DS 620 Machine Learning and Deep Learning
HOS01Ab: Data Preparation

04/06/2021 Developed by Minh Nguyen
04/06/2021 Reviewed by Shanshan Yu
03/31/2022 Reviewed by Amanda Vaughan
10/03/2022 Reviewed by Alekhya Malla
School of Technology & Computing (STC) @ City University of Seattle (CityU)

Before You Start

- The directory path shown in screenshots may be different from yours.
- The document uses Google Colaboratory as the default compiler. If you want to run the code on a local machine, you need to configure the environment on your own.
- Some steps are not explained in the tutorial**.** If you are not sure what to do:
    1. Consult the resources listed below.
    2. If you cannot solve the problem after a few tries, ask a TA for help.

Learning Outcomes

- Understand the overview of a machine learning project
- Understand how to handle missing data
- Understand how to encode categorical features
- Understand how to scale numerical features
- Understand column transformer

Resources

- BLASTCHAR. (2018, February 23). *Telco Customer Churn* [Dataset]. Kaggle. https://www.kaggle.com/datasets/blastchar/telco-customer-churn
- Brownlee, J. (2020a, August 17). *Ordinal and one-hot encodings for categorical data*. Machine Learning Mastery. https://machinelearningmastery.com/one-hot-encoding-for-categorical-data/
- Brownlee, J. (2020b, August 25). *How to use data scaling improve deep learning model stability and performance*. Machine Learning Mastery. https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/
- Cook, A. (2021, November 9). *Categorical Variables*. Kaggle. https://www.kaggle.com/code/alexisbcook/categorical-variables
- Géron, A. (2019). *Hands-on machine learning with scikit-learn, keras, and tensorflow* (2nd ed.). O'Reilly Media.
- Massaron, L., & Boschetti, A. (2016). *Regression analysis with python*. Packt Publishing.
- *scikit-learn: Machine learning in python.* (n.d.). Scikit-Learn. https://scikit-learn.org/stable/index.html

Introduction

In real world scenarios, data takes a variety of forms which are usually messy, unstructured, and incompatible for most ML models. As a data practitioner, you will almost never find your data already prepared in the right form to be immediately analyzed for your purposes. In fact, you will often find yourself spending most of your time preparing data in a Data Science project. To produce successful machine learning models, data needs to be high quality. An acronym that all machine learning practitioners need to remember is "GIGO" which is short for "garbage in, garbage out."

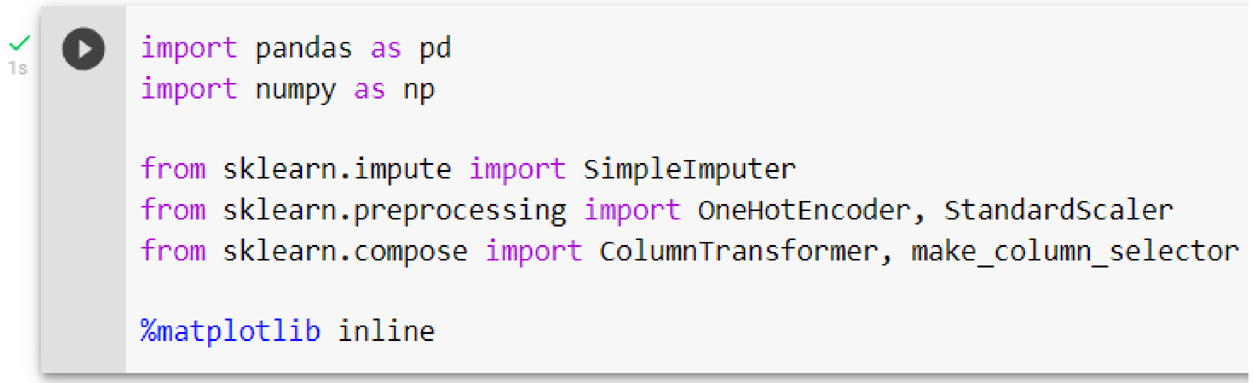
Overview of a Machine Learning project

1. Obtain data.
2. Explore and visualize the data to gain insights.
3. Prepare the data for Machine Learning algorithms.
4. Select a model and train it.
5. Fine-tune your model.
6. Present your solution.
7. Launch, monitor, and maintain your system.

These are the main steps a data science team needs to go through in real life when approaching a new data set. In this HOS, we will focus our attention on step 3. We will only briefly go over step 2 as it is not the main topic of this HOS.

