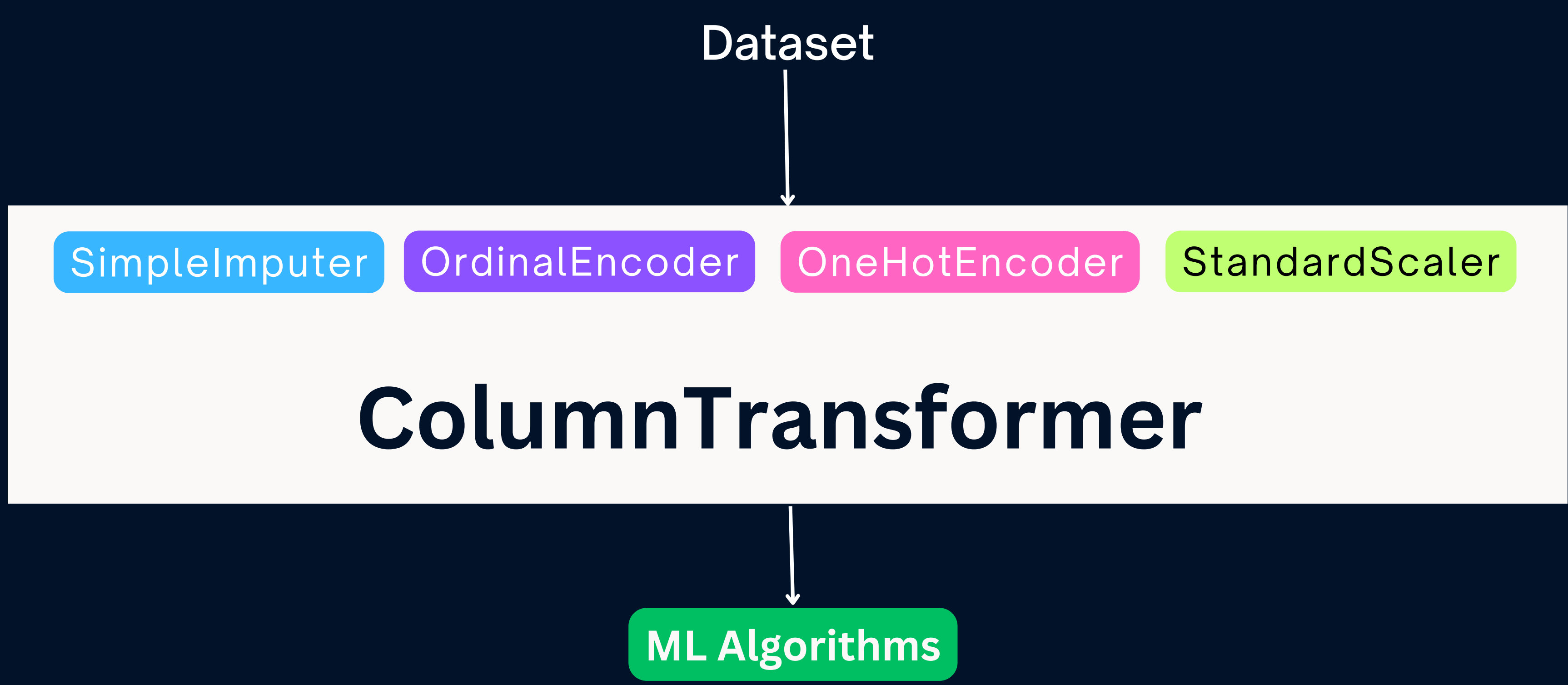
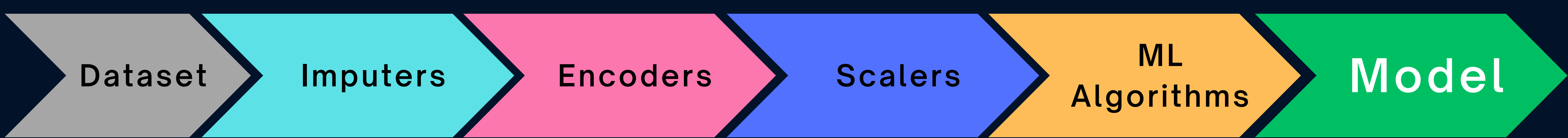
ColumnTransformer and Pipeline in Machine Learning





smitesh-tamboli

NEXT

ColumnTransformer

- ColumnTransformer is a class that allows to apply different transformations to different columns or variables in a dataset.
- It helps to streamline the preprocessing pipeline by applying specific transformations to specific subsets of features.
- CoulmnTransformer helps when a dataset contains different types of data such as discrete, continuous, nominal, and ordinal which require to preprocess before applying any machine learning algorithms.

Why Need?

Consider the below dataset where we need to perform different preprocessing techniques before sending data to machine learning algorithms

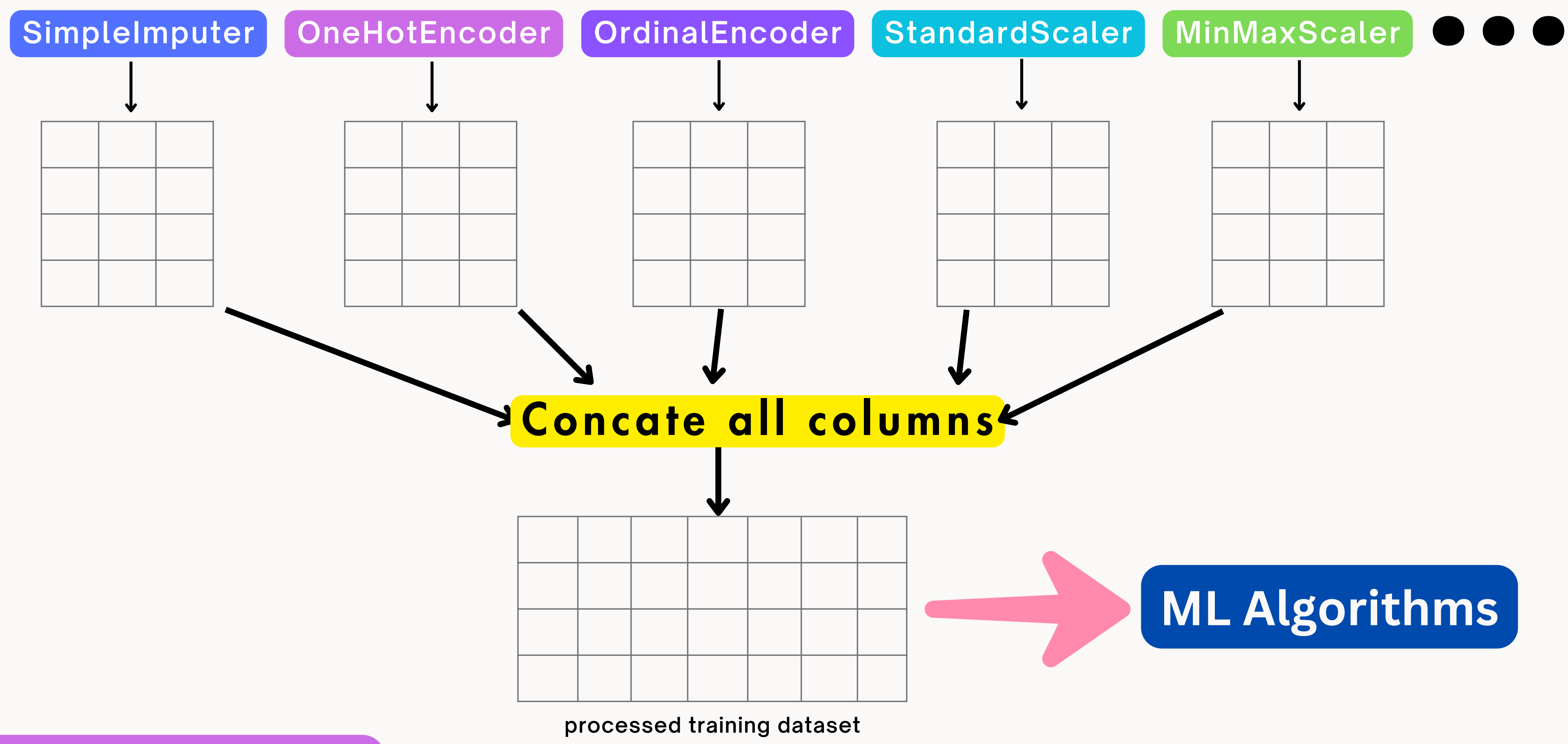
OneHotEncoder StandardScaler OneHotEncoder StandardScaler

