SCOTT LEVY
MIS 776
ASSIGNMENT 4

CLASSIFICATION

1) Bank data file opened and viewed
2)

In [2]:
```
DATA_DIR = '../MIS 776'
FILE_NAME = 'BankSet.csv'
data_path = os.path.join(DATA_DIR, FILE_NAME)
datBank = pd.read_csv(data_path)
```

In [3]:
```
datBank
```

Out[3]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	campaign	pdays	previous	poutcome	purchase
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	1	-1	0	unknown	no
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	1	339	4	failure	no
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	1	330	1	failure	no
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	4	-1	0	unknown	no

The mean loan balance is 1,422.66 and the average number of contacts made to a customer is 2.79

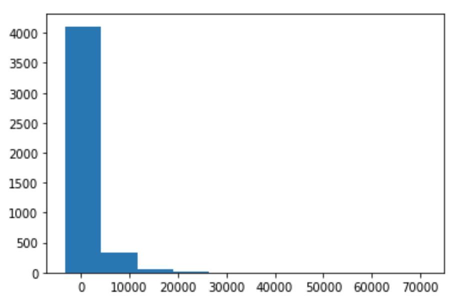
In [4]:
```
datBank.mean()
```

Out[4]:
```
age           41.170095
balance     1422.657819
day           15.915284
campaign       2.793630
pdays         39.766645
previous       0.542579
dtype: float64
```

3) Balance histogram

In [12]:
```
plt.hist(datBank['balance'])
```

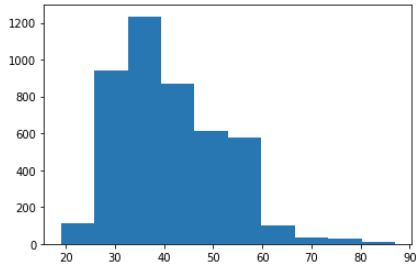
Out[12]:
```
(array([4.111e+03, 3.400e+02, 4.700e+01, 1.700e+01, 4.000e+00, 0.000e+00,
        1.000e+00, 0.000e+00, 0.000e+00, 1.000e+00]),
 array([-3313. ,  4137.1, 11587.2, 19037.3, 26487.4, 33937.5, 41387.6,
        48837.7, 56287.8, 63737.9, 71188. ]),
 <a list of 10 Patch objects>)
```



Balance is not a normal distribution, it is right-skewed.

4) Age histogram

```
In [13]: plt.hist(datBank['age'])
Out[13]: (array([ 111.,  944., 1235.,  869.,  612.,  576.,  100.,   36.,   30.,
            8.]),
          array([19. , 25.8, 32.6, 39.4, 46.2, 53. , 59.8, 66.6, 73.4, 80.2, 87. ]),
          <a list of 10 Patch objects>)
```



Age is not a normal distribution, it is right-skewed.

5)

```
In [10]: datBank = pd.concat([datBank, pd.get_dummies(datBank['purchase'], prefix='purchase', drop_first=True)], axis=1)
         datBank.rename(columns={'purchase_ yes':'purchase_code'}, inplace=True)
         datBank.drop(['purchase'], inplace=True, axis=1)
         datBank
Out[10]:
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	campaign	pdays	previous	poutcome	purchase_yes
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	1	-1	0	unknown	0
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	1	339	4	failure	0
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	1	330	1	failure	0
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	4	-1	0	unknown	0

```
In [17]: datBank['purchase_yes'].mean()
Out[17]: 0.11523999115239991
```

6) Correlation table:

```
In [13]: target_feature = ['purchase_yes']
         datBank[numeric_features+target_feature].corr()
Out[13]:
```

	age	balance	day	campaign	pdays	previous	purchase_yes
age	1.000000	0.083820	-0.017853	-0.005148	-0.008894	-0.003511	0.045092
balance	0.083820	1.000000	-0.008677	-0.009976	0.009437	0.026196	0.017905
day	-0.017853	-0.008677	1.000000	0.160706	-0.094352	-0.059114	-0.011244
campaign	-0.005148	-0.009976	0.160706	1.000000	-0.093137	-0.067833	-0.061147
pdays	-0.008894	0.009437	-0.094352	-0.093137	1.000000	0.577562	0.104087
previous	-0.003511	0.026196	-0.059114	-0.067833	0.577562	1.000000	0.116714
purchase_yes	0.045092	0.017905	-0.011244	-0.061147	0.104087	0.116714	1.000000

The variable that is most correlated with purchase_code is 'previous' (number of total contacts before this campaign began), and the variable that is least correlated is 'day' (day of the month of last contact). This is important because we want to be able to determine which variables ultimately have the greatest impacts on whether a customer decides to purchase a term deposit or not.

7) Aggregates:

In [15]:
```
1 datBank.groupby('job').mean()
```

Out[15]:

job	age	balance	day	campaign	pdays	previous	purchase_yes
admin.	39.682008	1226.736402	16.324268	2.631799	49.993724	0.644351	0.121339
blue-collar	40.156448	1085.161734	15.482030	2.846723	41.590909	0.493658	0.072939
entrepreneur	42.011905	1645.125000	15.255952	2.589286	32.273810	0.428571	0.089286
housemaid	47.339286	2083.803571	15.294643	2.500000	26.401786	0.357143	0.125000
management	40.540764	1766.928793	16.254902	2.973168	40.968008	0.549020	0.135191
retired	61.869565	2319.191304	15.556522	2.465217	35.073913	0.591304	0.234783
self-employed	41.453552	1392.409836	16.180328	3.278689	28.256831	0.590164	0.109290
services	38.570743	1103.956835	15.515588	2.822542	36.371703	0.443645	0.091127
student	26.821429	1543.821429	16.392857	2.392857	45.714286	0.964286	0.226190
technician	39.470052	1330.996094	16.183594	2.731771	39.265625	0.576823	0.108073
unemployed	40.906250	1089.421875	16.093750	2.679688	36.625000	0.484375	0.101562
unknown	48.105263	1501.710526	15.842105	2.552632	36.236842	0.500000	0.184211

In [16]:
```
1 datBank.groupby('marital').mean()
```

Out[16]:

marital	age	balance	day	campaign	pdays	previous	purchase_yes
divorced	45.475379	1122.390152	15.753788	2.604167	38.827652	0.439394	0.145833
married	43.454415	1463.195567	15.905971	2.847336	38.466929	0.519128	0.099035
single	33.927258	1460.414716	16.008361	2.751672	43.220736	0.642977	0.139632

In [17]:
```
1 datBank.groupby('education').mean()
```

Out[17]:

education	age	balance	day	campaign	pdays	previous	purchase_yes
primary	46.833333	1411.544248	15.505900	2.865782	35.069322	0.460177	0.094395
secondary	40.062446	1196.814397	15.977884	2.734172	40.934085	0.528621	0.106245
tertiary	39.645926	1775.423704	16.009630	2.901481	39.824444	0.612593	0.142963
unknown	45.299465	1701.245989	15.946524	2.486631	41.983957	0.508021	0.101604

In [18]:
```
1 datBank.groupby('housing').mean()
```

Out[18]:

housing	age	balance	day	campaign	pdays	previous	purchase_yes
no	43.511723	1595.277268	16.209990	2.80632	26.402141	0.467890	0.153415
yes	39.374756	1290.309496	15.689332	2.78390	50.013286	0.599844	0.085971

In [19]:
```
1 datBank.groupby('loan').mean()
```

Out[19]:

loan	age	balance	day	campaign	pdays	previous	purchase_yes
no	41.220627	1513.857963	15.932376	2.771018	41.088512	0.558486	0.124804
yes	40.890014	917.163531	15.820550	2.918958	32.439942	0.454414	0.062229

```
In [20]: datBank.groupby('poutcome').mean()
```

Out[20]:

poutcome	age	balance	day	campaign	pdays	previous	purchase_yes
failure	41.555102	1644.646939	14.395918	1.955102	243.167347	2.851020	0.128571
other	39.873096	1424.472081	15.101523	2.350254	219.385787	3.385787	0.192893
success	44.170543	1949.410853	14.581395	1.736434	163.713178	3.015504	0.643411
unknown	41.083671	1374.862078	16.205938	2.964912	-1.000000	0.000000	0.090958

There are a few interesting things that can be gleaned from these aggregates. There is a high correlation of purchases among people who are retired or students. There is a low correlation of purchases among people who are blue-collar workers, entrepreneurs, or in the services industry. Likewise, there is a low correlation of purchases among people who already have home loans or personal loans. This is important because it allows us to direct our marketing campaigns specifically towards those who are more likely to purchase the term loans.

8) Numerical encoding for 'job':

```
In [19]: datBank = pd.concat([datBank, pd.get_dummies(datBank['job'], drop_first=True)], axis=1)

         datBank
```

Out[19]:

ault	balance	housing	loan	contact	day	...	entrepreneur	housemaid	management	retired	self-employed	services	student	technician
no	1787	no	no	cellular	19	...	0	0	0	0	0	0	0	0
no	4789	yes	yes	cellular	11	...	0	0	0	0	0	1	0	0
no	1350	yes	no	cellular	16	...	0	0	1	0	0	0	0	0
no	1476	yes	yes	unknown	3	...	0	0	1	0	0	0	0	0
no	0	yes	no	unknown	5	...	0	0	0	0	0	0	0	0
no	747	no	no	cellular	23	...	0	0	1	0	0	0	0	0
no	307	yes	no	cellular	14	...	0	0	0	0	1	0	0	0

9) 9, 10, 11 code all in pic below 11)

10)

11)

```
In [21]: training_features = ['age','balance','day','campaign','pdays','previous','blue-collar','entrepreneur','housemaid','
         X=datBank[training_features]
         y=datBank[target_feature]

In [22]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.30, random_state=500)

In [23]: from sklearn.tree import DecisionTreeClassifier
         class_tree = DecisionTreeClassifier(max_depth=3)
         class_tree.fit(X_train, y_train)
```

12) Decision tree & confusion matrix:

```
In [29]:  from sklearn.metrics import confusion_matrix
          y_train_pred = class_tree.predict(X_train)
          confusion_matrix(y_train,y_train_pred)

Out[29]: array([[2784,   14],
                [ 338,   28]])
```

```
In [30]:  pd.crosstab(y_train['purchase_yes'], y_train_pr
Out[30]:
```

Predicted	0	1	All
True			
0	2784	14	2798
1	338	28	366
All	3122	42	3164

```
In [31]:  from sklearn.metrics import accuracy_score
          accuracy = accuracy_score(y_true=y_train, y_pred=y_train_pred)
          accuracy.round(3)

Out[31]: 0.889

In [32]:  from sklearn.metrics import precision_score
          precision = precision_score(y_true=y_train, y_pred=y_train_pred,
          precision.round(3)

Out[32]: array([0.892, 0.667])

In [33]:  from sklearn.metrics import recall_score
          recall = recall_score(y_true=y_train, y_pred=y_train_pred, avera
          recall.round(3)

Out[33]: array([0.995, 0.077])

In [35]:  F1 = 2 * (precision * recall) / (precision + recall)
          F1

Out[35]: array([0.94054054, 0.1372549 ])

In [34]:  pd.Series(data=class_tree.fea

Out[34]: pdays            0.706
         age              0.207
         day              0.048
         student          0.039
         entrepreneur     0.000
         balance          0.000
         campaign         0.000
         previous         0.000
         blue-collar      0.000
         unknown          0.000
         unemployed       0.000
         management       0.000
         retired          0.000
         self-employed    0.000
         services         0.000
         technician       0.000
         housemaid        0.000
         dtype: float64
```

The decision tree accuracy is 88.9%. It is 89.2% accurate at predicting “no” and 66.7% accurate at predicting “yes.” This means that out of all the “yes” outcomes preicted, 66.7% of them actually ended up being “yes.” However, it’s recall for “no” is 99.5% while it’s recall for “yes” is only 7.7%, which is not very encouraging, since our primary interest here is predicting “yes.” The recall score means that out of the entire population of actual “yes” outcomes, the model only correctly predicted 7.7% of them. It’s F-Measure is 0.94 for “no” and 0.13 for “yes.” It also shows the primary predictor as “pdays” which is the number of days that passed after the client was last contacted from a previous campaign. While this shows the importance of marketing, I do not feel it should be used exclusively since we can see that "age" also plays a noticeable role too.

13) Validation partition:

```
In [36]: y_test_pred = class_tree.predict(X_test)
         confusion_matrix(y_test,y_test_pred)

Out[36]: array([[1195,    7],
                [ 152,    3]])

In [37]: accuracy = accuracy_score(y_true=y_test, y_pred=y_test_pred)
         accuracy.round(3)

Out[37]: 0.883
```

The accuracy has dropped ever so slightly from 88.9% to 88.3% which indicates maybe a tiny amount of overfitting. However, of greater concern is that it's precision in predicting "yes" has dropped from 66.7% to only 30%.

14) Naïve Bayes:

```
In [38]: from sklearn.naive_bayes import GaussianNB
         nb = GaussianNB()
         nb.fit(X_train, y_train)
         y_train_pred_nb = nb.predict(X_train)
         confusion_matrix(y_train,y_train_pred_nb)

Out[38]: array([[2494,  304],
                [ 270,   96]])

In [46]: pd.crosstab(y_train['purchase_yes'], y_train_pr

Out[46]:
```

Predicted	0	1	All
True			
0	2494	304	2798
1	270	96	366
All	2764	400	3164

```
In [47]: accuracy = accuracy_score(y_true=y_train, y_pred=y_train_pred_nb)
         accuracy.round(3)

Out[47]: 0.819

In [48]: precision = precision_score(y_true=y_train, y_pred=y_train_pred_nb, average = None)
         precision.round(3)

Out[48]: array([0.902, 0.24 ])

In [49]: recall = recall_score(y_true=y_train, y_pred=y_train_pred_nb, average = None)
         recall.round(3)

Out[49]: array([0.891, 0.262])

In [50]: F1 = 2 * (precision * recall) / (precision + recall)
         F1

Out[50]: array([0.89679971, 0.25065274])

In [51]: y_test_pred_nb = nb.predict(X_test)
         confusion_matrix(y_test,y_test_pred_nb)

Out[51]: array([[1061,  141],
                [ 118,   37]])

In [52]: accuracy = accuracy_score(y_true=y_test, y_pred=y_test_pred_nb)
         accuracy.round(3)

Out[52]: 0.809
```

The overall accuracy of the NB model is at 81.9% and the validation partition accuracy is 80.9%, so noticeable lower than that of the decision tree model, and there is evidence of more overfitting in the NB model than there was with the decision tree. Precision and recall of “no” are at 90.2% and 89.1% respectively, so in the range of the decision model. However, for “yes” the precision and recall are at 24% and 26.2%, vs 66.7% and 7.7% for the decision tree model. The NB F-Measures are 0.897 for “no” and 0.251 for “yes.”

15) In deciding which model is a better fit for our goals here, I’ve tried to focus in on exactly what we are hoping to achieve, which in my opinion is maximizing the number of the general population that get converted to “yes” outcomes (i.e. purchasing a term deposit). So the “no” outcomes are not as important to us when looking at these models. In addition, their statistics on their handling of “no” outcomes are pretty much the same anyway, making them a wash. So we are focused on the “yes” outcomes, and I feel that it is more important to look at the recall stats, as opposed to the precision stats, since recall is more closely related to our goal of converting the maximum amount of the general population. With these models, the Naïve Bayes has almost triple the recall score of the decision tree model (26.2% vs 7.7%). I believe this outweighs the lower overall accuracy of the Naïve Bayes model. So in my opinion, the Naïve Bayes model is the better choice to go with here.