Import Libraries

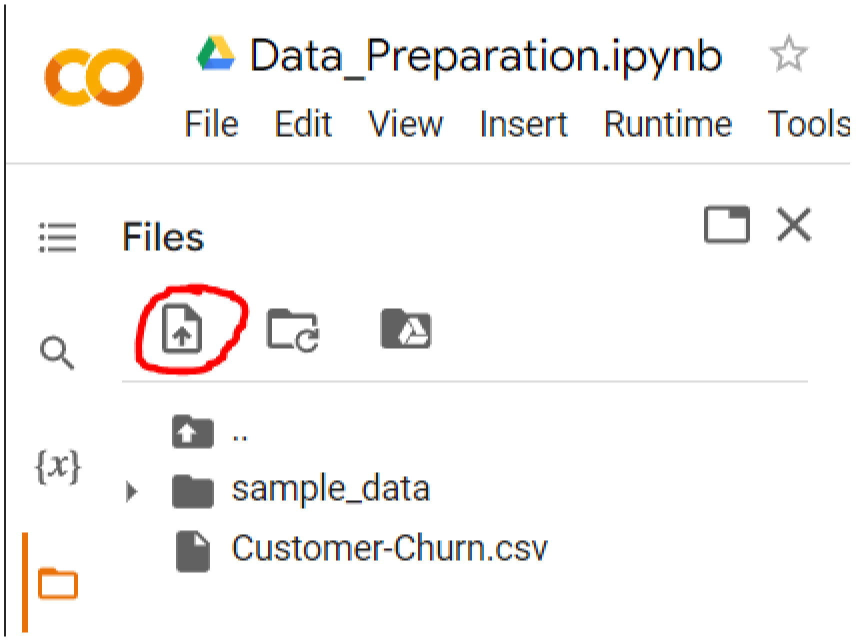
1. Go to https://colab.research.google.com. Create a new notebook and name it "Data_Preparation.ipynb".
2. Now, we need to import the necessary libraries. Type the following in your code cell then click the play button or Ctrl + Enter to run the cell.



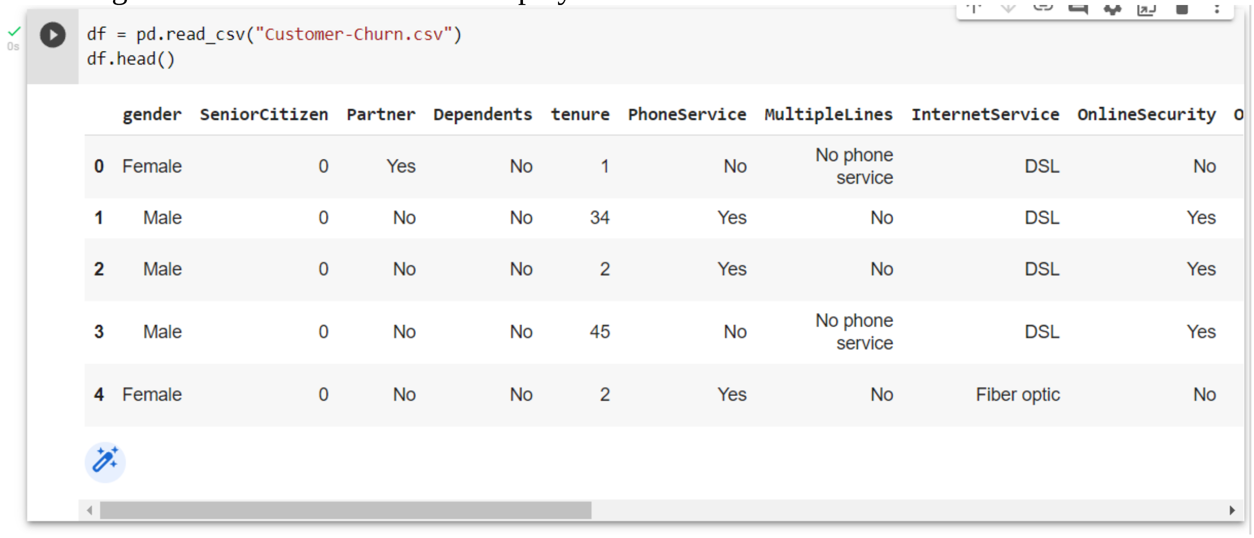
```
import pandas as pd
import numpy as np

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer, make_column_selector

%matplotlib inline
```

