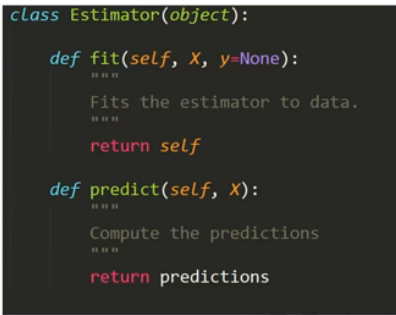
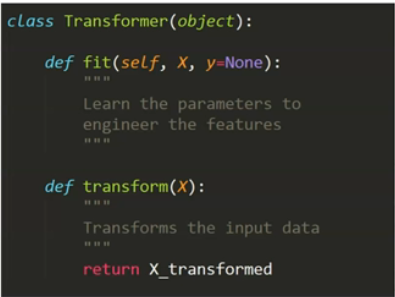
# Python Open-source for ML

1. Scikit-learn

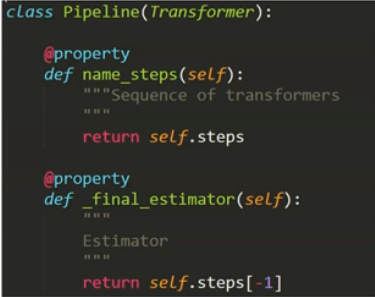
* Solid implementation of a wide range of ML algorithms and data transformations
* Clean, uniform and streamlined API
* Most algorithms follow the same functionality -> implementing new algorithms is super easy
* Transformers
* Estimators
* Pipeline
* Complete online documentation, with some theory and examples
* Well established in the community -> new packages follow Scikit-learn functionality to be quickly adopted by end users, e.g., Keras, MLXtend, category-encoders, Feature-engine
* Scikit-learn estimators



* A class with fit() and predict() methods
* Any ML algorithm like Lasso, Decision trees, SVMs, are coded as estimators within Scikit-learn
* Scikit-learn transformers
* Class that have fit() and transform() methods



* Scalers
* Encoders
* Imputers
* Discretizers
* Transformers
* Scikit-learn pipeline



* Class that allows to run transformers and estimators in sequence
* Most steps are transformers
* Last step can be an estimator

1. Open-source for feature engineering

* Scikit-learn
* Category encoders
* Feature-engine
* Featuretools
* Imbalanced learn

1. Open-source for feature selection

* Scikit-learn
* Feature-engine
* MLXtend

1. Open-source for model training

* Scikit-learn
* Keras
* MLXtend
* Py-earth
* Xgboost
* Lasagne
* Many others

# Open source for feature engineering

1. Scikit-learn transformers

* Missing data imputation
* SimpleImputer
* IterativeImputer
* Categorical variable encoding
* OneHotEncoder
* OrdinalEncoder
* Discretization
* KBinsDiscretizer
* Variable transformation
* PowerTransformer
* FunctionTransformer
* Variable combination
* Polynomial Features
* Scalers
* StandardScaler
* MinMaxScaler
* RobustScaler
* Etc.
* Text
* Word Count
* TFiDF

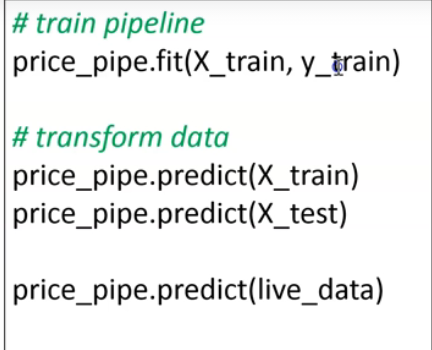
1. Feature Engine Transformers

* Imputation methods
* MeanMedianImputer
* RandomSampleImputer
* EndTailImputer
* AddMissingIndicator
* CategoricalImputer
* ArbitraryNumberImputer
* DropMissingData
* Encoding methods
* OneHotEncoder
* OrdinalEncoder
* CountFrequencyEncoder
* MeanEncoder
* WoEEncoder
* PRatioEncoder
* RareLabelEncoder
* DecisionTreeEncoder
* Discretization methods
* EqualFrequencyDiscretiser
* EqualWidthDiscretiser
* DecisionTreeDiscretiser
* ArbitraryDiscretiser
* Variable Transformation method
* LogTransformer
* ReciprocalTransformer
* PowerTransformer
* BoxCoxTransformer
* YeoJohnsonTransformer
* Outlier Handling methods
* Winsorizer
* ArbitraryOutlierCapper
* OutlierTrimmer
* Scikit-learn wrapper
* SklearnTransformerWrapper
* Variable Combinations
* MathematicalCombination
* CombineWithReferenceFeature

1. Pipeline with Feature engine

* See slides for example





* Save into a Python pickle

# Intro to OOP

1. Learn and transform, learn and predict
2. Procedural programming in ML

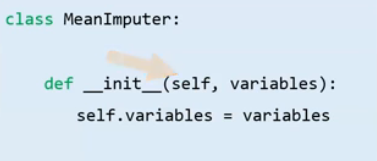
* Code
* Learn the params
* Make the transformations
* Make the predictions
* Data
* Store the params
* Mean values, regression coef, etc.

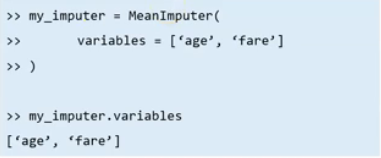
1. OOP

* In OOP, we write code in the form of objects
* This object can store data and can also store instructions or procedures (code) to modify that data, or do sth else, like obtaining predictions
* Data -> attributes, properties
* Code or instructions -> methods (procedures)
* Parameters get automatically refreshed every time model is re-trained
* No need of manual hard-coding
* Methods
  + Fit: learns params
  + Transform: transforms data with the learned params
* Attributes: store the learn params

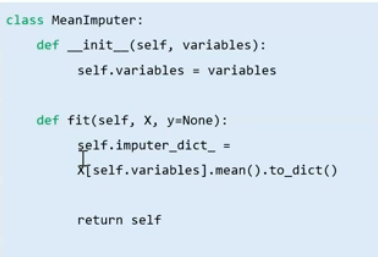
1. Creating a class in OOP

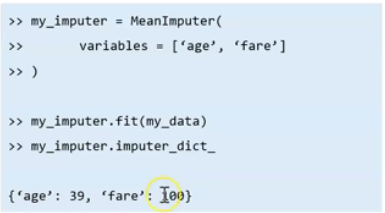
* A Class - **\_\_init\_\_()**

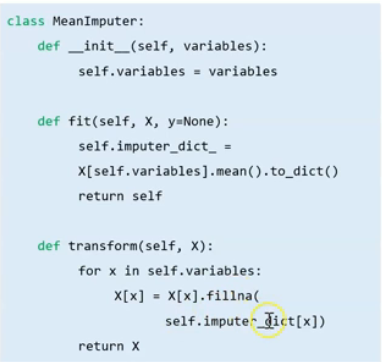




* The properties or params that the class takes whenever it is initialized, are indicated in the ***\_\_init\_\_()*** method
* The first params will always be a variable called ***self***
* We can give any number of params to \_\_init\_\_()
* A Class – **methods**

****

****

****

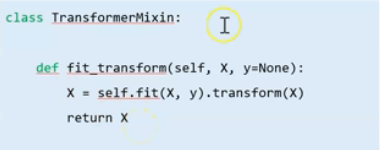
* Functions defined inside a class and can only be called from an instance of that class
* The first params will always be a variable called ***self***
* Our ***fit()*** methods learns params
* Our ***transform()*** method transforms data

# Inheritance and the Scikit-learn API

1. Inheritance

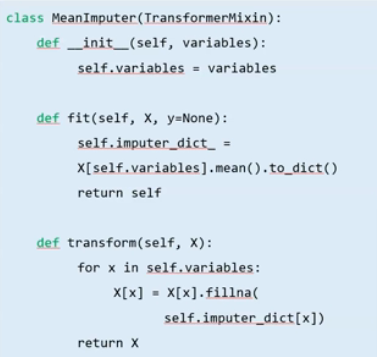
* The process by which one class takes the **attributes and methods** of another

1. Parent Class



* The properties or params that the class takes whenever it is initialized, are indicated in the \_\_init\_\_() method
* The first param will always be a variable called self
* We can give any number of param to \_\_init\_\_()

1. Our MeanImputer – Child Class



* Inherits the methods **fit\_transform()** from the TransformerMixin

# Create scikit-learn compatible transformers

# Should feature selection be part of the pipeline

1. Feature selection in the pipeline

* If deploying for the first time,
* Plus:
  + We don’t have to hard code the predictive features
* However,
  + Need to deploy code to engineer all features in the dataset
  + Error handling and unit testing for all the code to engineering features
* What if we re-train our model frequently?
* Advantages
  + Can quickly retrain a model on the same input data
  + No need to hard-code the new set of predictive features after each re-training
* Disadvantages
  + Lack of data versatility
  + No additional data can be fed through the pipeline
* In summary
* Suitable:
  + Model built and refreshed on same data
  + Model built and refreshed on smaller datasets
* Not suitable
  + If model built using datasets with a high feature space
  + If model constantly enriched with new data sources