Loan Model Project / Yeoman Yoon

Import libraries and dataset

In [1]:
```
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

In [2]:
```
data = pd.read_csv('Loan_Modelling.csv')
```

In [3]:
```
print(f"There are {data.shape[1]} columns and {data.shape[0]} rows in the data set ")
data.head(10)
```

```
There are 14 columns and 5000 rows in the dataset
```

Out[3]:

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	Personal_Loan
0	1	25	1	49	91107	4	1.6	1	0	0
1	2	45	19	34	90089	3	1.5	1	0	0
2	3	39	15	11	94720	1	1.0	1	0	0
3	4	35	9	100	94112	1	2.7	2	0	0
4	5	35	8	45	91330	4	1.0	2	0	0
5	6	37	13	29	92121	4	0.4	2	155	0
6	7	53	27	72	91711	2	1.5	2	0	0
7	8	50	24	22	93943	1	0.3	3	0	0
8	9	35	10	81	90089	3	0.6	2	104	0
9	10	34	9	180	93023	1	8.9	3	0	1

Data Dictionary

- ID: Customer ID
- Age: Customer's age in completed years
- Experience: #years of professional experience
- Income: Annual income of the customer (in thousand dollars)
- ZIP Code: Home Address ZIP code.
- Family: the Family size of the customer
- CCAvg: Average spending on credit cards per month (in thousand dollars)
- Education: Education Level. 1: Undergrad; 2: Graduate;3: Advanced/Professional
- Mortgage: Value of house mortgage if any. (in thousand dollars)
- Personal_Loan: Did this customer accept the personal loan offered in the last campaign? → **Dependent variable**
- Securities_Account: Does the customer have securities account with the bank?
- CD_Account: Does the customer have a certificate of deposit (CD) account with the bank?
- Online: Do customers use internet banking facilities?
- CreditCard: Does the customer use a credit card issued by any other Bank (excluding All life Bank)?

In [4]:
```
data.drop(["ID"], axis=1, inplace = True) # Drop ID, we are not using this, ZIP Code maybe irrelavent. (will keep for now)

# data.drop(["ID", "ZIPCode", "Securities_Account", "CD_Account", "Online", "CreditCard"], axis=1, inplace = True) # Dropping these didn't change the result much.
```

In [5]:
```
data.info() #no missing values. good data type.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 13 columns):
Age                   5000 non-null int64
Experience            5000 non-null int64
Income                5000 non-null int64
ZIPCode               5000 non-null int64
Family                5000 non-null int64
CCAvg                 5000 non-null float64
Education             5000 non-null int64
Mortgage              5000 non-null int64
Personal_Loan         5000 non-null int64
Securities_Account    5000 non-null int64
CD_Account            5000 non-null int64
Online                5000 non-null int64
CreditCard            5000 non-null int64
dtypes: float64(1), int64(12)
memory usage: 507.9 KB
```

In [6]:
```
data.describe().T # negative number is definately an error. some outliers in income, CCAvg and Mortgage
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
Age	5000.0	45.338400	11.463166	23.0	35.0	45.0	55.0	67.0
Experience	5000.0	20.104600	11.467954	-3.0	10.0	20.0	30.0	43.0
Income	5000.0	73.774200	46.033729	8.0	39.0	64.0	98.0	224.0
ZIPCode	5000.0	93169.257000	1759.455086	90005.0	91911.0	93437.0	94608.0	96651.0
Family	5000.0	2.396400	1.147663	1.0	1.0	2.0	3.0	4.0
CCAvg	5000.0	1.937938	1.747659	0.0	0.7	1.5	2.5	10.0
Education	5000.0	1.881000	0.839869	1.0	1.0	2.0	3.0	3.0
Mortgage	5000.0	56.498800	101.713802	0.0	0.0	0.0	101.0	635.0
Personal_Loan	5000.0	0.096000	0.294621	0.0	0.0	0.0	0.0	1.0
Securities_Account	5000.0	0.104400	0.305809	0.0	0.0	0.0	0.0	1.0
CD_Account	5000.0	0.060400	0.238250	0.0	0.0	0.0	0.0	1.0
Online	5000.0	0.596800	0.490589	0.0	0.0	1.0	1.0	1.0
CreditCard	5000.0	0.294000	0.455637	0.0	0.0	0.0	1.0	1.0

Error Correction (No missing variables)

Experience has negative values

In [7]:
```
# (data["Experience"]<0).value_counts(ascending = True)
# data["Experience"].value_counts(ascending = True)

data["Experience"][data["Experience"]<4].value_counts()
```

Out[7]:
```
 3    129
 2     85
 1     74
 0     66
-1     33
-2     15
-3      4
Name: Experience, dtype: int64
```

Negative value may represent 0 or absolute value of itself.
My Decision: all 0.
My Reason: if numbers accidentally had negative sign, it is likely that value count of negetives will be symetric in count with postives, but it is going down.

In [8]:
```
# data["Experience"][data["Experience"]<0] = 0 # gives error

data["Experience"] = data["Experience"].replace([-1,-2,-3],0)
```

Univariate Analysis:

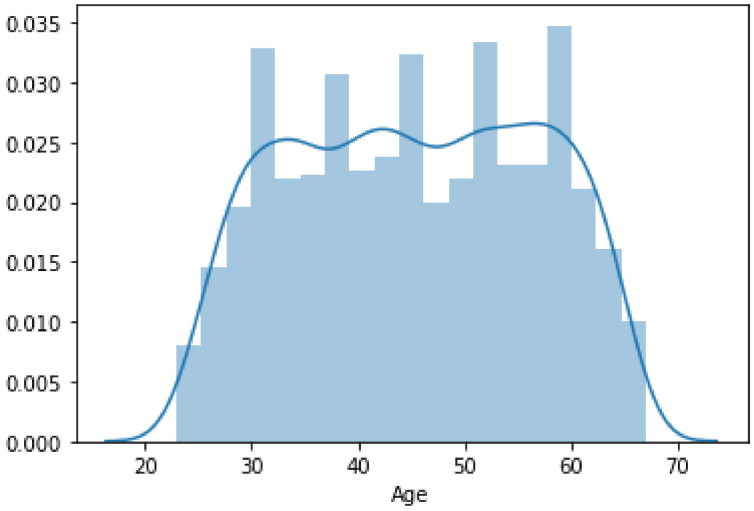
distplot(preferably continuous variable):

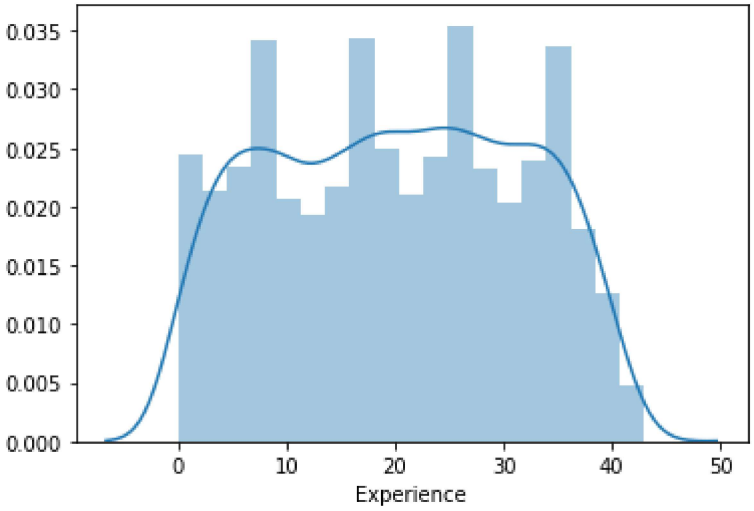
Age, Experience, Income, ZIPCode, CCAvg, Mortgage

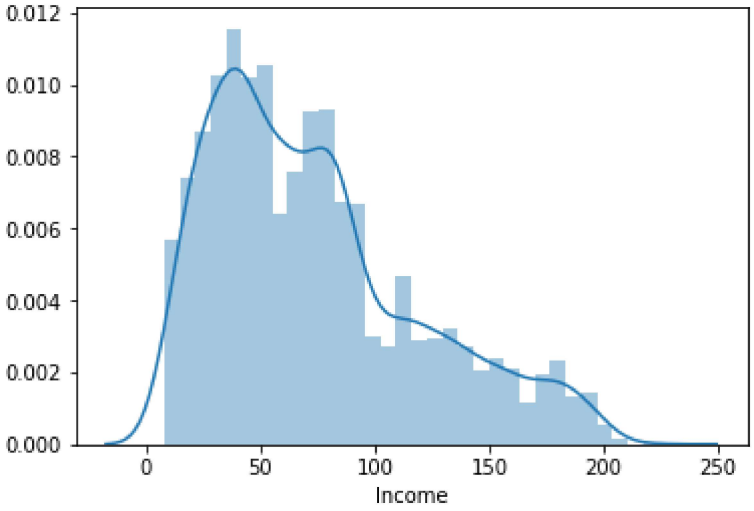
countplot(preferably categorical variable):