Obtain data

3. Upload the dataset "Customer-Churn.csv" to your Colab notebook.



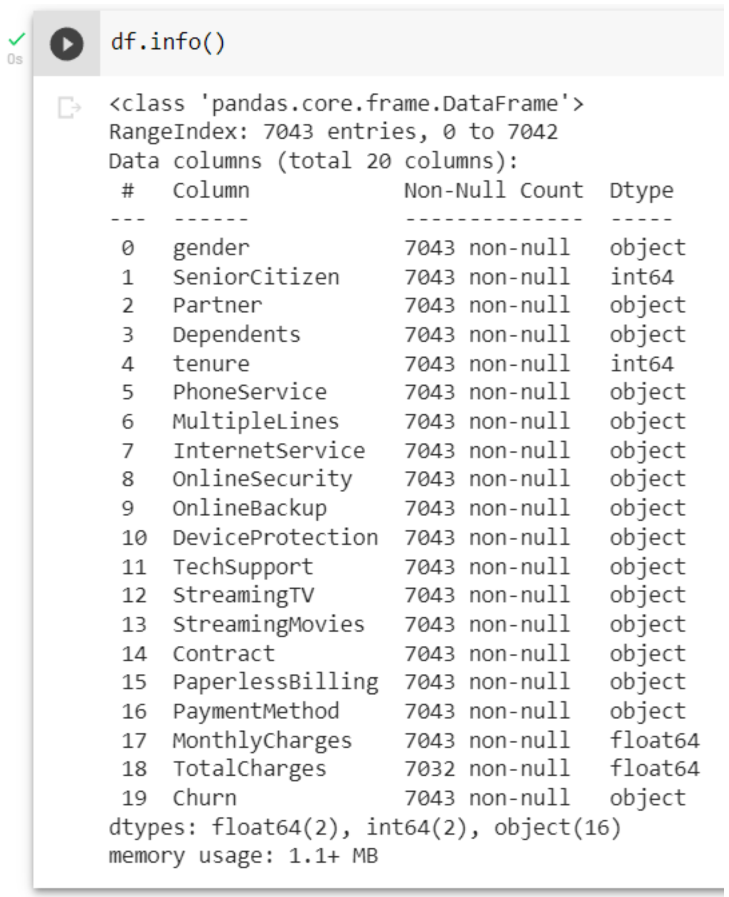
4. Now, let's import the data. Click the + Code button to add a new code cell. Type the following code in the cell then click the play button or Ctrl + Enter to run the cell.



```
df = pd.read_csv("Customer-Churn.csv")
df.head()
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	O
0	Female	0	Yes	No	1	No	No phone service	DSL	No	
1	Male	0	No	No	34	Yes	No	DSL	Yes	
2	Male	0	No	No	2	Yes	No	DSL	Yes	
3	Male	0	No	No	45	No	No phone service	DSL	Yes	
4	Female	0	No	No	2	Yes	No	Fiber optic	No	

5. Next, we'll get information regarding the columns and shape of the data. Click the + Code button to add a new code cell. Type the following code in the cell then click the play button or Ctrl + Enter to run the cell.



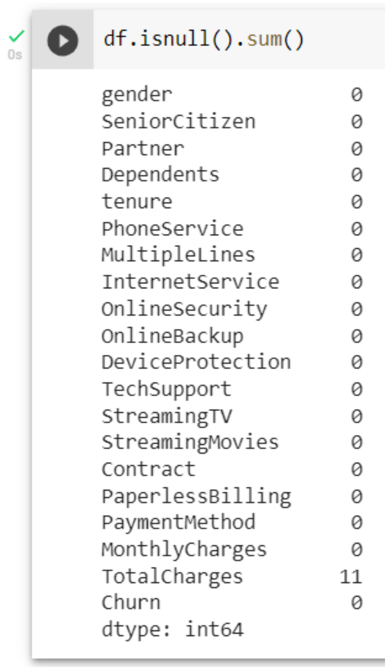
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   gender            7043 non-null   object 
 1   SeniorCitizen     7043 non-null   int64  
 2   Partner           7043 non-null   object 
 3   Dependents        7043 non-null   object 
 4   tenure            7043 non-null   int64  
 5   PhoneService      7043 non-null   object 
 6   MultipleLines     7043 non-null   object 
 7   InternetService   7043 non-null   object 
 8   OnlineSecurity    7043 non-null   object 
 9   OnlineBackup      7043 non-null   object 
 10  DeviceProtection  7043 non-null   object 
 11  TechSupport       7043 non-null   object 
 12  StreamingTV       7043 non-null   object 
 13  StreamingMovies   7043 non-null   object 
 14  Contract          7043 non-null   object 
 15  PaperlessBilling  7043 non-null   object 
 16  PaymentMethod     7043 non-null   object 
 17  MonthlyCharges    7043 non-null   float64
 18  TotalCharges      7032 non-null   float64
 19  Churn             7043 non-null   object 
dtypes: float64(2), int64(2), object(16)
memory usage: 1.1+ MB
```

From this we can see that most columns are categorical.

Handle missing data
Missing data occurs most of the time in real life data and is usually a result of bias in its recording and treatment. Unfortunately, most machine learning models, especially linear models, cannot deal with this problem. Therefore, it is the machine learning practitioner’s job to address this issue either by dropping missing data or imputing it.

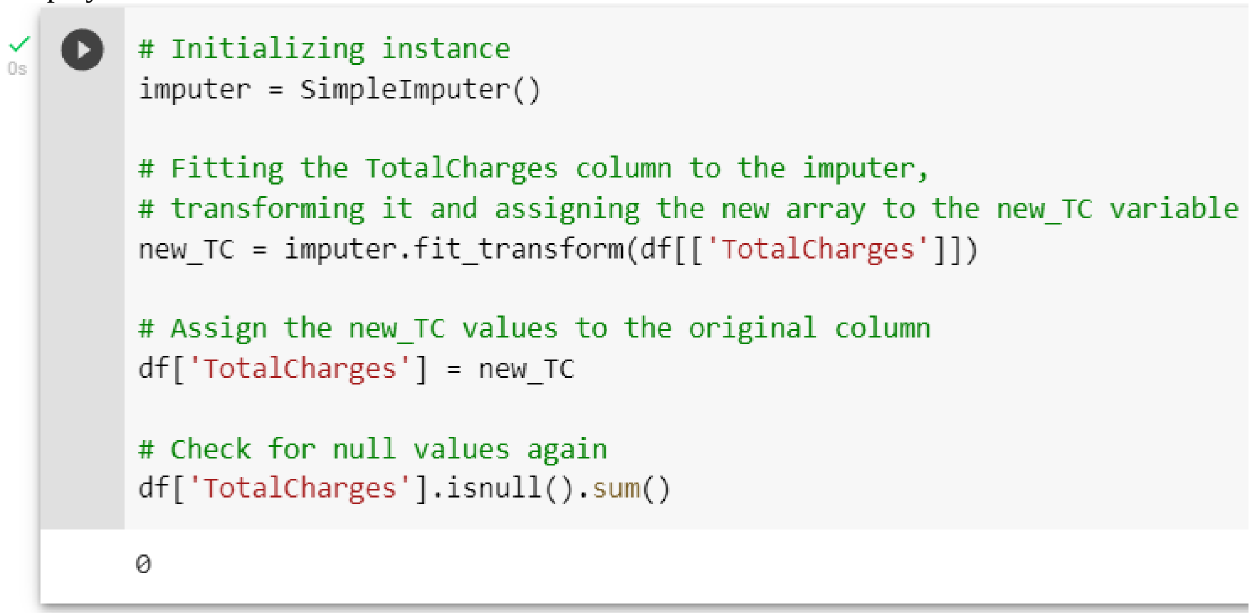
6. Let’s check for missing data. Click the + Code button to add a new code cell. Type the following code in the cell then click the play button or Ctrl + Enter to run the cell.



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

gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges       11
Churn               0
dtype: int64
```

Here we can see that there is only one column with missing values. Usually, when the number of missing values is not significant, we will drop them as it won't affect model training. However, when dropping rows will heavily reduce the amount of data, a good strategy is to impute missing values.

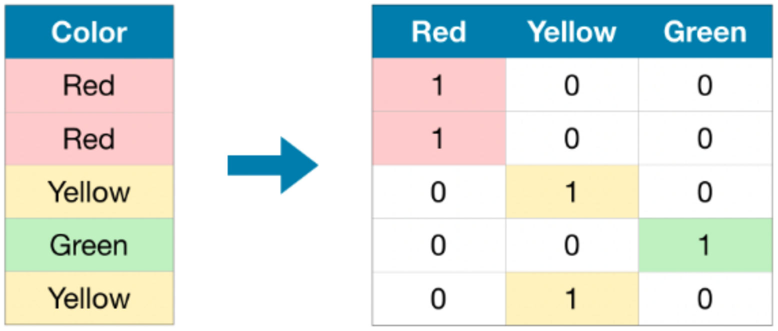
7. Click the + Code button to add a new code cell. Type the following code in the cell then click the play button or Ctrl + Enter to run the cell.



```
# Initializing instance
imputer = SimpleImputer()

# Fitting the TotalCharges column to the imputer,
# transforming it and assigning the new array to the new_TC variable
new_TC = imputer.fit_transform(df[['TotalCharges']])

# Assign the new_TC values to the original column
df['TotalCharges'] = new_TC

# Check for null values again
df['TotalCharges'].isnull().sum()

0
```

Encoding categorical variables
Most machine learning algorithms only accept input data in a numerical matrix format. This means that they won't accept any columns with text data. Therefore, text data needs to be

converted into numerical format through a process called One Hot Encoding. Going into details of this process is beyond the scope of this exercise. At the fundamental level, One Hot Encoding turns a column into a matrix of yes (1) - no (0) values.

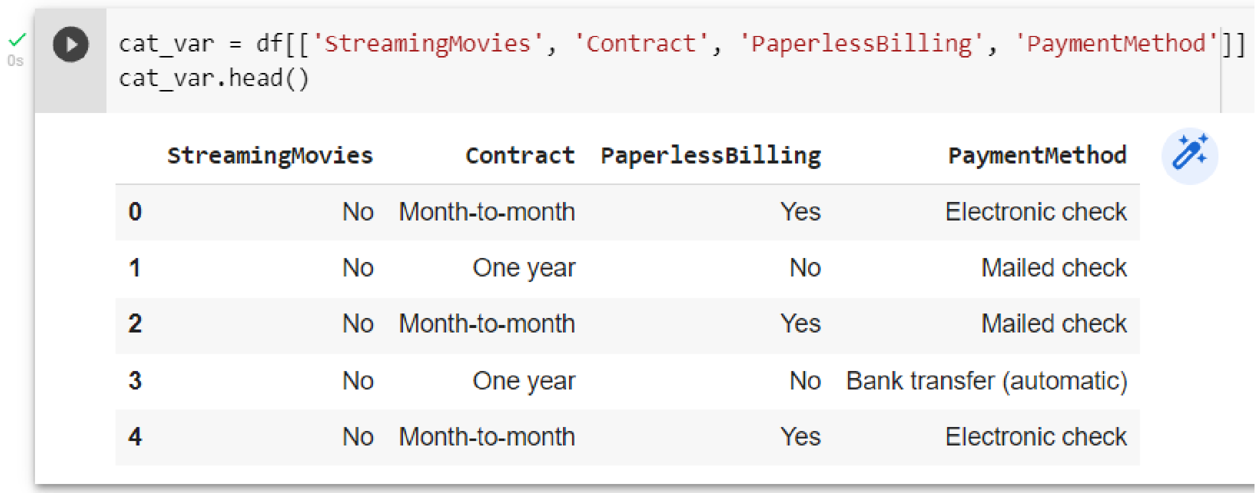


Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

Figure 1: One Hot Encoding Example (Cook, 2021)

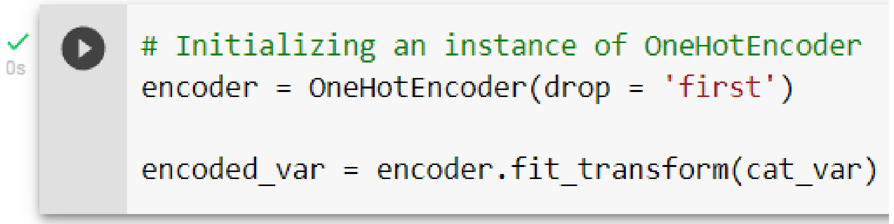
8. Let's prepare the categorical data. Click the + Code button to add a new code cell. Type the following code in the cell then click the play button or Ctrl + Enter to run the cell.



```
cat_var = df[['StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod']]
cat_var.head()
```

	StreamingMovies	Contract	PaperlessBilling	PaymentMethod
0	No	Month-to-month	Yes	Electronic check
1	No	One year	No	Mailed check
2	No	Month-to-month	Yes	Mailed check
3	No	One year	No	Bank transfer (automatic)
4	No	Month-to-month	Yes	Electronic check

9. Now we can encode the data. Click the + Code button to add a new code cell. Type the following code in the cell then click the play button or Ctrl + Enter to run the cell.

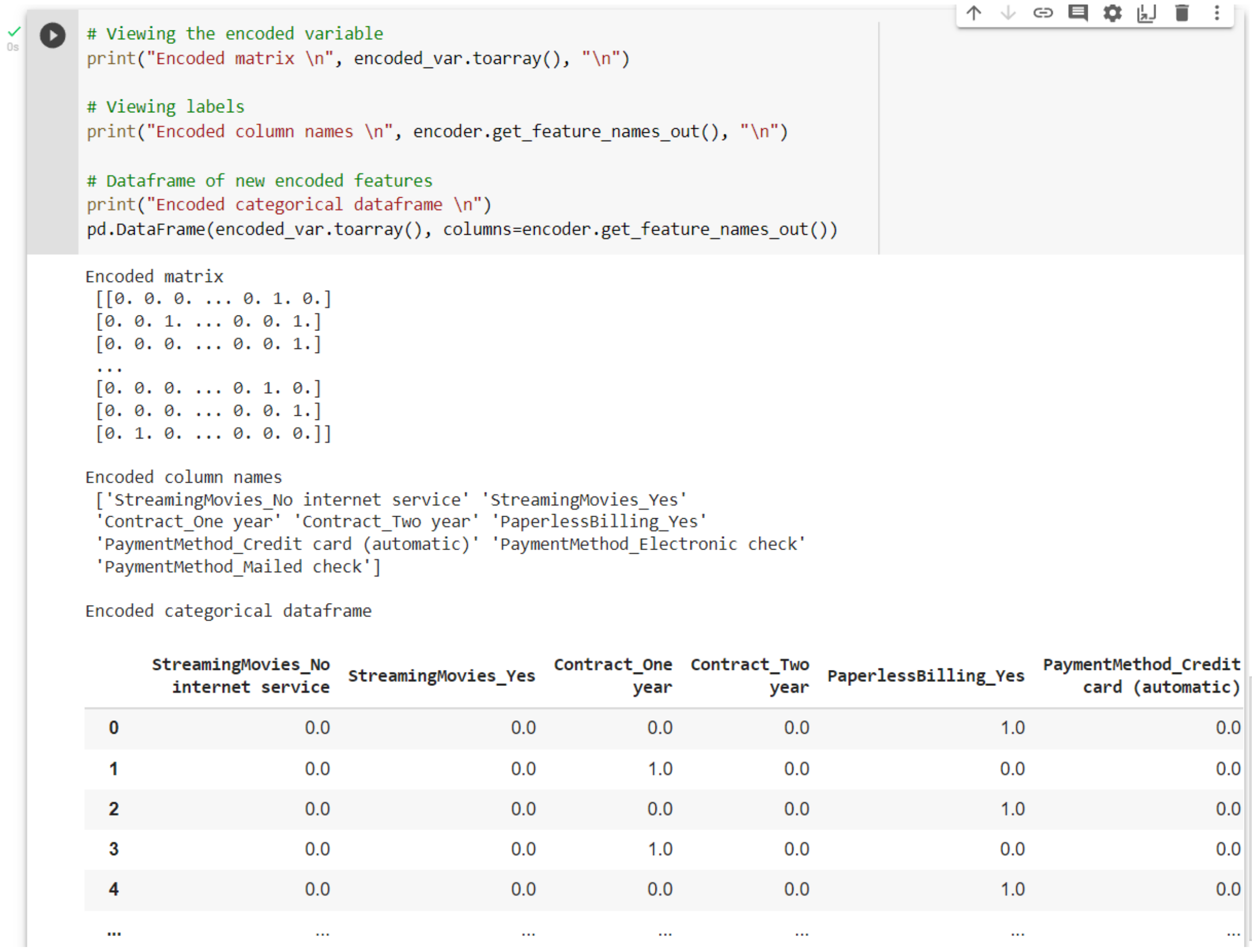


```
# Initializing an instance of OneHotEncoder
encoder = OneHotEncoder(drop = 'first')

encoded_var = encoder.fit_transform(cat_var)
```

Specifying the parameter drop as 'first' will automatically drop the first column of the encoded matrix. This prevents multi collinearity for Linear Model training.

You will often see this "fit_transform" method. It is a command to tell the instance variable to store information about the data and perform the transformation on the data.

10. We can also view the encoded data information. Click the + Code button to add a new code cell. Type the following code in the cell then click the play button or Ctrl + Enter to run the cell.



```
# Viewing the encoded variable
print("Encoded matrix \n", encoded_var.toarray(), "\n")

# Viewing labels
print("Encoded column names \n", encoder.get_feature_names_out(), "\n")

# Dataframe of new encoded features
print("Encoded categorical dataframe \n")
pd.DataFrame(encoded_var.toarray(), columns=encoder.get_feature_names_out())
```

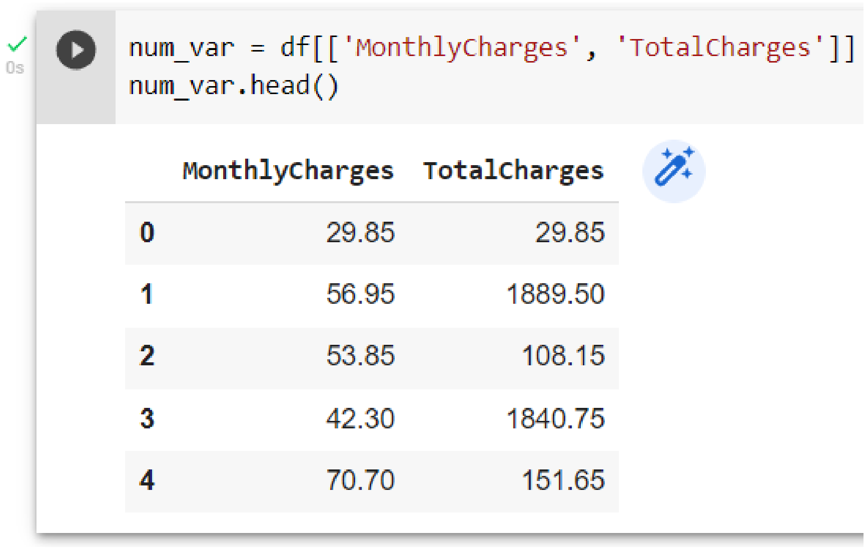
```
Encoded matrix
 [[0. 0. 0. ... 0. 1. 0.]
 [0. 0. 1. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]
 ...
 [0. 0. 0. ... 0. 1. 0.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 1. 0. ... 0. 0. 0.]]

Encoded column names
 ['StreamingMovies_No internet service' 'StreamingMovies_Yes'
 'Contract_One year' 'Contract_Two year' 'PaperlessBilling_Yes'
 'PaymentMethod_Credit card (automatic)' 'PaymentMethod_Electronic check'
 'PaymentMethod_Mailed check']

Encoded categorical dataframe
```

	StreamingMovies_No internet service	StreamingMovies_Yes	Contract_One year	Contract_Two year	PaperlessBilling_Yes	PaymentMethod_Credit card (automatic)
0	0.0	0.0	0.0	0.0	1.0	0.0
1	0.0	0.0	1.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	1.0	0.0
3	0.0	0.0	1.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	1.0	0.0
...	...	...	...	...	...	...

Scaling numerical features

Re-scaling is essential when using gradient descent-based algorithms because it facilitates quicker convergence to a solution for finding coefficients. Rescaling also helps other optimization techniques such as regularization and stochastic learning, and easily detects outlying and anomalous values.

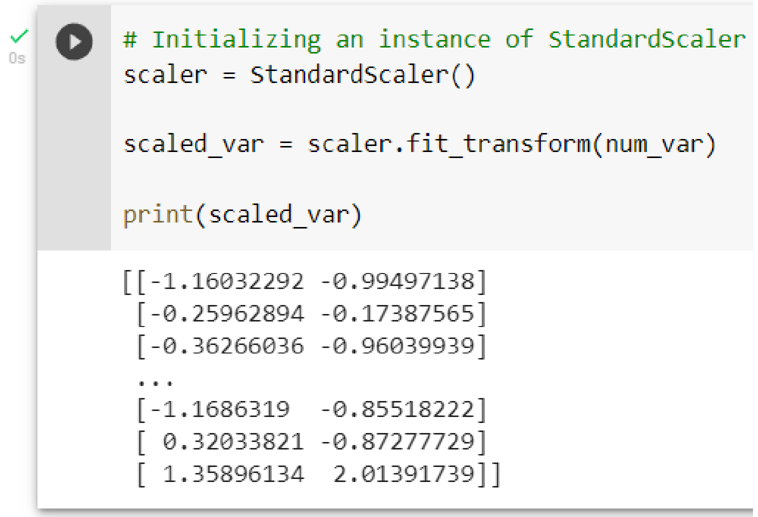
11. Now we'll prepare the numerical data. Click the + Code button to add a new code cell. Type the following code in the cell then click the play button or Ctrl + Enter to run the cell.



```
num_var = df[['MonthlyCharges', 'TotalCharges']]
num_var.head()
```

	MonthlyCharges	TotalCharges
0	29.85	29.85
1	56.95	1889.50
2	53.85	108.15
3	42.30	1840.75
4	70.70	151.65

12. Next, we’ll rescale the columns. Click the + Code button to add a new code cell. Type the following code in the cell then click the play button or Ctrl + Enter to run the cell.



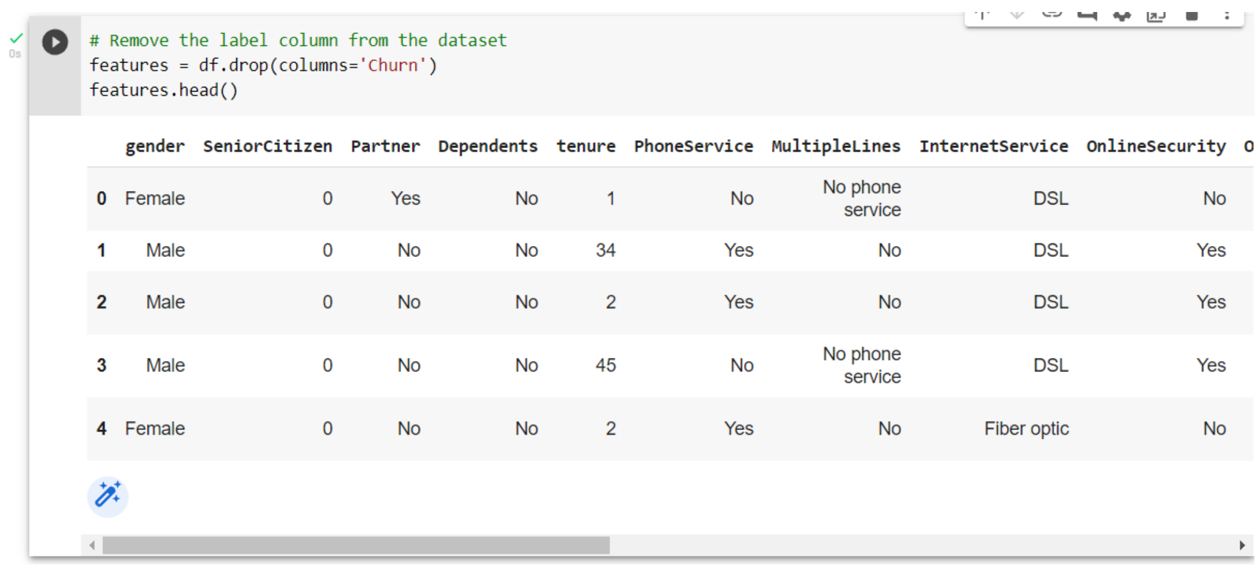
```
# Initializing an instance of StandardScaler
scaler = StandardScaler()

scaled_var = scaler.fit_transform(num_var)

print(scaled_var)
```

```
[[-1.16032292 -0.99497138]
 [-0.25962894 -0.17387565]
 [-0.36266036 -0.96039939]
 ...
 [-1.1686319  -0.85518222]
 [ 0.32033821 -0.87277729]
 [ 1.35896134  2.01391739]]
```

Column transformer

So far, we have been working with categorical data and numerical data separately. What if there was a way to do this in one step? The ColumnTransformer() class is your answer. This class creates a Transformer that applies the specified type of transformation on the specified column. Specifying a column can be based on either data type or column name.

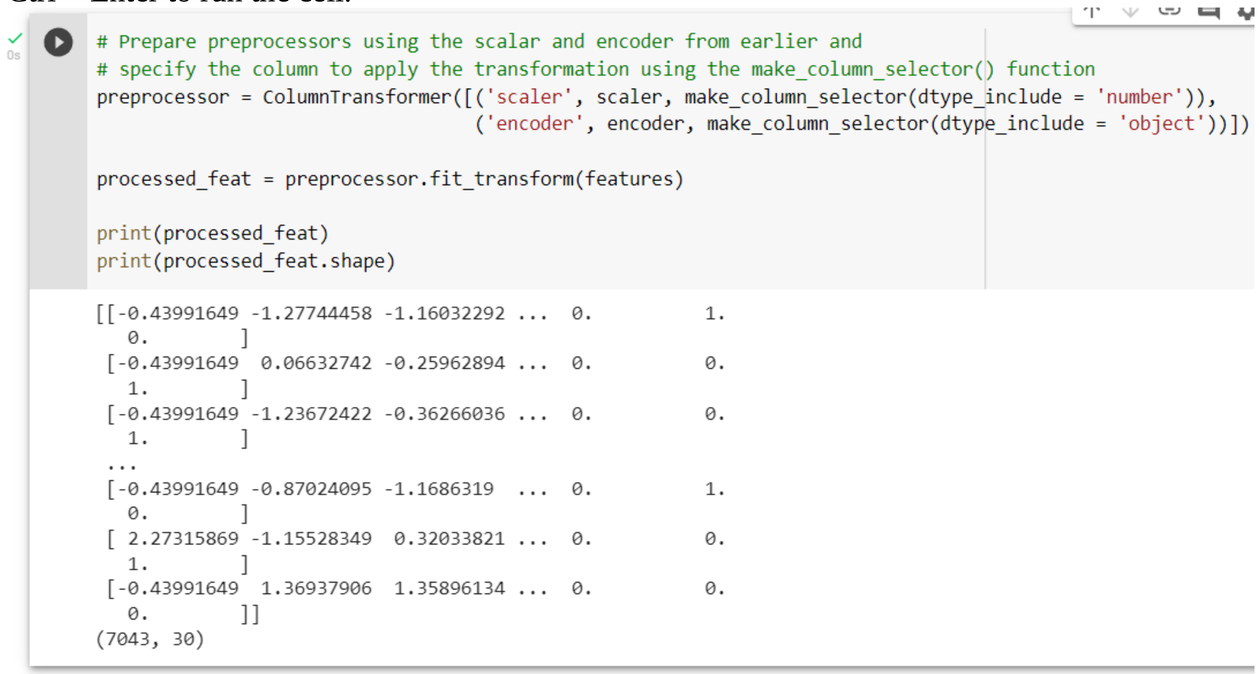
13. Let’s prepare the input matrix. Click the + Code button to add a new code cell. Type the following code in the cell then click the play button or Ctrl + Enter to run the cell.



```
# Remove the label column from the dataset
features = df.drop(columns='Churn')
features.head()
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	O
0	Female	0	Yes	No	1	No	No phone service	DSL	No	
1	Male	0	No	No	34	Yes	No	DSL	Yes	
2	Male	0	No	No	2	Yes	No	DSL	Yes	
3	Male	0	No	No	45	No	No phone service	DSL	Yes	
4	Female	0	No	No	2	Yes	No	Fiber optic	No	

14. Now we’ll prepare the preprocessor object and apply it to the data. Click the + Code button to add a new code cell. Type the following code in the cell then click the play button or Ctrl + Enter to run the cell.

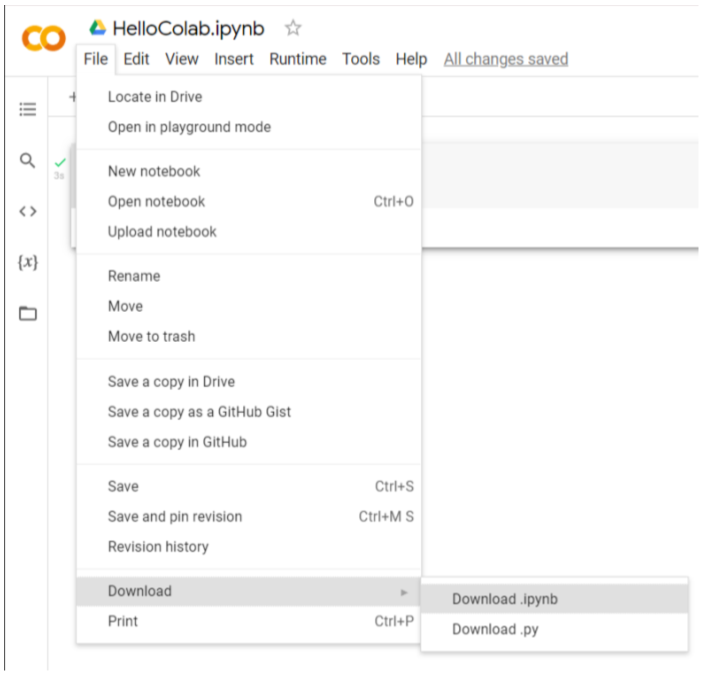


```
# Prepare preprocessors using the scalar and encoder from earlier and
# specify the column to apply the transformation using the make_column_selector() function
preprocessor = ColumnTransformer([('scaler', scaler, make_column_selector(dtype_include = 'number')),
                                  ('encoder', encoder, make_column_selector(dtype_include = 'object'))])

processed_feat = preprocessor.fit_transform(features)

print(processed_feat)
print(processed_feat.shape)
```

```
[[-0.43991649 -1.27744458 -1.16032292 ...  0.          1.
   0.        ]
 [-0.43991649  0.06632742 -0.25962894 ...  0.          0.
   1.        ]
 [-0.43991649 -1.23672422 -0.36266036 ...  0.          0.
   1.        ]
 ...
 [-0.43991649 -0.87024095 -1.1686319  ...  0.          1.
   0.        ]
 [ 2.27315869 -1.15528349  0.32033821 ...  0.          0.
   1.        ]
 [-0.43991649  1.36937906  1.35896134 ...  0.          0.
   0.        ]]
(7043, 30)
```

Push your work to GitHub
Click File → Download → Download .ipynb.

The file will be downloaded to your Download folder (or another folder according to your browser settings).



Move the downloaded ipynb file to your Module01 folder.

Follow the instructions on the CityU STC TA Center Github.io [Submit your work page](#).