What is machine learning? In Machine Learning, Machine identify the procedure by itdelf with respect to input to perform certain action. it is also called ML Model, Algorithm and BOT. In ML both input and output are known but machine decide by itself that which procedure is suitable for it.

ML includes procedural programs- its like recipie of food- or which are actualy ways to perform certain tasks. In procedural programing we have input.

To write these libraries there are multiple libraries. One of the biggest is Scikit Learn.It:

takes data find patterns help to evaluate the results.

Dependent and Independent data? In an hypothetical data of causes of death events dues to heart, Death-Happening is dependent upon other data while Death-Hapening itself is independent. Moreover "x' represents the dependent while 'y' represents the independent variable.

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [4]:
```
heart_disease= pd.read_csv('heart_failure_clinical_records_dataset.csv')
heart_disease
```

Out[4]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	p
0	75.0	0	582	0	20	1	26
1	55.0	0	7861	0	38	0	26
2	65.0	0	146	0	20	0	16
3	50.0	1	111	0	20	0	21
4	65.0	1	160	1	20	0	32
...	...	...	...	...	...	...	
294	62.0	0	61	1	38	1	15
295	55.0	0	1820	0	38	0	27
296	45.0	0	2060	1	60	0	74
297	45.0	0	2413	0	38	0	14
298	50.0	0	196	0	45	0	39

299 rows × 13 columns

In [3]:

```
#creating independent variable.
X= heart_disease.drop('DEATH_EVENT', axis=1)
X
```

Out[3]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	p
0	75.0	0	582	0	20	1	26
1	55.0	0	7861	0	38	0	26
2	65.0	0	146	0	20	0	16
3	50.0	1	111	0	20	0	21
4	65.0	1	160	1	20	0	32
...	...	...	...	...	...	...	
294	62.0	0	61	1	38	1	15
295	55.0	0	1820	0	38	0	27
296	45.0	0	2060	1	60	0	74
297	45.0	0	2413	0	38	0	14
298	50.0	0	196	0	45	0	39

299 rows × 12 columns

In [4]:

```
#Creating dependent varible.
Y=heart_disease['DEATH_EVENT']
Y
```

Out[4]:

```
0      1
1      1
2      1
3      1
4      1
      ..
294    0
295    0
296    0
297    0
298    0
Name: DEATH_EVENT, Length: 299, dtype: int64
```

In [5]:

```
#Choosing ML model
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier() # creating an instance.
clf.get_params()
```

Out[5]:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

.get_params have shown parameters which can be used to manipulate data

In [6]:

```
#Fitting/ Tuning of data.
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.3)# 30 % of
```

In [7]:

```
clf.fit(X_train,Y_train)
```

Out[7]:

```
RandomForestClassifier()
```

In [8]:

```
#Evaluate Model
predicting= clf.predict(X_test)
predicting
```

Out[8]:

```
array([1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 0], dtype=int64)
```

In [9]:

```
clf.score(X_train, Y_train)
```

Out[9]:

```
1.0
```

In [10]:

```
clf.score(X_test, Y_test)
```

Out[10]:

```
0.8111111111111111
```

```
In [11]: #Improving Mode
         #To find the value of estimator where system's score is maximum.
         for i in range(10,210,10):
             print(f"value of estimator is {i}")
             clf=RandomForestClassifier(i).fit(X_train, Y_train)
             print(f'accuracy of test{clf.score(X_test, Y_test)}')
```

```
value of estimator is 10
accuracy of test0.7555555555555555
value of estimator is 20
accuracy of test0.7888888888888889
value of estimator is 30
accuracy of test0.7777777777777778
value of estimator is 40
accuracy of test0.7888888888888889
value of estimator is 50
accuracy of test0.7888888888888889
value of estimator is 60
accuracy of test0.7888888888888889
value of estimator is 70
accuracy of test0.8
value of estimator is 80
accuracy of test0.7777777777777778
value of estimator is 90
accuracy of test0.7777777777777778
value of estimator is 100
accuracy of test0.7777777777777778
value of estimator is 110
accuracy of test0.7777777777777778
value of estimator is 120
accuracy of test0.7777777777777778
value of estimator is 130
accuracy of test0.7888888888888889
value of estimator is 140
accuracy of test0.7777777777777778
value of estimator is 150
accuracy of test0.8
value of estimator is 160
accuracy of test0.7666666666666667
value of estimator is 170
accuracy of test0.8
value of estimator is 180
accuracy of test0.7777777777777778
value of estimator is 190
accuracy of test0.7777777777777778
value of estimator is 200
accuracy of test0.8
```

The best result is 84.44 % which is at i=150,160,170. Therefore, we will give one of these values to estimator or i to get the best result instantly.

