0.) Import and Clean data

In [1]:
```
import pandas as pd
from google.colab import drive
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import plot_tree
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

In [ ]:
```
#drive.mount('/content/gdrive/', force_remount = True)
```

```
Mounted at /content/gdrive/
```

In [4]:
```
df = pd.read_csv("bank-additional-full.csv", sep= ";")
```

In [5]:
```
df.head()
```

Out[5]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pday
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	99
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	99
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	99
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	99
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	99

5 rows × 21 columns

In [6]:
```
df = df.drop(["default", "pdays",        "previous",       "poutcome",       "emp.var.rate",
"cons.price.idx",         "cons.conf.idx",          "euribor3m",      "nr.employed"], axis =
1)
df = pd.get_dummies(df, columns = ["loan",
"job","marital","housing","contact","day_of_week", "campaign", "month",
"education"],drop_first = True)
```

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

Out[7]:

	age	duration	y	loan_unknown	loan_yes	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_reti
0	56	261	no	0	0	0	0	1	0	
1	57	149	no	0	0	0	0	0	0	
2	37	226	no	0	0	0	0	0	0	
3	40	151	no	0	0	0	0	0	0	
4	56	307	no	0	1	0	0	0	0	

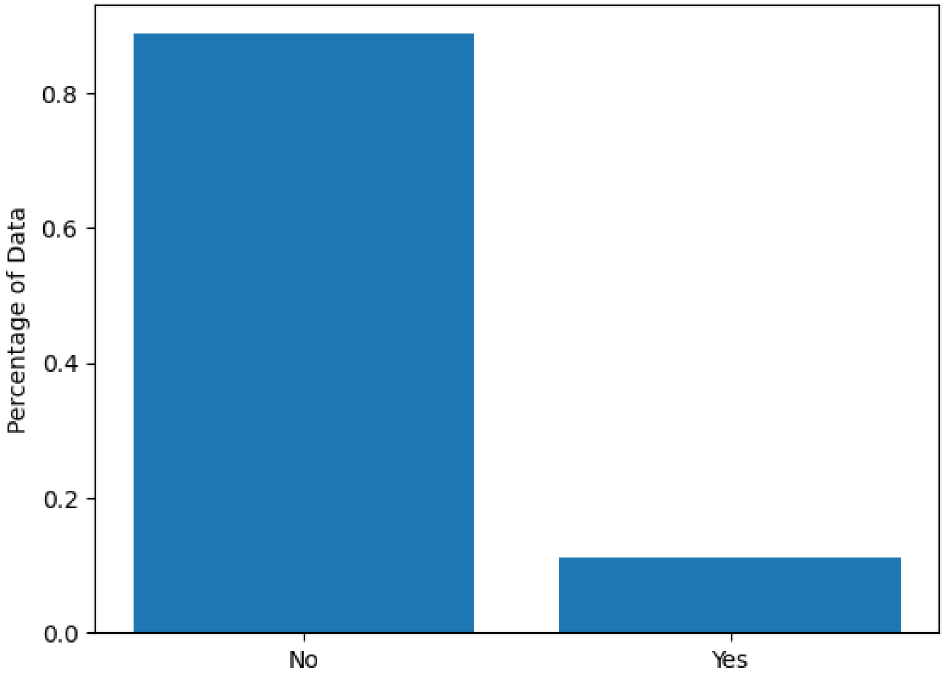
5 rows × 83 columns

In [8]:

```
y = pd.get_dummies(df["y"], drop_first = True)
X = df.drop(["y"], axis = 1)
```

In [9]:

```
obs = len(y)
plt.bar(["No","Yes"],[len(y[y.yes==0])/obs,len(y[y.yes==1])/obs])
plt.ylabel("Percentage of Data")
plt.show()
```



In [10]:

```
# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

scaler = StandardScaler().fit(X_train)

X_scaled = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

1.) Based on the visualization above, use your expert opinion to transform the data based on what we

learned this quarter

In [54]:

```
################
###TRANSFORM###
################


# Transform by SMOTE
from imblearn.over_sampling import SMOTE


smote = SMOTE()
smote_X, smote_y = smote.fit_resample(X_train, y_train)
```

In [55]:

```
X_scaled = smote_X
y_train = smote_y
X_scaled
y_train
```

Out[55]:

	yes
0	0
1	0
2	0
3	0
4	0
...	...
51155	1
51156	1
51157	1
51158	1
51159	1

51160 rows × 1 columns

In [58]:

```
len(X_scaled)
```

Out[58]:

```
51160
```

In [59]:

```
len(y_train)
```

Out[59]:

```
51160
```

2.) Build and visualize a decision tree of Max Depth 3. Show the confusion matrix.

In [26]:

```
dtree_main = DecisionTreeClassifier(max_depth = 3)
dtree_main.fit(X_scaled, y_train)
```

Out[26]:

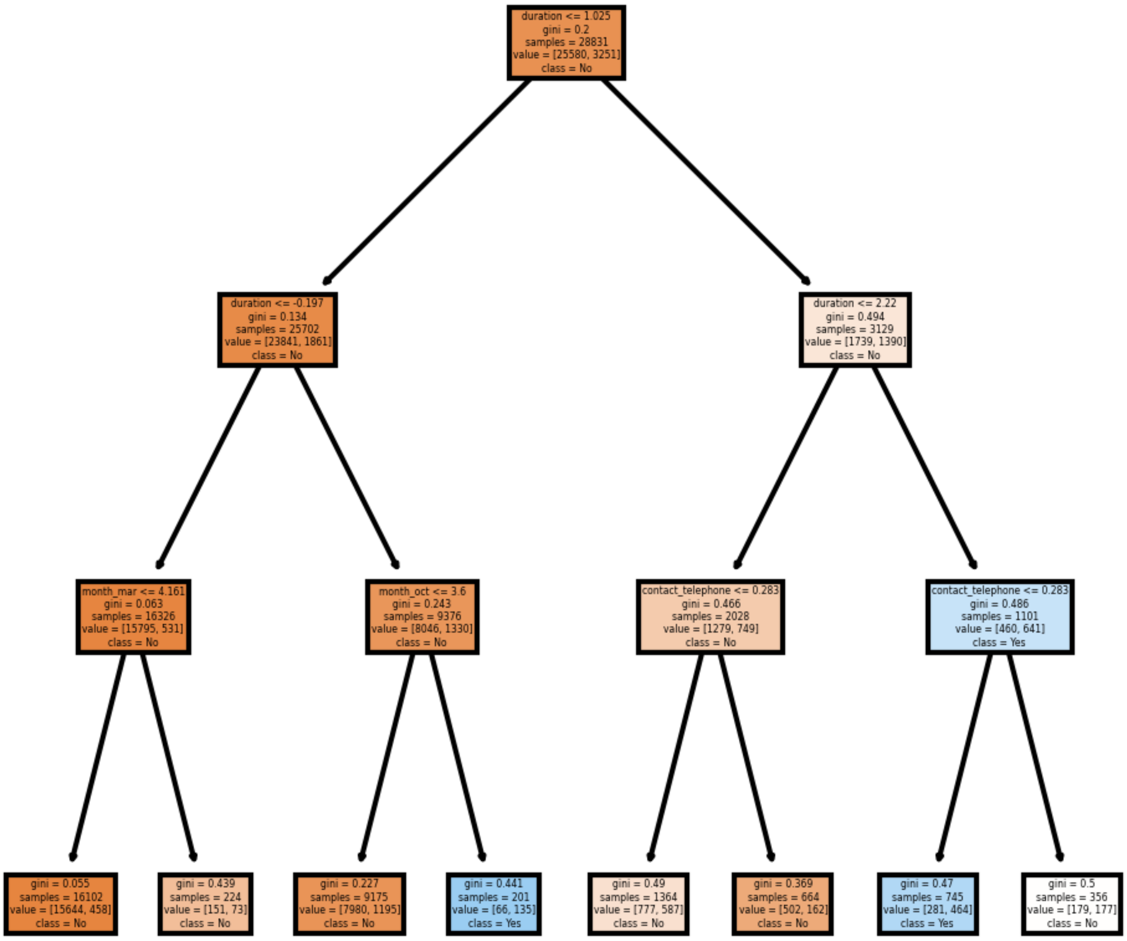
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3)
```

In [27]:

```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
plot_tree(dtree_main, filled = True, feature_names = X.columns, class_names=
["No","Yes"])



#fig.savefig('imagename.png')
```

Out[27]:

```
[Text(0.5, 0.875, 'duration <= 1.025\ngini = 0.2\nsamples = 28831\nvalue = [25580, 3251]\nclass = N
o'),
 Text(0.25, 0.625, 'duration <= -0.197\ngini = 0.134\nsamples = 25702\nvalue = [23841, 1861]\nclass =
No'),
 Text(0.125, 0.375, 'month_mar <= 4.161\ngini = 0.063\nsamples = 16326\nvalue = [15795, 531]\nclass =
No'),
 Text(0.0625, 0.125, 'gini = 0.055\nsamples = 16102\nvalue = [15644, 458]\nclass = No'),
 Text(0.1875, 0.125, 'gini = 0.439\nsamples = 224\nvalue = [151, 73]\nclass = No'),
 Text(0.375, 0.375, 'month_oct <= 3.6\ngini = 0.243\nsamples = 9376\nvalue = [8046, 1330]\nclass = N
o'),
 Text(0.3125, 0.125, 'gini = 0.227\nsamples = 9175\nvalue = [7980, 1195]\nclass = No'),
 Text(0.4375, 0.125, 'gini = 0.441\nsamples = 201\nvalue = [66, 135]\nclass = Yes'),
 Text(0.75, 0.625, 'duration <= 2.22\ngini = 0.494\nsamples = 3129\nvalue = [1739, 1390]\nclass = N
o'),
 Text(0.625, 0.375, 'contact_telephone <= 0.283\ngini = 0.466\nsamples = 2028\nvalue = [1279, 749]\ncl
ass = No'),
 Text(0.5625, 0.125, 'gini = 0.49\nsamples = 1364\nvalue = [777, 587]\nclass = No'),
 Text(0.6875, 0.125, 'gini = 0.369\nsamples = 664\nvalue = [502, 162]\nclass = No'),
 Text(0.875, 0.375, 'contact_telephone <= 0.283\ngini = 0.486\nsamples = 1101\nvalue = [460, 641]\ncla
ss = Yes'),
 Text(0.8125, 0.125, 'gini = 0.47\nsamples = 745\nvalue = [281, 464]\nclass = Yes'),
 Text(0.9375, 0.125, 'gini = 0.5\nsamples = 356\nvalue = [179, 177]\nclass = No')]
```



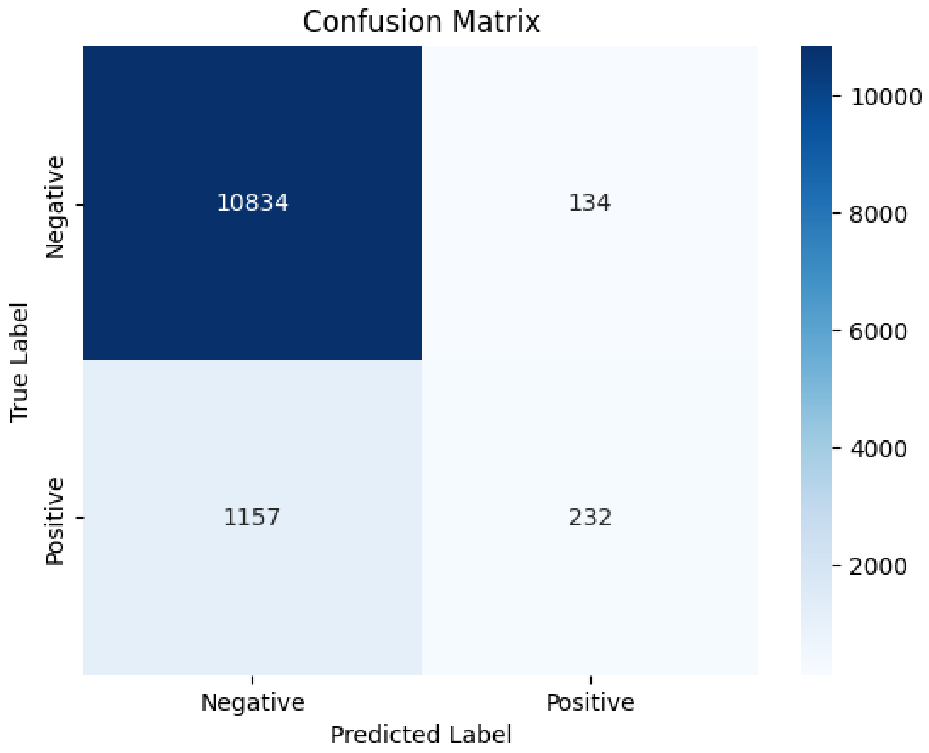
1b.) Confusion matrix on out of sample data. Visualize and store as variable

In [29]:

```
y_pred = dtree_main.predict(X_test)
y_true = y_test
cm_raw = confusion_matrix(y_true, y_pred)
```

In [30]:

```
class_labels = ['Negative', 'Positive']

# Plot the confusion matrix as a heatmap
sns.heatmap(cm_raw, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels,
yticklabels=class_labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



3.) Use bagging on your descision tree

In [31]:

```
#placeholder for optimizing max depth
dtree = DecisionTreeClassifier(max_depth = 3)
```

In [32]:

```
bagging = BaggingClassifier(estimator = dtree,
                  n_estimators = 100,
                  max_samples = .5,
                  max_features = 1.)
```

In [33]:

```
bagging.fit(X_scaled, y_train)


y_pred = bagging.predict(X_test)
```

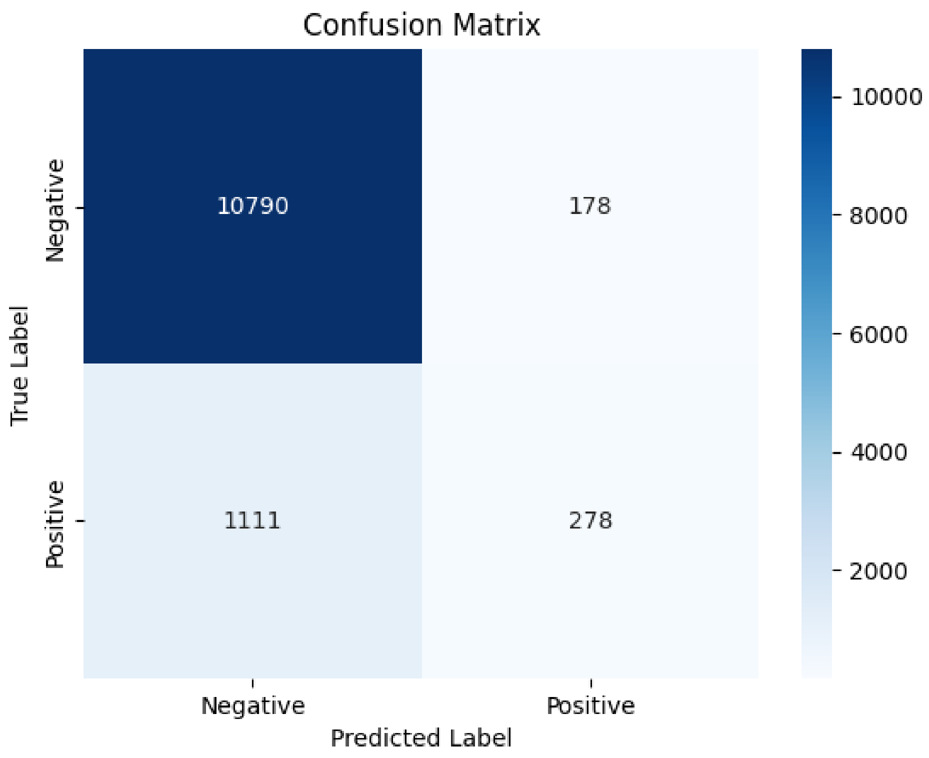
```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_bagging.py:802: DataConversionWarning: A col
umn-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), f
or example using ravel().
  y = column_or_1d(y, warn=True)
```

In [34]:

```
y_true = y_test
cm_raw = confusion_matrix(y_true, y_pred)
```

In [35]:

```
class_labels = ['Negative', 'Positive']


# Plot the confusion matrix as a heatmap
sns.heatmap(cm_raw, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels,
yticklabels=class_labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



4.) Boost your tree

In [36]:

```
from sklearn.ensemble import AdaBoostClassifier
```

In [37]:

```
#placeholder for optimizing max depth
dtree = DecisionTreeClassifier(max_depth = 3)
```

In [38]:

```
boost = AdaBoostClassifier(estimator = dtree,
                  n_estimators = 100,
                  learning_rate = .1)
```

In [39]:

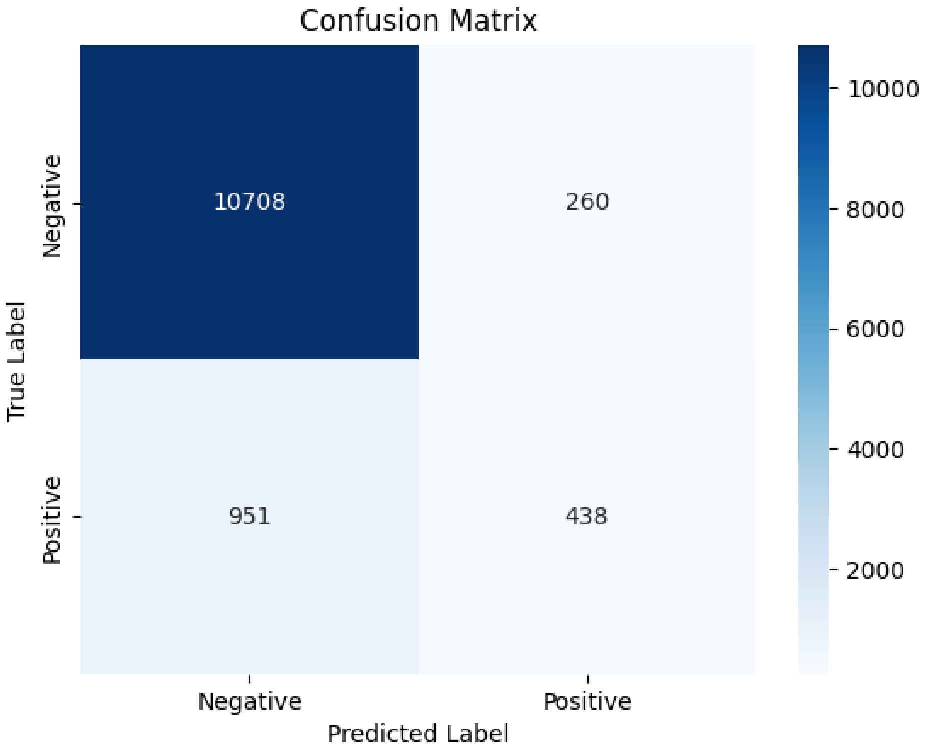
```
boost.fit(X_scaled, y_train)


y_pred = boost.predict(X_test)


y_true = y_test
cm_raw = confusion_matrix(y_true, y_pred)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A col
umn-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), f
or example using ravel().
  y = column_or_1d(y, warn=True)
```

In [40]:

```
# Plot the confusion matrix as a heatmap
sns.heatmap(cm_raw, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels,
yticklabels=class_labels)
```

```
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



5.) Train a Logistic Regression (Super Learner) on the Decision tree, Boosted tree, Bagged tree.

Interpret coefficients and significance

In [ ]:

```
# pip install mlens
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: mlens in /usr/local/lib/python3.8/dist-packages (0.2.3)
Requirement already satisfied: scipy>=0.17 in /usr/local/lib/python3.8/dist-packages (from mlens) (1.1
0.1)
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.8/dist-packages (from mlens) (1.2
2.4)
```

In [41]:

```
from sklearn.linear_model import LogisticRegression
```

In [42]:

```
base_predictions = [dtree_main.predict(X_train),
boost.predict(X_train),
bagging.predict(X_train)]
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but Dec
isionTreeClassifier was fitted without feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but Ada
BoostClassifier was fitted without feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but Bag
gingClassifier was fitted without feature names
  warnings.warn(
```

In [43]:

```
base_predictions
```

Out[43]:

```
[array([1, 0, 0, ..., 1, 0, 1], dtype=uint8),
 array([1, 1, 1, ..., 1, 1, 1], dtype=uint8),
 array([1, 1, 1, ..., 1, 1, 1], dtype=uint8)]
```

In [44]:

```
base_predictions = [list(dtree_main.predict(X_train)),
                    list(boost.predict(X_train)),
                    list(bagging.predict(X_train))]
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but Dec
isionTreeClassifier was fitted without feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but Ada
BoostClassifier was fitted without feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but Bag
gingClassifier was fitted without feature names
  warnings.warn(
```

In [47]:

```
np.array(base_predictions)
```

Out[47]:

```
array([[1, 0, 0, ..., 1, 0, 1],
       [1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1]], dtype=uint8)
```

In [48]:

```
n = len(base_predictions[0])
```

In [50]:

```
base_predictions = [np.array(base_predictions)[:,i] for i in range(n)]
```

In [51]:

```
super_learner = LogisticRegression()
```

In [52]:

```
super_learner.fit(base_predictions, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A col
umn-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), f
or example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[52]:

```
▾ LogisticRegression
LogisticRegression()
```

In [53]:

```
super_learner.coef_
```

Out[53]:

```
array([[1.17918485, 0.11003523, 0.23200188]])
```

Based on the results, it seems like the Decision Tree is exactly the best.

And it weights 1.17 $\times \hat{y}_{DecisionTree}$ + 0.11 $\times \hat{y}_{BoostedTree}$ + 0.23 $\times \hat{y}_{BaggedTree}$.

In [ ]: