Assignment

Use the "from the expert" (FTE) jupyter notebook as a starter for this assignment, and ask your instructor questions if you need help.

Use our saved churn data from week 2 with machine learning to predict if customers will churn or not, similar to what we did in the FTE:

- break up data into features and targets
- split data into train and test sets
- use at least one ML model to fit to the training data
- evaluate performance on the train and test sets: at least evaluate accuracy and compare it with the "no information rate"
- plot a confusion matrix
- write something describing how the ML algorithm could be used in a business setting
- Write a short summary of what you did with the overall process - describe any important EDA findings, data cleaning and preparation, modeling, and evaluation in your summary.

Optional: For an addition challenge, try the following:

- fit more ML models and compare their scores
- optimize the hyperparameters of your models
- examine more metrics such as the classification report and ROC/AUC
- plot the distribution of the probability predictions (from the `predict_proba()` function from our model) for each class (1s and 0s)

DS process status

Here is our data science process, and where we are (#4):

1. Business understanding

Can we use machine learning to predict if a customer will churn before they leave?

2. Data understanding

Week 1 - EDA and visualization.

3. Data preparation

Last week - cleaning and feature engineering.

4. Modeling

This week. Fit a ML model to the data.

5. Evaluation

This week. Check the performance of our models and evaluate how it fits our goals from step 1.

6. Deployment

This week. Describe how the model might be deployed and used at the business. Will there be an API that customer service reps can use when customers call? Should there be a system where a report gets sent to someone in customer retention or marketing with at-risk customers? We should really think about these things in the first step, although we can consider them here this time.

In [4]:
```
#import warnings
#warnings.filterwarnings("ignore")
```

In [5]:
```
#import pandas as pd
#from sklearn.model_selection import train_test_split
#from sklearn.linear_model import LogisticRegression
```

In [6]:
```
df = pd.read_csv('../Week2/prepped_churn_data.csv', index_col='customerID')
df
```

Out[6]:

	tenure	PhoneService	Contract	PaymentMethod	MonthlyCharges	TotalCharg
customerID						
7590-VHVEG	1	0	0	0	29.85	29.
5575-GNVDE	34	1	1	1	56.95	1889.
3668-QPYBK	2	1	0	1	53.85	108.
7795-CFOCW	45	0	1	2	42.30	1840.
9237-HQITU	2	1	0	0	70.70	151.
...	...	...	...	...	...	
6840-RESVB	24	1	1	1	84.80	1990.
2234-XADUH	72	1	1	3	103.20	7362.
4801-JZAZL	11	0	0	0	29.60	346.
8361-LTMKD	4	1	0	1	74.40	306.
3186-AJIEK	66	1	2	2	105.65	6844.

7043 rows × 8 columns

In [7]:
```
df.head(10)
```

Out[7]:

customerID	tenure	PhoneService	Contract	PaymentMethod	MonthlyCharges	TotalCharg
7590-VHVEG	1	0	0	0	29.85	29.
5575-GNVDE	34	1	1	1	56.95	1889.
3668-QPYBK	2	1	0	1	53.85	108.
7795-CFOCW	45	0	1	2	42.30	1840.
9237-HQITU	2	1	0	0	70.70	151.
9305-CDSKC	8	1	0	0	99.65	820.
1452-KIOVK	22	1	0	3	89.10	1949.
6713-OKOMC	10	0	0	1	29.75	301.
7892-POOKP	28	1	0	0	104.80	3046.
6388-TABGU	62	1	1	2	56.15	3487.

In [8]:

```
df.tail(10)
```

Out[8]:

customerID	tenure	PhoneService	Contract	PaymentMethod	MonthlyCharges	TotalCharg
9767-FFLEM	38	1	0	3	69.50	2625.
0639-TSIQW	67	1	0	3	102.95	6886.
8456-QDAVC	19	1	0	2	78.70	1495.
7750-EYXWZ	12	0	1	0	60.65	743.
2569-WGERO	72	1	2	2	21.15	1419.
6840-RESVB	24	1	1	1	84.80	1990.
2234-XADUH	72	1	1	3	103.20	7362.
4801-JZAZL	11	0	0	0	29.60	346.
8361-LTMKD	4	1	0	1	74.40	306.
3186-AJIEK	66	1	2	2	105.65	6844.