Moreover, model is developed on the basis of train data. Model is unaware of test data, the score of test is actually a comparison between the o/p of train and test data which is 84 percent right.

In [12]:

```
clf=RandomForestClassifier(n_estimators=170).fit(X_train, Y_train)
print(f'accuracy of test{clf.score(X_test, Y_test)}')
```

```
accuracy of test0.7777777777777778
```

In [13]:

```
#Saving Model
import pickle
pickle.dump(clf,open('heart_disastor_predictor','wb') )
```

In [14]:

```
load_model=pickle.load(open('heart_disastor_predictor','rb'))
load_model.score(X_test, Y_test)
```

Out[14]:

```
0.7777777777777778
```

Detailed Overview of Every step.

Getting your Data Ready:

```
Split data into independent and dependent variable.
Filling the missing values
Coverting the data types.
```

In [15]:

```
ph_data = pd.read_csv('ph_data.csv')
ph_data
```

Out[15]:

	phone_brand	phone_price	phone_category	phone_memory	installments_amount	installments
0	TECNO	999	SMRTPHN	64000000	27	
1	TECNO	999	SMRTPHN	64000000	27	
2	TECNO	999	SMRTPHN	64000000	27	
3	NOKIA	899	SMRTPHN	64000000	24	
4	OPPO	2599	SMRTPHN	128000000	72	
...	...	...	...	...	...	
270	NOKIA	121	BASIC MOBILES	2500000	6	
271	NOKIA	90	BASIC MOBILES	2500000	5	
272	NOKIA	69	BASIC MOBILES	2500000	4	
273	NOKIA	69	BASIC MOBILES	2500000	4	
274	NOKIA	121	BASIC MOBILES	2500000	6	

275 rows × 6 columns

In [16]:

```
x = ph_data.drop('phone_price', axis=1)
x
```

Out[16]:

	phone_brand	phone_category	phone_memory	installments_amount	installments_period
0	TECNO	SMRTPHN	64000000	27	36
1	TECNO	SMRTPHN	64000000	27	36
2	TECNO	SMRTPHN	64000000	27	36
3	NOKIA	SMRTPHN	64000000	24	36
4	OPPO	SMRTPHN	128000000	72	36
...	...	...	...	...	...
270	NOKIA	BASIC MOBILES	2500000	6	36
271	NOKIA	BASIC MOBILES	2500000	5	36
272	NOKIA	BASIC MOBILES	2500000	4	36
273	NOKIA	BASIC MOBILES	2500000	4	36
274	NOKIA	BASIC MOBILES	2500000	6	36

275 rows × 5 columns

In [ ]:

```

```

In [17]:

```
y=ph_data['phone_price']
y
```

Out[17]:

```
0       999
1       999
2       999
3       899
4      2599
       ... 
270     121
271      90
272      69
273      69
274     121
Name: phone_price, Length: 275, dtype: int64
```

In [18]:

```
ph_data.dtypes
```

Out[18]:

```
phone_brand             object
phone_price              int64
phone_category          object
phone_memory             int64
installments_amount      int64
installments_period      int64
dtype: object
```

In [19]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2)
```

In [20]:
```
#Build ML model
from sklearn.ensemble import RandomForestClassifier
phone_model = RandomForestClassifier()
phone_model.fit(x_train, y_train)
phone_model.score(x_train,y_train)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-20-eb680721ab99> in <module>
      2 from sklearn.ensemble import RandomForestClassifier
      3 phone_model = RandomForestClassifier()
----> 4 phone_model.fit(x_train, y_train)
      5 phone_model.score(x_train,y_train)

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in fit(self, X, y, sa
mple_weight)
    301                     "sparse multilabel-indicator for y is not supported."
    302                 )
--> 303         X, y = self._validate_data(X, y, multi_output=True,
    304                                    accept_sparse="csc", dtype=DTYPE)
    305         if sample_weight is not None:

~\anaconda3\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, res
et, validate_separately, **check_params)
    430                 y = check_array(y, **check_y_params)
    431             else:
--> 432                 X, y = check_X_y(X, y, **check_params)
    433             out = X, y
    434 

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **k
wargs)
     70                           FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74 

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_X_y(X, y, ac
cept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_
2d, allow_nd, multi_output, ensure_min_samples, ensure_min_features, y_numeric,
estimator)
    793         raise ValueError("y cannot be None")
    794 
--> 795     X = check_array(X, accept_sparse=accept_sparse,
    796                     accept_large_sparse=accept_large_sparse,
    797                     dtype=dtype, order=order, copy=copy,

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **k
wargs)
     70                           FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74 

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array,
accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensur
e_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator)
    596                     array = array.astype(dtype, casting="unsafe", copy=
False)
    597                 else:
--> 598                     array = np.asarray(array, order=order, dtype=dtype)
    599             except ComplexWarning:
    600                 raise ValueError("Complex data not supported\n"

~\anaconda3\lib\site-packages\numpy\core\_asarray.py in asarray(a, dtype, orde
r)
     81
     82     """
---> 83     return array(a, dtype, copy=False, order=order)
     84
     85

~\anaconda3\lib\site-packages\pandas\core\generic.py in __array__(self, dtype)
   1779
   1780     def __array__(self, dtype=None) -> np.ndarray:
-> 1781         return np.asarray(self._values, dtype=dtype)
   1782
   1783     def __array_wrap__(self, result, context=None):

~\anaconda3\lib\site-packages\numpy\core\_asarray.py in asarray(a, dtype, orde
r)
     81
     82     """
---> 83     return array(a, dtype, copy=False, order=order)
     84
     85

ValueError: could not convert string to float: 'SAMSUNG'
```

So, now we have to tranform our data so that it can be read by compiler. For this, we will use an encoder i.e OneHotEncoder from preprocessing library and also ColumnsTransformer from compose library to trandform this data. First we will encode it and then tranform it.

ColumnTransformer([('give name of transfoer', encoder used, feature/columns to be transformed)], remainder= 'passthrough')

Remainder is used for the rest of the column and passthorugh means not to apply tranforming technique on rest of the column which are not used.

Method 1

```
In [21]: from sklearn.preprocessing import OneHotEncoder
         from sklearn.compose import ColumnTransformer

         features_cat = ["phone_category",]
         en_code = OneHotEncoder()

         transformer = ColumnTransformer([("data_tranform", en_code, features_cat)], remai

         transformed_x=transformer.fit_transform(x)
         pd.DataFrame(transformed_x)
```

```
Out[21]:
            0   1       2          3    4    5
       0    0   1   TECNO   64000000   27   36
       1    0   1   TECNO   64000000   27   36
       2    0   1   TECNO   64000000   27   36
       3    0   1   NOKIA   64000000   24   36
       4    0   1    OPPO  128000000   72   36
     ...  ...  ...    ...        ...  ...  ...
     270    1   0   NOKIA    2500000    6   36
     271    1   0   NOKIA    2500000    5   36
     272    1   0   NOKIA    2500000    4   36
     273    1   0   NOKIA    2500000    4   36
     274    1   0   NOKIA    2500000    6   36

