TASK #1: UNDERSTAND THE PROBLEM STATEMENT AND BUSINESS CASE

TELECOM CUSTOMERS CHURN PREDICTION

- In this hands-on project, we will train several classification algorithms namely Logistic Regression, Support Vector Machine, K-Nearest Neighbors, and Random Forest Classifier to predict the churn rate of Telecommunication Customers.

- Telecom service providers use customer attrition analysis as one of their key business metrics because the cost of retaining an existing customer is far less than acquiring a new one.
- Machine Learning algorithms help companies analyze customer attrition rate based on several factors which includes various services subscribed by the customers, tenure rate, gender, senior citizen, payment method, etc.

Image Source: https://pixabay.com/illustrations/family-customer-target-group-ball-563968/
Dataset Source : https://www.kaggle.com/blastchar/telco-customer-churn

INSTRUCTOR

- Adjunct professor & online instructor
- Passionate about artificial intelligence, machine learning, and electric vehicles
- Taught 200,000+ students globally
- MBA (2018), Ph.D. (2014), M.A.Sc (2011)

Ryan Ahmed, Ph.D.

MINI CHALLENGE #1:

- Name top 5 telecom companies in North America in 2020?

In [ ]:

TASK #2: IMPORT LIBRARIES/DATASETS AND PERFORM

EXPLORATORY DATA ANALYSIS

In [1]:
```
!pip install cufflinks
# Cufflinks is a third-party wrapper library around Plotly
```

```
otebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (1.1.1)
Requirement already satisfied: testpath in c:\users\johnw\anaconda3\lib\site-packages (from nbconvert->notebo
ok>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (0.4.4)
Requirement already satisfied: mistune<2,>=0.8.1 in c:\users\johnw\anaconda3\lib\site-packages (from nbconver
t->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (0.8.4)
Requirement already satisfied: nbclient<0.6.0,>=0.5.0 in c:\users\johnw\anaconda3\lib\site-packages (from nbc
onvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (0.5.3)
Requirement already satisfied: jupyterlab-pygments in c:\users\johnw\anaconda3\lib\site-packages (from nbconv
ert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (0.1.2)
Requirement already satisfied: bleach in c:\users\johnw\anaconda3\lib\site-packages (from nbconvert->notebook
>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (3.3.0)
Requirement already satisfied: defusedxml in c:\users\johnw\anaconda3\lib\site-packages (from nbconvert->note
book>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (0.7.1)
Requirement already satisfied: pandocfilters>=1.4.1 in c:\users\johnw\anaconda3\lib\site-packages (from nbcon
vert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (1.4.3)
Requirement already satisfied: entrypoints>=0.2.2 in c:\users\johnw\anaconda3\lib\site-packages (from nbconve
rt->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (0.3)
Requirement already satisfied: async-generator in c:\users\johnw\anaconda3\lib\site-packages (from nbclient<
0.6.0,>=0.5.0->nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (1.10)
Requirement already satisfied: nest-asyncio in c:\users\johnw\anaconda3\lib\site-packages (from nbclient<0.6.
```

In [3]:
```
import numpy as np # Multi-dimensional array object
import pandas as pd # Data Manipulation
import matplotlib.pyplot as plt # Data Visualization
import seaborn as sns # Data Visualization
import plotly.express as px # Interactive Data Visualization
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot # Offline version of the Plotly mod
import cufflinks as cf # Works as a connector between the pandas library and plotly
cf.go_offline()
init_notebook_mode(connected=True) # To connect Jupyter notebook with JavaScript
from jupyterthemes import jtplot # Jupyter theme
jtplot.style(theme= 'monokai', context= 'notebook', ticks= True, grid= False)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-3-9ebd28b41aea> in <module>
      8 cf.go_offline()
      9 init_notebook_mode(connected=True) # To connect Jupyter notebook with JavaScript
---> 10 from jupyterthemes import jtplot # Jupyter theme
     11 jtplot.style(theme= 'monokai', context= 'notebook', ticks= True, grid= False)

ModuleNotFoundError: No module named 'jupyterthemes'
```

In [4]:
```
# Read the CSV file
telecom_df=pd.read_csv('telecom_churn.csv')
```

In [5]:
```
# Load the top 5 instances
telecom_df.head()
```

Out[5]:

	state	account_length	area_code	phone_number	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_
0	16	128	415	2845	0	1	25	265.1	
1	35	107	415	2301	0	1	26	161.6	
2	31	137	415	1616	0	0	0	243.4	
3	35	84	408	2510	1	0	0	299.4	
4	36	75	415	155	1	0	0	166.7	

5 rows × 21 columns

In [6]:
```
# Load the bottom 5 instances
telecom_df.tail()
```

Out[6]:

	state	account_length	area_code	phone_number	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes
4995	11	50	408	2000	0	1	40	235.7
4996	49	152	415	394	0	0	0	184.2
4997	7	61	415	313	0	0	0	140.6
4998	7	109	510	3471	0	0	0	188.8
4999	46	86	415	2412	0	1	34	129.4

5 rows × 21 columns

In [7]:
```
# Check the shape of the dataframe
telecom_df.shape
```

Out[7]:
```
(5000, 21)
```

In [8]:
```
# Display the feature columns
telecom_df.columns
```

Out[8]:
```
Index(['state', 'account_length', 'area_code', 'phone_number',
       'international_plan', 'voice_mail_plan', 'number_vmail_messages',
       'total_day_minutes', 'total_day_calls', 'total_day_charge',
       'total_eve_minutes', 'total_eve_calls', 'total_eve_charge',
       'total_night_minutes', 'total_night_calls', 'total_night_charge',
       'total_intl_minutes', 'total_intl_calls', 'total_intl_charge',
       'number_customer_service_calls', 'class'],
      dtype='object')
```

In [9]:
```
# Obtain the summary of the dataframe
telecom_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 21 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   state                           5000 non-null   int64  
 1   account_length                  5000 non-null   int64  
 2   area_code                       5000 non-null   int64  
 3   phone_number                    5000 non-null   int64  
 4   international_plan              5000 non-null   int64  
 5   voice_mail_plan                 5000 non-null   int64  
 6   number_vmail_messages           5000 non-null   int64  
 7   total_day_minutes               5000 non-null   float64
 8   total_day_calls                 5000 non-null   int64  
 9   total_day_charge                5000 non-null   float64
 10  total_eve_minutes               5000 non-null   float64
 11  total_eve_calls                 5000 non-null   int64  
 12  total_eve_charge                5000 non-null   float64
 13  total_night_minutes             5000 non-null   float64
 14  total_night_calls               5000 non-null   int64  
 15  total_night_charge              5000 non-null   float64
 16  total_intl_minutes              5000 non-null   float64
 17  total_intl_calls                5000 non-null   int64  
 18  total_intl_charge               5000 non-null   float64
 19  number_customer_service_calls   5000 non-null   int64  
 20  class                           5000 non-null   int64  
dtypes: float64(8), int64(13)
memory usage: 820.4 KB
```

MINI CHALLENGE #2:

- What is the maximum and average daily minutes?

In [10]:
```
telecom_df.describe()
```

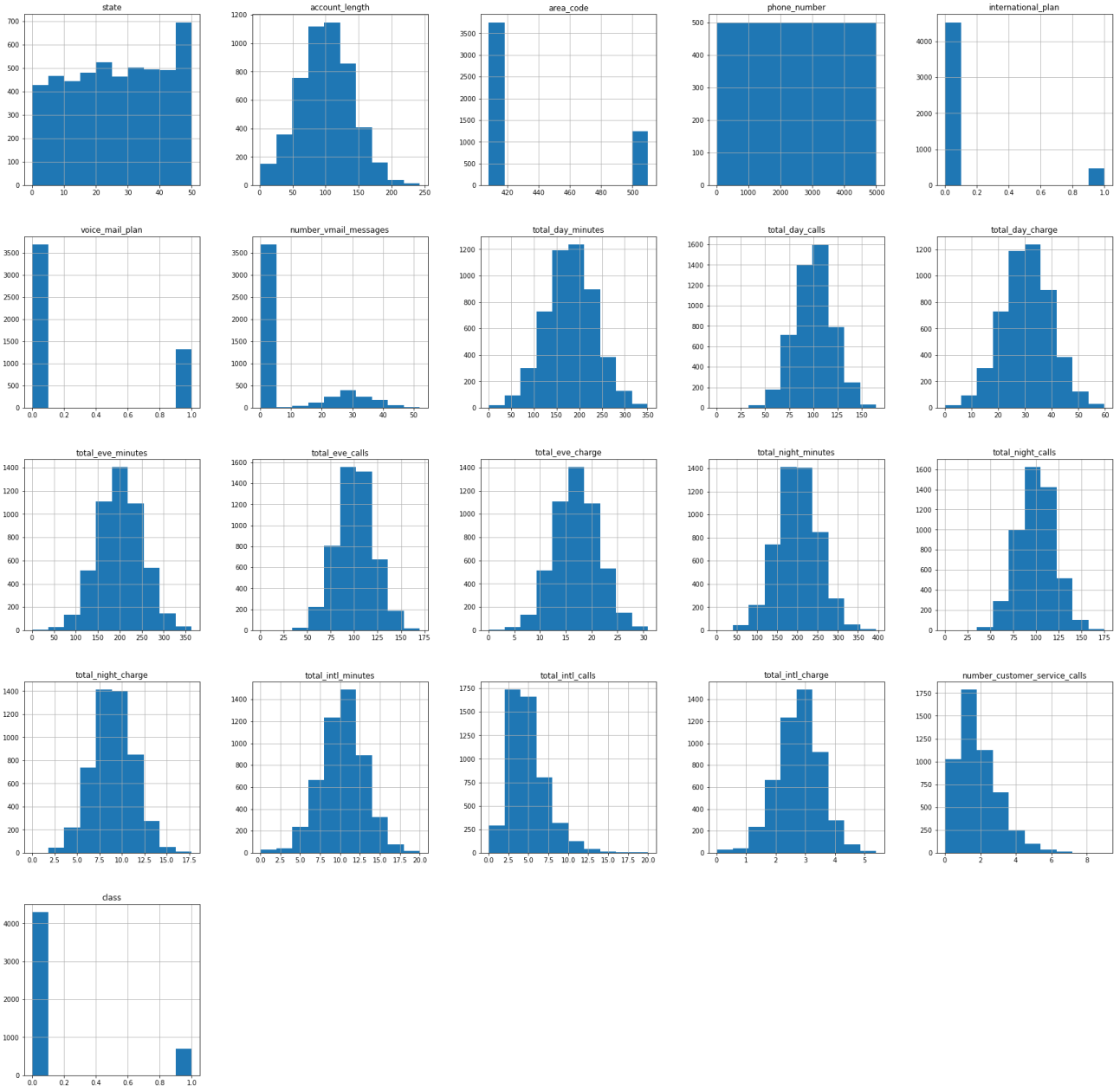
Out[10]:

	state	account_length	area_code	phone_number	international_plan	voice_mail_plan	number_vmail_messages	total_day_mi
count	5000.00000	5000.00000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.0(
mean	25.99840	100.25860	436.911400	2499.500000	0.094600	0.264600	7.755200	180.2(
std	14.80348	39.69456	42.209182	1443.520003	0.292691	0.441164	13.546393	53.8(
min	0.00000	1.00000	408.000000	0.000000	0.000000	0.000000	0.000000	0.0(
25%	13.00000	73.00000	408.000000	1249.750000	0.000000	0.000000	0.000000	143.7(
50%	26.00000	100.00000	415.000000	2499.500000	0.000000	0.000000	0.000000	180.1(
75%	39.00000	127.00000	415.000000	3749.250000	0.000000	1.000000	17.000000	216.2(
max	50.00000	243.00000	510.000000	4999.000000	1.000000	1.000000	52.000000	351.5(

8 rows × 21 columns

TASK #3: PERFORM DATA VISUALIZATION

In [11]:
```
telecom_df.hist(figsize=(30,30))
plt.show()
```



In [12]:
```
telecom_df
```

Out[12]:

	state	account_length	area_code	phone_number	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes
0	16	128	415	2845	0	1	25	265.1
1	35	107	415	2301	0	1	26	161.6
2	31	137	415	1616	0	0	0	243.4
3	35	84	408	2510	1	0	0	299.4
4	36	75	415	155	1	0	0	166.7
...	...	...	...	...	...	...	...	...
4995	11	50	408	2000	0	1	40	235.7
4996	49	152	415	394	0	0	0	184.2
4997	7	61	415	313	0	0	0	140.6
4998	7	109	510	3471	0	0	0	188.8
4999	46	86	415	2412	0	1	34	129.4

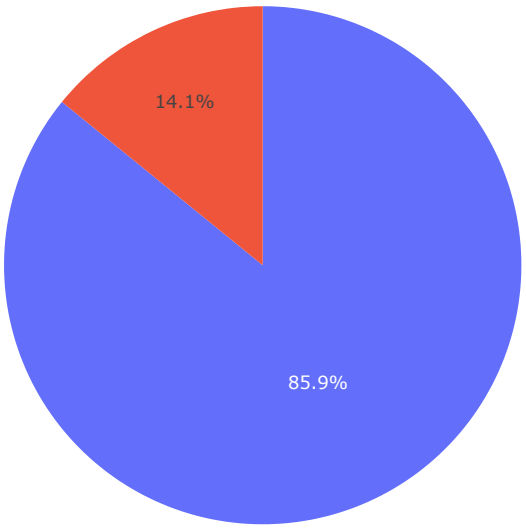
5000 rows × 21 columns

In [13]:
```
telecom_df['class'].value_counts()
```

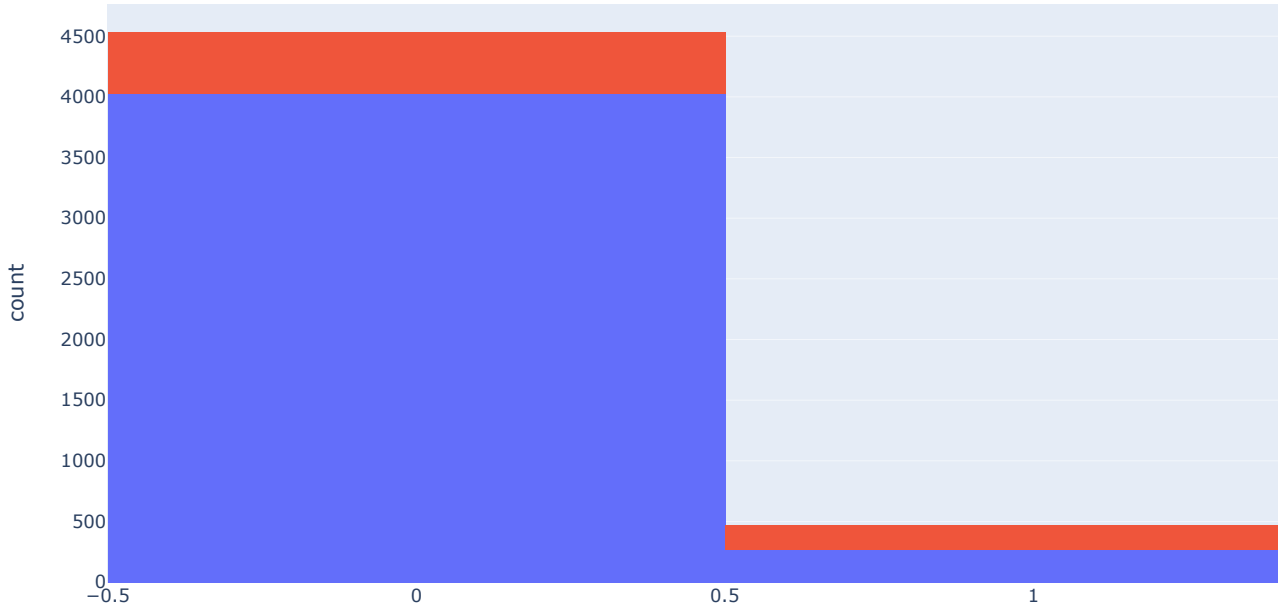
Out[13]:
```
0    4293
1     707
Name: class, dtype: int64
```

In [ ]:
```

```

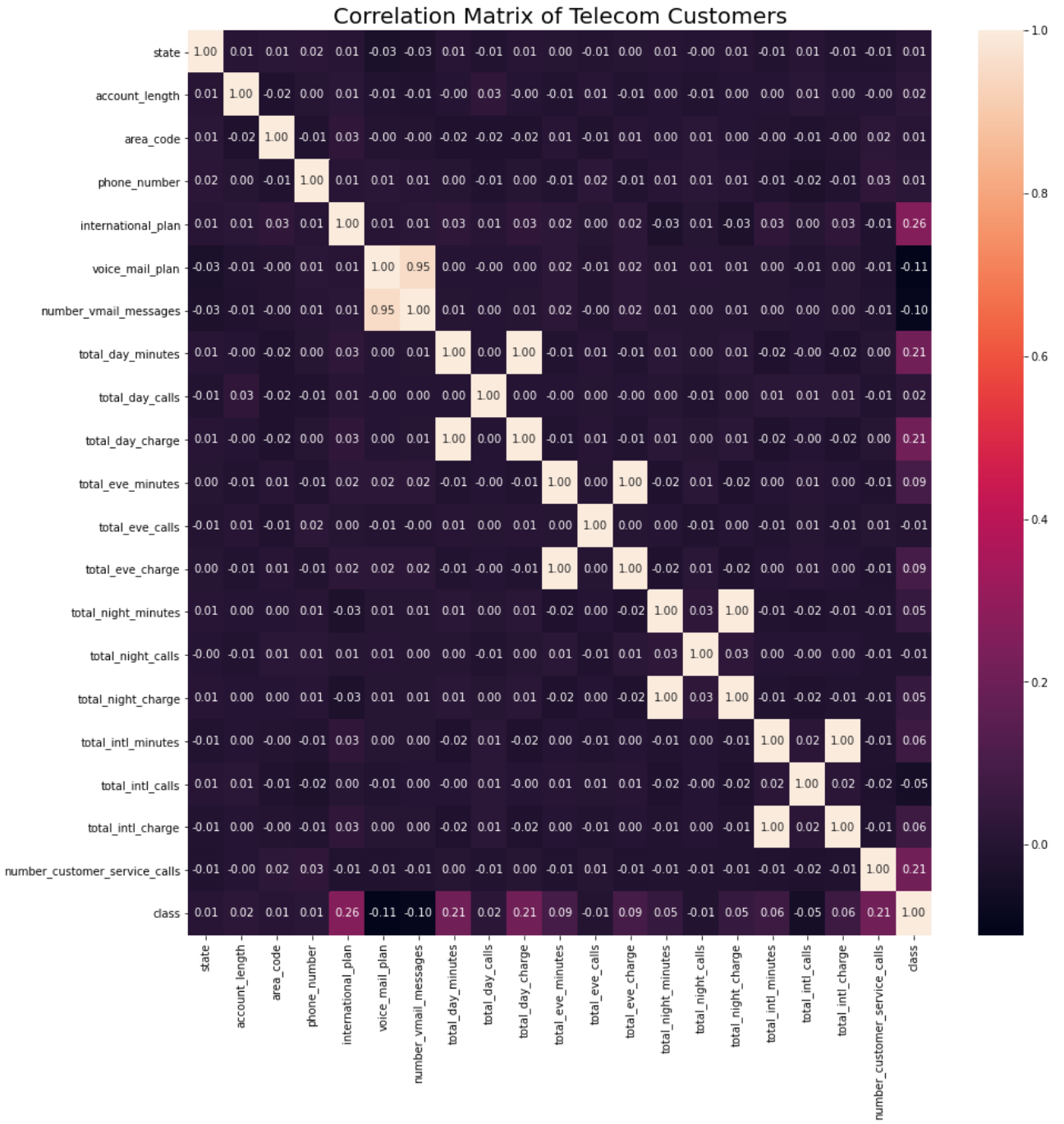
In [73]:
```
# Plot pie Chart to get the information about the percentage of Telecom Customers churning using Plotly histogram

import plotly.graph_objects as go
fig=go.Figure(data=[go.Pie(labels=['Retained (0)', 'Exited (1)'], values=telecom_df['class'].value_counts())])
fig.show()
```



In [15]:
```
# Plot histogram graph for the international plan service used by the Telecom customers with respect to churned/R
fig=px.histogram(telecom_df, x='international_plan', color='class')
fig.show()
```

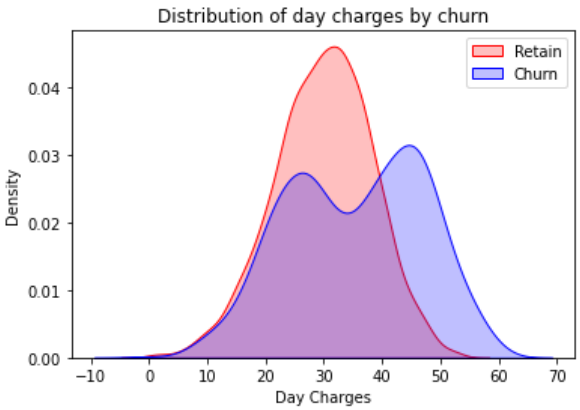


In [16]:
```
# Correlation Matrix

corr_matrix = telecom_df.corr()
plt.figure(figsize = (15, 15))
sns.heatmap(corr_matrix,annot=True,fmt='.2f')
plt.title("Correlation Matrix of Telecom Customers", fontsize = 20)
plt.show()

# It is clearly shown that "voice_mail_plan" and "number_vmail_messages" are highly correlated.
# It is clearly shown that "total day charge" and "total daily minutes" are highly correlated.
```



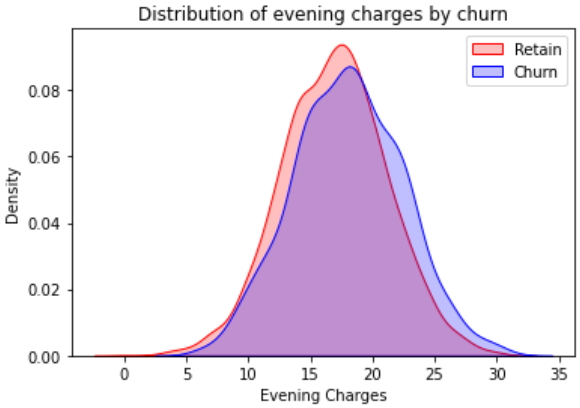
In [17]:
```
# Churn by day charges
ax = sns.kdeplot(telecom_df.total_day_charge[(telecom_df["class"] == 0)],
                 color = "Red", shade = True)
ax = sns.kdeplot(telecom_df.total_day_charge[(telecom_df["class"] == 1)],
                 color = "Blue", shade = True)

ax.legend(["Retain", "Churn"], loc = "upper right")
ax.set_ylabel("Density")
ax.set_xlabel("Day Charges")
ax.set_title("Distribution of day charges by churn")
```

Out[17]:
```
Text(0.5, 1.0, 'Distribution of day charges by churn')
```



In [18]:
```
# Churn by evening charges
ax = sns.kdeplot(telecom_df.total_eve_charge[(telecom_df["class"] == 0)],
                 color = "Red", shade = True)
ax = sns.kdeplot(telecom_df.total_eve_charge[(telecom_df["class"] == 1)],
                 color = "Blue", shade = True)

ax.legend(["Retain", "Churn"], loc = "upper right")
ax.set_ylabel("Density")
ax.set_xlabel("Evening Charges")
ax.set_title("Distribution of evening charges by churn")
```

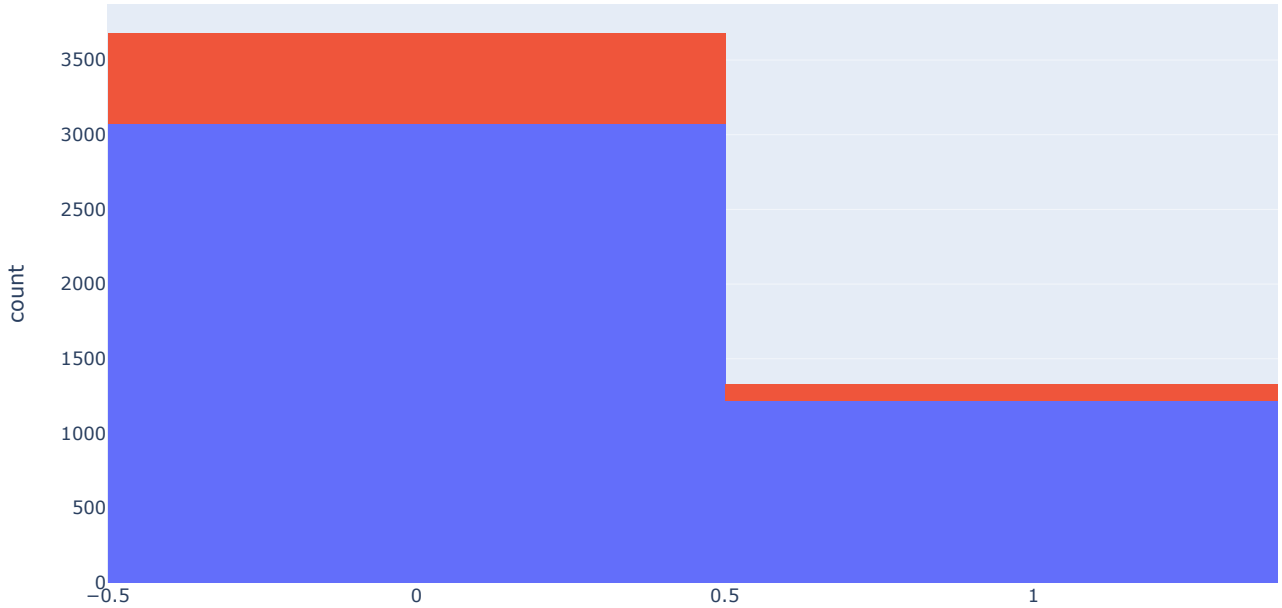
Out[18]:
```
Text(0.5, 1.0, 'Distribution of evening charges by churn')
```



MINI CHALLENGE #3:

- Plot the plotly histogram on voice mail plan correlated with Churn feature

In [19]:
```
fig=px.histogram(telecom_df, x='voice_mail_plan', color='class')
fig.show()
```



TASK #4: IDENTIFY FEATURE IMPORTANCE & PREPARE THE DATA BEFORE MODEL TRAINING

In [20]:
```
# Unnecessary features would decrease the training speed, the model interpretability and the generalization perfo
# Therefore, finding and selecting the most useful features in the dataset is crucial.
# Assigning input features to X and output (Churn) to y

X = telecom_df.drop(["class", "area_code", "phone_number"], axis = "columns") # area_code and phone_number featur
y = telecom_df["class"]
```

In [21]:
```
X
```

Out[21]:

	state	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls	total_day_cha
0	16	128	0	1	25	265.1	110	45
1	35	107	0	1	26	161.6	123	27
2	31	137	0	0	0	243.4	114	41
3	35	84	1	0	0	299.4	71	50
4	36	75	1	0	0	166.7	113	28
...	...	...	...	...	...	...	...	
4995	11	50	0	1	40	235.7	127	40
4996	49	152	0	0	0	184.2	90	31
4997	7	61	0	0	0	140.6	89	23
4998	7	109	0	0	0	188.8	67	32
4999	46	86	0	1	34	129.4	102	22

5000 rows × 18 columns

In [22]:
```
y
```

Out[22]:
```
0       0
1       0
2       0
3       0
4       0
       ..
4995    0
4996    1
4997    0
4998    0
4999    0
Name: class, Length: 5000, dtype: int64
```

In [23]:
```
X.shape
```

Out[23]:
```
(5000, 18)
```

In [24]:
```
y.shape
```

Out[24]:
```
(5000,)
```

In [69]:
```
# Perform train/test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.3)
```

In [70]:
```
X_train.shape
```

Out[70]:
```
(3500, 18)
```

In [71]:
```
X_test.shape
```

Out[71]:
```
(1500, 18)
```

In [ ]:

MINI CHALLENGE #4:

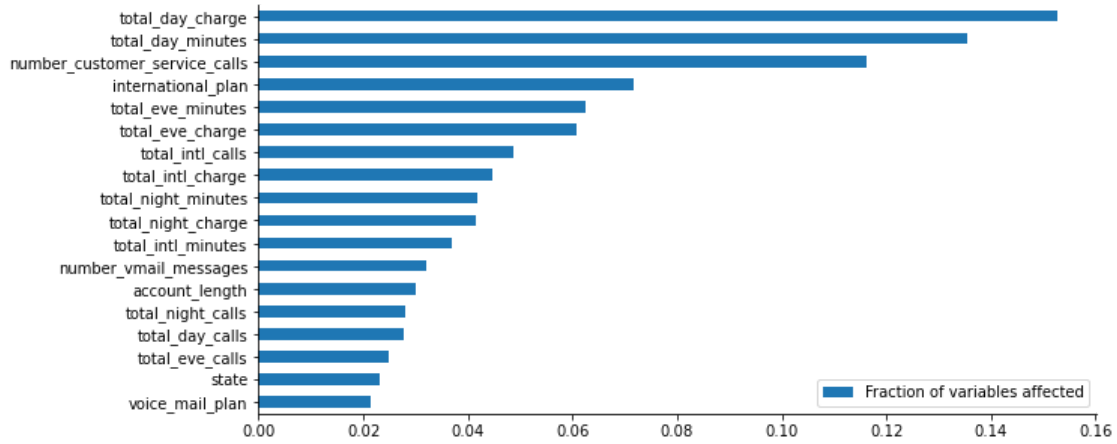
- Verify that the train/test split was successfull

In [ ]:

In [72]:
```
from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier()
rf.fit(X_train,y_train.values.ravel())
```

Out[72]:
```
RandomForestClassifier()
```

In [29]:
```
# Plot the feature importance

feat_scores= pd.DataFrame({"Fraction of variables affected" : rf.feature_importances_},index = X.columns)
feat_scores= feat_scores.sort_values(by = "Fraction of variables affected")
feat_scores.plot(kind = "barh", figsize = (10, 5))
sns.despine()
```

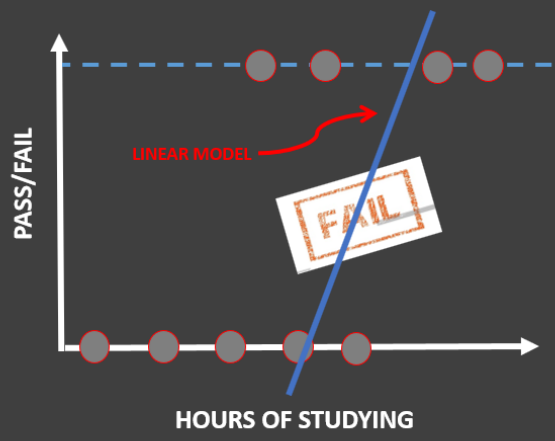


In [30]:
```
# The above graph is generated by Random Forest algorithm
# The graph indicates that "total_day_minutes" tops the list of important features followed by "total_day_minutes
```

TASK #5: TRAIN AND EVALUATE A LOGISTIC REGRESSION CLASSIFIER

LOGISTIC REGRESSION: INTUITION

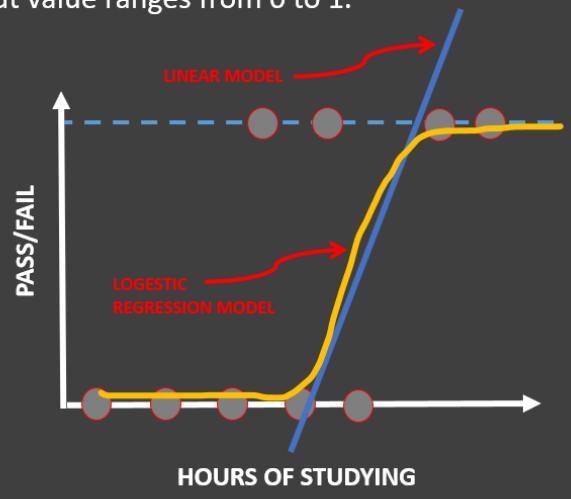
- Linear Regression is used to predict outputs on a continuous spectrum.
  Example: Predicting revenue based on the outside air temperature.
- Logistic Regression is used to predict binary outputs with 2 possible values (0 or 1).
  Example: Logistic model output can be one of two classes: pass/fail, win/lose, healthy/sick



Hours Studying	Pass/Fail
1	0
1.5	0
2	0
3	1
3.25	0
4	1
5	1

LOGISTIC REGRESSION: MATH

- Linear Regression is not suitable for classification problem.
- Linear Regression is unbounded, so Logistic Regression will be better candidate in which the output value ranges from 0 to 1.

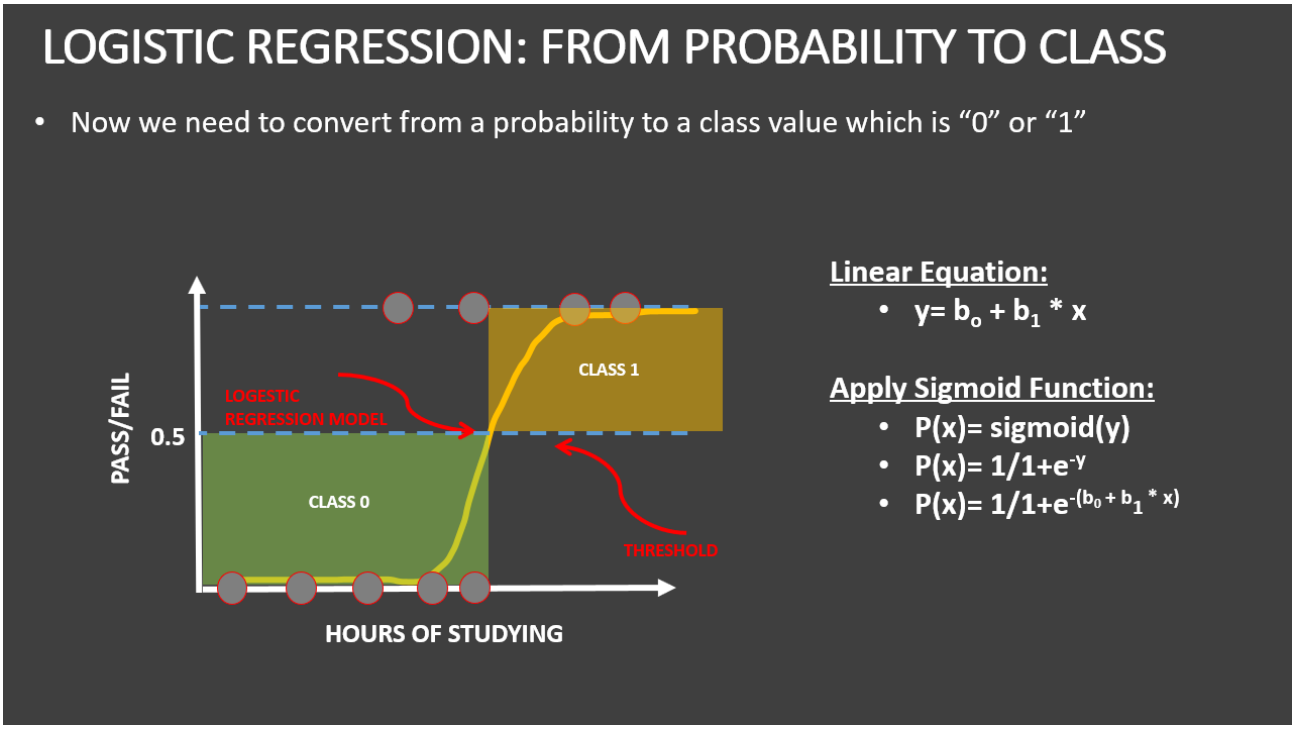


Linear Equation:

- $y = b_o + b_1 * x$

Apply Sigmoid Function:

- $P(x) = \text{sigmoid}(y)$
- $P(x) = 1/1+e^{-y}$
- $P(x) = 1/1+e^{-(b_0 + b_1 * x)}$



In [31]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

model_LR=LogisticRegression()
model_LR.fit(X_train,y_train)
```

```
C:\Users\johnw\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: 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 (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-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
```

Out[31]:
```
LogisticRegression()
```

In [32]:
```
y_predict=model_LR.predict(X_test)
```

In [33]:
```
# precision is the ratio of TP/(TP+FP)
# recall is the ratio of TP/(TP+FN)
# F-beta score can be interpreted as a weighted harmonic mean of the precision and recall
# where an F-beta score reaches its best value at 1 and worst score at 0.
print(classification_report(y_test,y_predict))
```

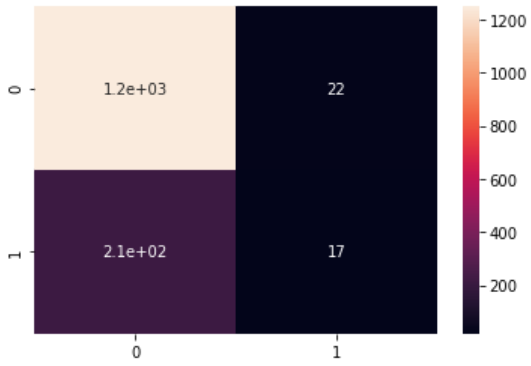
```
              precision    recall  f1-score   support

           0       0.85      0.98      0.91      1271
           1       0.44      0.07      0.13       229

    accuracy                           0.84      1500
   macro avg       0.65      0.53      0.52      1500
weighted avg       0.79      0.84      0.79      1500
```

MINI CHALLENGE #5:

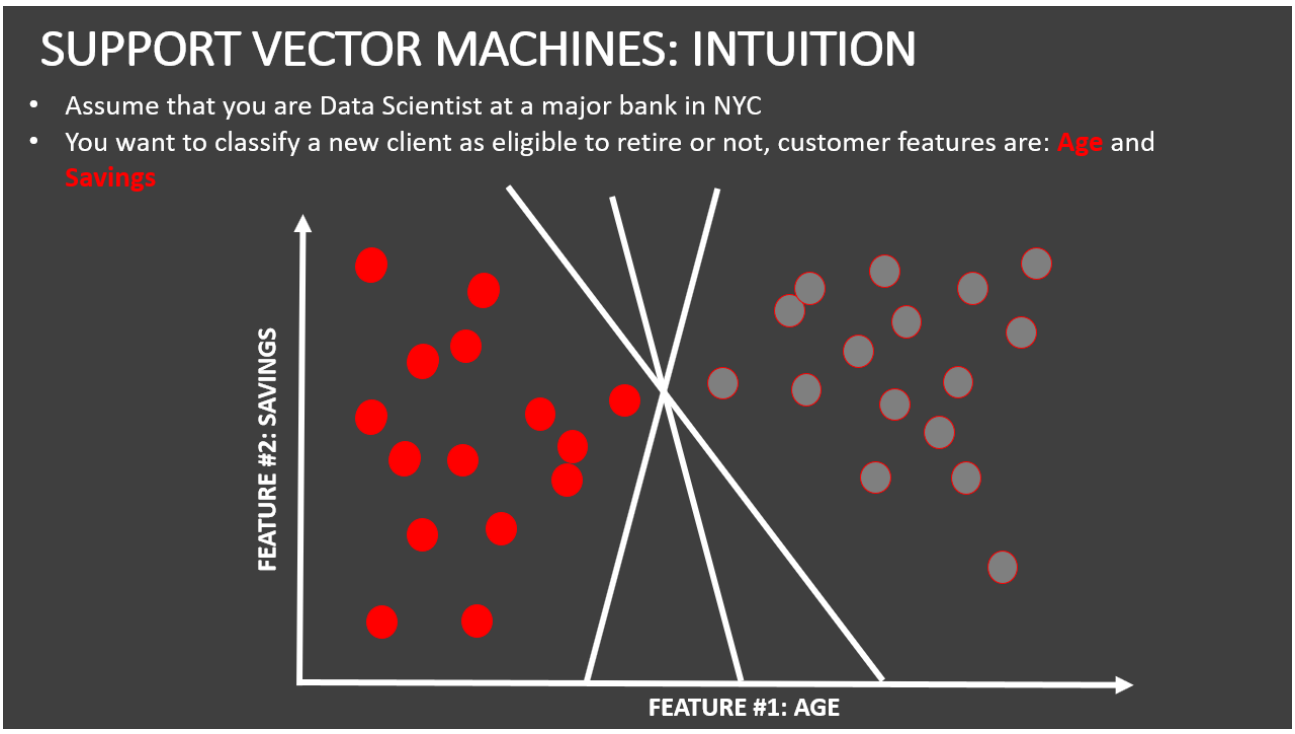
- Print out the confusion Matrix and comment on the results.

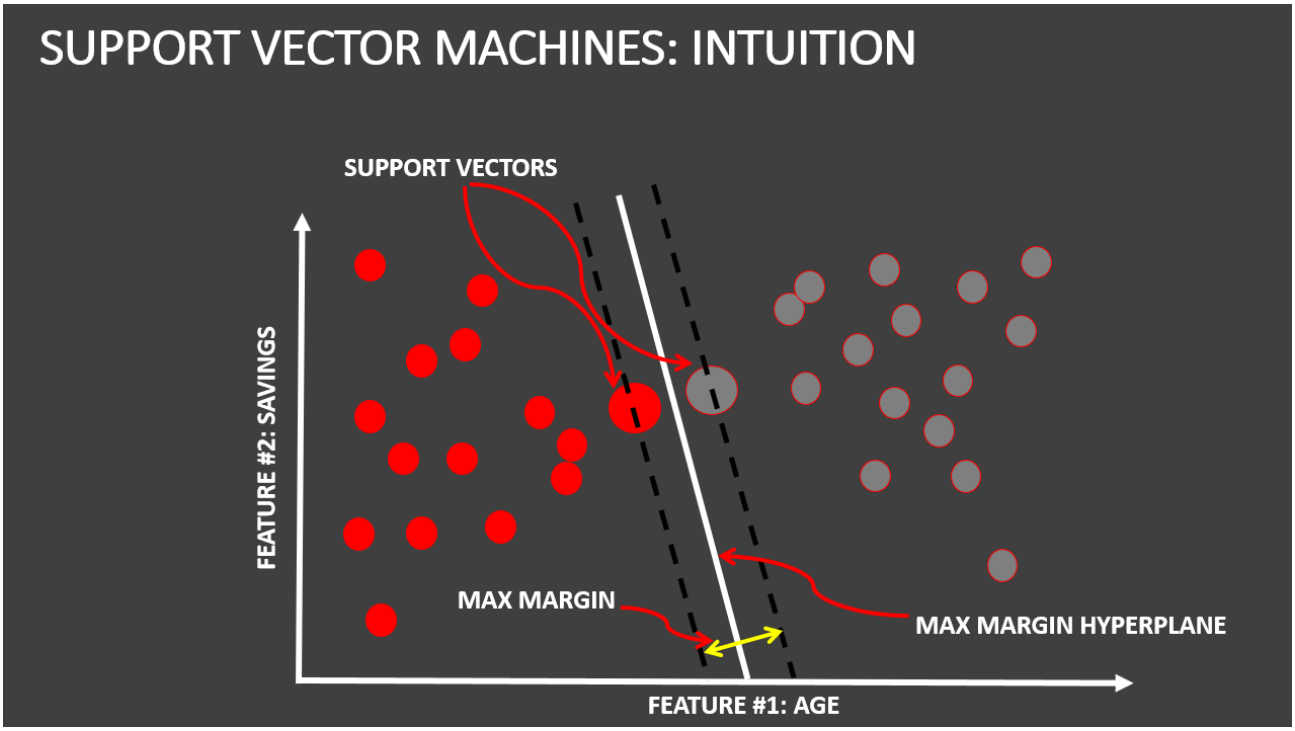
In [34]:
```
cm=confusion_matrix(y_test,y_predict)
sns.heatmap(cm, annot=True)
```

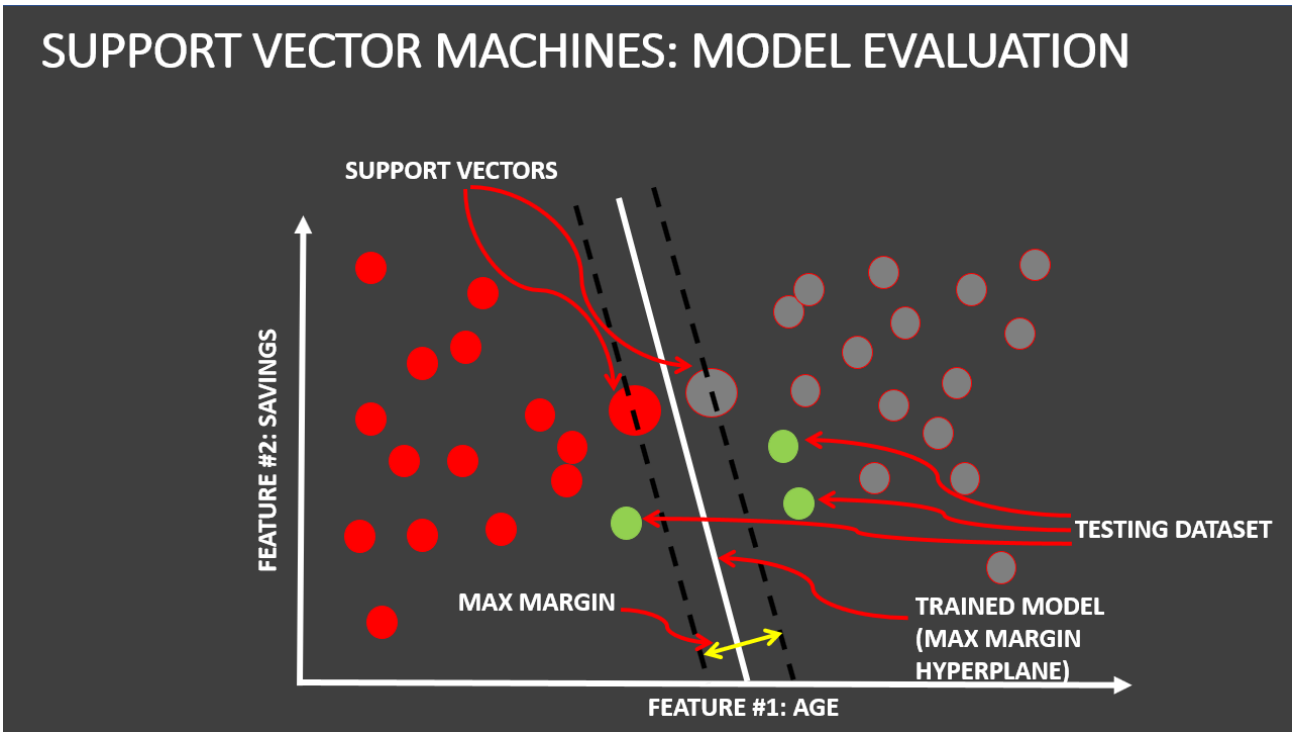
Out[34]: <AxesSubplot:>



TASK #6: TRAIN AND EVALUATE A SUPPORT VECTOR MACHINE CLASSIFIER




SUPPORT VECTOR MACHINES: INTUITION
SUPPORT VECTORS
FEATURE #2: SAVINGS
MAX MARGIN
MAX MARGIN HYPERPLANE
FEATURE #1: AGE


SUPPORT VECTOR MACHINES: MODEL EVALUATION
SUPPORT VECTORS
FEATURE #2: SAVINGS
TESTING DATASET
MAX MARGIN
TRAINED MODEL
(MAX MARGIN
HYPERPLANE)
FEATURE #1: AGE

In [35]:
```
from sklearn.calibration import CalibratedClassifierCV # For probability score output
from sklearn.svm import LinearSVC

model_svm=LinearSVC(max_iter=10000)
model_svm=CalibratedClassifierCV(model_svm)
model_svm.fit(X_train,y_train)
```

```
C:\Users\johnw\anaconda3\lib\site-packages\sklearn\svm\_base.py:985: ConvergenceWarning:

Liblinear failed to converge, increase the number of iterations.

C:\Users\johnw\anaconda3\lib\site-packages\sklearn\svm\_base.py:985: ConvergenceWarning:

Liblinear failed to converge, increase the number of iterations.

C:\Users\johnw\anaconda3\lib\site-packages\sklearn\svm\_base.py:985: ConvergenceWarning:

Liblinear failed to converge, increase the number of iterations.

C:\Users\johnw\anaconda3\lib\site-packages\sklearn\svm\_base.py:985: ConvergenceWarning:

Liblinear failed to converge, increase the number of iterations.

C:\Users\johnw\anaconda3\lib\site-packages\sklearn\svm\_base.py:985: ConvergenceWarning:

Liblinear failed to converge, increase the number of iterations.
```

Out[35]:
```
CalibratedClassifierCV(base_estimator=LinearSVC(max_iter=10000))
```

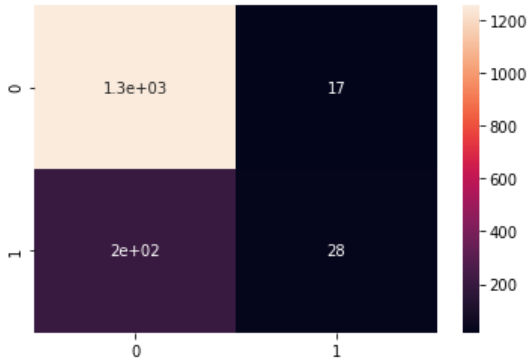
In [36]:
```
y_predict=model_svm.predict(X_test)
```

In [37]:
```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.86      0.99      0.92      1271
           1       0.62      0.12      0.20       229

    accuracy                           0.85      1500
   macro avg       0.74      0.55      0.56      1500
weighted avg       0.83      0.85      0.81      1500
```

In [38]:
```
cm=confusion_matrix(y_test,y_predict)
sns.heatmap(cm, annot=True)
```

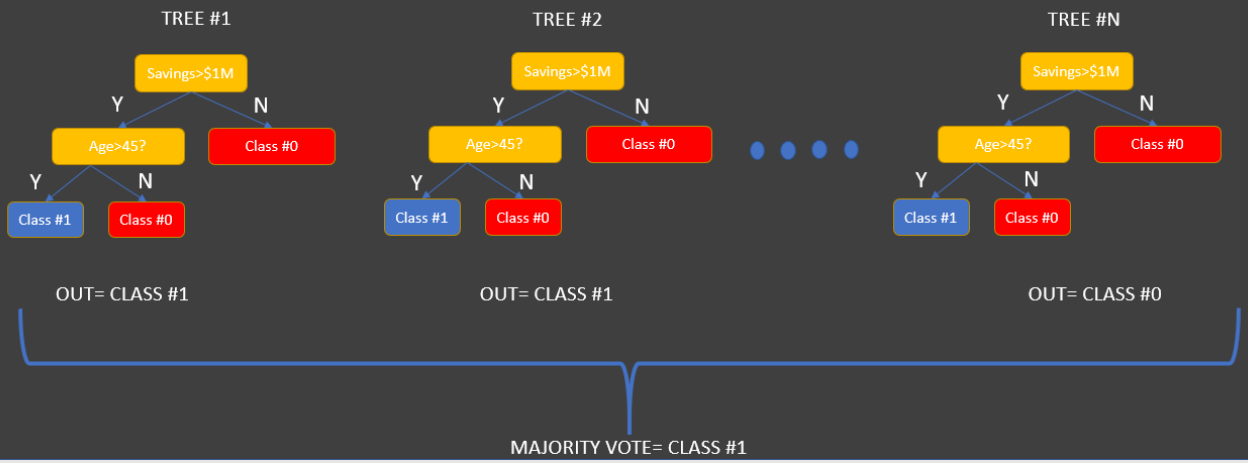
Out[38]:
```
<AxesSubplot:>
```



TASK #7: TRAIN AND EVALUATE A RANDOM FOREST CLASSIFIER

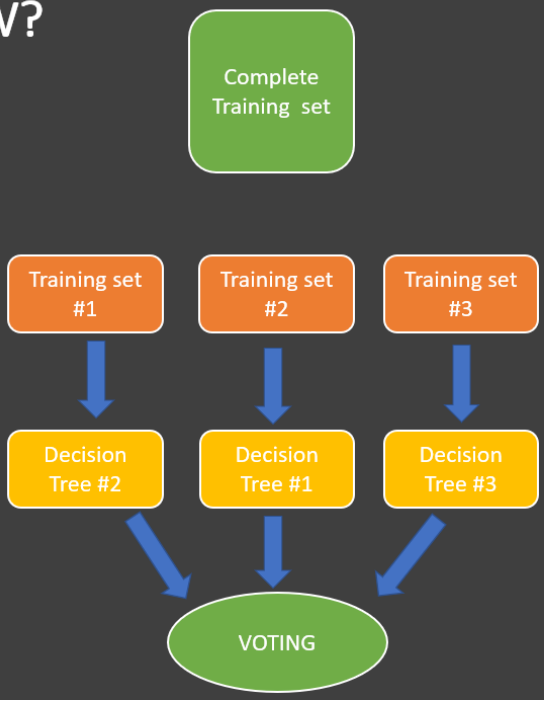
RANDOM FOREST CLASSIFIER: INTUITION

- Random Forest classifier is a type of ensemble algorithm
- It creates a set of decision trees from randomly selected subset of training set
- It then combines votes from different decision trees to decide the final class of the test object.



RANDOM FOREST: WHY AND HOW?

- It overcomes the issues with single decision trees by reducing the effect of noise
- Overcomes overfitting problem by taking average of all the predictions, canceling out biases
- Suppose training set: [X1, X2, X3, X4] with labels: [L1, L2, L3, L4]
- Random Forest creates three decision trees taking inputs as follows:
  - [X1, X2, X3], [X1, X2, X4], [X2, X3, X4]
- Example: Combining votes from a pool of expert, each will bring their own experience and background to solve the problem resulting in a better outcome.
- Runs effectively on large database
- For large data, it produces highly accurate predictions
- Random Forest can maintain accuracy when a large proportion of data is missing



In [39]:
```
from sklearn.ensemble import RandomForestClassifier

model_rf=RandomForestClassifier()
model_rf.fit(X_train,y_train)
```

Out[39]:
```
RandomForestClassifier()
```

In [40]:
```
y_predict=model_rf.predict(X_test)
```

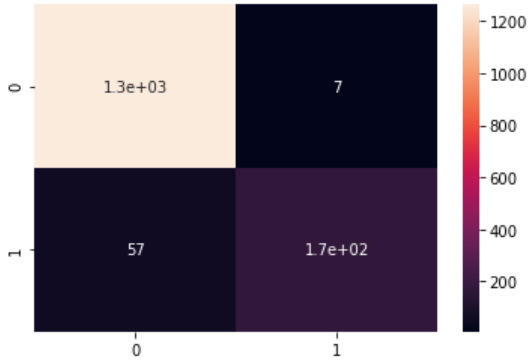
In [ ]:
```

```

In [41]:
```
print(classification_report(y_test,y_predict))
```

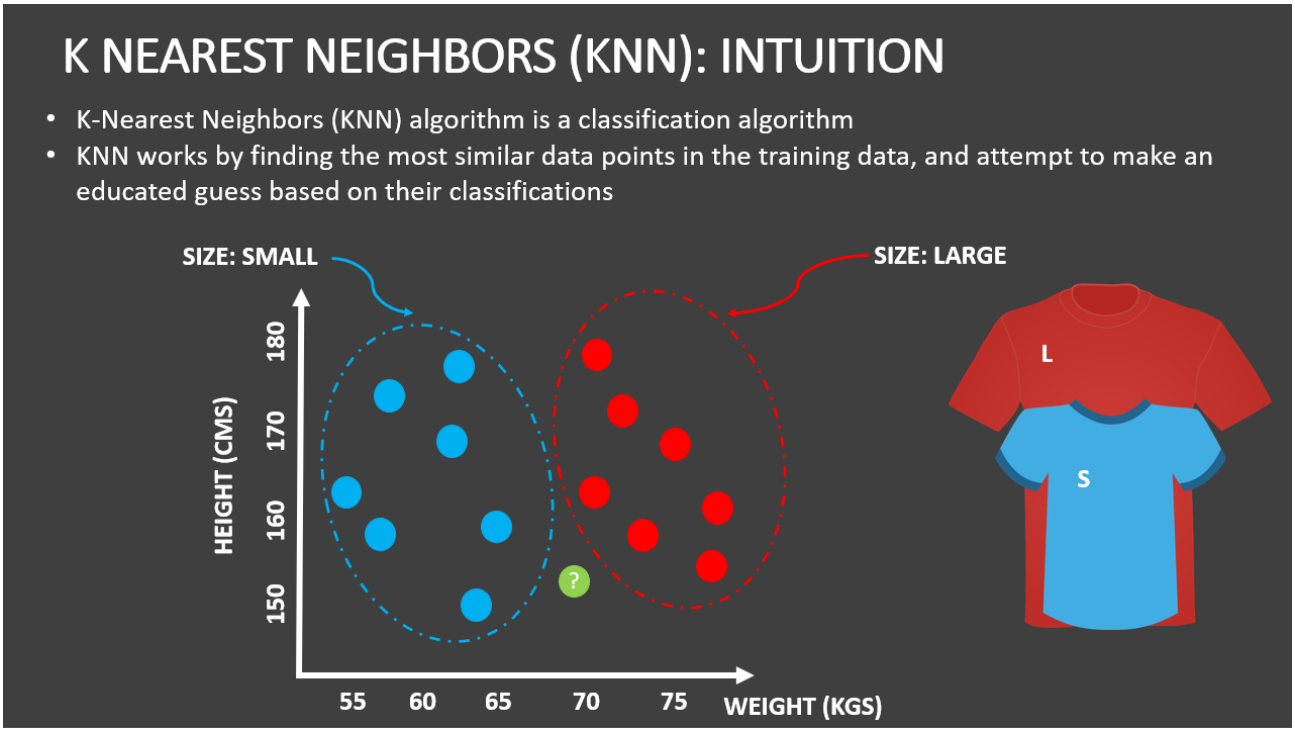
```
              precision    recall  f1-score   support

           0       0.96      0.99      0.98      1271
           1       0.96      0.75      0.84       229

    accuracy                           0.96      1500
   macro avg       0.96      0.87      0.91      1500
weighted avg       0.96      0.96      0.96      1500
```

In [42]:
```
cm=confusion_matrix(y_test,y_predict)
sns.heatmap(cm,annot=True)
```

Out[42]: <AxesSubplot:>



TASK #8: TRAIN AND EVALUATE A K-NEAREST NEIGHBOUR (KNN)

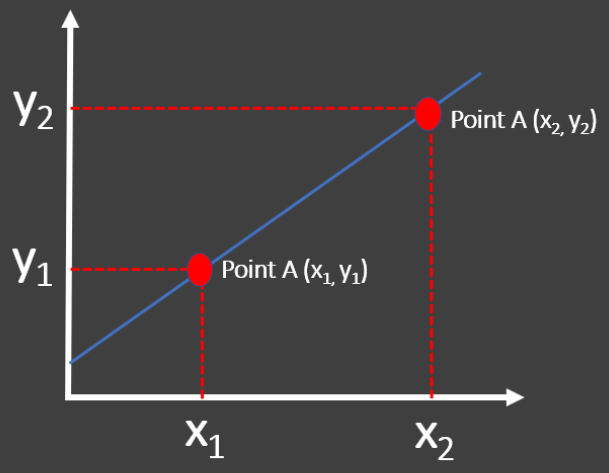


K NEARSET NEIGHBORS (KNN): ALGORITHM STEP

- Select a value for k (e.g.: 1, 2, 3, …..)
- Calculate the Euclidian distance between the point to be classified and every other point in the training data set
- Pick the k closest data points (points with the k smallest distances)
- Run a majority cote among selected data points, the dominating classification is the winner. Point is classified based on the dominant class
- Repeat

EUCILIDEAN DISTANCE: INTUITION

- Euclidean Distance= $\sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$



K NEARSET NEIGHBORS (KNN): EXAMPLE

- KNN will look for the 5 data points that are closest to the new customer data point
- The algorithm will determine which category (class) are these 5 points in
- Since 4 points had class “SMALL” and 1 had “LARGE”, then new customer shall be assigned small size

Height	Weight	T-Shirt Size	Euclidian Dist	Vote
158	58	S	4.242640687	
158	59	S	3.605551275	
158	63	S	3.605551275	
160	59	S	2.236067977	3
160	60	S	1.414213562	1
163	60	S	2.236067977	3
163	61	S	2	2
160	64	L	3.16227766	5
163	64	L	4	
165	61	L	4.123105626	
165	62	L	5.656858249	

New Customer Information:

Height: 161
Weight: 61

Assume, k= 5

Example Source: https://www.listendata.com/2017/12/k-nearest-neighbor-step-by-step-tutorial.html

In [43]:
```
from sklearn.neighbors import KNeighborsClassifier

model_knn=KNeighborsClassifier()
model_knn.fit(X_train,y_train)
```

Out[43]:
```
KNeighborsClassifier()
```

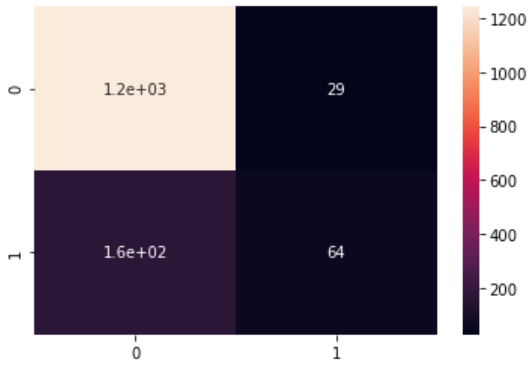
In [44]:
```
y_predict=model_knn.predict(X_test)
```

In [45]:
```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.88      0.98      0.93      1271
           1       0.69      0.28      0.40       229

    accuracy                           0.87      1500
   macro avg       0.79      0.63      0.66      1500
weighted avg       0.85      0.87      0.85      1500
```

In [46]:
```
cm=confusion_matrix(y_test,y_predict)
sns.heatmap(cm,annot=True)
```

Out[46]:
```
<AxesSubplot:>
```



MINI CHALLENGE #6:

Which of the following answers represents the Euclidean distance between the two points A(1, 3) and B(2, 3)?

1. 2
2. 4
3. 1
4. 8

In [ ]:

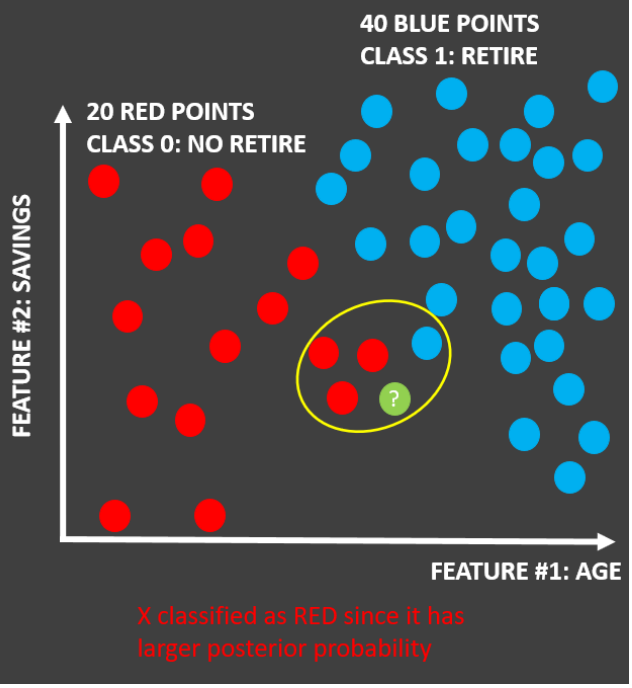
TASK #9: TRAIN AND EVALUATE A NAIVE BAYES CLASSIFIER

NAÏVE BAYES: REVIEW

- Let's combine prior probability and likelihood to create a posterior probability
- Prior probabilities: Suggests that X may be classified as BLUE because there are twice as much as blue points
- Likelihood: Suggests that X is RED because there are more RED points in the vicinity of X
- Baye's rule combines to form a posterior probability

Posterior Probability of X is being RED
= Prior Prob of RED * Likelihood of X being RED
= 20/60 * 3/20= 1/20

Posterior Probability of X is being BLUE
= Prior Prob of BLUE * Likelihood of X being BLUE
= 40/60 * 1/40= 1/60



NAÏVE BAYES: SOME MATH

- Naïve bayes is a classification technique based on Bayes' theorem

$$P(\text{Retire}|X) = \frac{P(X|\text{Retire}) * P(\text{Retire})}{P(X)}$$

LIKELIHOOD — PRIOR PROBABILITY OF RETIRING — MARGINAL LIKELIHOOD

- X: New customer's features; age and savings
- P(Retire|X): probability of customer retiring given his/her features such as age and savings
- P(retire): Prior probability of retiring without any prior knowledge
- P(X|Retire): Likelihood
- P(X): Marginal likelihood, the probability of any point added lies into the circle

P(Retire)= # of Retiring/Total points= 40/60
P(X|Retire)= # of similar observations for retiring/Total # retiring = 1/40
P(X)= # of similar observations/total # points= 4/60

$$P(Retire|X) = \frac{40/60 * 1/40}{4/60} = 1/60 / 4/60 = 0.25$$

CALCULATING THE PROBABILITY OF NON-RETIRING

LIKELIHOOD

$$P(\text{No Retire}|X) = \frac{P(X|\text{No Retire}) * P(\text{No Retire})}{P(X)}$$

PRIOR PROBABILITY OF RETIRING

MARGINAL LIKELIHOOD

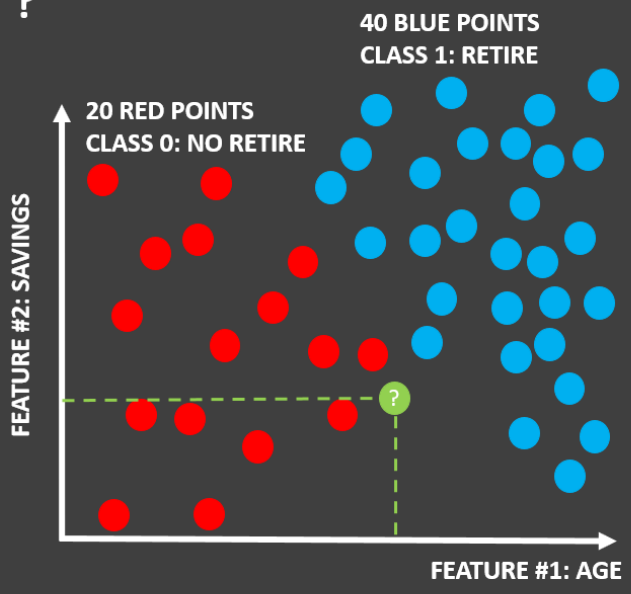
P(No Retire) = # of No Retiring/Total points= 20/60
P(X|No Retire) = # of similar observations for no retiring/Total # no retiring = 3/20
P(X) = # of similar observations/total # points= 4/60

$$P(\text{No Retire}|X) = \frac{20/60 * 3/20}{4/60} = 3/60 / 4/60 = 0.75$$

NOTE: P(Non Retire|X) = 1 – 0.25 = 0.75

NAÏVE BAYES: WHY NAÏVE ?

- It is called naïve because it assumes that the presence of a certain feature in a class is independent of the presence of other features
- EXAMPLE #1: Age/Savings, the assumption is not necessarily true since age and savings might be dependant on each others
- EXAMPLE #2: fruit can be classified as watermelon if its color is green, tastes sweet, and round
- These features might be dependant on each others, however, we assume they are all independent and that's why its NAÏVE.



In [47]:
```
from sklearn.naive_bayes import GaussianNB
```

In [48]:
```
model_gnb=GaussianNB()
model_gnb.fit(X_train, y_train)
```

Out[48]:
```
GaussianNB()
```

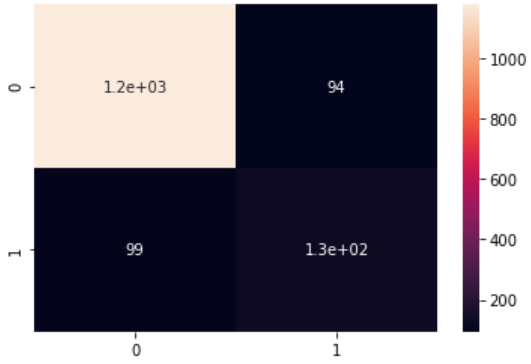
In [49]:
```
y_predict=model_gnb.predict(X_test)
```

In [50]:
```
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.92      0.93      0.92      1271
           1       0.58      0.57      0.57       229

    accuracy                           0.87      1500
   macro avg       0.75      0.75      0.75      1500
weighted avg       0.87      0.87      0.87      1500
```

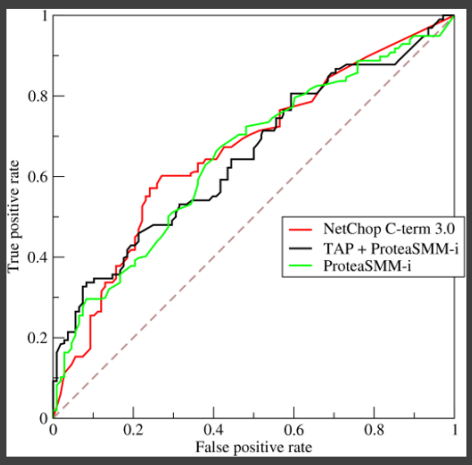
In [51]:
```
cm = confusion_matrix(y_test, y_predict)
sns.heatmap(cm, annot = True)
```

Out[51]: <AxesSubplot:>



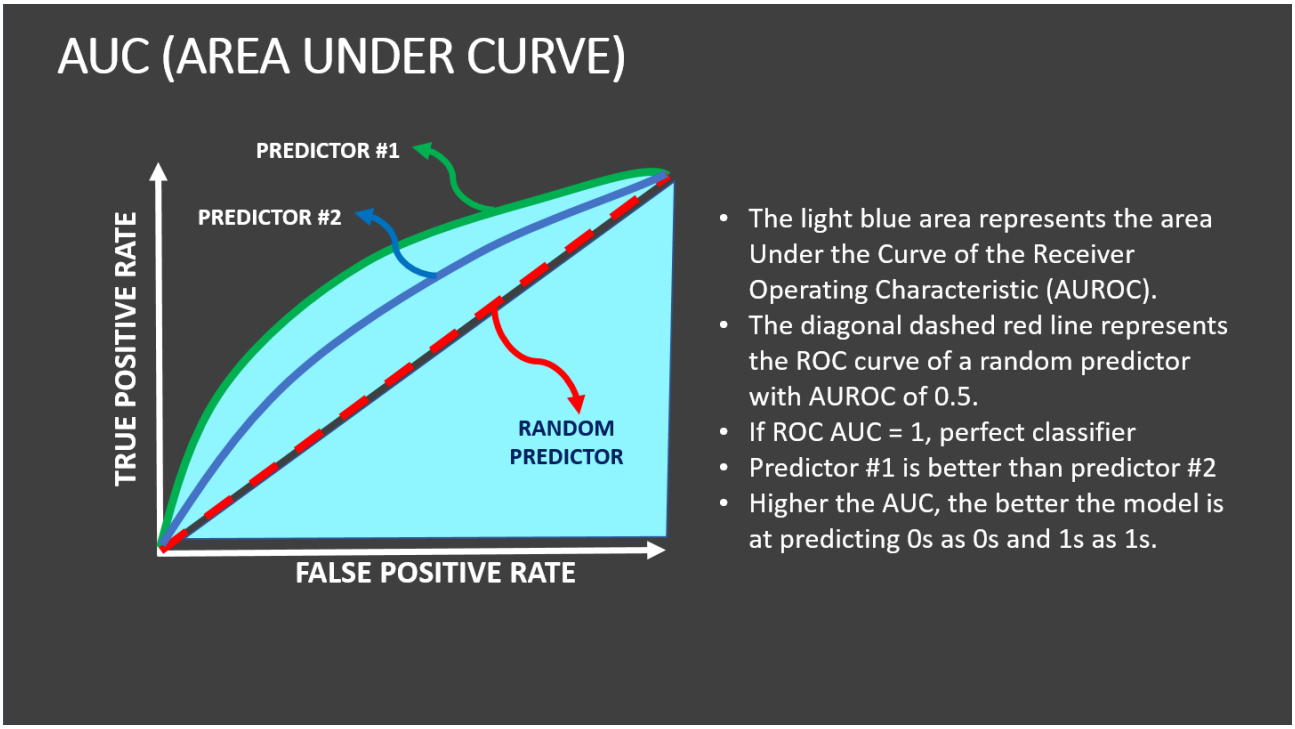
TASK #10: PLOT ROC CURVES FOR THE 5 MODELS AND FIND AUC SCORES

ROC (RECEIVER OPERATING CHARACTERISTIC CURVE)



- ROC Curve is a metric that assesses the model ability to distinguish between binary (0 or 1) classes.
- The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.
- The true-positive rate is also known as sensitivity, recall or probability of detection in machine learning.
- The false-positive rate is also known as the probability of false alarm and can be calculated as (1 – specificity).
- Points above the diagonal line represent good classification (better than random)
- The model performance improves if it becomes skewed towards the upper left corner.

Photo Credit: https://commons.wikimedia.org/wiki/File:Roccurves.png



Check this out: [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html)](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html)

In [52]:
```
model_LR.predict_proba(X_test)
```

Out[52]:
```
array([[0.64195719, 0.35804281],
       [0.75215219, 0.24784781],
       [0.65277082, 0.34722918],
       ...,
       [0.99052979, 0.00947021],
       [0.97478131, 0.02521869],
       [0.40156239, 0.59843761]])
```

In [53]:
```
model_LR.predict_proba(X_test)[:, 1]
```

Out[53]:
```
array([0.35804281, 0.24784781, 0.34722918, ..., 0.00947021, 0.02521869,
       0.59843761])
```

In [54]:
```
y_test
```

Out[54]:
```
3241    1
1552    0
1282    0
2835    0
4119    0
       ..
527     0
4027    1
616     0
3067    0
3190    1
Name: class, Length: 1500, dtype: int64
```

In [55]:
```
from sklearn.metrics import roc_curve
fpr1, tpr1, thresh1 = roc_curve(y_test, model_LR.predict_proba(X_test)[:, 1], pos_label= 1)
```

In [56]:
```
fpr1
```

Out[56]:
```
array([0.        , 0.        , 0.00157356, 0.00157356, 0.00236035,
       0.00236035, 0.00314713, 0.00314713, 0.00629426, 0.00629426,
       0.00708104, 0.00708104, 0.0086546 , 0.0086546 , 0.01022817,
       0.01022817, 0.01101495, 0.01101495, 0.01258851, 0.01258851,
       0.0133753 , 0.0133753 , 0.01416208, 0.01416208, 0.01730921,
       0.01730921, 0.02045633, 0.02045633, 0.02360346, 0.02360346,
       0.02596381, 0.02596381, 0.02911094, 0.02911094, 0.03304485,
       0.03304485, 0.03461841, 0.03461841, 0.03619197, 0.03619197,
       0.03697876, 0.03697876, 0.03855232, 0.03855232, 0.04012589,
       0.04012589, 0.04169945, 0.04169945, 0.04248623, 0.04248623,
       0.04327301, 0.04327301, 0.0440598 , 0.0440598 , 0.04484658,
       0.04484658, 0.04642014, 0.04642014, 0.04720692, 0.04720692,
       0.05035405, 0.05035405, 0.05192762, 0.05192762, 0.0527144 ,
       0.0527144 , 0.05428796, 0.05428796, 0.05586153, 0.05586153,
       0.06215578, 0.06215578, 0.06372935, 0.06372935, 0.06451613,
       0.06451613, 0.06530291, 0.06530291, 0.06687648, 0.06687648,
       0.06923682, 0.06923682, 0.07081039, 0.07081039, 0.07710464,
       0.07710464, 0.07946499, 0.07946499, 0.08025177, 0.08025177,
       0.08103855, 0.08103855, 0.08261212, 0.08261212, 0.09126672,
       0.09126672, 0.0920535 , 0.0920535 , 0.09441385, 0.09441385,
       0.09520063, 0.09520063, 0.09834776, 0.09834776, 0.09992132,
       0.09992132, 0.10621558, 0.10621558, 0.10778914, 0.10778914,
       0.10936271, 0.10936271, 0.1140834 , 0.1140834 , 0.11565696,
       0.11565696, 0.11644375, 0.11644375, 0.11723053, 0.11723053,
       0.122738  , 0.122738  , 0.12745869, 0.12745869, 0.12903226,
       0.12903226, 0.13217939, 0.13217939, 0.13532651, 0.13532651,
       0.14162077, 0.14162077, 0.14240755, 0.14240755, 0.15184894,
       0.15184894, 0.15263572, 0.15263572, 0.15578285, 0.15578285,
       0.15735641, 0.15735641, 0.16601101, 0.16601101, 0.16915814,
       0.16915814, 0.17151849, 0.17151849, 0.17309205, 0.17309205,
       0.17938631, 0.17938631, 0.18253344, 0.18253344, 0.18568057,
       0.18568057, 0.18725413, 0.18725413, 0.19118804, 0.19118804,
       0.19433517, 0.19433517, 0.20220299, 0.20220299, 0.20377655,
       0.20377655, 0.20849725, 0.20849725, 0.21007081, 0.21007081,
       0.21400472, 0.21400472, 0.21715185, 0.21715185, 0.2195122 ,
       0.2195122 , 0.22501967, 0.22501967, 0.22974036, 0.22974036,
       0.24154209, 0.24154209, 0.24390244, 0.24390244, 0.24468922,
       0.24468922, 0.245476  , 0.245476  , 0.24626279, 0.24626279,
       0.24940991, 0.24940991, 0.25334382, 0.25334382, 0.25727773,
       0.25727773, 0.26829268, 0.26829268, 0.27222659, 0.27222659,
       0.27458694, 0.27458694, 0.27537372, 0.27537372, 0.2761605 ,
       0.2761605 , 0.2808812 , 0.2808812 , 0.29504327, 0.29504327,
       0.29661684, 0.29661684, 0.30920535, 0.30920535, 0.30999213,
       0.30999213, 0.32572777, 0.32572777, 0.32966168, 0.32966168,
       0.33202203, 0.33202203, 0.33674272, 0.33674272, 0.33988985,
       0.33988985, 0.34146341, 0.34146341, 0.34697089, 0.34697089,
       0.35877262, 0.35877262, 0.36349331, 0.36349331, 0.3729347 ,
       0.3729347 , 0.38316286, 0.38316286, 0.38473643, 0.38473643,
       0.38552321, 0.38552321, 0.3902439 , 0.3902439 , 0.39260425,
       0.39260425, 0.40519276, 0.40519276, 0.40676633, 0.40676633,
       0.42800944, 0.42800944, 0.43745083, 0.43745083, 0.44767899,
       0.44767899, 0.44925256, 0.44925256, 0.46656176, 0.46656176,
       0.46734854, 0.46734854, 0.47128245, 0.47128245, 0.47915028,
       0.47915028, 0.48387097, 0.48387097, 0.48937844, 0.48937844,
       0.49095201, 0.49095201, 0.49252557, 0.49252557, 0.50668765,
       0.50668765, 0.50983478, 0.50983478, 0.51848938, 0.51848938,
       0.52871755, 0.52871755, 0.53265146, 0.53265146, 0.54523997,
       0.54523997, 0.5483871 , 0.5483871 , 0.57199056, 0.57199056,
       0.57671125, 0.57671125, 0.58300551, 0.58300551, 0.59795437,
       0.59795437, 0.59874115, 0.59874115, 0.60503541, 0.60503541,
       0.62470496, 0.62470496, 0.62549174, 0.62549174, 0.62785208,
       0.62785208, 0.64673485, 0.64673485, 0.64830842, 0.64830842,
       0.65145555, 0.65145555, 0.65381589, 0.65381589, 0.66483084,
       0.66483084, 0.66797797, 0.66797797, 0.67269866, 0.67269866,
       0.68135327, 0.68135327, 0.7018096 , 0.7018096 , 0.70967742,
       0.70967742, 0.7104642 , 0.7104642 , 0.72934697, 0.72934697,
       0.73013375, 0.73013375, 0.74744296, 0.74744296, 0.74901652,
       0.74901652, 0.78756884, 0.78756884, 0.8465775 , 0.8465775 ,
       0.87568843, 0.87568843, 0.8992919 , 0.8992919 , 0.90558615,
       0.90558615, 0.94177813, 0.94177813, 0.95357986, 0.95357986,
       1.        ])
```

In [57]:
```
tpr1
```

Out[57]:
```
array([0.        , 0.00436681, 0.00436681, 0.00873362, 0.00873362,
       0.02183406, 0.02183406, 0.03056769, 0.03056769, 0.03930131,
       0.03930131, 0.04803493, 0.04803493, 0.05240175, 0.05240175,
       0.05676856, 0.05676856, 0.06113537, 0.06113537, 0.06550218,
       0.06550218, 0.069869  , 0.069869  , 0.07423581, 0.07423581,
       0.08733624, 0.08733624, 0.09606987, 0.09606987, 0.10480349,
       0.10480349, 0.10917031, 0.10917031, 0.11790393, 0.11790393,
       0.12227074, 0.12227074, 0.12663755, 0.12663755, 0.13100437,
       0.13100437, 0.13537118, 0.13537118, 0.14847162, 0.14847162,
       0.15283843, 0.15283843, 0.16157205, 0.16157205, 0.16593886,
       0.16593886, 0.1790393 , 0.1790393 , 0.18777293, 0.18777293,
       0.19213974, 0.19213974, 0.19650655, 0.19650655, 0.20087336,
       0.20087336, 0.20524017, 0.20524017, 0.22270742, 0.22270742,
       0.22707424, 0.22707424, 0.24017467, 0.24017467, 0.24454148,
       0.24454148, 0.2489083 , 0.2489083 , 0.25764192, 0.25764192,
       0.26200873, 0.26200873, 0.27074236, 0.27074236, 0.27510917,
       0.27510917, 0.27947598, 0.27947598, 0.28384279, 0.28384279,
       0.28820961, 0.28820961, 0.29257642, 0.29257642, 0.29694323,
       0.29694323, 0.30567686, 0.30567686, 0.31004367, 0.31004367,
       0.31441048, 0.31441048, 0.3231441 , 0.3231441 , 0.32751092,
       0.32751092, 0.33187773, 0.33187773, 0.34061135, 0.34061135,
       0.34497817, 0.34497817, 0.35371179, 0.35371179, 0.3580786 ,
       0.3580786 , 0.36244541, 0.36244541, 0.37554585, 0.37554585,
       0.37991266, 0.37991266, 0.38427948, 0.38427948, 0.40174672,
       0.40174672, 0.40611354, 0.40611354, 0.41484716, 0.41484716,
       0.41921397, 0.41921397, 0.4279476 , 0.4279476 , 0.43231441,
       0.43231441, 0.43668122, 0.43668122, 0.44541485, 0.44541485,
       0.44978166, 0.44978166, 0.45851528, 0.45851528, 0.4628821 ,
       0.4628821 , 0.46724891, 0.46724891, 0.47161572, 0.47161572,
       0.48034934, 0.48034934, 0.49781659, 0.49781659, 0.50218341,
       0.50218341, 0.50655022, 0.50655022, 0.51091703, 0.51091703,
       0.51528384, 0.51528384, 0.51965066, 0.51965066, 0.52401747,
       0.52401747, 0.52838428, 0.52838428, 0.53275109, 0.53275109,
       0.5371179 , 0.5371179 , 0.54148472, 0.54148472, 0.55021834,
       0.55021834, 0.55458515, 0.55458515, 0.56768559, 0.56768559,
       0.5720524 , 0.5720524 , 0.57641921, 0.57641921, 0.58078603,
       0.58078603, 0.58515284, 0.58515284, 0.58951965, 0.58951965,
       0.59388646, 0.59388646, 0.59825328, 0.59825328, 0.60262009,
       0.60262009, 0.6069869 , 0.6069869 , 0.61135371, 0.61135371,
       0.61572052, 0.61572052, 0.62008734, 0.62008734, 0.62882096,
       0.62882096, 0.63318777, 0.63318777, 0.63755459, 0.63755459,
       0.64628821, 0.64628821, 0.65065502, 0.65065502, 0.65938865,
       0.65938865, 0.66375546, 0.66375546, 0.66812227, 0.66812227,
       0.67248908, 0.67248908, 0.6768559 , 0.6768559 , 0.68122271,
       0.68122271, 0.68995633, 0.68995633, 0.69432314, 0.69432314,
       0.69868996, 0.69868996, 0.70305677, 0.70305677, 0.70742358,
       0.70742358, 0.71615721, 0.71615721, 0.72052402, 0.72052402,
       0.72489083, 0.72489083, 0.72925764, 0.72925764, 0.73362445,
       0.73362445, 0.73799127, 0.73799127, 0.74672489, 0.74672489,
       0.7510917 , 0.7510917 , 0.75982533, 0.75982533, 0.76419214,
       0.76419214, 0.76855895, 0.76855895, 0.77292576, 0.77292576,
       0.77729258, 0.77729258, 0.78165939, 0.78165939, 0.7860262 ,
       0.7860262 , 0.79039301, 0.79039301, 0.79475983, 0.79475983,
       0.79912664, 0.79912664, 0.80349345, 0.80349345, 0.80786026,
       0.80786026, 0.81222707, 0.81222707, 0.81659389, 0.81659389,
       0.8209607 , 0.8209607 , 0.82532751, 0.82532751, 0.82969432,
       0.82969432, 0.83842795, 0.83842795, 0.84279476, 0.84279476,
       0.84716157, 0.84716157, 0.8558952 , 0.8558952 , 0.86462882,
       0.86462882, 0.86899563, 0.86899563, 0.87336245, 0.87336245,
       0.87772926, 0.87772926, 0.88209607, 0.88209607, 0.88646288,
       0.88646288, 0.89082969, 0.89082969, 0.89519651, 0.89519651,
       0.89956332, 0.89956332, 0.90393013, 0.90393013, 0.90829694,
       0.90829694, 0.91266376, 0.91266376, 0.91703057, 0.91703057,
       0.92139738, 0.92139738, 0.92576419, 0.92576419, 0.930131  ,
       0.930131  , 0.93449782, 0.93449782, 0.94323144, 0.94323144,
       0.94759825, 0.94759825, 0.95196507, 0.95196507, 0.95633188,
       0.95633188, 0.96069869, 0.96069869, 0.9650655 , 0.9650655 ,
       0.96943231, 0.96943231, 0.97379913, 0.97379913, 0.97816594,
       0.97816594, 0.98253275, 0.98253275, 0.98689956, 0.98689956,
       0.99126638, 0.99126638, 0.99563319, 0.99563319, 1.        ,
       1.        ])
```

In [58]:
```
thresh1
```

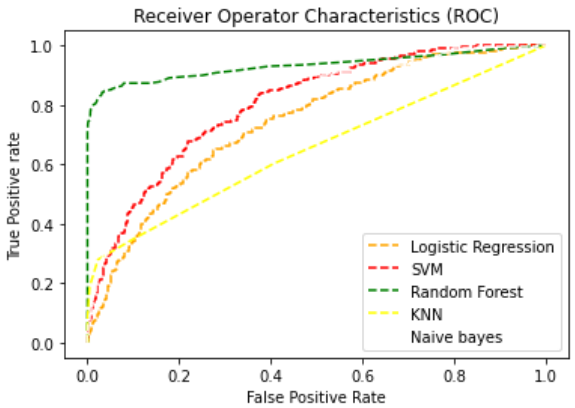
Out[58]:
```
array([1.87835599, 0.87835599, 0.74707459, 0.74212427, 0.71659062,
       0.66225388, 0.64919381, 0.64386058, 0.60517572, 0.59843761,
       0.59825332, 0.58570527, 0.57640431, 0.5682731 , 0.56430438,
       0.55043147, 0.54986925, 0.54434091, 0.53623558, 0.53405683,
       0.53285425, 0.52790987, 0.52588198, 0.52571344, 0.5037831 ,
       0.49605053, 0.48000377, 0.4779864 , 0.46902361, 0.46662017,
       0.46292047, 0.46227307, 0.43887515, 0.43720766, 0.42135244,
       0.41606996, 0.41323697, 0.41108464, 0.40212597, 0.40140581,
       0.39835189, 0.39711234, 0.3968138 , 0.39603722, 0.38986157,
       0.38842065, 0.38717157, 0.38404673, 0.3823115 , 0.38134162,
       0.3799374 , 0.3785115 , 0.37574863, 0.37459606, 0.37422513,
       0.37248862, 0.37055445, 0.36864019, 0.36863367, 0.36806493,
       0.36223015, 0.36048271, 0.35827135, 0.35724495, 0.35569852,
       0.35433029, 0.35241094, 0.35053113, 0.3495858 , 0.34950178,
       0.34239898, 0.34151096, 0.33922566, 0.33460039, 0.3336164 ,
       0.33324678, 0.33169605, 0.32989188, 0.32683108, 0.32369806,
       0.32214099, 0.32108661, 0.32051056, 0.31946862, 0.30659972,
       0.30546718, 0.30223743, 0.29993238, 0.29876655, 0.29742458,
       0.29680543, 0.29613418, 0.29549711, 0.29459006, 0.28504602,
       0.28342023, 0.28219765, 0.28203461, 0.28023149, 0.28021545,
       0.27927504, 0.27821759, 0.27674432, 0.27598569, 0.27539644,
       0.27504417, 0.26867109, 0.26633017, 0.26450179, 0.26397017,
       0.26241127, 0.26088795, 0.25714585, 0.25642696, 0.25598441,
       0.25504233, 0.25488529, 0.25469134, 0.25432282, 0.25023816,
       0.24638738, 0.24627978, 0.24066449, 0.24031421, 0.23983863,
       0.23931006, 0.23589648, 0.23530956, 0.23179417, 0.23161216,
       0.22813646, 0.22803378, 0.22768142, 0.22690664, 0.21989271,
       0.21984534, 0.21962828, 0.21845613, 0.2168279 , 0.21547232,
       0.21471852, 0.21452361, 0.21055107, 0.21002315, 0.20846545,
       0.20781696, 0.20661253, 0.20615605, 0.20586652, 0.20551352,
       0.2026785 , 0.20264305, 0.20111353, 0.20028459, 0.19763286,
       0.19744624, 0.19719094, 0.19654327, 0.19499558, 0.19487913,
       0.19399598, 0.19358157, 0.1889483 , 0.1882946 , 0.18780335,
       0.18778549, 0.18629095, 0.18627854, 0.18548438, 0.18526577,
       0.18333265, 0.1831412 , 0.1811355 , 0.18062161, 0.18008482,
       0.17987219, 0.17776291, 0.17745666, 0.17488584, 0.17464412,
       0.16802615, 0.16790787, 0.1651101 , 0.16429021, 0.1635149 ,
       0.16338514, 0.16321082, 0.16287833, 0.16269826, 0.16198888,
       0.16095722, 0.16094022, 0.15953784, 0.15893115, 0.15785139,
       0.1576233 , 0.15443527, 0.15423008, 0.15274608, 0.15265609,
       0.15168121, 0.15138506, 0.15124576, 0.15088916, 0.1508684 ,
       0.15042439, 0.14904658, 0.1488639 , 0.14465864, 0.14455376,
       0.1439087 , 0.14388359, 0.14033392, 0.13981023, 0.13903486,
       0.13891766, 0.13603765, 0.13571129, 0.13469851, 0.13468813,
       0.13364661, 0.13349879, 0.13239675, 0.13230668, 0.13173264,
       0.13171558, 0.13121286, 0.1307146 , 0.12937999, 0.12907687,
       0.12762411, 0.12751371, 0.1250742 , 0.12493209, 0.12269679,
       0.12267441, 0.1203141 , 0.12024414, 0.12002789, 0.11984173,
       0.11979382, 0.11940938, 0.11835211, 0.11826588, 0.11782899,
       0.11781093, 0.11354878, 0.11303349, 0.11279099, 0.11266745,
       0.10934831, 0.10933988, 0.10699178, 0.10685428, 0.10487209,
       0.10486316, 0.10466736, 0.1045613 , 0.09906975, 0.09906136,
       0.09905353, 0.09892249, 0.09747521, 0.09703544, 0.09621419,
       0.0959508 , 0.09509059, 0.09499097, 0.09423842, 0.0940986 ,
       0.09382834, 0.09375162, 0.09309761, 0.09302987, 0.09006114,
       0.0899818 , 0.08935357, 0.08924562, 0.08743023, 0.08718021,
       0.08522361, 0.08487237, 0.08438237, 0.08426773, 0.08262693,
       0.08260938, 0.08239798, 0.08217429, 0.07887059, 0.07853403,
       0.07774065, 0.07758252, 0.07642918, 0.07628647, 0.07376079,
       0.07365666, 0.07304174, 0.07298449, 0.07183966, 0.07181577,
       0.06924251, 0.06918459, 0.06892176, 0.06868301, 0.06838123,
       0.06836337, 0.06547088, 0.06527792, 0.06515232, 0.06509227,
       0.06425582, 0.06420565, 0.06398194, 0.06397428, 0.06195303,
       0.06190761, 0.06165987, 0.06149775, 0.06101933, 0.06094064,
       0.05984247, 0.05967303, 0.0557572 , 0.05552568, 0.05427297,
       0.05414657, 0.05394659, 0.05383956, 0.05103741, 0.05073316,
       0.05064993, 0.05064553, 0.04819145, 0.04806385, 0.04788888,
       0.04760973, 0.04288649, 0.04283448, 0.03531302, 0.03504294,
       0.03111732, 0.03092268, 0.02822598, 0.02811102, 0.02739892,
       0.02688494, 0.0198671 , 0.01968973, 0.01735677, 0.01719522,
       0.00257936])
```

In [59]:
```
# ROC curve
from sklearn.metrics import roc_curve

fpr1, tpr1, thresh1 = roc_curve(y_test, model_LR.predict_proba(X_test)[:, 1], pos_label = 1)
fpr2, tpr2, thresh2 = roc_curve(y_test, model_svm.predict_proba(X_test)[:, 1], pos_label = 1)
fpr3, tpr3, thresh3 = roc_curve(y_test, model_rf.predict_proba(X_test)[:, 1], pos_label = 1)
fpr4, tpr4, thresh4 = roc_curve(y_test, model_knn.predict_proba(X_test)[:, 1], pos_label = 1)
fpr5, tpr5, thresh5 = roc_curve(y_test, model_gnb.predict_proba(X_test)[:, 1], pos_label = 1)
```

In [60]:
```
# AUC score

from sklearn.metrics import roc_auc_score

auc_score1 = roc_auc_score(y_test, model_LR.predict_proba(X_test)[:, 1])
auc_score2 = roc_auc_score(y_test, model_svm.predict_proba(X_test)[:, 1])
auc_score3 = roc_auc_score(y_test, model_rf.predict_proba(X_test)[:, 1])
auc_score4 = roc_auc_score(y_test, model_knn.predict_proba(X_test)[:, 1])
auc_score5 = roc_auc_score(y_test, model_gnb.predict_proba(X_test)[:, 1])

print("Logistic Regression: ", auc_score1) # Logistic Regression
print("Support Vector Machine: ", auc_score2) # Support Vector Machine
print("Random Forest: ", auc_score3) # Random Forest
print("K-Nearest Neighbors: ", auc_score4) # K-Nearest Neighbors
print("Naive Bayes: ", auc_score5) # Naive Bayes
```

```
Logistic Regression:  0.7418770764690321
Support Vector Machine:  0.8020504433808953
Random Forest:  0.9317698473505371
K-Nearest Neighbors:  0.650378445607248
Naive Bayes:  0.8460243455794185
```

In [61]:
```
plt.plot(fpr1, tpr1, linestyle = "--", color = "orange", label = "Logistic Regression")
plt.plot(fpr2, tpr2, linestyle = "--", color = "red", label = "SVM")
plt.plot(fpr3, tpr3, linestyle = "--", color = "green", label = "Random Forest")
plt.plot(fpr4, tpr4, linestyle = "--", color = "yellow", label = "KNN")
plt.plot(fpr5, tpr5, linestyle = "--", color = "white", label = "Naive bayes")

plt.title('Receiver Operator Characteristics (ROC)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

plt.legend(loc = 'best')
plt.savefig('ROC', dpi = 300)
plt.show()
```



The graph represents that Random Forest algorithm produced the best AUC. Therefore, it is clear that Random Forest model did a better job of classifying the churned/retained telecom customers.

TASK #11: CONCLUSION & PROJECT RECAP

In [62]:
```
y_predict = model_rf.predict(X_test)
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.96      0.99      0.98      1271
           1       0.96      0.75      0.84       229

    accuracy                           0.96      1500
   macro avg       0.96      0.87      0.91      1500
weighted avg       0.96      0.96      0.96      1500
```

Amongst all the trained models, Random Forest Classifier algorithm produced the highest Area under the ROC curve (AUC).

The following scores are the results of the Random Forest Classifier model

1. Accuracy: ~96% label accuracy
2. Precision: ~96% labeled as Retained customers and ~94% labeled as churned customers
3. Recall: ~99% labeled as Retained customers and ~76% labeled as churned customers

Note: We can improve this model even more better by using "Grid Search" method.

Great resource on Grid Search: https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/ (https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/)

GREAT JOB!

MINI CHALLENGE SOLUTIONS

MINI CHALLENGE #1 SOLUTION:

- Name top 5 telecom companies in North America in 2020?

1. AT&T
2. Verizon
3. Comcast
4. Time Warner Cable
5. Bell

https://www.businesschief.com/technology-and-ai/top-five-telecom-giants-north-america (https://www.businesschief.com/technology-and-ai/top-five-telecom-giants-north-america)

MINI CHALLENGE #2 SOLUTION:

- What is the maximum and average daily minutes?

In [63]:
```
# Display the statistical details of the dataframe
telecom_df.describe()
```

Out[63]:

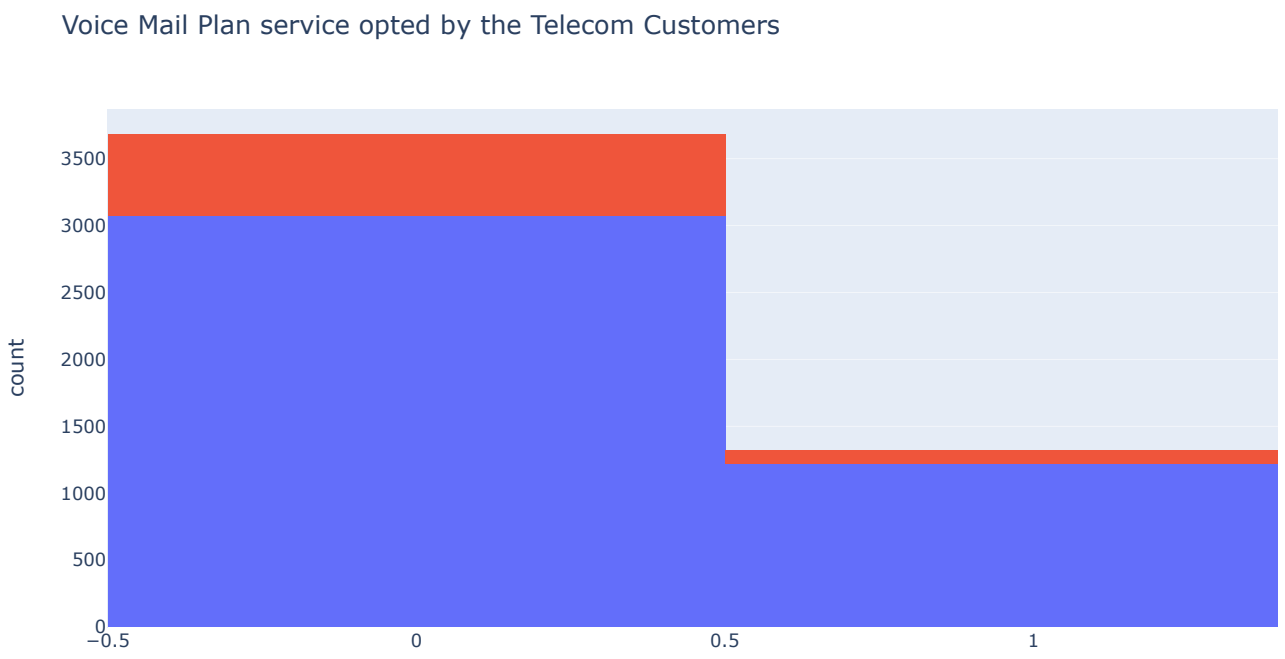
	state	account_length	area_code	phone_number	international_plan	voice_mail_plan	number_vmail_messages	total_day_mi
count	5000.00000	5000.00000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.0(
mean	25.99840	100.25860	436.911400	2499.500000	0.094600	0.264600	7.755200	180.2(
std	14.80348	39.69456	42.209182	1443.520003	0.292691	0.441164	13.546393	53.8(
min	0.00000	1.00000	408.000000	0.000000	0.000000	0.000000	0.000000	0.0(
25%	13.00000	73.00000	408.000000	1249.750000	0.000000	0.000000	0.000000	143.7(
50%	26.00000	100.00000	415.000000	2499.500000	0.000000	0.000000	0.000000	180.1(
75%	39.00000	127.00000	415.000000	3749.250000	0.000000	1.000000	17.000000	216.2(
max	50.00000	243.00000	510.000000	4999.000000	1.000000	1.000000	52.000000	351.5(

8 rows × 21 columns

MINI CHALLENGE #3 SOLUTION:

- Plot the plotly histogram on voice mail plan correlated with Churn feature

In [64]:
```
fig = px.histogram(telecom_df, x = "voice_mail_plan",
                   color = "class",
                   title = "Voice Mail Plan service opted by the Telecom Customers")
fig.show()
```



MINI CHALLENGE #4 SOLUTION:

- Verify that the split was successful

In [65]:
```
X_train.shape
```

Out[65]:
```
(3500, 18)
```

In [66]:
```
X_test.shape
```

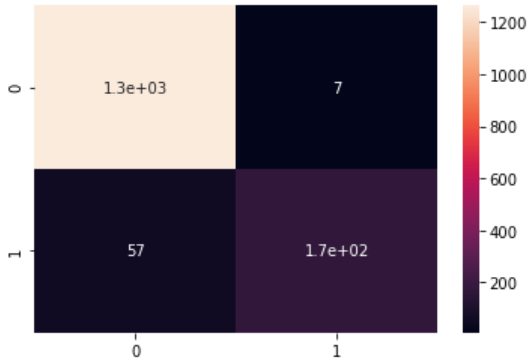
Out[66]:
```
(1500, 18)
```

MINI CHALLENGE #5 SOLUTION:

- Print out the confusion Matrix and comment on the results

In [67]:
```
cm = confusion_matrix(y_test, y_predict)
sns.heatmap(cm, annot = True)
```

Out[67]:
```
<AxesSubplot:>
```



MINI CHALLENGE #6 SOLUTION:

Which of the following answers will be Euclidean distance between the two points A(1, 3) and B(2, 3)?

1. 2
2. 4
3. 1
4. 8

sqrt((1-2)^2 + (3-3)^2)= 1

In [ ]:

In [ ]:

In [ ]:

In [ ]: