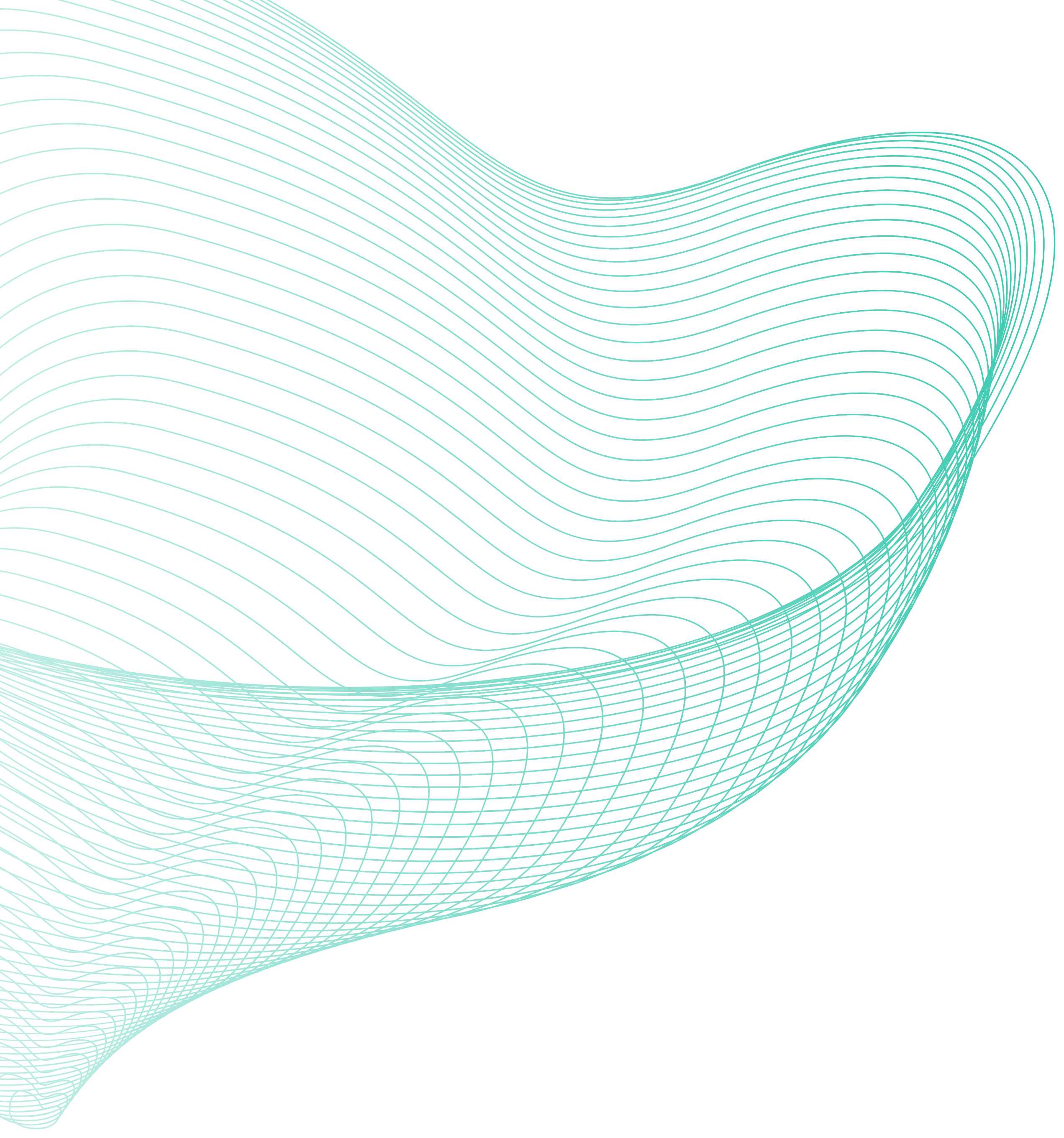
ML-DA project: Predicting house prices

Edoardo Pozzi, 4831619



STEPS

1. PRE-PROCESSING

- Filling missing value
- Data manipulation
- Removing unusefull rows

2. PRE-MODELLING

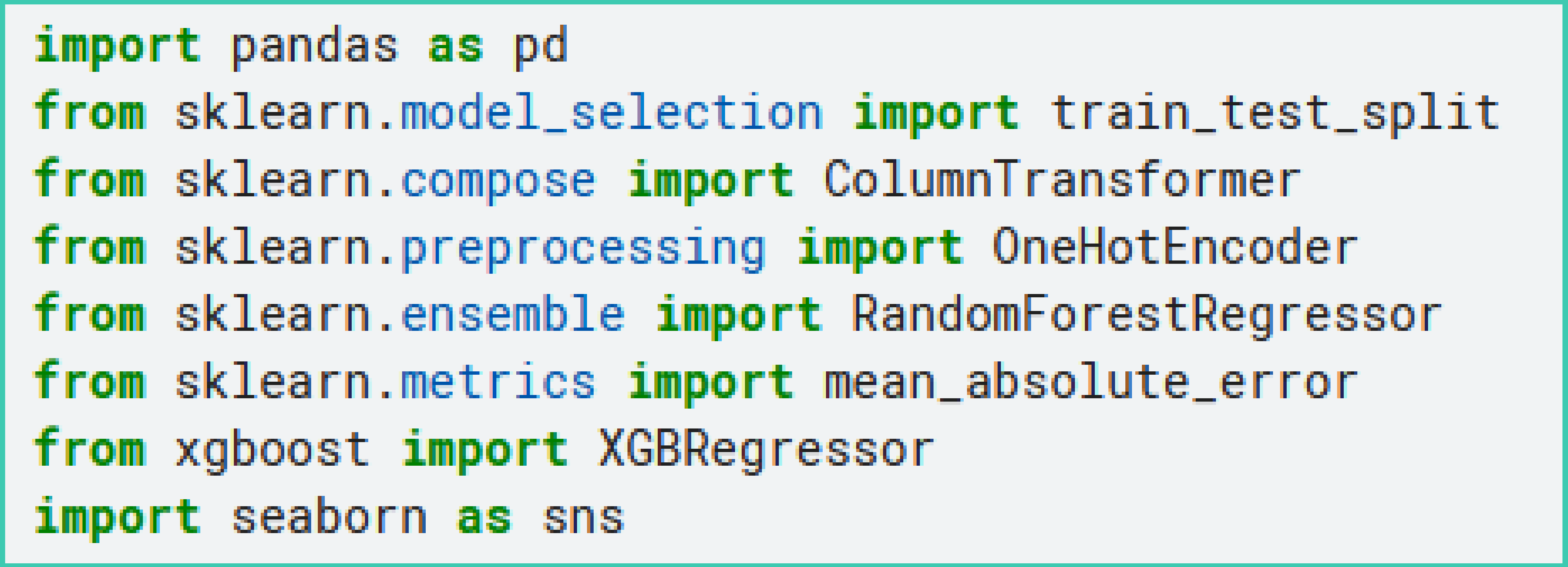
- Target and predictors selection
- Splitting dataset
- One-hot encoding

3. MODELLING

- Model definition
- Fitting
- Model prediction precision

TOOLS

- Pandas
- Splitter
- Enconder
- Gradient boosting
- Metrics



```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from xgboost import XGBRegressor
import seaborn as sns
```

Dataset

It's a CSV composed by:

