In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
df = pd.read_csv('churn_prediction[1].csv')
```

In [3]:
```
df.shape
```

Out[3]: (28382, 21)

In [4]:
```
df.tail()
```

Out[4]:

	customer_id	vintage	age	gender	dependents	occupation	city	customer_nw_cate
28377	30297	1845	10	Female	0.0	student	1020.0	
28378	30298	4919	34	Female	0.0	self_employed	1046.0	
28379	30299	297	47	Male	0.0	salaried	1096.0	
28380	30300	2585	50	Male	3.0	self_employed	1219.0	
28381	30301	2349	18	Male	0.0	student	1232.0	

5 rows × 21 columns

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

Out[5]:

	customer_id	vintage	age	gender	dependents	occupation	city	customer_nw_category
0	1	3135	66	Male	0.0	self_employed	187.0	2
1	2	310	35	Male	0.0	self_employed	NaN	2
2	4	2356	31	Male	0.0	salaried	146.0	2
3	5	478	90	NaN	NaN	self_employed	1020.0	2
4	6	2531	42	Male	2.0	self_employed	1494.0	3

5 rows × 21 columns

In [6]:
```
df.dtypes
```

Out[6]:
```
customer_id                      int64
vintage                          int64
age                              int64
gender                          object
dependents                     float64
occupation                      object
city                           float64
customer_nw_category             int64
branch_code                      int64
days_since_last_transaction    float64
current_balance                float64
previous_month_end_balance     float64
average_monthly_balance_prevQ  float64
average_monthly_balance_prevQ2 float64
current_month_credit           float64
previous_month_credit          float64
current_month_debit            float64
previous_month_debit           float64
current_month_balance          float64
previous_month_balance         float64
churn                            int64
dtype: object
```

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

Out[7]:

	customer_id	vintage	age	dependents	city	customer_nw_cate
count	28382.000000	28382.000000	28382.000000	25919.000000	27579.000000	28382.00
mean	15143.508667	2364.336446	48.208336	0.347236	796.109576	2.22
std	8746.454456	1610.124506	17.807163	0.997661	432.872102	0.66
min	1.000000	180.000000	1.000000	0.000000	0.000000	1.00
25%	7557.250000	1121.000000	36.000000	0.000000	409.000000	2.00
50%	15150.500000	2018.000000	46.000000	0.000000	834.000000	2.00
75%	22706.750000	3176.000000	60.000000	0.000000	1096.000000	3.00
max	30301.000000	12899.000000	90.000000	52.000000	1649.000000	3.00

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
customer_id                      0
vintage                          0
age                              0
gender                         525
dependents                    2463
occupation                      80
city                           803
customer_nw_category             0
branch_code                      0
days_since_last_transaction   3223
current_balance                  0
previous_month_end_balance       0
average_monthly_balance_prevQ    0
average_monthly_balance_prevQ2   0
current_month_credit             0
previous_month_credit            0
current_month_debit              0
previous_month_debit             0
current_month_balance            0
previous_month_balance           0
churn                            0
dtype: int64
```

In [9]:
```
df['gender'].fillna(df['gender'].mode()[0], inplace= True)
```

In [10]:
```
df['city'].fillna(df['city'].mean(), inplace= True)
```

In [11]:
```
df['dependents'].fillna(df['dependents'].mean(), inplace= True)
```

In [12]:
```
df['occupation'].fillna(df['occupation'].mode()[0], inplace= True)
```

In [13]:
```
df['days_since_last_transaction'].fillna(df['days_since_last_transaction'].mea
n(), inplace= True)
```

In [14]:
```
df.isnull().sum()
```

Out[14]:
```
customer_id                    0
vintage                        0
age                            0
gender                         0
dependents                     0
occupation                     0
city                           0
customer_nw_category           0
branch_code                    0
days_since_last_transaction    0
current_balance                0
previous_month_end_balance     0
average_monthly_balance_prevQ  0
average_monthly_balance_prevQ2 0
current_month_credit           0
previous_month_credit          0
current_month_debit            0
previous_month_debit           0
current_month_balance          0
previous_month_balance         0
churn                          0
dtype: int64
```

