UDACITY

Return to Classroom

DISCUSS ON STUDENT HUB

Predict Customer Churn with Clean Code

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Udacity Student

You have done an amazing job in this project and it will open a huge door in your future work. MLOps is a very important topic on every field of machine learning. This first step is an warm up so you can deep dive into MLOps tools and knowledge!

As a reference, I would like to share some articles and resources to boost your learning:

Python Clean Code: 6 Best Practices to Make Your Python Functions More Readable

Tech Book Talk: Clean Code in Python

Introduction to MLOps

I really hope you enjoy the project as we mentors did and we are looking forward your next step!

Code Quality

All the code written for this project should follow the PEP 8 guidelines. Objects have meaningful names and syntax. Code is properly commented and organized. Imports are correctly ordered.

Running the below can assist with formatting.

```
autopep8 --in-place --aggressive --aggressive script.py
```

Then students should aim for a score exceeding 7 when using `pylint`

```
pylint script.py
```

Code Quality

PEP8 & Pylint

Awesome! 😄

In this part of your project, your code should follow the PEP 8 Guidelines and also using the Pylint Tool your code must achieve at least 7 points.

[✔] PEP8 Style
[✔] Pylint higher than 7

The file contains a summary of the purpose and description of the project. Someone should be able to run the code by reading the README.

Code Quality

README File

Your project contains a README.md file in order to summarize the purpose and description of the project. Moreover, Udacity has a very nice free course to guide you in README files creation.

Predict Customer Churn

This is my solution to the **Predict Customer Churn** Project, of the ML DevOps Engineer Nanodegree Udacity. This aims to:

- Write a ML pipeline to predict churn, using this dataset
- Make the code reusable, by following some code best practices.

Coding Best Practices

Employing some code organization and readability can greatly improve our quality of life while coding, making it more pleasant. The idea is that early directories/codes structuring will make us work more at the beginning of the project, but in the long run will cause us to spend significant less time reducing technical debt. Here is a short list of its benefits:

- automated unit/integration function testing allows faster debugging.
- self-explanatory code makes us quickly recap what was done and collaborate easier.
- automated pipelines provides faster/safer ml models retraining.

Project ML Pipeline Goal

The ML goal of this project is to produce a pipeline to automatically create machine learning models to predict churn. The task, along with the dataset used, is better described here.

The pipeline must automatically generate:

- basic EDA plots (available in `images/eda/`)
- model scoring object (available in `models/`)
- model performance plots (available in `images/results/`)
- model's variable importance plots (available in `images/results/`)
- model's interpretability SHAP values plot (available in `images/results/`)
- pipeline sucess/errors/warning logging (available in `logs/`)

Running Files

Create a Dedicate Env [Optional]

Before running the files, it is recommended to create a dedicated environment to run the files. To do this you can:

All functions have a document string that correctly identifies the inputs, outputs, and purpose of the function. All files have a document string that identifies the purpose of the file, the author, and the date the file was created.

Code Quality

Docstrings

Perfect! 😉

Your project is very clean and all functions have the docstring in order to indentify the inputs, outputs and the purpose of the given function.

I would like to share more content about docstrings since it can be a huge difference for code quality:

PEP 257 -- Docstring Conventions
Python Docstrings

Testing & Logging

Each function in `churn_script_logging_and_tests.py` is complete with tests for the input function.

Testing & Logging

`churn_script_logging_and_tests.py` Tests

Great job! 😄

Your scripts run tests in order to achieve the task and all tests pass!

Each function in `churn_script_logging_and_tests.py` is complete with logging for if the function successfully passes the tests or errors.

Testing & Logging

`churn_script_logging_and_tests.py` Logging

Great! 😉

Every function in your python script has the logging statement

All log information should be stored in a `.log` file, so it can be viewed post the run of the script.

Testing & Logging

`.log`

Great job logging all information in your `.log` file

I would like to share some resources for better understanding the `logging best practices in Python`

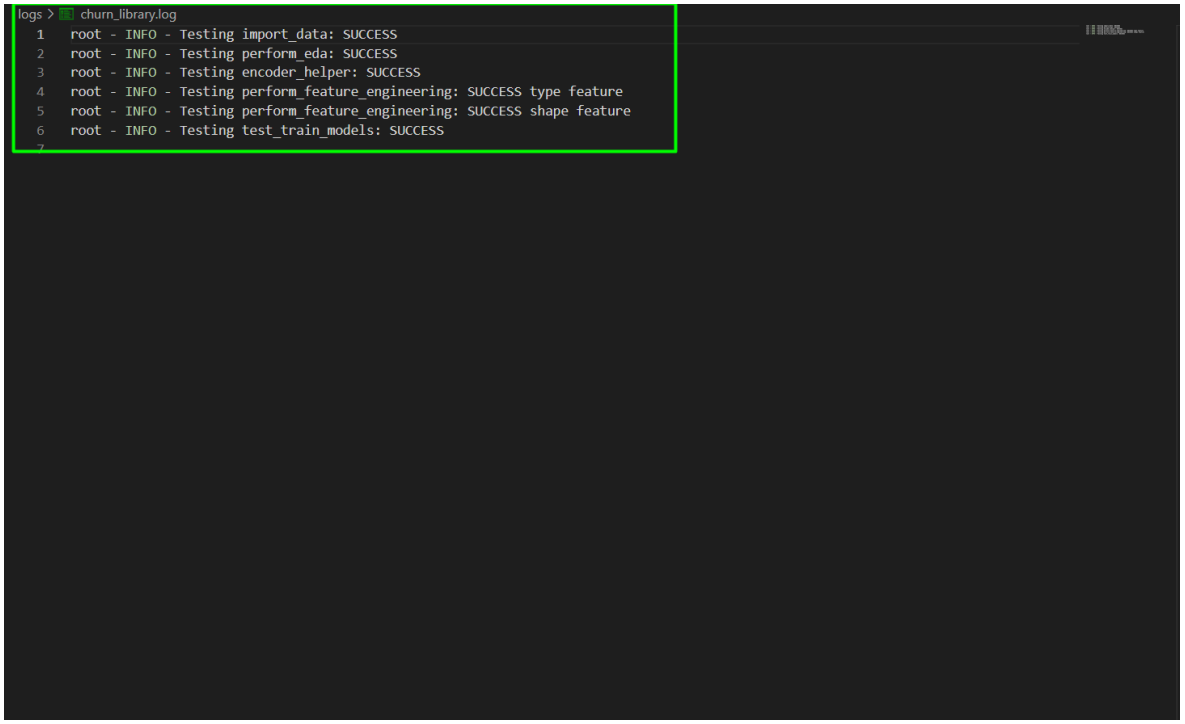
Logging
6 Python Logging Best Practices You Should Be Aware Of

The log messages should easily be understood and traceable that appear in the `.log` file.

Testing & Logging

`.log` understanding

The logs you have written are completely understandable.



The README should inform a user how they would test and log the result of each function.

Something similar to the below should produce the `.log` file with the result from running all tests.

```
ipython churn_script_logging_and_tests_solution.py
```

Testing & Logging

README.md Information about Tests

Great job! 😄

You have add to your README how to run the tests.

Save Images & Models

Store result plots including at least one:

1. Univariate, quantitative plot
2. Univariate, categorical plot
3. Bivariate plot

Save Images & Models

Result plots

Perfect 😄

The Result Plots are implemented and stored in the correct folders.

Store result plots including:

1. ROC curves
2. Feature Importances

Save Images & Models

Result plots - ROC Curve

Perfect 😄

The ROC Curve and the Feature Importance plots are implemented and stored in the correct folders.

As a suggestion, you can also use the ROC curve to improve your project

Store at least two models. Recommended using `joblib` and storing models with `.pkl` extension.

Save Images & Models

Models

Great 😄

The models are stored using the `.pkl` extension and using the recommended `joblib` library.

Problem Solving

Code in `churn_library.py` completes the process for solving the data science process including:

1. EDA

2. Feature Engineering (including encoding of categorical variables)
3. Model Training
4. Prediction
5. Model Evaluation

Problem Solving

Code

You code is complete and all functions implemented.

Use one-hot encoding or mean of the response to fill in categorical columns. Currently, the notebook does this in an inefficient way that can be refactored by looping. Make this code more efficient using the same method as in the notebook or using one-hot encoding. Tip: Creating a list of categorical column names can help with looping through these items and create an easier way to extend this logic.

Problem Solving

One-Hot-Encoding

You have implemented correctly the encoding of the categorical features.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

START