Bank Customer Churn Analysis And Prediction

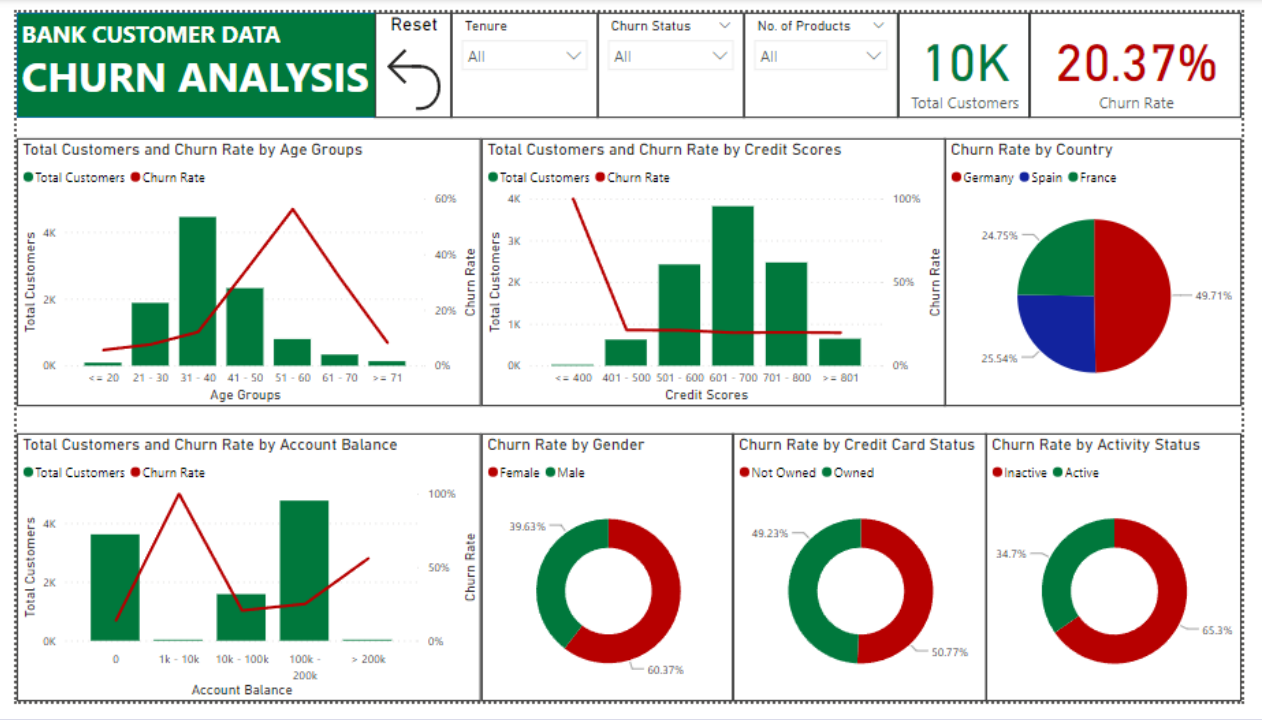
Madhava Krishnan N S | Mar 30, 2024 | www.linkedin.com/in/madhava-krishnan-n-s

Introduction

This project explores the patterns driving customer attrition using Power BI for visualization and Jupyter notebooks for modeling.

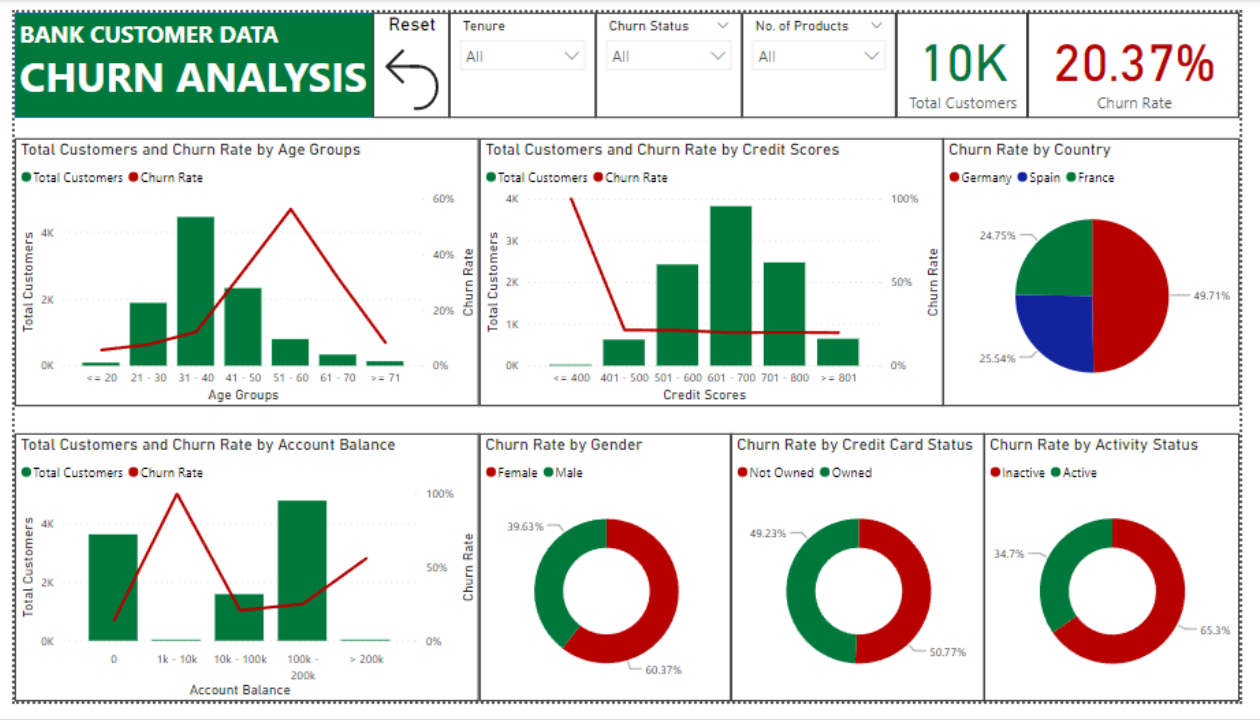
I downloaded customer churn data from Kaggle, cleaned and categorized it for analysis, and created a Power BI dashboard. Predictive analysis was done in Jupyter notebooks, using GridSearchCV to fine-tune multiple algorithms and select the best model. Join me in uncovering the dynamics influencing customer behavior and loyalty through the following results.

Analysis Results – Dashboard & Insights



- The total number of customers is 10,000, with an overall churn rate of 20.37%.
- Customers who have purchased 4 products show a 100% churn rate, while those who have purchased 3 products have an 82.71% churn rate.
- The age group of customers ranging from 51 to 60 exhibits the highest churn rate, standing at 56.21%.
- Customers with a credit score below 400 experience a 100% churn rate.

Analysis Results – Dashboard & Insights

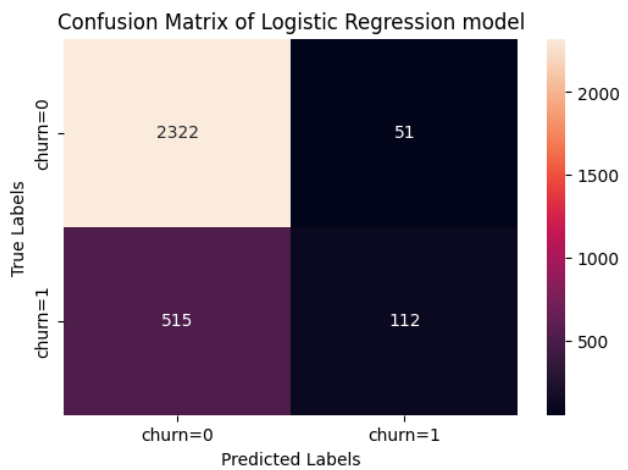


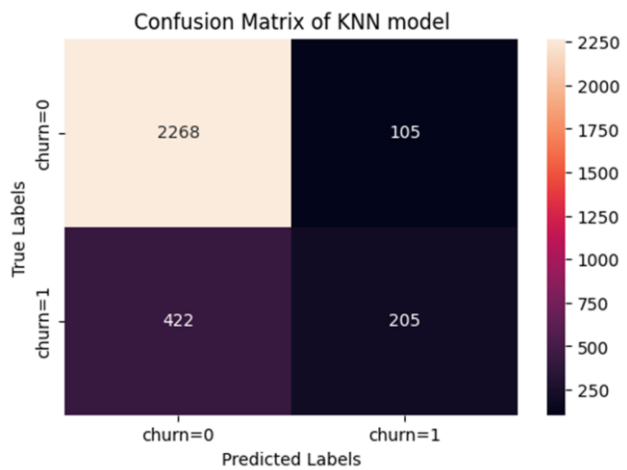
- Germany holds the highest churn rate at 49.71%.
- Customers with a bank balance ranging from 1,000 to 10,000 European dollars experience a 100% churn rate.
- The churn rate among female customers is 60.37%, indicating a higher rate compared to male customers.
- Inactive customers exhibit a higher churn rate of 65.3% compared to active customers.

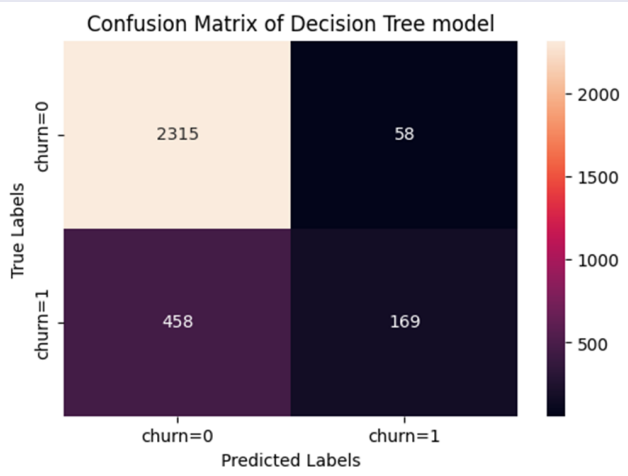
Prediction Results

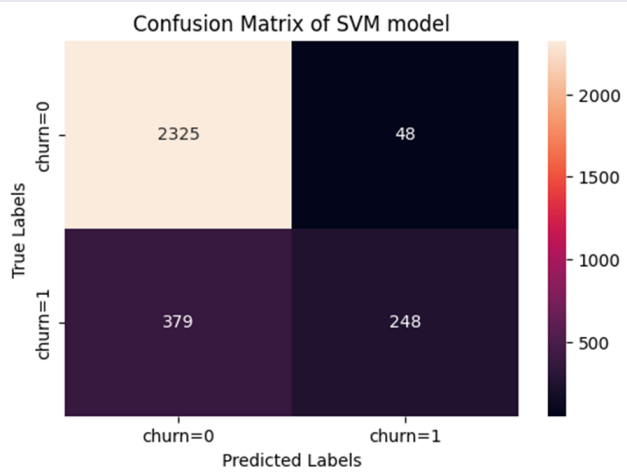
Models/ Scores	Logistic Regression	KNN	Decision Tree	SVM
Accuracy Score	0.8113	0.8243	0.8280	0.8576
Jaccard Score	0.8040	0.8114	0.8177	0.8448
F1 Score	0.76	0.80	0.79	0.84
Log loss	0.4333	N/A	N/A	N/A

After evaluating various performance metrics, including the average score, Jaccard score, F1 score, and reviewing the confusion matrix, it is evident that the Support Vector Machine (SVM) model outperformed the other three models in all aspects.









Conclusion & Credits

- Our exploration into customer churn has revealed valuable insights crucial for business strategies.
- The Power BI dashboard serves as a comprehensive visual representation, showcasing the impact of various factors on churn rates.
- The prediction methodology, implemented through Jupyter notebooks, delivers actionable results by employing advanced models.
- Through these methodologies, we not only understand the current state of customer churn but also empower decision-makers with predictive tools to mitigate future attrition effectively.

- I appended the Jupyter Notebook in the following slides.

- The icons were taken from:
  - Credit card icons created by monkik - Flaticon
  - Deposit icons created by Dewi Sari - Flaticon
  - Community engagement icons created by Canticons - Flaticon
  - Machine learning icons created by mpanicon - Flaticon

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

In [2]:
```
pd.set_option('display.max_columns',None)
```

In [3]:
```
def plot_cf(Y_test,yhat,title):

    from sklearn.metrics import confusion_matrix

    cf_matrix = confusion_matrix(Y_test, yhat)

    df_cm = pd.DataFrame(cf_matrix, index=["churn=0", "churn=1"], columns=["churn=0", "churn=1"])

    plt.figure(figsize=(6, 4))
    sns.heatmap(df_cm, annot=True, fmt='d')
    plt.xlabel("Predicted Labels")
    plt.ylabel("True Labels")
    plt.title(f"Confusion Matrix of {title}")
    plt.show()
```

Data Collection

In [4]:
```
df = pd.read_csv(r"C:\Users\madhavandata\Downloads\Churn_Modelling.csv")
```

In [5]:
```
df
```

Out[5]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSa
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	10134
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	11254
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	11393
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	9382
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	7908

Data Wrangling

In [6]:
```
df_model = df.drop(columns=['RowNumber', 'CustomerId', 'Surname'])
```

In [7]:
```
df_model
```

Out[7]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0
...	...	...	...	...	...	...	...	...	...	...	...
9995	771	France	Male	39	5	0.00	2	1	0	96270.64	0
9996	516	France	Male	35	10	57369.61	1	1	1	101699.77	0
9997	709	France	Female	36	7	0.00	1	0	1	42085.58	1
9998	772	Germany	Male	42	3	75075.31	2	1	0	92888.52	1
9999	792	France	Female	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 11 columns

In [8]:
```
for i in df_model.columns:
    print(i)
    print("")
    print(df[i].unique())
    print("")
```

```
CreditScore

[619 608 502 699 850 645 822 376 501 684 528 497 476 549 635 616 653 587
 726 732 636 510 669 846 577 756 571 574 411 591 533 553 520 722 475 490
 804 582 472 465 556 834 660 776 829 637 550 698 585 788 655 601 656 725
 511 614 742 687 555 603 751 581 735 661 675 738 813 657 604 519 664 678
 757 416 665 777 543 506 493 652 750 729 646 647 808 524 769 730 515 773
 814 710 413 623 670 622 785 605 479 685 538 562 721 628 668 828 674 625
 432 770 758 795 686 789 589 461 584 579 663 682 793 691 485 650 754 535
```

jupyter Customer Churn Prediction (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Markdown

In [9]:
```
df_model.dtypes
```

Out[9]:
```
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

In [10]:
```
geo_dummies=pd.get_dummies(df["Geography"])
```

In [11]:
```
geo_dummies
```

Out[11]:

	France	Germany	Spain
0	True	False	False
1	False	False	True
2	True	False	False
3	True	False	False
4	False	False	True
...	...	...	...
9995	True	False	False
9996	True	False	False
9997	True	False	False
9998	False	True	False
9999	True	False	False

10000 rows × 3 columns

In [12]:
```
gender_dummies = pd.get_dummies(df["Gender"])
```

In [12]:
```
gender_dummies = pd.get_dummies(df["Gender"])
```

In [13]:
```
gender_dummies
```

Out[13]:

	Female	Male
0	True	False
1	True	False
2	True	False
3	True	False
4	True	False
...	...	...
9995	False	True
9996	False	True
9997	True	False
9998	False	True
9999	True	False

10000 rows × 2 columns

In [14]:
```
df_encoded = df_model.drop(columns=['Gender','Geography'])
```

In [15]:
```
df_encoded = pd.concat([df_encoded,geo_dummies,gender_dummies],axis=1)
```

In [16]:
```
df_encoded
```

Out[16]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	France	Germany	Spain	Female	Male
0	619	42	2	0.00	1	1	1	101348.88	1	True	False	False	True	False
1	608	41	1	83807.86	1	0	1	112542.58	0	False	False	True	True	False
2	502	42	8	159660.80	3	1	0	113931.57	1	True	False	False	True	False
3	699	39	1	0.00	2	0	0	93826.63	0	True	False	False	True	False
4	850	43	2	125510.82	1	1	1	79084.10	0	False	False	True	True	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

In [17]:
```
df_encoded = df_encoded.replace(True,1)
```

In [18]:
```
df_encoded = df_encoded.replace(False,0)
```

In [19]:
```
df_encoded
```

Out[19]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	France	Germany	Spain	Female	Male
0	619	42	2	0.00	1	1	1	101348.88	1	1	0	0	1	0
1	608	41	1	83807.86	1	0	1	112542.58	0	0	0	1	1	0
2	502	42	8	159660.80	3	1	0	113931.57	1	1	0	0	1	0
3	699	39	1	0.00	2	0	0	93826.63	0	1	0	0	1	0
4	850	43	2	125510.82	1	1	1	79084.10	0	0	0	1	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	771	39	5	0.00	2	1	0	96270.64	0	1	0	0	0	1
9996	516	35	10	57369.61	1	1	1	101699.77	0	1	0	0	0	1
9997	709	36	7	0.00	1	0	1	42085.58	1	1	0	0	1	0
9998	772	42	3	75075.31	2	1	0	92888.52	1	0	1	0	0	1
9999	792	28	4	130142.79	1	1	0	38190.78	0	1	0	0	1	0

10000 rows × 14 columns

In [20]:
```
df_predictors = df_encoded.drop(columns=["Exited"])
```

In [21]:
```
df_target = df["Exited"]
```

In [22]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn import preprocessing
```

In [23]:
```
df_pred_std = preprocessing.StandardScaler().fit(df_predictors).transform(df_predictors.astype("float"))
```

In [24]:
```
df_pred_std
```

Out[24]:
```
array([[-0.32622142,  0.29351742, -1.04175968, ..., -0.57380915,
         1.09598752, -1.09598752],
```

In [25]:
```
X_train,X_test,Y_train,Y_test = train_test_split(df_pred_std,df_target,test_size=0.3,random_state=1)
```

Logistic Regression

In [26]:
```
parameters = {
    "C":[0.01,0.1,1,10,100],
    "penalty":["l2"],
    "solver":["lbfgs"]
}
```

In [27]:
```
from sklearn.linear_model import LogisticRegression
```

In [28]:
```
lr = LogisticRegression()
```

In [29]:
```
lr_cv = GridSearchCV(lr,parameters,cv=5,scoring="accuracy")
```

In [30]:
```
lr_cv.fit(X_train,Y_train)
```

Out[30]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(),
             param_grid={'C': [0.01, 0.1, 1, 10, 100], 'penalty': ['l2'],
                         'solver': ['lbfgs']},
             scoring='accuracy')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [31]:
```
lr_cv.best_params_
```

Out[31]:
```
{'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
```

In [32]:
```
lr_cv.best_score_
```

Out[32]:
```
0.8121428571428572
```

In [33]:
```
lr_cv.score(X_test,Y_test)
```

Out[33]:
```
0.8113333333333334
```

In [55]:
```
metrics.accuracy_score(Y_test,lrhat)
```

Out[55]:
```
0.8113333333333334
```

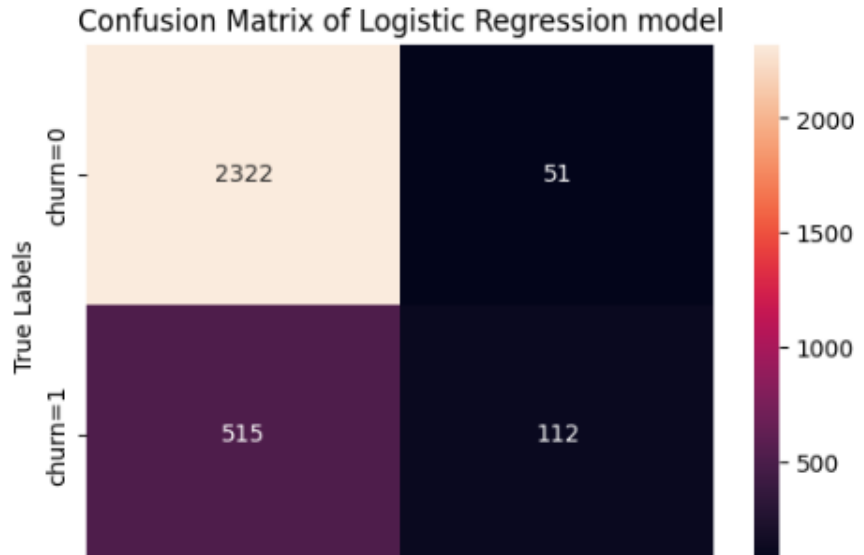
In [35]:
```
lrhat = lr_cv.predict(X_test)
```

In [36]:
```
from sklearn import metrics
```

In [37]:
```
print(metrics.classification_report(Y_test,lrhat))
```

```
              precision    recall  f1-score   support

           0       0.82      0.98      0.89      2373
           1       0.69      0.18      0.28       627

    accuracy                           0.81      3000
   macro avg       0.75      0.58      0.59      3000
weighted avg       0.79      0.81      0.76      3000
```

In [38]:
```
plot_cf(Y_test,lrhat,"Logistic Regression model")
```



In [39]:
```
metrics.log_loss(Y_test,(lr_cv.predict_proba(X_test)))
```

Out[39]:
```
0.43333454309882646
```

In [40]:
```
metrics.jaccard_score(Y_test,lrhat,pos_label=0)
```

Out[40]:
```
0.804016620498615
```

SVM

In [57]:
```
from sklearn.svm import SVC
```

In [58]:
```
svm_parameters = {
    "kernel": ["linear","rbf","poly","sigmoid"],
}
```

In [59]:
```
svm = SVC()
svm_cv = GridSearchCV(svm,svm_parameters,cv=5,scoring="accuracy")
```

In [60]:
```
svm_cv.fit(X_train,Y_train)
```

Out[60]:
```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'kernel': ['linear', 'rbf', 'poly', 'sigmoid']},
             scoring='accuracy')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [61]:
```
svm_cv.best_params_
```

Out[61]:
```
{'kernel': 'rbf'}
```

In [62]:
```
svm_cv.best_score_
```

Out[62]:
```
0.8532857142857143
```

In [63]:
```
svmhat = svm_cv.predict(X_test)
```

In [65]:
```
metrics.accuracy_score(Y_test,svmhat)
```

Out[65]: 0.8576666666666667

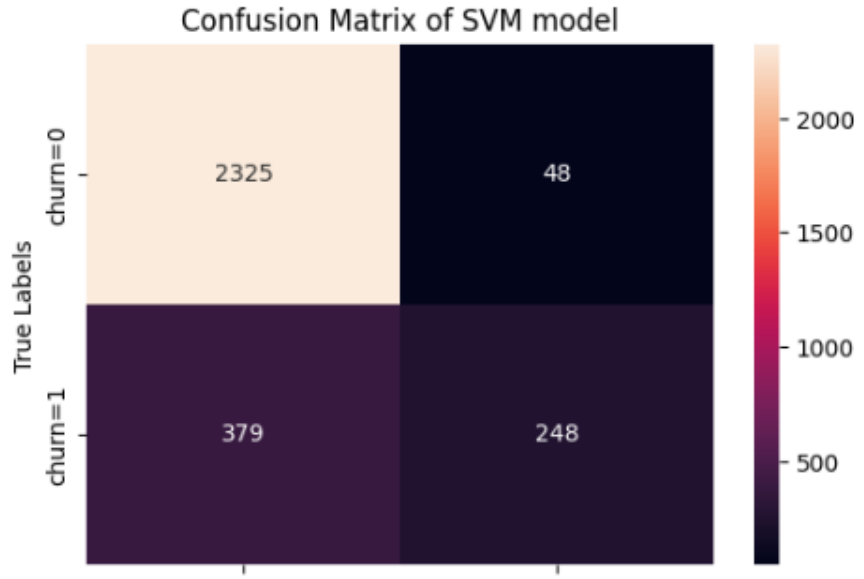
In [66]:
```
metrics.jaccard_score(Y_test,svmhat,pos_label=0)
```

Out[66]: 0.8448401162790697

In [67]:
```
print(metrics.classification_report(Y_test,svmhat))
```

```
              precision    recall  f1-score   support

           0       0.86      0.98      0.92      2373
           1       0.84      0.40      0.54       627

    accuracy                           0.86      3000
   macro avg       0.85      0.69      0.73      3000
weighted avg       0.86      0.86      0.84      3000
```

In [68]:
```
plot_cf(Y_test,svmhat,"SVM model")
```



KNN

In [41]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [42]:
```
knn_parameters = {
    'n_neighbors':[1,2,3,4,5,6,7,8,9,10],
    "algorithm" : ["auto","ball_tree","kd_tree","brute"],
    "p" : [1,2]
}
```

In [43]:
```
knn = KNeighborsClassifier()
knn_cv = GridSearchCV(knn,knn_parameters,cv=5,scoring="accuracy")
```

In [44]:
```
knn_cv.fit(X_train,Y_train)
```

Out[44]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'p': [1, 2]},
             scoring='accuracy')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [45]:
```
knn_cv.best_params_
```

Out[45]:
```
{'algorithm': 'auto', 'n_neighbors': 7, 'p': 1}
```

In [46]:
```
knn_cv.best_score_
```

Out[46]:
```
0.8298571428571428
```

In [48]:
```
knnhat = knn_cv.predict(X_test)
```

In [50]:
```
metrics.accuracy_score(Y_test,knnhat)
```

Out[50]:
```
0.8243333333333334
```

In [56]:
```
metrics.jaccard_score(Y_test,knnhat,pos_label=0)
```

Out[56]: 0.8243333333333334

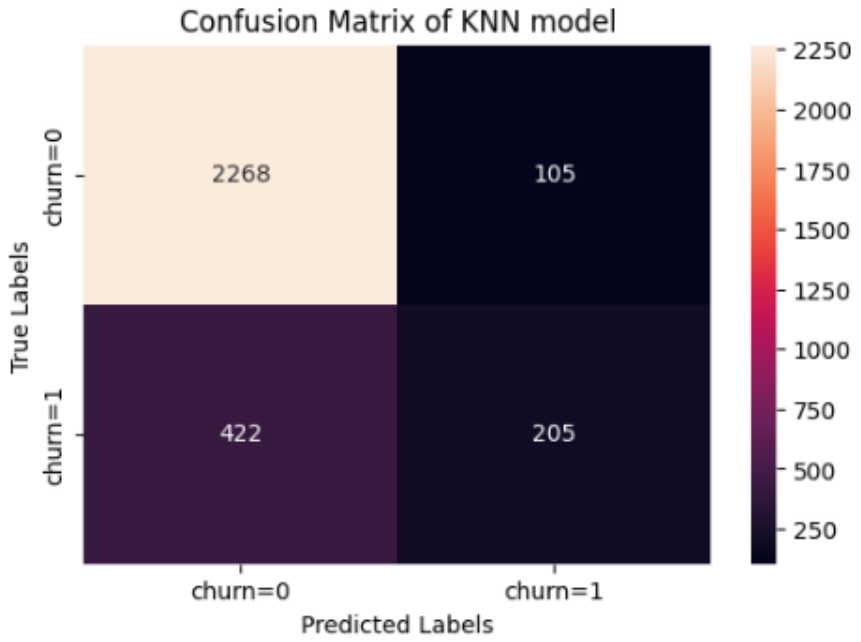
In [56]:
```
metrics.jaccard_score(Y_test,knnhat,pos_label=0)
```

Out[56]: 0.8114490161001789

In [52]:
```
print(metrics.classification_report(Y_test,knnhat))
```

```
              precision    recall  f1-score   support

           0       0.84      0.96      0.90      2373
           1       0.66      0.33      0.44       627

    accuracy                           0.82      3000
   macro avg       0.75      0.64      0.67      3000
weighted avg       0.81      0.82      0.80      3000
```

In [53]:
```
plot_cf(Y_test,knnhat,"KNN model")
```



Decision Tree

In [69]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [70]:
```
tree_parameters = {'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
     'max_depth': [2*n for n in range(1,10)],
     'max_features': ['auto', 'sqrt'],
     'min_samples_leaf': [1, 2, 4],
     'min_samples_split': [2, 5, 10]}

tree = DecisionTreeClassifier()
```

In [71]:
```
tree_cv = GridSearchCV(tree,tree_parameters,cv=5,scoring="accuracy")
```

In [72]:
```
tree_cv.fit(X_train,Y_train)
```

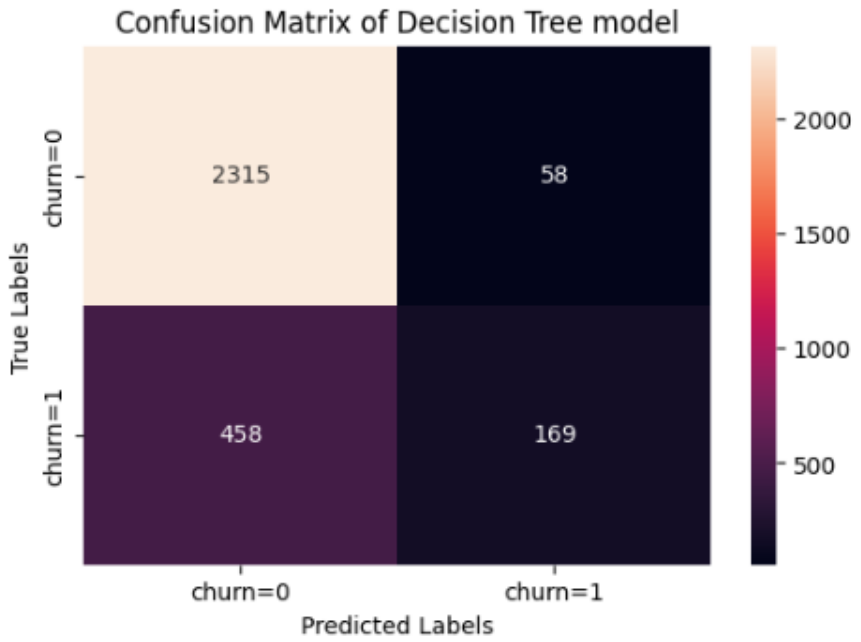
```
C:\Users\madhavandata\anaconda3\envs\madhavanenvironment\lib\site-packages\sklearn\model_selection\_validation.py:425: FitFailedWarning:
1620 fits failed out of a total of 3240.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
1620 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\madhavandata\anaconda3\envs\madhavanenvironment\lib\site-packages\sklearn\model_selection\_validation.py", line 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\madhavandata\anaconda3\envs\madhavanenvironment\lib\site-packages\sklearn\base.py", line 1145, in wrapper
    estimator._validate_params()
  File "C:\Users\madhavandata\anaconda3\envs\madhavanenvironment\lib\site-packages\sklearn\base.py", line 638, in _validate_params
    validate_parameter_constraints(
  File "C:\Users\madhavandata\anaconda3\envs\madhavanenvironment\lib\site-packages\sklearn\utils\_param_validation.py", line
```

In [75]:
```
tree_cv.best_params_
```

Out[75]:
```
{'criterion': 'entropy',
 'max_depth': 6,
```

```
macro avg       0.79      0.62      0.65      3000
weighted avg    0.82      0.83      0.79      3000
```

In [82]:
```
plot_cf(Y_test,treehat,"Decision Tree model")
```



In [84]:
```
best_tree = tree_cv.best_estimator_
```

In [100]:
```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

plt.figure(figsize=(100, 24))
plot_tree(best_tree, filled=True, feature_names=df_predictors.columns, class_names=df_target.unique().astype(str), rounded=True)
plt.show()
```

In [84]:
```
best_tree = tree_cv.best_estimator_
```

In [100]:
```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

plt.figure(figsize=(100, 24))
plot_tree(best_tree, filled=True, feature_names=df_predictors.columns, class_names=df_target.unique().astype(str), rounded=True)
plt.show()
```