In [15]:
```
df.drop_duplicates()
```

Out[15]:

	customer_id	vintage	age	gender	dependents	occupation	city	customer_nw
0	1	3135	66	Male	0.000000	self_employed	187.000000	
1	2	310	35	Male	0.000000	self_employed	796.109576	
2	4	2356	31	Male	0.000000	salaried	146.000000	
3	5	478	90	Male	0.347236	self_employed	1020.000000	
4	6	2531	42	Male	2.000000	self_employed	1494.000000	
...	...	...	...	...	...	...	...	
28377	30297	1845	10	Female	0.000000	student	1020.000000	
28378	30298	4919	34	Female	0.000000	self_employed	1046.000000	
28379	30299	297	47	Male	0.000000	salaried	1096.000000	
28380	30300	2585	50	Male	3.000000	self_employed	1219.000000	
28381	30301	2349	18	Male	0.000000	student	1232.000000	

28382 rows × 21 columns

In [16]:
```
df.dropna()
```

Out[16]:

	customer_id	vintage	age	gender	dependents	occupation	city	customer_nw
0	1	3135	66	Male	0.000000	self_employed	187.000000	
1	2	310	35	Male	0.000000	self_employed	796.109576	
2	4	2356	31	Male	0.000000	salaried	146.000000	
3	5	478	90	Male	0.347236	self_employed	1020.000000	
4	6	2531	42	Male	2.000000	self_employed	1494.000000	
...	...	...	...	...	...	...	...	
28377	30297	1845	10	Female	0.000000	student	1020.000000	
28378	30298	4919	34	Female	0.000000	self_employed	1046.000000	
28379	30299	297	47	Male	0.000000	salaried	1096.000000	
28380	30300	2585	50	Male	3.000000	self_employed	1219.000000	
28381	30301	2349	18	Male	0.000000	student	1232.000000	

28382 rows × 21 columns

In [17]:
```
df.head()
```

Out[17]:

	customer_id	vintage	age	gender	dependents	occupation	city	customer_nw_cat
0	1	3135	66	Male	0.000000	self_employed	187.000000	
1	2	310	35	Male	0.000000	self_employed	796.109576	
2	4	2356	31	Male	0.000000	salaried	146.000000	
3	5	478	90	Male	0.347236	self_employed	1020.000000	
4	6	2531	42	Male	2.000000	self_employed	1494.000000	

5 rows × 21 columns

In [18]:
```
df.tail()
```

Out[18]:

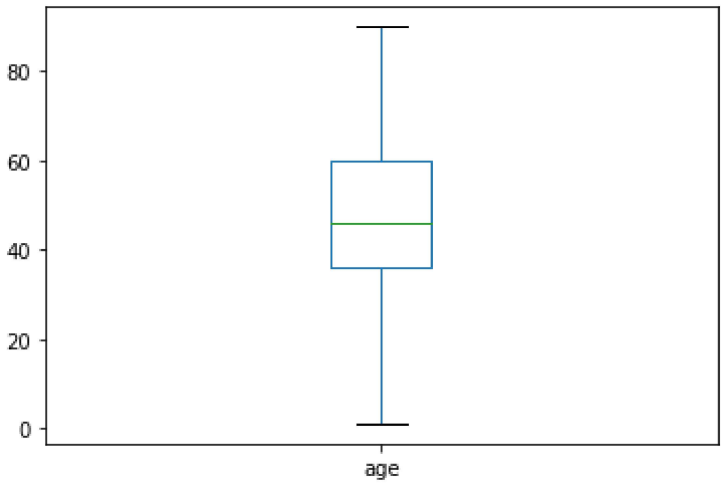
	customer_id	vintage	age	gender	dependents	occupation	city	customer_nw_cate
28377	30297	1845	10	Female	0.0	student	1020.0	
28378	30298	4919	34	Female	0.0	self_employed	1046.0	
28379	30299	297	47	Male	0.0	salaried	1096.0	
28380	30300	2585	50	Male	3.0	self_employed	1219.0	
28381	30301	2349	18	Male	0.0	student	1232.0	

5 rows × 21 columns

In [ ]:
```
```

In [19]:
```
df['age'].plot.box()
```

Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x233646334c8>
```



In [20]:
```
df.columns
```

Out[20]:
```
Index(['customer_id', 'vintage', 'age', 'gender', 'dependents', 'occupation',
       'city', 'customer_nw_category', 'branch_code',
       'days_since_last_transaction', 'current_balance',
       'previous_month_end_balance', 'average_monthly_balance_prevQ',
       'average_monthly_balance_prevQ2', 'current_month_credit',
       'previous_month_credit', 'current_month_debit', 'previous_month_debi
t',
       'current_month_balance', 'previous_month_balance', 'churn'],
      dtype='object')
```

In [21]:
```
df= pd.get_dummies(df)
```

In [22]:
```
x = df.drop(['churn'], axis=1)
y = df['churn']
```

In [23]:
```
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y= train_test_split(x,y ,random_state= 42, stra
tify= y, )
from sklearn.preprocessing import MinMaxScaler
scaler =MinMaxScaler()
```

In [24]:
```
train_x_scaled = scaler.fit_transform(train_x)
train_x_scaled = pd.DataFrame(train_x_scaled )
train_x_scaled.head()
```

Out[24]:

	0	1	2	3	4	5	6	7	8	9
0	0.368548	0.058417	0.494382	0.000000	0.483617	0.5	0.340933	0.035616	0.003551	0.00172
1	0.383069	0.021621	0.370787	0.038462	0.101942	0.5	0.709893	0.191781	0.001105	0.00079
2	0.781485	0.059124	0.460674	0.006678	0.040049	0.5	0.298264	0.191775	0.001349	0.00148
3	0.141056	0.312603	0.797753	0.000000	0.618932	0.5	0.074252	0.000000	0.001184	0.00080
4	0.031122	0.004010	0.516854	0.006678	0.838592	0.5	0.131353	0.016438	0.000994	0.00197

5 rows × 25 columns

In [25]:
```
test_x_scaled = scaler.transform(test_x)
test_x_scaled = pd.DataFrame(test_x_scaled)
test_x_scaled.head()
```

Out[25]:

	0	1	2	3	4	5	6	7	8	9
0	0.060561	0.147810	0.629213	0.038462	0.884709	1.0	0.344279	0.057534	0.001499	0.00118
1	0.827360	0.058023	1.000000	0.006678	0.311286	0.5	0.044970	0.920548	0.001282	0.00090
2	0.705446	0.140184	0.348315	0.000000	0.009102	0.5	0.001046	0.191775	0.001103	0.00084
3	0.589802	0.167545	0.775281	0.038462	0.348908	0.5	0.291362	0.000000	0.001848	0.00143
4	0.205545	0.030270	0.528090	0.006678	0.483076	1.0	0.036394	0.063014	0.002262	0.00154

5 rows × 25 columns

LOGISTIC REGRESSION

In [26]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score

lr= LogisticRegression()
lr.fit(train_x,train_y)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:
940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[26]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [27]:
```
train_predict= lr.predict(train_x)
train_predict
```

Out[27]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

F1_score

In [28]:
```
k= f1_score(train_predict, train_y)
print('Training score', k)
```

```
Training score 0.16410488245931285
```

In [29]:
```
test_predict1=lr.predict(test_x)
test_predict1
```

Out[29]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [30]:
```
k = f1_score(test_predict1, test_y)
print('Test f1_score', k)
```