In [9]:
```
df.sample(5)
```

Out[9]:

customerID	tenure	PhoneService	Contract	PaymentMethod	MonthlyCharges	TotalCharg
4729-XKASR	1	0	0	0	24.75	24.
0701-TJSEF	9	1	0	2	68.25	576.
5405-ZMYXQ	8	1	0	3	74.60	548.
1452-KIOVK	22	1	0	3	89.10	1949.
4573-JKNAE	12	1	2	2	19.35	212.

In [10]:
```
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   int64  
 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   tenure_charge_ratio 7043 non-null   float64
dtypes: float64(3), int64(5)
memory usage: 495.2+ KB
```

4. Modeling

In [35]:
```
features = df.drop('Churn', axis=1)
targets = df['Churn']
```

In [36]:
```
features.head()
```

Out[36]:

customerID	tenure	PhoneService	Contract	PaymentMethod	MonthlyCharges	TotalCharg
7590-VHVEG	1	0	0	0	29.85	29.
5575-GNVDE	34	1	1	1	56.95	1889.
3668-QPYBK	2	1	0	1	53.85	108.
7795-CFOCW	45	0	1	2	42.30	1840.
9237-HQITU	2	1	0	0	70.70	151.

In [37]:
```
targets.head()
```

Out[37]:
```
customerID
7590-VHVEG    0
5575-GNVDE    0
3668-QPYBK    1
7795-CFOCW    0
9237-HQITU    1
Name: Churn, dtype: int64
```

In [38]:
```
x_train, x_test, y_train, y_test = train_test_split(features, targets, random_state
```

In [39]:
```
x_train.shape
```

Out[39]:
```
(5282, 7)
```

In [40]:
```
x_test.shape
```

Out[40]:
```
(1761, 7)
```

In [41]:
```
y_train.shape
```

Out[41]:
```
(5282,)
```

In [42]:
```
x_train, x_test, y_train, y_test = train_test_split(features, targets, random_state
```

In [43]:
```
len(x_train)
```

Out[43]:
```
5282
```

In [44]:
```
len(x_test)
```

Out[44]:
```
1761
```

In [45]:
```
lr_model = LogisticRegression(max_iter=1000)
```

In [46]:
```
lr_model.fit(x_train, y_train)
```

Out[46]:
```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

5. Evaluation

In [47]:
```
df['Churn'].value_counts(normalize=True)
```

Out[47]:
```
Churn
0    0.73463
1    0.26537
Name: proportion, dtype: float64
```

our "no information" rate is 73.5%

In [48]:
```
print(lr_model.score(x_train, y_train))
print(lr_model.score(x_test, y_test))
```

```
0.7904202953426732
0.8018171493469619
```

Train and test are higher than no information rate accuracy

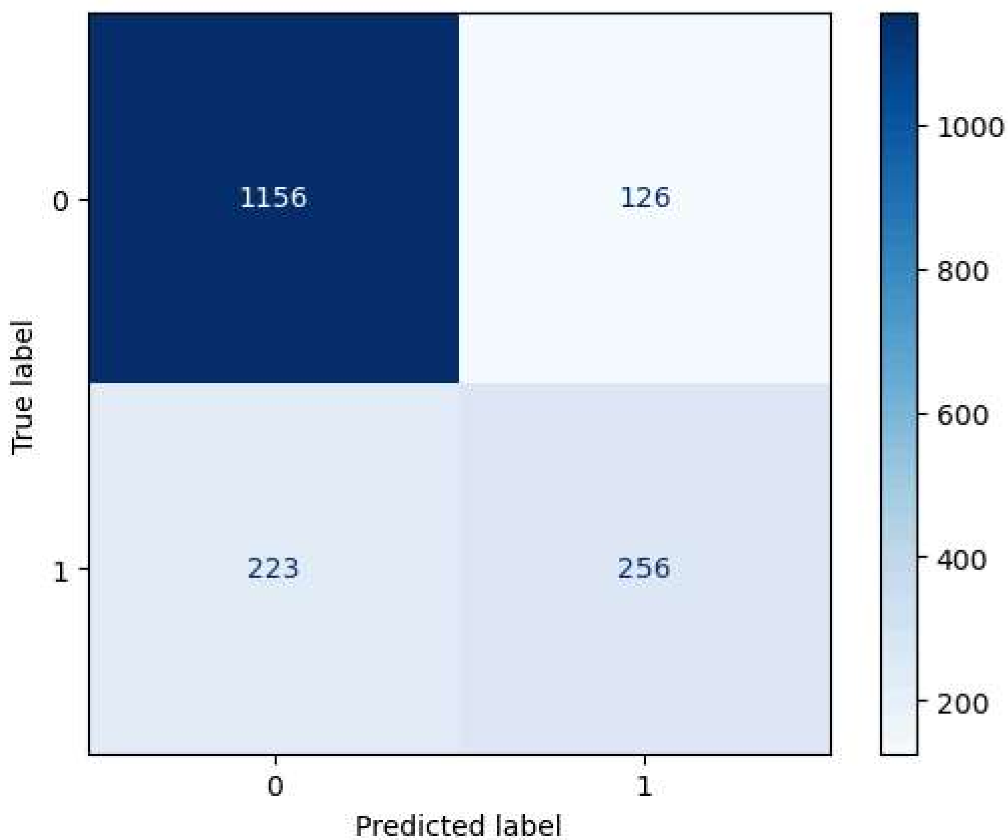
The test score is not very much lower than our training score, it's a sign we are not overfitting

In [50]:
```
# packages necessary for this block of code
#from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
#import matplotlib.pyplot as plt

#gather the predictions for our test dataset
predictions = lr_model.predict(x_test)

# construct the confusion matix - this retrns an array
cm = confusion_matrix(y_test, predictions, labels=lr_model.classes_)

# format and display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=lr_model.classes_
disp.plot(cmap=plt.cm.Blues)
plt.show()
```



Based on these results our true positive rate is 53%

In [51]:
```
lr_model.predict_proba(x_test)[:5]
```

Out[51]:
```
array([[0.48674824, 0.51325176],
       [0.94884374, 0.05115626],
       [0.99695269, 0.00304731],
       [0.35932228, 0.64067772],
       [0.99525148, 0.00474852]])
```

In [52]:
```
lr_model.predict(x_test)[:5]
```

Out[52]:
```
array([1, 0, 0, 1, 0], dtype=int64)
```

In [53]:
```
(lr_model.predict_proba(x_test)[:5, 1] > 0.19).astype('int')
```

Out[53]:
```
array([1, 0, 0, 1, 0])
```

In [54]:
```
predictions_lower_thresh = (lr_model.predict_proba(x_test)[:, 1] > 0.19).astype('in
predictions_lower_thresh
```

Out[54]:
```
array([1, 0, 0, ..., 0, 1, 0])
```

In [57]:
```
#from sklearn.metrics import accuracy_score, confusion_matrix
print(accuracy_score(y_test, predictions_lower_thresh))
tn, fp, fn, tp  = confusion_matrix(y_test, predictions_lower_thresh).flatten()
print(tp / (tp + fn))
```

```
0.6802952867688813
0.9123173277661796
```

By changing the threshold, our new true positive rate is 91%

In [58]:
```
lr_model.coef_
```

Out[58]:
```
array([[-5.23540731e-02, -6.11129428e-01, -1.13520182e+00,
        -1.97325042e-01,  2.17162359e-02,  2.77673956e-04,
        -9.66317001e-02]])
```

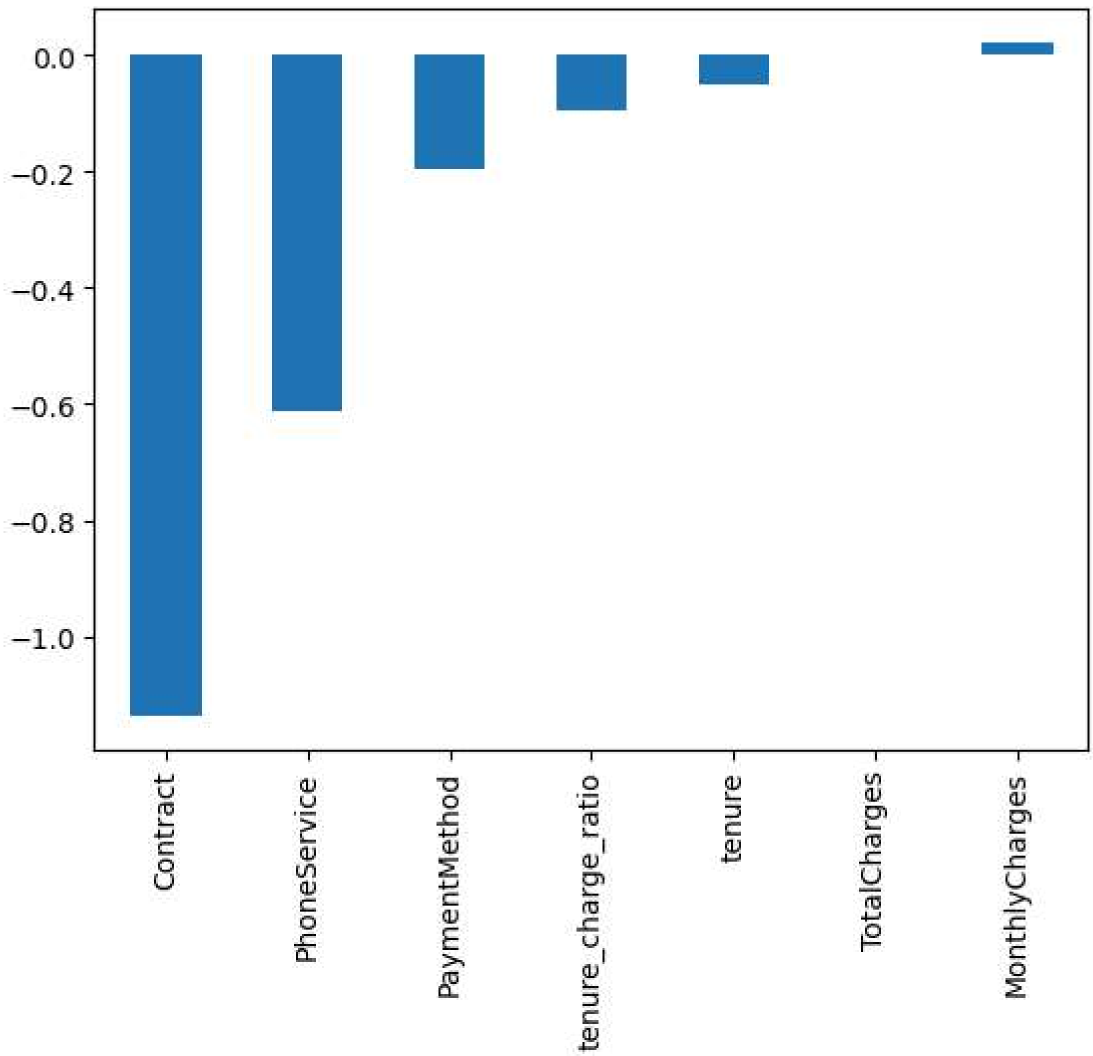
In [59]:
```
features.columns
```

Out[59]:
```
Index(['tenure', 'PhoneService', 'Contract', 'PaymentMethod', 'MonthlyCharges',
       'TotalCharges', 'tenure_charge_ratio'],
      dtype='object')
```

In [60]:
```
coef_df = pd.DataFrame(data=lr_model.coef_, columns=features.columns)
```

In [61]:
```
coef_df.T.sort_values(by=0).plot.bar(legend=False)
```

Out[61]:
```
<Axes: >
```



In [62]:
```
coef_df.T
```

Out[62]:

	0
tenure	-0.052354
PhoneService	-0.611129
Contract	-1.135202
PaymentMethod	-0.197325
MonthlyCharges	0.021716
TotalCharges	0.000278
tenure_charge_ratio	-0.096632

In [63]:
```
10**-1.14
```

Out[63]: 0.07244359600749903

In [64]:
```
10**0.02
```

Out[64]: 1.0471285480508996

6. Deployment

To deploy this ML algorith, we could create an API that could be used by software engineers to integrate it into software for collections or customer service reps. The reps could then use the software to predict the probability that a customer might churn based on the provided data.

Summary

Using customer data, we successfully deployed a machine learning model to predict churn. We kept data cleaning minimal, mainly converting categorical values to numeric ones. The Phi-K correlation showed that tenure had the strongest link to churn, with lower tenure values often indicating churn. Our logistic regression model achieved 91% accuracy on the test data, compared to the majority class fraction of 73.5%. As of this momment, the model performs well.