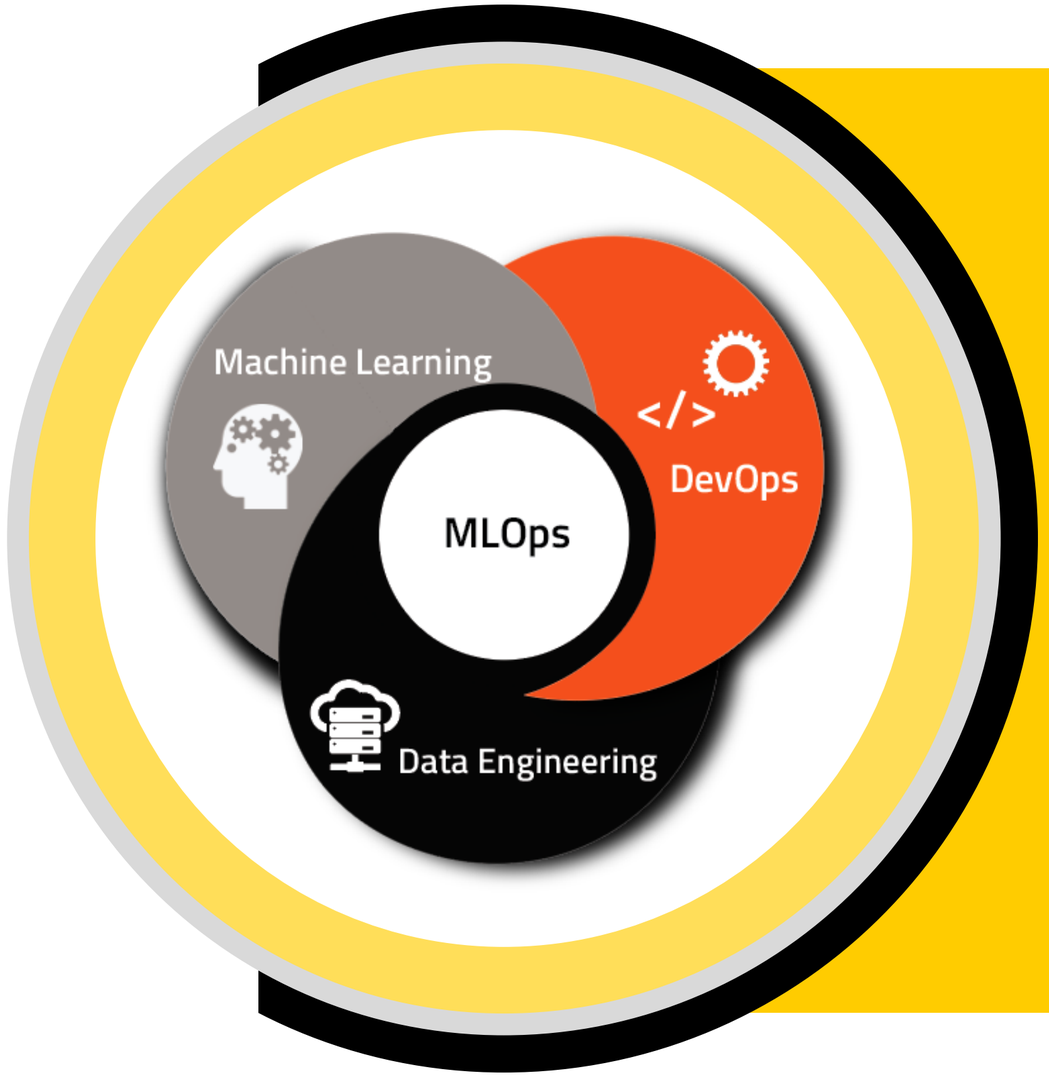
StatusNeo

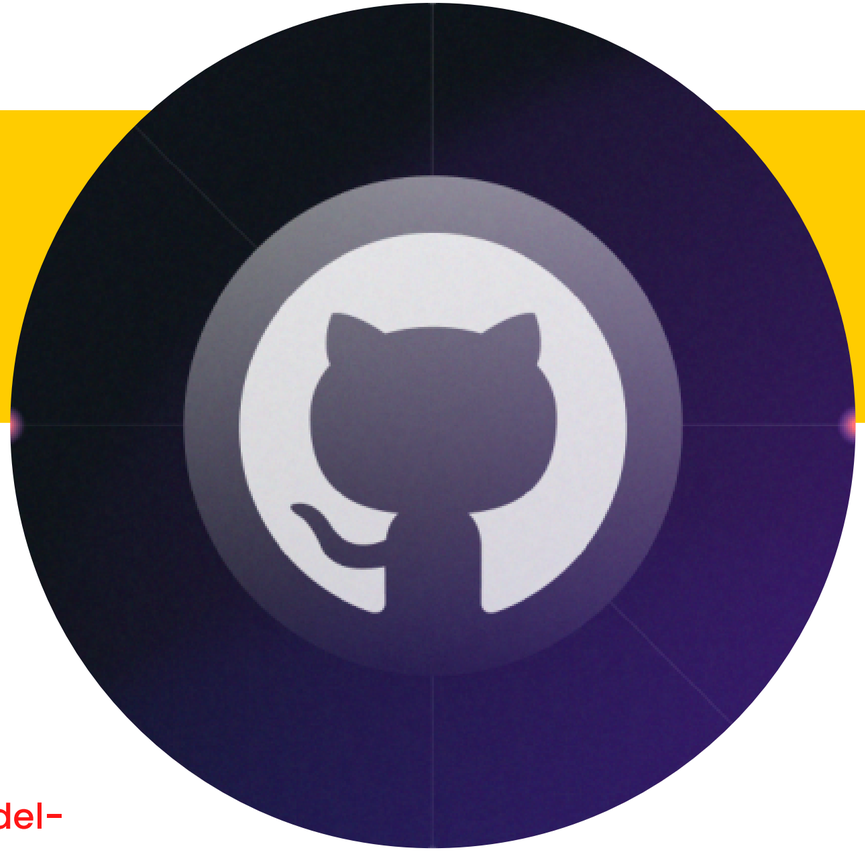


MLOps

Packaging Model for Production

SESSION 4

Download the code



Just a quick reminder if you haven't done this yet. Login to Kaggle and go to:

https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data

Download the train.csv and test.csv and put them in the section-05-production-model-package/regression_model/datasets directory

https://github.com/trainindata/deploying-machine-learning-models/tree/master/section-05-production-model-package/regression_model/datasets

What is a Python Package

A python module is basically just a python file and a package is a collection of modules.

A python package has certain **standardized** files which have to be present so that it can be **published** and then **installed** in other **Python applications**.

Why are we taking this approach?

A package allows us to wrap our train model and make it available to other consuming applications as a dependency, but with the additional benefits of

- version control
- clear metadata and
- reproducibility

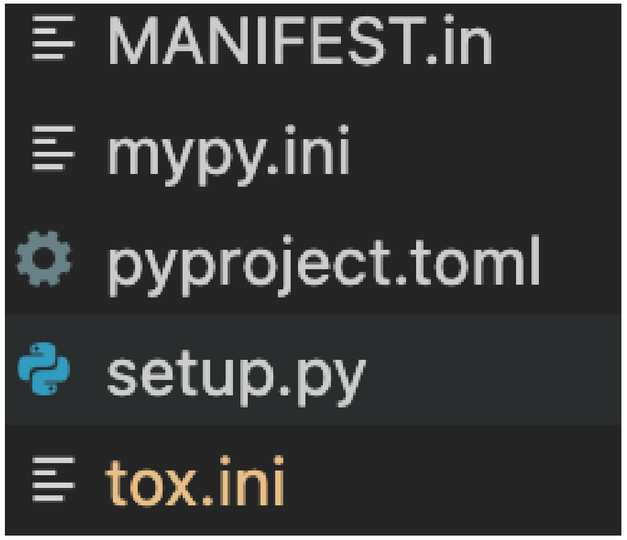
Now, in order to create a package, we'll have to follow certain Python standards and conventions and we'll go into those in detail in this section.

```
parent/
├── regression_model/
│   └── config/
│   └── datasets/
│   └── processing/
│   └── trained_models/
│   ├── config.yml
│   ├── pipeline.py
│   ├── predict.py
│   ├── train_pipeline.py
│   └── VERSION
├── requirements/
│   ├── requirements.txt
│   └── test_requirements.txt
├── setup.py
├── MANIFEST.in
├── pyproject.toml
├── tox.ini
└── tests/
```

We'll end up with a package structure that looks like this.

StatusNeo

Let's walkthrough the code

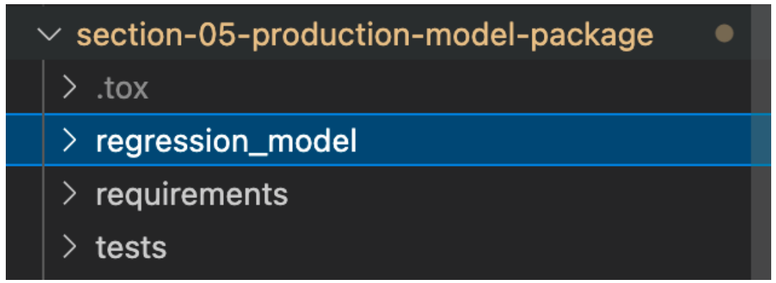


These are all things that we're using, either for packaging or for configuring things like linting and type checking these kinds of tooling configurations

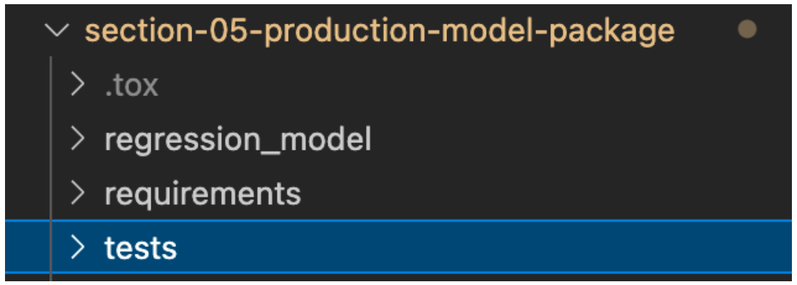
There is this hidden .tox directory is where Tox installs virtual environments.

StatusNeo

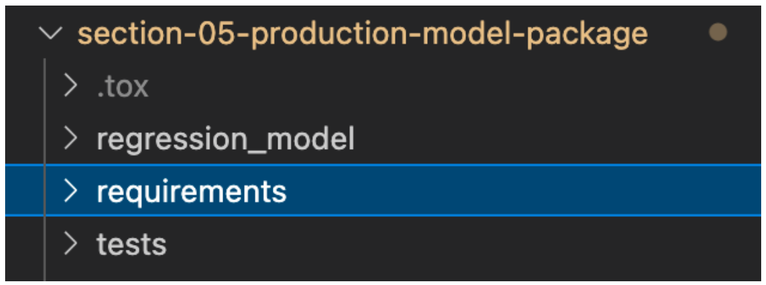
Let's walkthrough the code



Regression Model directory is where majority of the model related functioning is

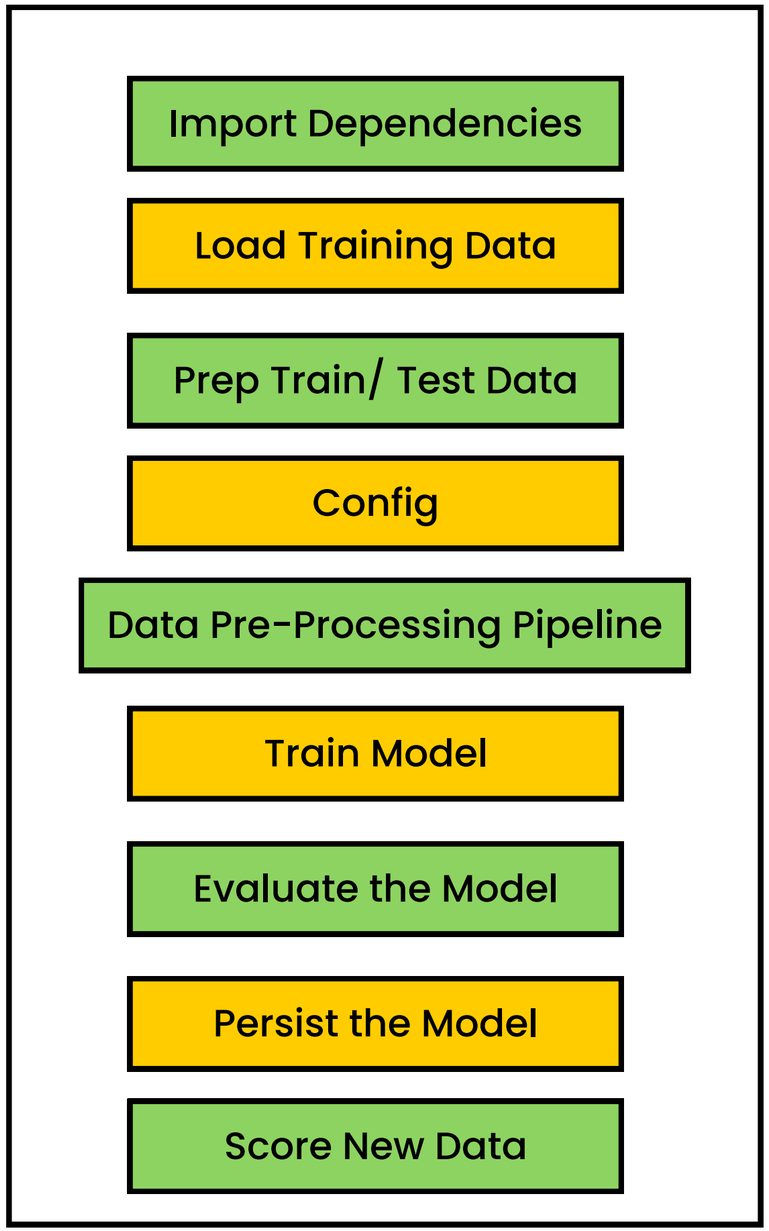


Test Directory consist of sample test for our model



A requirements directory, which is where we formalize the dependencies for our package