```
Test f1_score 0.15945945945945944
```

auc-roc

In [31]:
```
from sklearn.metrics import roc_auc_score
```

In [32]:
```
train_predict= lr.predict_proba(train_x)
train_predict
```

Out[32]:
```
array([[0.8981373 , 0.1018627 ],
       [0.71851558, 0.28148442],
       [0.64856994, 0.35143006],
       ...,
       [0.71743673, 0.28256327],
       [0.77400649, 0.22599351],
       [0.82189461, 0.17810539]])
```

In [33]:
```
test_predict= lr.predict_proba(test_x)
```

In [34]:
```
roc_auc_score(train_y, train_predict[:,1])
```

Out[34]:
```
0.7153420222365816
```

In [35]:
```
roc_auc_score(test_y, test_predict[:,1])
```

Out[35]:
```
0.7181972148173873
```

In [36]:
```
lr.score(train_x,train_y)
```

Out[36]:
```
0.8262707883115663
```

lr test score

In [37]:
```
test_predict0= lr.predict(test_x)
test_predict0[:10], lr.score(test_x,test_y)
```

Out[37]:
```
(array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64), 0.8246899661781285)
```

KNEIGHBORSCLASSIFIER

In [38]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [39]:
```
clf = KNeighborsClassifier()
```

In [40]:
```
clf.fit(train_x, train_y)
```

Out[40]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [41]:
```
train_predict= clf.predict(train_x)
k = f1_score(train_predict, train_y)
print('Training score',k)
```

```
Training score 0.5735341236783082
```

In [42]:
```
test_predict= clf.predict(test_x)
k= f1_score(test_predict, test_y)
print('test_f1score',k)
```

```
test_f1score 0.44929645803008245
```

In [ ]:
```

```

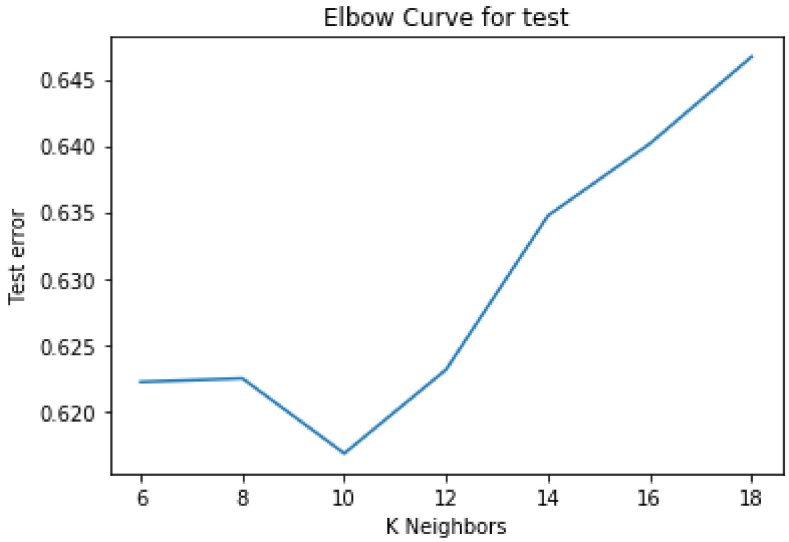
In [43]:
```
def Elbow(K):
    Test_Error=[]
    for i in K:
        clf = KNeighborsClassifier(n_neighbors = i)
        clf.fit(train_x,train_y)
        temp= clf.predict(test_x)
        temp= f1_score(temp, test_y)
        error= 1-temp
        Test_Error.append(error)
    return Test_Error
```

In [44]:
```
k = range(6, 20, 2)
```

In [45]:
```
Test= Elbow(k)
```

In [46]:
```
plt.plot(k, Test)
plt.xlabel('K Neighbors')
plt.ylabel('Test error')
plt.title('Elbow Curve for test')
```

Out[46]:
```
Text(0.5, 1.0, 'Elbow Curve for test')
```



In [47]:
```
clf= KNeighborsClassifier(n_neighbors=10)
clf.fit(train_x,train_y)
test_predict=clf.predict(test_x)
k1= f1_score(test_predict, test_y)
train_predict= clf.predict(train_x)
k2= f1_score(train_predict,train_y)
print(k1)
print(k2)
```

```
0.38321369409166206
0.442116868798236
```

Knn test score

In [48]:
```
test_predict1=clf.predict(test_x)
test_predict1[:10],clf.score(test_x,test_y)
```

Out[48]:
```
(array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64), 0.8425873731679819)
```

In [49]:
```
test_y.value_counts(normalize=True)
```

Out[49]:
```
0    0.814684
1    0.185316
Name: churn, dtype: float64
```

Cross validation(k-fold)

In [50]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
k_fold = KFold(n_splits=10, shuffle=True, random_state=0)
```