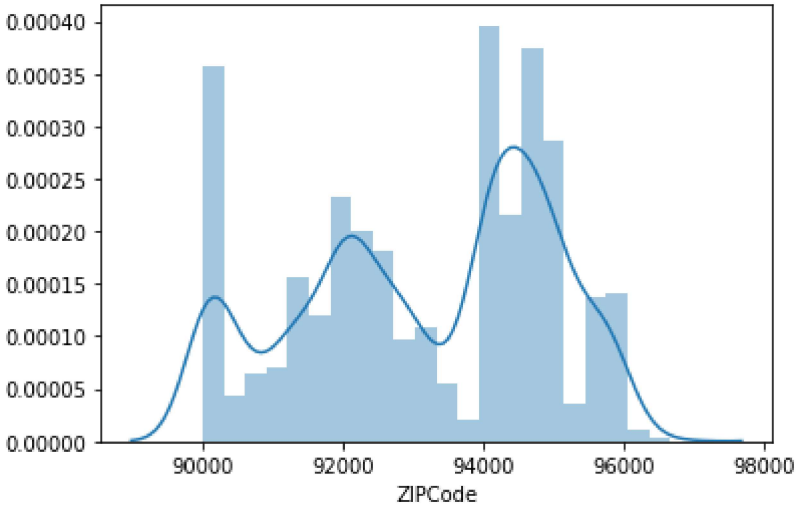
Family, Education, Personal_Loan, Securities_Account, CD_Account, Online, CreditCard

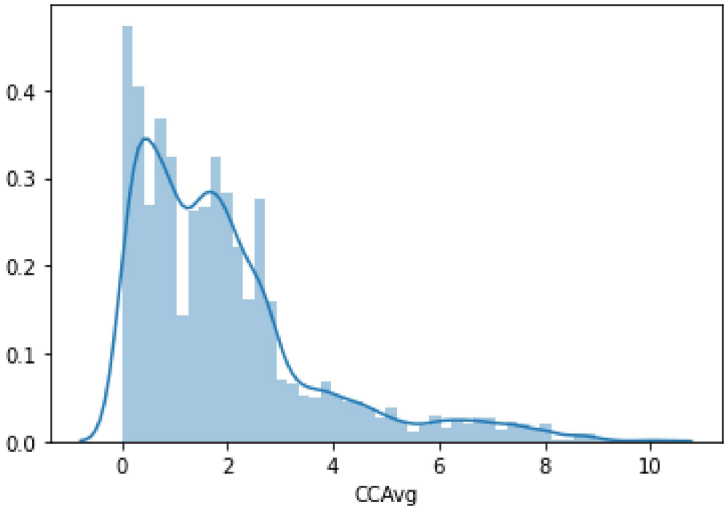
In [9]:
```
distPlotList = ["Age", "Experience", "Income", "ZIPCode", "CCAvg", "Mortgage"]
countPlotList = [ "Family", "Education", "Personal_Loan",
                  "Securities_Account", "CD_Account", "Online", "CreditCard"]
for i in distPlotList:
    sns.distplot(data[i])
    plt.show()

for i in countPlotList:
    sns.countplot(data[i])
    plt.show()
```

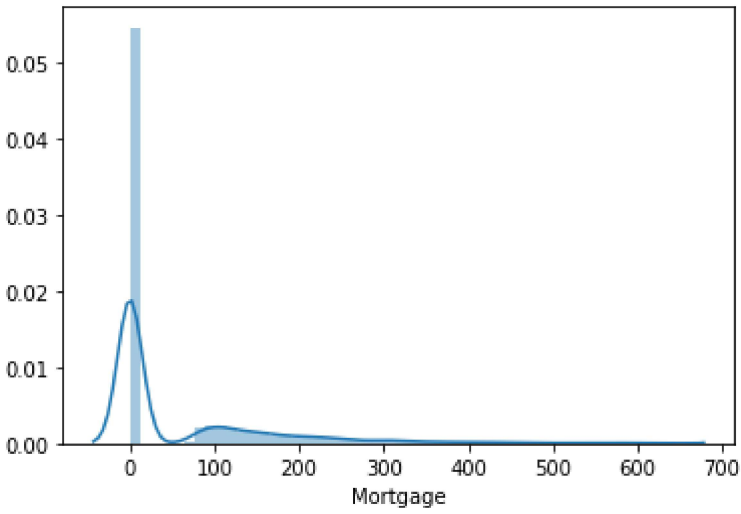

0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
20
30
40
50
60
70
Age

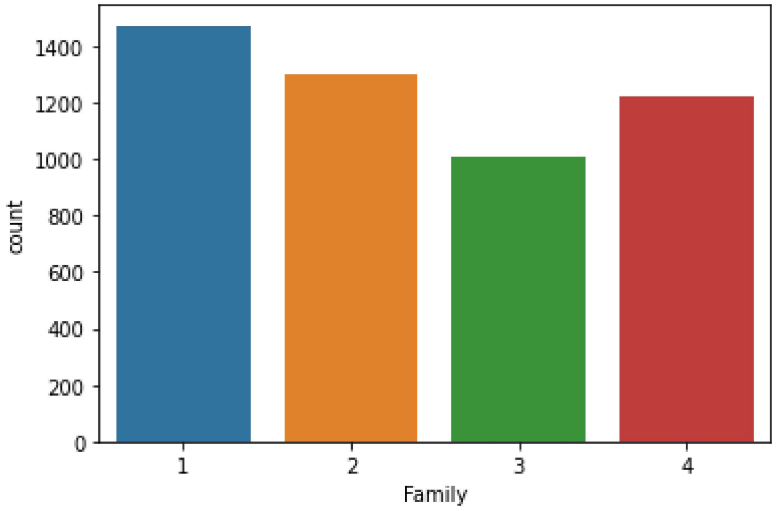

0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
10
20
30
40
50
Experience

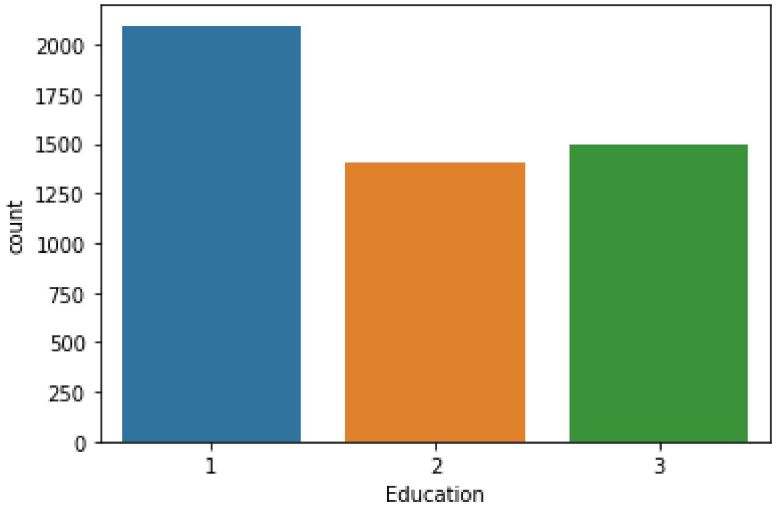

0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
50
100
150
200
250
Income

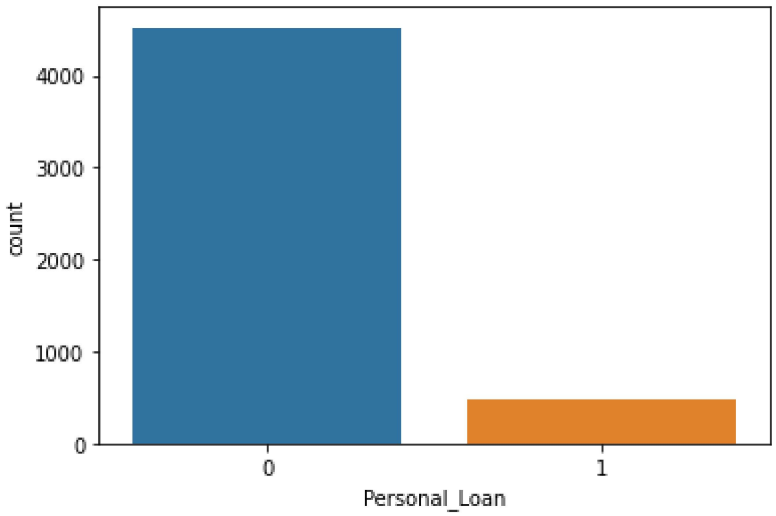

0.00040
0.00035
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
90000
92000
94000
96000
98000
ZIPCode

