Customer Churn Analysis and Prediction for a Telecommunications Company

Project Overview:

The project focuses on analyzing customer churn and building predictive models to understand the factors contributing to customer retention for a telecommunications company. Customer churn, which refers to customers discontinuing their services, is a crucial metric for telecom businesses, given that retaining existing customers is typically more cost-effective than acquiring new ones. To address this challenge, this project employs a comprehensive approach that includes feature engineering, encoding and transforming data, and the utilization of **Artificial Neural Networks (ANN)**.

The project leverages these techniques to analyze a dataset that provides insights into customer behavior and whether they have churned. By transforming and encoding the data, we make it suitable for ANN, a powerful machine learning tool for predictive modeling. ANN can learn complex patterns within the data and help identify factors contributing to customer churn.

Outcome:

Based on the results, we can draw the following conclusions:

Logistic Regression Model:

The logistic regression model achieved an accuracy of 80%, indicating that it correctly classified 80% of the samples. The model's performance is reasonably good, with higher precision and recall for Churn=No compared to Churn=Yes. The F1-score, which balances precision and recall, is also higher for Churn=No.

OLS Regression Model:

The OLS regression model explains approximately 21.5% of the variance in the dependent variable (Churn). The model suggests that both tenure and monthly charges have statistically significant effects on churn. The coefficients provide insight into how these variables influence churn.

In [1]:
```
import pandas as pd
import os
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
import statsmodels.api as sm
import statsmodels.api as smf
import statsmodels.stats.api as sms
```

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import roc_curve, roc_auc_score, auc
from sklearn.metrics import classification_report, confusion_matrix
import tensorflow as tf
from tensorflow import keras

warnings.filterwarnings("ignore")
%matplotlib inline
```

In [2]:

```
directory = "F:\Jupyter\Telecom Churn Analysis"
filename = "data_telco_customer_churn.csv"

file_path = os.path.join(directory, filename)

if os.path.exists(file_path):
    data = pd.read_csv(file_path)
    print(data.head())

else:
    print("The file does not exists")
```

```
  Dependents  tenure       OnlineSecurity         OnlineBackup  \
0        Yes       9                   No                   No
1         No      14                   No                  Yes
2         No      64                  Yes                   No
3         No      72                  Yes                  Yes
4         No       3  No internet service  No internet service

  InternetService     DeviceProtection          TechSupport        Contract  \
0             DSL                  Yes                  Yes  Month-to-month
1     Fiber optic                  Yes                   No  Month-to-month
2             DSL                  Yes                  Yes        Two year
3             DSL                  Yes                  Yes        Two year
4              No  No internet service  No internet service  Month-to-month

  PaperlessBilling  MonthlyCharges Churn
0              Yes           72.90   Yes
1              Yes           82.65    No
2               No           47.85   Yes
3               No           69.65    No
4              Yes           23.60    No
```

In [3]:

```
data.sample(5)
```

Out[3]:

	Dependents	tenure	OnlineSecurity	OnlineBackup	InternetService	DeviceProtection	TechSupport	Contract	PaperlessBilling	MonthlyC
107	No	36	No	No	Fiber optic	No	No	Month-to-month	Yes	
4795	Yes	47	Yes	No	Fiber optic	Yes	No	Month-to-month	No	
335	No	65	No internet service	No internet service	No	No internet service	No internet service	Two year	Yes	
4843	No	46	Yes	No	DSL	Yes	Yes	Two year	No	
1324	No	4	Yes	No	DSL	Yes	Yes	Month-to-month	No	

In [4]:

```
data.isnull().sum()
```

Out[4]:

```
Dependents          0
tenure              0
OnlineSecurity      0
OnlineBackup        0
InternetService     0
DeviceProtection    0
TechSupport         0
Contract            0
PaperlessBilling    0
MonthlyCharges      0
Churn               0
dtype: int64
```

In [5]:

```
data.info
```

Out[5]:
```
<bound method DataFrame.info of      Dependents  tenure       OnlineSecurity         OnlineBackup  \
0           Yes       9                   No                   No
1            No      14                   No                  Yes
2            No      64                  Yes                   No
3            No      72                  Yes                  Yes
4            No       3  No internet service  No internet service
...         ...     ...                  ...                  ...
4925         No      15                   No                   No
4926        Yes      10                   No                   No
4927         No      58  No internet service  No internet service
4928         No       1                   No                   No
4929        Yes       4  No internet service  No internet service

     InternetService     DeviceProtection          TechSupport  \
0                DSL                  Yes                  Yes
1        Fiber optic                  Yes                   No
2                DSL                  Yes                  Yes
3                DSL                  Yes                  Yes
4                 No  No internet service  No internet service
...              ...                  ...                  ...
4925     Fiber optic                  Yes                  Yes
4926     Fiber optic                  Yes                  Yes
4927              No  No internet service  No internet service
4928     Fiber optic                   No                   No
4929              No  No internet service  No internet service

            Contract PaperlessBilling  MonthlyCharges Churn
0     Month-to-month              Yes           72.90   Yes
1     Month-to-month              Yes           82.65    No
2           Two year               No           47.85   Yes
3           Two year               No           69.65    No
4     Month-to-month              Yes           23.60    No
...              ...              ...             ...   ...
4925  Month-to-month              Yes          103.45    No
4926  Month-to-month              Yes           91.10    No
4927        Two year               No           20.75    No
4928  Month-to-month              Yes           69.75   Yes
4929  Month-to-month               No           20.40    No

[4930 rows x 11 columns]>
```

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

Out[6]:
```
Dependents           object
tenure                int64
OnlineSecurity       object
OnlineBackup         object
InternetService      object
DeviceProtection     object
TechSupport          object
Contract             object
PaperlessBilling     object
MonthlyCharges      float64
Churn                object
dtype: object
```

In [7]:
```
data.MonthlyCharges.values
```

Out[7]:
```
array([72.9 , 82.65, 47.85, ..., 20.75, 69.75, 20.4 ])
```

In [8]:
```
pd.to_numeric(data.MonthlyCharges, errors='coerce')
```

Out[8]:
```
0        72.90
1        82.65
2        47.85
3        69.65
4        23.60
         ...
4925    103.45
4926     91.10
4927     20.75
4928     69.75
4929     20.40
Name: MonthlyCharges, Length: 4930, dtype: float64
```

In [9]:
```
data.shape
```

Out[9]:
```
(4930, 11)
```

In [10]:
```
data.size
```

Out[10]:
```
54230
```

In [11]:
```
data.columns
```

Out[11]:
```
Index(['Dependents', 'tenure', 'OnlineSecurity', 'OnlineBackup',
       'InternetService', 'DeviceProtection', 'TechSupport', 'Contract',
       'PaperlessBilling', 'MonthlyCharges', 'Churn'],
      dtype='object')
```

In [12]:
```
from sklearn.preprocessing import LabelEncoder

label = LabelEncoder()
data['Churn'] = label.fit_transform(data['Churn'])
```

In [13]:
```
data['Churn'].dtypes
```

Out[13]:
```
dtype('int32')
```

In [14]:
```
target = data['Churn']
independent_vars = data[['tenure', 'MonthlyCharges']]
independent_vars = sm.add_constant(independent_vars)

model = sm.OLS(target, independent_vars)
result = model.fit()

print(result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Churn   R-squared:                       0.215
Model:                            OLS   Adj. R-squared:                  0.215
Method:                 Least Squares   F-statistic:                     674.4
Date:                Fri, 20 Oct 2023   Prob (F-statistic):          1.31e-259
Time:                        21:30:27   Log-Likelihood:                -2377.8
No. Observations:                4930   AIC:                             4762.
Df Residuals:                    4927   BIC:                             4781.
Df Model:                           2
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const              0.2357      0.014     16.703      0.000       0.208       0.263
tenure            -0.0078      0.000    -33.389      0.000      -0.008      -0.007
MonthlyCharges     0.0044      0.000     22.871      0.000       0.004       0.005
==============================================================================
Omnibus:                      614.374   Durbin-Watson:                   2.013
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              456.942
Skew:                           0.645   Prob(JB):                    5.97e-100
Kurtosis:                       2.251   Cond. No.                         200.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Coefficients: The coefficients represent the estimated effect of each independent variable on the dependent variable. In your model:

- The constant (intercept) is 0.2357.
- The coefficient for 'tenure' is approximately -0.0078, indicating that, on average, for each unit increase in 'tenure,' - 'Churn' decreases by 0.0078 units.
- The coefficient for 'MonthlyCharges' is approximately 0.0044, indicating that, on average, for each unit increase in 'MonthlyCharges,' 'Churn' increases by 0.0044 units.

t-statistics and P>|t| (P-values): These values test the significance of each coefficient. In this model, all coefficients have very low p-values, indicating that they are statistically significant.

Omnibus, Durbin-Watson, Jarque-Bera: These are tests of the model's assumptions. For example, the Omnibus and Jarque-Bera tests assess the normality of residuals, and the Durbin-Watson test checks for autocorrelation. In this model, the low p-values in the Omnibus and Jarque-Bera tests suggest that residuals may not be normally distributed, and the Durbin-Watson value of 2.013 indicates possible positive autocorrelation.

Kurtosis: This measures the "tailedness" of the residual distribution. A value of 2.251 suggests slightly heavy tails.

Cond. No.: This is a measure of multicollinearity. A value of 200 suggests there might be some multicollinearity in your model.

In [15]:
```
# Checking Linear Assumptions

y_pred = result.predict(independent_vars)
residuals = target - y_pred
sms.linear_harvey_collier(result)
```

Out[15]:
```
TtestResult(statistic=-1.7147879179528045, pvalue=0.0864470136711781, df=4926)
```

In [16]:
```
print("Total 'Yes': ")
data[data['Churn'] == 0].value_counts().sum()
```

```
Total 'Yes':
```

Out[16]: 3614

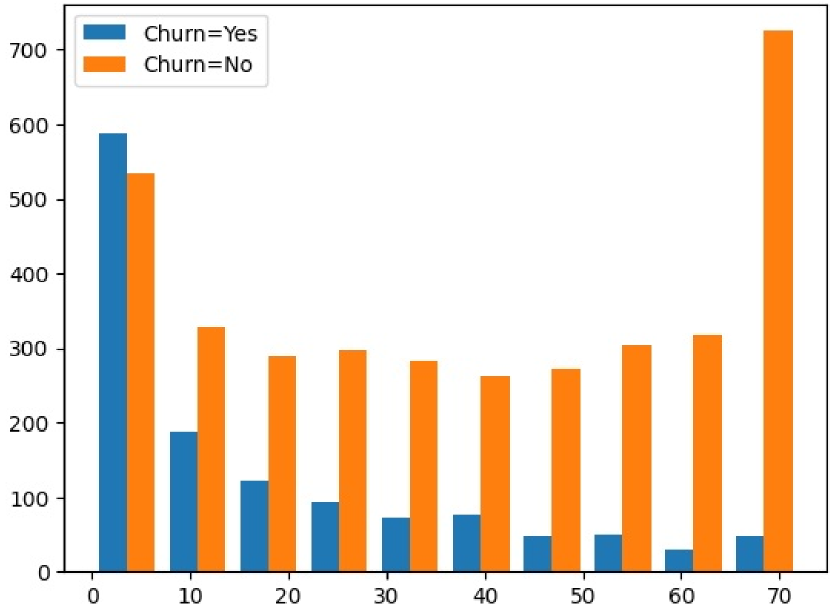
In [17]:
```
print("Total 'No': ")

data[data['Churn'] == 1].value_counts().sum()
```

```
Total 'No':
```

Out[17]: 1316

In [18]:
```
tenure_no = data[data['Churn'] == 0].tenure
tenure_yes = data[data['Churn'] == 1].tenure

plt.hist([tenure_yes, tenure_no], label=['Churn=Yes', 'Churn=No'])
plt.legend()
```

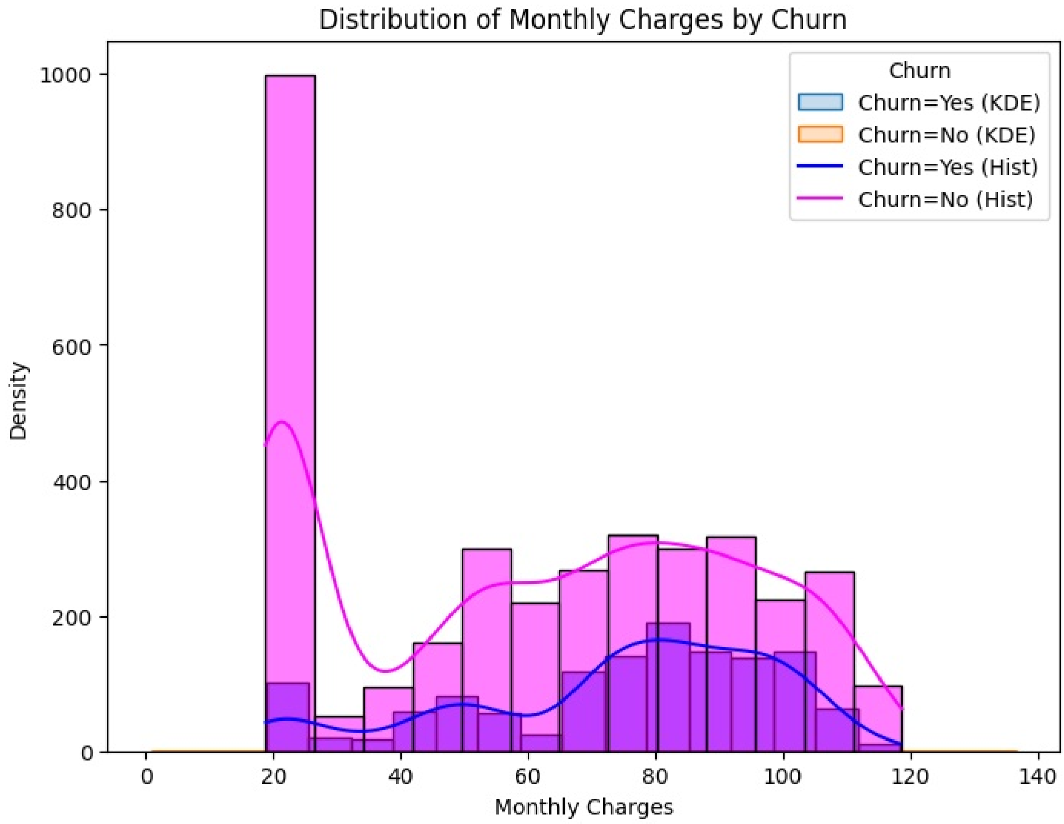
Out[18]: <matplotlib.legend.Legend at 0x18ff1cfe4d0>



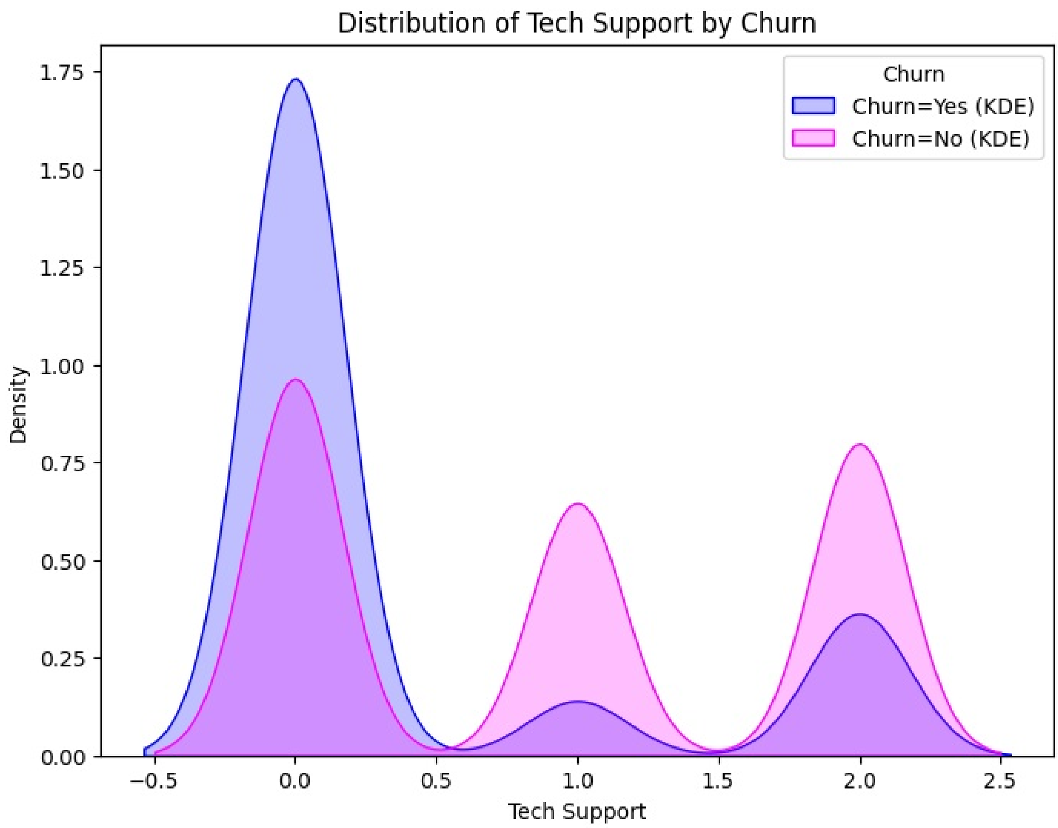
In [19]:
```
data['TechSupport'] = label.fit_transform(data['TechSupport'])
```

In [20]:
```
# Separate 'MonthlyCharges' data for Churn=Yes and Churn=No
monthly_charges_yes = data[data['Churn'] == 1]['MonthlyCharges']
monthly_charges_no = data[data['Churn'] == 0]['MonthlyCharges']

# Separate 'TechSupport' data for Churn=Yes and Churn=No
tech_support_yes = data[data['Churn'] == 1]['TechSupport']
tech_support_no = data[data['Churn'] == 0]['TechSupport']
```

In [21]:
```
fig, ax = plt.subplots(figsize=(8, 6))

# Plot KDE curves for 'Churn=Yes' and 'Churn=No' for 'MonthlyCharges'
sns.kdeplot(monthly_charges_yes, label='Churn=Yes', shade=True, ax=ax)
sns.kdeplot(monthly_charges_no, label='Churn=No', shade=True, ax=ax)

sns.histplot(monthly_charges_yes, color='b', kde=True, ax=ax)
sns.histplot(monthly_charges_no, color='magenta', kde=True, ax=ax)


ax.set_xlabel('Monthly Charges')
ax.set_ylabel('Density')
ax.set_title('Distribution of Monthly Charges by Churn')
ax.legend(title='Churn', labels=['Churn=Yes (KDE)', 'Churn=No (KDE)', 'Churn=Yes (Hist)', 'Churn=No (Hist)'])
plt.show()
```



In [22]:
```
fig, ax = plt.subplots(figsize=(8, 6))

# Plot KDE curves for 'Churn=Yes' and 'Churn=No' for 'TechSupport'
sns.kdeplot(tech_support_yes, label='Churn=Yes', shade=True, color='b', ax=ax)
sns.kdeplot(tech_support_no, label='Churn=No', shade=True, color='magenta', ax=ax)

ax.set_xlabel('Tech Support')
ax.set_ylabel('Density')
ax.set_title('Distribution of Tech Support by Churn')
ax.legend(title='Churn', labels=['Churn=Yes (KDE)', 'Churn=No (KDE)'])

plt.show()
```



In [23]:
```
for column in data:
    print(f"{column}: {data[column].unique()}")
```

```
Dependents: ['Yes' 'No']
tenure: [ 9 14 64 72  3 40 17 11  8 47 18  5  1 48 13 58  7  4 70 34 31 37 15 71
 10 43 22 33 69 54 63 55 66 56 32 26 24  2 51 23 49 28 36 45 42  6 61 59
 67 65  0 16 52 41 25 62 20 50 30 60 19 35 57 27 44 53 12 46 39 29 38 68
 21]
OnlineSecurity: ['No' 'Yes' 'No internet service']
OnlineBackup: ['No' 'Yes' 'No internet service']
InternetService: ['DSL' 'Fiber optic' 'No']
DeviceProtection: ['Yes' 'No internet service' 'No']
TechSupport: [2 0 1]
Contract: ['Month-to-month' 'Two year' 'One year']
PaperlessBilling: ['Yes' 'No']
MonthlyCharges: [ 72.9   82.65  47.85 ...  58.45  23.65 108.5 ]
Churn: [1 0]
```

In [24]:
```
for column in data:
    if data[column].dtypes == 'object':
        print(f"{column}: {data[column].unique()}")
```

```
Dependents: ['Yes' 'No']
OnlineSecurity: ['No' 'Yes' 'No internet service']
OnlineBackup: ['No' 'Yes' 'No internet service']
InternetService: ['DSL' 'Fiber optic' 'No']
DeviceProtection: ['Yes' 'No internet service' 'No']
Contract: ['Month-to-month' 'Two year' 'One year']
PaperlessBilling: ['Yes' 'No']
```

Encoding All the Columns Based On Their Given Datatypes and Values

In [25]:
```
from sklearn.base import BaseEstimator, TransformerMixin

class CategoricalEncoder(BaseEstimator, TransformerMixin):
    def __init__(self, columns):
        self.columns = columns
        self.label_encoders = {}

    def fit(self, X, y=None):
        for col in self.columns:
            le = LabelEncoder()
            le.fit(X[col])
            self.label_encoders[col] = le
        return self

    def transform(self, X):
        X_copy = X.copy()
        for col in self.columns:
            X_copy[col] = self.label_encoders[col].transform(X[col])
        return X_copy

columns_to_encode = ['Dependents', 'OnlineSecurity', 'OnlineBackup', 'InternetService', 'DeviceProtection', 'Co

encoder = CategoricalEncoder(columns=columns_to_encode)

encoded_data = encoder.fit_transform(data)
```

In [26]:
```
encoded_data
```

Out[26]:

	Dependents	tenure	OnlineSecurity	OnlineBackup	InternetService	DeviceProtection	TechSupport	Contract	PaperlessBilling	MonthlyC
0	1	9	0	0	0	2	2	0	1	
1	0	14	0	2	1	2	0	0	1	
2	0	64	2	0	0	2	2	2	0	
3	0	72	2	2	0	2	2	2	0	
4	0	3	1	1	2	1	1	0	1	
...	...	...	...	...	...	...	...	...	...	
4925	0	15	0	0	1	2	2	0	1	
4926	1	10	0	0	1	2	2	0	1	
4927	0	58	1	1	2	1	1	2	0	
4928	0	1	0	0	1	0	0	0	1	
4929	1	4	1	1	2	1	1	0	0	