In [51]:
```
clf = KNeighborsClassifier(n_neighbors = 11)
scoring = 'accuracy'
score = cross_val_score(clf, x, y, cv=k_fold, n_jobs=1, scoring=scoring)
print(score)
```

```
[0.83233533 0.8436069  0.84883721 0.85306554 0.84143763 0.84813249
 0.83897111 0.84883721 0.82804792 0.84319944]
```

In [54]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [55]:
```
dt_model=DecisionTreeClassifier(random_state = 10)
```

In [56]:
```
dt_model.fit(train_x, train_y)
```

Out[56]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=Non
e,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=10, splitter='best')
```

In [57]:
```
train_y.value_counts(normalize=True)
```

Out[57]:
```
0    0.814667
1    0.185333
Name: churn, dtype: float64
```

In [58]:
```
test_y.value_counts(normalize=True)
```

Out[58]:
```
0    0.814684
1    0.185316
Name: churn, dtype: float64
```

In [59]:
```
train_score = dt_model.score(train_x, train_y)
```

In [60]:
```
test_score= dt_model.score(test_x, test_y)
```

In [61]:
```
print(train_score, test_score)
```

```
1.0 0.7891770011273957
```

In [62]:
```
dt_model.predict(test_x)
```

Out[62]:
```
array([0, 0, 1, ..., 1, 1, 0], dtype=int64)
```

In [63]:
```
dt_model.predict_proba(test_x)
```

Out[63]:
```
array([[1., 0.],
       [1., 0.],
       [0., 1.],
       ...,
       [0., 1.],
       [0., 1.],
       [1., 0.]])
```

In [64]:
```
y_pred = dt_model.predict_proba(test_x)[:,1]
```

In [65]:
```
y_new = []
for i in range(len(y_pred)):
    if y_pred[i]<=0.7:
        y_new.append(0)
    else:
        y_new.append(1)
```

In [66]:
```
from sklearn.metrics import accuracy_score
```

In [67]:
```
accuracy_score(test_y, y_new)
```

Out[67]: 0.7891770011273957

In [68]:
```
train_accuracy=[]
test_accuracy = []
for depth in range(1,10):
    dt_model = DecisionTreeClassifier(max_depth= depth, random_state=10)
    dt_model.fit(train_x, train_y)
    train_accuracy.append(dt_model.score(train_x,train_y))
    test_accuracy.append(dt_model.score(test_x, test_y))
```

In [69]:
```
frame= pd.DataFrame({'max_depth':range(1,10), 'train_acc':train_accuracy,'test
_acc':test_accuracy})
frame.head()
```

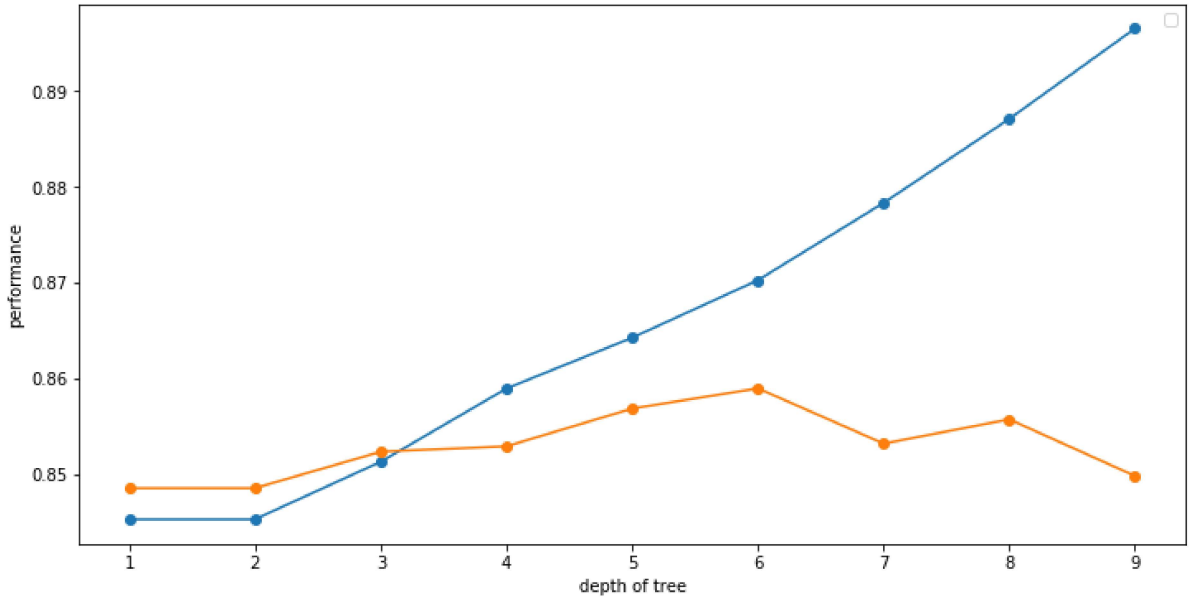
Out[69]:

	max_depth	train_acc	test_acc
0	1	0.845250	0.848506
1	2	0.845250	0.848506
2	3	0.851264	0.852311
3	4	0.858921	0.852875
4	5	0.864230	0.856821

In [70]:
```
plt.figure(figsize=(12,6))
plt.plot(frame['max_depth'], frame['train_acc'], marker='o')
plt.plot(frame['max_depth'], frame['test_acc'], marker='o')
plt.xlabel('depth of tree')
plt.ylabel('performance')
plt.legend()
```