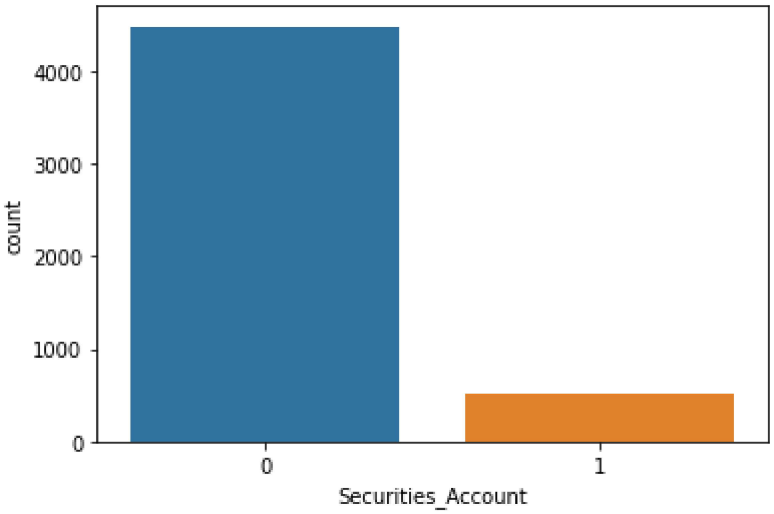

0.4
0.3
0.2
0.1
0.0
0
2
4
6
8
10
CCAvg

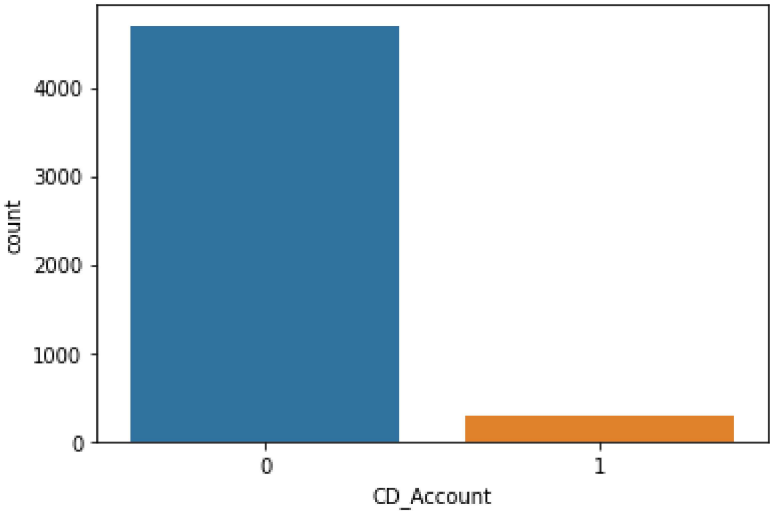

0.05
0.04
0.03
0.02
0.01
0.00
0
100
200
300
400
500
600
700
Mortgage

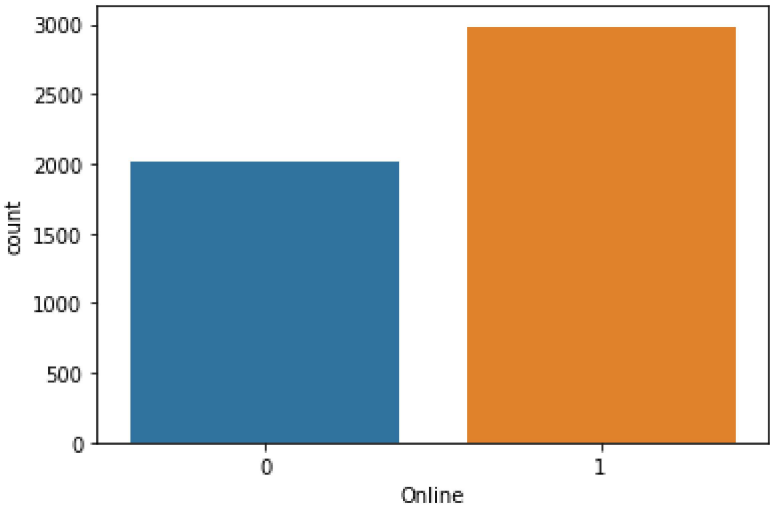

1400
1200
1000
800
600
400
200
0
count
1
2
3
4
Family

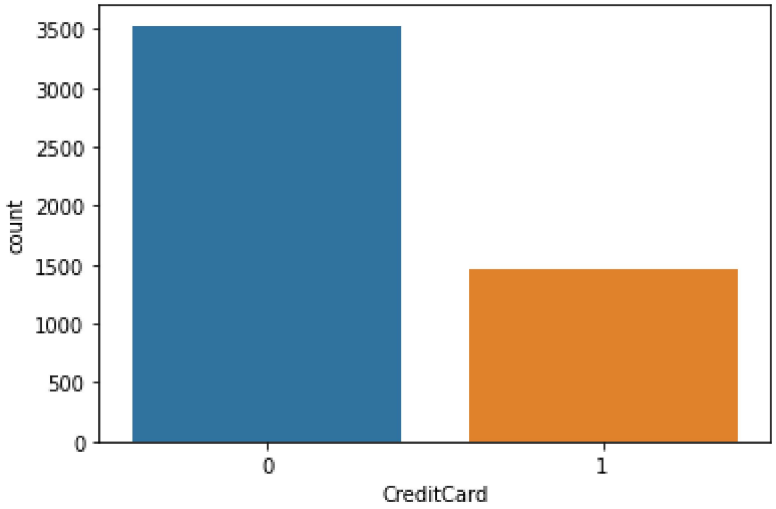

2000
1750
1500
1250
1000
750
500
250
0
count
1
2
3
Education


4000
3000
2000
1000
0
count
0
1
Personal_Loan


4000
3000
2000
1000
0
count
0
1
Securities_Account


4000
3000
2000
1000
0
count
0
1
CD_Account


3000
2500
2000
1500
1000
500
0
count
0
1
Online



In [10]:
```
a= data["Mortgage"][data["Mortgage"]==0].value_counts().sum()
b= data["Mortgage"][data["Mortgage"]!=0].value_counts().sum()

print (f"{a},{b}")
```

```
3462,1538
```

Personal Loan, Securities Account, and CD Account have serious unbalanced spread of data.
Age and Experience has very similar graph.
Some high sticking points in age and experience make me doubt if the data was pre-binned in some places.
Consider binning Mortgage with 0 and 1.

Bivariate analysis

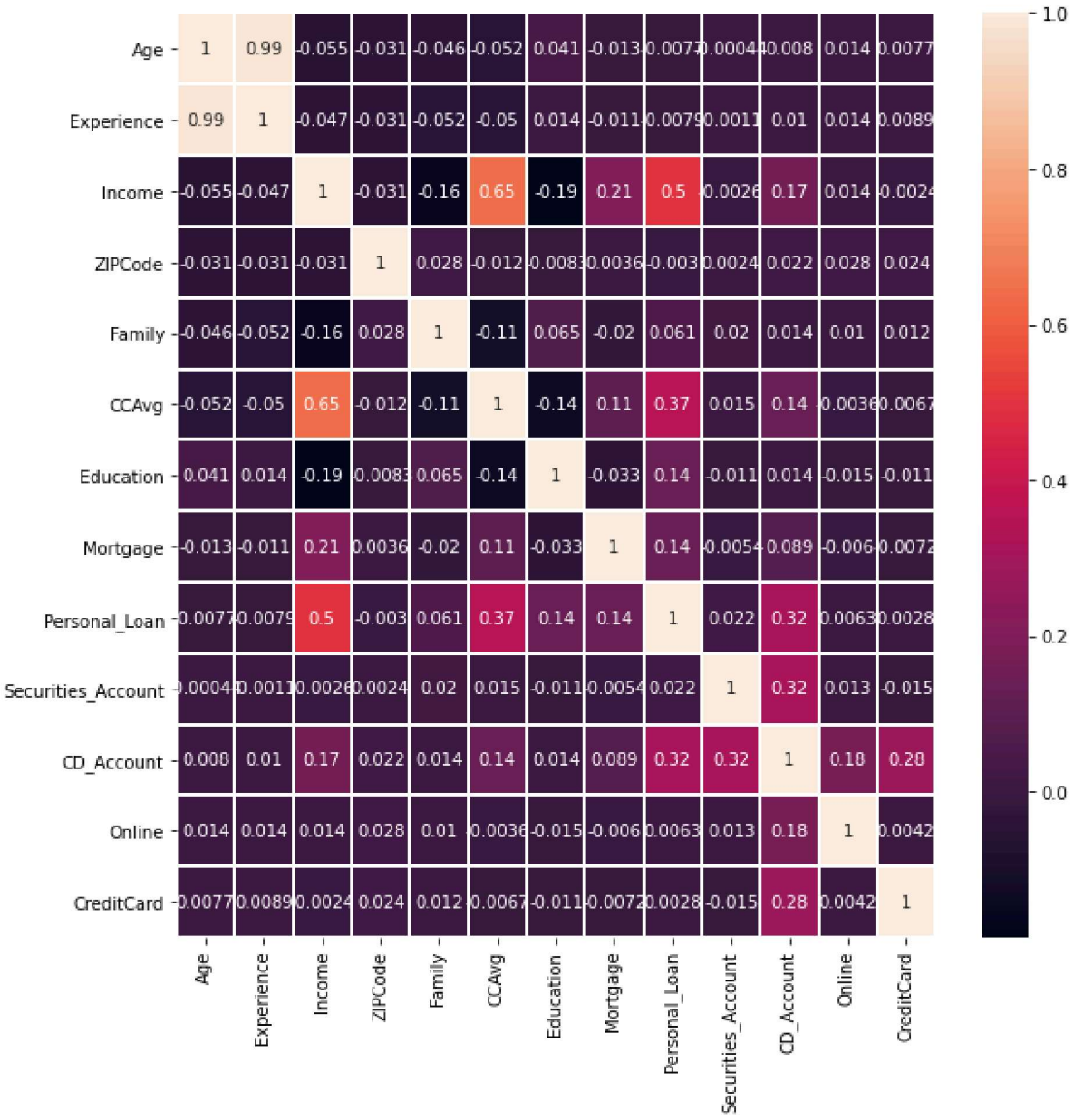
In [11]:
```
data.corr()
```

Out[11]:

	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	M
Age	1.000000	0.994198	-0.055269	-0.030530	-0.046418	-0.052012	0.041334	-(
Experience	0.994198	1.000000	-0.046729	-0.030649	-0.052213	-0.049912	0.013536	-(
Income	-0.055269	-0.046729	1.000000	-0.030709	-0.157501	0.645984	-0.187524	(
ZIPCode	-0.030530	-0.030649	-0.030709	1.000000	0.027512	-0.012188	-0.008266	(
Family	-0.046418	-0.052213	-0.157501	0.027512	1.000000	-0.109275	0.064929	-(
CCAvg	-0.052012	-0.049912	0.645984	-0.012188	-0.109275	1.000000	-0.136124	(
Education	0.041334	0.013536	-0.187524	-0.008266	0.064929	-0.136124	1.000000	-(
Mortgage	-0.012539	-0.010840	0.206806	0.003614	-0.020445	0.109905	-0.033327	1
Personal_Loan	-0.007726	-0.007858	0.502462	-0.002974	0.061367	0.366889	0.136722	(
Securities_Account	-0.000436	-0.001111	-0.002616	0.002422	0.019994	0.015086	-0.010812	-(
CD_Account	0.008043	0.010046	0.169738	0.021671	0.014110	0.136534	0.013934	(
Online	0.013702	0.013975	0.014206	0.028317	0.010354	-0.003611	-0.015004	-(
CreditCard	0.007681	0.008910	-0.002385	0.024033	0.011588	-0.006689	-0.011014	-(

In [12]:
```
plt.subplots(figsize=(10,10))
sns.heatmap(data.corr(), annot =True, linewidth=1)
```

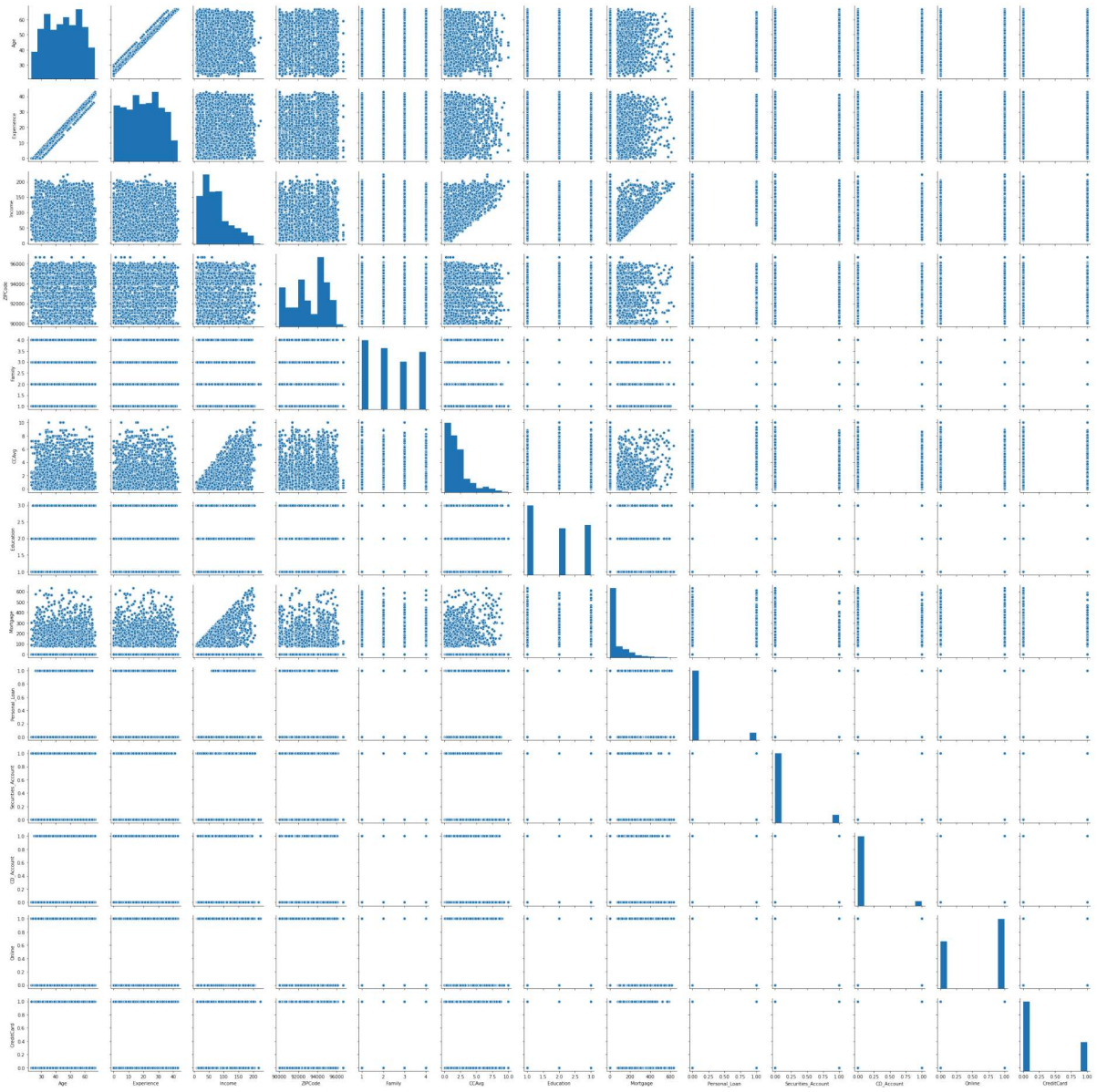
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x25aa601fb88>



It is important to note Age and Experience has correlation of almost 1 and very similar correlations with other variables as well.
Zipcode has almost no correlation with any variables (|x| < 0.035)
Consider dropping both.

In [13]:
```
sns.pairplot(data)
```

Out[13]:
```
<seaborn.axisgrid.PairGrid at 0x25aa5fbbb88>
```



Correlation of Age and Experience is too high.
CCAvg, Personal Loan, and Income have significant correlation to each other.

Data Cleaning

Require dummie variables, Binning zipcode

In [14]:
```
# data['ZIPCode'].unique() # 90xxx ~ 96xxx
```

In [15]:
```
binned_ZIPCode = pd.cut(data["ZIPCode"],[90000, 91000, 92000, 93000, 94000, 95
000, 97000])
```

In [16]:
```
binned_ZIPCode.value_counts()
```

Out[16]:
```
(94000, 95000]    1472
(92000, 93000]     988
(95000, 97000]     855
(90000, 91000]     703
(91000, 92000]     565
(93000, 94000]     417
Name: ZIPCode, dtype: int64
```

In [17]:
```
data['binned_ZIPCode'] = pd.cut(
    data["ZIPCode"], [90000, 91000, 92000, 93000, 94000, 95000, 97000],
    labels = ["90001 to 91000", "91001 to 92000", "92001 to 93000", "93001 to 94000", "94001 to 95000", "95001 to 97000"]
)

data.drop(["ZIPCode"], axis= 1, inplace =True)
```

In [18]:
```
data.head()
```

Out[18]:

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Personal_Loan	Securities_Ac
0	25	1	49	4	1.6	1	0	0	
1	45	19	34	3	1.5	1	0	0	
2	39	15	11	1	1.0	1	0	0	
3	35	9	100	1	2.7	2	0	0	
4	35	8	45	4	1.0	2	0	0	

In [19]:
```
data = pd.get_dummies(data, columns=["binned_ZIPCode"])
data.head()
```

Out[19]:

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Personal_Loan	Securities_Ac
0	25	1	49	4	1.6	1	0	0	
1	45	19	34	3	1.5	1	0	0	
2	39	15	11	1	1.0	1	0	0	
3	35	9	100	1	2.7	2	0	0	
4	35	8	45	4	1.0	2	0	0	

In [20]:
```
# data.drop(["Age"], axis=1 ,inplace =True) #Dropping Age didn't change accuracy of Logistic Regression...&
```

Even though the ZIP Code is continuous with region, it is not obvious regions have similar characteristics. Therefore, categorized and created dummi-variables.

Data Distribution for Test

In [21]:
```
X = data.drop("Personal_Loan", axis =1)
Y = data["Personal_Loan"]

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, rando
m_state=1)
```

In [22]:
```
print("{0:0.2f}% data is in training set".format((len(x_train)/len(data.inde
x)) * 100))
print("{0:0.2f}% data is in test set".format((len(x_test)/len(data.index)) * 1
00))
```

```
70.00% data is in training set
30.00% data is in test set
```

Data Balancing

Personal Loan, Securities Account, and CD Account have serious unbalanced spread of data.

In [23]:
```
print("Original Personal_Loan True Values    : {0} ({1:0.2f}%)".format(len(dat
a.loc[data['Personal_Loan'] == 1]), (len(data.loc[data['Personal_Loan'] ==
1])/len(data.index)) * 100))
print("Original Personal_Loan False Values   : {0} ({1:0.2f}%)".format(len(dat
a.loc[data['Personal_Loan'] == 0]), (len(data.loc[data['Personal_Loan'] ==
0])/len(data.index)) * 100))
print("")
print("Training Personal_Loan True Values    : {0} ({1:0.2f}%)".format(len(y_t
rain[y_train[:] == 1]), (len(y_train[y_train[:] == 1])/len(y_train)) * 100))
print("Training Personal_Loan False Values   : {0} ({1:0.2f}%)".format(len(y_t
rain[y_train[:] == 0]), (len(y_train[y_train[:] == 0])/len(y_train)) * 100))
print("")
print("Test Personal_Loan True Values        : {0} ({1:0.2f}%)".format(len(y_t
est[y_test[:] == 1]), (len(y_test[y_test[:] == 1])/len(y_test)) * 100))
print("Test Personal_Loan False Values       : {0} ({1:0.2f}%)".format(len(y_t
est[y_test[:] == 0]), (len(y_test[y_test[:] == 0])/len(y_test)) * 100))
print("")
```

```
Original Personal_Loan True Values    : 480 (9.60%)
Original Personal_Loan False Values   : 4520 (90.40%)

Training Personal_Loan True Values    : 331 (9.46%)
Training Personal_Loan False Values   : 3169 (90.54%)

Test Personal_Loan True Values        : 149 (9.93%)
Test Personal_Loan False Values       : 1351 (90.07%)
```

In [24]:
```
print("Original Securities_Account True Values    : {0} ({1:0.2f}%)".format(le
n(data.loc[data['Securities_Account'] == 1]), (len(data.loc[data['Securities_A
ccount'] == 1])/len(data.index)) * 100))
print("Original Securities_Account False Values   : {0} ({1:0.2f}%)".format(le
n(data.loc[data['Securities_Account'] == 0]), (len(data.loc[data['Securities_A
ccount'] == 0])/len(data.index)) * 100))
print("")
print("Training Securities_Account True Values    : {0} ({1:0.2f}%)".format(le
n(x_train[x_train['Securities_Account'] == 1]), (len(x_train[x_train['Securiti
es_Account'] == 1])/len(x_train)) * 100))
print("Training Securities_Account False Values   : {0} ({1:0.2f}%)".format(le
n(x_train[x_train['Securities_Account'] == 0]), (len(x_train[x_train['Securiti
es_Account'] == 0])/len(x_train)) * 100))
print("")
print("Test Securities_Account True Values        : {0} ({1:0.2f}%)".format(le
n(x_test[x_test['Securities_Account'] == 1]), (len(x_test[x_test['Securities_A
ccount'] == 1])/len(x_test)) * 100))
print("Test Securities_Account False Values       : {0} ({1:0.2f}%)".format(le
n(x_test[x_test['Securities_Account'] == 0]), (len(x_test[x_test['Securities_A
ccount'] == 0])/len(x_test)) * 100))
print("")
```

```
Original Securities_Account True Values    : 522 (10.44%)
Original Securities_Account False Values   : 4478 (89.56%)

Training Securities_Account True Values    : 366 (10.46%)
Training Securities_Account False Values   : 3134 (89.54%)

Test Securities_Account True Values        : 156 (10.40%)
Test Securities_Account False Values       : 1344 (89.60%)
```

In [25]:
```
print("Original CD_Account True Values    : {0} ({1:0.2f}%)".format(len(data.l
oc[data['CD_Account'] == 1]), (len(data.loc[data['CD_Account'] == 1])/len(dat
a.index)) * 100))
print("Original CD_Account False Values   : {0} ({1:0.2f}%)".format(len(data.l
oc[data['CD_Account'] == 0]), (len(data.loc[data['CD_Account'] == 0])/len(dat
a.index)) * 100))
print("")
print("Training CD_Account True Values    : {0} ({1:0.2f}%)".format(len(x_trai
n[x_train['CD_Account'] == 1]), (len(x_train[x_train['CD_Account'] == 1])/len
(x_train)) * 100))
print("Training CD_Account False Values   : {0} ({1:0.2f}%)".format(len(x_trai
n[x_train['CD_Account'] == 0]), (len(x_train[x_train['CD_Account'] == 0])/len
(x_train)) * 100))
print("")
print("Test CD_Account True Values        : {0} ({1:0.2f}%)".format(len(x_test
[x_test['CD_Account'] == 1]), (len(x_test[x_test['CD_Account'] == 1])/len(x_te
st)) * 100))
print("Test CD_Account False Values       : {0} ({1:0.2f}%)".format(len(x_test
[x_test['CD_Account'] == 0]), (len(x_test[x_test['CD_Account'] == 0])/len(x_te
st)) * 100))
print("")
```

```
Original CD_Account True Values    : 302 (6.04%)
Original CD_Account False Values   : 4698 (93.96%)

Training CD_Account True Values    : 219 (6.26%)
Training CD_Account False Values   : 3281 (93.74%)

Test CD_Account True Values        : 83 (5.53%)
Test CD_Account False Values       : 1417 (94.47%)
```

Logistic Regression

In [26]:
```
from sklearn import metrics
from sklearn.linear_model import LogisticRegression

# Fit the model on train
model = LogisticRegression(solver="liblinear",class_weight="balanced")
model.fit(x_train, y_train)
#predict on test



y_predict_LR = model.predict(x_test)

coef_df = pd.DataFrame(model.coef_)
coef_df['intercept'] = model.intercept_
print(coef_df)
```

```
          0         1         2         3        4         5         6  \
0 -0.235298  0.229293  0.050007  0.543203  0.27756  1.277733  0.000494

          7         8        9        10        11      12       13        14
\
0 -1.396873  3.628517 -0.54137 -1.053017 -0.487549 -0.4975 -0.50426 -0.607696

         15        16  intercept
0 -0.496149 -0.640155  -3.233309
```

In [27]:
```
# ? LogisticRegression
```

0 Age,
1 Experience,
2 Income,
3 Family,
4 CCAvg,
5 Education,
6 Mortgage,
7 Securities_Account,
8 CD_Account,
9 Online,
10 CreditCard,
11~16 ZIPCode

In [28]:
```
model_score = model.score(x_test, y_test)
print(model_score)
```

```
0.8913333333333333
```

In [29]:
```
cm_LR=metrics.confusion_matrix(y_test, y_predict_LR, labels=[1, 0])

df_cm = pd.DataFrame(cm_LR, index = [i for i in ["1","0"]],
                  columns = [i for i in ["Predict 1","Predict 0"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True)
```

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x25aadd94c08>

In [30]:
```
print(cm_LR[0])
print(cm_LR[1])
```

```
[128  21]
[ 142 1209]
```

In [31]:
```
recall_LR = cm_LR[0][0]/(cm_LR[0][0]+cm_LR[0][1])
recall_LR
```

Out[31]: 0.8590604026845637

In [32]:
```
specificity_LR = cm_LR[1][1]/(cm_LR[1][1]+cm_LR[1][0])
specificity_LR
```

Out[32]: 0.8948926720947447

94.5% is very accurate model. However, Recall is only 56.4%. I think we can do better by reducing False Negative rate in order to avoid losing potential customer. However, it is also costly to advertise too many people. Accuracy may be good for this model but we can still improve the model. (Dropping Age Or Experience didn't change the accuracy.)

class_weight="balanced" made Recall 85.9% (overall accuracy did go down to 89%)

Decision Tree

In [33]:
```
dTree = DecisionTreeClassifier(criterion = 'gini', max_depth = 6,random_state=
1,class_weight="balanced")
dTree.fit(x_train, y_train)

y_predict_Tree = dTree.predict(x_test)
```

In [34]:
```
# len(y_predict_Tree)
```

In [35]:
```
print("Accuracy on training set : ",dTree.score(x_train, y_train))
print("Accuracy on test set : ",dTree.score(x_test, y_test))
```

```
Accuracy on training set :  0.9751428571428571
Accuracy on test set :  0.962
```

In [36]:
```
cm_Tree=metrics.confusion_matrix(y_test, y_predict_Tree, labels=[1, 0])

df_cm = pd.DataFrame(cm_Tree, index = [i for i in ["1","0"]],
                  columns = [i for i in ["Predict 1","Predict 0"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True)
```

Out[36]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x25aaf873f88>
```

In [37]:
```
print(cm_Tree[0])
print(cm_Tree[1])
```

```
[142   7]
[  50 1301]
```

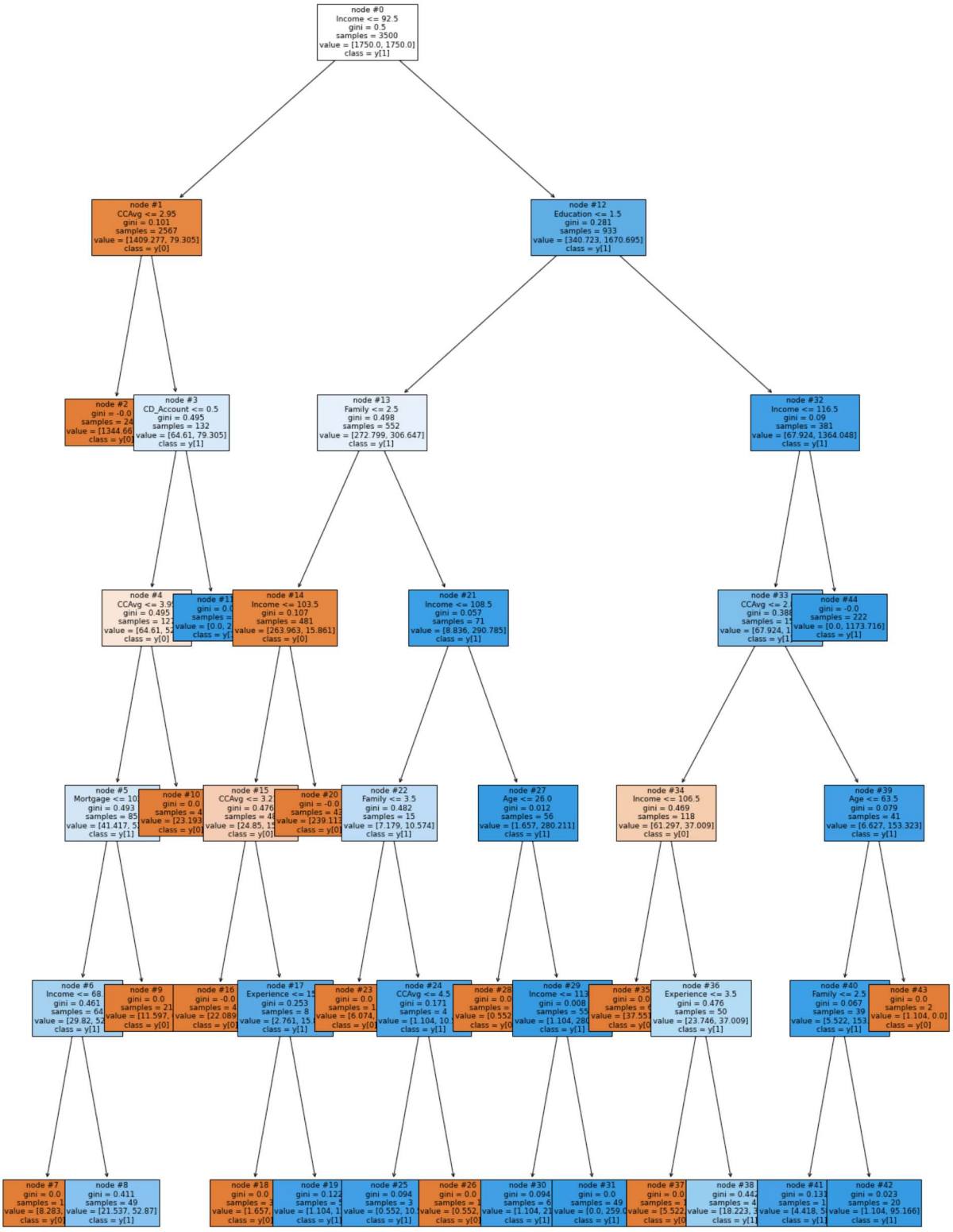
In [38]:
```
recall_Tree = cm_Tree[0][0]/(cm_Tree[0][0]+cm_Tree[0][1])
recall_Tree
```

Out[38]: 0.9530201342281879

In [39]:
```
specificity_Tree = cm_Tree[1][1]/(cm_Tree[1][1]+cm_Tree[1][0])
specificity_Tree
```

Out[39]: 0.9629903774981495

In [40]:
```
plt.figure(figsize=(20,30))
tree.plot_tree(dTree,feature_names=list(X.columns),filled=True,fontsize=9,node_ids=True,class_names=True)
plt.show()
```



Accuracy goes up significantly until max depth of 6. class_weight="balanced" made recall value to go higher.

From looking at the Tree, only: **[Income, CCAvg, Education, CD_Account, Family, Mortgage, Age, Experience]** were used.

This may indicate all the other columns/variables are meaningless. --> after testing without them it is almost the same. but for the sake of project I will leave them.

Income is the most important variable in this model (Root Node)

ERROR EDA

In [41]:
```
prediction_Tree = pd.concat([x_test, y_test], axis=1) # merge the test set.
prediction_Tree["prediction"] = y_predict_Tree # add prediction

wrong_prediction_FN = prediction_Tree[prediction_Tree["prediction"]==0][predic
tion_Tree["Personal_Loan"]==1].copy() #False Negative
wrong_prediction_FN
```

```
C:\Users\yeoma\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: UserWarni
ng: Boolean Series key will be reindexed to match DataFrame index.
  after removing the cwd from sys.path.
```

Out[41]:

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Securities_Account	CD_A
349	26	2	60	2	3.0	1	132	0	
927	65	40	95	3	3.7	2	138	0	
1518	43	17	64	4	3.0	3	221	0	
1577	34	8	65	1	3.0	1	227	0	
1126	32	8	104	2	3.7	1	0	0	
2539	32	7	98	1	4.2	1	171	1	
671	65	41	105	1	3.0	2	282	1	

In [42]:
```
wrong_prediction_FP = prediction_Tree[prediction_Tree["prediction"]==1][predic
tion_Tree["Personal_Loan"]==0].copy() # False Positive
wrong_prediction_FP
```

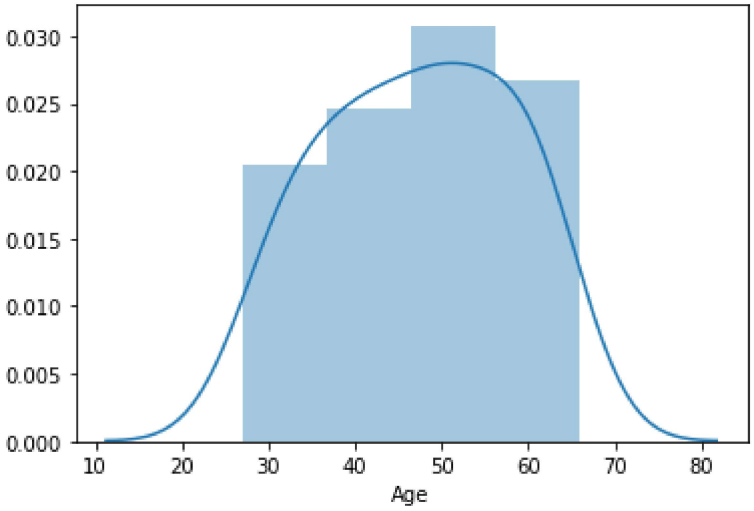
```
C:\Users\yeoma\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: UserWarni
ng: Boolean Series key will be reindexed to match DataFrame index.
  """Entry point for launching an IPython kernel.
```

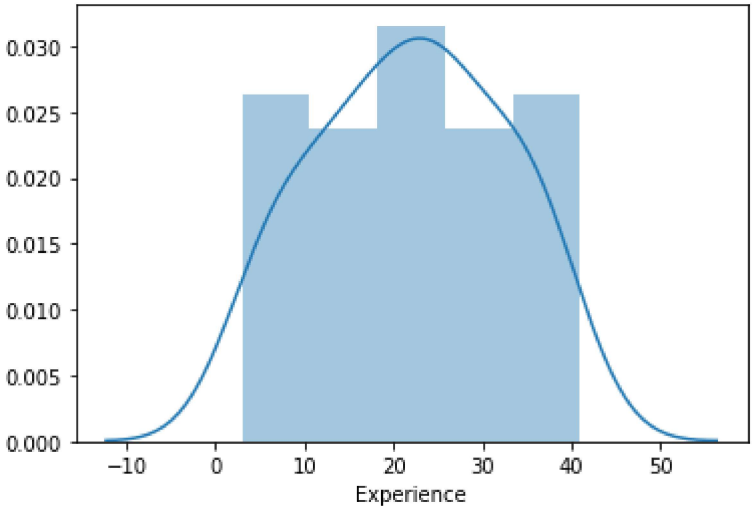
Out[42]:

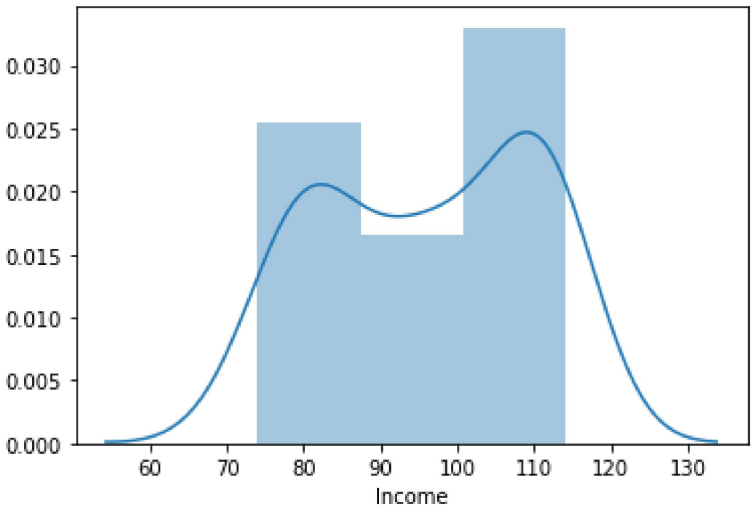
	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Securities_Account	CD_A
932	51	27	112	3	1.8	2	0	1	
792	41	16	98	1	4.0	3	0	0	
2982	59	33	111	3	4.4	1	0	0	
3420	66	41	114	1	0.8	3	0	0	
3067	31	5	101	1	2.9	3	170	1	
3277	43	19	81	2	3.2	1	0	0	
729	58	28	90	1	3.0	3	0	0	
836	42	17	74	3	3.0	1	0	0	
1074	39	14	75	3	3.0	1	0	0	
3792	62	36	109	4	1.7	3	0	0	
12	48	23	114	2	3.8	3	0	1	
394	33	9	80	4	3.4	1	0	0	
3409	29	5	113	2	2.0	2	84	0	
4392	52	27	81	4	3.8	2	0	0	
4580	50	24	102	2	6.3	1	0	0	
4554	41	16	109	3	1.0	1	0	1	
2351	55	31	74	2	3.2	3	0	0	
1875	27	3	112	3	2.5	1	389	1	
2016	41	17	93	4	0.8	1	218	0	
3274	31	5	110	2	1.5	3	0	0	
190	60	36	93	1	4.3	1	0	0	
587	50	24	94	1	4.9	1	272	0	
4229	54	24	83	1	3.0	3	0	0	
4571	58	28	95	1	3.0	3	0	0	
2098	59	35	94	1	3.8	1	272	0	
1147	37	13	111	1	0.8	2	0	0	
3730	30	6	112	3	2.5	1	0	0	
3645	42	17	79	1	3.7	3	0	1	
1030	61	35	112	4	1.7	3	0	0	
3042	52	26	78	3	3.0	2	0	0	
1832	54	29	79	4	3.8	2	0	1	
123	37	13	84	1	3.6	2	0	1	
2349	59	35	94	1	4.3	1	76	1	
3468	43	19	113	2	1.8	2	0	0	
2630	63	37	113	4	1.7	3	0	0	

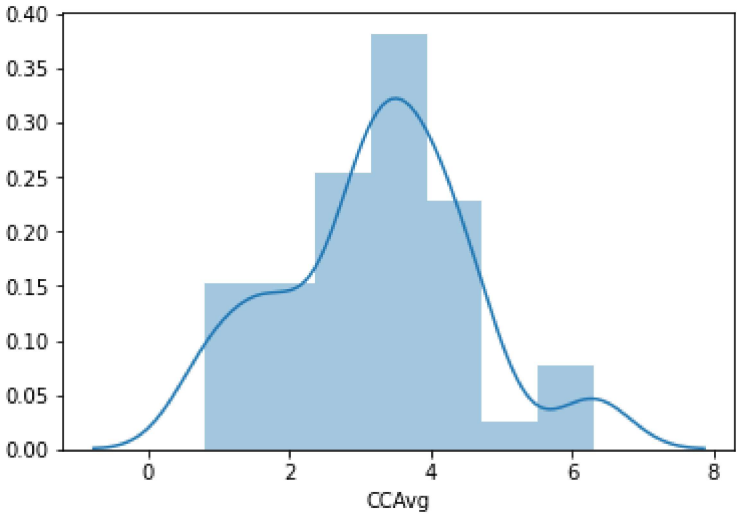
	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Securities_Account	CD_A
2665	35	9	105	2	4.5	3	0	0	
2523	49	23	100	2	6.3	1	0	0	
3308	48	23	108	2	3.8	3	0	0	
4925	64	39	82	4	3.4	2	0	0	
990	34	10	81	4	3.4	1	0	0	
4816	50	24	83	3	3.0	2	0	0	
784	48	22	98	2	6.3	1	0	0	
2738	35	9	103	2	4.5	3	0	0	
4933	47	23	94	1	4.7	1	0	0	
2470	33	7	81	2	4.5	3	187	0	
719	61	35	110	3	4.4	1	0	1	
2625	61	36	108	4	3.4	2	0	0	
3544	45	19	109	3	1.1	1	0	0	
1401	40	15	84	1	3.7	3	0	0	
829	55	30	81	4	3.8	2	0	0	

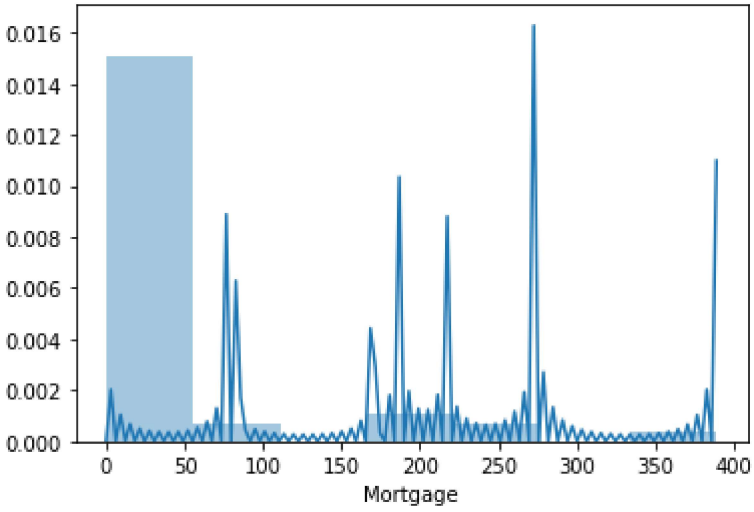
In [43]:
```
distPlotList_up = ["Age", "Experience", "Income", "CCAvg", "Mortgage"]
countPlotList_up = [ "Family", "Education", "Personal_Loan",
                    "Securities_Account", "CD_Account", "Online", "CreditCard"]

for i in distPlotList_up:
    sns.distplot(wrong_prediction_FP[i])
    plt.show()

for i in countPlotList_up:
    sns.countplot(data[i])
    plt.show()
```

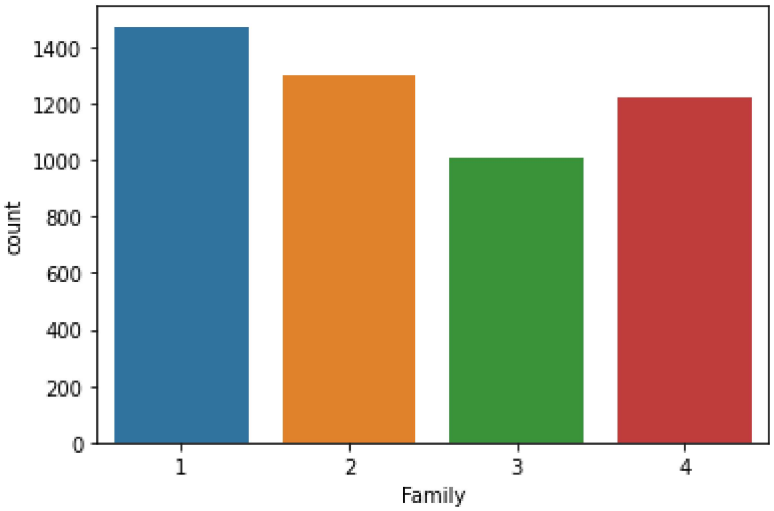

0.030
0.025
0.020
0.015
0.010
0.005
0.000
10
20
30
40
50
60
70
80
Age

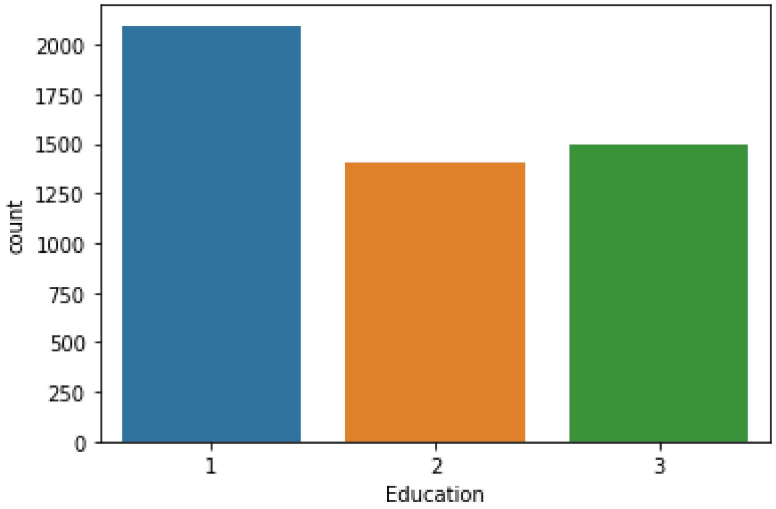

0.030
0.025
0.020
0.015
0.010
0.005
0.000
−10
0
10
20
30
40
50
Experience

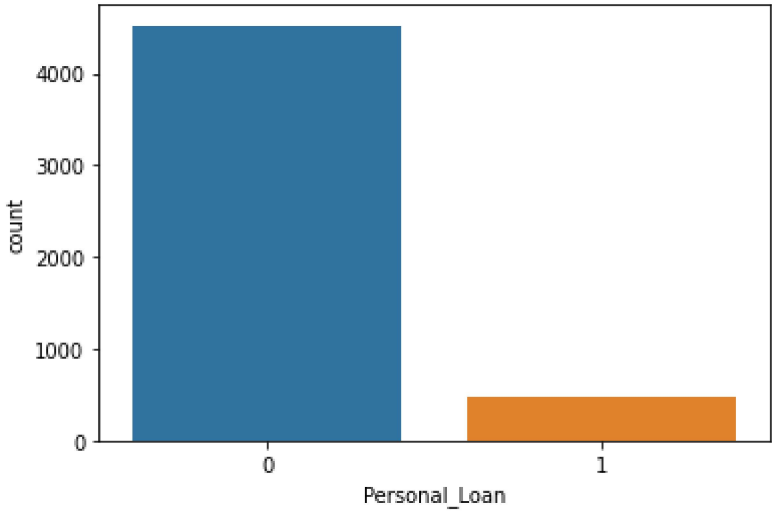

0.030
0.025
0.020
0.015
0.010
0.005
0.000
60
70
80
90
100
110
120
130
Income

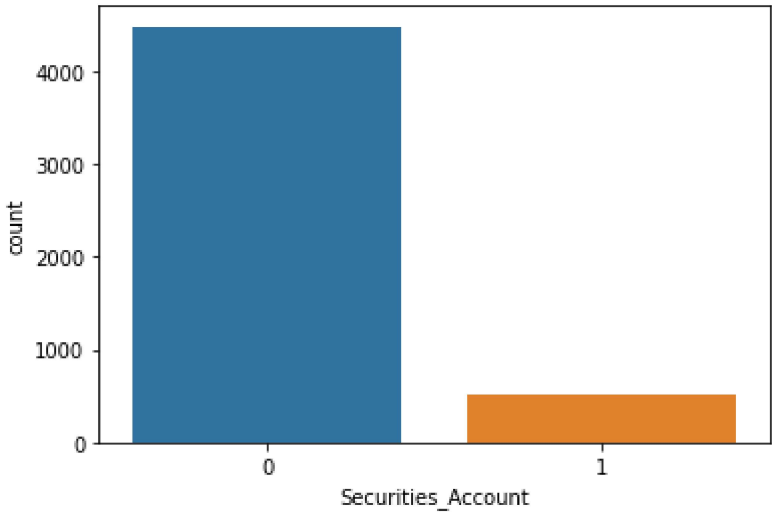

0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8
CCAvg

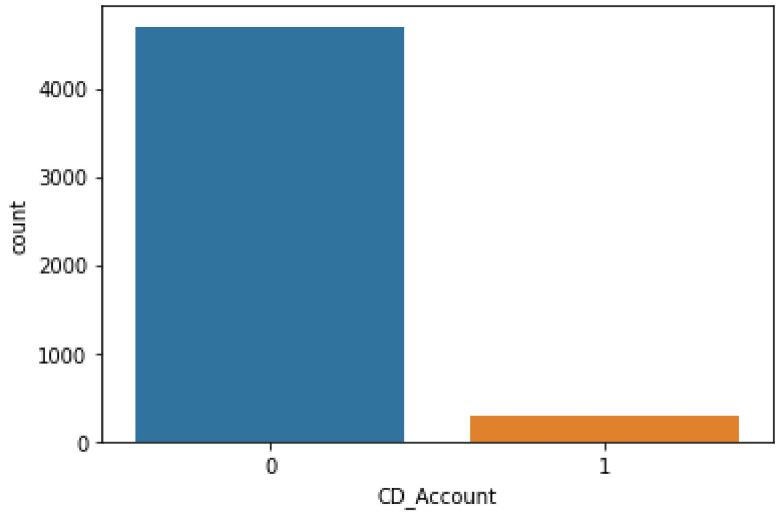

0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
50
100
150
200
250
300
350
400
Mortgage

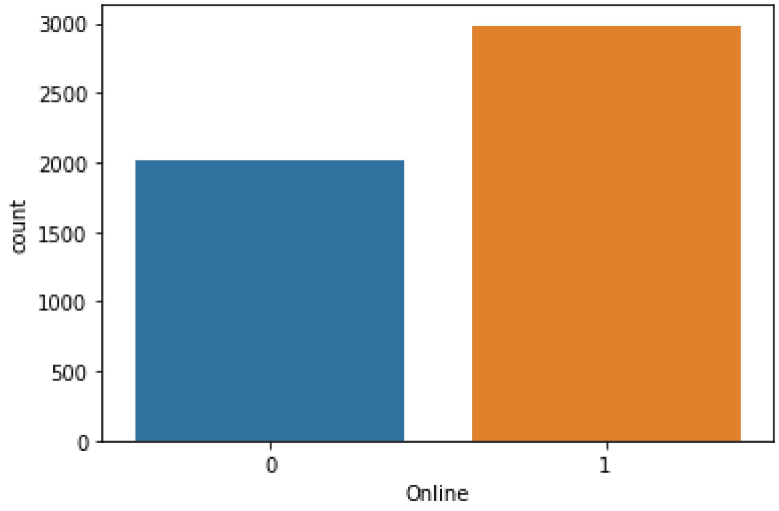

1400
1200
1000
800
600
400
200
0
count
1
2
3
4
Family

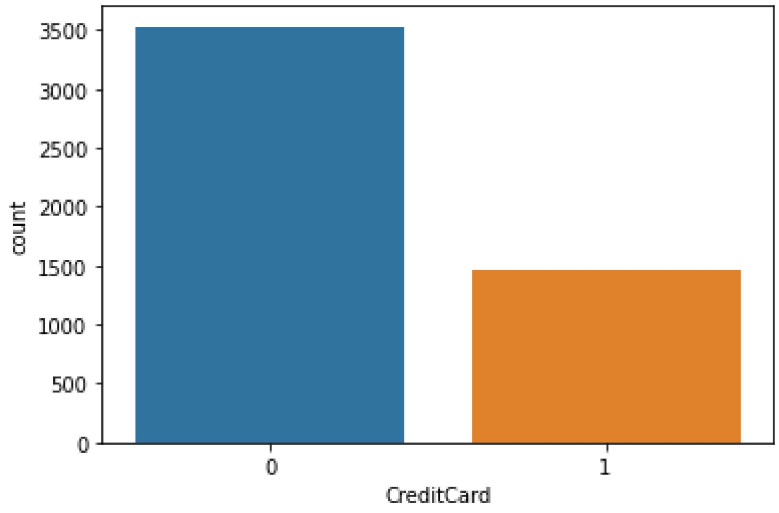

2000
1750
1500
1250
1000
750
500
250
0
count
1
2
3
Education


4000
3000
2000
1000
0
count
0
1
Personal_Loan


4000
3000
2000
1000
0
count
0
1
Securities_Account







As expected, the noise does not particularly have pattern as it is nature/definition of noise.

Conclusion:

After carefully reviewing the Loan data and model, we discovered:

1. The most important variable in deciding personal loan is **Income**
2. In order to reduce data entry cost, It is okay to remove variables-["ZIPCode", "Securities_Account", "CD_Account", "Online", "CreditCard"]
3. It is still important to introduce the customers to purchase loan. Opportunity lost seems to be more costly in this case.

In [ ]: