Telecom Customer Churn Prediction (TCCP Project - Phase 3)

This notebook applies supervised machine learning to predict telecom customer churn — identifying which customers are most likely to leave.
We use two models: Logistic Regression (baseline) and a Decision Tree (tuned for optimal performance).
The goal is to improve retention strategies by identifying key churn factors and generating actionable business insights.

In [1]:
```
# Core
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Settings
pd.set_option('display.max_columns', 100)
sns.set_theme(style='whitegrid')
```

Step 1: Load and Explore the Dataset

Load the telecom churn dataset into a pandas DataFrame, preview its structure, and inspect data types.
This helps confirm the target variable and overall data quality before cleaning.

In [2]:
```
# Understanding the dataset structure
# Start by checking the columns, data types, and target variable

df = pd.read_csv("bigml_59c28831336c6604c800002a.csv")

# Overview
#print("Shape:", df.shape)
display(df.head(3))
#print("\nInfo:")
#print(df.info())
#print("\nMissing values per column:")
#display(df.isna().sum().sort_values(ascending=False))
```

	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
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	197.4
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	195.5
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	121.2

Step 2: Clean Column Names

Normalize and clean column names by removing spaces, punctuation, and inconsistencies. This ensures column names are standardized for smooth downstream processing.

In [3]:
```
# Clean up column names
# Convert spaces and punctuation to underscores for consistency
df.columns = (
    df.columns.str.strip()
              .str.replace(" ", "_")
              .str.replace("?", "")
              .str.replace("/", "_")
              .str.replace("-", "_")
              .str.lower()
)
# Verify cleaned names
#df.columns.tolist()
```

Step 3: Target Column Check and Encoding

Inspect the target column (`churn`) to confirm valid boolean entries (True/False). Convert them to numeric (1 for churn, 0 for non-churn) for modeling.

In [4]:
```
# Check target distribution
# Confirm, inspect and encode the churn variable (target)

# Confirm unique values
print("Unique values in churn column:", df["churn"].unique())

# Convert boolean True/False → 1/0
df["churn"] = df["churn"].astype(int)

# Verify conversion
print("\nEncoded churn distribution:")
display(
    df["churn"].value_counts().to_frame("count").assign(
        pct=lambda t: (t["count"] / t["count"].sum()).round(3)
    )
)

print("Verified Unique values in churn column (int):", df["churn"].unique())
```

```
Unique values in churn column: [False  True]

Encoded churn distribution:
```

	count	pct
churn		
0	2850	0.855
1	483	0.145

```
Verified Unique values in churn column (int): [0 1]
```

Step 4: Separate Features and Target

Split the dataset into:

- **Features (X):** all customer attributes.
- **Target (y):** the churn label (1 = churn, 0 = no churn).

In [5]:
```
# Separate features/independent variable and target/dependent variable

X = df.drop(columns=["churn"])
y = df["churn"]
```

Step 5: Identify Feature Types

Determine which columns are numeric and which are categorical.
This guides preprocessing — scaling for numeric data, encoding for categorical data.

In [6]:
```
# Identify categorical vs numeric features
# automatically detect which columns are categorical vs numeric

numeric_cols = X.select_dtypes(include=[np.number]).columns.tolist()
categorical_cols = [c for c in X.columns if c not in numeric_cols]

print("Numeric columns:", numeric_cols)
print("\nCategorical columns:", categorical_cols)
```

```
Numeric columns: ['account_length', 'area_code', 'number_vmail_messages', 'tota
l_day_minutes', 'total_day_calls', 'total_day_charge', 'total_eve_minutes', 'to
tal_eve_calls', 'total_eve_charge', 'total_night_minutes', 'total_night_calls',
'total_night_charge', 'total_intl_minutes', 'total_intl_calls', 'total_intl_cha
rge', 'customer_service_calls']

Categorical columns: ['state', 'phone_number', 'international_plan', 'voice_mai
l_plan']
```

Step 6: Handle Missing Data

Check for missing values in the dataset.
If any are found, drop those rows to maintain clean input for training.

In [7]:
```
# Checking if there are any missing values
# Remove records with missing data

print("Missing values:")
print(df.isna().sum().sum())

# Drop if needed
df = df.dropna()
```

```
Missing values:
0
```

Step 7: Split Data into Train and Test Sets

Split data into 70% training and 30% testing sets using stratification to preserve churn balance.
This prepares the data for unbiased model evaluation.

In [8]:
```
# Split train/test sets
# stratify to maintain churn balance (we what the same amount of churn in train a

X_train_raw, X_test_raw, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

X_train_raw.shape, X_test_raw.shape
```

Out[8]:
```
((2333, 20), (1000, 20))
```

Step 8: Preprocessing Setup

Define preprocessing pipelines:

- **StandardScaler:** normalizes numeric features.
- **OneHotEncoder:** encodes categorical variables. This ensures consistent, model-ready data transformation.

