Week 5: Cloud and API deployment

Name: Sebastián J. Castro

Batch code: LISP01

Submissionn date: 13/05/2021

Submitted to: Data Glacier

What tools will I use in this task?

1. Pycaret:
   Pycaret is an open-source, low-code machine learning library and end-to-end model management tool built-in Python for automating machine learning workflows. It is easy to use, simple, and it enable ML model deployment quickly and efficiently.
2. FastAPI:
   FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints. It is fast (very high performance), one of the fastest python frameworks available, indeed. It is fast and easy to code.

Steps:

For this task, I will be using a very popular case of study by Darden School of Business, published in Harvard Business. The case is regarding the story of two people who are going to be married in the future. The guy named *Greg* wanted to buy a ring to propose to a girl named *Sarah*. The problem is to find the ring Sarah will like, but after a suggestion from his close friend, Greg decides to buy a diamond stone instead so that Sarah can decide her choice. Greg then collects data of 6000 diamonds with their price and attributes like cut, color, shape, etc.

1. Importing dataset:

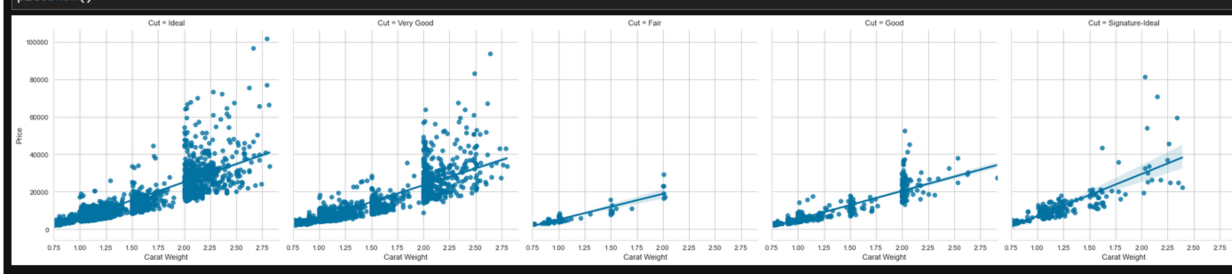
```python
# load the dataset from pycaret
from pycaret.datasets import get_data
data = get_data('diamond')
```

	Carat Weight	Cut	Color	Clarity	Polish	Symmetry	Report	Price
0	1.10	Ideal	H	SI1	VG	EX	GIA	5169
1	0.83	Ideal	H	VS1	ID	ID	AGSL	3470
2	0.85	Ideal	H	SI1	EX	EX	GIA	3183
3	0.91	Ideal	E	SI1	VG	VG	GIA	4370
4	0.83	Ideal	G	SI1	EX	EX	GIA	3171

2. Quick visualization:

In this step we are looking for asses the relationship of independent features (weight, cut, color, clarity, etc.) with the target variable “Price”.

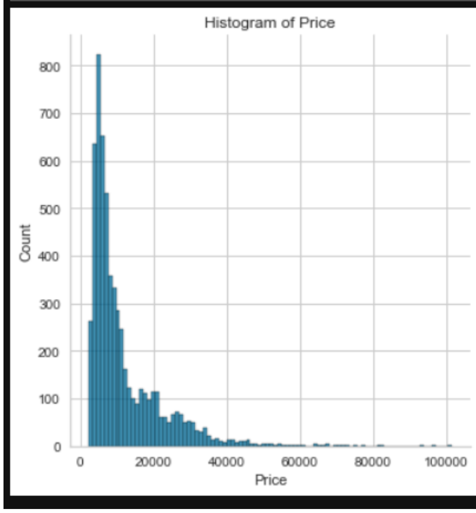
```python
sns.lmplot(x="Carat Weight", y="Price", data = data, col="Cut")
plt.show()
```



3. Target distribution:

Now we check the distribution of the target variable.

```python
# Target distribution
sns.displot(data=data, x="Price")
plt.title("Histogram of Price")
plt.show()
```

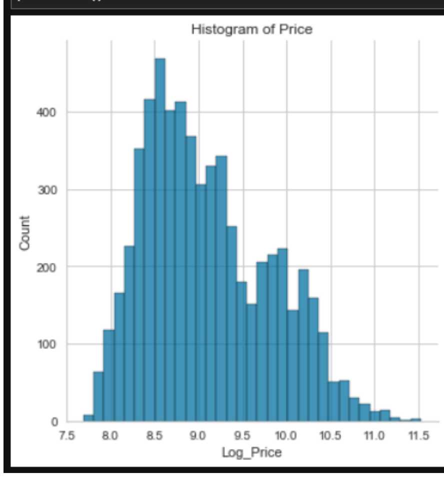


The distribution of “Price” is right-skewed, we can quickly check to see if log transformation can make “Price” approximately normal to give fighting chance to algorithms that assume normality.

4. Target variable log transformation

```python
# create a copy of data
data_copy = data.copy()# create a new feature Log_Price
data_copy['Log_Price'] = np.log(data['Price'])
```

```python
sns.displot(data=data_copy, x="Log_Price")
plt.title("Histogram of Price")
plt.show()
```



This confirms our hypothesis. The transformation will help us to get away with skewness and make the target variable approximately normal. Based on this, we will transform the “Price” variable before training our models.

5. Data preparation

Common to all modules in PyCaret, the “setup” is the first and the only mandatory step in any machine learning experiment performed in PyCaret. This function takes care of all the data preparation required prior to training models. Besides performing some basic default processing tasks, PyCaret also offers a wide array of pre-processing features.

```
# init setup
from pycaret.regression import *
s = setup(data, target = 'Price', transform_target = True)
```

Processing:

Initiated	. . . . 16:47:21
Status	. . . . Preprocessing Data

Following data types have been inferred automatically, if they are correct press enter to continue or type 'quit' otherwise.

	Data Type
Carat Weight	Numeric
Cut	Categorical
Color	Categorical
Clarity	Categorical
Polish	Categorical
Symmetry	Categorical
Report	Categorical
Price	Label

Pycaret setup function infers data types. Besides, Pycaret will transform the “Price” variable behind the scene using box-cox transformation as transform_target = True. It will affect the distribution of the target in a similar way as log transformation.

6. Model Training and Selection

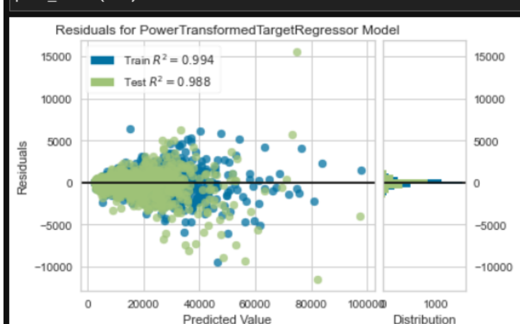
Now that data preparation is done, we can start the training process by using compare_models functionality. This function trains all the algorithms available in the model library and evaluates multiple performance metrics using cross-validation.

```
# compare all models
best = compare_models()
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
catboost	CatBoost Regressor	577.2807	1689718.6510	1243.0357	0.9836	0.0620	0.0449	0.3810
lightgbm	Light Gradient Boosting Machine	616.5210	1762471.0746	1286.2445	0.9828	0.0657	0.0482	0.0470
xgboost	Extreme Gradient Boosting	656.8723	1887969.3812	1338.9884	0.9815	0.0694	0.0511	0.1880
rf	Random Forest Regressor	712.2956	2249659.9077	1453.3809	0.9778	0.0779	0.0566	0.1450
et	Extra Trees Regressor	735.9206	2336103.2472	1500.2567	0.9767	0.0805	0.0585	0.1800
gbr	Gradient Boosting Regressor	744.8629	2789573.2805	1587.4840	0.9734	0.0773	0.0571	0.0440
dt	Decision Tree Regressor	919.3571	3495951.6414	1829.2282	0.9654	0.1004	0.0726	0.0080
ada	AdaBoost Regressor	1988.8263	16247554.5986	3971.2705	0.8390	0.1905	0.1539	0.0410
knn	K Neighbors Regressor	3017.6116	35507608.3300	5923.2306	0.6418	0.3637	0.2312	0.0150
omp	Orthogonal Matching Pursuit	3301.8823	86173413.9762	9028.8587	0.1109	0.2838	0.2233	0.0050
llar	Lasso Least Angle Regression	6460.3561	111534735.0643	10534.2799	-0.1248	0.7092	0.5591	0.0050
en	Elastic Net	6460.3561	111534748.0000	10534.2806	-0.1248	0.7092	0.5591	0.0050
lasso	Lasso Regression	6460.3561	111534748.0000	10534.2806	-0.1248	0.7092	0.5591	0.0050
ridge	Ridge Regression	3446.3770	495479510.8000	19279.1864	-3.9195	0.2264	0.1758	0.0050
br	Bayesian Ridge	3507.1561	562639285.1005	20305.8243	-4.4736	0.2276	0.1766	0.0060
huber	Huber Regressor	3489.5988	590414929.4700	20445.2880	-4.6708	0.2288	0.1744	0.0240
lar	Least Angle Regression	3582.3456	672966307.7370	21609.7413	-5.3680	0.2289	0.1776	0.0060
lr	Linear Regression	3579.5695	673320477.6000	21611.2203	-5.3712	0.2288	0.1774	0.0050
par	Passive Aggressive Regressor	17991.1792	360730908497.3094	274177.1203	-3009.6728	0.3568	0.4150	0.0060

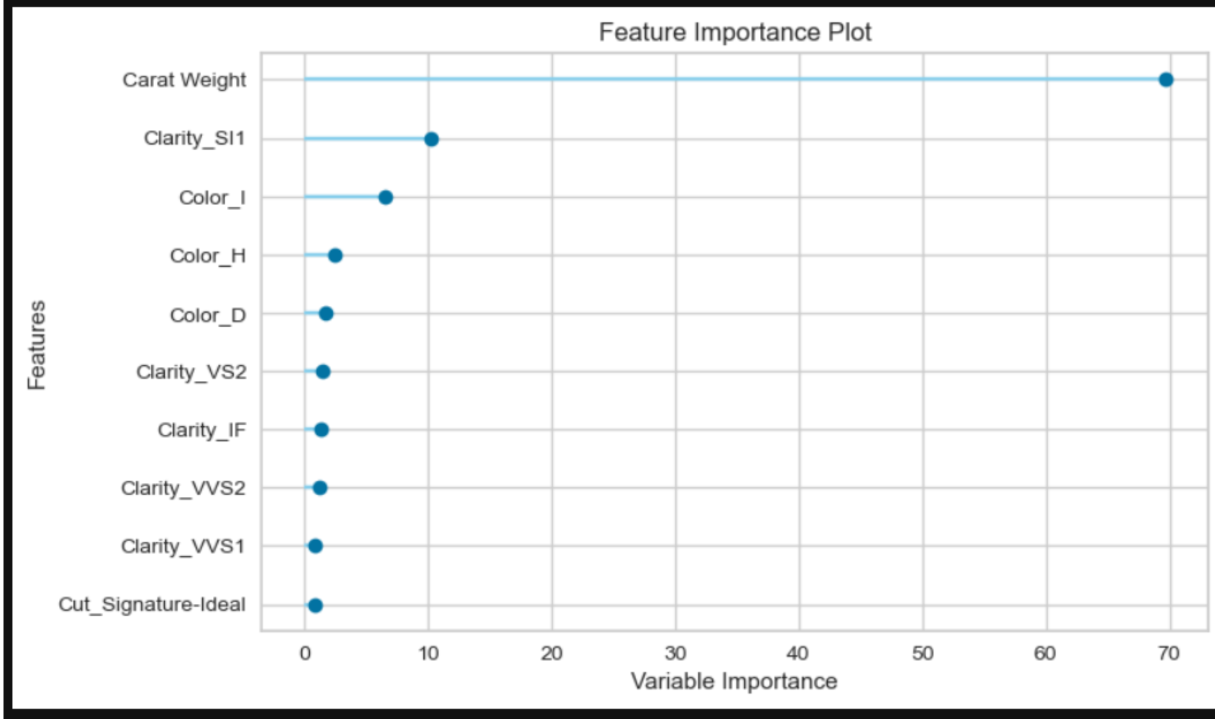
The best model based on *Mean Absolute Error (MAE)* is CatBoost Regressor. MAE using 10-fold cross-validation is $577 compared to the average diamond value of $11,600. This is less than 5%. Not bad for the efforts we have put in so far. Let's see the residuals of the trained model:

```
# check the residuals of trained model
plot_model(ctb)
```



We can also check the feature importance:

```
plot_model(ctb, plot = 'feature')
```



7. Finalize and Save Pipeline:

Let's now finalize the best model i.e. train the best model on the entire dataset including the test set and then save the pipeline as a pickle file.

```
# finalize the model
final_best = finalize_model(ctb)# save model to disk
save_model(final_best, 'diamond-pipeline')
```

```
Transformation Pipeline and Model Succesfully Saved
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True, features_todrop=[],
                                       id_columns=[], ml_usecase='regression',
                                       numerical_features=[], target='Price',
                                       time_features=[])),
                 ('imputer',
                  Simple_Imputer(categorical_strategy='not_available',
                                 fill_value_categorical=None,
                                 fill_value_numerical=None,
                                 numeric_strategy='...
                 ('feature_select', 'passthrough'), ('fix_multi', 'passthrough'),
                 ('dfs', 'passthrough'), ('pca', 'passthrough'),
                 ['trained_model',
                  PowerTransformedTargetRegressor(border_count=254,
                                                  loss_function='RMSE',
                                                  power_transformer_method='box-cox',
                                                  power_transformer_standardize=True,
                                                  random_state=5732,
                                                  regressor=<catboost.core.CatBoostRegressor object at 0x000001CC7F35BEE0>,
                                                  task_type='CPU',
                                                  verbose=False)]],
          verbose=False),
 'diamond-pipeline.pkl')
```

8. Deployment

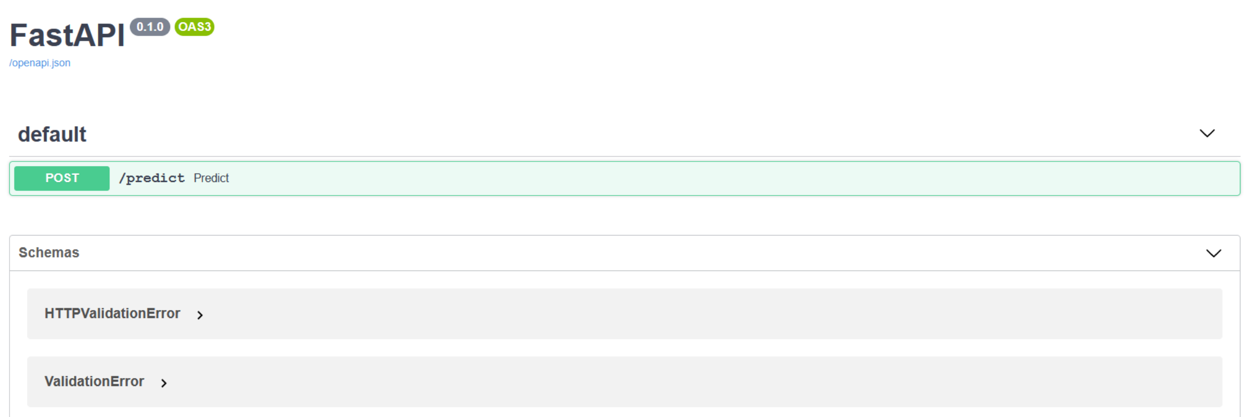
We will create an API using FastAPI. The main.py file necessary for doing that is the following:

```python
# 1. Library imports
import pandas as pd
from pycaret.regression import load_model, predict_model
from fastapi import FastAPI
import uvicorn

# 2. Create the app object
app = FastAPI()

#. Load trained Pipeline
model = load_model('diamond-pipeline')

# Define predict function
@app.post('/predict')
def predict(carat_weight, cut, color, clarity, polish, symmetry, report):
    data = pd.DataFrame([[carat_weight, cut, color, clarity, polish, symmetry, report]])
    data.columns = ['Carat Weight', 'Cut', 'Color', 'Clarity', 'Polish', 'Symmetry', 'Report']

    predictions = predict_model(model, data=data)
    return {'prediction': int(predictions['Label'][0])}

if __name__ == '__main__':
    uvicorn.run(app, host='127.0.0.1', port=8000)
```

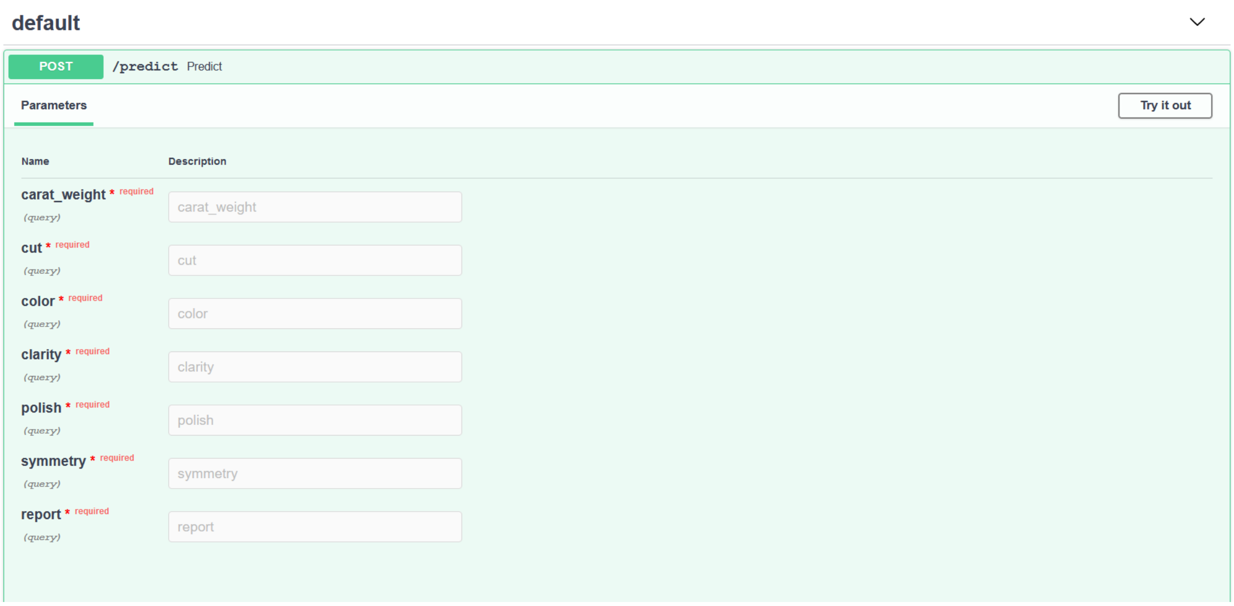
We then run this script by running the following command in the command prompt.

```
(DS) C:\Users\Admin\Documents\2021\DataGlacier\Proyectos\Deploy_API>uvicorn main:app --reload
```

This will initialize an API service on localhost. Then, we have to type http://localhost:8000/docs at a browser and it should show something like this:



Then we click on "POST" button and it will open a form like this:



Then, we click on "Try it out" and fill in some values in the form and click on "Execute".

Under the response body we have a prediction value of 23198 (this is based on values I entered in the form). This means that given all the attributes I have entered, the predicted price of this diamond is $23,198.

We can use the request library of Python to connect to API and generate predictions.

```
import requests

def get_predictions(carat_weight, cut, color, clarity, polish, symmetry, report):
    url = 'http://localhost:8000/predict?carat_weight={carat_weight}&cut={cut}&color={color}&clarity={clarity}&polish={polish}&symmetry={symmetry}&report={report
    .format(carat_weight = carat_weight, cut = cut,\
     color = color, clarity = clarity, polish = polish, symmetry = symmetry, report = report)

    x = requests.post(url)
    print(x.text)
```

```
get_predictions(2.2, "Ideal", "E", "SI1", "VG", "ID", "GIA")
```

```
{"prediction":23198}
```