	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Seats	Price
0	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	5.0	1.75
1	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	5.0	12.50
2	Chennai	2011	46000	Petrol	Manual	First	18.2 kmpl	5.0	4.50
3	Chennai	2012	87000	Diesel	Manual	First	20.77 kmpl	7.0	6.00
4	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmpl	5.0	17.74

OrdinalEncoder OrdinalEncoder

- Each preprocessing technique generates one or multiple columns based on the nature of the technique. For example, StandardScaler, and OrdinalEncoder generate single columns while OneHotEncoder generates multiple columns.
- If we apply all the preprocessing methods individually, we have to concate all the columns in a single dataset which is sometimes a tedious task for a large number of columns.

Without ColumnTransfer



OneHotEncoder

```python
# Perform OneHotEncoder on Location, FuelType and Transmission
# Location: 11 columns
# Fuel_Type: 4 columns
# Transmission: 2 columns
# Total 17 column must be generated through OneHotEncoder

ohe = OneHotEncoder(sparse_output=False)
X_train_ohe =
ohe.fit_transform(X_train[['Location','Fuel_Type','Transmission']])

X_test_ohe =
ohe.transform(X_test[['Location','Fuel_Type','Transmission']])
X_train_ohe.shape # (4779, 17)
```

Generates 17 columns

OrdinalEncoder

```
# Ordinal Columns
# I am considering Year as an ordinal column as price may vary based on Year
# Year: lower (2006) → higher (2019)
# Owner_Type: lower (Fourth & Above) → higher (First)
# Seats: 4, 5, 7, 8

ordEnc = OrdinalEncoder(categories=[
    ["2010","2011","2012","2013","2014","2015","2016","2017","2018","2019"],
    ["Fourth & Above","Third","Second","First"],
    [4.0, 5.0,7.0,8.0]
    ])

X_train_ordencode =
ordEnc.fit_transform(X_train[["Year","Owner_Type","Seats"]])

X_test_ordencode = ordEnc.transform(X_test[["Year","Owner_Type","Seats"]])
```

Generates
3 columns

StandardScaler

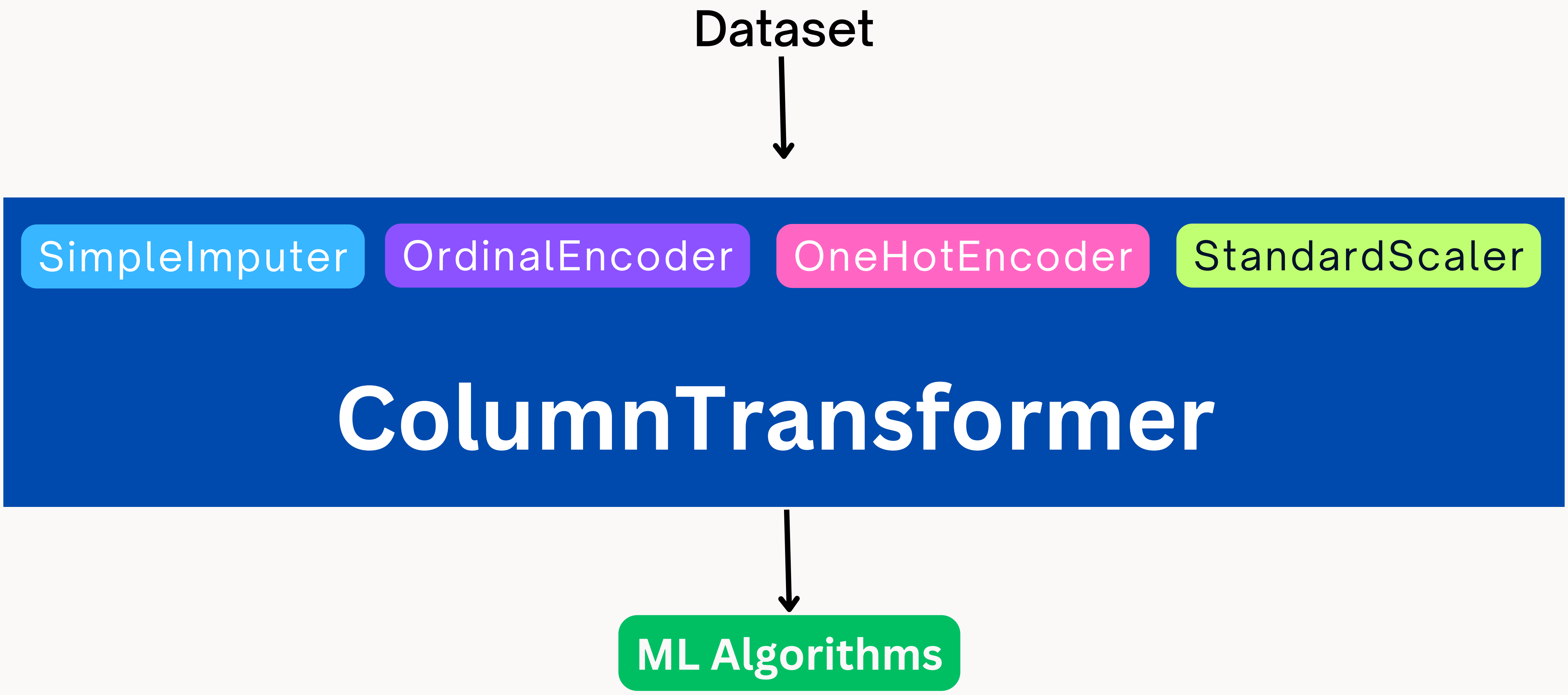
```
# Apply Standardization to Kilometers_Driven, Mileage

stdScalar = StandardScaler()

X_train_scaled =
stdScalar.fit_transform(X_train[['Kilometers_Driven','Mileage']])

X_test_scaled =
stdScalar.transform(X_test[['Kilometers_Driven','Mileage']])
```

Generates
2 columns

Concate All Columns

```
# Concate all the columns and create single dataset which need to pass
to Algorithms

X_train_all = np.concatenate((X_train_scaled, X_train_ordencode,
X_train_ohe), axis=1)

X_test_all = np.concatenate((X_test_scaled, X_test_ordencode,
X_test_ohe), axis= 1)
```

With ColumnTransfer




```
encoding_trns = ColumnTransformer(
    [
        ("OneHot_Encoder",
          OneHotEncoder(sparse_output=False, handle_unknown='ignore'),
          [0, 3, 4 ]
        ),

        ("Ordinal_Encoder",
          OrdinalEncoder(
            categories=[
                ["2010","2011","2012","2013","2014","2015","2016","2017","2018","2019"],
                ["Fourth & Above","Third","Second","First"],
                [4.0, 5.0,7.0,8.0]
                    ]),
          [1,5,7] )
    ],
    remainder='passthrough'
)

# Apply Standardization to Kilometers_Driven, Mileage
stnd_scaler_trns = ColumnTransformer(
    [("std_scaler",
      StandardScaler(),
      [2, 6 ] ) ],
    remainder='passthrough'
)
```

ColumnTransfer

- ColumnTransformer allows to apply different transformations like OneHotEncoding, StandardScaler, MinMaxScaler, etc.. to different columns or variables in a dataset.
- It helps to streamline the preprocessing pipeline by applying specific transformations to specific subsets of features.
- CoulmnTransformer helps when a dataset contains different types of data such as discrete, continuous, nominal, and ordinal which require preprocessing before applying any machine learning algorithms.

```
encoding_trns = ColumnTransformer(
    [
        ( “name”, transformer, columns ),
        ( “name”, transformer, columns ),
        ...
    ],
    remainder='passthrough'
)
```

List of tuples

passthrough: do not perform any action on the remaining columns

drop: drop remaining columns

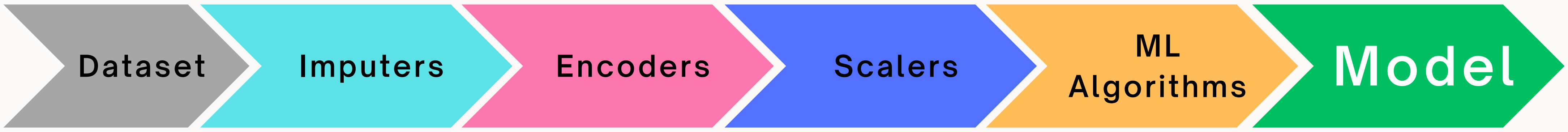
(“name”, transformer, columns)

column Index

```
encoding_trns = ColumnTransformer(
    [
        ("OneHot_Encoder",  OneHotEncoder(sparse_output=False, handle_unknown='ignore'),[0, 3, 4 ]  ),

        ("Ordinal_Encoder", OrdinalEncoder(
            categories=[
                ["2010","2011","2012","2013","2014","2015","2016","2017","2018","2019"],
                ["Fourth & Above","Third","Second","First"],
                [4.0, 5.0,7.0,8.0]
                    ]), [1,5,7] )
    ],
    remainder='passthrough'
)
```

Pipeline

- A pipeline is a sequence of data processing steps that are chained together.
- The purpose of the pipeline is to streamline the workflow by bundling together multiple preprocessing steps and model fitting into a single object.



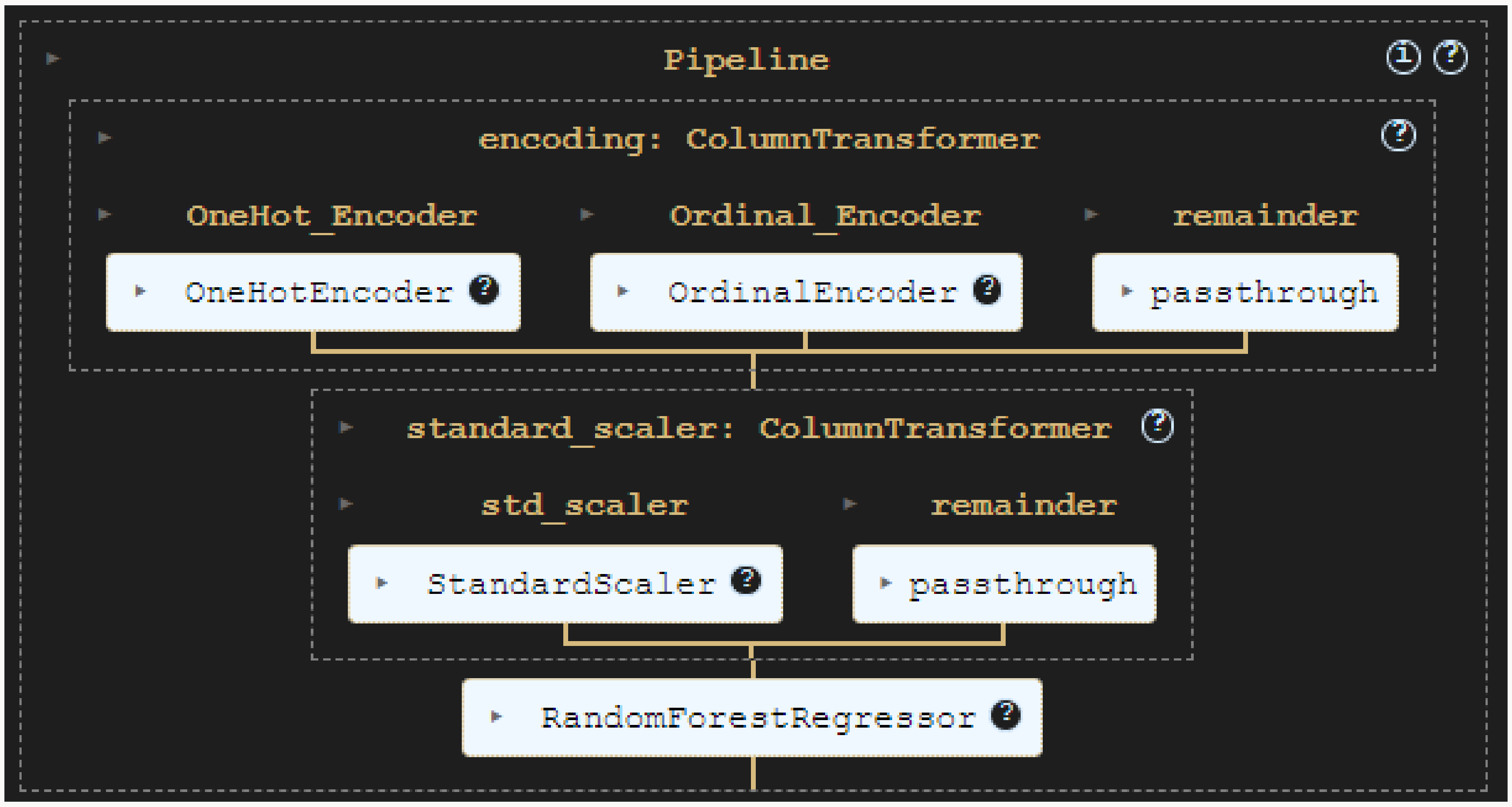

```
rf = RandomForestRegressor(n_estimators = 100)

pipe = Pipeline(
[
    ('encoding',encoding_trns),
    ('standard_scaler',stnd_scaler_trns ) ,
    ('random_forest',rf)
] )

pipe.fit(X_train, y_train)
```

where **encoding_trns and stnd_scaler_trns** are ***column transformers*** and **rf** is ***RandomForestRegressor***

Pipeline Code

```
encoding_trns = ColumnTransformer(
    [
        ( "OneHot_Encoder",
          OneHotEncoder(sparse_output=False, handle_unknown='ignore'),
          [0, 3, 4 ] ),

        ( "Ordinal_Encoder",
          OrdinalEncoder(
            categories=[
                ["2010","2011","2012","2013","2014","2015","2016","2017","2018","2019"],
                ["Fourth & Above","Third","Second","First"],
                [4.0, 5.0,7.0,8.0]
                    ]),
          [1,5,7] )
    ],
    remainder='passthrough'
)
# Apply Standardization to Kilometers_Driven, Mileage
stnd_scaler_trns = ColumnTransformer(
    [( "std_scaler", StandardScaler(), [2, 6 ]  )],
    remainder='passthrough'
)

rf = RandomForestRegressor(n_estimators = 100)

# Create Pipeline
pipe = Pipeline([
    ('encoding',encoding_trns),
    ('standard_scaler',stnd_scaler_trns ) ,
    ('random_forest',rf)
])

pipe.fit(X_train, y_train)
```

Pipeline Diagram



Benefits of Pipeline

- It bundles multiple preprocessing steps and models fitting into a single object which simplifies the code and makes it easy to manage
- Automate the preprocessing and modeling process.
- In production, we do not need to remember the order of preprocessing steps.
- Improve code readability
- Encapsulate all preprocessing steps and model fitting into a single object which makes it easy to reuse the same processing steps and model across

smitesh-tamboli