DS Automation Assignment

Using our prepared churn data from week 2:

- use pycaret to find an ML algorithm that performs best on the data
  - Choose a metric you think is best to use for finding the best model; by default, it is accuracy but it could be AUC, precision, recall, etc. The week 3 FTE has some information on these different metrics.
- save the model to disk
- create a Python script/file/module with a function that takes a pandas dataframe as an input and returns the probability of churn for each row in the dataframe
  - your Python file/function should print out the predictions for new data (new_churn_data.csv)
  - the true values for the new data are [1, 0, 0, 1, 0] if you're interested
- test your Python module and function with the new data, new_churn_data.csv
- write a short summary of the process and results at the end of this notebook
- upload this Jupyter Notebook and Python file to a Github repository, and turn in a link to the repository in the week 5 assignment dropbox

Optional challenges:

- return the probability of churn for each new prediction, and the percentile where that prediction is in the distribution of probability predictions from the training dataset (e.g. a high probability of churn like 0.78 might be at the 90th percentile)
- use other autoML packages, such as TPOT, H2O, MLBox, etc, and compare performance and features with pycaret
- create a class in your Python module to hold the functions that you created
- accept user input to specify a file using a tool such as Python's `input()` function, the `click` package for command-line arguments, or a GUI
- Use the unmodified churn data (new_unmodified_churn_data.csv) in your Python script. This will require adding the same preprocessing steps from week 2 since this data is like the original unmodified dataset from week 1.

In [1]:
```
# import libraries
import pandas as pd
import os as os
from pycaret.classification import setup, compare_models, predict_model, save_model, load_model
```

In [2]:
```
# load data
df = pd.read_csv('./data/prepared_churn_data.csv', index_col='customerID')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7043 entries, 7590-VHVEG to 3186-AJIEK
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   tenure            7043 non-null   float64
 1   PhoneService      7043 non-null   int64  
 2   Contract          7043 non-null   int64  
 3   PaymentMethod     7043 non-null   int64  
 4   MonthlyCharges    7043 non-null   float64
 5   TotalCharges      7043 non-null   float64
 6   Churn             7043 non-null   int64  
 7   charge_per_tenure  7043 non-null   float64
dtypes: float64(4), int64(4)
memory usage: 495.2+ KB
```

In [3]:
```
# Setup pycaret, ignoring customerID
automl = setup(df, target='Churn', preprocess=False, ignore_features=['customerID'])
```

	Description	Value
0	Session id	4422
1	Target	Churn
2	Target type	Binary
3	Original data shape	(7043, 8)
4	Transformed data shape	(7043, 8)
5	Transformed train set shape	(4930, 8)
6	Transformed test set shape	(2113, 8)
7	Ignore features	1
8	Numeric features	7

In [4]:
```
# have pycaret compare models
best_model = compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lr	Logistic Regression	0.7927	0.8377	0.5183	0.6349	0.5702	0.4355	0.4397	0.2200
gbc	Gradient Boosting Classifier	0.7899	0.8376	0.5099	0.6294	0.5629	0.4266	0.4310	0.0530
lda	Linear Discriminant Analysis	0.7892	0.8264	0.5008	0.6307	0.5579	0.4219	0.4270	0.0040
ridge	Ridge Classifier	0.7888	0.8263	0.4480	0.6491	0.5295	0.3993	0.4110	0.0040
ada	Ada Boost Classifier	0.7878	0.8368	0.4947	0.6285	0.5525	0.4163	0.4220	0.0190
lightgbm	Light Gradient Boosting Machine	0.7844	0.8289	0.5084	0.6146	0.5559	0.4152	0.4189	0.3040
rf	Random Forest Classifier	0.7736	0.8071	0.4854	0.5901	0.5319	0.3846	0.3883	0.0480
svm	SVM - Linear Kernel	0.7732	0.7446	0.3997	0.6162	0.4769	0.3428	0.3596	0.0050
et	Extra Trees Classifier	0.7677	0.7871	0.4953	0.5731	0.5307	0.3776	0.3798	0.0340
knn	K Neighbors Classifier	0.7623	0.7517	0.4442	0.5678	0.4974	0.3450	0.3500	0.0650
qda	Quadratic Discriminant Analysis	0.7467	0.8227	0.7454	0.5165	0.6099	0.4316	0.4476	0.0040
nb	Naive Bayes	0.7436	0.8190	0.7270	0.5128	0.6012	0.4207	0.4347	0.0040
dummy	Dummy Classifier	0.7347	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0040
dt	Decision Tree Classifier	0.7345	0.6648	0.4977	0.5007	0.4989	0.3184	0.3186	0.0050

In [5]:
```
# save best model using the class name as the filename... this will make subsequent trials touch-free
model_name = best_model.__class__.__name__
save_model(best_model, model_name)
```

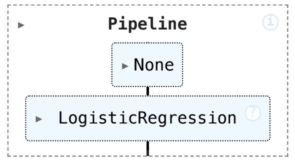
```
Transformation Pipeline and Model Successfully Saved
```

Out[5]:
```
(Pipeline(memory=Memory(location=None),
          steps=[('placeholder', None),
                 ('trained_model',
                  LogisticRegression(C=1.0, class_weight=None, dual=False,
                                     fit_intercept=True, intercept_scaling=1,
                                     l1_ratio=None, max_iter=1000,
                                     multi_class='auto', n_jobs=None,
                                     penalty='l2', random_state=4422,
                                     solver='lbfgs', tol=0.0001, verbose=0,
                                     warm_start=False))],
          verbose=False),
 'LogisticRegression.pkl')
```

In [6]:
```
# load the pickled model
loaded_model =load_model(model_name)
loaded_model
```

```
Transformation Pipeline and Model Successfully Loaded
```

Out[6]:



In [7]:
```python
# test the loaded model by making a prediction on unseen data
unseen_data = df.copy()
unseen_data = unseen_data.drop('Churn', axis=1)
predict_model(loaded_model, data=unseen_data.iloc[-2:-1])
```

Out[7]:

	tenure	PhoneService	Contract	PaymentMethod	MonthlyCharges	TotalCharges	charge_per_tenure	prediction_label	prediction_score
customerID									
8361-LTMKD	4.0	1	0	2	74.400002	306.600006	76.650002	1	0.5698

In [8]:
```python
# vefify the prediction
df.iloc[-2:-1]
```

Out[8]:

	tenure	PhoneService	Contract	PaymentMethod	MonthlyCharges	TotalCharges	Churn	charge_per_tenure
customerID								
8361-LTMKD	4.0	1	0	2	74.4	306.6	1	76.65

In [9]:
```python
# load libraries from python file
from predict_churn import make_prediction

# load unseen data from a prepared file
new_data = pd.read_csv('./data/new_churn_data.csv', index_col='customerID')

# make the prediction and print the result
print( make_prediction(new_data, model_name) )
```

```
Transformation Pipeline and Model Successfully Loaded
            Label   Score
customerID
9305-CKSKC      1  0.6192
1452-KNGVK      0  0.6934
6723-OKKJM      0  0.9309
7832-POPKP      0  0.7712
6348-TACGU      0  0.8076
```

Summary

In this exercise, we are using pycaret to automate machine learning tasks.

After loading the data, I initialized PyCaret with the 'setup' function. I passed it the training and test data, and the target variable. PyCaret has functions for preprocessing data, but I'm opting not to use them because we've already preprocessed the data in week 2. I also invoke the ignore_features parameter so that CustomerID was not evaluated in the models. In past exercises, I dropped that column before training my models... it's nice to have the ignore option.

I then have PyCaret evaluate and compare multiple models, sorting by accuracy (the default).

Each time we run compare_models, there's a chance that a new model will be chosen, so I'm using the model's class name as a variable. That way, we can run the notebook multiple times without editing parameters.

I test storing and loading the pickled models, and test the loaded model against unseen (no target) data.

Finally, we invoke a function 'make_prediction' from a Python file. I took the liberty of bending the assignment parameters a bit here, passing both our data and the model name to the function. Again, this is so that, no matter which model PyCaret recommends, we can run the notebook without editing parameters.

My results are a bit disappointing... not once have I returned a match against expected results. In the example submitted, expected results are [1,0,0,1,0], but I got a false negative in (zero-indexed) row 3 [1,0,0,0,0]