Department of Data Science

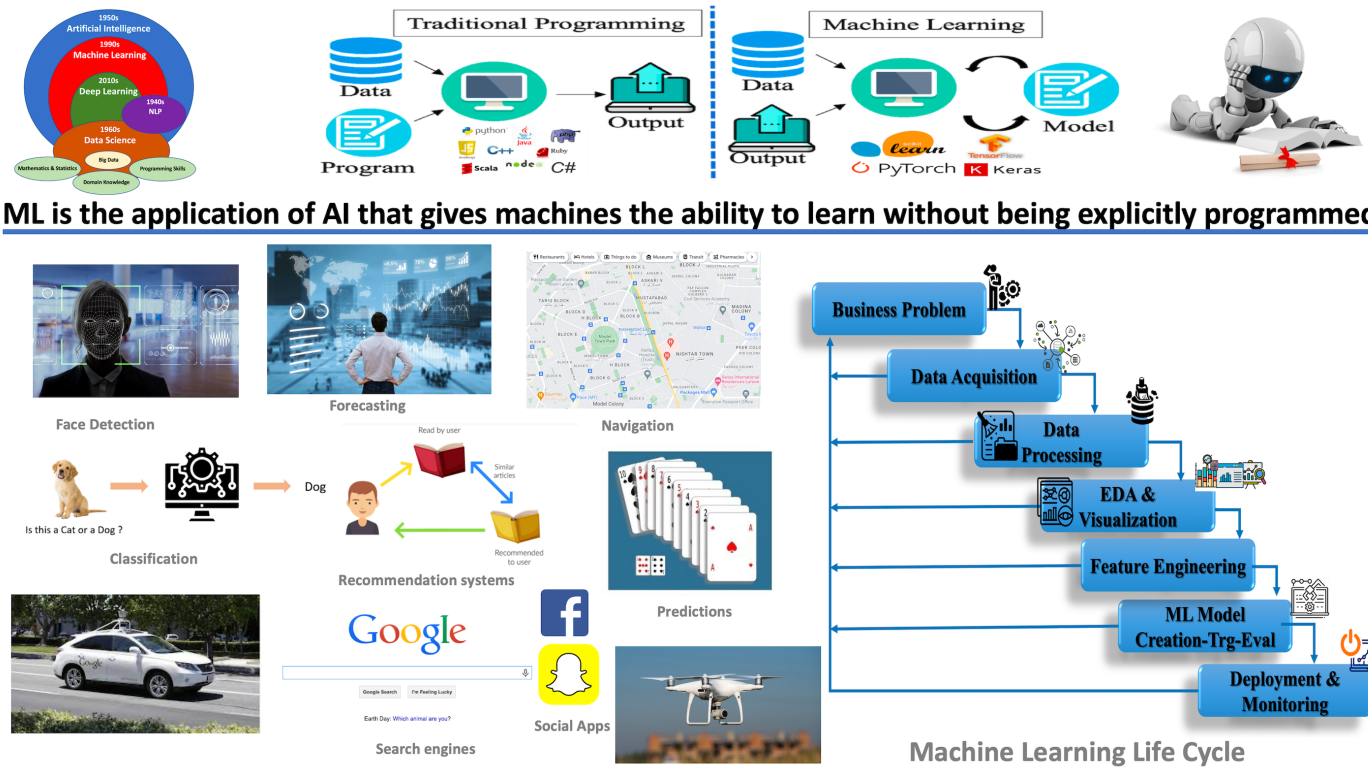
Course: Tools and Techniques for Data Science

Instructor: Muhammad Arif Butt, Ph.D.

Lecture 6.16 (Building a Machine Learning Pipeline (A-Z))

Open in Colab

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



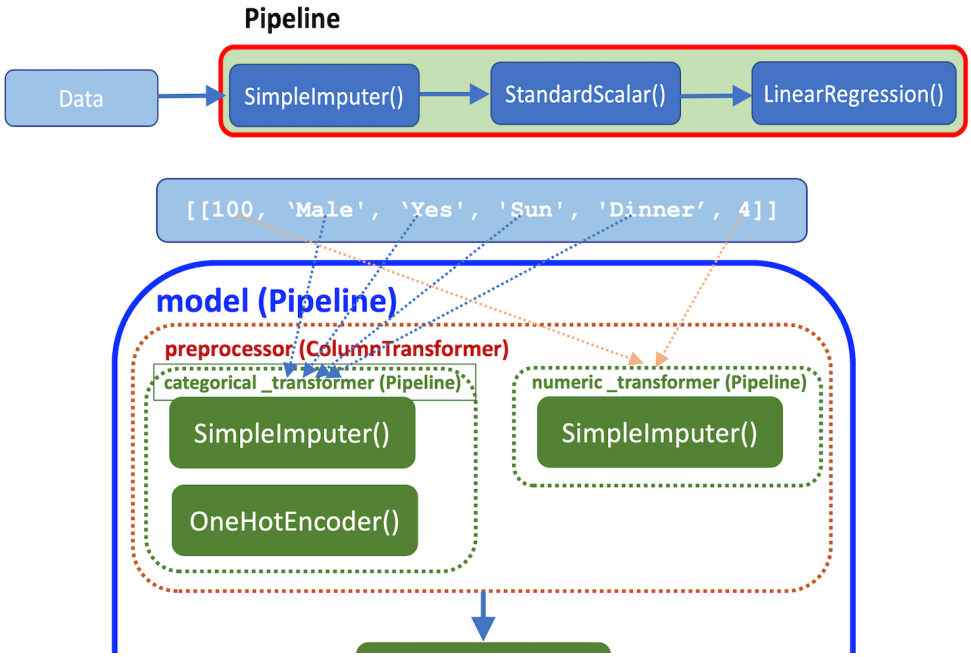
In [ ]:

```
1
```

Learning agenda of this notebook

- Data Preprocessing Techniques and Sklearn Transformers
  - Detecting and handling outliers
  - Missing values Imputation
  - Encoding Categorical Features
  - Feature Scaling
  - Extracting Information
  - Combining Information
- **Example 1:** Applying Transformers on Individual Columns

- **Example 2:** Use of sklearn's `ColumnTransformer` class
- **Overview of Machine Learning Pipeline**
  - What is a Machine Learning Pipeline?
  - Why to create a Machine Learning Pipeline?
  - How to create a Machine Learning Pipeline?
- **Example 3:** Use of sklearn's `Pipeline` Object
- **Example 4:** Use of sklearn's `Pipeline` Object(Advance)