    275 rows × 6 columns
```

Method 2

In [22]:
```
transformed_new_x=pd.get_dummies(ph_data[['phone_brand', 'phone_category']])
transformed_new_x
```

Out[22]:

	phone_brand_ALCATEL	phone_brand_APPLE	phone_brand_HONOR	phone_brand_HUAWEI	ph
0	0	0	0	0	
1	0	0	0	0	
2	0	0	0	0	
3	0	0	0	0	
4	0	0	0	0	
...	...	...	...	...	
270	0	0	0	0	
271	0	0	0	0	
272	0	0	0	0	
273	0	0	0	0	
274	0	0	0	0	

275 rows × 15 columns

In [23]:
```
x_train, x_test, y_train,y_test=train_test_split(transformed_new_x,y,test_size=0.
phone_model.fit(x_train, y_train)
```

Out[23]:
```
RandomForestClassifier()
```

In [24]:
```
phone_model.score(x_test, y_test)
```

Out[24]:
```
0.10843373493975904
```

In [25]:
```
y
```

Out[25]:
```
0       999
1       999
2       999
3       899
4      2599
       ... 
270     121
271      90
272      69
273      69
274     121
Name: phone_price, Length: 275, dtype: int64
```

Filling Missing Values:

In [26]:
```
new_data=pd.read_csv('phones_data.csv')
new_data
```

Out[26]:

	Unnamed: 0	brand_name	model_name	os	popularity	best_price	lowest_price	highest_
0	0	ALCATEL	1 1/8GB Bluish Black (5033D-2JALUAA)	Android	422	1690.0	1529.0	18
1	1	ALCATEL	1 5033D 1/16GB Volcano Black (5033D-2LALUAF)	Android	323	1803.0	1659.0	24
2	2	ALCATEL	1 5033D 1/16GB Volcano Black (5033D-2LALUAF)	Android	299	1803.0	1659.0	24
3	3	ALCATEL	1 5033D 1/16GB Volcano Black (5033D-2LALUAF)	Android	287	1803.0	1659.0	24
4	4	Nokia	1.3 1/16GB Charcoal	Android	1047	1999.0	NaN	
...	...	...	...	...	...	...	...	
1219	1219	Apple	iPhone XS Max 64GB Gold (MT522)	iOS	1101	22685.0	16018.0	279
1220	1220	Apple	iPhone XS Max Dual Sim 64GB Gold (MT732)	iOS	530	24600.0	21939.0	337
1221	1221	HUAWEI	nova 5T 6/128GB Black (51094MEU)	Android	1174	8804.0	7999.0	99
1222	1222	ZTE	nubia Red Magic 5G 8/128GB Black	Android	752	18755.0	18500.0	190
1223	1223	Sigma mobile	x-style 35 Screen	NaN	952	907.0	785.0	9

1224 rows × 13 columns

In [27]:

```
#removng higest price column ust to focus on lowest and only to predict that.
new_data=new_data.drop('highest_price', axis=1)
```

In [28]:

```
new_data.isna().sum()
```

Out[28]:

```
Unnamed: 0          0
brand_name          0
model_name          0
os                197
popularity          0
best_price          0
lowest_price      260
sellers_amount      0
screen_size         2
memory_size       112
battery_size       10
release_date        0
dtype: int64
```

In [29]:

```
#Filling NaN of os
new_data['os'].fillna('missing', inplace=True)
#Filling Nan of memorysize
new_data['memory_size'].fillna(new_data['memory_size'].mean, inplace=True)
#Filling Nan of battery_size
new_data['battery_size'].fillna(new_data['battery_size'].mean, inplace=True)
#Filling Nan of screen_size
new_data['screen_size'].fillna(new_data['screen_size'].mean, inplace=True)
```

In [30]:

```
new_data.isna().sum()
```

Out[30]:

```
Unnamed: 0          0
brand_name          0
model_name          0
os                  0
popularity          0
best_price          0
lowest_price      260
sellers_amount      0
screen_size         0
memory_size         0
battery_size        0
release_date        0
dtype: int64
```

#Here E.g if we have to predict lowest price, we will not add mean values to lowest price bec it would create error. THerefore we will remove it

In [31]:

```
new_data.dropna(inplace=True)
```

```
In [32]: new_data.isna().sum()
```

```
Out[32]: Unnamed: 0        0
         brand_name        0
         model_name        0
         os                0
         popularity        0
         best_price        0
         lowest_price      0
         sellers_amount    0
         screen_size       0
         memory_size       0
         battery_size      0
         release_date      0
         dtype: int64
```

No NaN value anymore.

Method 2 to Fill Values

data filling in sklearn is called imputation. for that impute library is used. imputer= ColumnTransformer([(name, calling_imputer, column_name imputer to be applied)])

In [5]:
```
new_data=pd.read_csv('phones_data.csv')
new_data
```

Out[5]:

	Unnamed: 0	brand_name	model_name	os	popularity	best_price	lowest_price	highest_
0	0	ALCATEL	1 1/8GB Bluish Black (5033D-2JALUAA)	Android	422	1690.0	1529.0	1
1	1	ALCATEL	1 5033D 1/16GB Volcano Black (5033D-2LALUAF)	Android	323	1803.0	1659.0	2
2	2	ALCATEL	1 5033D 1/16GB Volcano Black (5033D-2LALUAF)	Android	299	1803.0	1659.0	2
3	3	ALCATEL	1 5033D 1/16GB Volcano Black (5033D-2LALUAF)	Android	287	1803.0	1659.0	2
4	4	Nokia	1.3 1/16GB Charcoal	Android	1047	1999.0	NaN	
...	...	...	...	...	...	...	...	
1219	1219	Apple	iPhone XS Max 64GB Gold (MT522)	iOS	1101	22685.0	16018.0	27
1220	1220	Apple	iPhone XS Max Dual Sim 64GB Gold (MT732)	iOS	530	24600.0	21939.0	33
1221	1221	HUAWEI	nova 5T 6/128GB Black (51094MEU)	Android	1174	8804.0	7999.0	9
1222	1222	ZTE	nubia Red Magic 5G 8/128GB Black	Android	752	18755.0	18500.0	19
1223	1223	Sigma mobile	x-style 35 Screen	NaN	952	907.0	785.0	

1224 rows × 13 columns

In [6]:

```
new_data.isna().sum()
```

Out[6]:

```
Unnamed: 0          0
brand_name          0
model_name          0
os                197
popularity          0
best_price          0
lowest_price      260
highest_price     260
sellers_amount      0
screen_size         2
memory_size       112
battery_size       10
release_date        0
dtype: int64
```

In [7]:

```
#Consider lowest_price as center-point of data whih is to be predicted.
new_data.dropna(subset=["lowest_price"], inplace=True)
new_data.isna().sum()
```

Out[7]:

```
Unnamed: 0          0
brand_name          0
model_name          0
os                173
popularity          0
best_price          0
lowest_price        0
highest_price       0
sellers_amount      0
screen_size         2
memory_size       101
battery_size       10
release_date        0
dtype: int64
```

Now, all rows containing NaN values are removed.

In [8]:

```
#Sp;itting data into X and Y
x1=new_data.drop('lowest_price', axis=1)
y1=new_data['lowest_price']
```

In [9]:
```
new_data.isna().sum()
```

Out[9]:
```
Unnamed: 0          0
brand_name          0
model_name          0
os                173
popularity          0
best_price          0
lowest_price        0
highest_price       0
sellers_amount      0
screen_size         2
memory_size       101
battery_size       10
release_date        0
dtype: int64
```

In [34]:

```python
#Imputation
#Filling data with SKlearn.
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

#Fill categorical values with Numerical and missing values.
#Creating imputers to be applied on particular columns which have NaN values
os_feature=SimpleImputer(strategy="constants", fill_value="missing")
screen_feature=SimpleImputer(strategy="mean")
memory_feature=SimpleImputer(strategy="mean")

#Defining Columns:
#os_feature=["os"]
screen_feature=["screen_size"]
memory_size=["memory_size"]

#Applyting imputer: ColumnTransformer([("name to be given", calling imputer to be
imputer= ColumnTransformer([
                            ("screen_feature", screen_feature, screen_feature),
                            ("memory_feature", memory_feature, memory_feature)])

#fiiting model:

filled_x1= imputer.fit_transform(x1)

filled_x1
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-34-4bb9e96f3967> in <module>
     22 #fiiting model:
     23 
---> 24 filled_x1= imputer.fit_transform(x1)
     25 
     26 filled_x1

~\anaconda3\lib\site-packages\sklearn\compose\_column_transformer.py in fit_t
ransform(self, X, y)
    525         # set n_features_in_ attribute
    526         self._check_n_features(X, reset=True)
--> 527         self._validate_transformers()
    528         self._validate_column_callables(X)
    529         self._validate_remainder(X)