StatusNeo



Jupyter Notebook to Production Code

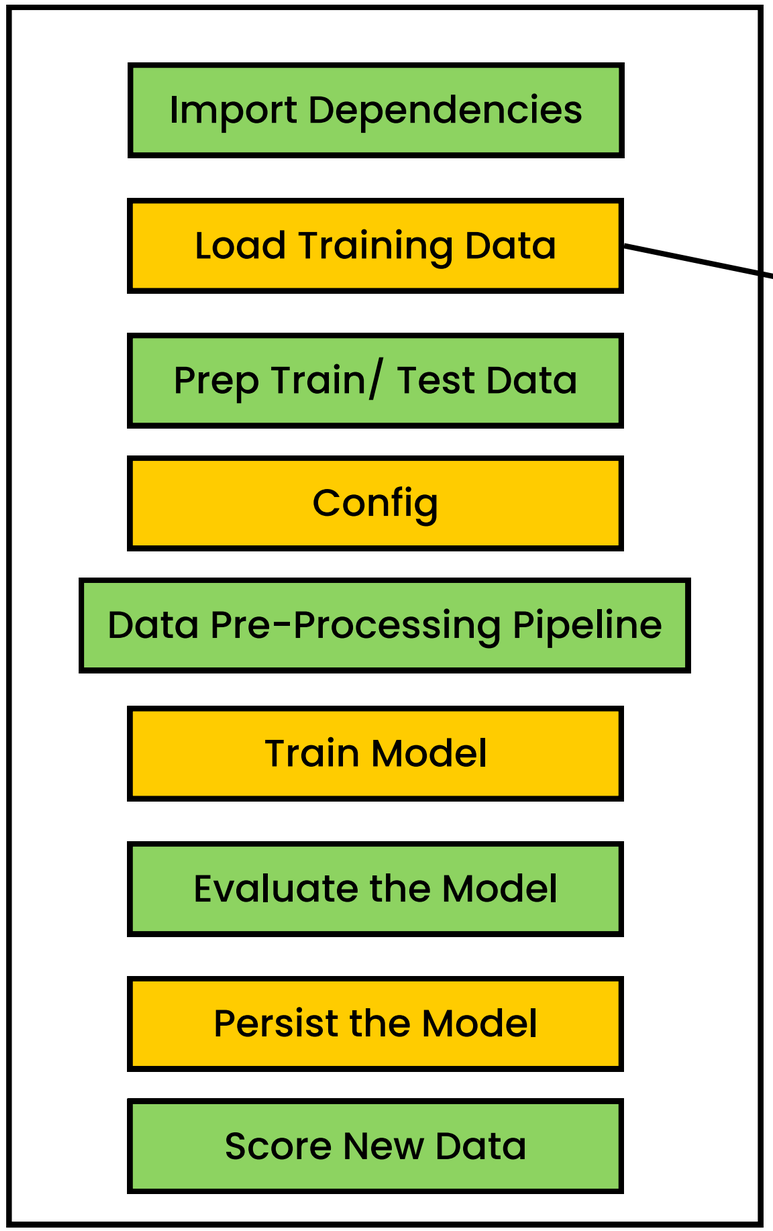
StatusNeo

- Import Dependencies
- Load Training Data
- Prep Train/ Test Data
- Config
- Data Pre-Processing Pipeline
- Train Model
- Evaluate the Model
- Persist the Model
- Score New Data

In [3]:

```
# data manipulation and plotting
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# for saving the pipeline
import joblib

# from Scikit-learn
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler, Binarizer

# from feature-engine
from feature_engine.imputation import (
    AddMissingIndicator,
    MeanMedianImputer,
    CategoricalImputer,
)

from feature_engine.encoding import (
    RareLabelEncoder,
    OrdinalEncoder,
)

from feature_engine.transformation import LogTransformer

from feature_engine.selection import DropFeatures
from feature_engine.wrappers import SklearnTransformerWrapper

import preprocessors as pp
```

StatusNeo




```
# load dataset
data = pd.read_csv('train.csv')

# rows and columns of the data
print(data.shape)

# visualise the dataset
data.head()
```

(1460, 81)

StatusNeo

- Import Dependencies
- Load Training Data
- Prep Train/ Test Data
- Config
- Data Pre-Processing Pipeline
- Train Model
- Evaluate the Model
- Persist the Model
- Score New Data

Separating the data into train and test involves randomness, therefore, we need to set the seed.

In [6]:

```
# Let's separate into train and test set
# Remember to set the seed (random_state for this sklearn function)

X_train, X_test, y_train, y_test = train_test_split(
    data.drop(['Id', 'SalePrice'], axis=1), # predictive variables
    data['SalePrice'], # target
    test_size=0.1, # portion of dataset to allocate to test set
    random_state=0, # we are setting the seed here
)

X_train.shape, X_test.shape
```

StatusNeo

- Import Dependencies
- Load Training Data
- Prep Train/ Test Data
- Config
- Data Pre-Processing Pipeline
- Train Model
- Evaluate the Model
- Persist the Model
- Score New Data

Configuration

In [8]:

```
# categorical variables with NA in train set
CATEGORICAL_VARS_WITH_NA_FREQUENT = ['BsmtQual', 'BsmtExposure',
                                     'BsmtFinType1', 'GarageFinish']


CATEGORICAL_VARS_WITH_NA_MISSING = ['FireplaceQu']


# numerical variables with NA in train set
NUMERICAL_VARS_WITH_NA = ['LotFrontage']


TEMPORAL_VARS = ['YearRemodAdd']
REF_VAR = "YrSold"

# this variable is to calculate the temporal variable,
# can be dropped afterwards
DROP_FEATURES = ["YrSold"]

# variables to log transform
NUMERICALS_LOG_VARS = ["LotFrontage", "1stFlrSF", "GrLivArea"]


# variables to binarize
BINARIZE_VARS = ['ScreenPorch']

# variables to map
QUAL_VARS = ['ExterQual', 'BsmtQual',
             'HeatingQC', 'KitchenQual', 'FireplaceQu']

EXPOSURE_VARS = ['BsmtExposure']

FINISH_VARS = ['BsmtFinType1']

GARAGE_VARS = ['GarageFinish']

FENCE_VARS = ['Fence']


# categorical variables to encode
CATEGORICAL_VARS = ['MSSubClass',  'MSZoning',  'LotShape',  'LandContour',
                    'LotConfig', 'Neighborhood', 'RoofStyle', 'Exterior1st',
                    'Foundation', 'CentralAir', 'Functional', 'PavedDrive',
                    'SaleCondition']
```

StatusNeo

- Import Dependencies
- Load Training Data
- Prep Train/ Test Data
- Config
- Data Pre-Processing Pipeline
- Train Model
- Evaluate the Model
- Persist the Model
- Score New Data

In [10]:

```
# set up the pipeline
price_pipe = Pipeline([

    # ===== IMPUTATION =====
    # impute categorical variables with string missing
    ('missing_imputation', CategoricalImputer(
        imputation_method='missing', variables=CATEGORICAL_VARS_WITH_NA_MISSING)),

    ('frequent_imputation', CategoricalImputer(
        imputation_method='frequent', variables=CATEGORICAL_VARS_WITH_NA_FREQUENT)),

    # add missing indicator
    ('missing_indicator', AddMissingIndicator(variables=NUMERICAL_VARS_WITH_NA)),

    # impute numerical variables with the mean
    ('mean_imputation', MeanMedianImputer(
        imputation_method='mean', variables=NUMERICAL_VARS_WITH_NA
    )),


    # == TEMPORAL VARIABLES ====
    ('elapsed_time', pp.TemporalVariableTransformer(
        variables=TEMPORAL_VARS, reference_variable=REF_VAR)),

    ('drop_features', DropFeatures(features_to_drop=[REF_VAR])),



    # ==== VARIABLE TRANSFORMATION =====
    ('log', LogTransformer(variables=NUMERICALS_LOG_VARS)),

#     ('yeojohnson', YeoJohnsonTransformer(variables=NUMERICALS_YEO_VARS)),

    ('binarizer', SklearnTransformerWrapper(
        transformer=Binarizer(threshold=0), variables=BINARIZE_VARS)),


    # === mappers ===
    ('mapper_qual', pp.Mapper(
        variables=QUAL_VARS, mappings=QUAL_MAPPINGS)),

    ('mapper_exposure', pp.Mapper(
        variables=EXPOSURE_VARS, mappings=EXPOSURE_MAPPINGS)),

    ('mapper_finish', pp.Mapper(
        variables=FINISH_VARS, mappings=FINISH_MAPPINGS)),

    ('mapper_garage', pp.Mapper(
        variables=GARAGE_VARS, mappings=GARAGE_MAPPINGS)),

#     ('mapper_fence', pp.Mapper(
#         variables=FENCE_VARS, mappings=FENCE_MAPPINGS)),


    # == CATEGORICAL ENCODING
    ('rare_label_encoder', RareLabelEncoder(
        tol=0.01, n_categories=1, variables=CATEGORICAL_VARS
    )),
```

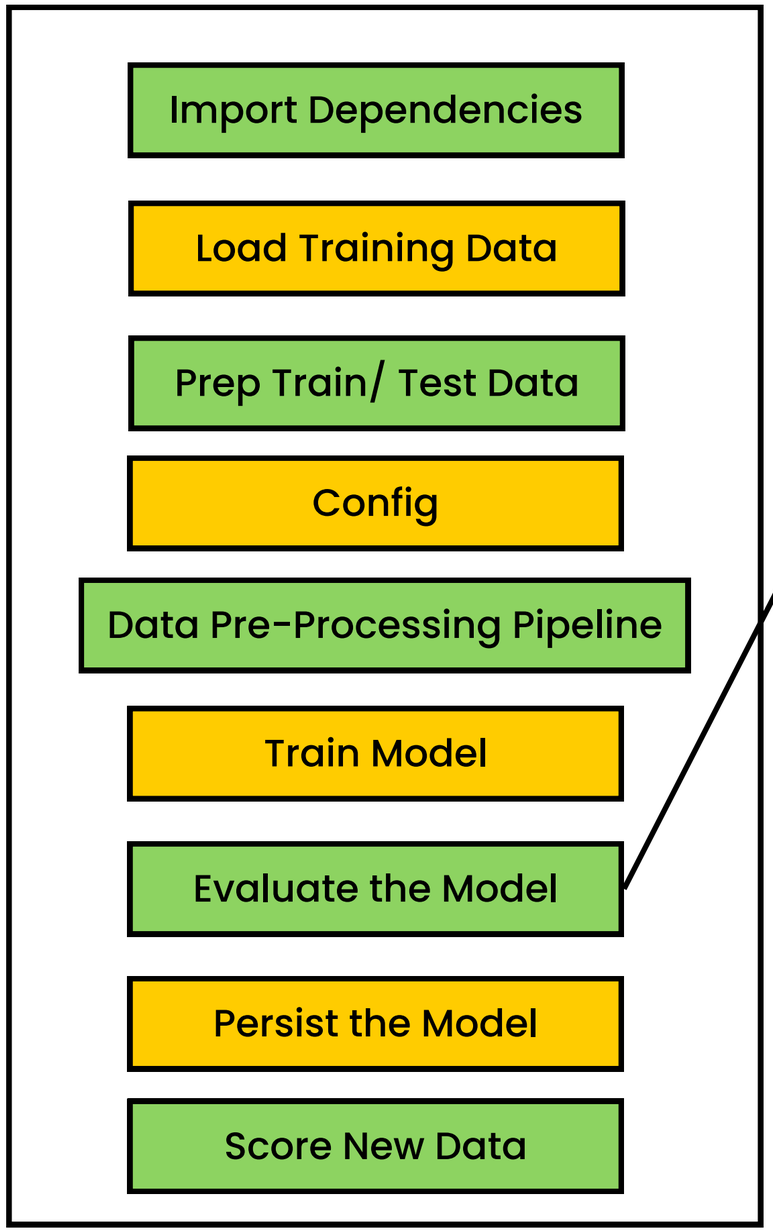
StatusNeo

- Import Dependencies
- Load Training Data
- Prep Train/ Test Data
- Config
- Data Pre-Processing Pipeline
- Train Model
- Evaluate the Model
- Persist the Model
- Score New Data

In [11]:

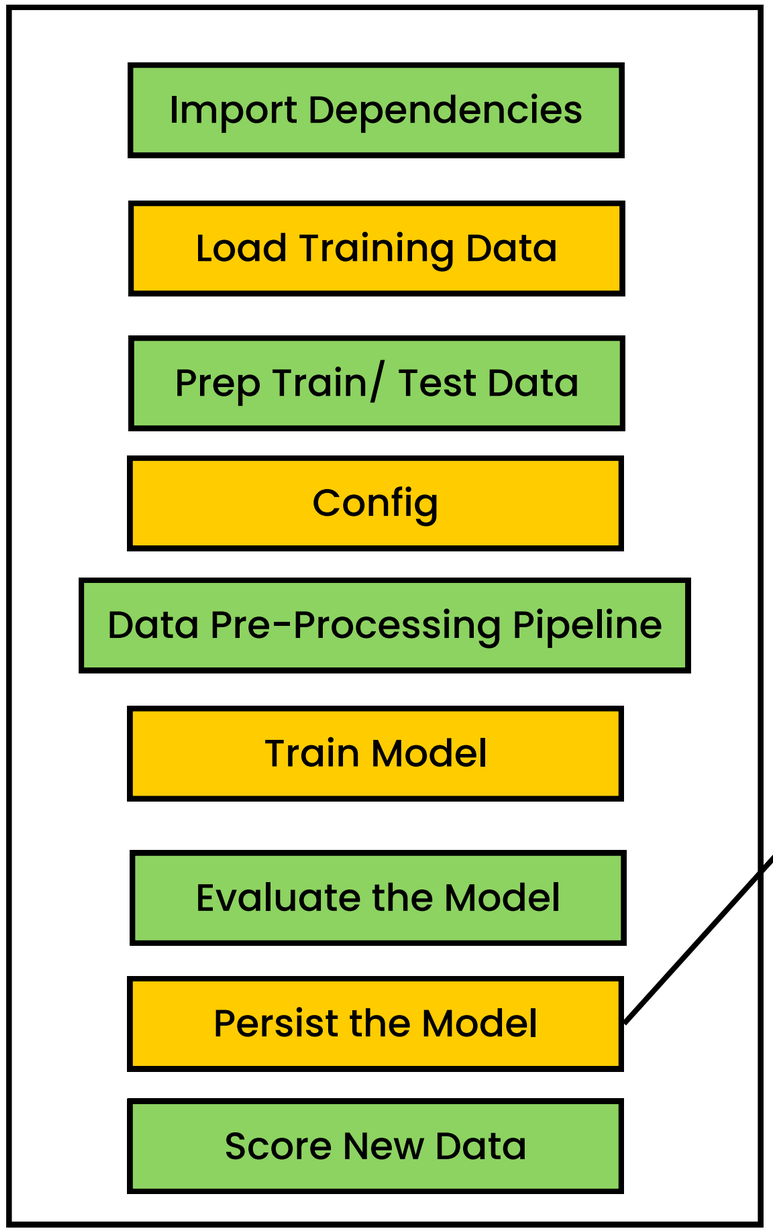
```
# train the pipeline
price_pipe.fit(X_train, y_train)
```

StatusNeo



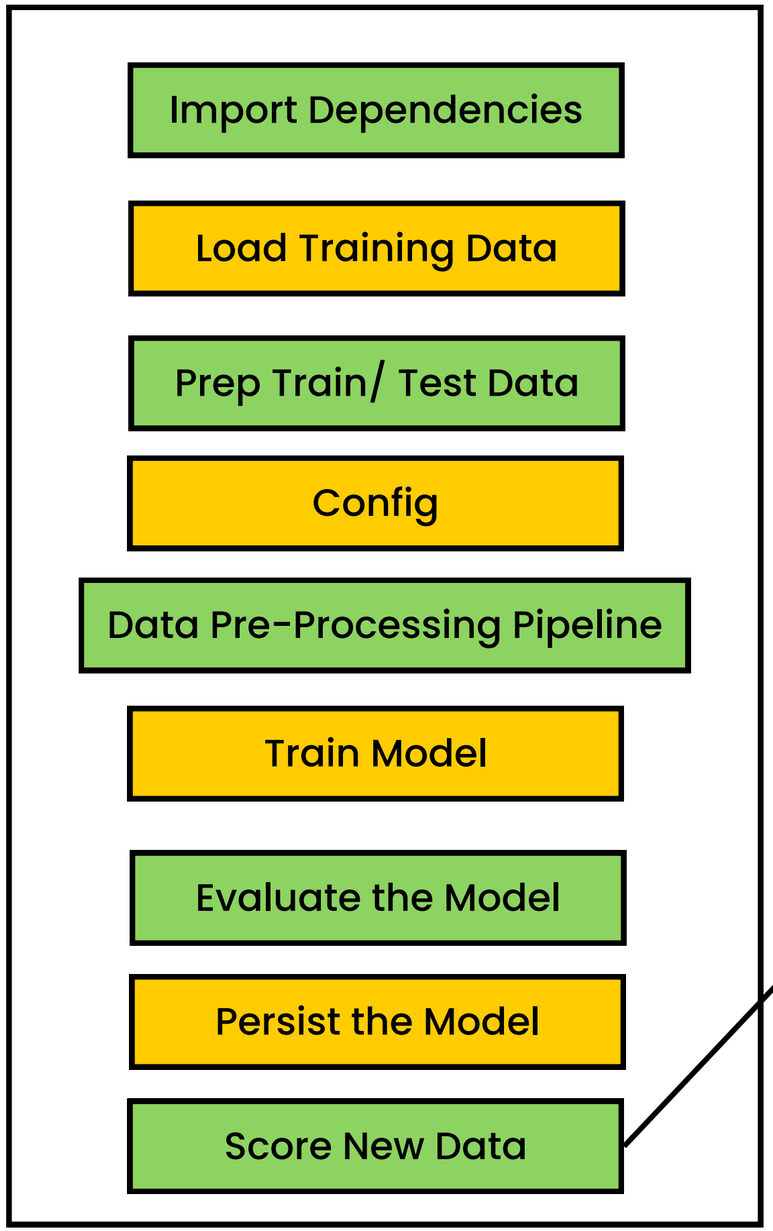
2]:

```
# evaluate the model:
# ====================

# make predictions for train set
pred = price_pipe.predict(X_train)

# determine mse, rmse and r2
print('train mse: {}'.format(int(
    mean_squared_error(np.exp(y_train), np.exp(pred)))))
print('train rmse: {}'.format(int(
    mean_squared_error(np.exp(y_train), np.exp(pred), squared=False))))
print('train r2: {}'.format(
    r2_score(np.exp(y_train), np.exp(pred))))
print()

# make predictions for test set
pred = price_pipe.predict(X_test)

# determine mse, rmse and r2
print('test mse: {}'.format(int(
    mean_squared_error(np.exp(y_test), np.exp(pred)))))
print('test rmse: {}'.format(int(
    mean_squared_error(np.exp(y_test), np.exp(pred), squared=False))))
print('test r2: {}'.format(
    r2_score(np.exp(y_test), np.exp(pred))))
print()

print('Average house price: ', int(np.exp(y_train).median()))
```




```
# now let's save the scaler

joblib.dump(price_pipe, 'price_pipe.joblib')
```

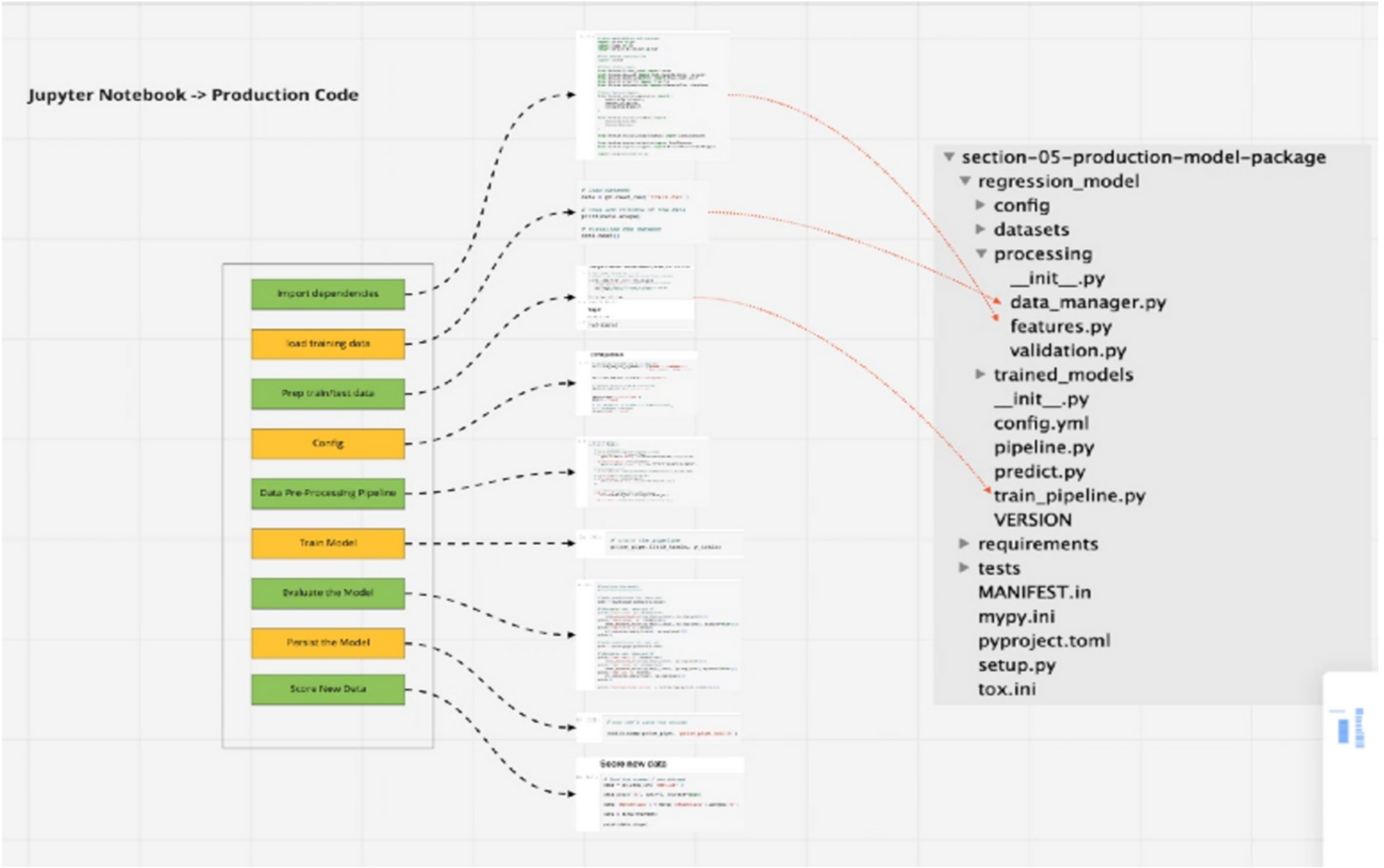
StatusNeo



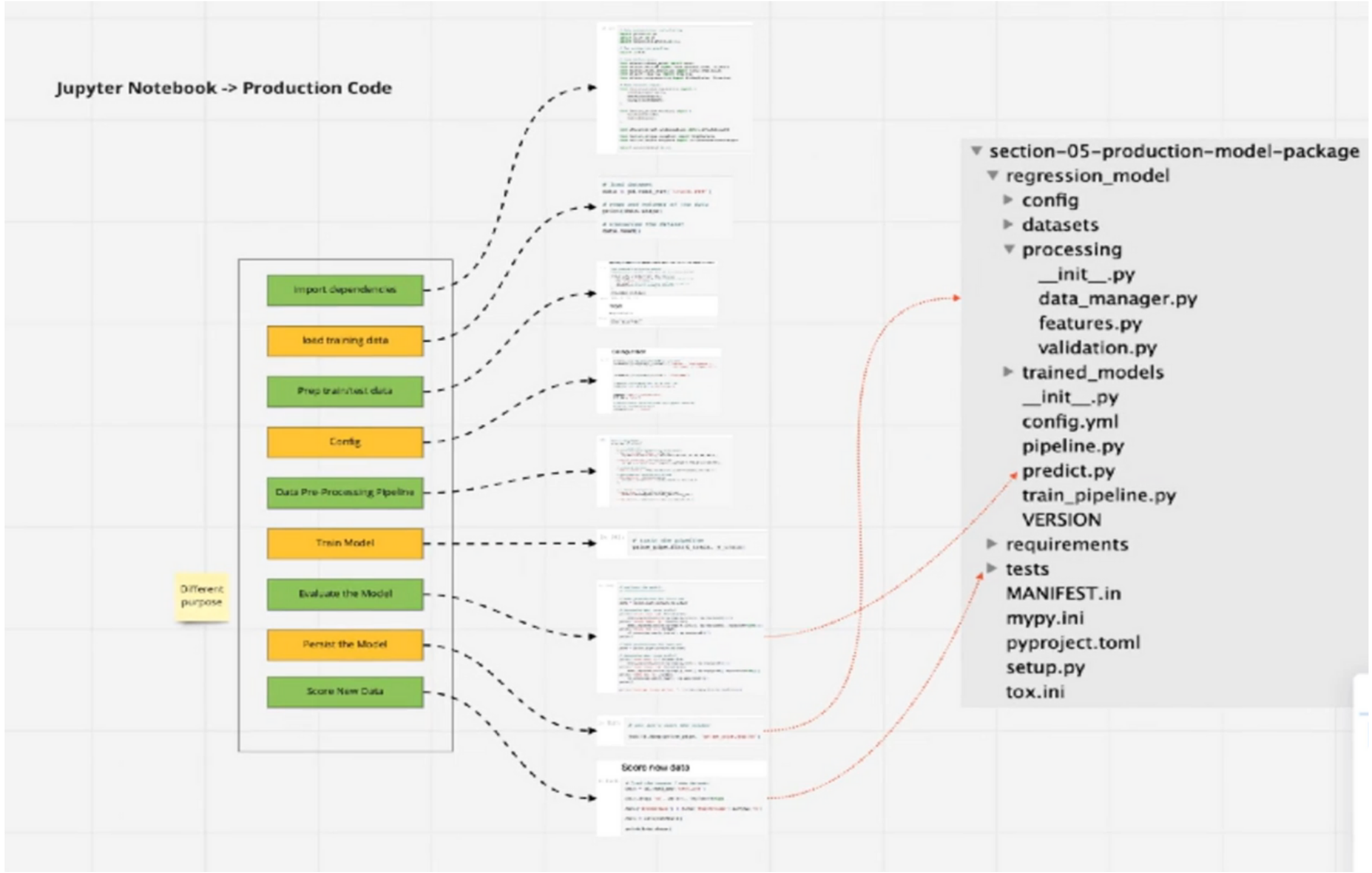
Score new data

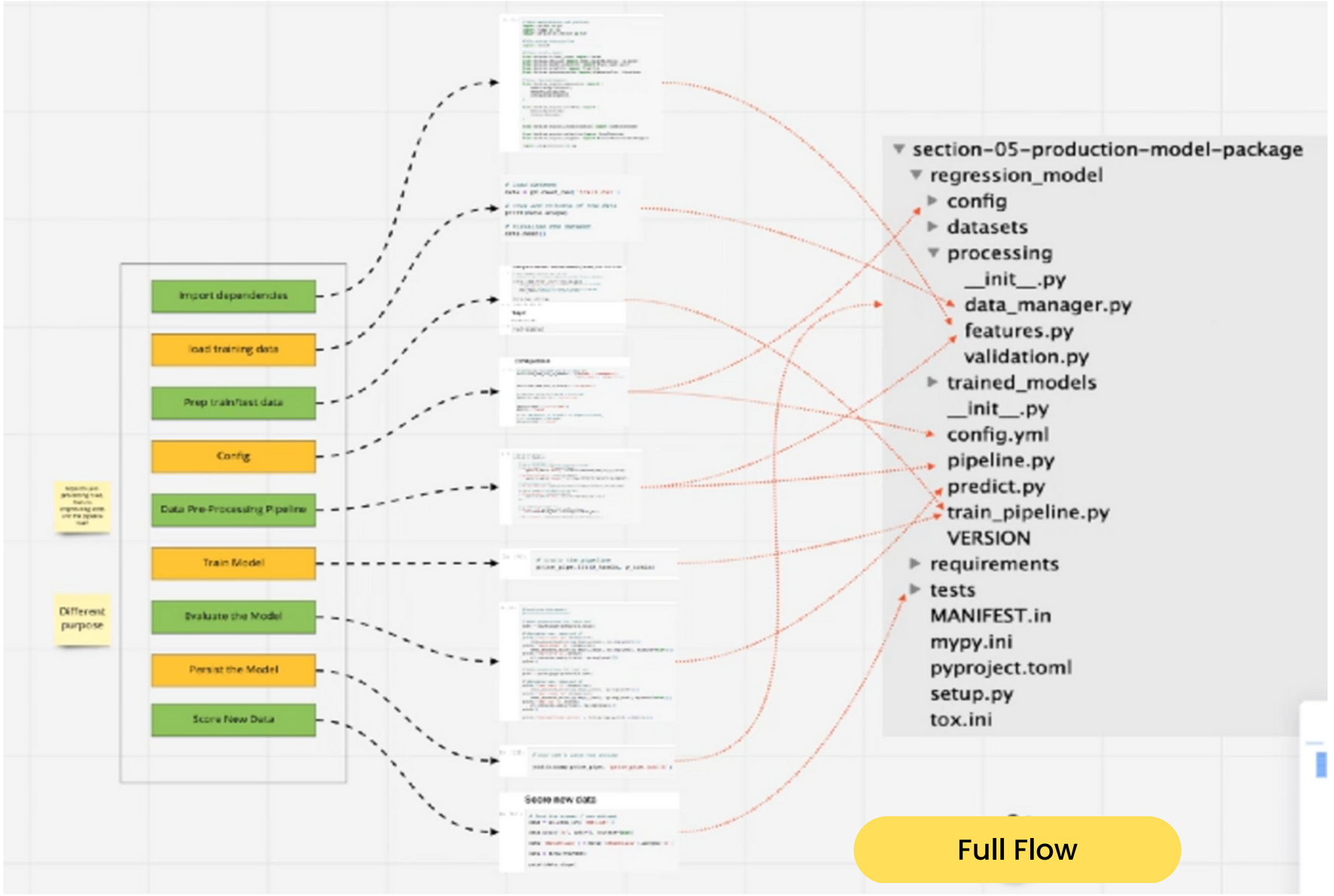
In [16]:

```
# load the unseen / new dataset
data = pd.read_csv('test.csv')

data.drop('Id', axis=1, inplace=True)

data['MSSubClass'] = data['MSSubClass'].astype('O')

data = data[FEATURES]

print(data.shape)
```


Jupyter Notebook -> Production Code
Import dependencies
load training data
Prep train/test data
Config
Data Pre-Processing Pipeline
Train Model
Evaluate the Model
Persist the Model
Score New Data
section-05-production-model-package
regression_model
config
datasets
processing
__init__.py
data_manager.py
features.py
validation.py
trained_models
__init__.py
config.yml
pipeline.py
predict.py
train_pipeline.py
VERSION
requirements
tests
MANIFEST.in
mypy.ini
pyproject.toml
setup.py
tox.ini


Jupyter Notebook -> Production Code
Import dependencies
load training data
Prep train/test data
Config
Data Pre-Processing Pipeline
Train Model
Evaluate the Model
Persist the Model
Score New Data
Score new data
section-05-production-model-package
regression_model
config
datasets
processing
__init__.py
data_manager.py
features.py
validation.py
trained_models
__init__.py
config.yml
pipeline.py
predict.py
train_pipeline.py
VERSION
requirements
tests
MANIFEST.in
mypy.ini
pyproject.toml
setup.py
tox.ini

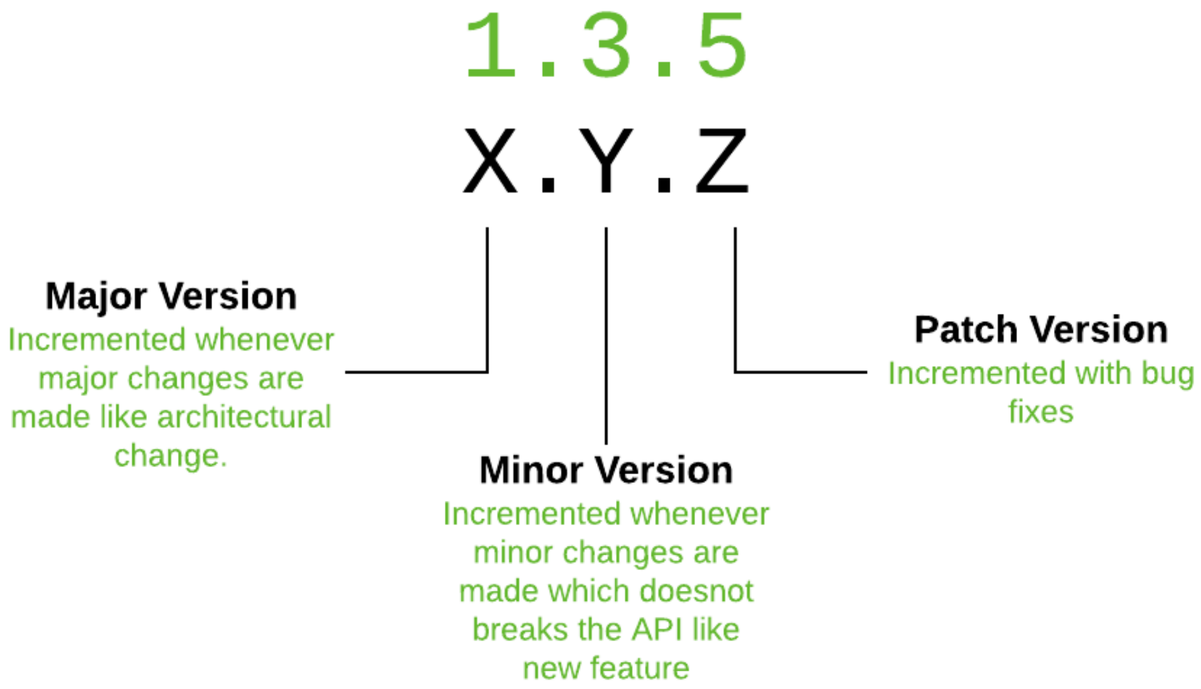

Jupyter Notebook -> Production Code
Import dependencies
load training data
Prep train/test data
Config
Data Pre-Processing Pipeline
Train Model
Evaluate the Model
Persist the Model
Score New Data
Different purpose
Score new data
section-05-production-model-package
regression_model
config
datasets
processing
__init__.py
data_manager.py
features.py
validation.py
trained_models
__init__.py
config.yml
pipeline.py
predict.py
train_pipeline.py
VERSION
requirements
tests
MANIFEST.in
mypy.ini
pyproject.toml
setup.py
tox.ini


Import dependencies
load training data
Prep train/test data
Config
Data Pre-Processing Pipeline
Train Model
Evaluate the Model
Persist the Model
Score New Data
Different purpose
section-05-production-model-package
regression_model
config
datasets
processing
__init__.py
data_manager.py
features.py
validation.py
trained_models
__init__.py
config.yml
pipeline.py
predict.py
train_pipeline.py
VERSION
requirements
tests
MANIFEST.in
mypy.ini
pyproject.toml
setup.py
tox.ini
Full Flow

Requirements File

StatusNeo

Semantic Versions



You have a major version followed by a period followed by the minor version, followed by a period followed

by what's known as a patch version.

And for a well maintained package, you'd expect that a minor version increment does not break the API and a major version increment is likely to break the API.

But you have to be a bit careful because some less well maintain packages may actually introduce breaking changes, even in a minor version bump.

Play it Conservatively!

```
4    numpy>=1.21.0,<2.0.0
5    pandas>=1.3.5,<2.0.0
6    pydantic>=1.8.1,<2.0.0
7    scikit-learn>=1.1.3,<2.0.0
8    strictyaml>=1.3.2,<2.0.0
9    ruamel.yaml>=0.16.12,<1.0.0
10   feature-engine>=1.0.2,<2.0.0
11   joblib>=1.0.1,<2.0.0
```

We're saying that anything for no greater than version one point to zero point zero, but it has to be less than version one point two one.
So we're not allowing any minor version increases here.
And it's up to you and your projects, how much risk you want to take.

StatusNeo

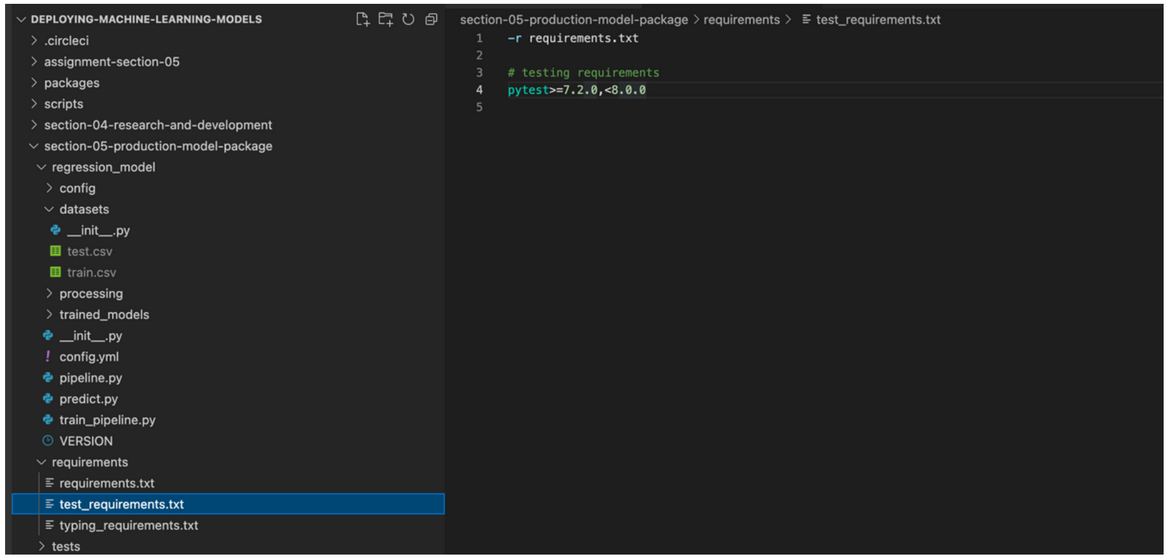
Play it Conservatively!

```
  [notice] To update, run: python3.10 -m pip install --upgrade pip
  ➜  section-05-production-model-package git:(master) pip3 install -r requirements/requirements.txt
➜  section-05-production-model-package git:(master) pip3 install -r requirements/requirements.txt
Collecting numpy<2.0.0,>=1.21.0
  Downloading numpy-1.24.2-cp310-cp310-macosx_10_9_x86_64.whl (19.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 19.8/19.8 MB 2.0 MB/s eta 0:00:00
Collecting pandas<2.0.0,>=1.3.5
  Downloading pandas-1.5.3-cp310-cp310-macosx_10_9_x86_64.whl (12.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.0/12.0 MB 1.8 MB/s eta 0:00:00
Collecting pydantic<2.0.0,>=1.8.1
  Downloading pydantic-1.10.4-cp310-cp310-macosx_10_9_x86_64.whl (2.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.8/2.8 MB 2.4 MB/s eta 0:00:00
Collecting scikit-learn<2.0.0,>=1.1.3
  Downloading scikit_learn-1.2.1-cp310-cp310-macosx_10_9_x86_64.whl (9.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.1/9.1 MB 1.7 MB/s eta 0:00:00
Collecting strictyaml<2.0.0,>=1.3.2
  Downloading strictyaml-1.6.2.tar.gz (130 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 130.8/130.8 kB 1.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting ruamel.yaml<1.0.0,>=0.16.12
  Using cached ruamel.yaml-0.17.21-py3-none-any.whl (109 kB)
Collecting feature-engine<2.0.0,>=1.0.2
  Using cached feature_engine-1.5.2-py2.py3-none-any.whl (290 kB)
Collecting joblib<2.0.0,>=1.0.1
  Using cached joblib-1.2.0-py3-none-any.whl (297 kB)
Collecting pytz>=2020.1
  Using cached pytz-2022.7.1-py2.py3-none-any.whl (499 kB)
Collecting python-dateutil>=2.8.1
  Using cached python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
Collecting typing-extensions>=4.2.0
  Using cached typing_extensions-4.4.0-py3-none-any.whl (26 kB)
Collecting scipy>=1.3.2
  Downloading scipy-1.10.0-cp310-cp310-macosx_10_15_x86_64.whl (35.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 35.1/35.1 MB 1.2 MB/s eta 0:00:00
Collecting threadpoolctl>=2.0.0
```

We can install all of these requirements with the commands

- pip3 install -r requirements/requirements.txt

Another requirement file : test_requirements.txt



So it's a way of just capturing everything in another requirements file. And we've split the requirements into these two files because there will be scenarios where we don't actually need to install these test requirements because these are only required when we want to test our package or when we want to run style checks, linting and type checks.

Why it is important to define a version

- These two files are really important. If we don't define which versions of our dependencies we expect, then it can result in our package becoming very brittle and broken.
- The most basic error in a package would be that you didn't have a requirement to file.
- That would just mean that when somebody else tried to install and use the package, it would fail because it wouldn't have its necessary dependencies.
- A more likely scenario is that we neglect to include a version, and if there is no version specified, PIP is just going to assume you want the latest version of a particular dependency.
- And it may well be that that version has progressed and it's released new features, a new major version perhaps, and it's got breaking changes in the API.
- And that means that your package or our package in this case is going to break.So it's really important for us to define these versions.

What is Tox? How does it work?

As you can see, Tox is generic virtual environment management and test command line tool.

- So what that really means for our purposes here is that tox means we don't have to worry about different operating systems. We can run tox on Windows, macOS, Linux and get the same behavior across the platforms.
- We don't have to worry about things like setting up python paths, configuring environment variables.
- We do all of that stuff inside our tox ini file. So it's a really powerful way for us to be able to run tests of different versions of Python.

https://tox.wiki/en/latest/

If you go train model , currently you just see __init_.py file. Now headover to command prompt and install tox


```
➜  section-05-production-model-package git:(master) pip3 install tox
Collecting tox
  Downloading tox-4.4.5-py3-none-any.whl (148 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 148.8/148.8 kB 1.1 MB/s eta 0:00:00
Collecting chardet>=5.1
  Downloading chardet-5.1.0-py3-none-any.whl (199 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 199.1/199.1 kB 2.7 MB/s eta 0:00:00
Collecting filelock>=3.9
  Using cached filelock-3.9.0-py3-none-any.whl (9.7 kB)
Requirement already satisfied: packaging>=23 in /usr/local/lib/python3.10/site-packages (from tox) (23.0)
Collecting platformdirs>=2.6.2
  Downloading platformdirs-3.0.0-py3-none-any.whl (14 kB)
Collecting pyproject-api>=1.5
  Downloading pyproject_api-1.5.0-py3-none-any.whl (12 kB)
Collecting colorama>=0.4.6
  Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Collecting tomli>=2.0.1
  Downloading tomli-2.0.1-py3-none-any.whl (12 kB)
Collecting virtualenv>=20.17.1
  Downloading virtualenv-20.19.0-py3-none-any.whl (8.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.7/8.7 MB 3.6 MB/s eta 0:00:00
Collecting pluggy>=1
  Downloading pluggy-1.0.0-py2.py3-none-any.whl (13 kB)
Collecting cachetools>=5.3
  Downloading cachetools-5.3.0-py3-none-any.whl (9.3 kB)
Collecting distlib<1,>=0.3.6
  Using cached distlib-0.3.6-py2.py3-none-any.whl (468 kB)
Installing collected packages: distlib, tomli, pluggy, platformdirs, filelock, colorama, chardet, cachetools, virtualenv, pyproject-api, tox
Successfully installed cachetools-5.3.0 chardet-5.1.0 colorama-0.4.6 distlib-0.3.6 filelock-3.9.0 platformdirs-3.0.0 pluggy-1.0.0 pyproject-api-1.5.0 tomli-2.0.1 tox-4.4.5 virtua
lenv-20.19.0
```