CHURN PREDICTION ANALYSIS FOR BLUE-VALLEY BANK

Overview

At the Blue-Valley Bank,we noticed a worrying trend that clients were leaving and by the time this was realized it was too late to keep them.As the senior data scientist ,i led and investigation into five years of operational data to understand the root cause of these losses.By exploring features in the data uncovering of hidden signals that indicated customers churning was indentified

Using these insights we built a predictive model to segment customers by their churn risk that is high,moderate and low.Through this the model transformed the bank's approach from reacting after the fact to proactively reaching out to vulnerable clients.Now armed with the model built the Blue valley bank is at a better position to retain valuable clients and prevent losses

Business Understanding

In the blue valley bank it has been observed that churning of clients is one of the highest things that is causing the financial loss of the bank and by doing the project we will be able to identify which features are causing churn and segmenting them using the

Problem Statement

Blue-Valley Bank has faced rising customer churn over the past five years,resulting in significant financial losses.The bank lacked a way to predict which customers were at risk of leavinf early enough to intervene.This project focuses on analyzing customer data to build a suitable predictive model that identifies high risk churners and supports targeted retention efforts

Main Objective

How can we predict which clients of Blue-Valley bank are likely to churn and how can we segment them based on their churn risk using an appropriate predictive model?

Specific Objective

- Which features most influence features will influence churn?
- Which predictive model between logistic and decision tree will most effectively identify clients likely to churn in terms of performance?
- How can clients be segmented into high,moderate and low churn risk categories based on their predicated probability of churning?

DATA UNDERSTANDING

The source of my data set is from kaggle
https://www.kaggle.com/datasets/pentakrishnakishore/bank-customer-churn-data

My dataset is relevant since it has columns like age ,vintage which shows how long a client has stayed in a bank,customer net worth,last transaction which when doing feature engineering we can check how many days since the last transaction was done,we have dependents these are the

people rellying upon the client,we have occupation what the client does we have the monthly ,average and current balances that helps us gauge what balance the clients have.

we first import libraries, load the dataset and the understand our data

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler,OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_sc
from imblearn.over_sampling import SMOTE
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report,confusion_matrix,roc_curve,
from sklearn.metrics import roc_curve, auc
```

In [2]:
```
#Loading the dataset
df=pd.read_csv('churn_prediction.csv')
df
```

Out[2]:

	customer_id	vintage	age	gender	dependents	occupation	city	customer_nw_cat
0	1	2101	66	Male	0.0	self_employed	187.0	
1	2	2348	35	Male	0.0	self_employed	NaN	
2	4	2194	31	Male	0.0	salaried	146.0	
3	5	2329	90	NaN	NaN	self_employed	1020.0	
4	6	1579	42	Male	2.0	self_employed	1494.0	
...	...	...	...	...	...	...	...	
28377	30297	2325	10	Female	0.0	student	1020.0	
28378	30298	1537	34	Female	0.0	self_employed	1046.0	
28379	30299	2376	47	Male	0.0	salaried	1096.0	
28380	30300	1745	50	Male	3.0	self_employed	1219.0	
28381	30301	1175	18	Male	0.0	student	1232.0	

28382 rows × 21 columns

In [3]:
```
#check data infomation and preview
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28382 entries, 0 to 28381
Data columns (total 21 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   customer_id                     28382 non-null  int64  
 1   vintage                         28382 non-null  int64  
 2   age                             28382 non-null  int64  
 3   gender                          27857 non-null  object 
 4   dependents                      25919 non-null  float64
 5   occupation                      28302 non-null  object 
 6   city                            27579 non-null  float64
 7   customer_nw_category            28382 non-null  int64  
 8   branch_code                     28382 non-null  int64  
 9   current_balance                 28382 non-null  float64
 10  previous_month_end_balance      28382 non-null  float64
 11  average_monthly_balance_prevQ   28382 non-null  float64
 12  average_monthly_balance_prevQ2  28382 non-null  float64
 13  current_month_credit            28382 non-null  float64
 14  previous_month_credit           28382 non-null  float64
 15  current_month_debit             28382 non-null  float64
 16  previous_month_debit            28382 non-null  float64
 17  current_month_balance           28382 non-null  float64
 18  previous_month_balance          28382 non-null  float64
 19  churn                           28382 non-null  int64  
 20  last_transaction                28382 non-null  object 
dtypes: float64(12), int64(6), object(3)
memory usage: 4.5+ MB
```

We have 28382 entries with 21 columns 6 integer columns 3 objects and 11 float columns

In [4]:
```
# we also check the descriptive analysis
df.describe()
```

Out[4]:

	customer_id	vintage	age	dependents	city	customer_nw_ca
count	28382.000000	28382.000000	28382.000000	25919.000000	27579.000000	28382.(
mean	15143.508667	2091.144105	48.208336	0.347236	796.109576	2.2
std	8746.454456	272.676775	17.807163	0.997661	432.872102	0.(
min	1.000000	73.000000	1.000000	0.000000	0.000000	1.(
25%	7557.250000	1958.000000	36.000000	0.000000	409.000000	2.(
50%	15150.500000	2154.000000	46.000000	0.000000	834.000000	2.(
75%	22706.750000	2292.000000	60.000000	0.000000	1096.000000	3.(
max	30301.000000	2476.000000	90.000000	52.000000	1649.000000	3.(

DATA PREPARATION

1. Handle missing values
2. check for duplicates and remove
3. check for outliers

In [5]:
```
# we will remove check for missing values and remove them
df.isna().sum().sum()
```

Out[5]:
```
3871
```

In [6]:
```
#lets see the columns that have missing values
df.isna().sum()
```

Out[6]:
```
customer_id                        0
vintage                            0
age                                0
gender                           525
dependents                      2463
occupation                        80
city                             803
customer_nw_category               0
branch_code                        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
last_transaction                   0
dtype: int64
```

For the missing values since the values have few missing values we will drop them the highest has 2463

In [7]:
```
#df.dropna(subset=['gender','occupation','city','dependents'],inplace=True)
df.dropna(inplace=True)
```

In [8]:
```
#check if the missing values are gone for the three columns
df.isna().sum()
```

Out[8]:
```
customer_id                    0
vintage                        0
age                            0
gender                         0
dependents                     0
occupation                     0
city                           0
customer_nw_category           0
branch_code                    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
last_transaction               0
dtype: int64
```

In [9]:
```
df.head()#we check if there is another column that still has missing values
```

Out[9]:

	customer_id	vintage	age	gender	dependents	occupation	city	customer_nw_categor
0	1	2101	66	Male	0.0	self_employed	187.0	
2	4	2194	31	Male	0.0	salaried	146.0	
4	6	1579	42	Male	2.0	self_employed	1494.0	
5	7	1923	42	Female	0.0	self_employed	1096.0	
6	8	2048	72	Male	0.0	retired	1020.0	

5 rows × 21 columns

In [10]:
```
#change age to integer
df['age'].astype(int)
```

Out[10]:
```
0        66
2        31
4        42
5        42
6        72
         ..
28377    10
28378    34
28379    47
28380    50
28381    18
Name: age, Length: 24832, dtype: int32
```

i noticed for the last transaction it had missing values and thats why it was not being detected so i had to convert to date time format and then drop the missing values since its 9.7% of the 28382 entries which is quite few

In [11]:
```
# we will try to convert the column to right date time we see if it will see
df['last_transaction']=pd.to_datetime(df['last_transaction'],format='%Y/%m/%d
df.head()
```

Out[11]:

	customer_id	vintage	age	gender	dependents	occupation	city	customer_nw_categor
0	1	2101	66	Male	0.0	self_employed	187.0	
2	4	2194	31	Male	0.0	salaried	146.0	
4	6	1579	42	Male	2.0	self_employed	1494.0	
5	7	1923	42	Female	0.0	self_employed	1096.0	
6	8	2048	72	Male	0.0	retired	1020.0	

5 rows × 21 columns

In [12]:
```
# we will check the missing values
df['last_transaction'].isna().sum().sum()
```

Out[12]:
```
2765
```

In [13]:
```
df['last_transaction'].unique()
```

Out[13]:
```
array(['2019-05-21T00:00:00.000000000',                           'NaT',
       '2019-11-03T00:00:00.000000000', '2019-11-01T00:00:00.000000000',
       '2019-09-24T00:00:00.000000000', '2019-07-12T00:00:00.000000000',
       '2019-12-12T00:00:00.000000000', '2019-12-31T00:00:00.000000000',
       '2019-12-26T00:00:00.000000000', '2019-12-11T00:00:00.000000000',
       '2019-11-14T00:00:00.000000000', '2019-03-03T00:00:00.000000000',
       '2019-11-09T00:00:00.000000000', '2019-12-28T00:00:00.000000000',
       '2019-10-23T00:00:00.000000000', '2019-11-30T00:00:00.000000000',
       '2019-12-08T00:00:00.000000000', '2019-12-18T00:00:00.000000000',
       '2019-12-27T00:00:00.000000000', '2019-12-22T00:00:00.000000000',
       '2019-12-04T00:00:00.000000000', '2019-11-25T00:00:00.000000000',
       '2019-08-28T00:00:00.000000000', '2019-10-15T00:00:00.000000000',
       '2019-12-21T00:00:00.000000000', '2019-09-25T00:00:00.000000000',
       '2019-12-05T00:00:00.000000000', '2019-09-22T00:00:00.000000000',
       '2019-12-24T00:00:00.000000000', '2019-12-20T00:00:00.000000000',
       '2019-07-24T00:00:00.000000000', '2019-12-17T00:00:00.000000000',
       '2019-09-27T00:00:00.000000000', '2019-12-14T00:00:00.000000000',
       '2019-09-06T00:00:00.000000000', '2019-10-16T00:00:00.000000000',
       '2019-12-13T00:00:00.000000000', '2019-12-03T00:00:00.000000000',
```

In [14]:
```
df.dropna(subset=['last_transaction'],inplace=True)
```

In [15]:
```
df['last_transaction'].isna().sum().sum()#the null values in the last_transac
```

Out[15]:
```
0
```

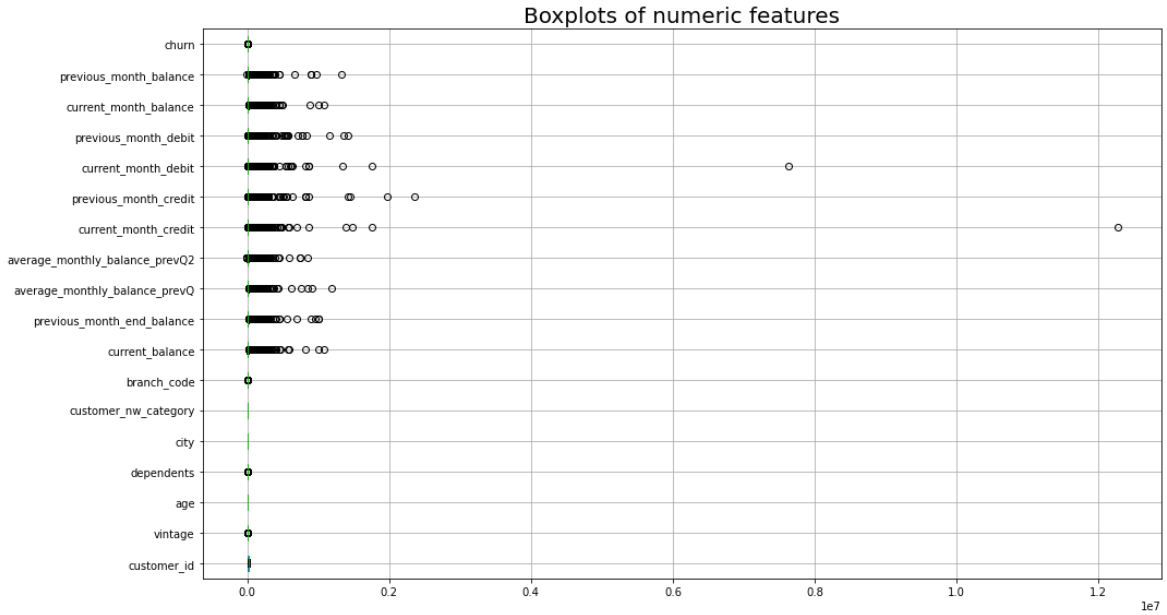
Next step is checking for duplicates

In [16]:
```
df.duplicated().sum()# no duplicates present
```

Out[16]:
```
0
```

Next step checking for outliers

In [17]:
```
df.select_dtypes(include='number').boxplot(figsize=(15,8),vert=False)
plt.title("Boxplots of numeric features",fontsize=20)
plt.tight_layout()
plt.show()
```



As you can see majority here have outliers so i will try to remove the outliers so as to prevent biasness in the dataset

In [18]:
```
def outliers(df,col):
    upper_limit=df[col].mean() +3 *df[col].std()
    lower_limit=df[col].mean()-3 * df[col].std()
#create a list that will store the index of the outliers
    #ls=df.index[(df[cols] <lower_limit) | (df[cols] >upper_limit)]
    ls= df[(df[col] < lower_limit) | (df[col] > upper_limit)].index.tolist()

    return ls
```

In [19]:
```
index_list=[]

columns = ['current_balance', 'average_monthly_balance_prevQ',
           'average_monthly_balance_prevQ2', 'current_month_credit',
           'previous_month_credit', 'current_month_debit',
           'previous_month_debit', 'current_month_balance',
           'previous_month_balance']
for col in columns:
    index_list.extend(outliers(df,col))
    index_list=list(set(index_list))
```

In [20]:
```
df_cleaned=df.drop(index=index_list)
```

In [21]:
```
df_cleaned=df
```

EXPLORATIVE DATA ANALYSIS

1. check for skeweness among features
2. check for distribution between churners and non churners
3. did some feature engineering by adding columns from subtracting from other columns

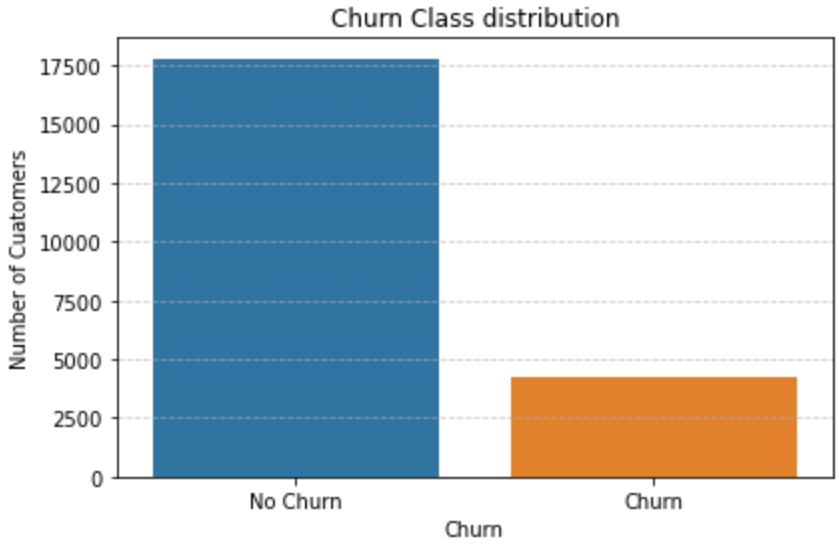
step1:Univariate Analysis

In [22]:
```
#check distribution between churners and non churners we see if they are equa
df['churn'].value_counts()
# as you can see we have an imbalanced dataset where we have more non churner
```

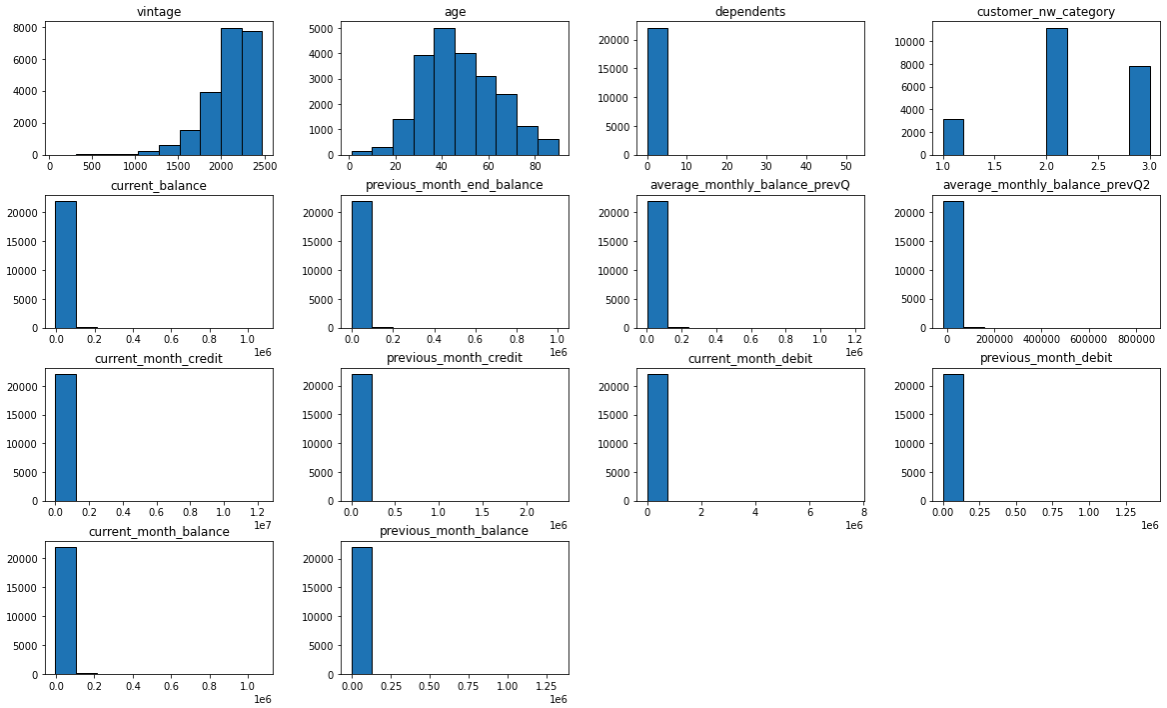
Out[22]:
```
0    17793
1     4274
Name: churn, dtype: int64
```

In [23]:
```
#lets visualize this
churn_counts=df['churn'].value_counts().sort_index()

#plot
plt.figure(figsize=(6,4))
sns.barplot(x=churn_counts.index,y=churn_counts.values)
plt.title('Churn Class distribution')
plt.xlabel('Churn')
plt.ylabel("Number of Cuatomers")
plt.xticks([0,1],['No Churn','Churn'])
plt.grid(axis='y',linestyle='--',alpha=0.7)
plt.tight_layout()
plt.show()
```



In [24]:
```
#For the numerical variables we will use histograms to check for skewness
exclude_columns=['customer_id','branch_code','churn','city']
columns=df.select_dtypes(include='number').drop(columns=exclude_columns)
columns.hist(figsize=(20,12),bins=10,edgecolor='black',grid=False)
plt.title("histogram of numeric features",fontsize=20)
plt.show()
```



The graph interpretation:
Most of the graphs except vintage are rightly skewed for age is almost symmetrical so we have to scale,handle imbalance,one hot encode for categorical columns in the feature engineering stage

In [25]:
```
# Preprocessing for derived columns
df['credit_change'] = df['current_month_credit'] - df['previous_month_credit'
df['debit_change'] = df['current_month_debit'] - df['previous_month_debit']
df['balance_change'] = df['current_balance'] - df['previous_month_end_balance
df['last_transaction'] = pd.to_datetime(df['last_transaction'])
df['days_since_last_transaction'] = (df['last_transaction'].max() - df['last_
```

Objective 1:Which features most influence whether a client will churn or not at Blue-Valley Bank

1. i will start with logistic regression for interpretability
2. i will then add decision tree classifier to capture non linear effects and interactions
3. i will the compare top features from both models to cross-validate insights

we will use this two models to actually help us capture the features or columns that influence churn

Model Preprocessing

1. one hot encode for categorical features

2. Normalize numerical features
3. Use Smote for imbalance

step1:preprocess the data by first one hot encoding and normalizing numerical values

In [26]:
```
df_encoded=pd.get_dummies(df,columns=['occupation','gender'],drop_first=True)
df_encoded
```

Out[26]:

	customer_id	vintage	age	dependents	city	customer_nw_category	branch_code	c
0	1	2101	66	0.0	187.0	2	755	
4	6	1579	42	2.0	1494.0	3	388	
5	7	1923	42	0.0	1096.0	2	1666	
6	8	2048	72	0.0	1020.0	1	1	
7	9	2009	46	0.0	623.0	2	317	
...	...	...	...	...	...	...	...	
28375	30295	2398	42	0.0	146.0	2	286	
28377	30297	2325	10	0.0	1020.0	2	1207	
28378	30298	1537	34	0.0	1046.0	2	223	
28379	30299	2376	47	0.0	1096.0	2	588	
28381	30301	1175	18	0.0	1232.0	2	474	

22067 rows × 28 columns

for normalizing my data i will use min max scaler since as we have seen in the graph on the EDA analysis we have skewed dataset and if my dataset was normally distributed we would have used standard scaler

In [27]:
```
# here we used min max scaler since in the EDA we noticed the values are not
columns = ['current_balance', 'average_monthly_balance_prevQ',
           'average_monthly_balance_prevQ2', 'current_month_credit',
           'previous_month_credit', 'current_month_debit',
           'previous_month_debit', 'current_month_balance',
           'previous_month_balance']
scaler=MinMaxScaler()
df_encoded[columns]=scaler.fit_transform(df_encoded[columns])
```

we will then define our X and y which is features and target respectively

In [28]:
```
X=df_encoded.drop(columns=['churn','last_transaction'])
y=df_encoded['churn']
# we dropped last transaction since its in date time remember at EDA we did f
```

In [29]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state
```

we will the use smote to handle class imbalance as seen in the EDA analysis

In [30]:
```
#We will the use class balanced to handle To handle imbalance as you saw on t
smote=SMOTE(random_state=42)
X_train_resampled,y_train_resampled=smote.fit_resample(X_train,y_train)
```

In [31]:
```
#We will now use the logistic regression as the base line model to find the c
logreg = LogisticRegression(fit_intercept=False, C=1e12, solver='liblinear')
model_log = logreg.fit(X_train_resampled, y_train_resampled)
model_log
```

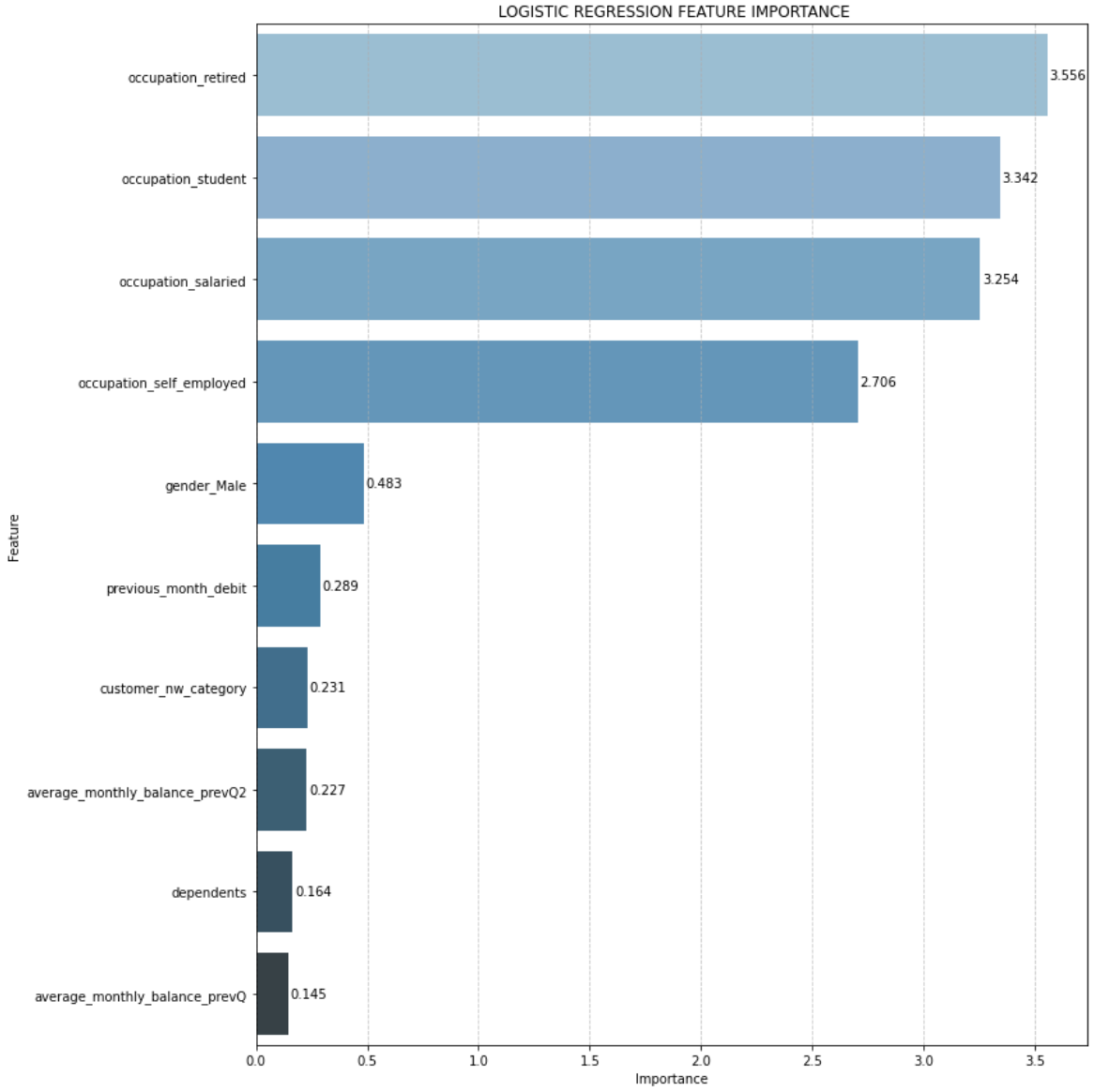
Out[31]:
```
LogisticRegression(C=1000000000000.0, fit_intercept=False, solver='liblinea
r')
```

In [32]:
```
#here we create a dataframe for absolute coefficients as importance
coeff=pd.DataFrame({
    'Feature':X_train_resampled.columns,
    'Importance':np.abs(model_log.coef_[0]),

})
```

In [33]:
```
# we will format the format importance to 3 decimal places for easier readabi
coeff['Importance']=coeff['Importance'].round(3)
coeff=coeff.sort_values(by='Importance',ascending=False).head(10)
```

In [34]:
```
# we will draw a graph for this
plt.figure(figsize=(12,12))
sns.barplot(x='Importance',y='Feature',data=coeff,palette='Blues_d')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title(' LOGISTIC REGRESSION FEATURE IMPORTANCE')
plt.grid(True,axis='x',linestyle='--',alpha=0.7)
# we will show the values on bars
for i,(importance,feature)in enumerate(zip(coeff['Importance'],coeff['Feature
    plt.text(importance +0.01,i,str(importance),va='center')
plt.tight_layout()
plt.show()
```



for what we have done at the above a postive coefficient means that the feature increase chance of churn and negative is the vice versa using absolute coefficients measure the stregnth of the feature in regards to the churn column a big absolute value has strong impact a small absolute value has the feature has little impact and also we use it since it doesnt have a built in importance metric like decision trees

we will use decision tree model to also extract important features in our model

In [35]:
```
#we train the decision tree on the smote resampled data
model2=DecisionTreeClassifier(max_depth=5,class_weight='balanced',random_stat

model2.fit(X_train_resampled,y_train_resampled)#this is the training
```

Out[35]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=5, random_state=4
2)
```

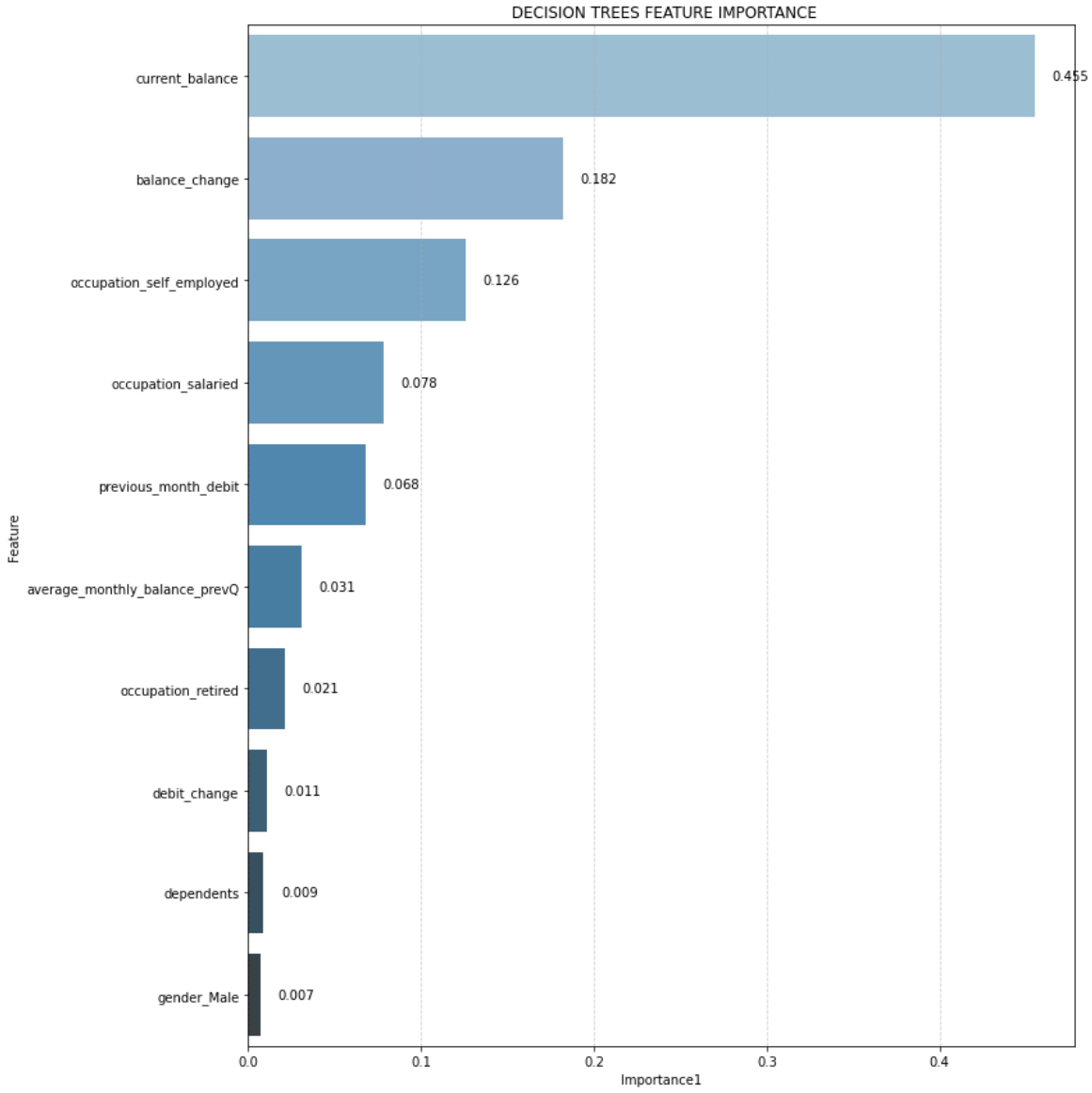
In [36]:
```
#we will now get the feature importances
tree_coeff=pd.DataFrame({
    'Feature':X_train_resampled.columns,
    'Importance':model2.feature_importances_
})
```

In [37]:
```
print(tree_coeff)
```

```
                         Feature  Importance
0                    customer_id    0.000000
1                        vintage    0.000000
2                            age    0.000000
3                     dependents    0.009341
4                           city    0.000000
5            customer_nw_category    0.002300
6                    branch_code    0.002022
7                 current_balance    0.454808
8      previous_month_end_balance    0.001861
9   average_monthly_balance_prevQ    0.031135
10  average_monthly_balance_prevQ2    0.000000
11           current_month_credit    0.001144
12          previous_month_credit    0.000973
13            current_month_debit    0.000000
14           previous_month_debit    0.067501
15          current_month_balance    0.000000
16         previous_month_balance    0.000000
17                  credit_change    0.000000
18                   debit_change    0.011304
19                 balance_change    0.181887
20    days_since_last_transaction    0.003677
21             occupation_retired    0.021294
22            occupation_salaried    0.078495
23       occupation_self_employed    0.125582
24             occupation_student    0.000000
25                    gender_Male    0.006677
```

In [38]:
```
# we will format the format importance to 3 decimal places for easier readabi
tree_coeff['Importance']=tree_coeff['Importance'].round(3)
tree_coeff=tree_coeff.sort_values(by='Importance',ascending=False).head(10)
```

In [39]:
```
# we will draw a graph for this
plt.figure(figsize=(12,12))
sns.barplot(x='Importance',y='Feature',data=tree_coeff,palette='Blues_d')
plt.xlabel('Importance1')
plt.ylabel('Feature')
plt.title(' DECISION TREES FEATURE IMPORTANCE')
plt.grid(True,axis='x',linestyle='--',alpha=0.6)
# we will show the values on bars
for i,(importance,feature)in enumerate(zip(tree_coeff['Importance'],tree_coef
    plt.text(importance +0.01,i,str(importance),va='center')
plt.tight_layout()
plt.show()
```



In [40]:
```
#WE WILL COMBINE THE BOTH MODEL RESULTS AND THEN GIVE THEM SIDE BY SIDE
comparison=pd.merge(
coeff.rename(columns={'Importance':'Logistic_Importance'}),
tree_coeff.rename(columns={'Importance':'Tree_Importance'}),
on='Feature',
how='outer'
)
# we will sort thr model using logistic regression
comparison=comparison.sort_values(by='Logistic_Importance',ascending=False)
# we will then display it side by side
print(comparison)
```

```
                          Feature  Logistic_Importance  Tree_Importance
0              occupation_retired                3.556            0.021
1              occupation_student                3.342              NaN
2             occupation_salaried                3.254            0.078
3        occupation_self_employed                2.706            0.126
4                     gender_Male                0.483            0.007
5            previous_month_debit                0.289            0.068
6            customer_nw_category                0.231              NaN
7   average_monthly_balance_prevQ2                0.227              NaN
8                      dependents                0.164            0.009
9    average_monthly_balance_prevQ                0.145            0.031
10                current_balance                  NaN            0.455
11                 balance_change                  NaN            0.182
12                   debit_change                  NaN            0.011
```

In [ ]:
```

```

cross_validation:From our observations we see that for logistic regresion the occupation and gender features are dominating and for the tree importance current balance is leading why? so for the logistic regression as much as we scaled the features categorical features where not scaled and since we did one hot encoding we create many columns and this increases the coefficiency and numeric as you can see have small coefficients for the decision trees as we can observe the high coefficient column in regards to churn is current balance and the benefit of this is that it also checks non linear relationships

for chosing the features that are highly important i would go for the once in decision trees since it also checks non linear relationships

Objective 2:

- **Which predictive model between logistic and decision tree is better in terms of performance in indetifying churn?**

For the first objective we used logistic regression to check for feature importance with churn so we did the same for decision trees so we will do the ROC/AUC and confusion matrix to see which one performs better

In [41]:
```
# we will start evaluating code using the logistic regression
y_pred=model_log.predict(X_test)
#we will also predict probabilitied  for the  class 1
y_pred_proba=model_log.predict_proba(X_test)[:,1]
```

In [42]:
```
accuracy=accuracy_score(y_test,y_pred)
f1=f1_score(y_test,y_pred)
recall=recall_score(y_test,y_pred)
precision=precision_score(y_test,y_pred)
# here shows the accuracy,f1 score,precision and recall
```

In [43]:
```
print(f" model:logistic  regression:")
print(f"accuracy:{accuracy}")
print(f"F1_Score:{f1} ")
print(f"Recall:{recall}")
print(f"Precision:{precision}")
```

```
 model:logistic  regression:
accuracy:0.7288173991844132
F1_Score:0.2624768946395564
Recall:0.24482758620689654
Precision:0.28286852589641437
```

In [44]:
```
# FOR The decision tree performance
y_pred2=model2.predict(X_test)
y_pred_proba2=model2.predict_proba(X_test)[:,1]
```

In [45]:
```
accuracy=accuracy_score(y_test,y_pred2)
f1=f1_score(y_test,y_pred2)
recall=recall_score(y_test,y_pred2)
precision=precision_score(y_test,y_pred2)
```

In [46]:
```
print(f" model:decision Trees:")

print(f"accuracy{accuracy}")

print(f" f1_Score{f1} ")

print(f"Recall{recall}")

print(f"Precision{precision}")
```

```
 model:decision Trees:
accuracy0.7689170820117807
 f1_Score0.5114942528735632
Recall0.6137931034482759
Precision0.43842364532019706
```

THIS IS THE PERFORMANCE EVALUATION

Based on the performance our accuracy score for the logistic regression is: 0.72 our recall is quite low 0.24 our precision is 0.27 while

decision trees the accuracy is 0.76 which is 76% f1 score is 51% or 0.51 recall is 0.6 or 61% precision is 0.4 or 43%

In comparison to the two models decision trees performs way better even though decision trees needs some improvements here and there its way better than logistic regression

we also observed that even in terms of feature importance decision trees seem to indentify non linear relationships and not affected by the one hot encoding done unlike logistic regression which even after we handled imbalance we handled scaling and did one hot encoding its performance was low

in conclusion:we will chose the decision trees as the final model but we will improve it then use it for segmenting churners into high low and moderate risk churners

Objective 3:

- **How can clients be segmented into high,moderate and low churn risk categories based on their predicated probability of churning?**

since we did identified decision trees as the best model for the project we will do the following:

1. We will check for overfitting and we will improve our model based on this
2. We will then segment our clients into high,moderate and low risk churn categories

step1:checking overfitting we will do this my comparing the train and test if the train is way higher than test set we will do prunning which is removing unwanted features so as to improve model performance

In [47]:
```
#here we are testing or rather comparing the test and trained data to see if
#if overfitting is present the training data will either be abit or way highe

train=model2.predict(X_train_resampled)
test=model2.predict(X_test)
#Accuracy
train_accuracy=accuracy_score(y_train_resampled,train)
test_accuracy=accuracy_score(y_test,test)
print(f"accurate scores are :Train: {train_accuracy},    Test: {test_accuracy
```

```
accurate scores are :Train: 0.7795283879570496,    Test: 0.7689170820117807
```

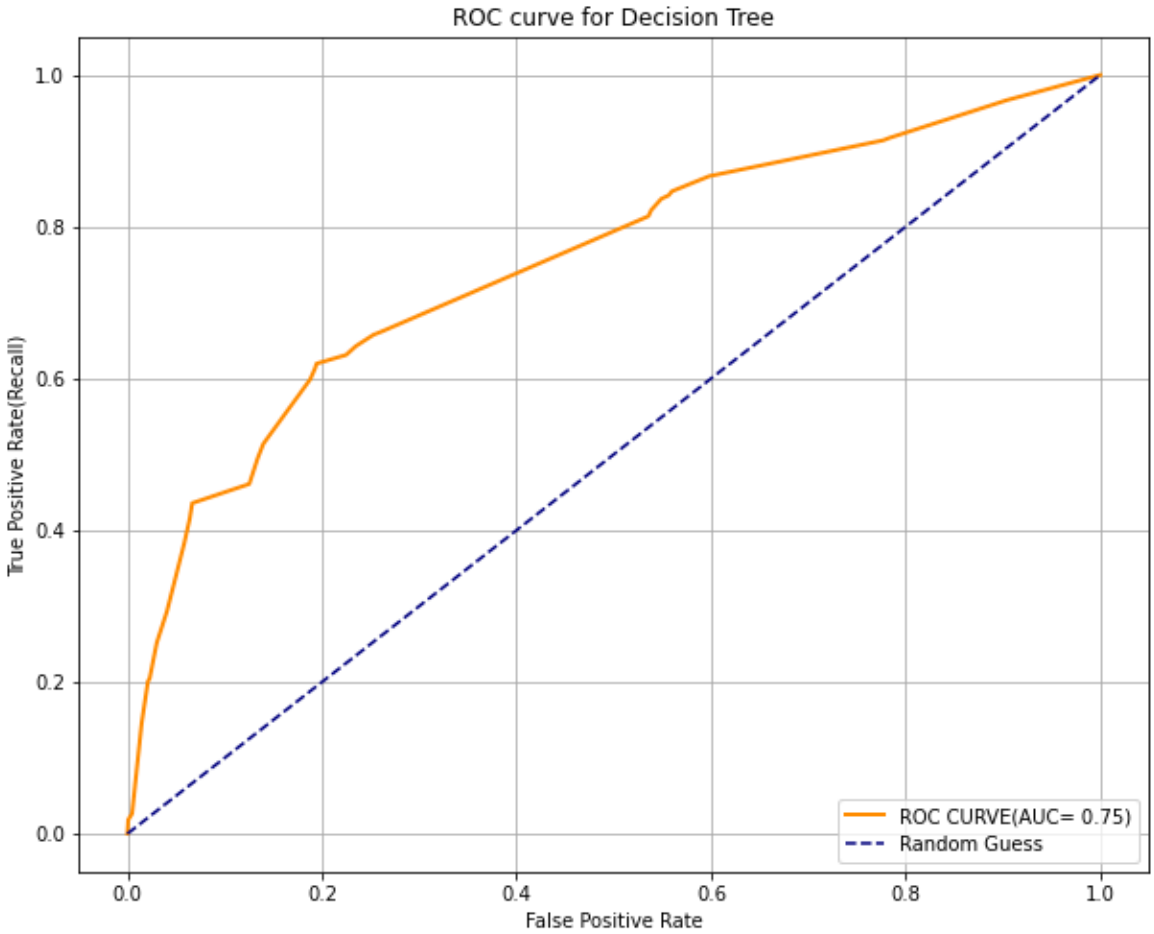
Assessing results

based ont the results of the training and testing of data there is a slight overfitting of the model hence it could be contributing to low precision. We will draw an ROC /AUC CURVE to see how it looks then we use prunning to solve the overfitting

In [48]:
```
# we will first calculate the probailty scores of each datapoints
y_score=model2.predict_proba(X_test)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,y_score)
AUC=auc(fpr,tpr)
print('AUC:',AUC)
```

```
AUC: 0.7515783516255417
```

In [49]:
```
# we will the plot the ROC CURVE
plt.figure(figsize=(10,8))
sns.lineplot(x=fpr,y=tpr,color='darkorange',lw=2,label=f'ROC CURVE(AUC={AUC:
plt.plot([0,1],[0,1],color='navy',linestyle='--',lw=1.5,label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate(Recall)')
plt.title('ROC curve for Decision Tree')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



summary of my graph

My model is doing fairly good in distinguishing between churners and non churners

However its not really hugging the top left corner which it should near 1.0 so this indicates we nedd to improve the model and prunning is the best option here

PRUNNING

We will do pre pruning the decision tree this will help in limiting the depth and controlling how the splitting happens here will help in preventing overfiiting

the above the y_pred will give the class labels (0 or 1) for evaluating precision and also recla y_proba gives the probabilities used in the ROC AND AUC SCORE

We will evaluate performance

In [*]:
```
# we fast find the maximum depth
depths=range(1,33)

train_auc=[]
test_auc=[]
for dp in depths:
    clf=DecisionTreeClassifier(max_depth=dp,criterion='entropy',random_state
    clf.fit(X_train_resampled,y_train_resampled)
    y_train_pred=clf.predict_proba(X_train_resampled)[:,1]
    y_test_pred=clf.predict_proba(X_test)[:,1]
    train_auc.append(auc(*roc_curve(y_train_resampled,y_train_pred)[:2]))
    test_auc.append(auc(*roc_curve(y_test,y_test_pred)[:2]))

plt.figure(figsize=(7,4))
plt.plot(depths,train_auc,label="Train AUC")
plt.plot(depths,test_auc,label="Test AUC")
plt.xlabel("Tree Depth")
plt.ylabel("AUC Score")
plt.title("AUC VS Tree depth")
plt.legend()
plt.tight_layout()
plt.show()
```

for this case: Train auc starts low and increases near 1.0 Test auc starts off and indeed increase with Train and then declines sharply and stays low and it also shows over fitting the sweet spot i think is between 15-18

In [*]:
```
#minimum_splits : minimum number of samples needed to split an internal node.
maximum_splits = np.arange(0.1, 1.1, 0.1)
train_auc = []
test_auc = []

for s in maximum_splits:
    clf = DecisionTreeClassifier(min_samples_split=s, criterion='entropy', ra
    clf.fit(X_train_resampled, y_train_resampled)

    y_train_pred = clf.predict_proba(X_train_resampled)[:, 1]
    y_test_pred = clf.predict_proba(X_test)[:, 1]

    train_auc.append(auc(*roc_curve(y_train_resampled, y_train_pred)[:2]))
    test_auc.append(auc(*roc_curve(y_test, y_test_pred)[:2]))
```

In [*]:
```
#plotting
plt.figure(figsize=(7,4))
plt.plot(maximum_splits,train_auc,label='Train AUC')
plt.plot(maximum_splits,test_auc,label="Test AUC")
plt.xlabel("Minimum Samples Split")
plt.ylabel('AUC Score')
plt.title("AUC vs Minimum Samples Split")
plt.legend()
plt.tight_layout()
plt.show()
```

interpretation of the graph

for the Train AUC is very high near 1.0 this means that the training data is extremely well therefore overfitting Test AUC is much lower but gradually improves as the min sample split keeps on increasing therefore showing not generalizing data well as it should the sweet spot is between 0.6-0.7

In [*]:
```
# we will do the minimum sample leaf which controls the minimum numbers of sa
leaf=np.arange(0.1,0.6,0.1)
train_auc=[]
test_auc=[]
for l in leaf:
    min_samples=int(l *len(X_train_resampled))
    min_samples=max(1,min_samples)


    clf=DecisionTreeClassifier(min_samples_leaf=min_samples,criterion='entrop
    clf.fit(X_train_resampled,y_train_resampled)

    y_train_pred=clf.predict_proba(X_train_resampled)[:,1]
    y_test_pred=clf.predict_proba(X_test)[:,1]

    train_auc.append(auc(*roc_curve(y_train_resampled,y_train_pred)[:2]))
    test_auc.append(auc(*roc_curve(y_test,y_test_pred)[:2]))
```

In [*]:
```
plt.figure(figsize=(7,4))
plt.plot(leaf,train_auc,label='Train AUC')
plt.plot(leaf,test_auc,label="Test AUC")
plt.xlabel("min_samples_leaf ")
plt.ylabel("AUC Score")
plt.title("AUC VS min_samples_leaf")
plt.legend()
plt.tight_layout()
```

for this graph overfitting is seen betwen 0.10 to 19 and underfitting is between 0.40 -0.50 the sweet spot is between 0.20-0.30

In [*]:
```
#to get the features that will give the best model
max_features=range(1,X.shape[1]+1)# this is 1 to number of features
train_auc=[]
test_auc=[]

for f in max_features:
    clf=DecisionTreeClassifier(max_features=f,criterion='entropy',random_stat
    clf.fit(X_train_resampled,y_train_resampled)

    y_train_pred=clf.predict_proba(X_train_resampled)[:,1]
    y_test_pred=clf.predict_proba(X_test)[:,1]

    train_auc.append(auc(*roc_curve(y_train_resampled,y_train_pred)[:2]))
    test_auc.append(auc(*roc_curve(y_test,y_test_pred)[:2]))
```

In [*]:
```
#plot
plt.figure(figsize=(7,4))
plt.plot(max_features,train_auc,label='Train AUC')
plt.plot(max_features,test_auc,label='Test AUC')
plt.xlabel("Max_Features")
plt.ylabel("AUC SCORE")
plt.title("AUC VS max_features")
plt.legend()
plt.tight_layout()
plt.show()
```

the best features to use is between 10-15 since it starts slowly from zero increases gets a slight drop at around 17 and the increases towards increase in the other features

here is the final model with the prefered features

In [*]:
```
dt_discovered = DecisionTreeClassifier(
    criterion='entropy',
    max_depth=15,
    min_samples_split=0.6,
    min_samples_leaf=0.25,
    max_features=15,
    random_state=42
)
dt_discovered.fit(X_train_resampled ,y_train_resampled)

y_pred_discovered = dt_discovered.predict_proba(X_test)[:, 1]
fpr_opt, tpr_opt, _ = roc_curve(y_test, y_pred_discovered)
roc_auc_discovered = auc(fpr_opt, tpr_opt)

plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, label=f'Vanilla (AUC = {roc_auc_discovered:.2f})', linesty
plt.plot(fpr_opt, tpr_opt, label=f'Optimized (AUC = {roc_auc_discovered:.2f})
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve Comparison")
plt.legend()
plt.tight_layout()
plt.show()
roc_auc_discovered
```

since my ROC is 0.75 its pretty decent so we can now move to the next part of my objective with this optimal model to segment my clients into high medium and low risk churners

In [*]:
```
#we will create a dataframe with churn probabilities
risk_df=pd.DataFrame({'CustomerID':X_test.index,
                      'Churn_Probability':y_pred_discovered
})
```

In [*]:
```
# we will ten categorize the risk into low moderate and high risk
def assign_risk(prob):
    if prob>=0.7:
        return 'High'
    elif prob>=0.5:
        return 'Moderate'
    else:
        return 'Low'
risk_df['Churn_Risk']=risk_df['Churn_Probability'].apply(assign_risk)
```

In [*]:
```
plt.figure(figsize=(6,4))
sns.countplot(data=risk_df,x='Churn_Risk',order=['Low','Moderate','High'],pal
plt.title('Client Distribution by Churn Risk Level')
plt.xlabel('Churn Risk Category')
plt.ylabel('Number of Clients')
plt.show()
```

In [*]:
```
risk_df.sample(10)
```

IN CONCLUSION:

1. WE have found out that when doing a prediction project its the best to use the one which will give the optimal results like in our case decision trees was better at not only identifying churn but even features that identify churn
2. Secondly segmenting the clients from high,medium and low will help the bank know how to handle clients accordingly and even help the bank in planning for the clients to prevent churn
3. thirdly we have noticed that few are churning which is good news but its also good to take care of the churners since one client churning is a huge loss to the company
4. we also notice logistic regression is coefficients are highly influenced by one hot encoding unlike decision trees and alot of scaling is done for logistic regression

Recommendation: since we have known the features influencing churn like the current balance,self employed retention strategies like giving interest to them if their current balance is of a certain amount and also offer loans to the self employed at low interest so that they dont run away eventually segmenting clients based on risk is a good thing like for the high churners associate them with the high influencial features and come up with retentions strategies to keep them ,moderate churners also find retention strategies that may make them have that moderate aspect of churning and low the same choosing the the model that can highly recognize non linear relationships unlike the logistic regression that barely can recognize linear relationship

gotten code assistance from codes done in class medium website,sklearn documentation and geeks for geeks