```
No handles with labels found to put in legend.
```

Out[70]:
```
<matplotlib.legend.Legend at 0x233783fba08>
```



In [71]:
```
dt_model=DecisionTreeClassifier(max_depth = 5, max_leaf_nodes=20)
dt_model.fit(train_x, train_y)
dt_model.score(train_x, train_y)
dt_model.score(test_x, test_y)
```

Out[71]:
```
0.8565388951521984
```

In [72]:
```
dt_model.score(train_x,train_y)
```

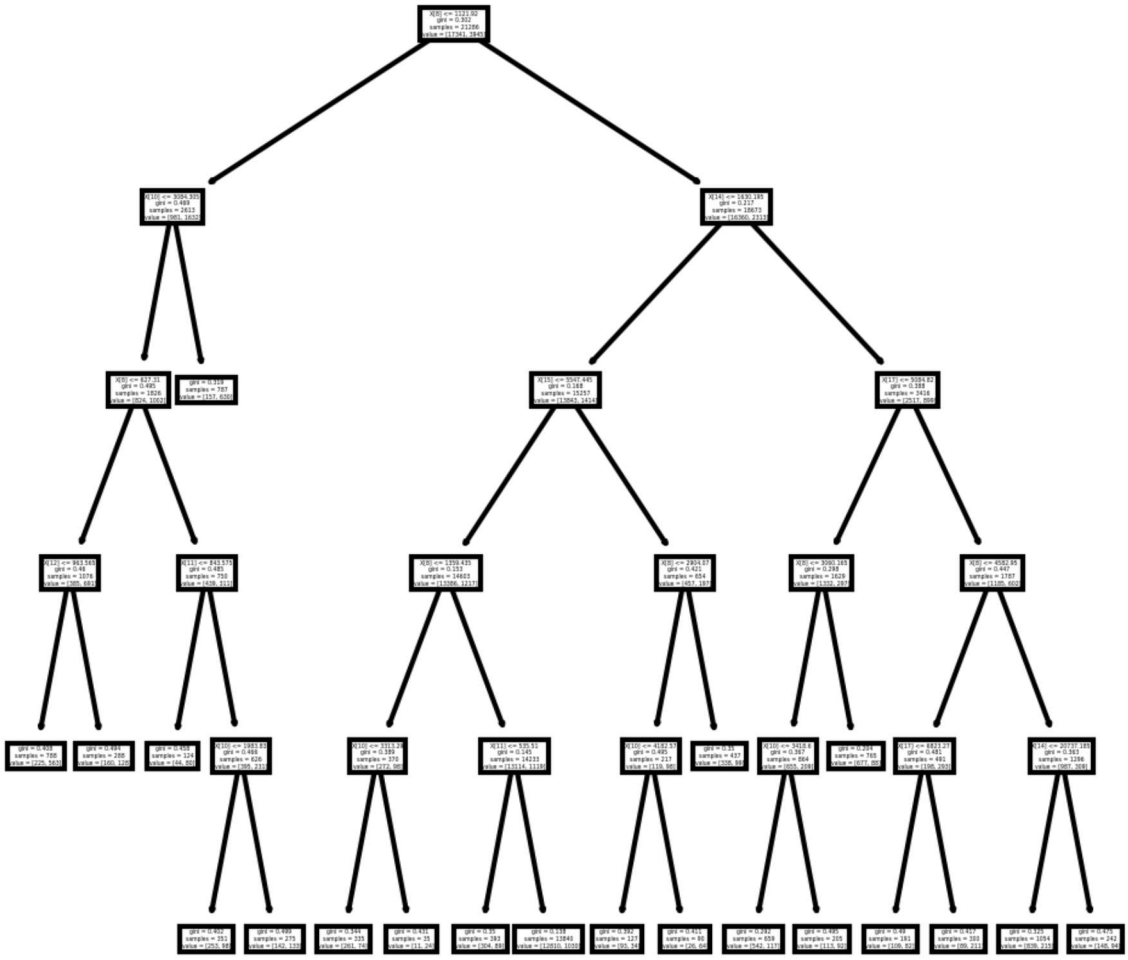
Out[72]:
```
0.8625857371042
```

In [73]:
```
pred3= dt_model.score(test_x, test_y)
pred3
```

Out[73]:
```
0.8565388951521984
```

In [74]:
```
from sklearn import tree
```

In [75]:
```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
tree.plot_tree(dt_model);
fig.savefig('tree.png')
```



In [76]:
```
!pip install graphviz
```

```
Requirement already satisfied: graphviz in c:\programdata\anaconda3\lib\site-
packages (0.14)
```

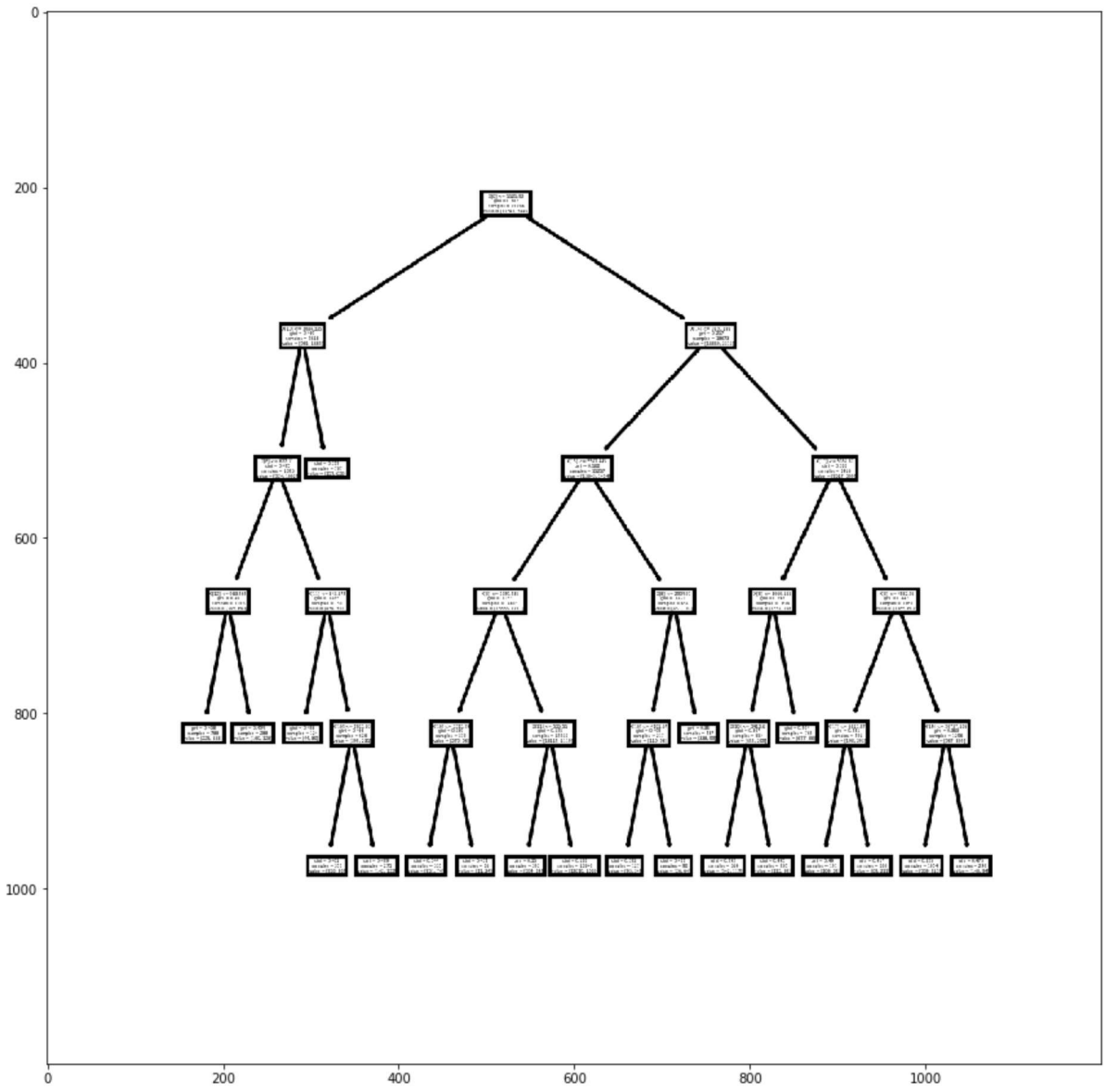
In [77]:
```
decission_tree= tree.export_graphviz(dt_model, out_file='tree.dot', feature_na
mes= train_x.columns, max_depth=2, filled=True)
```

In [78]:
```
!dot -Tpng tree.dot -o tree.png
```

```
'dot' is not recognized as an internal or external command,
operable program or batch file.
```

In [79]:
```
image = plt.imread('tree.png')
plt.figure(figsize=(15,15))
plt.imshow(image)
```

Out[79]: <matplotlib.image.AxesImage at 0x23378632f88>



Decisiontree test score

In [80]:
```
test_predict2=dt_model.predict(test_x)
test_predict2[:10], dt_model.score(test_x, test_y)
```

Out[80]: (array([0, 0, 1, 0, 0, 0, 1, 0, 0, 0], dtype=int64), 0.8565388951521984)

In [81]:
```
# df = pd.DataFrame(columns=['M1', 'M2', 'M3', 'Actual'])
# df['M1'] = test_pred0
# df['M2'] = test_pred1
# df['M3'] = test_pred2
# df['Actual'] = np.array(test_y)
```

In [82]:
```
from statistics import mode
final_pred = np.array([])
for i in range(0,len(test_x)):
    final_pred = np.append(final_pred, mode([test_predict0[i], test_predict1[i
],test_predict2[i]]))
```

In [83]:
```
from sklearn.metrics import accuracy_score
```

In [84]:
```
accuracy_score(test_y, final_pred)
```

Out[84]:
```
0.8458286358511837
```

In [85]:
```
accuracy_score(test_y, test_predict0), accuracy_score(test_y, test_predict1),
accuracy_score(test_y, test_predict2)
```

Out[85]:
```
(0.8246899661781285, 0.8425873731679819, 0.8565388951521984)
```

In [86]:
```
test_y['churn'] = prediction
submission = pd.DataFrame(test_x['customer_id'],test_y['churn'])
submission.to_csv("Submission.csv")
```