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 [23]:
```
import pandas as pd

df = pd.read_csv('D:/Cleaned_churn_data.csv', index_col='customerID')
df
```

Out[23]:

customerID	tenure	PhoneService	Contract	PaymentMethod	MonthlyCharges	TotalCharges	Churn
7590-VHVEG	1	0	0	2	29.85	29.85	0
5575-GNVDE	34	1	1	3	56.95	1889.50	0
3668-QPYBK	2	1	0	3	53.85	108.15	1
7795-CFOCW	45	0	1	0	42.30	1840.75	0
9237-HQITU	2	1	0	2	70.70	151.65	1
...	...	...	...	...	...	...	...
6840-RESVB	24	1	1	3	84.80	1990.50	0
2234-XADUH	72	1	1	1	103.20	7362.90	0
4801-JZAZL	11	0	0	2	29.60	346.45	0
8361-LTMKD	4	1	0	3	74.40	306.60	1
3186-AJIEK	66	1	2	0	105.65	6844.50	0

7043 rows × 7 columns

AutoML with pycaret

In [24]:
```
from pycaret.classification import *
```

In [25]:
```
automl = setup(df, target='Churn')
```

	Description	Value
0	Session id	8067
1	Target	Churn
2	Target type	Binary
3	Original data shape	(7043, 7)
4	Transformed data shape	(7043, 7)
5	Transformed train set shape	(4930, 7)
6	Transformed test set shape	(2113, 7)
7	Numeric features	6
8	Preprocess	True
9	Imputation type	simple
10	Numeric imputation	mean
11	Categorical imputation	mode
12	Fold Generator	StratifiedKFold
13	Fold Number	10
14	CPU Jobs	-1
15	Use GPU	False
16	Log Experiment	False
17	Experiment Name	clf-default-name
18	USI	207b

In [26]:
```
best_model = compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
gbc	Gradient Boosting Classifier	0.8004	0.8411	0.5176	0.6588	0.5790	0.4508	0.4568	0.1340
lr	Logistic Regression	0.7955	0.8344	0.5344	0.6370	0.5805	0.4469	0.4503	0.0600
ada	Ada Boost Classifier	0.7947	0.8401	0.4962	0.6502	0.5616	0.4309	0.4384	0.0560
ridge	Ridge Classifier	0.7917	0.0000	0.4527	0.6560	0.5349	0.4067	0.4187	0.0080
lda	Linear Discriminant Analysis	0.7844	0.8193	0.4901	0.6193	0.5461	0.4075	0.4129	0.0130
lightgbm	Light Gradient Boosting Machine	0.7803	0.8265	0.4978	0.6053	0.5457	0.4028	0.4064	0.2830
rf	Random Forest Classifier	0.7718	0.8003	0.4825	0.5851	0.5280	0.3796	0.3832	0.1080
et	Extra Trees Classifier	0.7617	0.7789	0.4963	0.5573	0.5240	0.3661	0.3677	0.1160
knn	K Neighbors Classifier	0.7604	0.7484	0.4343	0.5624	0.4894	0.3366	0.3417	0.3530
qda	Quadratic Discriminant Analysis	0.7507	0.8256	0.7470	0.5218	0.6139	0.4385	0.4544	0.0170
dummy	Dummy Classifier	0.7347	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0150
dt	Decision Tree Classifier	0.7211	0.6488	0.4817	0.4740	0.4775	0.2874	0.2876	0.0090
nb	Naive Bayes	0.7077	0.8083	0.7730	0.4695	0.5839	0.3788	0.4071	0.1900
svm	SVM - Linear Kernel	0.6998	0.0000	0.5767	0.5380	0.5027	0.3178	0.3521	0.0110

Finding Best Model

In [27]:
```
best_model
```

Out[27]:
```
▾                          GradientBoostingClassifier
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='log_loss', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_samples_leaf=1,
                           min_samples_split=2, min_weight_fraction_leaf=0.0,
                           n_estimators=100, n_iter_no_change=None,
                           random_state=8067, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

Saving and Loading Best Model

In [ ]:
```
save_model(best_model, 'GradientBoostingClassifier')
```

In [29]:
```
import pickle

with open('GradientBoostingClassifier.pk', 'wb') as f:
    pickle.dump(best_model, f)
```

In [30]:
```
with open('GradientBoostingClassifier.pk', 'rb') as f:
    loaded_model = pickle.load(f)
```

In [31]:
```
new_data = df.copy()
new_data.drop('Churn', axis=1, inplace=True)
loaded_model.predict(new_data)
```

Out[31]:
```
array([1, 0, 0, ..., 0, 1, 0], dtype=int8)
```

In [32]:
```
loaded_lda = load_model('GradientBoostingClassifier')
```

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

In [33]:
```
predict_model(loaded_lda, new_data)
```

Out[33]:

customerID	tenure	PhoneService	Contract	PaymentMethod	MonthlyCharges	TotalCharges	prediction_label	prediction_score
7590-VHVEG	1	0	0	2	29.850000	29.850000	1	0.5710
5575-GNVDE	34	1	1	3	56.950001	1889.500000	0	0.9476
3668-QPYBK	2	1	0	3	53.849998	108.150002	0	0.6439
7795-CFOCW	45	0	1	0	42.299999	1840.750000	0	0.9153
9237-HQITU	2	1	0	2	70.699997	151.649994	1	0.6443
...	...	...	...	...	...	...	...	...
6840-RESVB	24	1	1	3	84.800003	1990.500000	0	0.8993
2234-XADUH	72	1	1	1	103.199997	7362.899902	0	0.9136
4801-JZAZL	11	0	0	2	29.600000	346.450012	0	0.6951
8361-LTMKD	4	1	0	3	74.400002	306.600006	1	0.5393
3186-AJIEK	66	1	2	0	105.650002	6844.500000	0	0.9153

7043 rows × 8 columns

Predecting using python scripting

In [38]:
```
from IPython.display import Code

Code('D:/predict_churn.py')
```

Out[38]:
```
import pandas as pd
from pycaret.classification import predict_model, load_model

model = load_model('GradientBoostingClassifier')

def load_data(filepath):
    """
    Loads churn data into a DataFrame from a string filepath.
    """
    df = pd.read_csv(filepath, index_col='customerID')
    return df


def make_predictions(df, threshold=0.7):
    """
    Uses the pycaret best model to make predictions on data in the df dataframe.
    Rounds up to 1 if greater than or equal to the threshold.
    """
    predictions = predict_model(model, data=df)
    predictions['Churn_prediction'] = (predictions['prediction_score'] >= threshold)
    predictions['Churn_prediction'].replace({True: 'Churn', False: 'No churn'}, inplace=True)
    drop_cols = predictions.columns.tolist()
    drop_cols.remove('Churn_prediction')
    return predictions.drop(drop_cols, axis=1)


if __name__ == "__main__":
    df = load_data('D:/new_cleaned_churn_data.csv')
    predictions = make_predictions(df)
    print('predictions:')
    print(predictions)
```

In [39]:
```
%run D:/predict_churn.py
```

```
Transformation Pipeline and Model Successfully Loaded

predictions:
           Churn_prediction
customerID
9305-CKSKC         No churn
1452-KNGVK            Churn
6723-OKKJM            Churn
7832-POPKP         No churn
6348-TACGU            Churn
```

Summary

I want to figure out if customers will stop using our service. To do this, we use a tool called PyCaret that helps us with math stuff. First, we look at information about customers who have left before. Then, we use PyCaret to try different ways of guessing who will leave. We check how often our guesses are right and if we can tell the difference between customers who leave and those who don't. PyCaret tells us that the best way to guess is by using something called the Gradient Boosting Classifier.

Once we know the best way to guess, we save it on our computer. Then, we write a program to use this saved guess on new customers. The program reads information about new customers, uses the best guess we saved earlier, and tells us how likely each new customer is to leave.

To make sure our guess is good, we try it out on new customer information. The program tells us how likely each new customer is to leave. This helps us see if our guessing way is good enough to use in real life.

Using PyCaret makes it easy to find the best way to guess if customers will leave. It does the hard parts for us, so we can focus on understanding our information and making good choices based on it.