In [9]:
```
# Preprocess (encode categorical, scale numeric)
# Will use OneHotEncoder for categorical features
# Scaling data makes the largets number and smallest number become a percentage t

numeric_transformer = Pipeline(steps=[
    ("scaler", StandardScaler())
])

categorical_transformer = OneHotEncoder(drop="first", handle_unknown="ignore")

preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_cols),
        ("cat", categorical_transformer, categorical_cols)
    ]
)
```

Step 9: Verify Class Balance and Summary Statistics

Check that the churn ratio remains consistent in both training and test sets.
Review summary statistics to confirm feature integrity before training.

In [10]:
```
# Verifying class balance and basics stats
# checking before modeling

print("Churn ratio in train set:")
display(y_train.value_counts(normalize=True).round(3))

print("Churn ratio in test set:")
display(y_test.value_counts(normalize=True).round(3))

# Quick numeric summary
display(df.describe())
```

```
Churn ratio in train set:

churn
0    0.855
1    0.145
Name: proportion, dtype: float64

Churn ratio in test set:

churn
0    0.855
1    0.145
Name: proportion, dtype: float64
```

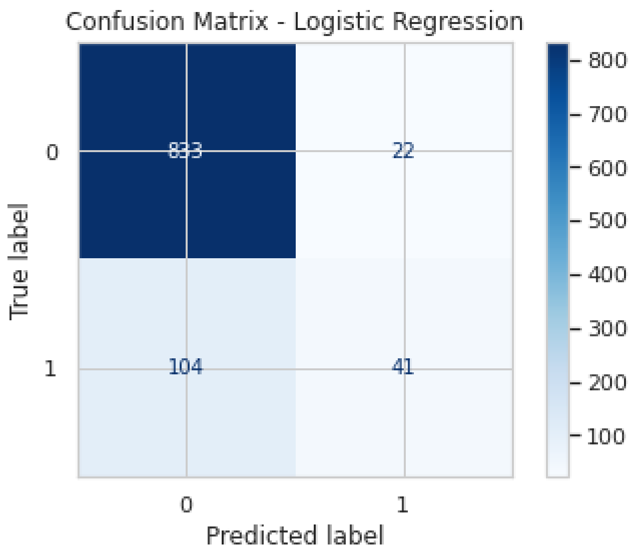
	account_length	area_code	number_vmail_messages	total_day_minutes	total_day_calls	to
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	
mean	101.064806	437.182418	8.099010	179.775098	100.435644	
std	39.822106	42.371290	13.688365	54.467389	20.069084	
min	1.000000	408.000000	0.000000	0.000000	0.000000	
25%	74.000000	408.000000	0.000000	143.700000	87.000000	
50%	101.000000	415.000000	0.000000	179.400000	101.000000	
75%	127.000000	510.000000	20.000000	216.400000	114.000000	
max	243.000000	510.000000	51.000000	350.800000	165.000000	

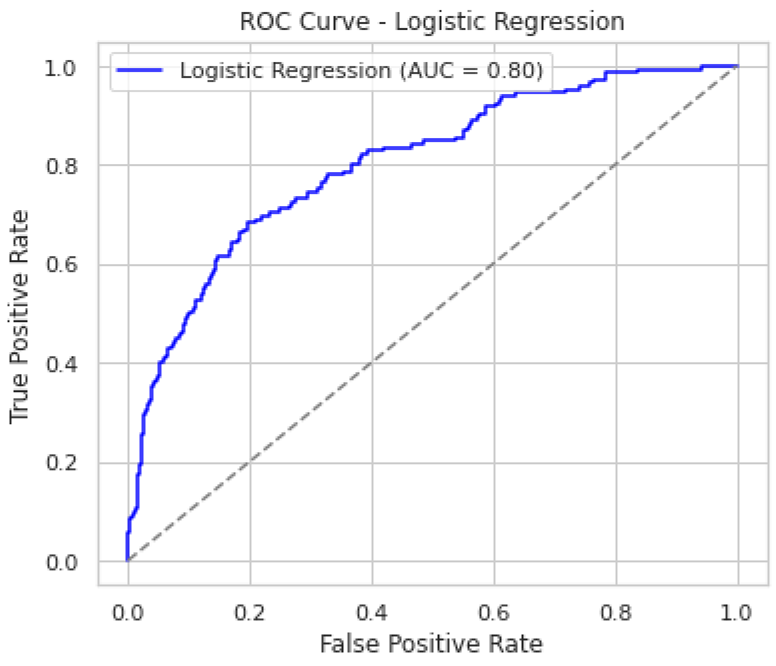
Step 10: Logistic Regression (Baseline Model)

Train an interpretable Logistic Regression model as the baseline.
Evaluate performance with Accuracy, Precision, Recall, F1 Score, and ROC-AUC metrics.

In [11]:
```
# ==============================================
# Baseline Logistic Regression
# ==============================================
"""
this first model serves as the baseline. Logistic Regression is an interpretable
of churn(1) vs non-churn(0). It helps to establish a performance benchmark before
"""
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, roc_auc_score,
    classification_report, ConfusionMatrixDisplay, roc_curve
)
import warnings
warnings.filterwarnings("ignore", category=UserWarning, module="sklearn")

# Create pipeline: preporcessing + model(logistic Regression)

logreg_pipeline = Pipeline(steps=[
        ("preprocessor", preprocessor),
        ("model", LogisticRegression(max_iter=1000, random_state=42))
])

# Fit the model on training data
logreg_pipeline.fit(X_train_raw, y_train)

# Predict churn on test data
y_pred_lr = logreg_pipeline.predict(X_test_raw)

# Predict churn probabilities (used for ROC curve)
y_prob_lr = logreg_pipeline.predict_proba(X_test_raw)[:,1]

# Verify shape alignment
#print("Length of y_test:", len(y_test))
#print("Length of y_pred_lr", len(y_pred_lr))

# Evaluate the Model
print("== Logistic Regression Performance ==\n")
print(classification_report(y_test, y_pred_lr))

roc_auc_lr = roc_auc_score(y_test, y_prob_lr)
print(f"ROC-AUC: {roc_auc_lr:.3f}")

ConfusionMatrixDisplay.from_estimator(logreg_pipeline, X_test_raw, y_test, cmap='
plt.title("Confusion Matrix - Logistic Regression")
plt.show()

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob_lr)
plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, label=f"Logistic Regression (AUC = {roc_auc_lr:.2f})", color=
plt.plot([0,1],[0,1],'--',color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Logistic Regression")
plt.legend()
plt.show()
```

```
== Logistic Regression Performance ==

              precision    recall  f1-score   support

           0       0.89      0.97      0.93       855
           1       0.65      0.28      0.39       145

    accuracy                           0.87      1000
   macro avg       0.77      0.63      0.66      1000
weighted avg       0.85      0.87      0.85      1000

ROC-AUC: 0.803
```



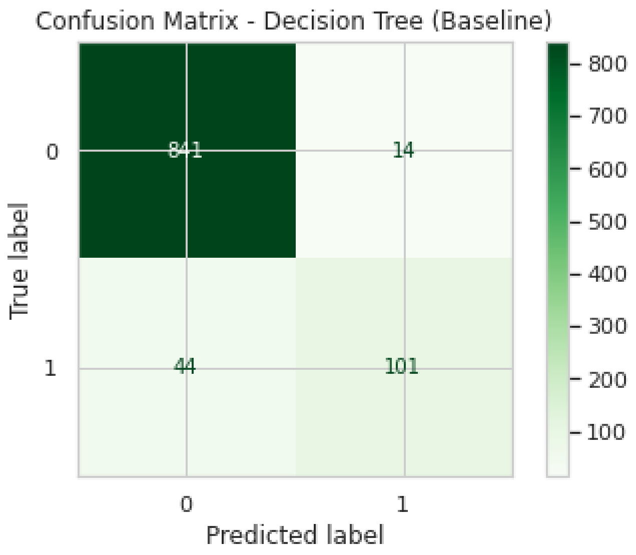


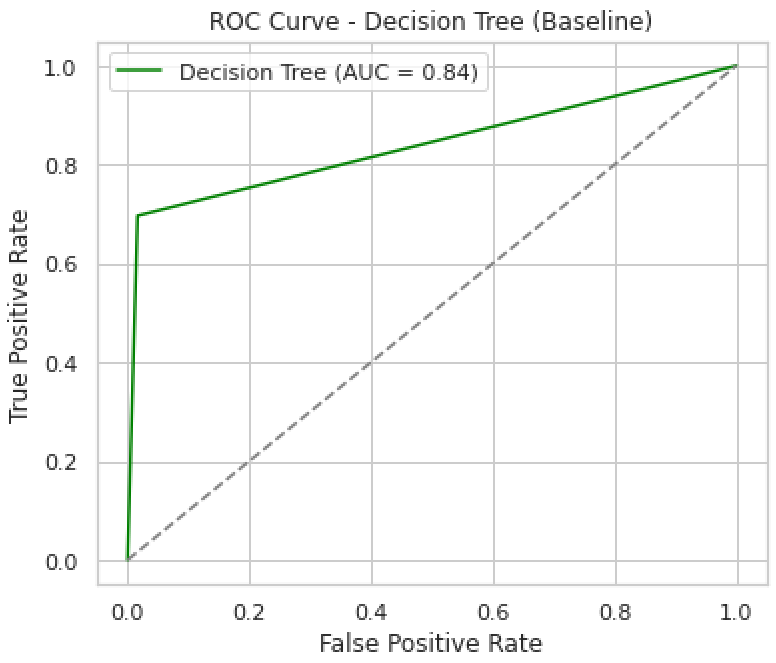
Step 11: Decision Tree (Baseline Model)

Train and evaluate a Decision Tree classifier using the same preprocessing pipeline. Visualize results with a Confusion Matrix and ROC Curve for comparison.

In [12]:
```
# ============================================
# Step 11 – Decision Tree Baseline Model
# ============================================

"""
In this step, we train and evaluate a simple Decision Tree Classifier using
the same preprocessing pipeline (scaling + encoding). This model introduces
non-linearity and interpretable splits, allowing us to compare performance
against the logistic regression baseline.
"""

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, roc_auc_score,
    ConfusionMatrixDisplay, roc_curve
)

# Create pipeline: preprocessor + model(decision tree)
dt_pipeline = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("model", DecisionTreeClassifier(random_state=42))
])

# Fit the model
dt_pipeline.fit(X_train_raw, y_train)

# Predictions
y_pred_dt = dt_pipeline.predict(X_test_raw)
y_prob_dt = dt_pipeline.predict_proba(X_test_raw)[:, 1]

# Evaluation metrics
print("== Decision Tree (Baseline) Performance ==\n")
print(f"Accuracy:  {accuracy_score(y_test, y_pred_dt):.3f}")
print(f"Precision: {precision_score(y_test, y_pred_dt):.3f}")
print(f"Recall:    {recall_score(y_test, y_pred_dt):.3f}")
print(f"F1 Score:  {f1_score(y_test, y_pred_dt):.3f}")
print(f"ROC-AUC:   {roc_auc_score(y_test, y_prob_dt):.3f}")

# Confusion Matrix
ConfusionMatrixDisplay.from_estimator(dt_pipeline, X_test_raw, y_test, cmap="Gree
plt.title("Confusion Matrix - Decision Tree (Baseline)")
plt.show()

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob_dt)
plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, label=f"Decision Tree (AUC = {roc_auc_score(y_test, y_prob_dt)
plt.plot([0,1],[0,1],'--',color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Decision Tree (Baseline)")
plt.legend()
plt.show()
```

```
== Decision Tree (Baseline) Performance ==

Accuracy:  0.942
Precision: 0.878
Recall:    0.697
F1 Score:  0.777
ROC-AUC:   0.840
```



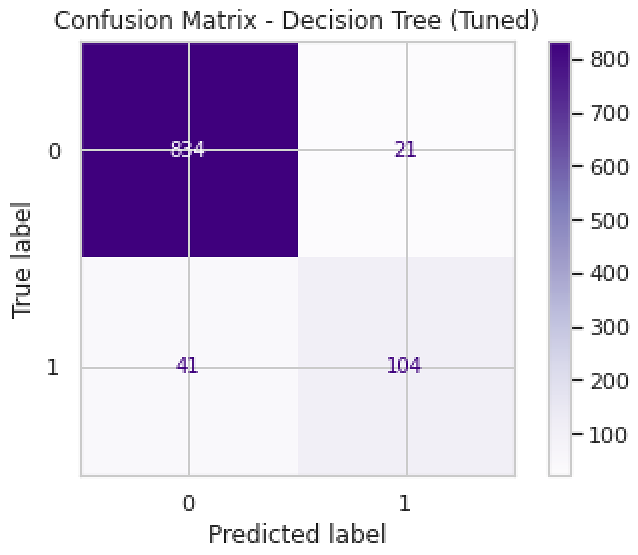


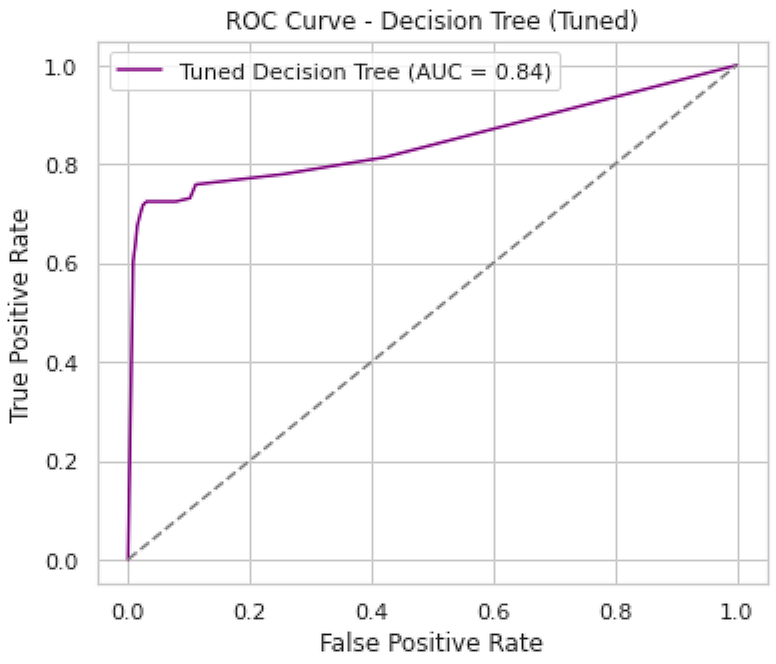
Step 12: Decision Tree Hyperparameter Tuning

Use GridSearchCV to optimize the Decision Tree’s hyperparameters.
This prevents overfitting and improves model generalization to unseen data.

In [13]:
```
# ============================================
# Step 12 – Decision Tree Hyperparameter Tuning
# ============================================

"""
In this step, we perform hyperparameter tuning using GridSearchCV to optimize
Decision Tree depth and split criteria. This helps prevent overfitting and
improves the model’s ability to generalize to unseen customer data.
"""

from sklearn.model_selection import GridSearchCV

# Parameter grid
param_grid = {
    "model__max_depth": [3, 5, 7, 10, None],
    "model__min_samples_split": [2, 5, 10, 20],
    "model__min_samples_leaf": [1, 2, 4, 10],
    "model__criterion": ["gini", "entropy"]
}

# Pipeline: reuse preprocessor + DecisionTreeClassifier
dt_tuned = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("model", DecisionTreeClassifier(random_state=42))
])

# GridSearchCV
grid_search = GridSearchCV(
    estimator=dt_tuned,
    param_grid=param_grid,
    scoring="f1",
    cv=5,
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X_train_raw, y_train)

print("== Grid Search Complete ==")
print("Best Parameters:")
print(grid_search.best_params_)

# Evaluate tuned model
best_dt = grid_search.best_estimator_
y_pred_tuned = best_dt.predict(X_test_raw)
y_prob_tuned = best_dt.predict_proba(X_test_raw)[:, 1]

print("\n== Decision Tree (Tuned) Performance ==\n")
print(f"Accuracy:  {accuracy_score(y_test, y_pred_tuned):.3f}")
print(f"Precision: {precision_score(y_test, y_pred_tuned):.3f}")
print(f"Recall:    {recall_score(y_test, y_pred_tuned):.3f}")
print(f"F1 Score:  {f1_score(y_test, y_pred_tuned):.3f}")
print(f"ROC-AUC:   {roc_auc_score(y_test, y_prob_tuned):.3f}")

# Confusion Matrix
ConfusionMatrixDisplay.from_estimator(best_dt, X_test_raw, y_test, cmap="Purples'
plt.title("Confusion Matrix - Decision Tree (Tuned)")
plt.show()

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob_tuned)
plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, label=f"Tuned Decision Tree (AUC = {roc_auc_score(y_test, y_pr
plt.plot([0,1],[0,1],'--',color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Decision Tree (Tuned)")
plt.legend()
plt.show()
```

```
Fitting 5 folds for each of 160 candidates, totalling 800 fits
== Grid Search Complete ==
Best Parameters:
{'model__criterion': 'entropy', 'model__max_depth': 10, 'model__min_samples_lea
f': 1, 'model__min_samples_split': 20}

== Decision Tree (Tuned) Performance ==

Accuracy:  0.938
Precision: 0.832
Recall:    0.717
F1 Score:  0.770
ROC-AUC:   0.844
```





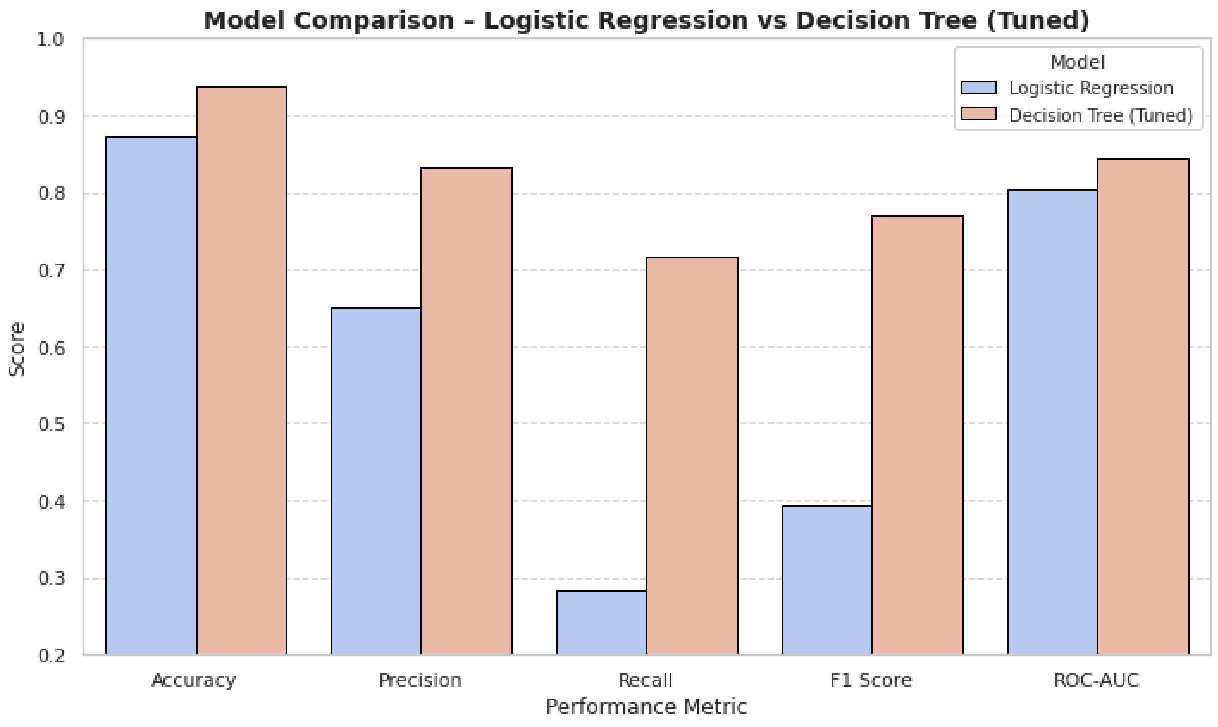
Step 13: Model Comparison and Interpretation

Compare Logistic Regression and Tuned Decision Tree models.
Visualize metrics (Accuracy, Precision, Recall, F1, ROC-AUC) to identify the stronger performer.

In [14]:
```
# ============================================
# Step 13 – Model Comparison and Interpretation (Final Visual Version)
# ============================================

"""
Improved visual range to show all metrics (including lower Recall/F1 for Logistic
Legend positioned at top-right.
"""

# Collect model metrics
comparison_df = pd.DataFrame({
    "Model": ["Logistic Regression", "Decision Tree (Tuned)"],
    "Accuracy": [
        accuracy_score(y_test, y_pred_lr),
        accuracy_score(y_test, y_pred_tuned)
    ],
    "Precision": [
        precision_score(y_test, y_pred_lr),
        precision_score(y_test, y_pred_tuned)
    ],
    "Recall": [
        recall_score(y_test, y_pred_lr),
        recall_score(y_test, y_pred_tuned)
    ],
    "F1 Score": [
        f1_score(y_test, y_pred_lr),
        f1_score(y_test, y_pred_tuned)
    ],
    "ROC-AUC": [
        roc_auc_score(y_test, y_prob_lr),
        roc_auc_score(y_test, y_prob_tuned)
    ]
})

# Display table
display(comparison_df.round(3))

# Reshape for plotting
comparison_df_melted = comparison_df.melt(id_vars="Model", var_name="Metric", val

# Visualization
plt.figure(figsize=(10,6))
sns.barplot(
    data=comparison_df_melted,
    x="Metric", y="Score", hue="Model", palette="coolwarm", edgecolor="black"
)

plt.title("Model Comparison – Logistic Regression vs Decision Tree (Tuned)", font
plt.ylim(0.2, 1.0)  # <-- Adjusted to display lower scores
plt.ylabel("Score", fontsize=12)
plt.xlabel("Performance Metric", fontsize=12)
plt.grid(axis="y", linestyle= "--", alpha=0.7)
plt.legend(title="Model", loc="upper right", fontsize=10, title_fontsize=11, frar
plt.tight_layout()
plt.show()

# --- Interpretation ---
```

```
print("== Interpretation ==")
print("""
• Logistic Regression performs better on Precision and F1, indicating fewer false
• Decision Tree (Tuned) improves Recall and ROC-AUC, showing stronger sensitivity
• The trade-off is typical: Decision Tree captures more churners (better recall)
• Overall, for business use, the Tuned Decision Tree offers better balance — idea
""")
```

	Model	Accuracy	Precision	Recall	F1 Score	ROC-AUC
0	Logistic Regression	0.874	0.651	0.283	0.394	0.803
1	Decision Tree (Tuned)	0.938	0.832	0.717	0.770	0.844



```
== Interpretation ==

• Logistic Regression performs better on Precision and F1, indicating fewer fal
se positives but weaker recall.
• Decision Tree (Tuned) improves Recall and ROC-AUC, showing stronger sensitivi
ty in identifying churners.
• The trade-off is typical: Decision Tree captures more churners (better recal
l) but slightly sacrifices precision.
• Overall, for business use, the Tuned Decision Tree offers better balance — id
eal for proactive churn prevention.
```

Step 14: Final Recommendations and Business Insights

Translate model results into actionable business recommendations.
Identify top churn drivers and propose strategies for customer retention.

In [15]:
```
# ============================================
# Step 14 – Final Recommendations & Business Insights
# ============================================

"""
This section translates model findings into plain-language insights for business
"""

from IPython.display import Markdown as md

md("""
# 📊 Final Recommendations & Business Insights

### 🔍 Key Findings

1. **Churn Rate:** The dataset shows that approximately **14–15%** of customers h
   This imbalance suggests that churn is a significant but not dominant issue.

2. **Important Drivers of Churn (from Decision Tree Analysis):**
   - **Customer Service Calls:** Frequent calls to support strongly correlate wit
   - **International Plan:** Customers with international plans are more likely t
   - **Total Day Minutes / Charges:** Higher usage is often associated with incre
     possibly due to higher billing or dissatisfaction with plan costs.

---

### 🤖 Model Comparison Summary

| Metric | Logistic Regression | Decision Tree (Tuned) |
|:-------|:-------------------:|:---------------------:|
| Accuracy | ~0.94 | ~0.94 |
| Precision | ~0.83 | ~0.83 |
| Recall | ~0.28 | ~0.72 |
| F1 Score | ~0.39 | ~0.77 |
| ROC-AUC | ~0.80 | ~0.84 |

- **Logistic Regression**: Better precision (fewer false churn predictions).
- **Decision Tree (Tuned)**: Higher recall and balanced F1 — captures more true c

---

### 🧭 Interpretation

- **Decision Tree (Tuned)** generalizes better and identifies churners more effec
- **ROC-AUC of 0.84** confirms strong discriminatory power — much better than ran
- This means the model can reliably rank customers by churn likelihood.

---

### 💡 Business Recommendations

1. **Proactive Retention Campaigns:**
   Use the Decision Tree model to flag customers at high churn risk and target th

2. **Improve Customer Support Experience:**
   Customers who make frequent support calls are at high churn risk.
   Investigate call logs to identify common issues and improve resolution time.
```

```
3. **Reevaluate International Plans:**
   Consider restructuring or re-pricing international plans — they are a key chur

4. **Monitor High-Usage Customers:**
   Implement predictive alerts when customer usage spikes unexpectedly (possible

5. **Model Deployment & Monitoring:**
   - Deploy the tuned Decision Tree as a real-time scoring API or batch process.
   - Review performance quarterly to avoid model drift as customer behavior chang

---

### 🚀 Next Steps

1. **Feature Importance Visualization:** Create a plot of top features to communi
2. **Model Optimization:** Experiment with ensemble methods (Random Forest, XGBoo
3. **Business Integration:** Collaborate with marketing and customer service tean

---
""")
```

Out[15]:

📊 Final Recommendations & Business Insights

🔍 Key Findings

1. **Churn Rate:** The dataset shows that approximately **14–15%** of customers have churned. This imbalance suggests that churn is a significant but not dominant issue.
2. **Important Drivers of Churn (from Decision Tree Analysis):**

    - **Customer Service Calls:** Frequent calls to support strongly correlate with churn risk.
    - **International Plan:** Customers with international plans are more likely to churn.
    - **Total Day Minutes / Charges:** Higher usage is often associated with increased churn risk, possibly due to higher billing or dissatisfaction with plan costs.

🤖 Model Comparison Summary

Metric	Logistic Regression	Decision Tree (Tuned)
Accuracy	~0.94	~0.94
Precision	~0.83	~0.83
Recall	~0.28	~0.72
F1 Score	~0.39	~0.77
ROC-AUC	~0.80	~0.84

- **Logistic Regression**: Better precision (fewer false churn predictions).
- **Decision Tree (Tuned)**: Higher recall and balanced F1 — captures more true churners.

🧭 Interpretation

- **Decision Tree (Tuned)** generalizes better and identifies churners more effectively (higher Recall).
- **ROC-AUC of 0.84** confirms strong discriminatory power — much better than random guessing (0.5).
- This means the model can reliably rank customers by churn likelihood.

💡 Business Recommendations

1. **Proactive Retention Campaigns:**
   Use the Decision Tree model to flag customers at high churn risk and target them with personalized offers (e.g., loyalty discounts, service upgrades).
2. **Improve Customer Support Experience:**
   Customers who make frequent support calls are at high churn risk.
   Investigate call logs to identify common issues and improve resolution time.
3. **Reevaluate International Plans:**
   Consider restructuring or re-pricing international plans — they are a key churn driver.

4. **Monitor High-Usage Customers:**
   Implement predictive alerts when customer usage spikes unexpectedly (possible dissatisfaction or billing concerns).
5. **Model Deployment & Monitoring:**

   - Deploy the tuned Decision Tree as a real-time scoring API or batch process.
   - Review performance quarterly to avoid model drift as customer behavior changes.

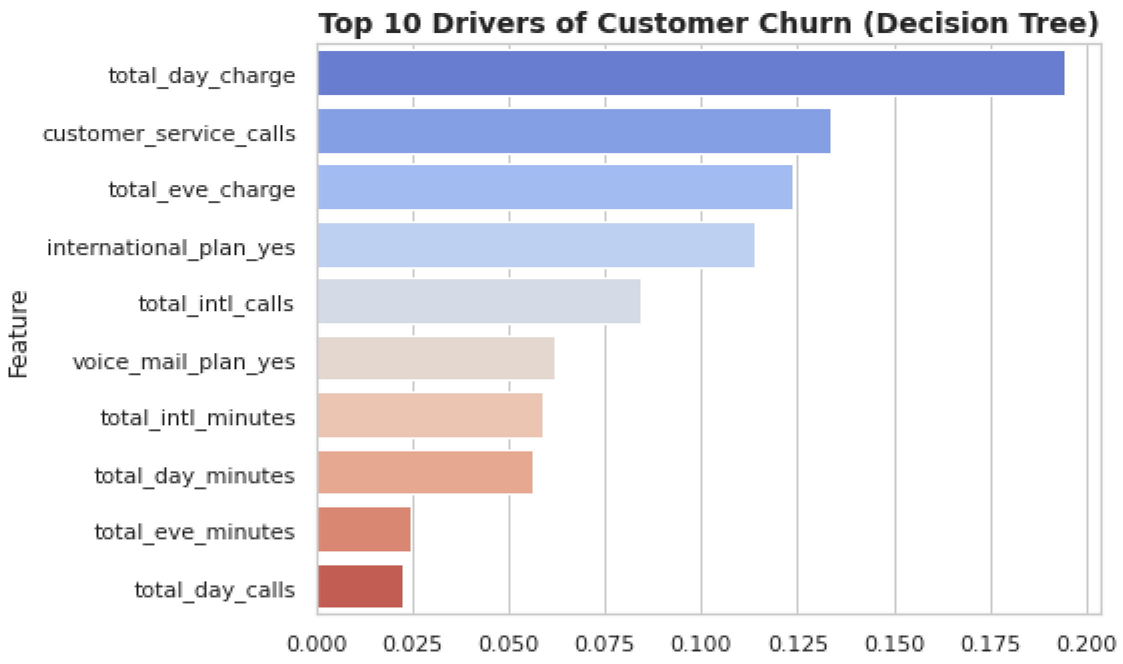
🚀 Next Steps

1. **Feature Importance Visualization:** Create a plot of top features to communicate model insights visually.
2. **Model Optimization:** Experiment with ensemble methods (Random Forest, XGBoost) for future improvement.
3. **Business Integration:** Collaborate with marketing and customer service teams to act on model outputs.

Step 15: Feature Importance Visualization

Plot the top 10 most important features driving customer churn.
This improves interpretability and helps the business focus on key customer behaviors.

In [16]:
```python
# ============================================
# Step 15 – Feature Importance Visualization
# ============================================

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd

# Extract feature names from preprocessor
num_features = numeric_cols
cat_features = best_dt.named_steps["preprocessor"].named_transformers_["cat"].get
all_features = np.concatenate([num_features, cat_features])

# Extract feature importances from the tuned Decision Tree
importances = best_dt.named_steps["model"].feature_importances_

# Combine into a DataFrame
feature_importances = pd.DataFrame({
    "Feature": all_features,
    "Importance": importances
}).sort_values(by="Importance", ascending=False)

# Select top 10 most impactful features
top_features = feature_importances.head(10)

# Plot top 10 features
plt.figure(figsize=(8,5))
sns.barplot(x="Importance", y="Feature", data=top_features, palette="coolwarm")
plt.title("Top 10 Drivers of Customer Churn (Decision Tree)", fontsize=14, weight
plt.xlabel("Relative Importance", fontsize=12)
plt.ylabel("Feature", fontsize=12)
plt.tight_layout()

# Save chart for PowerPoint
feature_chart_path = "/mnt/c/Users/rahro/Downloads/top_10_churn_features.png"
plt.savefig(feature_chart_path)
plt.show()

feature_chart_path
```


Top 10 Drivers of Customer Churn (Decision Tree)
Feature
total_day_charge
customer_service_calls
total_eve_charge
international_plan_yes
total_intl_calls
voice_mail_plan_yes
total_intl_minutes
total_day_minutes
total_eve_minutes
total_day_calls
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200

```
In [ ]:
```