~\anaconda3\lib\site-packages\sklearn\compose\_column_transformer.py in _vali
date_transformers(self)
    285             if (not (hasattr(t, "fit") or hasattr(t, "fit_transfor
m")) or not
    286                     hasattr(t, "transform")):
--> 287                 raise TypeError("All estimators should implement fit
 and "
    288                                 "transform, or can be 'drop' or 'pass
```

```
through' "
    289                                         "specifiers. '%s' (type %s) doesn't."
%

TypeError: All estimators should implement fit and transform, or can be 'dro
p' or 'passthrough' specifiers. '['screen_size']' (type <class 'list'>) does
n't.
```

In [20]:
```
new_data_1=pd.read_csv("auto-mpg.csv")
```

In [21]:
```
new_data_1.isna().sum()
```

Out[21]:
```
mpg              2
cylinders        0
displacement     0
horsepower       2
weight          10
acceleration     1
model year       1
origin          14
car name         7
dtype: int64
```

In [28]:
```
#Consider Lowest_price as center-point of data whih is to be predicted.
new_data_1.dropna(subset=["origin", "acceleration", "mpg", "model year"], inplace
new_data_1.isna().sum()
```

Out[28]:
```
mpg             0
cylinders       0
displacement    0
horsepower      2
weight          9
acceleration    0
model year      0
origin          0
car name        7
dtype: int64
```

In [24]:
```
#Sp;itting data into X and Y
x2=new_data_1.drop('origin', axis=1)
y2=new_data_1['origin']
```

In [33]:

```
#Imputation
#Filling data with SKlearn.
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

#Fill categorical values with Numerical and missing values.
#Creating imputers to be applied on particular columns which have NaN values
name_feature=SimpleImputer(strategy="constants", fill_value="missing")
horsepower_feature=SimpleImputer(strategy="mean")
weight_feature=SimpleImputer(strategy="mean")

#Defining Columns:
name_feature=["weight"]
horsepower=["horsepower"]
weight_size=["car name"]

#Applyting imputer: ColumnTransformer([("name to be given", calling imputer to be
imputer= ColumnTransformer([("name_feature", name_feature, name_feature),
                            ("horsepowern_feature", horsepower_feature, horsepowe
                            ("weight_feature", weight_feature, weight_feature)])

#fiiting model:

filled_x2= imputer.fit_transform(x2)

filled_x2
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-33-61827cbb5f22> in <module>
     22 #fiiting model:("weight_feature", weight_feature, weight_feature)
     23 
---> 24 filled_x2= imputer.fit_transform(x2)
     25 
     26 filled_x2

~\anaconda3\lib\site-packages\sklearn\compose\_column_transformer.py in fit_t
ransform(self, X, y)
    527         self._validate_transformers()
    528         self._validate_column_callables(X)
--> 529         self._validate_remainder(X)
    530 
    531         result = self._fit_transform(X, y, _fit_transform_one)

~\anaconda3\lib\site-packages\sklearn\compose\_column_transformer.py in _vali
date_remainder(self, X)
    317 
    318         # Make it possible to check for reordered named columns on tr
ansform
--> 319         self._has_str_cols = any(_determine_key_type(cols) == 'str'
    320                                  for cols in self._columns)
    321         if hasattr(X, 'columns'):
```

```
~\anaconda3\lib\site-packages\sklearn\compose\_column_transformer.py in <gene
xpr>(.0)
    317 
    318         # Make it possible to check for reordered named columns on tr
ansform
--> 319         self._has_str_cols = any(_determine_key_type(cols) == 'str'
    320                                  for cols in self._columns)
    321         if hasattr(X, 'columns'):

~\anaconda3\lib\site-packages\sklearn\utils\__init__.py in _determine_key_typ
e(key, accept_slice)
    268         except KeyError:
    269             raise ValueError(err_msg)
--> 270     raise ValueError(err_msg)
    271 
    272 

ValueError: No valid specification of the columns. Only a scalar, list or sli
ce of all integers or all strings, or boolean mask is allowed
```

IDK why this error is showing, will find out later on.