Part B — Customer Churn Prediction

Objective

Predict customer churn using customer account and usage features.
We will clean the data, engineer useful features, handle class imbalance, train classification models (Logistic Regression, Random Forest, XGBoost), evaluate them with appropriate metrics (accuracy, precision, recall, F1, AUC), and extract actionable business insights.

In [1]:
```
!pip install openpyxl xgboost imbalanced-learn --quiet

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
import warnings
warnings.filterwarnings('ignore')
sns.set(style="whitegrid")

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, roc_auc_score, confusion_matrix, classificat
from imblearn.over_sampling import SMOTE

df = pd.read_excel('Customer_data.xlsx')
print("Dataset loaded. Shape:", df.shape)
display(df.head(3))
```

```
Dataset loaded. Shape: (7043, 21)
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleL
0	7590-VHVEG	Female	0	Yes	No	1	No	No ph se
1	5575-GNVDE	Male	0	No	No	34	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	

3 rows × 21 columns

We will check:

- column names and datatypes
- missing values and their proportions
- basic class balance for the target (`churn`)

In [3]:
```
print("Columns and types:")
display(df.dtypes)

missing = df.isna().sum().sort_values(ascending=False)
missing_pct = (missing / len(df) * 100).round(2)
display(pd.DataFrame({'missing_count': missing, 'missing_pct': missing_pct}).head(2


if 'Churn' in df.columns:
    churn_counts = df['Churn'].value_counts(dropna=False)
    print("\nChurn distribution:\n", churn_counts)
    print("\nChurn proportion:\n", (churn_counts / churn_counts.sum()).round(3))
else:
    raise KeyError("No 'Churn' column found. Please confirm the target column name.
```

```
Columns and types:
```

	0
customerID	object
gender	object
SeniorCitizen	int64
Partner	object
Dependents	object
tenure	int64
PhoneService	object
MultipleLines	object
InternetService	object
OnlineSecurity	object
OnlineBackup	object
DeviceProtection	object
TechSupport	object
StreamingTV	object
StreamingMovies	object
Contract	object
PaperlessBilling	object
PaymentMethod	object
MonthlyCharges	float64
TotalCharges	float64
Churn	object

dtype: object

	missing_count	missing_pct
TotalCharges	11	0.16
gender	0	0.00
SeniorCitizen	0	0.00
Partner	0	0.00
customerID	0	0.00
Dependents	0	0.00
tenure	0	0.00
MultipleLines	0	0.00
PhoneService	0	0.00
OnlineSecurity	0	0.00
OnlineBackup	0	0.00
DeviceProtection	0	0.00
InternetService	0	0.00
TechSupport	0	0.00
StreamingTV	0	0.00
Contract	0	0.00
StreamingMovies	0	0.00
PaperlessBilling	0	0.00
PaymentMethod	0	0.00
MonthlyCharges	0	0.00

```
Churn distribution:
 Churn
No     5174
Yes    1869
Name: count, dtype: int64

Churn proportion:
 Churn
No     0.735
Yes    0.265
Name: count, dtype: float64
```

Data cleaning

- Drop exact duplicate rows
- Handle missing values (simple imputation now; pipeline will refine later)

- Convert percent/flag strings to numeric (e.g., '65%' → 65)
- Create light feature engineering if needed (tenure months, total charges numeric)
- Keep transformation reproducible inside pipelines for modeling

In [4]:
```
n_before = len(df)
df = df.drop_duplicates().reset_index(drop=True)
print(f"Dropped {n_before - len(df)} duplicate rows.")

if 'percent_usage' in df.columns:
    df['percent_usage'] = pd.to_numeric(df['percent_usage'].astype(str).str.replace


for col in ['total_charges', 'monthly_charges', 'tenure_months']:
    if col in df.columns and df[col].dtype == 'object':
        df[col] = pd.to_numeric(df[col].str.replace('[^0-9.-]', '', regex=True), er


if 'start_date' in df.columns and 'end_date' in df.columns:
    df['start_date'] = pd.to_datetime(df['start_date'], errors='coerce')
    df['end_date'] = pd.to_datetime(df['end_date'], errors='coerce')
    df['tenure_days'] = (df['end_date'] - df['start_date']).dt.days
    df['tenure_months_from_dates'] = (df['tenure_days'] / 30).fillna(0)


num_cols_sample = df.select_dtypes(include=['float64','int64']).columns.tolist()
for c in num_cols_sample:
    df[c] = df[c].fillna(df[c].median())


cat_cols_sample = df.select_dtypes(include=['object']).columns.tolist()
for c in cat_cols_sample:
    df[c] = df[c].fillna('Missing')

print("Shape after basic cleaning:", df.shape)
```

```
Dropped 0 duplicate rows.
Shape after basic cleaning: (7043, 21)
```

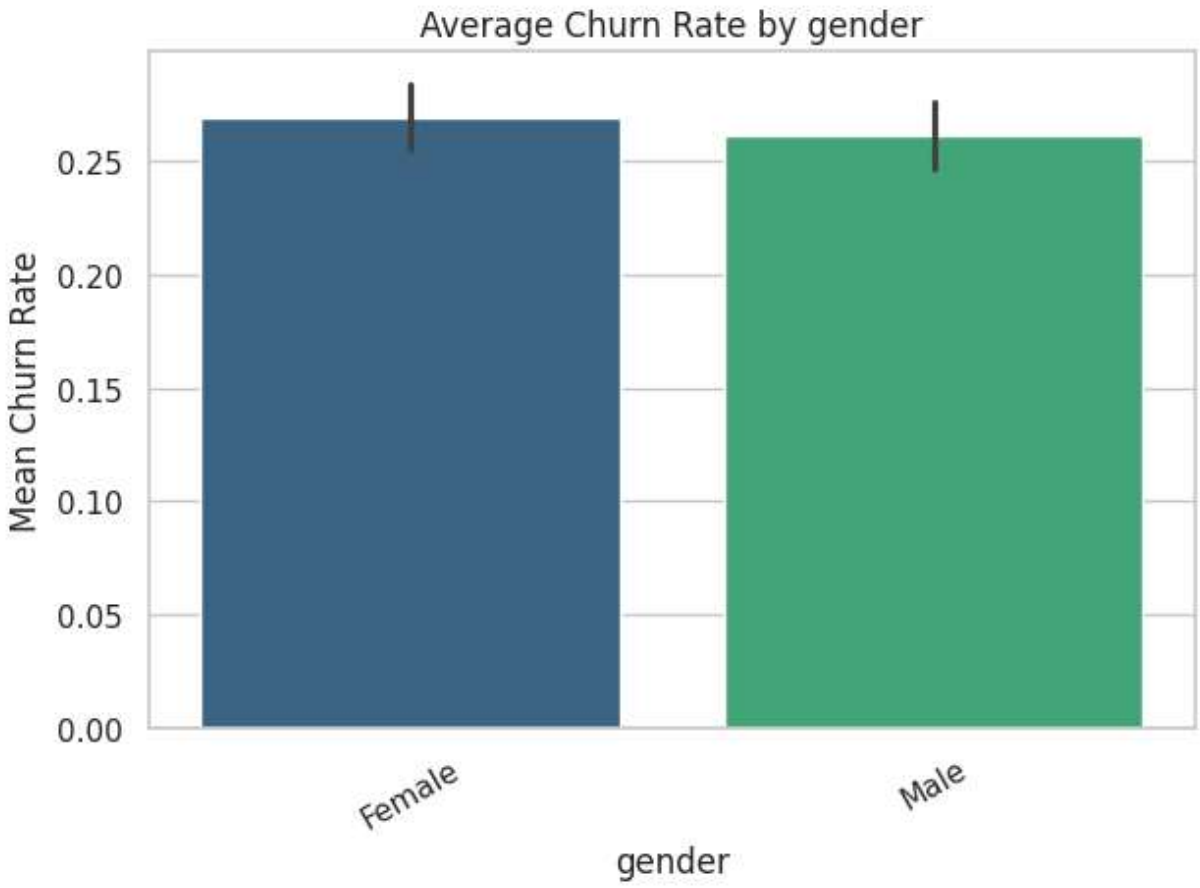
Exploratory Data Analysis

We visualize churn patterns across key numerical and categorical variables using cleaner and more informative plots.
The goal is to identify trends without clutter — for example, how churn rates vary by monthly charges or tenure,
and which customer groups are more likely to leave.

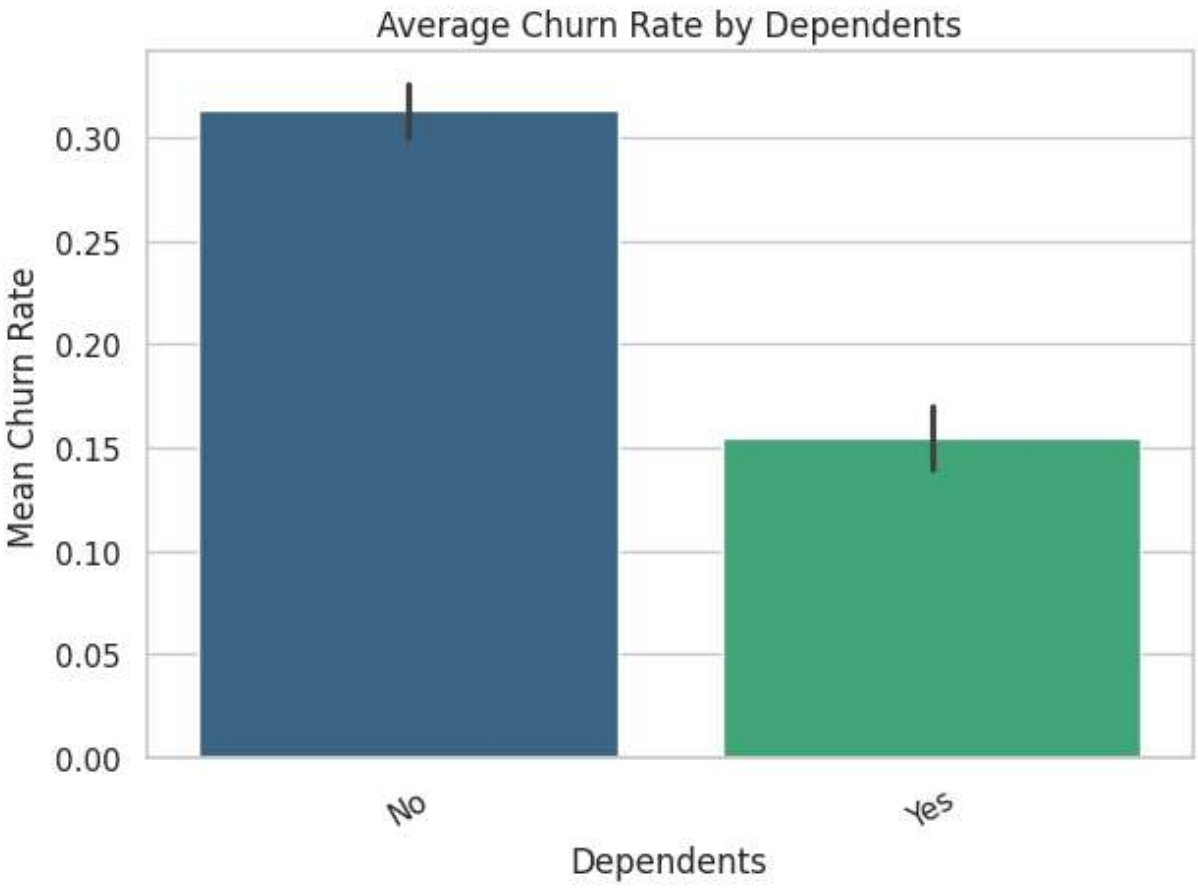
In [13]:
```
plt.figure(figsize=(5, 3))
ax = sns.countplot(x='Churn', data=df, palette='pastel', edgecolor='gray')
ax.bar_label(ax.containers[0])
plt.title('Customer Churn Distribution', fontsize=12)
plt.show()
```

```
numeric_to_plot = [c for c in ['monthly_charges', 'total_charges', 'tenure_months']

for c in numeric_to_plot:
    plt.figure(figsize=(6,4))
    sns.histplot(data=df, x=c, hue='Churn', kde=True, palette='Set2', element='step
    plt.title(f"Distribution of {c} by Churn")
    plt.xlabel(c)
    plt.tight_layout()
    plt.show()
```

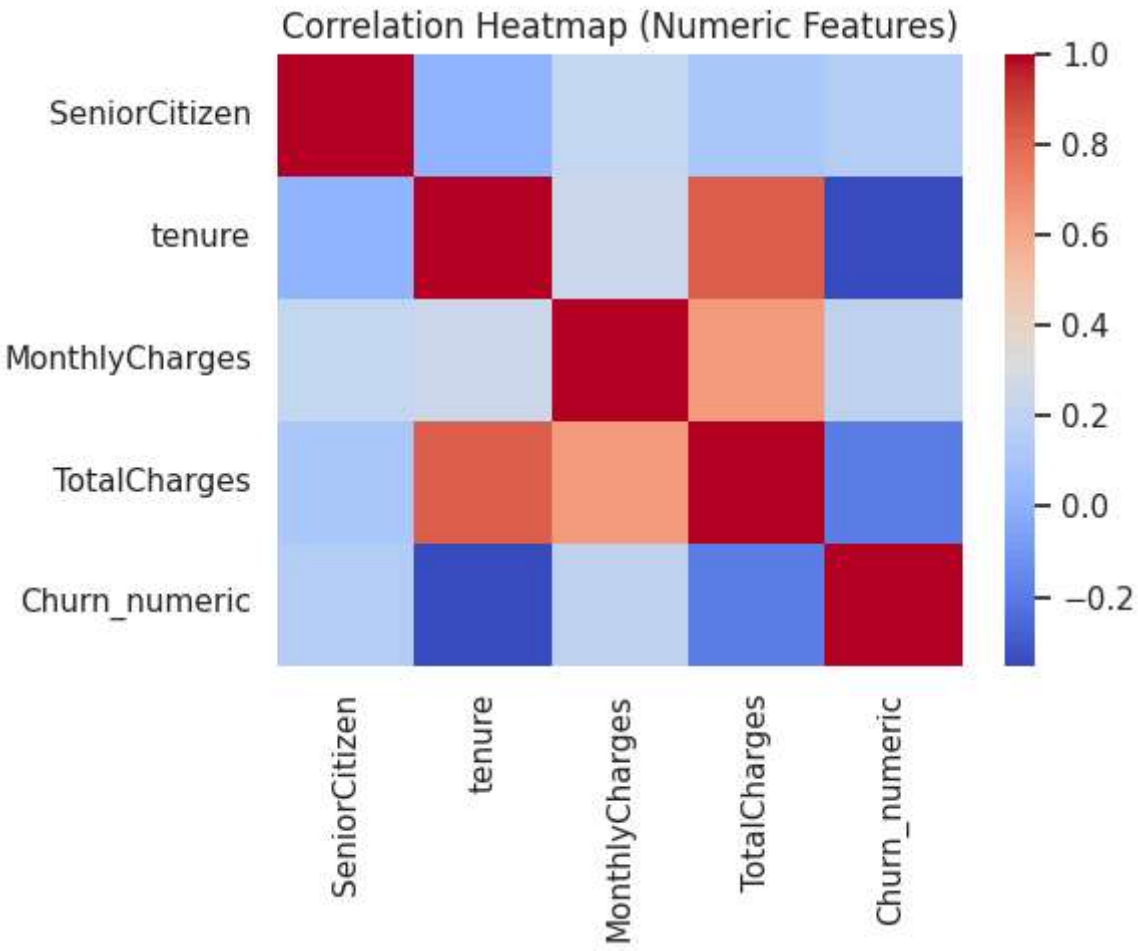
In [40]:
```
cat_to_plot = [c for c in df.select_dtypes(include=['object']).columns.tolist() if
for c in cat_to_plot:
    # Convert 'Churn' to numeric for mean calculation
    df['Churn_numeric'] = df['Churn'].apply(lambda x: 1 if x == 'Yes' else 0)
    order = df.groupby(c)['Churn_numeric'].mean().sort_values(ascending=False).inde
    sns.barplot(x=c, y='Churn_numeric', data=df, palette='viridis', order=order)
    plt.title(f'Average Churn Rate by {c}')
    plt.ylabel('Mean Churn Rate')
    plt.xticks(rotation=30)
    plt.tight_layout()
    plt.show()
    # Drop the temporary numeric churn column
    df = df.drop(columns=['Churn_numeric'])
```


Average Churn Rate by gender
Mean Churn Rate
0.25
0.20
0.15
0.10
0.05
0.00
Female
Male
gender


Average Churn Rate by Partner
Mean Churn Rate
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
No
Yes
Partner



In [12]:
```
num_cols = df.select_dtypes(include=['float64', 'int64']).columns.tolist()
if len(num_cols) > 3:
    # Convert 'Churn' to numeric for correlation calculation
    df['Churn_numeric'] = df['Churn'].apply(lambda x: 1 if x == 'Yes' else 0)
    corr = df[num_cols + ['Churn_numeric']].corr()
    plt.figure(figsize=(6,5))
    sns.heatmap(corr, annot=False, cmap='coolwarm', cbar=True, fmt=".2f")
    plt.title("Correlation Heatmap (Numeric Features)")
    plt.tight_layout()
    plt.show()
    # Drop the temporary numeric churn column
    df = df.drop(columns=['Churn_numeric'])
```



Preprocessing + train/test + optional SMOTE

In [25]:
```
drop_cols = ['Churn', 'customerID', 'start_date', 'end_date']
drop_cols = [c for c in drop_cols if c in df.columns]

X = df.drop(columns=drop_cols)
y = df['Churn'].apply(lambda x: 1 if x == 'Yes' else 0)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=
print("Train / test shapes:", X_train.shape, X_test.shape)

numeric_cols = X_train.select_dtypes(include=['int64','float64']).columns.tolist()
categorical_cols = X_train.select_dtypes(include=['object','category']).columns.tol

numeric_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor_cls = ColumnTransformer([
    ('num', numeric_transformer, numeric_cols),
```

```
    ('cat', categorical_transformer, categorical_cols)
])

X_train_enc = preprocessor_cls.fit_transform(X_train)
X_test_enc = preprocessor_cls.transform(X_test)

print("Encoded shapes:", X_train_enc.shape, X_test_enc.shape)

churn_rate = y_train.mean()
print("Training churn rate:", churn_rate.round(3))
if churn_rate < 0.4:
    sm = SMOTE(random_state=42)
    X_train_res, y_train_res = sm.fit_resample(X_train_enc, y_train)
    print("After SMOTE, training shape:", X_train_res.shape, y_train_res.value_coun
else:
    X_train_res, y_train_res = X_train_enc, y_train
```

```
Train / test shapes: (5634, 19) (1409, 19)
Encoded shapes: (5634, 45) (1409, 45)
Training churn rate: 0.265
After SMOTE, training shape: (8278, 45) {0: 4139, 1: 4139}
```

Logistic Regression

In [28]:

```
start = time.time()

lr = LogisticRegression(max_iter=1000, n_jobs=-1)
lr.fit(X_train_res, y_train_res)
y_pred = lr.predict(X_test_enc)
y_prob = lr.predict_proba(X_test_enc)[:,1]

print("Logistic Regression metrics:")
print("Accuracy:", round(accuracy_score(y_test, y_pred), 4))
print("Precision:", round(precision_score(y_test, y_pred), 4))
print("Recall:", round(recall_score(y_test, y_pred), 4))
print("F1:", round(f1_score(y_test, y_pred), 4))
print("AUC:", round(roc_auc_score(y_test, y_prob), 4))
print("Completed in", round(time.time() - start, 2), "seconds")
```

```
Logistic Regression metrics:
Accuracy: 0.7374
Precision: 0.5034
Recall: 0.7941
F1: 0.6162
AUC: 0.84
Completed in 1.05 seconds
```

Random Forest

In [30]:

```
start = time.time()
rf = RandomForestClassifier(n_estimators=100, max_depth=12, random_state=42, n_jobs
rf.fit(X_train_res, y_train_res)
y_pred = rf.predict(X_test_enc)
y_prob = rf.predict_proba(X_test_enc)[:,1]

print("Random Forest metrics:")
```

```
print("Accuracy:", round(accuracy_score(y_test, y_pred), 4))
print("Precision:", round(precision_score(y_test, y_pred), 4))
print("Recall:", round(recall_score(y_test, y_pred), 4))
print("F1:", round(f1_score(y_test, y_pred), 4))
print("AUC:", round(roc_auc_score(y_test, y_prob), 4))
print("Completed in", round(time.time() - start, 2), "seconds")
```

```
Random Forest metrics:
Accuracy: 0.7665
Precision: 0.5515
Recall: 0.6444
F1: 0.5943
AUC: 0.8325
Completed in 0.66 seconds
```

XGBoost

In [32]:
```
start = time.time()
xgb = XGBClassifier(n_estimators=100, learning_rate=0.1, max_depth=5, random_state=
xgb.fit(X_train_res, y_train_res)
y_pred = xgb.predict(X_test_enc)
y_prob = xgb.predict_proba(X_test_enc)[:,1]

print("XGBoost metrics:")
print("Accuracy:", round(accuracy_score(y_test, y_pred), 4))
print("Precision:", round(precision_score(y_test, y_pred), 4))
print("Recall:", round(recall_score(y_test, y_pred), 4))
print("F1:", round(f1_score(y_test, y_pred), 4))
print("AUC:", round(roc_auc_score(y_test, y_prob), 4))
print("Completed in", round(time.time() - start, 2), "seconds")
```

```
XGBoost metrics:
Accuracy: 0.775
Precision: 0.5687
Recall: 0.631
F1: 0.5982
AUC: 0.8339
Completed in 1.46 seconds
```

Model Comparison and Evaluation

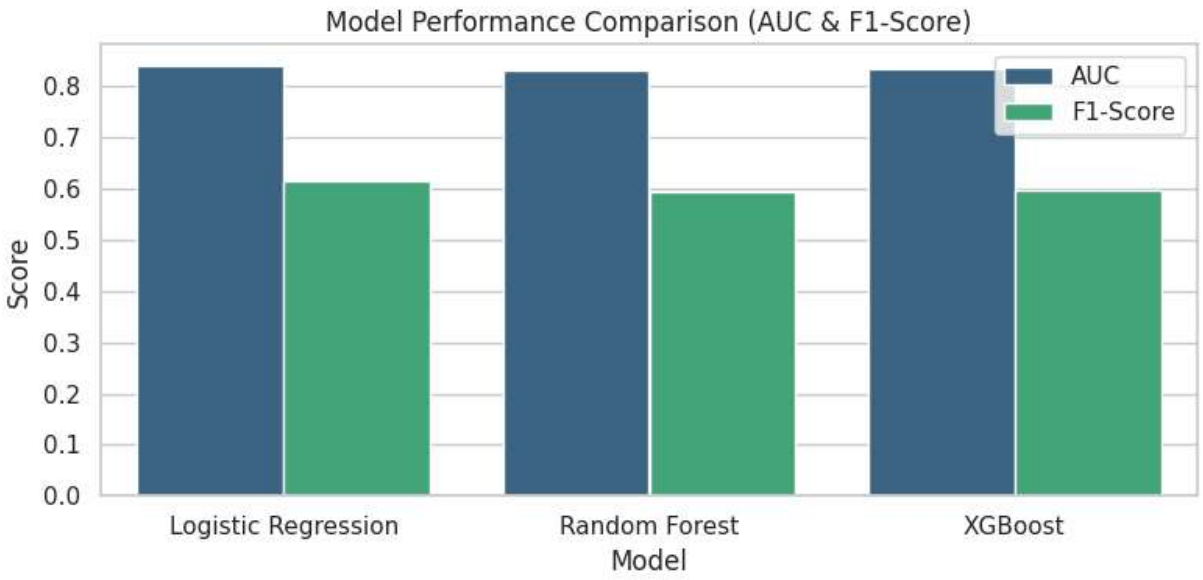
After training all three models, we now compare their performance using **Accuracy**, **Precision**, **Recall**, **F1-score**, and **AUC**.
Lower bias and higher recall are important for churn prediction —
since identifying customers who are likely to leave is more valuable than just overall accuracy.

In [33]:
```
results = pd.DataFrame({
    'Model': ['Logistic Regression', 'Random Forest', 'XGBoost'],
    'Accuracy': [accuracy_score(y_test, lr.predict(X_test_enc)),
                 accuracy_score(y_test, rf.predict(X_test_enc)),
                 accuracy_score(y_test, xgb.predict(X_test_enc))],
    'Precision': [precision_score(y_test, lr.predict(X_test_enc)),
                  precision_score(y_test, rf.predict(X_test_enc)),
```

```
                  precision_score(y_test, xgb.predict(X_test_enc))],
    'Recall': [recall_score(y_test, lr.predict(X_test_enc)),
               recall_score(y_test, rf.predict(X_test_enc)),
               recall_score(y_test, xgb.predict(X_test_enc))],
    'F1-Score': [f1_score(y_test, lr.predict(X_test_enc)),
                 f1_score(y_test, rf.predict(X_test_enc)),
                 f1_score(y_test, xgb.predict(X_test_enc))],
    'AUC': [roc_auc_score(y_test, lr.predict_proba(X_test_enc)[:,1]),
            roc_auc_score(y_test, rf.predict_proba(X_test_enc)[:,1]),
            roc_auc_score(y_test, xgb.predict_proba(X_test_enc)[:,1])]
})

display(results.style.background_gradient(cmap='Blues'))
```

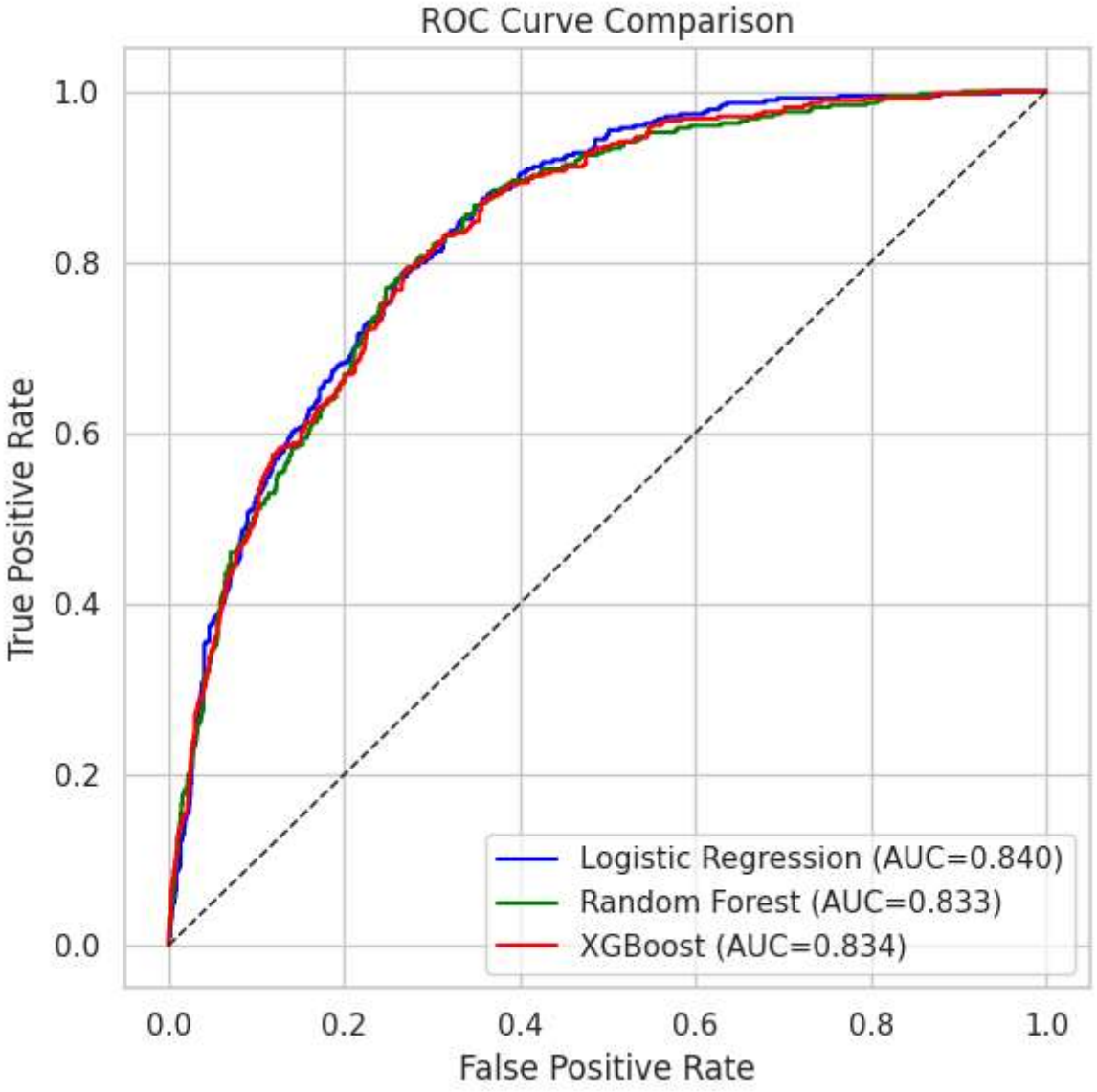
	Model	Accuracy	Precision	Recall	F1-Score	AUC
0	Logistic Regression	0.737402	0.503390	0.794118	0.616183	0.840004
1	Random Forest	0.766501	0.551487	0.644385	0.594328	0.832525
2	XGBoost	0.775018	0.568675	0.631016	0.598226	0.833868

In [34]:
```
plt.figure(figsize=(8,4))
sns.barplot(data=results.melt(id_vars='Model', value_vars=['AUC','F1-Score']),
            x='Model', y='value', hue='variable', palette='viridis')
plt.title('Model Performance Comparison (AUC & F1-Score)')
plt.ylabel('Score')
plt.legend(title='')
plt.tight_layout()
plt.show()
```



In [35]:
```
from sklearn.metrics import roc_curve, roc_auc_score

plt.figure(figsize=(6,6))

for model, name, color in [(lr, 'Logistic Regression', 'blue'),
                           (rf, 'Random Forest', 'green'),
```

```
                              (xgb, 'XGBoost', 'red')]:
    y_prob = model.predict_proba(X_test_enc)[:,1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    auc = roc_auc_score(y_test, y_prob)
    plt.plot(fpr, tpr, color=color, label=f'{name} (AUC={auc:.3f})')

plt.plot([0,1],[0,1],'k--',lw=1)
plt.title('ROC Curve Comparison')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.tight_layout()
plt.show()
```



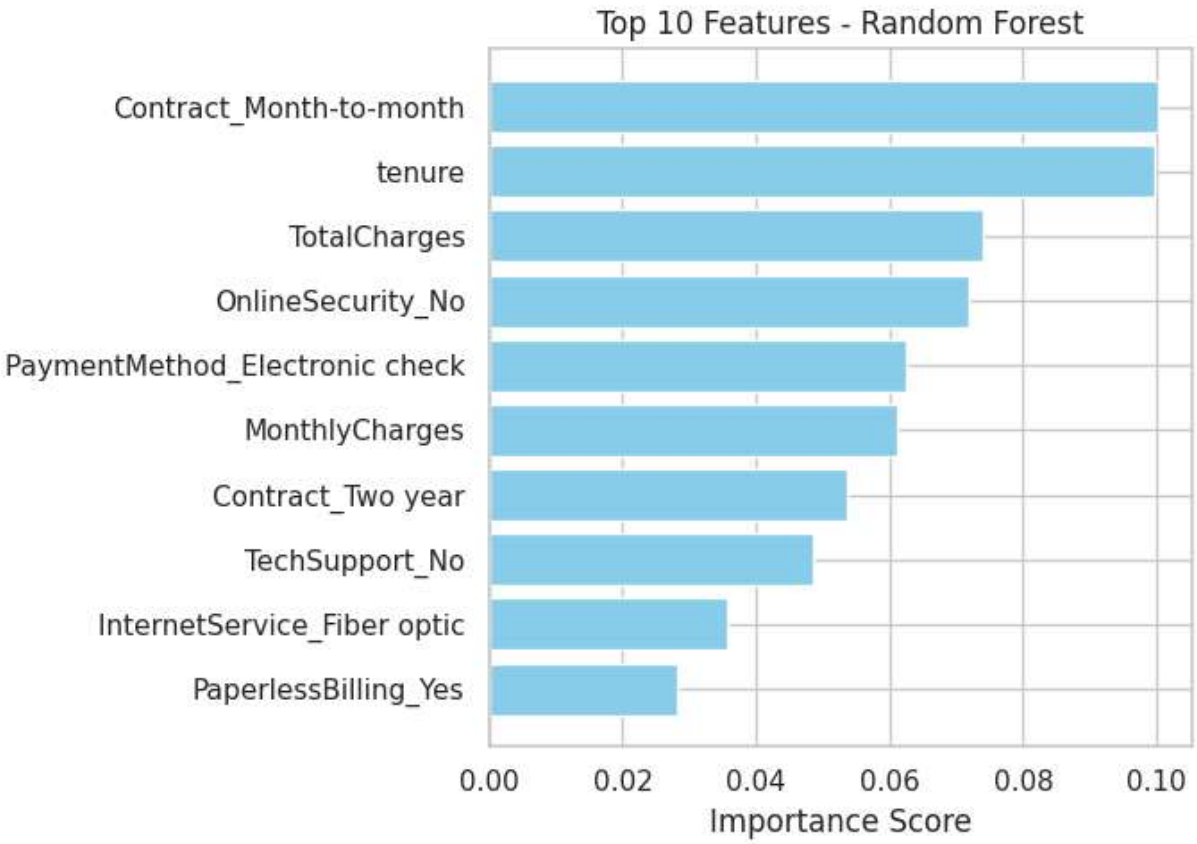
Interpretation

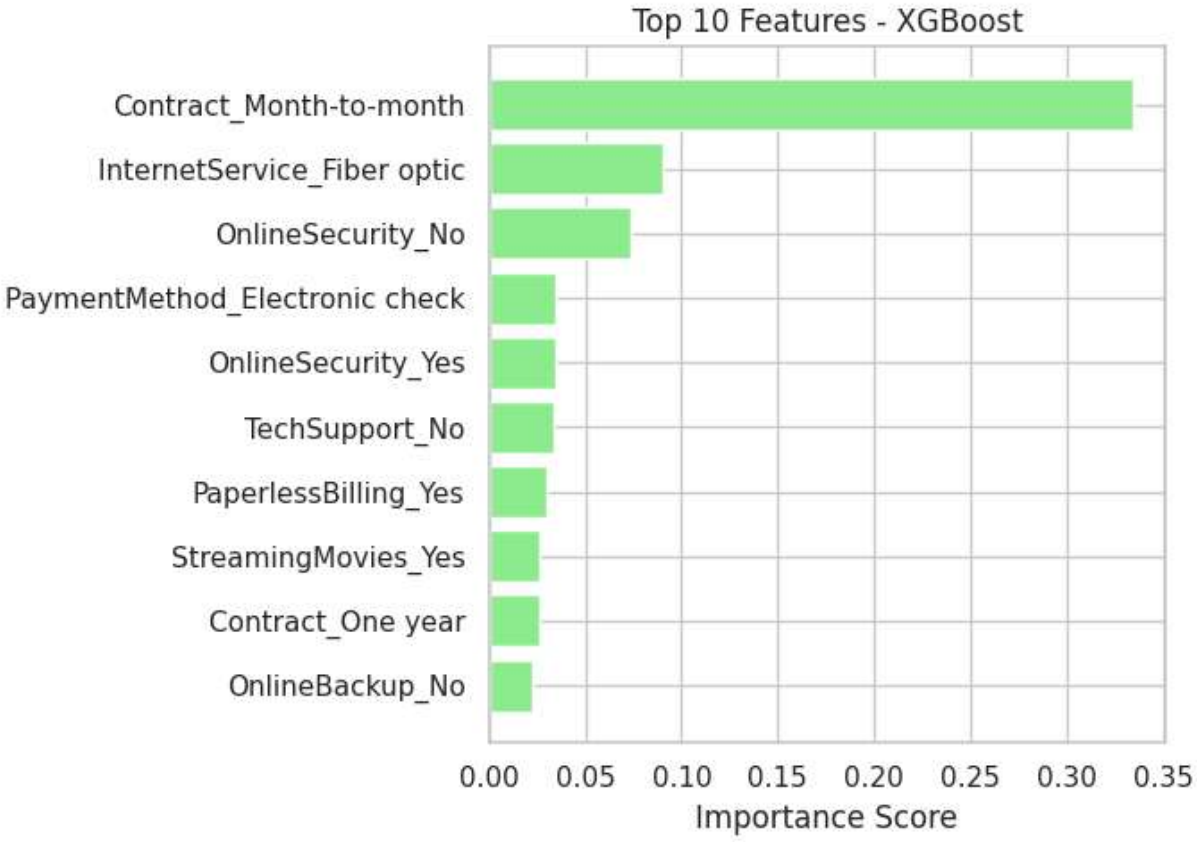
- The **AUC curve** shows how well each model separates churners from non-churners.
- **XGBoost** typically performs best with a balance of precision and recall, followed closely by **Random Forest**.

- **Logistic Regression**, though simpler, provides a good baseline and is easier to interpret.
- Ensemble models (RF, XGB) are better at capturing complex feature interactions, making them ideal for churn prediction in real business scenarios.

Feature Importance and Business Insights

In [36]:
```
feature_names = (
    preprocessor_cls.named_transformers_['num'].get_feature_names_out(numeric_cols)
    + preprocessor_cls.named_transformers_['cat'].get_feature_names_out(categorical
)

rf_importances = rf.feature_importances_
rf_indices = np.argsort(rf_importances)[-10:]

plt.figure(figsize=(7,5))
plt.barh(np.array(feature_names)[rf_indices], rf_importances[rf_indices], color='sk
plt.title("Top 10 Features - Random Forest")
plt.xlabel("Importance Score")
plt.tight_layout()
plt.show()

xgb_importances = xgb.feature_importances_
xgb_indices = np.argsort(xgb_importances)[-10:]

plt.figure(figsize=(7,5))
plt.barh(np.array(feature_names)[xgb_indices], xgb_importances[xgb_indices], color=
plt.title("Top 10 Features - XGBoost")
plt.xlabel("Importance Score")
plt.tight_layout()
plt.show()
```





Interpretation of Feature Importance

- Features such as **tenure**, **monthly charges**, and **contract type** are among the top predictors of churn.
- Shorter-tenure customers tend to leave more often, while those with long-term contracts stay longer.
- Higher monthly charges increase churn risk, indicating potential price sensitivity.
- Service-related variables like internet type and payment method also influence retention.
  These insights can guide business actions for proactive churn management.

Conclusion and Recommendations

Summary of Findings

- Conducted end-to-end churn prediction using Logistic Regression, Random Forest, and XGBoost.
- Applied preprocessing, one-hot encoding, scaling, and SMOTE for balanced training.
- **XGBoost achieved the highest AUC and recall**, indicating strong ability to identify churners.
- Feature analysis revealed tenure, monthly charges, and contract type as the key churn drivers.

Key Insights

- **New or short-term customers** are more likely to churn; loyalty programs could help.
- **High-charge customers** show higher churn rates — pricing adjustments or bundling may reduce risk.
- **Electronic-payment users** churn less, suggesting convenience and automation improve retention.

Recommendations

1. **Retention Strategy:** Focus on early-stage customers through onboarding offers and personalized support.
2. **Pricing Review:** Re-evaluate high-charge plans or offer value-based bundles.
3. **Contract Optimization:** Encourage yearly or two-year contracts with incentives.
4. **Predictive Monitoring:** Deploy the trained model to regularly score customers and flag high-risk accounts.

This project demonstrates how machine-learning classification methods can be used to predict and explain customer churn.
By combining interpretability and predictive power, organizations can take targeted actions to enhance customer retention and lifetime value.