1. Recap of Data Preprocessing Techniques and Scikit-Learn Transformers

- `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.

a. Detecting and handling outliers

b. Missing values Imputation

c. Encoding Categorical Features

d. Feature Scaling

e. Extracting Information

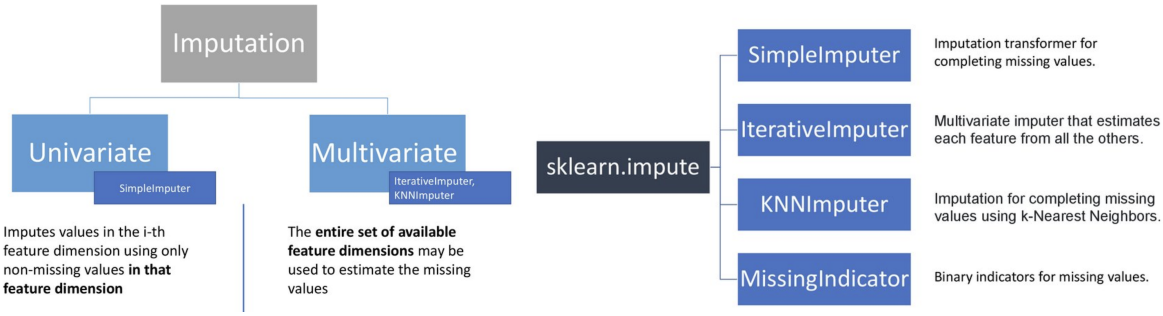
f. Combining Information

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

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

a. Missing Value Imputation

- **Why do we need to handle missing values?**
  - If the missing values are not handled properly, you may end up building a biased machine learning model which will lead to incorrect results.
  - Many machine learning algorithms fail if the dataset contains missing values. However, algorithms like K-nearest and Naive Bayes support data with missing values.
- **How to treat missing values?**
  - Analyze each column with missing values carefully to understand the reasons behind the missing values as it is crucial to find out the strategy for handling the missing values. There are 2 primary ways of handling missing values:
    - Deleting the Missing values: Drop rows (List-wise deletion) having missing values or drop the entire column
    - Imputing the Missing Values: Replace the missing value with a value
      - `Univariate Imputation`
        - Handling Missing Values using Panda's `fillna()` method
        - Handling Missing Values using sklearn's `SimpleImputer()` transformer
        - Use of Column Transformer
      - `Multivariate Imputation`
        - Handling Missing Values using sklearn's `IterativeImputer()` transformer
        - Handling Missing Values using sklearn's `KNNImputer()` transformer



b. Encoding Categorical Features

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

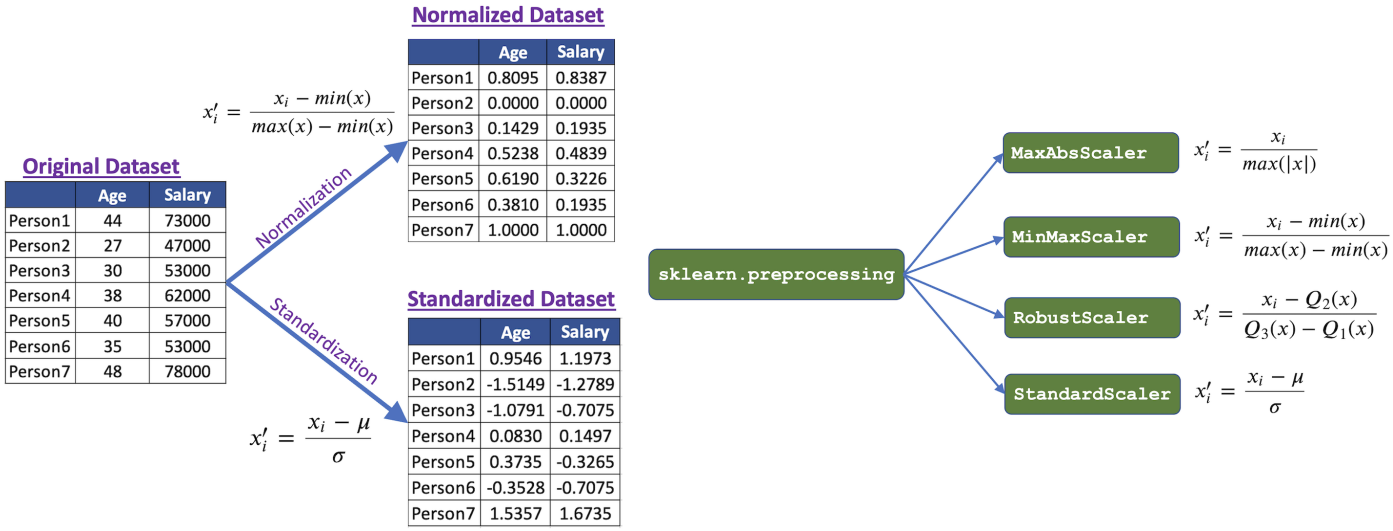
Continuous: Values that can

Data Types

Nominal: Values that can be slotted

c. Feature Scaling

Feature scaling is a process of transforming numeric columns to a common scale



	Age	Salary
Person1	44	73000
Person2	27	47000
Person3	30	53000
Person4	38	62000
Person5	40	57000
Person6	35	53000
Person7	48	78000

	Age	Salary
Person1	0.8095	0.8387
Person2	0.0000	0.0000
Person3	0.1429	0.1935
Person4	0.5238	0.4839
Person5	0.6190	0.3226
Person6	0.3810	0.1935
Person7	1.0000	1.0000

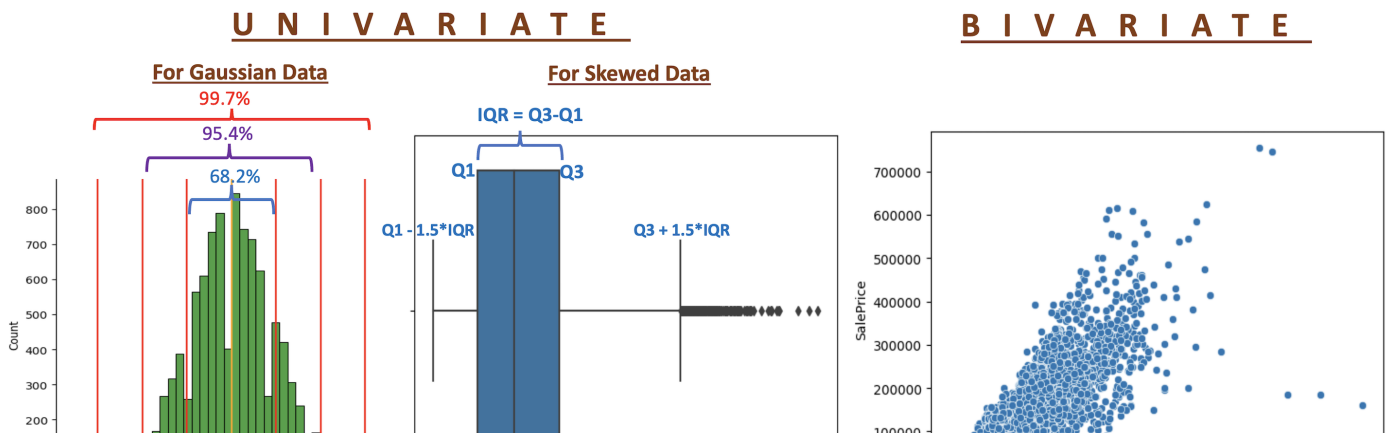
	Age	Salary
Person1	0.9546	1.1973
Person2	-1.5149	-1.2789
Person3	-1.0791	-0.7075
Person4	0.0830	0.1497
Person5	0.3735	-0.3265
Person6	-0.3528	-0.7075
Person7	1.5357	1.6735

- **Normalization** is rescaling of the data from original range, so that all values are within the new range of 0 and 1.
- **Standardization** is rescaling of the data from original range, so that all values are centered around mean with a standard deviation of 1.
- `Note:`
  - After scaling the distribution of data does not change.
  - Use standardization if your data is normally distributed, otherwise, use normalization
  - Standardization is more robust to outliers.

d. Handling Outliers

An outlier is a data point that differs significantly from other observations

- **How to identify outliers?**
  - `Z Score Method:` $x_i < \mu - 3\sigma$ OR $x_i > \mu + 3\sigma$
  - `IQR Method`
  - `Percentiles Method`
  - Multivariate Analysis using Scatter Plot
  - There are a bundle of methods that you may come accross in literature like:
  - Depth-based methods
  - Distance-based methods
  - Density-based methods
  - Mahalanobis distance based methods



2. Practical Recap of Preprocessing using Scikit-Learn Transformers

Example 1: Applying Transformers on Individual Columns

Load Dataset

In [1]:

```
1 import pandas as pd
2 import numpy as np
3 df = pd.read_csv('datasets/salary-package1.csv')
4 df
```

Out[1]:

	gender	cgpa	iq	salary
0	Male	2.3	Medium	68
1	Female	3.8	High	100
2	Male	3.0	Medium	75
3	Female	NaN	Low	48
4	Female	2.3	High	97
...	...	...	...	...
95	Female	2.1	Low	40
96	Female	2.3	High	68
97	Female	3.8	High	100
98	Female	3.0	Medium	75
99	Female	NaN	High	80

100 rows × 4 columns

In [2]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   gender  100 non-null    object 
 1   cgpa    90 non-null     float64
 2   iq      100 non-null    object 
 3   salary  100 non-null    int64  
dtypes: float64(1), int64(1), object(2)
memory usage: 3.2+ KB
```

In [3]:

```
df.isnull().sum()
```

Out[3]:

```
gender     0
cgpa      10
iq         0
salary     0
dtype: int64
```

In [4]:

```
df.gender.unique()
```

Out[4]:

```
array(['Male', 'Female'], dtype=object)
```

In [5]:

```
df.iq.unique()
```

Out[5]:

```
array(['Medium', 'High', 'Low'], dtype=object)
```

Train-Test Split

In [6]:

```
from sklearn.model_selection import train_test_split
X = df.drop('salary', axis=1)
y = df['salary']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
X_train.head()
```

Out[6]:

	gender	cgpa	iq
0	Male	2.3	Medium
1	Female	3.8	High
2	Male	3.0	Medium
3	Female	NaN	Low
4	Female	2.3	High

Apply `SimpleImputer` Transformer on `age` Column

In [7]:

```
from sklearn.impute import SimpleImputer
# define imputer
si = SimpleImputer(missing_values=np.nan, strategy='mean')
#train it
si.fit(X_train[['cgpa']])

# transform() will return a numpy array which is being assigned to the cgpa column of dataframe
X_train['cgpa'] = si.transform(X_train[['cgpa']])
X_test['cgpa'] = si.transform(X_test[['cgpa']])
X_train.head()
```

Out[7]:

	gender	cgpa	iq
0	Male	2.300000	Medium
1	Female	3.800000	High
2	Male	3.000000	Medium
3	Female	2.884507	Low
4	Female	2.300000	High

In [8]:

```
X_train.isnull().sum()
```

Out[8]:

```
gender    0
cgpa      0
iq        0
dtype: int64
```

Apply `OneHotEncoder` Transformer on `gender` Column

In [9]:

```
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder(drop='first',sparse=False,dtype=np.int32)
#train
ohe.fit(X_train[['gender']])

# transform() will return a numpy array which is being assigned to the gender column of datafram
X_train['gender'] = ohe.transform(X_train[['gender']])
X_test['gender'] = ohe.transform(X_test[['gender']])
X_train.head()
```

Out[9]:

	gender	cgpa	iq
0	1	2.300000	Medium
1	0	3.800000	High
2	1	3.000000	Medium
3	0	2.884507	Low
4	0	2.300000	High

Apply `OrdinalEncoder` Transformer on `iq` Column

In [10]:

```
from sklearn.preprocessing import OrdinalEncoder
oe = OrdinalEncoder(categories=[['Low', 'Medium', 'High']], dtype=np.int8)
#train
oe.fit(X_train[['iq']])
# transform() will return a numpy array which is being assigned to the gender column of datafram
X_train['iq'] = oe.transform(X_train[['iq']])
X_test['iq'] = oe.transform(X_test[['iq']])
#verify
X_train.head()
```

Out[10]:

	gender	cgpa	iq
0	1	2.300000	1
1	0	3.800000	2
2	1	3.000000	1
3	0	2.884507	0
4	0	2.300000	2

Instentiate and Train `LinearRegression()` Model

In [11]:

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(X_train,y_train)
```

Out[11]:

```
LinearRegression()
```

Predict

In [12]:

```
test_input = np.array(['Male', 3.5, 'High']).reshape(1,3)
test_input_transformed = np.array([1, 3.5, 2]).reshape(1,3)
test_input, test_input_transformed
```

Out[12]:

```
(array([['Male', '3.5', 'High']], dtype='<U32'), array([[1. , 3.5, 2. ]]))
```

In [13]:

```
import warnings
warnings.filterwarnings('ignore')
model.predict(test_input_transformed)
```

Out[13]:

```
array([73.12721804])
```

Example 2: Repeat Example 1 using Scikit-Learn's `ColumnTransformer` Class

[ColumnTransformer Class in Scikit-Learn: (https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html)](https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html)

ColumnTransformer allows to apply different Transformers on different columns of our

dataset in a simple and elegant way

Load Dataset

In [14]:

```
import pandas as pd
import numpy as np
df = pd.read_csv('datasets/salary-package1.csv')
df
```

Out[14]:

	gender	cgpa	iq	salary
0	Male	2.3	Medium	68
1	Female	3.8	High	100
2	Male	3.0	Medium	75
3	Female	NaN	Low	48
4	Female	2.3	High	97
...	...	...	...	...
95	Female	2.1	Low	40
96	Female	2.3	High	68
97	Female	3.8	High	100
98	Female	3.0	Medium	75
99	Female	NaN	High	80

100 rows × 4 columns

Train-Test Split

In [15]:

```
from sklearn.model_selection import train_test_split
X = df.drop('salary', axis=1)
y = df['salary']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

Create `ColumnTransformer` Object, Train and Transform

In [16]:

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder
from sklearn.compose import ColumnTransformer

col_tfm = ColumnTransformer(
                          transformers=[
                                        ('tfm1', OneHotEncoder(drop='first',sparse=False),    ['gende
                                        ('tfm2', SimpleImputer(missing_values=np.nan, strategy='mean
                                        ('tfm3', OrdinalEncoder(categories=[['Low', 'Medium', 'High'
                                         ],
                          remainder='passthrough')
type(col_tfm)
```

Out[16]:

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

In [17]:

```
col_tfm.fit(X_train)
```

Out[17]:

```
ColumnTransformer(remainder='passthrough',
                  transformers=[('tfm1',
                                 OneHotEncoder(drop='first', sparse=False),
                                 ['gender']),
                                ('tfm2', SimpleImputer(), ['cgpa']),
                                ('tfm3',
                                 OrdinalEncoder(categories=[['Low', 'Medium',
                                                             'High']]),
                                 ['iq'])])
```

In [18]:

```
arr_Xtrain = col_tfm.transform(X_train)
arr_Xtest = col_tfm.transform(X_test)
arr_Xtest
```

Out[18]:

```
array([[0.        , 2.3       , 0.        ],
       [1.        , 3.8       , 0.        ],
       [1.        , 3.        , 1.        ],
       [0.        , 2.1       , 0.        ],
       [0.        , 2.3       , 2.        ],
       [0.        , 3.1       , 0.        ],
       [1.        , 2.4       , 0.        ],
       [1.        , 1.8       , 2.        ],
       [0.        , 1.1       , 0.        ],
       [1.        , 3.2       , 2.        ],
       [0.        , 3.6       , 1.        ],
       [1.        , 4.        , 0.        ],
       [0.        , 3.1       , 2.        ],
       [1.        , 2.8       , 0.        ],
       [1.        , 3.7       , 2.        ],
       [0.        , 2.1       , 0.        ],
       [0.        , 2.3       , 2.        ],
       [0.        , 3.8       , 2.        ],
       [0.        , 3.        , 1.        ],
       [0.        , 2.88450704, 2.        ]])
```

Instentiate and Train `LinearRegression()` Model

In [19]:

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(arr_Xtrain, y_train)
```

Out[19]:

```
LinearRegression()
```

Predict

In [20]:

```
test_input = np.array([['Male', 3.5, 'High']])
test_input_transformed = np.array([[1, 3.5, 2]])
test_input
```

Out[20]:

```
array([['Male', '3.5', 'High']], dtype='<U32')
```

In [21]:

```
test_input = np.array(['Male', 3.5, 'High']).reshape(1,3)
test_input_transformed = np.array([1, 3.5, 2]).reshape(1,3)
test_input
```

Out[21]:

```
array([['Male', '3.5', 'High']], dtype='<U32')
```

In [22]:

```
import warnings
warnings.filterwarnings('ignore')
model.predict(test_input_transformed)
```

Out[22]:

```
array([73.12721804])
```

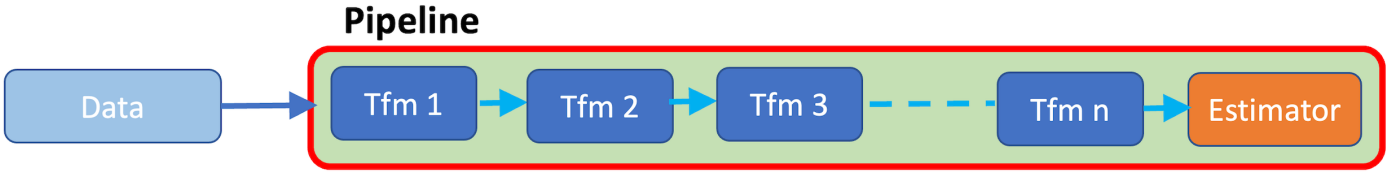
Limitations of `ColumnTransformer` :

- ColumnTransformer returns a numPy array even if we input a DataFrame object which makes it difficult to track the columns, i.e., we cannot track columns by name rather we have to track them by index.
- In a ColumnTransformer we cannot apply multiple transforms to a single column.

3. Overview of Machine Learning Pipeline

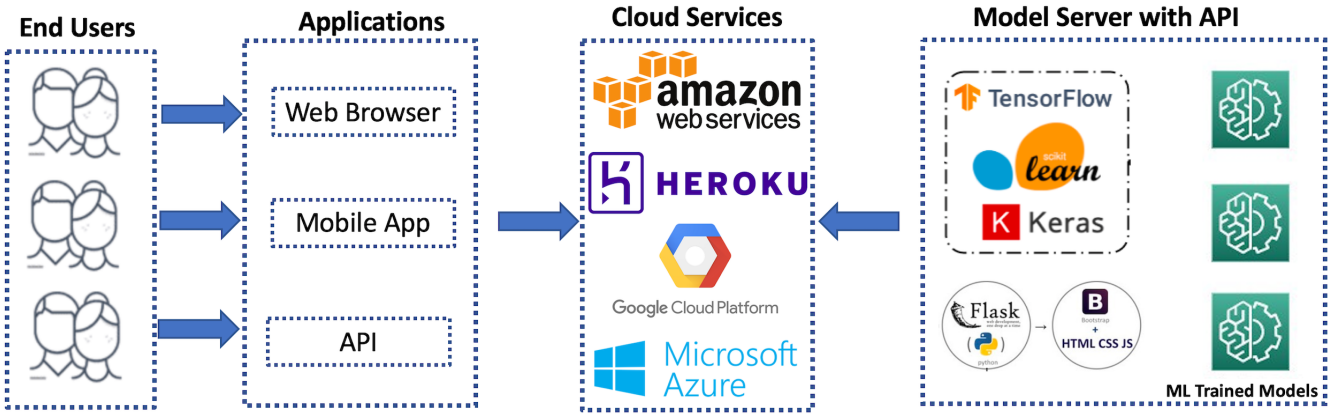
What is a Machine Learning Pipeline?

Pipeline is a mechanism that chains together multiple steps together so that the o/p of each step is used as i/p to the next step



A machine learning pipeline is simply a set of steps that you follow while working on your project.
This could include things like acquiring and preprocessing your data, training, hypertuning and validating models, and finally deploying them to make predictions.

Why to Create a Machine Learning Pipeline?



How to Create a Machine Learning Pipeline?

[ML Pipeline Object in Scikit-Learn: (https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html)](https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html)

Example 3: Use of Scikit-Learn Pipeline Object

Load Dataset

In [23]:

```
import pandas as pd
import numpy as np
df = pd.read_csv('datasets/advertising-withmissingdata.csv')
df
```

Out[23]:

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	NaN	45.1	10.4
2	NaN	45.9	69.3	9.3
3	151.5	41.3	NaN	18.5
4	180.8	10.8	58.4	12.9
...	...	...	...	...
195	38.2	3.7	13.8	7.6
196	94.2	NaN	8.1	9.7
197	177.0	9.3	6.4	12.8
198	283.6	42.0	NaN	25.5
199	NaN	8.6	8.7	13.4

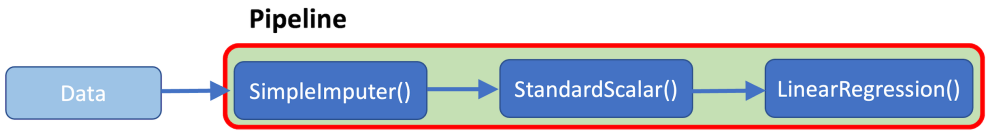
200 rows × 4 columns

In [24]:

```
df.isnull().sum()
```

Out[24]:

```
TV           4
radio       10
newspaper    2
sales        0
dtype: int64
```

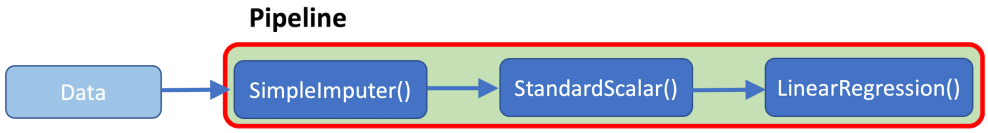


Train-Test Split

In [25]:

```
from sklearn.model_selection import train_test_split
X = df.drop('sales', axis=1)
y = df['sales']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=54)
```

Create Pipeline Object, Train and Predict



In [26]:

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline

# steps argument is a list of two valued tuples unlike transformers argument of ColumnTransforme
# SimpleImputer and StandardScaler will be applied to all the columns of the dataset
pipe = Pipeline(
                steps=[
                      ('si',  SimpleImputer(missing_values=np.nan, strategy='mean')),
                      ('ss', StandardScaler()),
                      ('lr', LinearRegression())
                     ]
                )
pipe
```

Out[26]:

```
Pipeline(steps=[('si', SimpleImputer()), ('ss', StandardScaler()),
                ('lr', LinearRegression())])
```

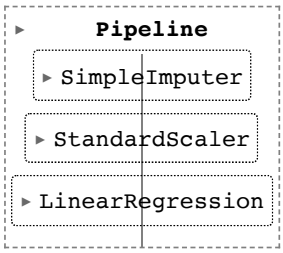
In [27]:

```
from sklearn import set_config
set_config(display='diagram')
```

In [28]:

```
pipe
```

Out[28]:



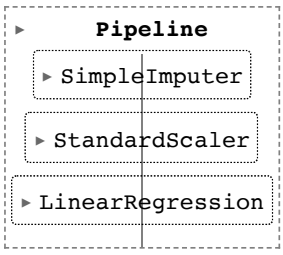
Train the Pipeline instead of Model

- On this specific pipeline object, when you call the `fit()` method (by passing it raw or unprocessed training data), it will internally call the `fit_transform()` method of the SimpleImputer, StandardScalar and the `fit()` method of LinearReression estimator.
- After this you have a trained pipeline object (as fit is inplace)

In [29]:

```
pipe.fit(X_train, y_train)
```

Out[29]:



Predict using the Pipeline instead of Model

- When you call the `predict()` method on this trained pipeline object (by passing raw or unprocessed test data), it will internally call only the `transform()` method of the SimpleImputer and StandardScalar in sequence. And then will call the `predict()`

method of the LinearRegression estimator

In [30]:

```
pipe.predict(X_test)
```

Out[30]:

```
array([11.5103185 , 10.74314548, 17.06532765,  6.81320042,  9.7687584 ,
       11.81989768, 19.25928657, 10.56907364, 15.59736803, 16.11914409,
        9.73165545, 14.03714358, 16.43272133, 17.27575117,  8.87968169,
       23.35102195,  8.20884078, 13.29275029,  9.79142007, 11.38200247,
       18.68712032, 12.0133993 , 18.87632571, 20.56500684, 14.01595916,
       13.54109255, 18.17737359, 24.17518038,  9.15739432, 14.75927552,
       19.98104419,  9.76168882, 19.52101244, 21.27322187, 18.19830953,
       20.92010821,  6.60271388, 15.34461646, 20.26487212,  6.14806508])
```

Evaluate using the Pipeline instead of Model

In [31]:

```
pipe.score(X_test, y_test)
```

Out[31]:

```
0.850615741795994
```

In [32]:

```
# You can call methods on a specific transformer of this trained pipeline object
# Let us call fit_transform() on StandardScalar, will return a NumPy array after scaling the dat
pipe['ss'].fit_transform(X_train)
```

Out[32]:

```
array([[-0.30146403, -0.2337849 , -0.82091279],
       [        nan, -0.81991227, -1.09359501],
       [ 0.7711532 ,  0.4256084 , -0.93290727],
       [-0.59184107, -1.44600287, -1.02542445],
       [ 1.55576382,  1.71775284,  0.64962348],
       [-0.82177228,  0.29239763, -0.60666247],
       [-0.42472612, -0.58679343,  0.15782162],
       [-0.29198233, -0.97976519,  0.98560693],
       [-0.83599483,  0.79193801,  1.18037995],
       [-0.60961925,         nan, -0.99133917],
       [ 0.12995329, -1.36607641, -0.98160052],
       [-0.42472612,  1.64448692,  1.11707872],
       [ 0.80315394,         nan, -0.74787291],
       [-0.88932938, -0.47356428,  0.60093023],
       [ 1.63517305,  1.25817569,         nan],
       [ 1.37087068,  1.30479946,  1.27776645],
       [ 1.64346954, -0.83323335, -1.07411771],
       [-0.48635716, -1.15959972,  0.2892934 ],
```

In [33]:

```
# Let us call fit_transform() on SimpleImputer, will return a NumPy array after imputing the mis
pipe['si'].fit_transform(X_train)
```

Out[33]:

```
array([[120.2       ,  19.6       ,  11.6       ],
       [145.63544304,  10.8       ,   6.        ],
       [210.7       ,  29.5       ,   9.3       ],
       [ 95.7       ,   1.4       ,   7.4       ],
       [276.9       ,  48.9       ,  41.8       ],
       [ 76.3       ,  27.5       ,  16.        ],
       [109.8       ,  14.3       ,  31.7       ],
       [121.        ,   8.4       ,  48.7       ],
       [ 75.1       ,  35.        ,  52.7       ],
       [ 94.2       ,  23.11      ,   8.1       ],
       [156.6       ,   2.6       ,   8.3       ],
       [109.8       ,  47.8       ,  51.4       ],
       [213.4       ,  23.11      ,  13.1       ],
       [ 70.6       ,  16.        ,  40.8       ],
       [283.6       ,  42.        ,  28.45886076],
       [261.3       ,  42.7       ,  54.7       ],
       [284.3       ,  10.6       ,   6.4       ],
       [104.6       ,   5.7       ,  34.4       ],
```

In [34]:

```
# Let us call fit() on estimator, will raise an error because X_train contains NaN values
#pipe['lr'].fit(X_train, y_train)
```

Predict

In [35]:

```
TV = 60
radio = np.nan
newspaper = 40
test_input = np.array([[TV, radio, newspaper]])
test_input = np.array([TV, radio, newspaper]).reshape(1,3)
test_input
```

Out[35]:

```
array([[60., nan, 40.]])
```

In [36]:

```
pipe.predict(test_input)
```

Out[36]:

```
array([10.0297736])
```

Example 4: Using ColumnTransformer in a Pipeline

- A transformer in a pipeline is applied on the entire input dataset or the o/p of the previous transformer.
- We can have missing values in numeric variables as well as missing values in categorical variables, so we have to use different imputation techniques on numeric columns and different techniques on categorical columns.
- Moreover we have to apply different encoding techniques to categorical columns, for nominal data we apply OneHotEncoder while for ordinal data we apply OrdinalEncoder
- At times we want to apply different transformation to different columns of our dataset. For this we can create multiple pipelines and later combine them together
  - For processing numeric data we can create a separate pipeline
  - For processing categorical data we can create a separate pipeline
  - Later we can combine these two pipelines to a single one, and later any transformer that we place in the pipeline will be applied to the entire output dataset of the previous step of pipeline :)

Load Dataset

In [37]:

```
import numpy as np
import pandas as pd
import seaborn as sns

#df = sns.load_dataset('tips')
df = pd.read_csv('datasets/tips-missingdata.csv')
df
```

Out[37]:

	total_bill	sex	smoker	day	time	size	tip
0	16.99	Female	No	Sun	Dinner	2	NaN
1	NaN	Male	NaN	Sun	Dinner	3	1.66
2	21.01	Male	No	Sun	Dinner	3	3.50
3	23.68	NaN	No	Sun	Dinner	2	3.31
4	24.59	Female	No	Sun	Dinner	4	3.61
...	...	...	...	...	...	...	...
239	29.03	Male	No	Sat	Dinner	3	5.92
240	27.18	Female	Yes	Sat	Dinner	2	2.00
241	22.67	Male	Yes	Sat	Dinner	2	2.00
242	NaN	Male	No	Sat	Dinner	2	1.75
243	18.78	Female	No	Thur	Dinner	2	3.00

244 rows × 7 columns

In [38]:

```
df.dtypes
```

Out[38]:

```
total_bill    float64
sex            object
smoker         object
day            object
time           object
size            int64
tip           float64
dtype: object
```

In [39]:

```
df.day.unique()
```

Out[39]:

```
array(['Sun', 'Sat', 'Thur', 'Fri'], dtype=object)
```

In [40]:

```
df.isna().sum()
```

Out[40]:

```
total_bill    3
sex           2
smoker        1
day           0
time          0
size          0
tip           1
dtype: int64
```

Drop rows where o/p label `tip` has missing values

In [41]:

```
df.dropna(axis=0, how='any', subset=['tip'], inplace=True)
df.isna().sum()
```

Out[41]:

```
total_bill    3
sex           2
smoker        1
day           0
time          0
size          0
tip           0
dtype: int64
```

In [42]:

```
df.shape
```

Out[42]:

```
(243, 7)
```

Train-Test Split

In [43]:

```
from sklearn.model_selection import train_test_split
X = df.drop('tip', axis=1)
y = df['tip']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=54)
```

> If we impute missing values under `total_bill` column before splitting it into train and test with the `mean()` of the whole column values. Then, this would result in information leakage, i.e., using information from the test set to fill the training set.

Design of Pipeline

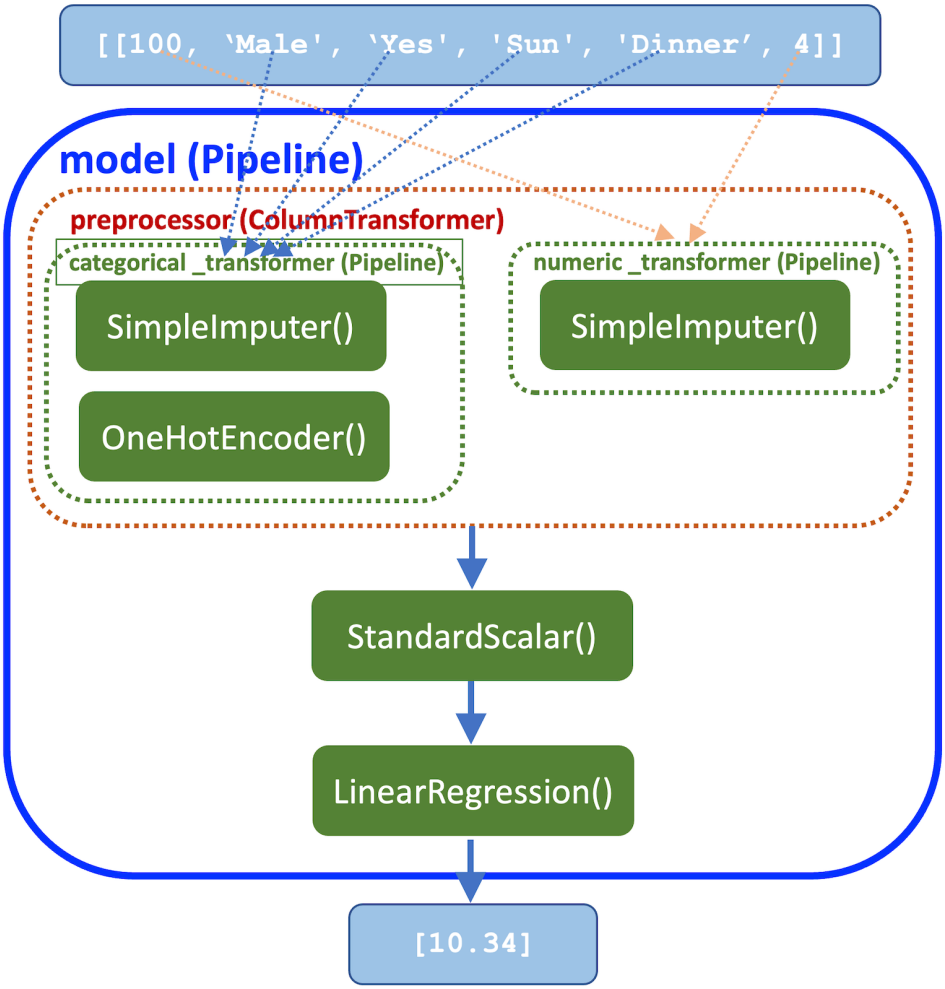
In [44]:

```
1 X_train
```

Out[44]:

	total_bill	sex	smoker	day	time	size
35	24.06	Male	No	Sat	Dinner	3
56	38.01	Male	Yes	Sat	Dinner	4
6	8.77	Male	No	Sun	Dinner	2
34	17.78	Male	No	Sat	Dinner	2
107	25.21	Male	Yes	Sat	Dinner	2
...	...	...	...	...	...	...
24	19.82	Male	No	Sat	Dinner	2
16	10.33	Female	No	Sun	Dinner	3
131	20.27	Female	No	Thur	Lunch	2
70	12.02	Male	No	Sat	Dinner	2
112	38.07	Male	No	Sun	Dinner	3

194 rows × 6 columns



Create Pipeline Object, Train and Predict

Create Two Pipeline Objects to handle Categorical and Numeric Data

In [45]:

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder

# Preprocess categorical columns ("sex", "smoker", "day", "time")
categorical_transformer = Pipeline(steps=[
                                       ('imputer', SimpleImputer(missing_values=np.nan, strategy='
                                       ('onehot', OneHotEncoder(drop='first', handle_unknown='igno
                                    ])

# Preprocess numeric columns ("total_bill", "size")
numeric_transformer = Pipeline(steps=[
                                        ('imputer', SimpleImputer(missing_values=np.nan, strategy=
                                    ])
```

Combine above two Pipelines using ColumnTransformer

In [46]:

```
from sklearn.compose import ColumnTransformer

preprocessor = ColumnTransformer(
                                transformers=[
                                            ('categorical', categorical_transformer, [1,2,3,4]),
                                            ('numerical', numeric_transformer, [0,5])
                                            ]
                                  )
```

Create the Final Pipeline Object

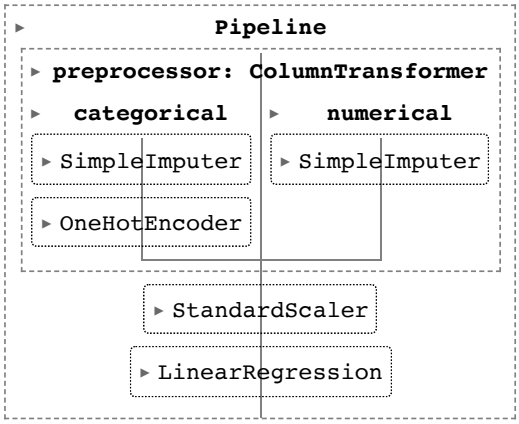
In [47]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression

model = Pipeline(steps=[
                         ('preprocessor', preprocessor),
                         ('scaling', StandardScaler()),
                         ('regressor', LinearRegression())
                        ])

model
```

Out[47]:



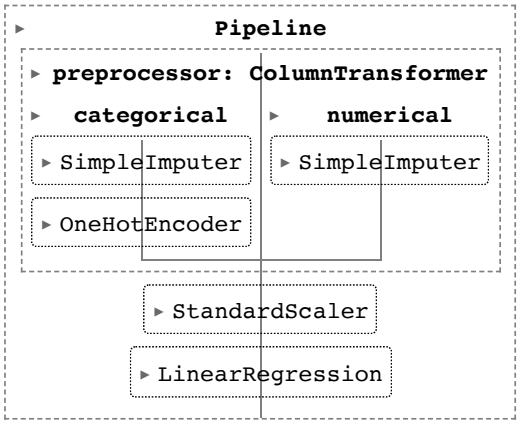
Train and Evaluate the Pipeline Object

In [48]:

```
# Note: when fit() is called on a Pipeline object, internally
# it will call the fit_transform() method on the transformers and fit() on the estimator

model.fit(X_train, y_train)
```

Out[48]:



In [49]:

```
# Note: when score() or predict() is called on a Pipeline object, internally
# only the transform() method is called on the transformers and score() / predict() on the estima

model.score(X_test, y_test)
```

Out[49]:

```
0.3919649062043292
```

Predict on new input

In [50]:

```
total_bill = 100
sex = 'Male'
smoker = 'Yes'
day = 'Sun'
time = 'Dinner'
size = 4
test_input = np.array([[total_bill, sex, smoker, day, time, size]], dtype='object')
test_input
```

Out[50]:

```
array([[100, 'Male', 'Yes', 'Sun', 'Dinner', 4]], dtype=object)
```

In [51]:

```
test_input.shape
```

Out[51]:

```
(1, 6)
```

In [52]:

```
# This will not work if you give string names of columns while creating the column transformer
model.predict(test_input)
```

Out[52]:

```
array([10.34508622])
```

Cross Validation using Pipeline

Okay, knowing all this, let's cross-validate our model pipeline using [cross_val_score() (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html)](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html)

In [53]:

```
from sklearn.model_selection import cross_val_score
cross_val_score(model,X_train, y_train, cv=5, scoring='r2')
```

Out[53]:

```
array([0.33120287, 0.3581081 , 0.57774671, 0.32420228, 0.47500525])
```

In [54]:

```
cross_val_score(model,X_train, y_train, cv=5, scoring='r2').mean()
```

Out[54]:

```
0.4132530407042142
```

Putting all the Code in one Cell

In [55]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings('ignore')

# Read the data
df = pd.read_csv('datasets/tips-missingdata.csv')
# Drop the rows with no labels
df.dropna(axis=0, how='any', subset=['tip'], inplace=True)
# Split data into X (data) & y (labels) and do a train-test split
X = df.drop('tip', axis=1)
y = df['tip']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=54)
X_train


# Preprocess categorical columns ("sex", "smoker", "day", "time")
categorical_transformer = Pipeline(steps=[
                                       ('imputer', SimpleImputer(missing_values=np.nan, strategy='
                                       ('onehot', OneHotEncoder(drop='first', handle_unknown='igno
                                    ])
# Preprocess numeric columns ("total_bill", "size")
numeric_transformer = Pipeline(steps=[
                                        ('imputer', SimpleImputer(missing_values=np.nan, strategy=
                                    ])
# Create a column transformer which combines all of the other transformers into one step
preprocessor = ColumnTransformer(
                                transformers=[
                                            ('categorical', categorical_transformer, [1,2,3,4]),
                                            ('numerical', numeric_transformer, [0,5])
                                            ]
                                 )
# Create the final pipeline
model = Pipeline(steps=[('preprocessor', preprocessor),
                        ('scaling', StandardScaler()),
                        ('regressor', LinearRegression())])

# Fit the model on the training data
model.fit(X_train, y_train)

print("R2 Score:", cross_val_score(model,X_train, y_train, cv=5, scoring='r2').mean())
test_input = np.array([[100, 'Male', 'Yes', 'Sun', 'Dinner', 4]], dtype='object')
print("Predicted tip amount: ", model.predict(test_input))
```

```
R2 Score: 0.4132530407042142
Predicted tip amount:  [10.34508622]
```

Task To Do

I'd suggest the following reading resources and next steps before doing the task.

- **Reading:** Scikit-Learn Pipeline() documentation (https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html).
- **Reading:** Imputing missing values before building an estimator (https://scikit-learn.org/stable/auto_examples/impute/plot_missing_values.html) (compares different methods of imputing values).
- **Practice:** Try tuning model hyperparameters with a `Pipeline()` and `GridSearchCV()` (https://scikit-learn.org/stable/modules/grid_search.html#composite-estimators-and-parameter-spaces).

In [56]:

```
import pandas as pd
import numpy as np
df = pd.read_csv('datasets/big_mart_sales.csv')
df
```

Out[56]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Estab
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8092	OUT049	
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692	OUT018	
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6180	OUT049	
3	FDX07	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	
4	NCD19	8.930	Low Fat	0.000000	Household	53.8614	OUT013	
...	...	...	...	...	...	...	...	
8518	FDF22	6.865	Low Fat	0.056783	Snack Foods	214.5218	OUT013	
8519	FDS36	8.380	Regular	0.046982	Baking Goods	108.1570	OUT045	
8520	NCJ29	10.600	Low Fat	0.035186	Health and Hygiene	85.1224	OUT035	
8521	FDN46	7.210	Regular	0.145221	Snack Foods	103.1332	OUT018	
8522	DRG01	14.800	Low Fat	0.044878	Soft Drinks	75.4670	OUT046	

8523 rows × 12 columns

Sales Prediction for Big Mart Outlets

- The data scientists at BigMart have collected 2013 sales data for 1559 products across 10 stores in different cities. Also, certain attributes of each product and store have been defined. The aim is to build a predictive model and predict the sales of each product at a particular outlet.
- Using this model, BigMart will try to understand the properties of products and outlets which play a key role in increasing sales.
- Please note that the data may have missing values as some stores might not report all the data due to technical glitches. Hence, it will be required to treat them accordingly.

Variable	Description
Item_Identifier	Unique product ID
Item_Weight	Weight of product
Item_Fat_Content	Whether the product is low fat or not
Item_Visibility	The % of total display area of all products in a store allocated to the particular product
Item_Type	The category to which the product belongs
Item_MRP	Maximum Retail Price (list price) of the product
Outlet_Identifier	Unique store ID
Outlet_Establishment_Year	The year in which store was established
Outlet_Size	The size of the store in terms of ground area covered
Outlet_Location_Type	The type of city in which the store is located
Outlet_Type	Whether the outlet is just a grocery store or some sort of supermarket
Item_Outlet_Sales	Sales of the product in the particular store. This is the outcome variable to be predicted.

In [ ]:

```
1
```