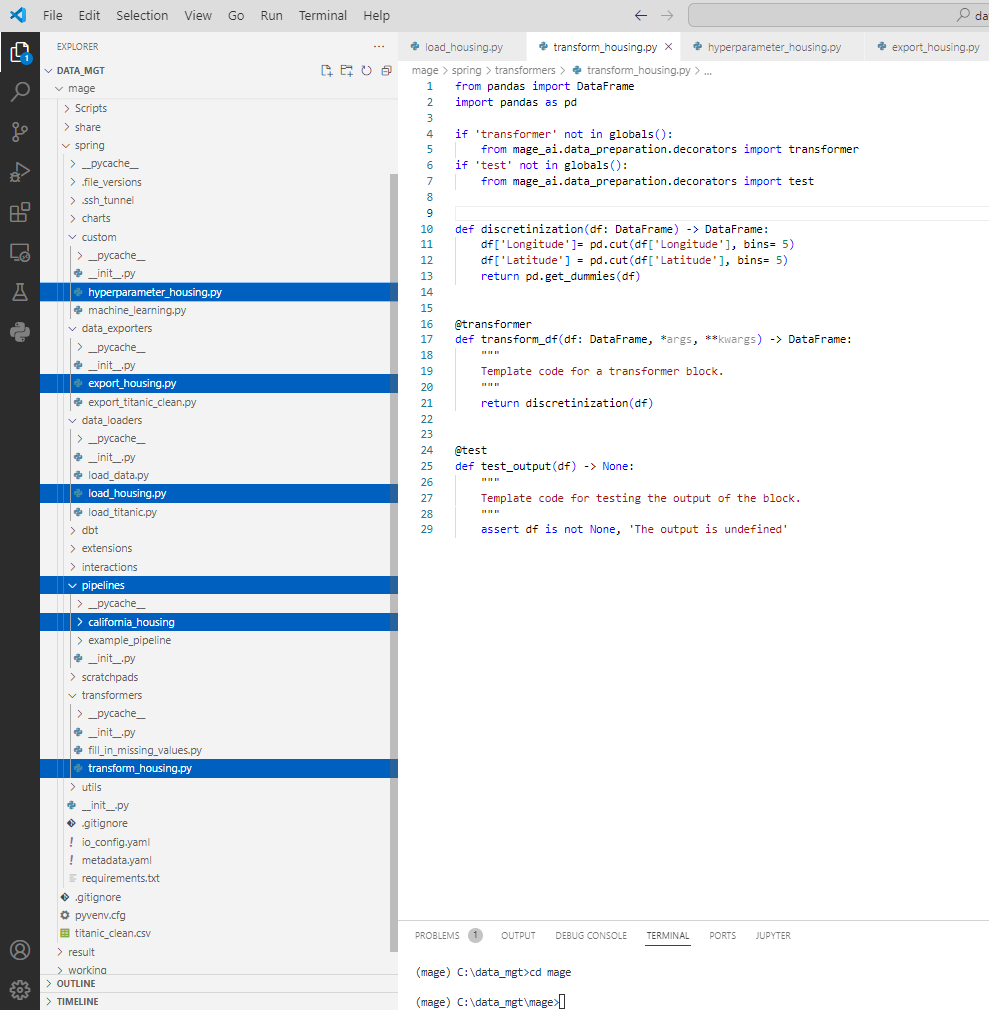
**4. hyperparameter\_for\_mage**

**Hyperparameter tuning** [**getting\_started**](https://scikit-learn.org/stable/getting_started.html) **&** [**model\_persistence**](https://scikit-learn.org/stable/model_persistence.html)

* 1. **Vs Code**

****

Dataloader block

from sklearn.datasets import fetch\_california\_housing

from pandas import DataFrame

import pandas as pd

if 'data\_loader' not in globals():

    from mage\_ai.data\_preparation.decorators import data\_loader

if 'test' not in globals():

    from mage\_ai.data\_preparation.decorators import test

print(fetch\_california\_housing().keys())

@data\_loader

def load\_data\_from\_api(\*\*kwargs) -> DataFrame:

    X = pd.DataFrame(fetch\_california\_housing().data,

                 columns=fetch\_california\_housing().feature\_names)

    y = pd.DataFrame(fetch\_california\_housing().target,

                 columns=fetch\_california\_housing().target\_names)

    return pd.merge(X, y, left\_index=True, right\_index=True)

@test

def test\_output(df) -> None:

    """

    Template code for testing the output of the block.

    """

    assert df is not None, 'The output is undefined'

Datatransformer block

from pandas import DataFrame

import pandas as pd

if 'transformer' not in globals():

    from mage\_ai.data\_preparation.decorators import transformer

if 'test' not in globals():

    from mage\_ai.data\_preparation.decorators import test

def discretinization(df: DataFrame) -> DataFrame:

    df['Longitude']= pd.cut(df['Longitude'], bins= 5)

    df['Latitude'] = pd.cut(df['Latitude'], bins= 5)

    return pd.get\_dummies(df)

@transformer

def transform\_df(df: DataFrame, \*args, \*\*kwargs) -> DataFrame:

    """

    Template code for a transformer block.

    """

    return discretinization(df)

@test

def test\_output(df) -> None:

    """

    Template code for testing the output of the block.

    """

    assert df is not None, 'The output is undefined'

@custom

import pandas as pd

import os

import pickle

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from scipy.stats import randint

if "custom" not in globals():

    from mage\_ai.data\_preparation.decorators import custom

if "test" not in globals():

    from mage\_ai.data\_preparation.decorators import test

def \_model\_save(rf\_model):

    """

    Save RandomForest model to a file.

    """

    # Here you would implement the logic to save your trained model to a file

    # Example:

    os.makedirs('../result', exist\_ok= True)

    with open('../result/random\_forest\_model.pkl', 'wb') as file:

        pickle.dump(rf\_model, file)

    pass

@custom

def random\_forest\_train(df: pd.DataFrame, \*args, \*\*kwargs):

    """

    Train a Random Forest Classifier and predict the 'Survived' column.

    Args:

        df: Data frame containing the training data.

    Returns:

        Data frame with a new column 'Survived\_predict' with predictions.

    """

    # Prepare the data

    X = df.drop(['MedHouseVal'], axis=1)

    y = df['MedHouseVal']

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)

    # define the parameter space that will be searched over

    param\_distributions = {'n\_estimators': randint(1, 5),

                           'max\_depth': randint(5, 10)}

    # now create a searchCV object and fit it to the data

    search = RandomizedSearchCV(estimator=RandomForestRegressor(random\_state=0),

                                n\_iter=10,

                                param\_distributions=param\_distributions,

                                random\_state=0)

    # Initialize the Random Forest Classifier

    rf\_model = RandomForestRegressor(random\_state=0),

    # Train the model, now create a searchCV object and fit it to the data

    search = RandomizedSearchCV(estimator=RandomForestRegressor(random\_state=0),

                                n\_iter=10,

                                param\_distributions=param\_distributions,

                                random\_state=0)

    search.fit(X\_train, y\_train)

    tf = pd.DataFrame(search.cv\_results\_)[['param\_max\_depth', 'param\_n\_estimators', 'params', 'mean\_test\_score', 'rank\_test\_score']]

    print(tf.sort\_values('rank\_test\_score'))

    print(f"The Best model's parameters is {search.best\_params\_}")

    print(f'The Best accuracy score of model is {search.score(X\_test, y\_test)}')

    # Predict using the trained model

    y\_pred = search.predict(X\_test)

    # Optionally save the model

    \_model\_save(search)

    # Assign predictions to a new column in the dataframe

    tf = pd.merge(X\_test, y\_test, left\_index=True, right\_index=True)

    tf['Hosing\_pred'] = y\_pred

    return tf

@test

def test\_output(output, \*args) -> None:

    """

    Template code for testing the output of the block.

    Args:

        output: The output from the random\_forest\_train function.

    """

    assert output is not None, "The output is undefined"

    assert 'Survived\_predict' in output.columns, "Prediction column is missing in the output dataframe"

    # You can add more tests to check the quality of your predictions,

    # such as accuracy score, confusion matrix, etc.

@data\_exporter

from mage\_ai.io.file import FileIO

from pandas import DataFrame

if 'data\_exporter' not in globals():

    from mage\_ai.data\_preparation.decorators import data\_exporter

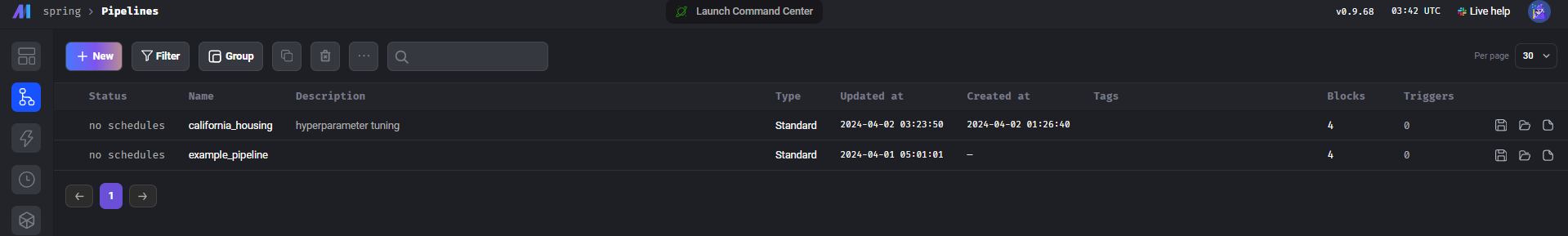
@data\_exporter

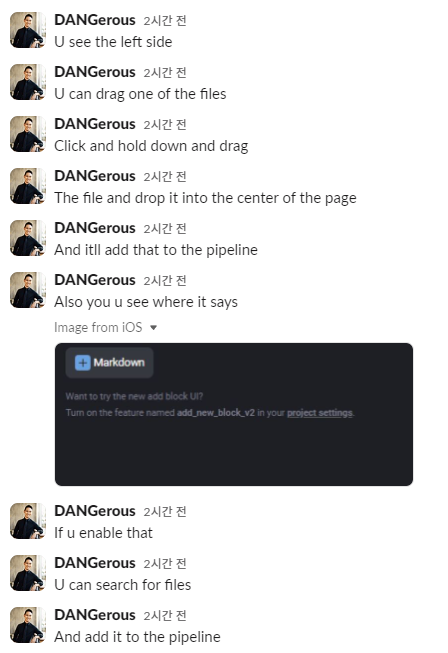
def export\_data\_to\_file(df: DataFrame, \*\*kwargs) -> None:

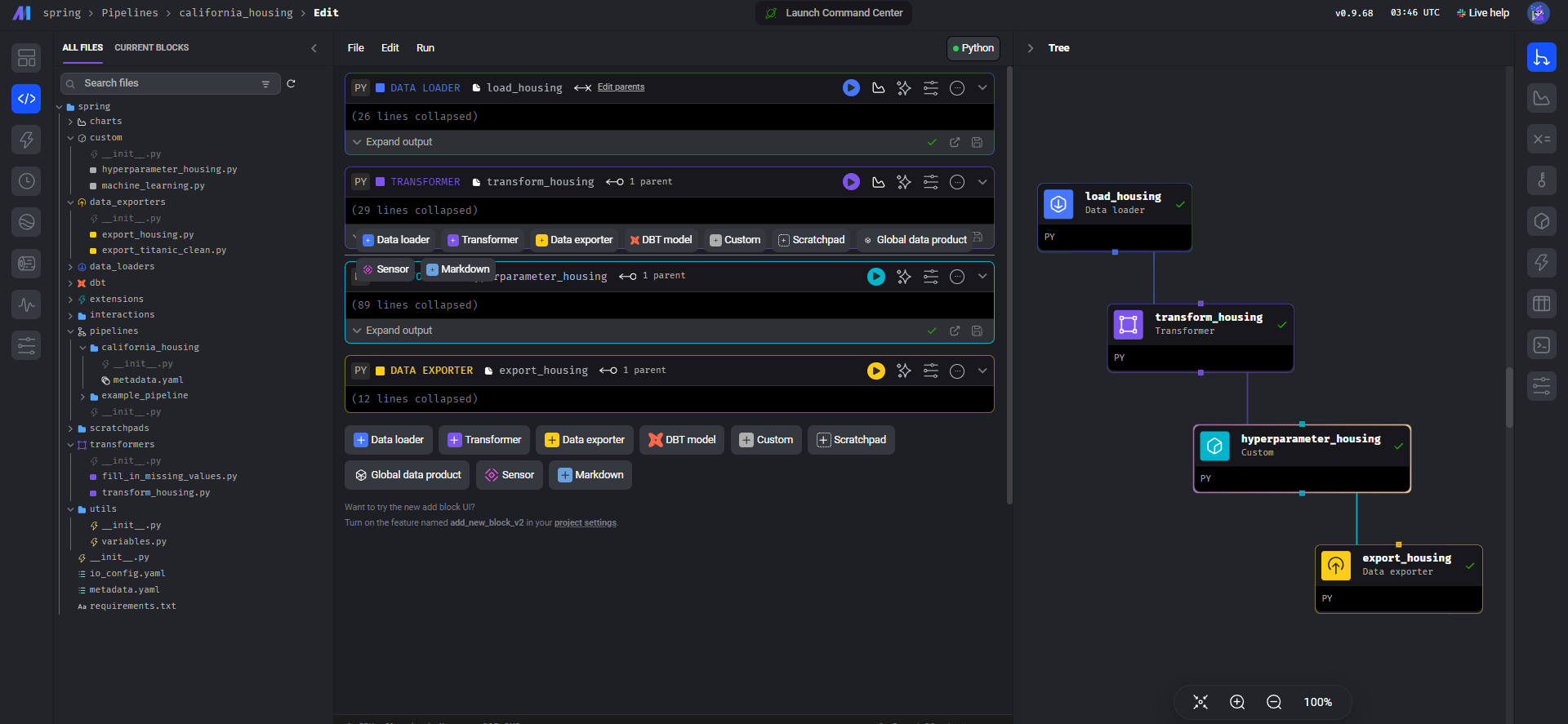
    filepath = '../result/housing\_predict.csv'

    FileIO().export(df, filepath)

* 1. **Mage.ai WEB UI**
* (mage) C:\data\_mgt\mage>mage start spring
* spring > Pipeline > click  > click pipeline name



****

****

* **py파일을 편집하고 싶으면 VS code에서 편집하고 저장(ctrl + s),** **화면 reset**