Number of rows: 1460

Number of columns: 80

Rows with missing target: 0

All the features are used for predicting

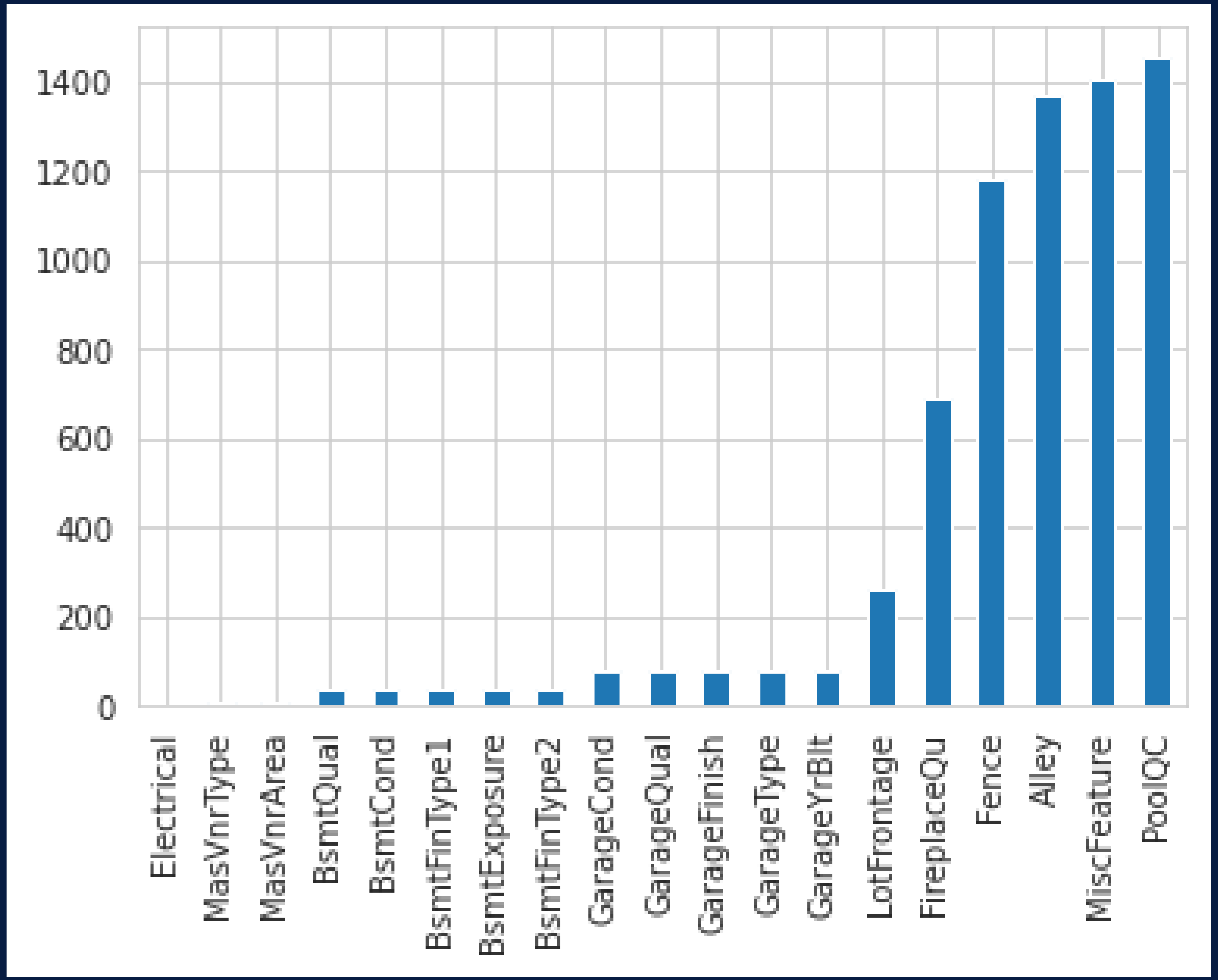


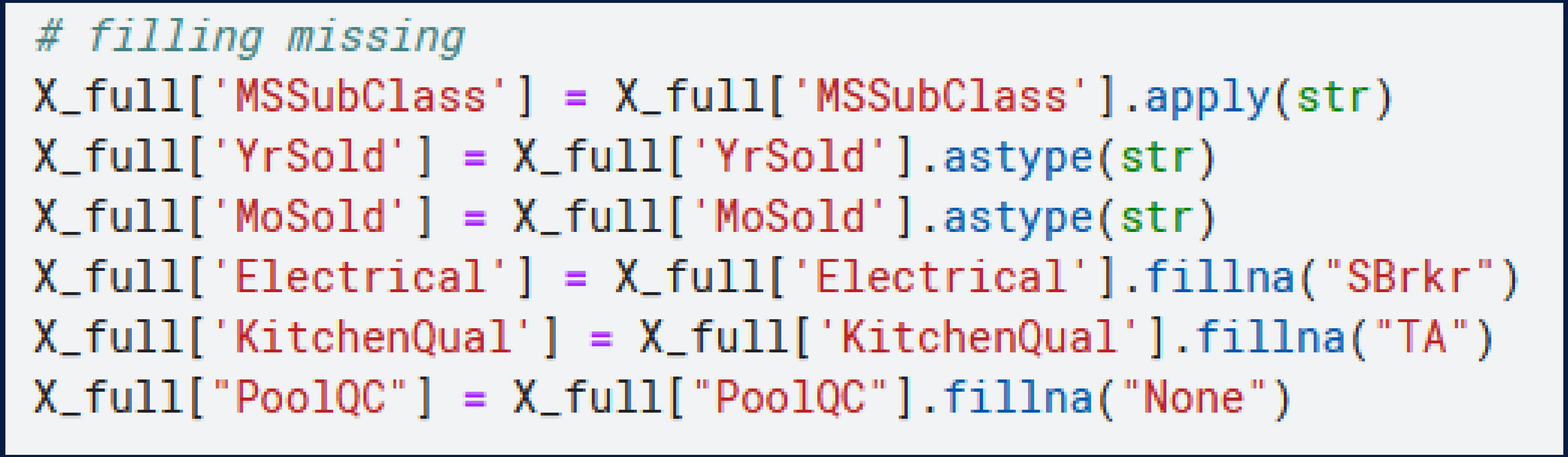
Chart of missing value by feature

MISSING DATA

Possible case:

- Too many missing => dropping columns
- Few missing => imputing (constant, mean, adjusted mean and so on)

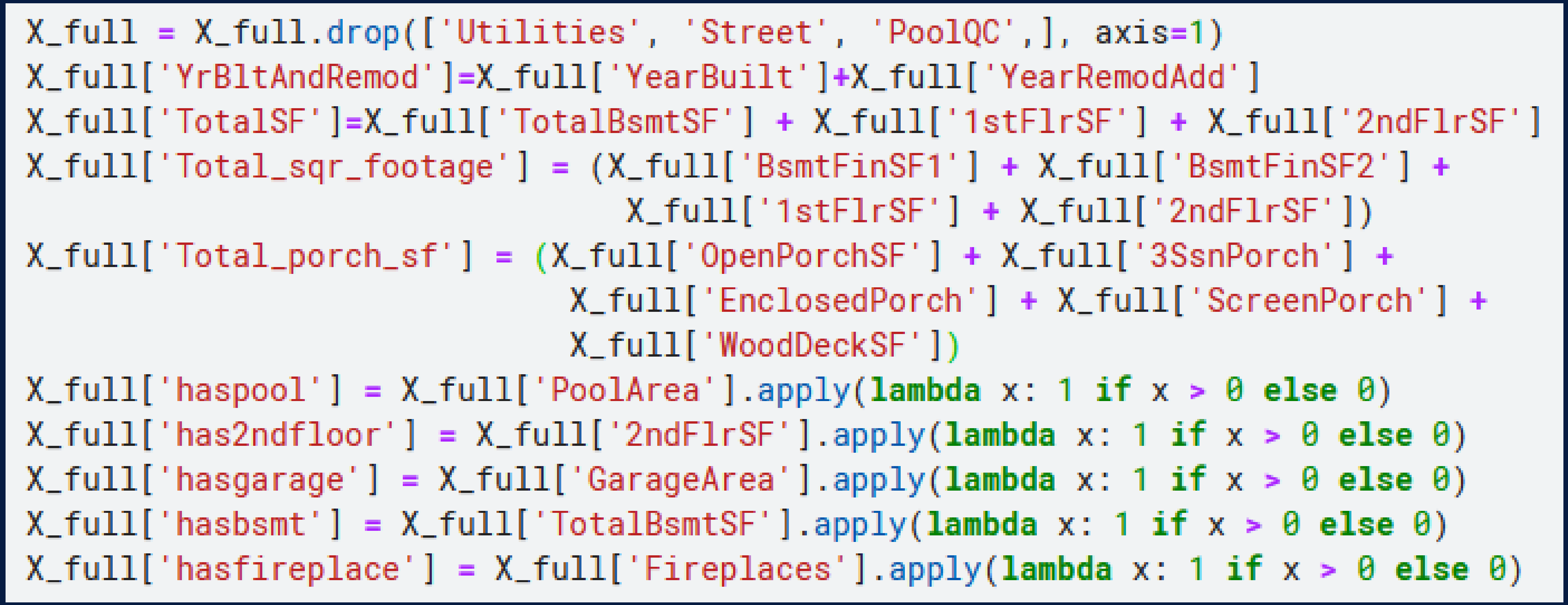
Imputing implies adding data randomly, if not done properly it could add artificial pattern not originally present, however there is no gain in information.



```python
# filling missing
X_full['MSSubClass'] = X_full['MSSubClass'].apply(str)
X_full['YrSold'] = X_full['YrSold'].astype(str)
X_full['MoSold'] = X_full['MoSold'].astype(str)
X_full['Electrical'] = X_full['Electrical'].fillna("SBrkr")
X_full['KitchenQual'] = X_full['KitchenQual'].fillna("TA")
X_full["PoolQC"] = X_full["PoolQC"].fillna("None")
```

DATA MANIPULATION

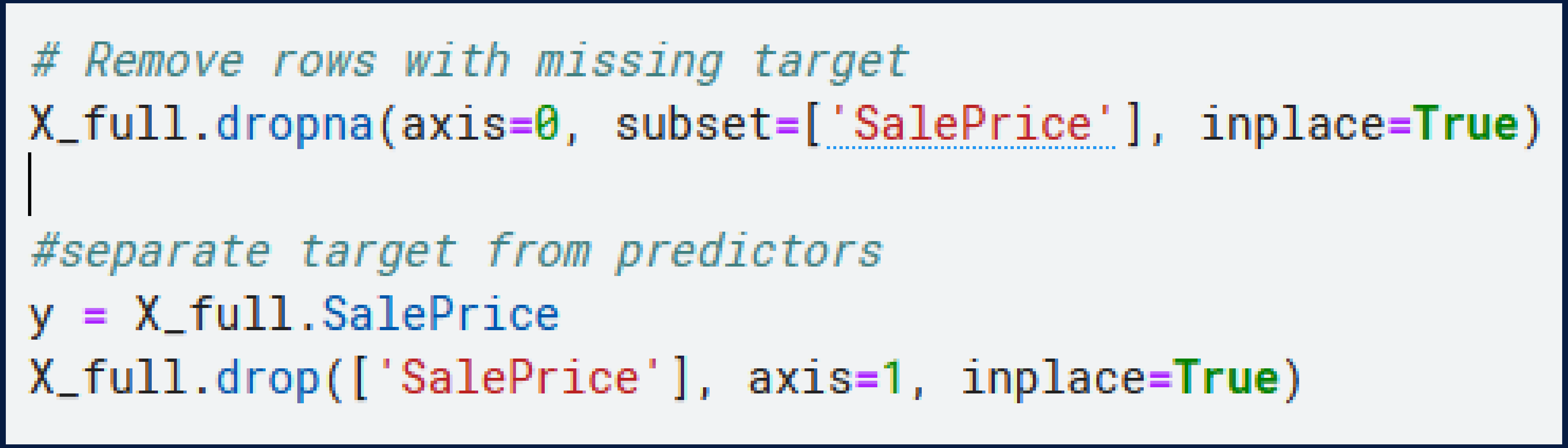
Some data could be added, such as the total dimension of the terrain (house + yard), or manipulated to create a different meaning such as "haspool" column used for differentiate between house with or without a pool in order to obtain a more accurate model.



```python
X_full = X_full.drop(['Utilities', 'Street', 'PoolQC',], axis=1)
X_full['YrBltAndRemod']=X_full['YearBuilt']+X_full['YearRemodAdd']
X_full['TotalSF']=X_full['TotalBsmtSF'] + X_full['1stFlrSF'] + X_full['2ndFlrSF']
X_full['Total_sqr_footage'] = (X_full['BsmtFinSF1'] + X_full['BsmtFinSF2'] +
                                X_full['1stFlrSF'] + X_full['2ndFlrSF'])
X_full['Total_porch_sf'] = (X_full['OpenPorchSF'] + X_full['3SsnPorch'] +
                             X_full['EnclosedPorch'] + X_full['ScreenPorch'] +
                             X_full['WoodDeckSF'])
X_full['haspool'] = X_full['PoolArea'].apply(lambda x: 1 if x > 0 else 0)
X_full['has2ndfloor'] = X_full['2ndFlrSF'].apply(lambda x: 1 if x > 0 else 0)
X_full['hasgarage'] = X_full['GarageArea'].apply(lambda x: 1 if x > 0 else 0)
X_full['hasbsmt'] = X_full['TotalBsmtSF'].apply(lambda x: 1 if x > 0 else 0)
X_full['hasfireplace'] = X_full['Fireplaces'].apply(lambda x: 1 if x > 0 else 0)
```

TARGET AND PREDICTORS

All the rows that don't contain the target couldn't be used to train or test, so they must be dropped.
y will be our target, the 'SalesPrice'
X will be our predictors



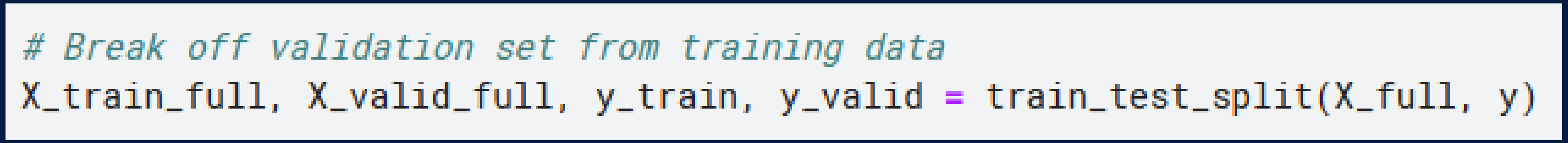
```python
# Remove rows with missing target
X_full.dropna(axis=0, subset=['SalePrice'], inplace=True)

#separate target from predictors
y = X_full.SalePrice
X_full.drop(['SalePrice'], axis=1, inplace=True)
```

DATA SPLITTING

Data must be splitted in 'train' and 'valid' set in order to avoid bias caused by the fact that during the training the model already knows some answers.

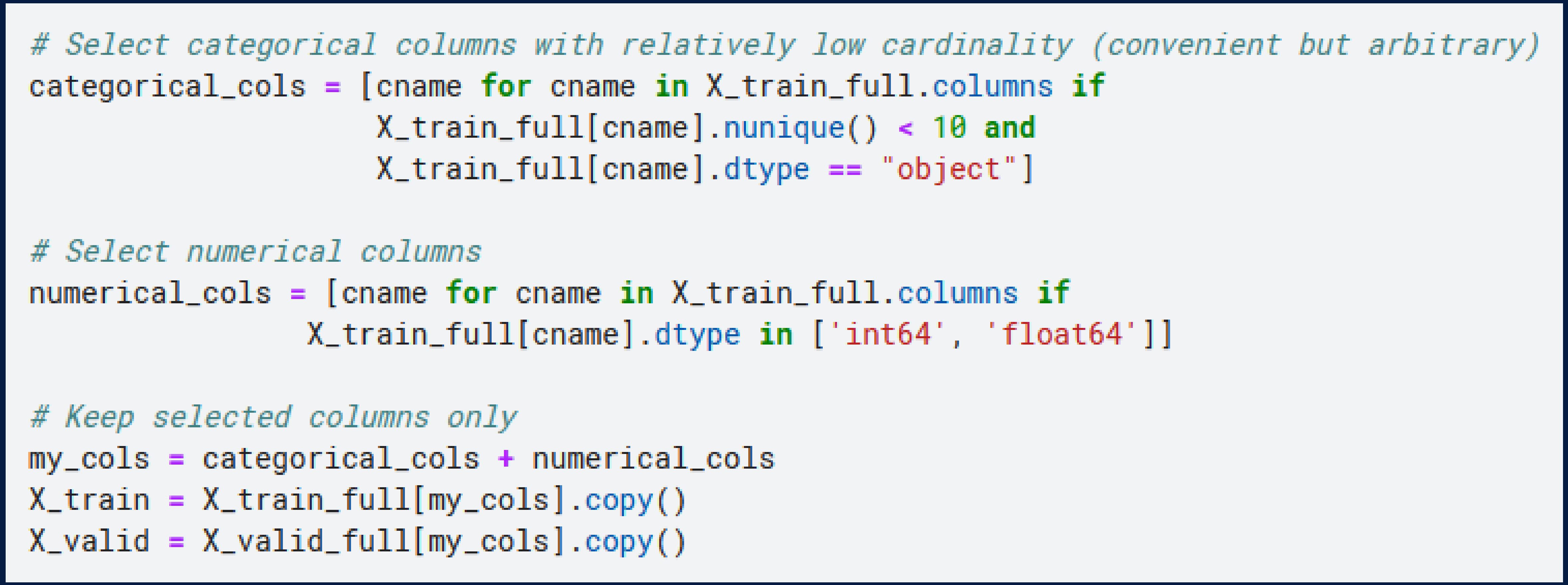
train_test_split function provided by sklearn does that automatically adjusting the percentage of splitting.



```
# Break off validation set from training data
X_train_full, X_valid_full, y_train, y_valid = train_test_split(X_full, y)
```

COLUMN SELECTION

When one-hot encoding a column, a function expands that column out to multiple columns, one for each unique value. The dataset can grow quickly, with the resulting data very sparse if you do it with columns that have many unique values. It's best to only use one-hot encoding on categorical columns with a few unique values.



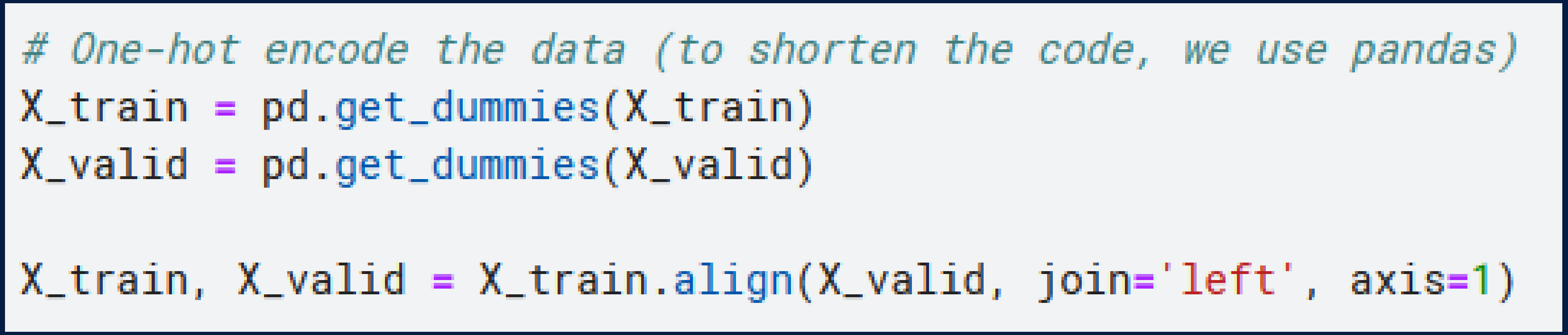
```python
# Select categorical columns with relatively low cardinality (convenient but arbitrary)
categorical_cols = [cname for cname in X_train_full.columns if
                    X_train_full[cname].nunique() < 10 and
                    X_train_full[cname].dtype == "object"]

# Select numerical columns
numerical_cols = [cname for cname in X_train_full.columns if
                X_train_full[cname].dtype in ['int64', 'float64']]

# Keep selected columns only
my_cols = categorical_cols + numerical_cols
X_train = X_train_full[my_cols].copy()
X_valid = X_valid_full[my_cols].copy()
```

ENCODING

.get_dummies() converts categorical variables into dummy/indicator variables in order to be used in models that can't accept categorical values.

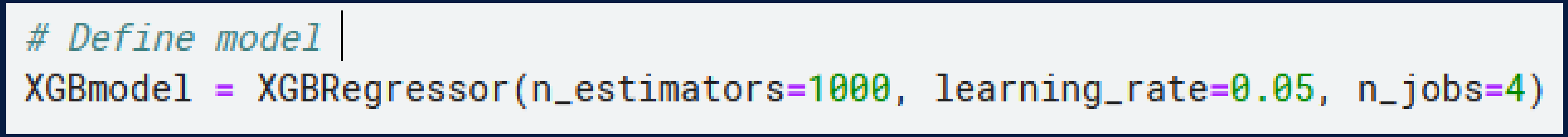


```
# One-hot encode the data (to shorten the code, we use pandas)
X_train = pd.get_dummies(X_train)
X_valid = pd.get_dummies(X_valid)

X_train, X_valid = X_train.align(X_valid, join='left', axis=1)
```

MODEL DEFINITION

Extreme gradient boosting relies on the intuition that the best possible next model, when combined with previous models, minimizes the overall prediction error.



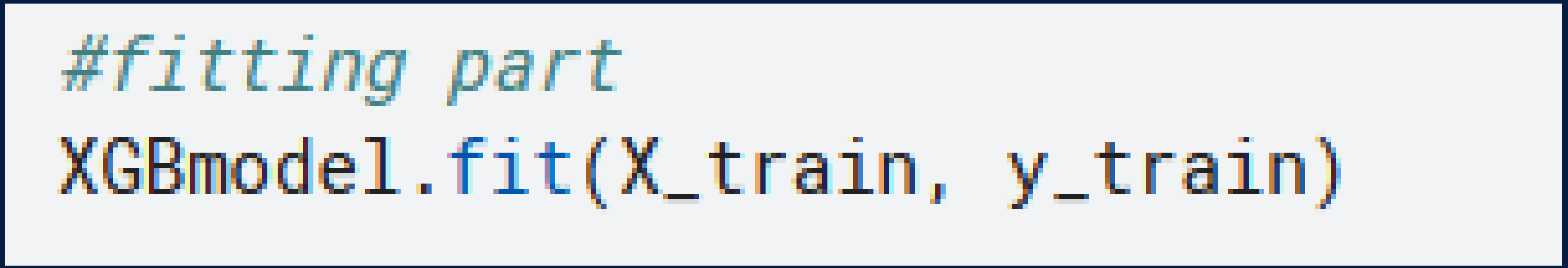
```
# Define model
XGBmodel = XGBRegressor(n_estimators=1000, learning_rate=0.05, n_jobs=4)
```

Possible parameters:

- n_estimator: number of trees
- learning rate: shrinkage done at every step you are making.
- n_jobs
- max depth tree
- Gamma: complexity regulator
- Max leaves

FITTING

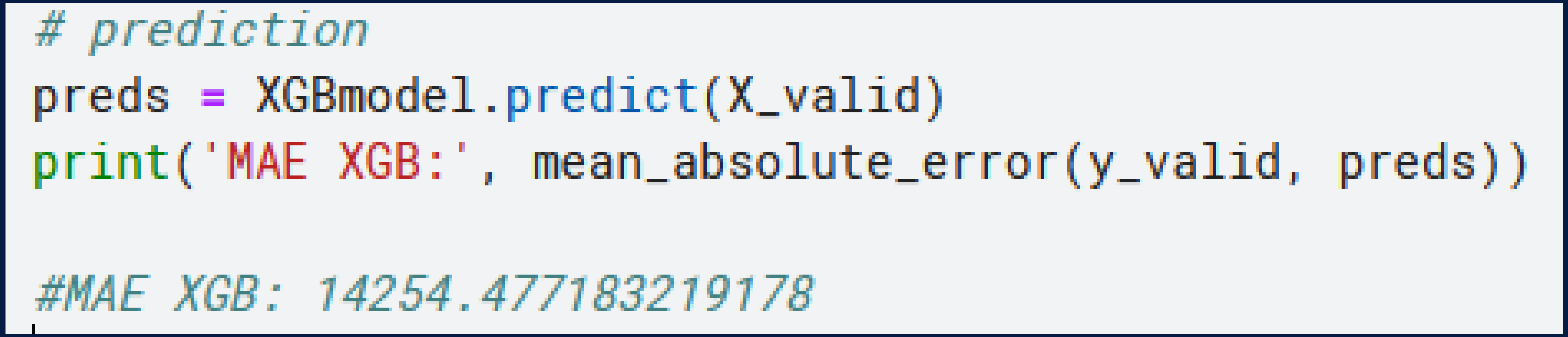
The train dataset is passed at the fit function in order to learn from it.



```
#fitting part
XGBmodel.fit(X_train, y_train)
```

PREDICTIONS AND MAE

After fitting the function predict is called to make prediction using the validation dataset that doesn't contain the target.
A way to calculate the precision of the model is through the 'mean absolute value' that measures the error between the predicted and real value.
In this case the average price is 180k this means a mean error on the prediction of 7%.


```
# prediction
preds = XGBmodel.predict(X_valid)
print('MAE XGB:', mean_absolute_error(y_valid, preds))

#MAE XGB: 14254.477183219178
```

Thanks for the attention