 years	Albania1987	2,156,624,900	Generation X
3	Albania	male	75+ years	Albania1987	2,156,624,900	G.I. Generation
4	Albania	male	25-34 years	Albania1987	2,156,624,900	Boomers
...	...	...	...	...	...	...
27815	Uzbekistan	female	35-54 years	Uzbekistan2014	63,067,077,179	Generation X
27816	Uzbekistan	female	75+ years	Uzbekistan2014	63,067,077,179	Silent
27817	Uzbekistan	male	5-14 years	Uzbekistan2014	63,067,077,179	Generation Z
27818	Uzbekistan	female	5-14 years	Uzbekistan2014	63,067,077,179	Generation Z
27819	Uzbekistan	female	55-74 years	Uzbekistan2014	63,067,077,179	Boomers

27820 rows × 6 columns

In [11]:
```
1 data1.select_dtypes(include = 'object').columns
```

Out[11]:
```
Index(['country', 'sex', 'age', 'country-year', ' gdp_for_year ($) ',
       'generation'],
      dtype='object')
```

In [12]:
```
1 #looking at the number of unique values in each
2 Categorical_columns = data1.select_dtypes(include = 'object').columns
```

In [16]:
```
for col in Categorical_columns:
    print(col , "==>",data1[col].unique())
```

```
country ==> ['Albania' 'Antigua and Barbuda' 'Argentina' 'Armenia' 'Aruba'
'Australia'
 'Austria' 'Azerbaijan' 'Bahamas' 'Bahrain' 'Barbados' 'Belarus' 'Belgium'
 'Belize' 'Bosnia and Herzegovina' 'Brazil' 'Bulgaria' 'Cabo Verde'
 'Canada' 'Chile' 'Colombia' 'Costa Rica' 'Croatia' 'Cuba' 'Cyprus'
 'Czech Republic' 'Denmark' 'Dominica' 'Ecuador' 'El Salvador' 'Estonia'
 'Fiji' 'Finland' 'France' 'Georgia' 'Germany' 'Greece' 'Grenada'
 'Guatemala' 'Guyana' 'Hungary' 'Iceland' 'Ireland' 'Israel' 'Italy'
 'Jamaica' 'Japan' 'Kazakhstan' 'Kiribati' 'Kuwait' 'Kyrgyzstan' 'Latvia'
 'Lithuania' 'Luxembourg' 'Macau' 'Maldives' 'Malta' 'Mauritius' 'Mexico'
 'Mongolia' 'Montenegro' 'Netherlands' 'New Zealand' 'Nicaragua' 'Norway'
 'Oman' 'Panama' 'Paraguay' 'Philippines' 'Poland' 'Portugal'
 'Puerto Rico' 'Qatar' 'Republic of Korea' 'Romania' 'Russian Federation'
 'Saint Kitts and Nevis' 'Saint Lucia' 'Saint Vincent and Grenadines'
 'San Marino' 'Serbia' 'Seychelles' 'Singapore' 'Slovakia' 'Slovenia'
 'South Africa' 'Spain' 'Sri Lanka' 'Suriname' 'Sweden' 'Switzerland'
 'Thailand' 'Trinidad and Tobago' 'Turkey' 'Turkmenistan' 'Ukraine'
 'United Arab Emirates' 'United Kingdom' 'United States' 'Uruguay'
 'Uzbekistan']
sex ==> ['male' 'female']
age ==> ['15-24 years' '35-54 years' '75+ years' '25-34 years' '55-74 years'
 '5-14 years']
country-year ==> ['Albania1987' 'Albania1988' 'Albania1989' ... 'Uzbekistan2
012'
 'Uzbekistan2013' 'Uzbekistan2014']
 gdp_for_year ($)  ==> ['2,156,624,900' '2,126,000,000' '2,335,124,988' ...
'51,821,573,338'
 '57,690,453,461' '63,067,077,179']
generation ==> ['Generation X' 'Silent' 'G.I. Generation' 'Boomers' 'Milleni
als'
 'Generation Z']
```

In [19]:
```
data1.shape
```

Out[19]:
```
(27820, 12)
```

In [18]:
```
for col in Categorical_columns:
    print(col , '==>' , data1[col].nunique())
```

```
country ==> 101
sex ==> 2
age ==> 6
country-year ==> 2321
 gdp_for_year ($)  ==> 2321
generation ==> 6
```

- One Hot Encoding can be applied to "sex" , 'age' , 'generation'

One Hot Encoding

Most of the Machine Learning Models are designed to work on numeric data. Hence , we need to convert categorical text data into numerical data for Model building.

A dummy variable replaces the categorical variable with a numerical variable.For each category , a Dummy variable will be created. The Dummy variable will have two values : 1 or 0 , based on whether that particular value is present in the row/record or not.

Its encoding of categorical variables into 0 or 1.

In [23]:
```
df_dummies = pd.get_dummies(data1, prefix = "Gender", columns = ['sex'])
```

In [26]:
```
df_dummies.head(5)
```

Out[26]:

	country	year	age	suicides_no	population	suicides/100k pop	country-year	HDI for year
0	Albania	1987	15-24 years	21	312900	6.71	Albania1987	NaN
1	Albania	1987	35-54 years	16	308000	5.19	Albania1987	NaN
2	Albania	1987	15-24 years	14	289700	4.83	Albania1987	NaN
3	Albania	1987	75+ years	1	21800	4.59	Albania1987	NaN
4	Albania	1987	25-34 years	9	274300	3.28	Albania1987	NaN

In [28]:
```
df_dummies.columns
```

Out[28]:
```
Index(['country', 'year', 'age', 'suicides_no', 'population',
       'suicides/100k pop', 'country-year', 'HDI for year',
       ' gdp_for_year ($) ', 'gdp_per_capita ($)', 'generation',
       'Gender_female', 'Gender_male'],
      dtype='object')
```

In [30]:
```
df_dummies = df_dummies[['country', 'year', 'age', 'Gender_female', 'Gend
       'suicides/100k pop', 'country-year', 'HDI for year',
       ' gdp_for_year ($) ', 'gdp_per_capita ($)', 'generation']]
```

In [31]:
```
df_dummies = df_dummies.copy()
df_dummies['sex'] = data1['sex']
```

```
<ipython-input-31-9ad589dd45f3>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_dummies['sex'] = data1['sex']
```

In [32]:
```
df_dummies.columns
```

Out[32]:
```
Index(['country', 'year', 'age', 'Gender_female', 'Gender_male', 'suicides_no',
       'population', 'suicides/100k pop', 'country-year', 'HDI for year',
       ' gdp_for_year ($) ', 'gdp_per_capita ($)', 'generation', 'sex'],
      dtype='object')
```

In [34]:
```
df_dummies = df_dummies[['country', 'year', 'age','sex', 'Gender_female',
       'suicides/100k pop', 'country-year', 'HDI for year',
       ' gdp_for_year ($) ', 'gdp_per_capita ($)', 'generation']]
```

In [37]:
```
#Now we can observe the changes to the dataset properly
#What has the dummy variables done
df_dummies.head(5)
```

Out[37]:

	country	year	age	sex	Gender_female	Gender_male	suicides_no	populati
0	Albania	1987	15-24 years	male	0	1	21	31
1	Albania	1987	35-54 years	male	0	1	16	30
2	Albania	1987	15-24 years	female	1	0	14	28
3	Albania	1987	75+ years	male	0	1	1	2
4	Albania	1987	25-34 years	male	0	1	9	27

- Wherever sex is "male" , the Dummy variable "Gender_female" is 0 and "Gender_male" is 1

- Wherever sex is "female" , the Dummy variable "Gender_male" is 0 and "Gender_female" is 1

Lets create Dummy variables for "Age" column

In [39]:
```
df_dummies = pd.get_dummies(data1, prefix = "Age", columns = ['age'])
```

In [41]:
```
df_dummies.head(5)
```

Out[41]:

tion	suicides/100k pop	country-year	HDI for year	gdp_for_year ($)	gdp_per_capita ($)	generation	Age_ ye
12900	6.71	Albania1987	NaN	2,156,624,900	796	Generation X	
08000	5.19	Albania1987	NaN	2,156,624,900	796	Silent	
89700	4.83	Albania1987	NaN	2,156,624,900	796	Generation X	
21800	4.59	Albania1987	NaN	2,156,624,900	796	G.I. Generation	
74300	3.28	Albania1987	NaN	2,156,624,900	796	Boomers	

In [43]:
```
df_dummies = df_dummies.copy()
df_dummies['age'] = data1['age']
```

In [44]:
```
df_dummies.columns
```

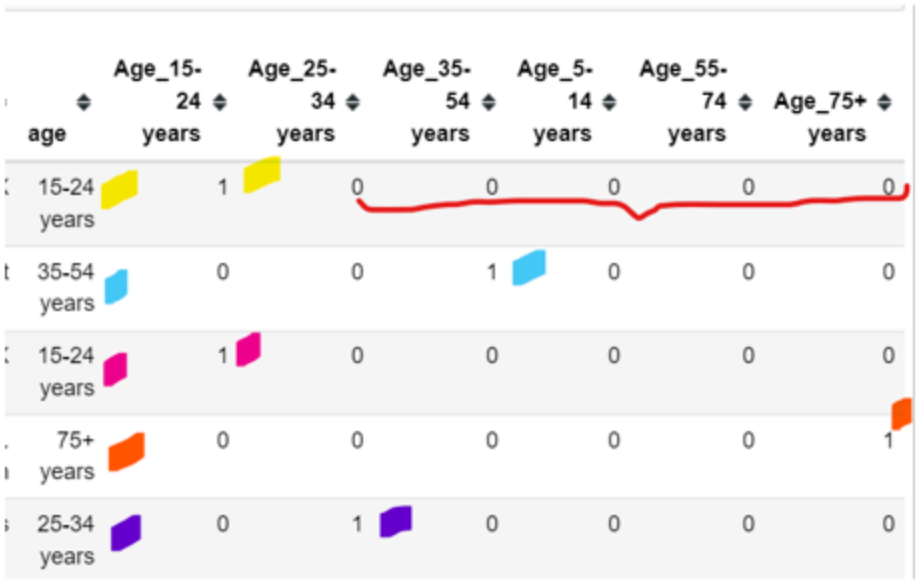
Out[44]:
```
Index(['country', 'year', 'sex', 'suicides_no', 'population',
       'suicides/100k pop', 'country-year', 'HDI for year',
       ' gdp_for_year ($) ', 'gdp_per_capita ($)', 'generation',
       'Age_15-24 years', 'Age_25-34 years', 'Age_35-54 years',
       'Age_5-14 years', 'Age_55-74 years', 'Age_75+ years', 'age'],
      dtype='object')
```

In [45]:
```
df_dummies = df_dummies[['country', 'year', 'sex', 'suicides_no', 'popula
       'suicides/100k pop', 'country-year', 'HDI for year',
       ' gdp_for_year ($) ', 'gdp_per_capita ($)', 'generation', 'age',
       'Age_15-24 years', 'Age_25-34 years', 'Age_35-54 years',
       'Age_5-14 years', 'Age_55-74 years', 'Age_75+ years']]
```

In [46]:
```
1 df_dummies.head(5)
```

Out[46]:

suicides/100k pop	country-year	HDI for year	gdp_for_year ($)	gdp_per_capita ($)	generation	age	Age_ ye
6.71	Albania1987	NaN	2,156,624,900	796	Generation X	15-24 years	
5.19	Albania1987	NaN	2,156,624,900	796	Silent	35-54 years	
4.83	Albania1987	NaN	2,156,624,900	796	Generation X	15-24 years	
4.59	Albania1987	NaN	2,156,624,900	796	G.I. Generation	75+ years	
3.28	Albania1987	NaN	2,156,624,900	796	Boomers	25-34 years	



Sklearn Label Encoder

In [47]:
```
1 #'generation'
```

In [53]:
```
1 data1.head(4)
```

Out[53]:

	country	year	sex	age	suicides_no	population	suicides/100k pop	country-year
0	Albania	1987	male	15-24 years	21	312900	6.71	Albania1987
1	Albania	1987	male	35-54 years	16	308000	5.19	Albania1987
2	Albania	1987	female	15-24 years	14	289700	4.83	Albania1987
3	Albania	1987	male	75+ years	1	21800	4.59	Albania1987

In [55]:
```
1 data1['generation'].unique()
```

Out[55]:
```
array(['Generation X', 'Silent', 'G.I. Generation', 'Boomers',
       'Millenials', 'Generation Z'], dtype=object)
```

In [54]:
```
1 df_label = data1.copy()
```

In [56]:
```
1 from sklearn.preprocessing import LabelEncoder
2 labelencoder = LabelEncoder()
3 df_label['generation'] = labelencoder.fit_transform(df_label.generation)
```

In [65]:
```
1 data1['generation'].tail(10)
```

Out[65]:
```
27810      Millenials
27811         Boomers
27812      Millenials
27813          Silent
27814      Millenials
27815    Generation X
27816          Silent
27817    Generation Z
27818    Generation Z
27819         Boomers
Name: generation, dtype: object
```

In [67]:
```
df_label['generation'].tail(10)
```

Out[67]:
```
27810    4
27811    0
27812    4
27813    5
27814    4
27815    2
27816    5
27817    3
27818    3
27819    0
Name: generation, dtype: int32
```

Label Encoding : How the categories in the column have been assigned the values 0 through 5

Boomers = 0 , G.I. Generation = 1 , Generation X = 2 , Generation Z = 3 , Millenials = 4 , Silent = 5

Sklearn OneHotEncoder

In [68]:
```
data_dummies = data1.copy()
```

In [69]:
```
from sklearn.preprocessing import OneHotEncoder
hotencoder = OneHotEncoder()
encoded = hotencoder.fit_transform(data_dummies.generation.values.reshape
#Returns a Numpy array of One Hot Encoded variables
encoded
```

Out[69]:
```
array([[0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1.],
       [0., 0., 1., 0., 0., 0.],
       ...,
       [0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0., 0.],
       [1., 0., 0., 0., 0., 0.]])
```

In [70]:
```
encoded.shape
```

Out[70]:
```
(27820, 6)
```

Now add this back to the data_dummies dataset

In [74]:
```
#convert the array into a DataFrame. Specifically, One Hot encoded DataFr
df_encoded = pd.DataFrame(encoded, columns=["generation_" + str(int(i)) f
```

In [75]:
```
1 df_encoded.head()
```

Out[75]:

	generation_0	generation_1	generation_2	generation_3	generation_4	generation_
0	0.0	0.0	1.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	1.0	0.0	0.0	
3	0.0	1.0	0.0	0.0	0.0	
4	1.0	0.0	0.0	0.0	0.0	

In [73]:
```
1 df_encoded.shape
```

Out[73]: (27820, 6)

In [76]:
```
1 data1['generation'].unique()
```

Out[76]:
```
array(['Generation X', 'Silent', 'G.I. Generation', 'Boomers',
       'Millenials', 'Generation Z'], dtype=object)
```

In [77]:
```
1 data_dummies = pd.concat([data_dummies , df_encoded], axis = 1)  #Concats
```

In [78]:
```
1 data_dummies.head(3)
```

Out[78]:

ntry-year	HDI for year	gdp_for_year ($)	gdp_per_capita ($)	generation	generation_0	generation_1
nia1987	NaN	2,156,624,900	796	Generation X	0.0	0.0
nia1987	NaN	2,156,624,900	796	Silent	0.0	0.0
nia1987	NaN	2,156,624,900	796	Generation X	0.0	0.0

You don't need to do this with every Analysis . This is only done when we are building a Model, where the Algorithm is incapable of looking at Alpha Numerics or words.

With One Hot Encoding we got n-1 dummy variables.

Meaning : generation column had 7 Categories , after One hot encoding , we see 6 variables got created which is 1 less.

ENd

In [ ]:
```
1
```

In [ ]:
```
1
```