



Department of Data Science

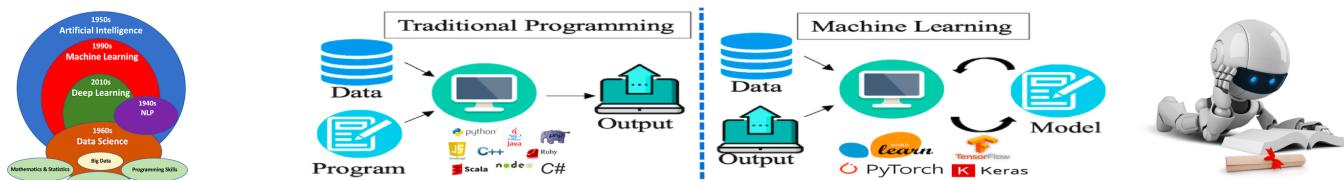
Course: Tools and Techniques for Data Science

Instructor: Muhammad Arif Butt, Ph.D.

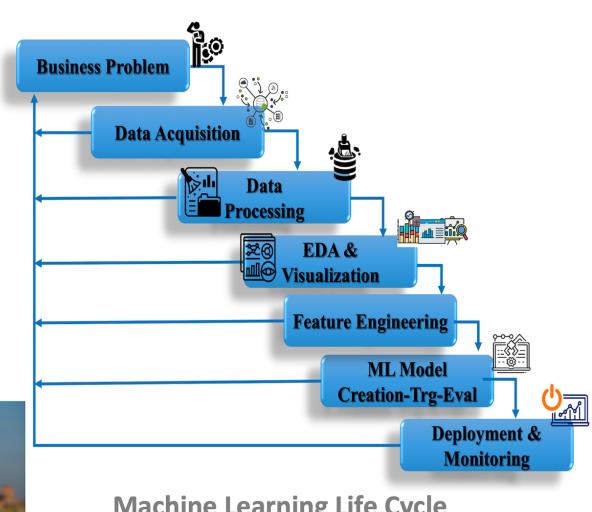
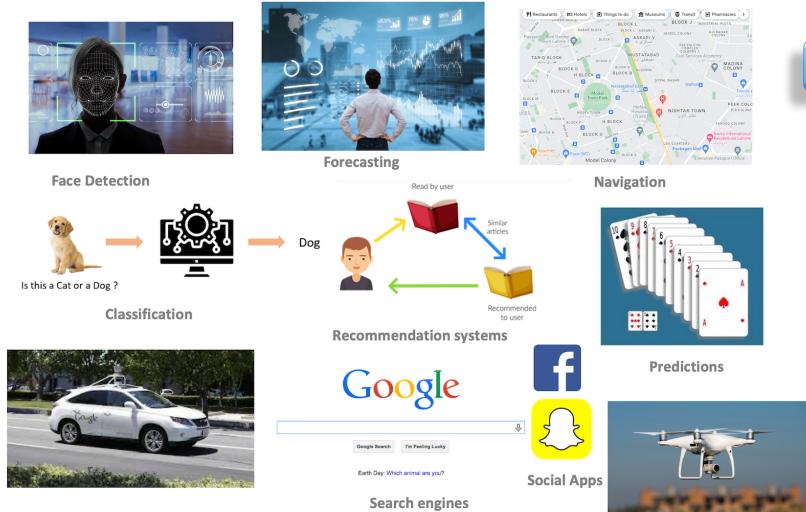
Lecture 6.9 (Data Preprocessing: Encoding Categorical Data)

Open in Colab

([https://colab.research.google.com/github/arifpcit/data-science/blob/master/Section-4-Mathematics-for-Data-Science/Lec-4.1\(Descriptive-Statistics\).ipynb](https://colab.research.google.com/github/arifpcit/data-science/blob/master/Section-4-Mathematics-for-Data-Science/Lec-4.1(Descriptive-Statistics).ipynb))



ML is the application of AI that gives machines the ability to learn without being explicitly programmed



Learning agenda of this notebook

- Overview of Data Pre-Processing and Feature Engineering
- Understanding Data Types
- Encoding Categorical Data (What? Why? and How?)
- A Step-by-Step Hello World to Encode Categorical Data
 - Scikit-Learn's `LabelEncoder()` vs `OrdinalEncoder()`

- Encoding Nominal Variables using Pandas `get_dummies()` and Scikit-Learn's `OneHotEncoder()`
- Perform Train-Test-Split and then do the Transformations
- Using ColumnTransformer for Encoding and Imputation

1. Overview of Data Pre-Processing and Feature Engineering

- Data Preprocessing involves actions that we need to perform on the dataset in order to make it ready to be fed to the machine learning model.
- Feature Engineering is the process of using domain knowledge to extract features from raw data via data mining techniques.

City	Size	Covered Area	No of bedrooms	Trees near by	No of bathrooms	Schools near by	Construction Date	Price
Lahore	2000	3500	3	1	3	1	25/10/2001	20.5 M
Karachi	2600	3000	2	0	4	1	16/05/1990	18 M
Islamabad	1800	2000	3	1	3	2	25/11/1995	20 M
Shaikhupura	1600	2600	1	2	NaN	0	08/06/2020	5 M
Lahore	2600	2000	3	3	1	1	03/09/2016	4 M
Karachi	3000	1000	2	2	1	NaN	19/01/1980	6 M
Islamabad	2000	3600	44	4	3	3	21/07/1999	30 M
Lahore	1000	2000	3	NaN	1	2	12/04/2015	10 M

- Pre-processing package of sklearn provides a bundle of utility functions and transformer classes for data preprocessing (will cover later).
 - **Detecting and handling outliers**
 - Univariate (Z-Score, IQR, Percentiles)
 - Multivariate Analysis (Depth-based, Distance-based, Density-based methods)
 - Trimming, Capping/Winsorization, Discretization
 - **Missing values Imputation**
 - Univariate Imputation (Panda's `fillna()` method, Sklearn's `SimpleImputer()` transformer)
 - Multivariate Imputation (Sklearn's `IterativeImputer()` and `KNNImputer()` transformers)
 - **Encoding Categorical Features**
 - Encode Nominal i/p features using Pandas `get_dummies()` and Scikit-Learn's `OneHotEncoder()`
 - Encode Ordinal i/p features using Scikit-Learn's `OrdinalEncoder()`
 - Encode categorical o/p column using Scikit-Learn's `LabelEncoder()`
 - **Feature Scaling**
 - Use numpy to perform maxabs, minmax, standard and robust scaling
 - Use Sklearn's `MaxAbsScalar`, `MinMaxScalar`, `StandardScalar`, `RobustScalar` transformers
 - **Extracting Information**
 - Use Sklearn's `CountVectorizer`, `DictVectorizer`, `TfidfVectorizer`, and `TfidfTransformer`
 - **Combining Information**

- Use FeatureUnion , Pipeline , PCA

2. Understanding Data Types

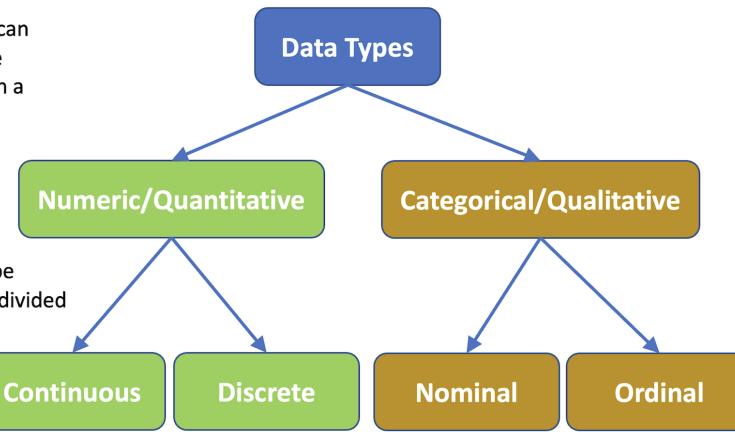
Numerical data is a type of data that expresses information in the form of numbers, while Categorical data is a type of data that is used to group information with similar characteristics

Continuous: Values that can be calculated and can have infinite number of values in a range

- Fare/Price
- Weight
- IQ/GPA

Discrete: Values that can be counted and cannot be subdivided meaningfully

- # of children
- # of visits to doctor
- # of students in a class
- World population



Nominal: Values that can be slotted into mutual exclusive categories that do not have specific order

- Blood Group
- Gender
- Color/Make of car
- Country/City

Ordinal: Values that can be slotted into mutually exclusive categories that can be ordered or ranked

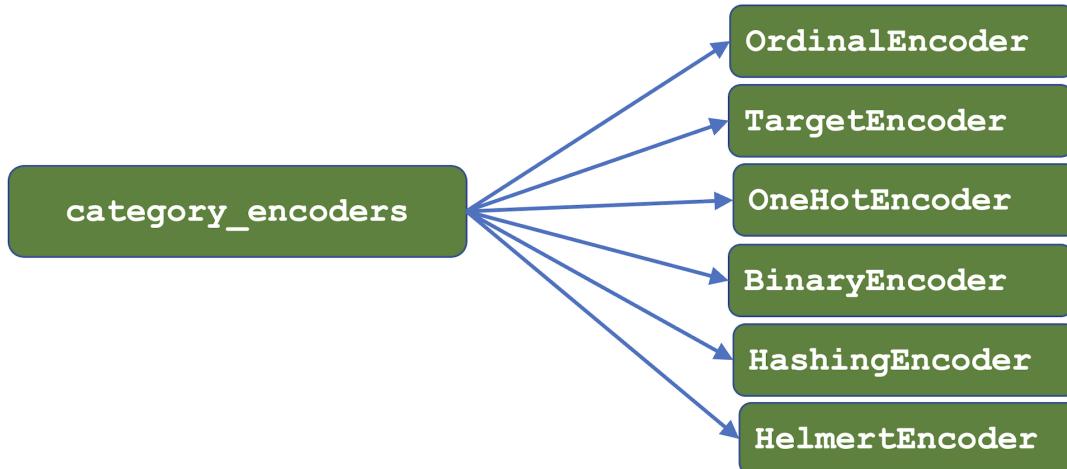
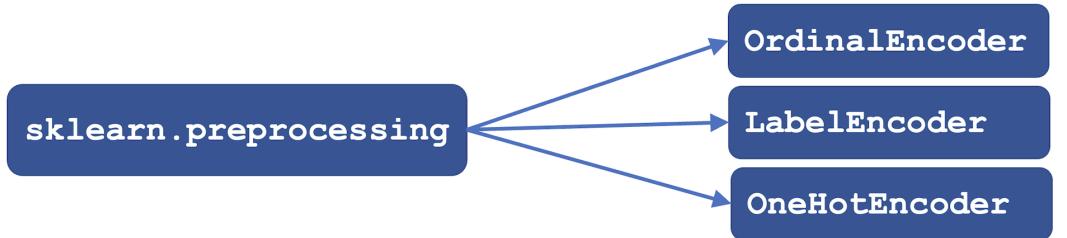
- Passenger class
- Student Letter Grade
- PSL Cricket Team Ranking

pclass		name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest	target
0	1.0	Allen, Miss. Elisabeth Walton	female	29.0000	0.0	0.0	24160	211.3375	B5	S	2	NaN	St Louis, MO	1
1	1.0	Allison, Master. Hudson Trevor	male	0.9167	1.0	2.0	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON	1
2	1.0	Allison, Miss. Helen Loraine	female	2.0000	1.0	2.0	113781	151.5500	C22 C26	S	None	NaN	Montreal, PQ / Chesterville, ON	0
3	1.0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1.0	2.0	113781	151.5500	C22 C26	S	None	135.0	Montreal, PQ / Chesterville, ON	0
4	1.0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1.0	2.0	113781	151.5500	C22 C26	S	None	NaN	Montreal, PQ / Chesterville, ON	0

3. Encoding Categorical Data (What? Why? and How?)

Encoding categorical data is a process of converting it into numerical values, so that it could be fed to machine learning models

- **Why do we need to encode categorical data?** Most machine learning algorithms cannot handle categorical variables unless we convert them to numerical values. Many algorithm's performances even vary based upon how the categorical variables are encoded



4. A Step-by-Step Hello World to Encode Categorical Data

a. Load Dataset

In [1]:

```

1 import pandas as pd
2 import numpy as np
3 df = pd.read_csv('datasets/disease.csv')
4 df.sample(n=7, random_state=54)
  
```

Out[1]:

	gender	city	age	bp	cough	disease
36	Female	Shaikhupura	38.0	normal	Mild	No
70	Female	Islamabad	68.0	normal	Strong	No
48	Male	Shaikhupura	66.0	low	Moderate	No
94	Male	Lahore	79.0	normal	Strong	Yes
81	Male	Islamabad	65.0	normal	Mild	No
46	Female	Karachi	NaN	normal	Moderate	No
38	Female	Islamabad	49.0	high	Mild	Yes

In [2]:

```
1 df.shape
```

Out[2]:

```
(100, 6)
```

In [3]:

```
1 df.nunique()
```

Out[3]:

```
gender      2
city        4
age         54
bp          3
cough       3
disease     2
dtype: int64
```

In [4]:

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype  
--- 
 0   gender    100 non-null   object 
 1   city      100 non-null   object 
 2   age       87 non-null   float64
 3   bp        100 non-null   object 
 4   cough     100 non-null   object 
 5   disease   100 non-null   object 
dtypes: float64(1), object(5)
memory usage: 4.8+ KB
```

In [5]:

```
1 df.disease.value_counts()
```

Out[5]:

```
No      54
Yes     46
Name: disease, dtype: int64
```

b. Scikit-Learn's LabelEncoder vs OrdinalEncoder

Label Encoding is a popular encoding technique for handling categorical variables, which assign each category value a unique integer starting from 0 to n-1 based on alphabetical ordering

- `LabelEncoder` is used for encoding output variable while `OrdinalEncoder` is used for encoding input feature variables of ordinal type, having an intrinsic order.
- `LabelEncoder` can fit one column at a time while `OrdinalEncoder` can fit multiple columns at the same time.
- Finally both encoders sort the values of a column alphabetically and then assign them the integer values. For example, in case of city column having two values "Islamabad" and "Karachi", Islamabad will be assigned an integer value of 0, while Karachi will be assigned a value of 1. However, in case of `OrdinalEncoder`, we can mention what integer values should be assigned to each specific category.

[Scikit Learn's LabelEncoder \(<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html>\)](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html)

[Scikit Learn's OrdinalEncoder \(<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html>\)](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html)

Apply LabelEncoder on disease Target Label

In [6]:

```
1 df.head()
```

Out[6]:

	gender	city	age	bp	cough	disease
0	Male	Lahore	60.0	low	Moderate	Yes
1	Male	Islamabad	27.0	low	Mild	No
2	Male	Islamabad	NaN	normal	Strong	No
3	Female	Lahore	31.0	high	Moderate	Yes
4	Female	Karachi	65.0	high	Mild	No

In [7]:

```
1 y = df.iloc[:, -1]
2 y
```

Out[7]:

```
0      Yes
1      No
2      No
3     Yes
4      No
...
95     No
96    Yes
97     No
98     No
99    Yes
Name: disease, Length: 100, dtype: object
```

In [8]:

```
1 from sklearn.preprocessing import LabelEncoder  
2 le = LabelEncoder()  
3 le.fit(y)
```

Out[8]:

```
LabelEncoder()
```

In [9]:

```
1 arr_disease = le.transform(y)
```

In [10]:

```
1 le.classes_
```

Out[10]:

```
array(['No', 'Yes'], dtype=object)
```

In [11]:

```
1 arr_disease
```

Out[11]:

```
array([1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1,  
1,  
     1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0,  
1,  
     1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0,  
0,  
     0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,  
0,  
     0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1])
```

In [12]:

```
1 arr_disease.shape
```

Out[12]:

```
(100,)
```

In [13]:

```
1 df
```

Out[13]:

	gender	city	age	bp	cough	disease
0	Male	Lahore	60.0	low	Moderate	Yes
1	Male	Islamabad	27.0	low	Mild	No
2	Male	Islamabad	NaN	normal	Strong	No
3	Female	Lahore	31.0	high	Moderate	Yes
4	Female	Karachi	65.0	high	Mild	No
...
95	Female	Shaikhupura	NaN	normal	Mild	No
96	Female	Lahore	51.0	high	Strong	Yes
97	Female	Shaikhupura	20.0	normal	Mild	No
98	Female	Karachi	5.0	low	Moderate	No
99	Female	Lahore	10.0	low	Strong	Yes

100 rows × 6 columns

Apply OrdinalEncoder on bp and cough Features

In [14]:

```
1 df.head()
```

Out[14]:

	gender	city	age	bp	cough	disease
0	Male	Lahore	60.0	low	Moderate	Yes
1	Male	Islamabad	27.0	low	Mild	No
2	Male	Islamabad	NaN	normal	Strong	No
3	Female	Lahore	31.0	high	Moderate	Yes
4	Female	Karachi	65.0	high	Mild	No

In [15]:

```
1 X_bp_cough = df.iloc[:, 3:5]
2 X_bp_cough.head()
```

Out[15]:

	bp	cough
0	low	Moderate
1	low	Mild
2	normal	Strong
3	high	Moderate
4	high	Mild

In [16]:

```
1 from sklearn.preprocessing import OrdinalEncoder
2 oe = OrdinalEncoder()
3 oe.fit(X_bp_cough)
4 oe.categories_
```

Out[16]:

```
[array(['high', 'low', 'normal'], dtype=object),
 array(['Mild', 'Moderate', 'Strong'], dtype=object)]
```

- The values of column are sorted alphabetically and then assigned integer values from zero onwards

In [17]:

```
1 from sklearn.preprocessing import OrdinalEncoder
2 oe = OrdinalEncoder(categories=[['low', 'normal', 'high'], ['Mild', 'Moderate', 'Strong']])
3 oe.fit(X_bp_cough)
4 oe.categories_
```

Out[17]:

```
[array(['low', 'normal', 'high'], dtype=object),
 array(['Mild', 'Moderate', 'Strong'], dtype=object)]
```

- OrdinalEncoder, unlike LabelEncoder can assign specific integer values by specifying the categories argument

In [18]:

```
1 arr_bp_cough = oe.transform(X_bp_cough)
2 arr_bp_cough.shape
```

Out[18]:

(100, 2)

In [19]:

```
1 arr_bp_cough
```

Out[19]:

```
array([[0, 1],
       [0, 0],
       [1, 2],
       [2, 1],
       [2, 0],
       [2, 0],
       [1, 2],
       [1, 1],
       [2, 2],
       [0, 0],
       [0, 0],
       [2, 1],
       [1, 2],
       [0, 0],
       [0, 0],
       [1, 2],
       [1, 0],
       [0, 1]]]
```

c. Encoding Nominal Variables using OneHotEncoder

One Hot Encoding is the process of creating dummy variables and each category is represented as a one-hot vector

Feature	One-Hot Encoding			Dummy Variable Trap	
Color	Red	Green	Pink	Green	Pink
Red	1	0	0	0	0
Red	1	0	0	0	0
Green	0	1	0	1	0
Pink	0	0	1	0	1
Green	0	1	0	1	0
Red	1	0	0	0	0
Green	0	1	0	1	0
Pink	0	0	1	0	1

"Red" = [1, 0, 0] = [0, 0]
 "Green" = [0, 1, 0] = [1, 0]
 "Pink" = [0, 0, 1] = [0, 1]

- Limitations:
 - Multicollinearity
 - If you add the one-hot vector they sum up to one. For LinearRegression we know that the input

Apply pd.get_dummies on gender and city Columns

In [20]:

```
1 df.head()
```

Out[20]:

	gender	city	age	bp	cough	disease
0	Male	Lahore	60.0	low	Moderate	Yes
1	Male	Islamabad	27.0	low	Mild	No
2	Male	Islamabad	Nan	normal	Strong	No
3	Female	Lahore	31.0	high	Moderate	Yes
4	Female	Karachi	65.0	high	Mild	No

In [21]:

```
1 X_gender = df.iloc[:, 0:1]
2 X_gender.head()
```

Out[21]:

gender

	gender
0	Male
1	Male
2	Male
3	Female
4	Female

In [22]:

```
1 #pd.get_dummies(data=X_gender)
2 pd.get_dummies(data=X_gender, drop_first=True)
```

Out[22]:

gender_Male

	gender_Male
0	1
1	1
2	1
3	0
4	0
...	...
95	0
96	0
97	0
98	0
99	0

100 rows × 1 columns

In [23]:

```
1 pd.get_dummies(data=df, columns=['gender', 'city'], drop_first=True)
```

Out[23]:

	age	bp	cough	disease	gender_Male	city_Karachi	city_Lahore	city_Shaikhupura
0	60.0	low	Moderate	Yes	1	0	1	0
1	27.0	low	Mild	No	1	0	0	0
2	NaN	normal	Strong	No	1	0	0	0
3	31.0	high	Moderate	Yes	0	0	1	0
4	65.0	high	Mild	No	0	1	0	0
...
95	NaN	normal	Mild	No	0	0	0	1
96	51.0	high	Strong	Yes	0	0	1	0
97	20.0	normal	Mild	No	0	0	0	1
98	5.0	low	Moderate	No	0	1	0	0
99	10.0	low	Strong	Yes	0	0	1	0

100 rows × 8 columns

Apply OneHotEncoder on gender and city Column

In [24]:

```
1 X_gender_city = df.iloc[:, 0:2]
2 X_gender_city
```

Out[24]:

	gender	city
0	Male	Lahore
1	Male	Islamabad
2	Male	Islamabad
3	Female	Lahore
4	Female	Karachi
...
95	Female	Shaikhupura
96	Female	Lahore
97	Female	Shaikhupura
98	Female	Karachi
99	Female	Lahore

100 rows × 2 columns

In [25]:

```
1 X_gender_city['city'].nunique()
```

Out[25]:

4

In [26]:

```
1 from sklearn.preprocessing import OneHotEncoder
2 ohe = OneHotEncoder(drop='first', sparse=False, dtype=np.int32)
3 ohe.fit(X_gender_city)
```

Out[26]:

```
OneHotEncoder(drop='first', dtype=<class 'numpy.int32'>, sparse=False)
```

In [27]:

```
1 ohe.categories_
```

Out[27]:

```
[array(['Female', 'Male'], dtype=object),
 array(['Islamabad', 'Karachi', 'Lahore', 'Shaikhupura'], dtype=object)]
```

In [28]:

```
1 arr_gender_city = ohe.transform(X_gender_city)
2 arr_gender_city.shape
```

Out[28]:

(100, 4)

In [29]:

```
1 arr_gender_city
```

Out[29]:

```
array([[1, 0, 1, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       [0, 0, 1, 0],
       [0, 1, 0, 0],
       [0, 0, 0, 1],
       [1, 0, 0, 1],
       [0, 1, 0, 0],
       [0, 0, 0, 1],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 0, 1, 0],
       [1, 0, 0, 1],
       [1, 0, 0, 1],
       [1, 0, 1, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1]])
```

5. Perform Train-Test-Split and then do the Transformations

- A better approach is to train-test-split first and then do the encoding so that there is no data leakage.
- Call `fit_transform()` on the training data, and only `transform()` on the test data. (NEVER call `fit()` on training data)
- Almost all feature engineering like encoding, standarisation, normalisation etc should be done after train-test-split.

a. Load Dataset

In [30]:

```
1 import numpy as np
2 import pandas as pd
3 from sklearn.preprocessing import OrdinalEncoder
4 from sklearn.preprocessing import LabelEncoder
5 from sklearn.preprocessing import OneHotEncoder
6 from sklearn.impute import SimpleImputer
7 from sklearn.model_selection import train_test_split
8 df = pd.read_csv('datasets/disease.csv')
9 df.head()
```

Out[30]:

	gender	city	age	bp	cough	disease
0	Male	Lahore	60.0	low	Moderate	Yes
1	Male	Islamabad	27.0	low	Mild	No
2	Male	Islamabad	NaN	normal	Strong	No
3	Female	Lahore	31.0	high	Moderate	Yes
4	Female	Karachi	65.0	high	Mild	No

b. Do a Train-Test-Split

In [31]:

```
1 X = df.drop('disease', axis=1)
2 X
```

Out[31]:

	gender	city	age	bp	cough
0	Male	Lahore	60.0	low	Moderate
1	Male	Islamabad	27.0	low	Mild
2	Male	Islamabad	NaN	normal	Strong
3	Female	Lahore	31.0	high	Moderate
4	Female	Karachi	65.0	high	Mild
...
95	Female	Shaikhupura	NaN	normal	Mild
96	Female	Lahore	51.0	high	Strong
97	Female	Shaikhupura	20.0	normal	Mild
98	Female	Karachi	5.0	low	Moderate
99	Female	Lahore	10.0	low	Strong

100 rows × 5 columns

In [32]:

```
1 y = df['disease']
2 y
```

Out[32]:

```
0    Yes
1    No
2    No
3    Yes
4    No
...
95   No
96   Yes
97   No
98   No
99   Yes
Name: disease, Length: 100, dtype: object
```

In [33]:

```
1 from sklearn.model_selection import train_test_split
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
3 len(X_train), len(y_train), len(X_test), len(y_test))
```

Out[33]:

(80, 80, 20, 20)

In [34]:

```
1 X_train.head()
```

Out[34]:

	gender	city	age	bp	cough
90	Female	Islamabad	59.0	low	Moderate
31	Male	Lahore	NaN	normal	Mild
58	Male	Karachi	23.0	low	Strong
74	Female	Islamabad	34.0	high	Strong
89	Male	Shaikhupura	46.0	low	Strong

In [35]:

```
1 y_train.head()
```

Out[35]:

```
90      No
31      No
58     Yes
74      No
89      No
Name: disease, dtype: object
```

In [36]:

```
1 X_test.head()
```

Out[36]:

	gender	city	age	bp	cough
36	Female	Shaikhupura	38.0	normal	Mild
70	Female	Islamabad	68.0	normal	Strong
48	Male	Shaikhupura	66.0	low	Moderate
94	Male	Lahore	79.0	normal	Strong
81	Male	Islamabad	65.0	normal	Mild

In [37]:

```
1 y_test.head()
```

Out[37]:

```
36      No
70      No
48      No
94    Yes
81      No
Name: disease, dtype: object
```

c. Apply OneHotEncoder to gender and city Columns

In [38]:

```
1 # OneHotEncoding -> gender,city
2 ohe = OneHotEncoder(drop='first',sparse=False,dtype=np.int32)
3
4 #Call fit() on the training data
5 ohe.fit(X_train[['gender','city']])
6 #Call transform() on both the training data as well as on the testing data
7 X_train_gender_city = ohe.transform(X_train[['gender','city']])
8 X_test_gender_city = ohe.transform(X_test[['gender','city']])
9
10 X_train_gender_city.shape, X_test_gender_city.shape
```

Out[38]:

```
((80, 4), (20, 4))
```

One column for gender and three columns for city

d. Apply OrdinalEncoder to bp and cough Columns

In [39]:

```
1 # Ordinalencoding -> cough
2 oe = OrdinalEncoder(categories=[[ 'low', 'normal', 'high'],['Mild', 'Moderate', 'Severe']])
3 #Call fit() on the training data
4 oe.fit(X_train[['bp', 'cough']])
5 #Call transform() on both the training data as well as on the testing data
6 X_train_bp_cough = oe.transform(X_train[['bp', 'cough']])
7 X_test_bp_cough = oe.transform(X_test[['bp', 'cough']])
8
9 X_train_bp_cough.shape, X_test_bp_cough.shape
```

Out[39]:

```
((80, 2), (20, 2))
```

e. Apply LabelEncoder on disease output Column

In [40]:

```
1 le = LabelEncoder()
2
3 #Call fit() on the training data
4 le.fit(y_train)
5 #Call transform() on both the training data as well as on the testing data
6 y_train = le.transform(y_train)
7 y_test = le.transform(y_test)
8
9 y_train.shape, y_test.shape
```

Out[40]:

```
((80,), (20,))
```

In [41]:

```
1 y_test
```

Out[41]:

```
array([0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0])
```

f. Apply SimpleImputer to age Column

In [42]:

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype  
--- 
 0   gender     100 non-null   object 
 1   city       100 non-null   object 
 2   age        87 non-null   float64
 3   bp         100 non-null   object 
 4   cough      100 non-null   object 
 5   disease    100 non-null   object 
dtypes: float64(1), object(5)
memory usage: 4.8+ KB
```

In [43]:

```
1 df.age.isnull().sum()
```

Out[43]:

13

In [44]:

```
1 from sklearn.impute import SimpleImputer
2 si = SimpleImputer(missing_values=np.nan, strategy='mean')
3 #Call fit() on the training data
4 si.fit(X_train[['age']])
5 #Call transform() on both the training data as well as on the testing data
6 X_train_age = si.transform(X_train[['age']])
7 X_test_age = si.transform(X_test[['age']])
8
9 X_train_age.shape, X_test_age.shape
```

Out[44]:

((80, 1), (20, 1))

In [45]:

```
1 X_test_age
```

Out[45]:

```
array([[38.,
       [68.,
        [66.,
        [79.,
        [65.,
        [41.94202899],
        [49.,
        [34.,
        [74.,
        [71.,
        [16.,
        [82.,
        [38.,
        [34.,
        [60.,
        [75.,
        [41.94202899],
        [65.,
        [ 6.,
        [26.]]])
```

g. Combine Results

- We need to combine the three resulting/transformed arrays:
 - X_train_gender_city, X_train_age, X_train_bp_cough,
 - X_test_gender_city, X_test_age, X_test_bp_cough

Combine the transformed resulting arrays of Train Dataset:

In [46]:

```
1 X_train_gender_city.shape, X_train_bp_cough.shape, X_train_age.shape
```

Out[46]:

((80, 4), (80, 2), (80, 1))

In [47]:

```
1 X_train_transformed = np.concatenate((X_train_gender_city, X_train_age, X_train_bp_cough))
2 X_train_transformed.shape
```

Out[47]:

(80, 7)

In [48]:

```
1 X_train_transformed
```

Out[48]:

```
array([[ 0.        ,  0.        ,  0.        ,  0.        ,  59.        ,
         0.        ,  1.        ],
       [ 1.        ,  0.        ,  1.        ,  0.        ,  41.9420289
9,
       1.        ,  0.        ],
       [ 1.        ,  1.        ,  0.        ,  0.        ,  23.        ],
       0.        ,  2.        ],
       [ 0.        ,  0.        ,  0.        ,  0.        ,  34.        ],
       2.        ,  2.        ],
       [ 1.        ,  0.        ,  0.        ,  1.        ,  46.        ],
       0.        ,  2.        ],
       [ 0.        ,  0.        ,  0.        ,  1.        ,  20.        ],
       1.        ,  0.        ]])
```

Combine the transformed resulting arrays of Test Dataset:

In [49]:

```
1 X_test_gender_city.shape, X_test_bp_cough.shape, X_test_age.shape
```

Out[49]:

((20, 4), (20, 2), (20, 1))

In [50]:

```
1 X_test_transformed = np.concatenate((X_test_gender_city, X_test_age, X_test_bp_cough))
2 X_test_transformed.shape
```

Out[50]:

(20, 7)

In [51]:

```
1 X_test_transformed
```

Out[51]:

```
array([[ 0.          ,  0.          ,  0.          ,  1.          , 38.]
```

Convert X_train_transformed array to a Dataframe for better visualization:

```
[ 0.          ,  0.          ,  0.          ,  0.          , 68.]
```

In [52]:

```
1 X_train_transformed_df = pd.DataFrame(X_train_transformed,
, 2                                         columns=['gender','city_1','city_2','city_
0.          ,  1.          ],
```

In [53]:

```
1 X_train.head(), 2.      ],
[ 1.          ,  0.          ,  0.          ,  0.          , 65.]
```

Out[53]:

	gender	city	age	bp	cough	
90	Female	1. Islamabad	59.01	low	Moderate	41.9420289
31	Male	0. Lahore	NaN	normal	'Mild'	49.
58	Male	2. Karachi	23.00	low	Strong	34.
74	Female	Islamabad	34.0	high	Strong	
89	Male	2 Shaikhupura	46.01	low	Strong	74.

In [54]:

```
0.          ,  0.          ],
[ 0.          ,  0.          ,  1.          ,  0.          , 71.]
```

```
, 1 X_train_transformed_df.head()
```

	gender	city_1	city_2	city_3	age	bp	cough	
0	0.0	0.0	0.0	0.0	59.000000	0.0	1.0	82.
1	1.0	1.0	0.0	0.0	41.942029	1.0	0.0	38.
2	1.0	1.0	0.0	0.0	23.000000	0.0	2.0	
3	0.0	0.0	0.0	0.0	34.000000	2.0	2.0	34.
4	1.0	0.0	0.0	1.0	46.000000	0.0	2.0	

Convert X_test_transformed array to a Dataframe for better visualization:

```
0.          ,  1.          ],
[ 0.          ,  0.          ,  0.          ,  0.          , 75.]
```

In [55]:

```
1 X_test_transformed_df = pd.DataFrame(X_test_transformed,
, 2                                         columns=['gender','city_1','city_2','city_
1.          ,  0.          ],
[ 1.          ,  0.          ,  0.          ,  1.          , 65.]
```

```
, 2 X_test_transformed_df = pd.DataFrame(X_test_transformed,
, 3                                         columns=['gender','city_1','city_2','city_
2.          ,  0.          ],
[ 0.          ,  0.          ,  1.          ,  0.          , 6.]
```

```
, 2 X_test_transformed_df = pd.DataFrame(X_test_transformed,
, 3                                         columns=['gender','city_1','city_2','city_
2.          ,  0.          ],
[ 0.          ,  0.          ,  1.          ,  0.          , 26.]
```

```
, 1 X_test_transformed_df = pd.DataFrame(X_test_transformed,
, 2                                         columns=['gender','city_1','city_2','city_
1.          ,  1.          ],
[])
```

In [56]:

```
1 x_test.head()
```

Out[56]:

	gender	city	age	bp	cough
36	Female	Shaikhupura	38.0	normal	Mild
70	Female	Islamabad	68.0	normal	Strong
48	Male	Shaikhupura	66.0	low	Moderate
94	Male	Lahore	79.0	normal	Strong
81	Male	Islamabad	65.0	normal	Mild

In [57]:

```
1 x_test_transformed_df.head()
```

Out[57]:

	gender	city_1	city_2	city_3	age	bp	cough
0	0.0	0.0	0.0	1.0	38.0	1.0	0.0
1	0.0	0.0	0.0	0.0	68.0	1.0	2.0
2	1.0	0.0	0.0	1.0	66.0	0.0	1.0
3	1.0	0.0	1.0	0.0	79.0	1.0	2.0
4	1.0	0.0	0.0	0.0	65.0	1.0	0.0

6. Using ColumnTransformer for Encoding and Imputation

Data Transformers in Scikit-Learn: (https://scikit-learn.org/stable/data_transforms.html)

ColumnTransformer allows to apply different Transformers on different columns of our dataset in a simple and elegant way

In [58]:

```
1 # One-Hot-Encoder for gender and city column
2 ohe = OneHotEncoder(drop='first', sparse=False, dtype=np.int32)
3 # OrdinalEncoder for bp and cough column
4 oe = OrdinalEncoder(categories=[['low', 'normal', 'high'], ['Mild', 'Moderate', 'Strong']])
5 # SimpleImputer for age column
6 si = SimpleImputer(missing_values=np.nan, strategy='mean')
```

In [59]:

```
1 from sklearn.compose import ColumnTransformer
2 # define ColumnTransformer
3 transformer = ColumnTransformer(transformers=[
4     ('tfm1', ohe, ['gender', 'city']),
5     ('tfm2', oe, ['bp', 'cough']),
6     ('tfm3', si, ['age']),
7 ],
8 remainder='passthrough')
9 type(transformer)
```

Out[59]:

```
sklearn.compose._column_transformer.ColumnTransformer
```

In [60]:

```
1 #Call fit() on the training data
2 transformer.fit(X_train)
3 #Call transform() on both the training data as well as on the testing data
4 X_train_transformed1 = transformer.transform(X_train)
5 X_test_transformed1 = transformer.transform(X_test)
6
7 X_train_transformed1.shape, X_test_transformed1.shape
```

Out[60]:

```
((80, 7), (20, 7))
```

In [61]:

```
1 X_train_transformed1
```

Out[61]:

```
array([[ 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
         1.        ,  59.       ],
       [ 1.        ,  0.        ,  1.        ,  0.        ,  1.        ,
         0.        , 41.94202899],
       [ 1.        ,  1.        ,  0.        ,  0.        ,  0.        ,
         2.        , 23.       ],
       [ 0.        ,  0.        ,  0.        ,  0.        ,  2.        ,
         2.        , 34.       ],
       [ 1.        ,  0.        ,  0.        ,  1.        ,  0.        ,
         2.        , 46.       ],
       [ 0.        ,  0.        ,  0.        ,  1.        ,  1.        ,
         0.        , 20.       ]])
```

In [62]:

```
1 df
```

Out[62]:

	gender	city	age	bp	cough	disease
0	Male	Lahore	60.0	low	Moderate	Yes
1	Male	Islamabad	27.0	low	Mild	No
2	Male	Islamabad	NaN	normal	Strong	No
3	Female	Lahore	31.0	high	Moderate	Yes
4	Female	Karachi	65.0	high	Mild	No
...
95	Female	Shaikhupura	NaN	normal	Mild	No
96	Female	Lahore	51.0	high	Strong	Yes
97	Female	Shaikhupura	20.0	normal	Mild	No
98	Female	Karachi	5.0	low	Moderate	No
99	Female	Lahore	10.0	low	Strong	Yes

100 rows × 6 columns

In [63]:

```
1 X_train_transformed_df1 = pd.DataFrame(X_train_transformed1,
2                                         columns=['gender','city_1','city_2','city_3',
3                                         'age','bp','cough']
```

Out[63]:

	gender	city_1	city_2	city_3	age	bp	cough
0	0.0	0.0	0.0	0.0	0.0	1.0	59.000000
1	1.0	0.0	1.0	0.0	1.0	0.0	41.942029
2	1.0	1.0	0.0	0.0	0.0	2.0	23.000000
3	0.0	0.0	0.0	0.0	2.0	2.0	34.000000
4	1.0	0.0	0.0	1.0	0.0	2.0	46.000000
...
75	1.0	1.0	0.0	0.0	1.0	0.0	42.000000
76	0.0	0.0	0.0	0.0	1.0	0.0	80.000000
77	1.0	0.0	1.0	0.0	1.0	2.0	70.000000
78	1.0	0.0	0.0	0.0	1.0	2.0	41.942029
79	0.0	0.0	0.0	0.0	2.0	0.0	73.000000

80 rows × 7 columns

In [64]:

1	x_train
---	---------

Out[64]:

	gender	city	age	bp	cough
90	Female	Islamabad	59.0	low	Moderate
31	Male	Lahore	NaN	normal	Mild
58	Male	Karachi	23.0	low	Strong
74	Female	Islamabad	34.0	high	Strong
89	Male	Shaikhupura	46.0	low	Strong
...
64	Male	Karachi	42.0	normal	Mild
23	Female	Islamabad	80.0	normal	Mild
15	Male	Lahore	70.0	normal	Strong
2	Male	Islamabad	NaN	normal	Strong
69	Female	Islamabad	73.0	high	Mild

80 rows × 5 columns