Business Understanding

SyriaTel, a telecommunications company, wants to reduce customer churn—that is, customers who leave the service ("stop doing business") after a short period. Losing customers directly impacts revenue, so understanding which customers are likely to leave soon is critical.

By analyzing customer behavior (such as call patterns, service usage, and plan type), the company can proactively target at-risk customers with promotions, improved service, or personalized offers to retain them and reduce financial losses.

Problem Statement

We aim to build a predictive model that can classify whether a customer is likely to churn or stay. This is a binary classification problem:

Target Variable: Churn (Yes / No)

Goal: Identify customers who are at high risk of leaving.

The model should provide:

Probability scores – how likely a customer is to churn.

Actionable insights – which features contribute most to churn risk.

Why this matters:

Allows SyriaTel to take preventive measures for at-risk customers.

Helps prioritize marketing efforts for maximum impact.

Reduces revenue loss from unexpected customer departures.

Importing the neccesary libraries

In [492…
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score, ro
from sklearn.tree import DecisionTreeClassifier
```

Loading the data

In [493…
```
df=pd.read_csv('./syriatelcurn.csv')
df.head()
```

Out[493…

	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 calls
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122

5 rows × 21 columns

Data Exploration

In [494…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account length          3333 non-null   int64  
 2   area code               3333 non-null   int64  
 3   phone number            3333 non-null   object 
 4   international plan      3333 non-null   object 
 5   voice mail plan         3333 non-null   object 
 6   number vmail messages   3333 non-null   int64  
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64  
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64  
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64  
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64  
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64  
 20  churn                   3333 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
```

In [495…

```
df.duplicated().sum()
```

Out[495…

```
0
```

In [496…

```
df.corr() ## checking for correlation
```

Out[496…

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	tota cl
account length	1.000000	-0.012463	-0.004628	0.006216	0.038470	0.006214	-0.006757	0.019260	-0.0
area code	-0.012463	1.000000	-0.001994	-0.008264	-0.009646	-0.008264	0.003580	-0.011886	0.0
number vmail messages	-0.004628	-0.001994	1.000000	0.000778	-0.009548	0.000776	0.017562	-0.005864	0.0
total day minutes	0.006216	-0.008264	0.000778	1.000000	0.006750	1.000000	0.007043	0.015769	0.0
total day calls	0.038470	-0.009646	-0.009548	0.006750	1.000000	0.006753	-0.021451	0.006462	-0.0
total day charge	0.006214	-0.008264	0.000776	1.000000	0.006753	1.000000	0.007050	0.015769	0.0
total eve minutes	-0.006757	0.003580	0.017562	0.007043	-0.021451	0.007050	1.000000	-0.011430	1.0
total eve calls	0.019260	-0.011886	-0.005864	0.015769	0.006462	0.015769	-0.011430	1.000000	-0.0
total eve charge	-0.006745	0.003607	0.017578	0.007029	-0.021449	0.007036	1.000000	-0.011423	1.0
total night minutes	-0.008955	-0.005825	0.007681	0.004323	0.022938	0.004324	-0.012584	-0.002093	-0.0
total night calls	-0.013176	0.016522	0.007123	0.022972	-0.019557	0.022972	0.007586	0.007710	0.0
total night charge	-0.008960	-0.005845	0.007663	0.004300	0.022927	0.004301	-0.012593	-0.002056	-0.0
total intl minutes	0.009514	-0.018288	0.002856	-0.010155	0.021565	-0.010157	-0.011035	0.008703	-0.0
total intl calls	0.020661	-0.024179	0.013957	0.008033	0.004574	0.008032	0.002541	0.017434	0.0
total intl charge	0.009546	-0.018395	0.002884	-0.010092	0.021666	-0.010094	-0.011067	0.008674	-0.0
customer service calls	-0.003796	0.027572	-0.013263	-0.013423	-0.018942	-0.013427	-0.012985	0.002423	-0.0
churn	0.016541	0.006174	-0.089728	0.205151	0.018459	0.205151	0.092796	0.009233	0.0

In [497…

```
corr_value = df["total day minutes"].corr(df["total day charge"]) # checking how the
print(f"Correlation between total day minutes and total day charge: {corr_value:.4f}
```

```
Correlation between total day minutes and total day charge: 1.0000
```

In [498…

```
df.columns
```

Out[498…

```
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',
       'customer service calls', 'churn'],
      dtype='object')
```

Dropping colums that will not need for the analysis

In [499…

```
columns_to_drop = [
    "phone number",
    "state",
    "area code",
    "total day charge",
    "total eve charge",
    "total night charge",
    "total intl charge"
]

df = df.drop(columns=columns_to_drop)
```

In [500…

```
df["international plan"] = df["international plan"].map({"yes":1, "no":0}) # yes for
df["voice mail plan"] = df["voice mail plan"].map({"yes":1, "no":0}) # # yes for 1 m
```

In [501…

```
df['churn'] = df['churn'].astype(int)  # we did type cast from boolean to integer wh
```

In [502…

```
df['churn'].value_counts() ## we note the classes are highly imbalanced
```

Out[502…

```
0    2850
1     483
Name: churn, dtype: int64
```

Feature- target selection

In [503…

```
X=df.drop('churn', axis= 1)
y=df['churn']
```

In [504…

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42,str
```

In [505…

```
X_test.info() # checking if the splitting was done correctly
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 667 entries, 601 to 1962
Data columns (total 13 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   account length          667 non-null    int64  
 1   international plan      667 non-null    int64  
 2   voice mail plan         667 non-null    int64  
 3   number vmail messages   667 non-null    int64  
 4   total day minutes       667 non-null    float64
 5   total day calls         667 non-null    int64  
 6   total eve minutes       667 non-null    float64
 7   total eve calls         667 non-null    int64  
 8   total night minutes     667 non-null    float64
 9   total night calls       667 non-null    int64  
 10  total intl minutes      667 non-null    float64
 11  total intl calls        667 non-null    int64  
 12  customer service calls  667 non-null    int64  
dtypes: float64(4), int64(9)
memory usage: 73.0 KB
```

In [506…

```
X.describe()  # to check the ranges in the X feature to see in terms of max to mean
```

Out[506…

	account length	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total eve minutes
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000
mean	101.064806	0.096910	0.276628	8.099010	179.775098	100.435644	200.980348
std	39.822106	0.295879	0.447398	13.688365	54.467389	20.069084	50.713844
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	74.000000	0.000000	0.000000	0.000000	143.700000	87.000000	166.600000
50%	101.000000	0.000000	0.000000	0.000000	179.400000	101.000000	201.400000
75%	127.000000	0.000000	1.000000	20.000000	216.400000	114.000000	235.300000
max	243.000000	1.000000	1.000000	51.000000	350.800000	165.000000	363.700000

Standadize the data

In [507…

```
# Initialize scaler
scaler = StandardScaler()

# Fit only on training data
X_train_scaled = scaler.fit_transform(X_train)

# Transform test data using same scaler
X_test_scaled = scaler.transform(X_test)
```

Fit the logistic regression model

In [508…

```
logmodel = LogisticRegression(class_weight="balanced", random_state=42)# class weigh
logmodel.fit(X_train_scaled, y_train)
```

Out[508…

```
LogisticRegression(class_weight='balanced', random_state=42)
```

Make predictions

In [509…

```
y_pred = logmodel.predict(X_test_scaled)
y_prob = logmodel.predict_proba(X_test_scaled)[:,1]  # probability for class 1 or th
```

Model evaluation

In [510…

```
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print("ROC AUC:", roc_auc_score(y_test, y_prob))
```

```
Accuracy: 0.7586206896551724
              precision    recall  f1-score   support

           0       0.95      0.76      0.84       570
           1       0.35      0.74      0.47        97

    accuracy                           0.76       667
   macro avg       0.65      0.75      0.66       667
weighted avg       0.86      0.76      0.79       667

ROC AUC: 0.8151021884608429
```

Interpretation

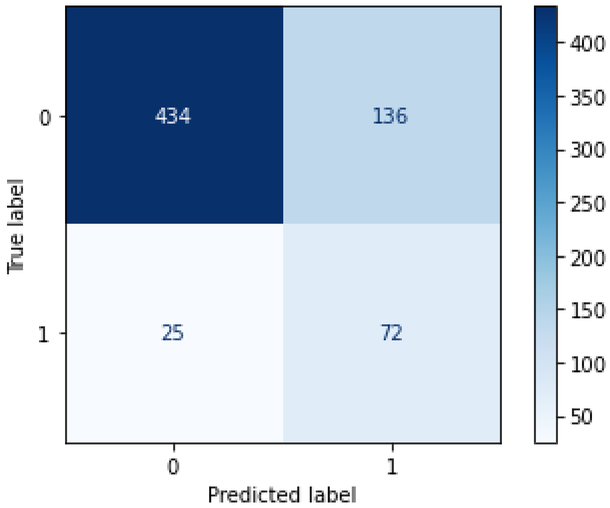
- Is that based on recall it was able to catch 74% of the customers who churned correctly
- Accuracy not very good measure for model performance as data was imbalance so went on to other performance metric such as recall,precision and F1 score
- Our goal was to catch how many left and on recall we got Around 74% of the people who churned were predicted correctly
- F1 score is low for churn as the precision for churn is also low,many false positives,flagged many as churn yet they were not churn,wasting resources trying to retain people who dont plan on churning.
- here missing a false negative is more costly to the company as it will classify a churner as a no churner,so recall is more important

Plotting

In [511…

```
# y_test → actual labels
# y_pred → predicted labels from your model

cm = confusion_matrix(y_test, y_pred)
display = ConfusionMatrixDisplay(confusion_matrix=cm)
display.plot(cmap=plt.cm.Blues)  # optional color
plt.show()
```



In [512…

```
TP = 72 # the manual way to calculate the recall and precision just for good measure
FP = 136
TN = 434
FN = 25


Precision = TP / (TP + FP)
Recall = TP / (TP + FN)
print(f"Precision: {Precision:.3f}")
print(f"Recall: {Recall:.3f}")
```

```
Precision: 0.346
Recall: 0.742
```

In [513…

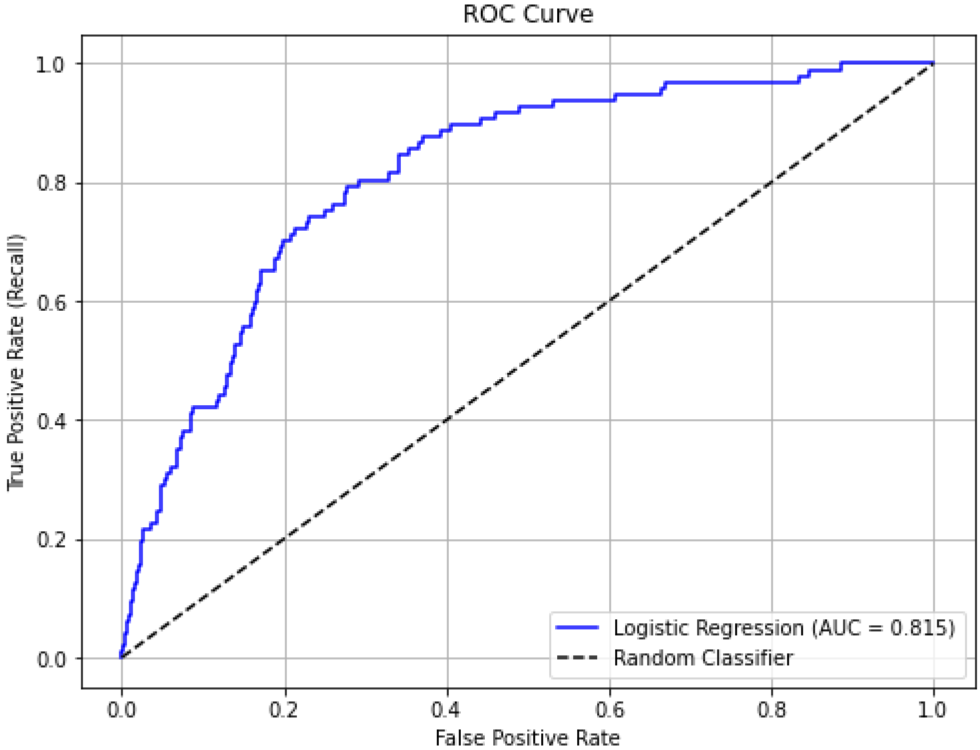
```
roc_auc = roc_auc_score(y_test, y_prob)
print("ROC AUC Score:", roc_auc)


fpr, tpr, thresholds = roc_curve(y_test, y_prob)

plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f'Logistic Regression (AUC = {roc_auc:.3f})', color='blue')
```

```
plt.plot([0,1], [0,1], 'k--', label='Random Classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate (Recall)')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

```
ROC AUC Score: 0.8151021884608429
```



Interpretation

0.815 AUC → the model is good at identifying churners vs non-churners

High recall for churners (0.74) + good AUC (0.815) → your model can effectively find customers at risk of leaving

Decision Tree

Intiating of the tree and fitting the model

In [514...

```
dt_model = DecisionTreeClassifier(
    max_depth=5,
    min_samples_split=20,
    class_weight="balanced",
    random_state=42
)  # here we are still balancing the data as the data for target has class heavily i

dt_model.fit(X_train, y_train)
```

Out[514...

```
DecisionTreeClassifier(class_weight='balanced', max_depth=5,
                       min_samples_split=20, random_state=42)
```

Predicting

In [515...

```
y_pred_dt = dt_model.predict(X_test) # predicting model
y_prob_dt = dt_model.predict_proba(X_test)[:,1] # predicting probability of selectin
```

Evaluating the Decision Tree method

In [516…

```
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print(classification_report(y_test, y_pred_dt))
print("ROC AUC:", roc_auc_score(y_test, y_prob_dt))
```

```
Accuracy: 0.904047976011994
              precision    recall  f1-score   support

           0       0.95      0.93      0.94       570
           1       0.65      0.73      0.69        97

    accuracy                           0.90       667
   macro avg       0.80      0.83      0.82       667
weighted avg       0.91      0.90      0.91       667

ROC AUC: 0.8132302405498282
```

Interpretation

Precision = 0.65 Out of all customers the model predicts will churn, 65% actually do churn. This means fewer false alarms.

Recall = 0.73 means The model correctly identifies 73% of all actual churners, so most churners are caught only 27% of churners are missed.

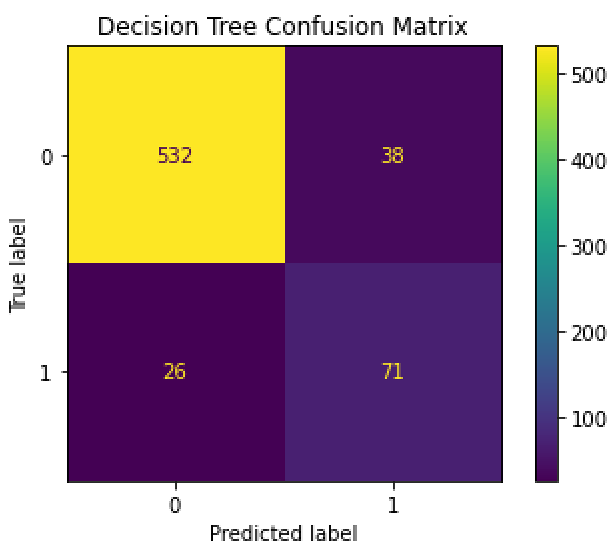
F1-score = 0.69 means A balance between precision and recall, showing the model is doing reasonably well at both catching churners and avoiding false positives.

ROC AUC = 0.813 meaning The model can discriminate between churners and non-churners fairly well.

Display the confusion matrix

In [517…

```
cm = confusion_matrix(y_test, y_pred_dt)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title("Decision Tree Confusion Matrix")
plt.show()
```



Compare both models using the AUC ROC CURVE

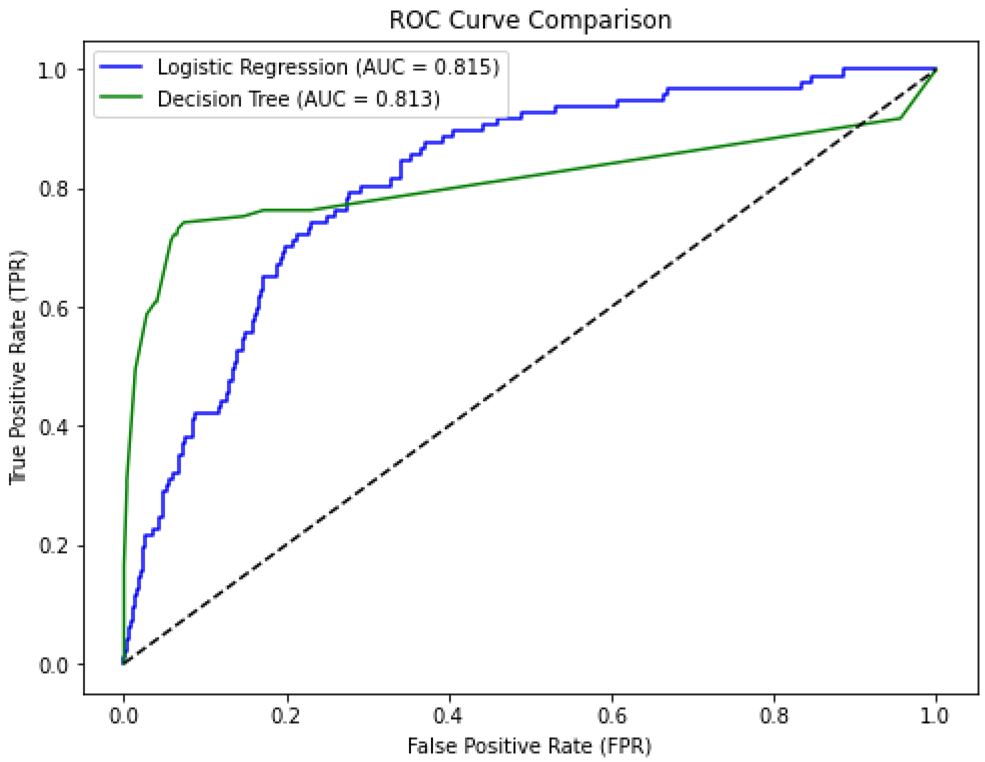
In [518…

```
# Probabilities for class 1
y_prob_log = logmodel.predict_proba(X_test_scaled)[:, 1]
y_prob_dt = dt_model.predict_proba(X_test)[:, 1]
```

```
# ROC curve for Logistic Regression
fpr_log, tpr_log, _ = roc_curve(y_test, y_prob_log)
auc_log = roc_auc_score(y_test, y_prob_log)

# ROC curve for Decision Tree
fpr_dt, tpr_dt, _ = roc_curve(y_test, y_prob_dt)
auc_dt = roc_auc_score(y_test, y_prob_dt)

# Plotting both curves
plt.figure(figsize=(8,6))
plt.plot(fpr_log, tpr_log, label=f'Logistic Regression (AUC = {auc_log:.3f})', color
plt.plot(fpr_dt, tpr_dt, label=f'Decision Tree (AUC = {auc_dt:.3f})', color='green')
plt.plot([0,1], [0,1], 'k--')  # random classifier line
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve Comparison')
plt.legend()
plt.show()
```



Curve interpretation

AUC values:

Logistic Regression is 0.815

Decision Tree is 0.813

Both models are very similar in discriminative ability.

Decision Tree may slightly improve precision for churners, but Logistic Regression has a slightly smoother ROC curve.

AUC above 0.8 for both models means both are reasonably good at distinguishing churners from non-churners.

Overall interpretation of Logistic regression Vs Decision Tree

Accuracy: 0.90 → 90% of all predictions correct. Slightly better overall than logistic regression.

Class 1 metrics (churners):

Precision = 0.65 → Now 65% of predicted churners actually churned. Much better than logistic regression which was around 35% .

Recall = 0.73 → Still 73% of actual churners are caught. Comparable to logistic regression.

F1-score = 0.69 → Higher than logistic regression which was around 47% , balancing recall and precision.

ROC AUC = 0.813 → Similar ability to distinguish churners vs non-churners,here both models had about the same ability to distinguish.

In relation to the business:

Decision tree gives fewer false alarms (higher precision), while still catching most churners (good recall).

More trustworthy if you want to target only the most likely churners and avoid unnecessary marketing costs.