4930 rows × 11 columns

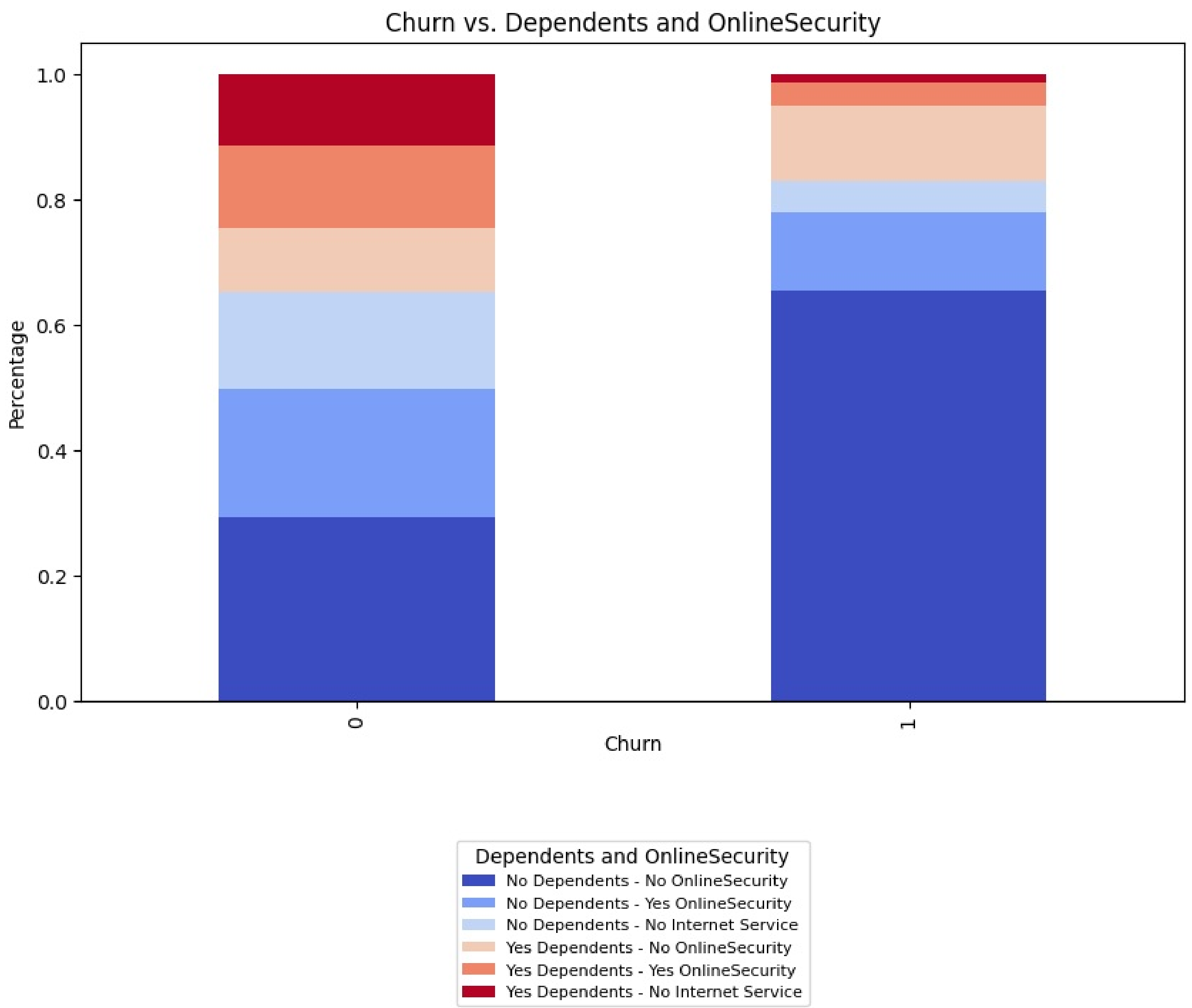
In [27]:
```
encoded_data.dtypes
```

Out[27]:
```
Dependents            int32
tenure                int64
OnlineSecurity        int32
OnlineBackup          int32
InternetService       int32
DeviceProtection      int32
TechSupport           int32
Contract              int32
PaperlessBilling      int32
MonthlyCharges      float64
Churn                 int32
dtype: object
```

In [28]:
```
for column in data:
    if encoded_data[column].dtypes == 'int':
        print(f"{column}: {encoded_data[column].unique()}")
```

```
Dependents: [1 0]
OnlineSecurity: [0 2 1]
OnlineBackup: [0 2 1]
InternetService: [0 1 2]
DeviceProtection: [2 1 0]
TechSupport: [2 0 1]
Contract: [0 2 1]
PaperlessBilling: [1 0]
Churn: [1 0]
```

In [29]:
```
# Subset the data for 'Churn', 'Dependents', and 'OnlineSecurity' columns
subset_data = encoded_data[['Churn', 'Dependents', 'OnlineSecurity']]

# Define labels for 'Dependents' and 'OnlineSecurity' categories
dependent_labels = {0: 'No', 1: 'Yes'}
online_security_labels = {0: 'No', 1: 'Yes', 2: 'No Internet Service'}

subset_data['Dependents'] = subset_data['Dependents'].map(dependent_labels)
subset_data['OnlineSecurity'] = subset_data['OnlineSecurity'].map(online_security_labels)

pivot = subset_data.pivot_table(index='Churn', columns=['Dependents', 'OnlineSecurity'], aggfunc=len, fill_valu

ax = pivot.div(pivot.sum(axis=1), axis=0).plot(kind='bar', stacked=True, colormap='coolwarm', figsize=(10, 6))
plt.xlabel('Churn')
plt.ylabel('Percentage')
plt.title('Churn vs. Dependents and OnlineSecurity')

legend_labels = ['No Dependents - No OnlineSecurity', 'No Dependents - Yes OnlineSecurity', 'No Dependents - No
                 'Yes Dependents - No OnlineSecurity', 'Yes Dependents - Yes OnlineSecurity', 'Yes Dependents -
legend = ax.legend(legend_labels, title='Dependents and OnlineSecurity', bbox_to_anchor=(0.5, -0.2), loc='upper

plt.show()
```



In [30]:
```
X = data.drop('Churn', axis='columns')
y = data['Churn']
```

In [31]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5)
```

In [32]:
```
X_train.shape
```

Out[32]:
```
(3944, 10)
```

In [33]:
```
X_test.shape
```

Out[33]:
```
(986, 10)
```

In [34]:
```
len(X_train.columns)
```

Out[34]:
```
10
```

In [35]:
```
# Setting Up Neural Networks
```

In [36]:
```
X = encoded_data.drop(columns=['Churn']).values  # Input features
y = encoded_data['Churn'].values  # Target variable

X = X.astype(np.float32)
y = y.astype(np.float32)

model = keras.Sequential([
    keras.layers.Dense(10, input_shape=(X.shape[1],), activation='relu'),
    keras.layers.Dense(5, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'],
              )

model.fit(X, y, epochs=100)
```

```
Epoch 1/100
```

```
155/155 [==============================] - 1s 3ms/step - loss: 2.1025 - accuracy: 0.6941
Epoch 2/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4768 - accuracy: 0.7544
Epoch 3/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4546 - accuracy: 0.7793
Epoch 4/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4423 - accuracy: 0.7846
Epoch 5/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4346 - accuracy: 0.7886
Epoch 6/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4356 - accuracy: 0.7874
Epoch 7/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4318 - accuracy: 0.7915
Epoch 8/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4294 - accuracy: 0.7890
Epoch 9/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4313 - accuracy: 0.7901
Epoch 10/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4262 - accuracy: 0.7961
Epoch 11/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4315 - accuracy: 0.7899
Epoch 12/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4306 - accuracy: 0.7901
Epoch 13/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4252 - accuracy: 0.7935
Epoch 14/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4277 - accuracy: 0.7870
Epoch 15/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4272 - accuracy: 0.7972
Epoch 16/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4279 - accuracy: 0.7933
Epoch 17/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4243 - accuracy: 0.7951
Epoch 18/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4296 - accuracy: 0.7941
Epoch 19/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4236 - accuracy: 0.7947
Epoch 20/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4247 - accuracy: 0.7963
Epoch 21/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4259 - accuracy: 0.7888
Epoch 22/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4239 - accuracy: 0.7935
Epoch 23/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4244 - accuracy: 0.7917
Epoch 24/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4230 - accuracy: 0.7951
Epoch 25/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4261 - accuracy: 0.7941
Epoch 26/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4240 - accuracy: 0.7929
Epoch 27/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4295 - accuracy: 0.7878
Epoch 28/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4224 - accuracy: 0.7947
Epoch 29/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4222 - accuracy: 0.7901
Epoch 30/100
155/155 [==============================] - 1s 4ms/step - loss: 0.4221 - accuracy: 0.7945
Epoch 31/100
155/155 [==============================] - 1s 5ms/step - loss: 0.4237 - accuracy: 0.7890
Epoch 32/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4221 - accuracy: 0.7955
Epoch 33/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4291 - accuracy: 0.7905
Epoch 34/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4218 - accuracy: 0.7917
Epoch 35/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4224 - accuracy: 0.7909
Epoch 36/100
155/155 [==============================] - 1s 4ms/step - loss: 0.4217 - accuracy: 0.7949
Epoch 37/100
155/155 [==============================] - 1s 5ms/step - loss: 0.4243 - accuracy: 0.7961
Epoch 38/100
155/155 [==============================] - 1s 3ms/step - loss: 0.4240 - accuracy: 0.7937
Epoch 39/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4218 - accuracy: 0.7921
Epoch 40/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4203 - accuracy: 0.7945
Epoch 41/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4205 - accuracy: 0.7935
Epoch 42/100
155/155 [==============================] - 1s 5ms/step - loss: 0.4222 - accuracy: 0.7915
Epoch 43/100
155/155 [==============================] - 1s 5ms/step - loss: 0.4272 - accuracy: 0.7925
Epoch 44/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4227 - accuracy: 0.7897
Epoch 45/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4205 - accuracy: 0.7939
```

```
Epoch 46/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4242 - accuracy: 0.7917
Epoch 47/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4223 - accuracy: 0.7951
Epoch 48/100
155/155 [==============================] - 1s 5ms/step - loss: 0.4219 - accuracy: 0.7976
Epoch 49/100
155/155 [==============================] - 1s 5ms/step - loss: 0.4240 - accuracy: 0.7949
Epoch 50/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4236 - accuracy: 0.7901
Epoch 51/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4216 - accuracy: 0.7941
Epoch 52/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4211 - accuracy: 0.7951
Epoch 53/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4206 - accuracy: 0.7955
Epoch 54/100
155/155 [==============================] - 1s 5ms/step - loss: 0.4228 - accuracy: 0.7927
Epoch 55/100
155/155 [==============================] - 1s 4ms/step - loss: 0.4206 - accuracy: 0.7959
Epoch 56/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4249 - accuracy: 0.7921
Epoch 57/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4213 - accuracy: 0.7929
Epoch 58/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4233 - accuracy: 0.7903
Epoch 59/100
155/155 [==============================] - 1s 4ms/step - loss: 0.4214 - accuracy: 0.7939
Epoch 60/100
155/155 [==============================] - 1s 5ms/step - loss: 0.4208 - accuracy: 0.7929
Epoch 61/100
155/155 [==============================] - 1s 4ms/step - loss: 0.4195 - accuracy: 0.7970
Epoch 62/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4225 - accuracy: 0.7933
Epoch 63/100
155/155 [==============================] - 0s 2ms/step - loss: 0.4207 - accuracy: 0.7919
Epoch 64/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4203 - accuracy: 0.7911
Epoch 65/100
155/155 [==============================] - 1s 4ms/step - loss: 0.4205 - accuracy: 0.7963
Epoch 66/100
155/155 [==============================] - 1s 6ms/step - loss: 0.4225 - accuracy: 0.7917
Epoch 67/100
155/155 [==============================] - 1s 4ms/step - loss: 0.4227 - accuracy: 0.7897
Epoch 68/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4208 - accuracy: 0.7949
Epoch 69/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4200 - accuracy: 0.7917
Epoch 70/100
155/155 [==============================] - 1s 5ms/step - loss: 0.4223 - accuracy: 0.7935
Epoch 71/100
155/155 [==============================] - 1s 6ms/step - loss: 0.4216 - accuracy: 0.7937
Epoch 72/100
155/155 [==============================] - 1s 4ms/step - loss: 0.4212 - accuracy: 0.7880
Epoch 73/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4228 - accuracy: 0.7909
Epoch 74/100
155/155 [==============================] - 1s 3ms/step - loss: 0.4194 - accuracy: 0.7927
Epoch 75/100
155/155 [==============================] - 1s 7ms/step - loss: 0.4254 - accuracy: 0.7939
Epoch 76/100
155/155 [==============================] - 1s 6ms/step - loss: 0.4180 - accuracy: 0.7966
Epoch 77/100
155/155 [==============================] - 1s 6ms/step - loss: 0.4217 - accuracy: 0.7933
Epoch 78/100
155/155 [==============================] - 1s 5ms/step - loss: 0.4214 - accuracy: 0.7921
Epoch 79/100
155/155 [==============================] - 1s 6ms/step - loss: 0.4224 - accuracy: 0.7923
Epoch 80/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4224 - accuracy: 0.7905
Epoch 81/100
155/155 [==============================] - 1s 3ms/step - loss: 0.4214 - accuracy: 0.7917
Epoch 82/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4205 - accuracy: 0.7923
Epoch 83/100
155/155 [==============================] - 1s 5ms/step - loss: 0.4205 - accuracy: 0.7961
Epoch 84/100
155/155 [==============================] - 1s 6ms/step - loss: 0.4204 - accuracy: 0.7933
Epoch 85/100
155/155 [==============================] - 1s 5ms/step - loss: 0.4227 - accuracy: 0.7927
Epoch 86/100
155/155 [==============================] - 1s 7ms/step - loss: 0.4226 - accuracy: 0.7970
Epoch 87/100
155/155 [==============================] - 1s 4ms/step - loss: 0.4187 - accuracy: 0.7917
Epoch 88/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4208 - accuracy: 0.7915
Epoch 89/100
155/155 [==============================] - 1s 4ms/step - loss: 0.4194 - accuracy: 0.7929
Epoch 90/100
```

```
155/155 [==============================] - 1s 6ms/step - loss: 0.4234 - accuracy: 0.7921
Epoch 91/100
155/155 [==============================] - 1s 6ms/step - loss: 0.4210 - accuracy: 0.7939
Epoch 92/100
155/155 [==============================] - 1s 5ms/step - loss: 0.4206 - accuracy: 0.7903
Epoch 93/100
155/155 [==============================] - 1s 3ms/step - loss: 0.4191 - accuracy: 0.7959
Epoch 94/100
155/155 [==============================] - 1s 8ms/step - loss: 0.4199 - accuracy: 0.7917
Epoch 95/100
155/155 [==============================] - 1s 6ms/step - loss: 0.4193 - accuracy: 0.7953
Epoch 96/100
155/155 [==============================] - 1s 7ms/step - loss: 0.4197 - accuracy: 0.7980
Epoch 97/100
155/155 [==============================] - 1s 6ms/step - loss: 0.4205 - accuracy: 0.7984
Epoch 98/100
155/155 [==============================] - 1s 4ms/step - loss: 0.4192 - accuracy: 0.7945
Epoch 99/100
155/155 [==============================] - 0s 3ms/step - loss: 0.4216 - accuracy: 0.7935
Epoch 100/100
155/155 [==============================] - 1s 3ms/step - loss: 0.4213 - accuracy: 0.7890
```

Out[36]:

```
<keras.callbacks.History at 0x18ff59b73d0>
```

In Epoch 87/100, we have got the accuracy 0.8014, which means the model is correctly classifying approximately 80.14% of the training data.

In [37]:

```python
yp = model.predict(X)
yp[:10]
```

```
155/155 [==============================] - 1s 3ms/step
```

Out[37]:

```
array([[0.44607124],
       [0.60677737],
       [0.00883169],
       [0.01145998],
       [0.35141975],
       [0.31024975],
       [0.07969682],
       [0.45320144],
       [0.69430137],
       [0.01434091]], dtype=float32)
```

In [38]:

```python
y[:5]
```

Out[38]:

```
array([1., 0., 1., 0., 0.], dtype=float32)
```

In [39]:

```python
y_pred = []
for element in yp:
    if element > 0.5:
        y_pred.append(1)
    else:
        y_pred.append(0)
```

In [40]:

```python
y_pred[:10]
```

Out[40]:

```
[0, 1, 0, 0, 0, 0, 0, 0, 1, 0]
```

In [41]:

```python
y_pred = model.predict(X)

y_pred_binary = (y_pred > 0.5).astype(int)
```

```
155/155 [==============================] - 1s 4ms/step
```

In [42]:

```python
class_report = classification_report(y, y_pred_binary, target_names=["Churn=No", "Churn=Yes"])
print(class_report)
```

```
              precision    recall  f1-score   support

    Churn=No       0.84      0.88      0.86      3614
   Churn=Yes       0.63      0.55      0.59      1316

    accuracy                           0.79      4930
   macro avg       0.74      0.72      0.73      4930
weighted avg       0.79      0.79      0.79      4930
```

Interpretation of the Result of Classification Report

Precision: Precision is a measure of how many of the positive predictions made by the model were correct. In this report, for "Churn=No," the precision is 0.84, meaning that 84% of the predictions for "Churn=No" were correct. For "Churn=Yes," the precision is 0.65, indicating that 65% of the predictions for "Churn=Yes" were correct.

Recall: Recall (also known as sensitivity or true positive rate) measures how many of the actual positive samples were correctly predicted by the model. In this report, for "Churn=No," the recall is 0.90, suggesting that the model correctly identified 90% of the actual "Churn=No" cases. For "Churn=Yes," the recall is 0.53, meaning that the model correctly identified 53% of the actual "Churn=Yes" cases.

F1-Score: The F1-score is the harmonic mean of precision and recall and provides a single metric that balances both values. In this report, for "Churn=No," the F1-score is 0.87, indicating a good balance between precision and recall. For "Churn=Yes," the F1-score is 0.58, which is lower, indicating that there might be a trade-off between precision and recall for "Churn=Yes."

Support: Support represents the number of actual occurrences of each class in the dataset. In this report, there are 3,614 instances of "Churn=No" and 1,316 instances of "Churn=Yes."

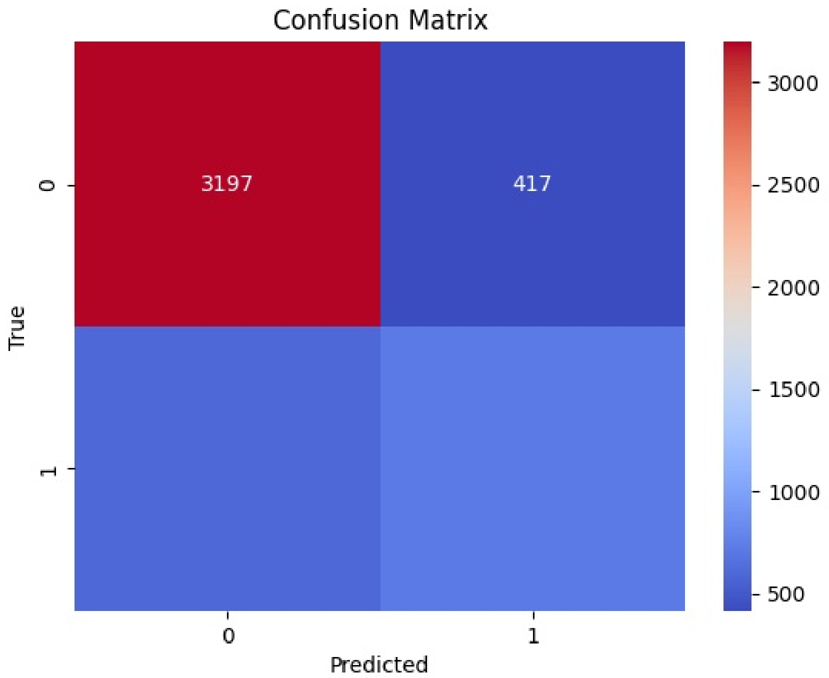
Accuracy: Accuracy is the overall correctness of the model's predictions. The report indicates that the overall accuracy of the model is 0.80, meaning that it correctly classified 80% of the total samples.

Macro Avg: The macro average is the average of the precision, recall, and F1-score for each class. In this case, the macro average precision is 0.75, the macro average recall is 0.71, and the macro average F1-score is 0.73.

Weighted Avg: The weighted average considers class imbalance in the dataset. It takes into account the number of samples for each class. The weighted average precision is 0.79, the weighted average recall is 0.80, and the weighted average F1-score is 0.79.

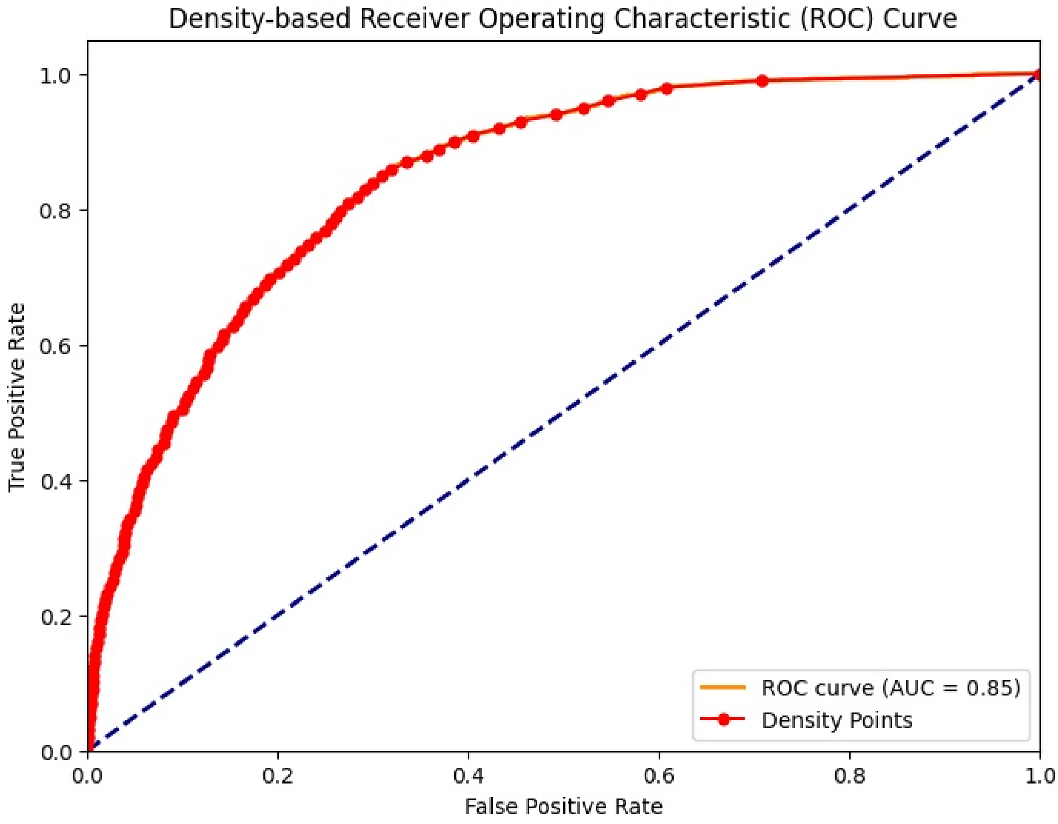
In summary, the model has a relatively high precision and recall for "Churn=No," indicating good performance in identifying non-churn cases. However, for "Churn=Yes," the model's precision and recall are lower, suggesting room for improvement in correctly identifying churn cases.

In [43]:
```
conf_matrix = confusion_matrix(y, y_pred_binary)
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="coolwarm")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```



In [44]:
```
y_pred_probs = model.predict(X)

fpr, tpr, thresholds = roc_curve(y, y_pred_probs)

# Calculate the AUC (Area Under the Curve)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

density_points = np.linspace(0, 1, 100)  # number of density points as needed
density_fpr = np.interp(density_points, tpr, fpr)
plt.plot(density_fpr, density_points, marker='o', markersize=5, color='red', label='Density Points')

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Density-based Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
155/155 [==============================] - 0s 2ms/step
```



Recommendations:

1.Customer Retention is Crucial: The dataset highlights the importance of customer retention for the telecom company. Retaining existing customers is more cost-effective than acquiring new ones. Strategies for reducing customer churn should be a top priority for the business.

2. Variable Impact: The logistic regression model reveals that some variables have a more significant impact on churn. OnlineSecurity, TechSupport, and Contract are factors that significantly influence customer churn. The company should focus on improving these services to retain customers.

3. Service Enhancement: The availability of OnlineSecurity and TechSupport services significantly affects customer decisions. The company can enhance these services to improve customer satisfaction and reduce churn.

4. Contract Length Matters: The type and duration of contracts also play a role in customer retention. Month-to-month contracts are associated with higher churn rates. Encouraging longer-term contracts can improve customer retention.

Based on these results, the company can consider the following actions:

Logistic Regression Model:

The company can use the logistic regression model to predict customer churn. Strategies for customer retention can be implemented based on the model's insights.

OLS Regression Model:

The OLS regression model highlights the importance of tenure and monthly charges in influencing customer churn. The company should focus on strategies to retain long-term customers and optimize pricing strategies.

In [ ]: