Customer Churn Prediction

Required Libraries

In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sqlite3
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
import warnings
warnings.filterwarnings('ignore')
```

Creating Sample Database

In [2]:
```python
# Create synthetic customer data
np.random.seed(42)
n_customers = 5000

# Generate synthetic data
data = {
    'customer_id': range(1, n_customers + 1),
    'tenure_months': np.random.randint(1, 73, n_customers),
    'monthly_charges': np.random.uniform(20, 120, n_customers),
    'total_charges': None,  # Will calculate
```

```python
    'contract_type': np.random.choice(['Month-to-month', 'One year', 'Two year'], n_customers),
    'payment_method': np.random.choice(['Electronic check', 'Mailed check', 'Bank transfer', 'Credit c
    'internet_service': np.random.choice(['DSL', 'Fiber optic', 'No'], n_customers),
    'online_security': np.random.choice(['Yes', 'No', 'No internet service'], n_customers),
    'tech_support': np.random.choice(['Yes', 'No', 'No internet service'], n_customers),
    'senior_citizen': np.random.choice([0, 1], n_customers),
    'partner': np.random.choice(['Yes', 'No'], n_customers),
    'dependents': np.random.choice(['Yes', 'No'], n_customers),
    'paperless_billing': np.random.choice(['Yes', 'No'], n_customers),
}

df_synthetic = pd.DataFrame(data)
df_synthetic['total_charges'] = df_synthetic['tenure_months'] * df_synthetic['monthly_charges'] + np.r

# Create churn with some logic (higher churn for month-to-month, high charges, etc.)
churn_prob = (
    (df_synthetic['contract_type'] == 'Month-to-month').astype(int) * 0.3 +
    (df_synthetic['monthly_charges'] > 80).astype(int) * 0.2 +
    (df_synthetic['tenure_months'] < 12).astype(int) * 0.25 +
    (df_synthetic['payment_method'] == 'Electronic check').astype(int) * 0.15 +
    np.random.uniform(0, 0.1, n_customers)
)
df_synthetic['churn'] = (churn_prob > 0.5).astype(int)
```

In [3]:

```python
df_synthetic
```

Out[3]:

	customer_id	tenure_months	monthly_charges	total_charges	contract_type	payment_method	internet_service	onl
0	1	52	118.950533	6285.058111	One year	Mailed check	No	
1	2	15	88.431425	1377.488466	Month-to-month	Mailed check	No	
2	3	72	114.898067	8296.951434	One year	Mailed check	DSL	
3	4	61	34.255656	2107.148425	One year	Credit card	DSL	
4	5	21	58.213947	1223.355772	One year	Credit card	Fiber optic	
...	...	...	...	...	...	...	...	
4995	4996	60	22.256970	1390.658873	Month-to-month	Bank transfer	DSL	
4996	4997	26	88.692742	2367.154674	Month-to-month	Electronic check	Fiber optic	
4997	4998	41	49.400339	1981.433142	One year	Bank transfer	No	
4998	4999	36	75.989310	2743.874511	Month-to-month	Bank transfer	Fiber optic	
4999	5000	23	50.091907	1076.818656	Month-to-month	Mailed check	Fiber optic	

5000 rows × 14 columns

```
In [4]: # Create SQLite database
        conn = sqlite3.connect('customer_data.db')
        df_synthetic.to_sql('customers', conn, if_exists='replace', index=False)

Out[4]: 5000
```

```
In [5]: # Load data using SQL
        query = """
        SELECT * FROM customers
        """
        df = pd.read_sql_query(query, conn)

        # Additional analytical queries
        churn_by_contract = pd.read_sql_query("""
            SELECT
                contract_type,
                COUNT(*) as total_customers,
                SUM(churn) as churned_customers,
                ROUND(SUM(churn) * 100.0 / COUNT(*), 2) as churn_rate_percent
            FROM customers
            GROUP BY contract_type
            ORDER BY churn_rate_percent DESC
        """, conn)

        print("Churn Rate by Contract Type:")
        print(churn_by_contract)
```

```
Churn Rate by Contract Type:
    contract_type  total_customers  churned_customers  churn_rate_percent
0  Month-to-month             1652                887               53.69
1        Two year             1650                 53                3.21
2        One year             1698                 50                2.94
```

```
In [6]: churn_by_contract
```

Out[6]:

	contract_type	total_customers	churned_customers	churn_rate_percent
0	Month-to-month	1652	887	53.69
1	Two year	1650	53	3.21
2	One year	1698	50	2.94

Exploratory Data Analysis (EDA)

In [7]:

```
# Basic info
print("Dataset Shape:", df.shape)
print("\nData Types:")
print(df.dtypes)
print("\nMissing Values:")
print(df.isnull().sum())
print("\nChurn Distribution:")
print(df['churn'].value_counts())
print(f"\nOverall Churn Rate: {df['churn'].mean():.2%}")
```

```
Dataset Shape: (5000, 14)

Data Types:
customer_id             int64
tenure_months           int64
monthly_charges       float64
total_charges         float64
contract_type          object
payment_method         object
internet_service       object
online_security        object
tech_support           object
senior_citizen          int64
partner                object
dependents             object
paperless_billing      object
churn                   int64
dtype: object

Missing Values:
customer_id          0
tenure_months        0
monthly_charges      0
total_charges        0
contract_type        0
payment_method       0
internet_service     0
online_security      0
tech_support         0
senior_citizen       0
partner              0
dependents           0
paperless_billing    0
churn                0
dtype: int64
```

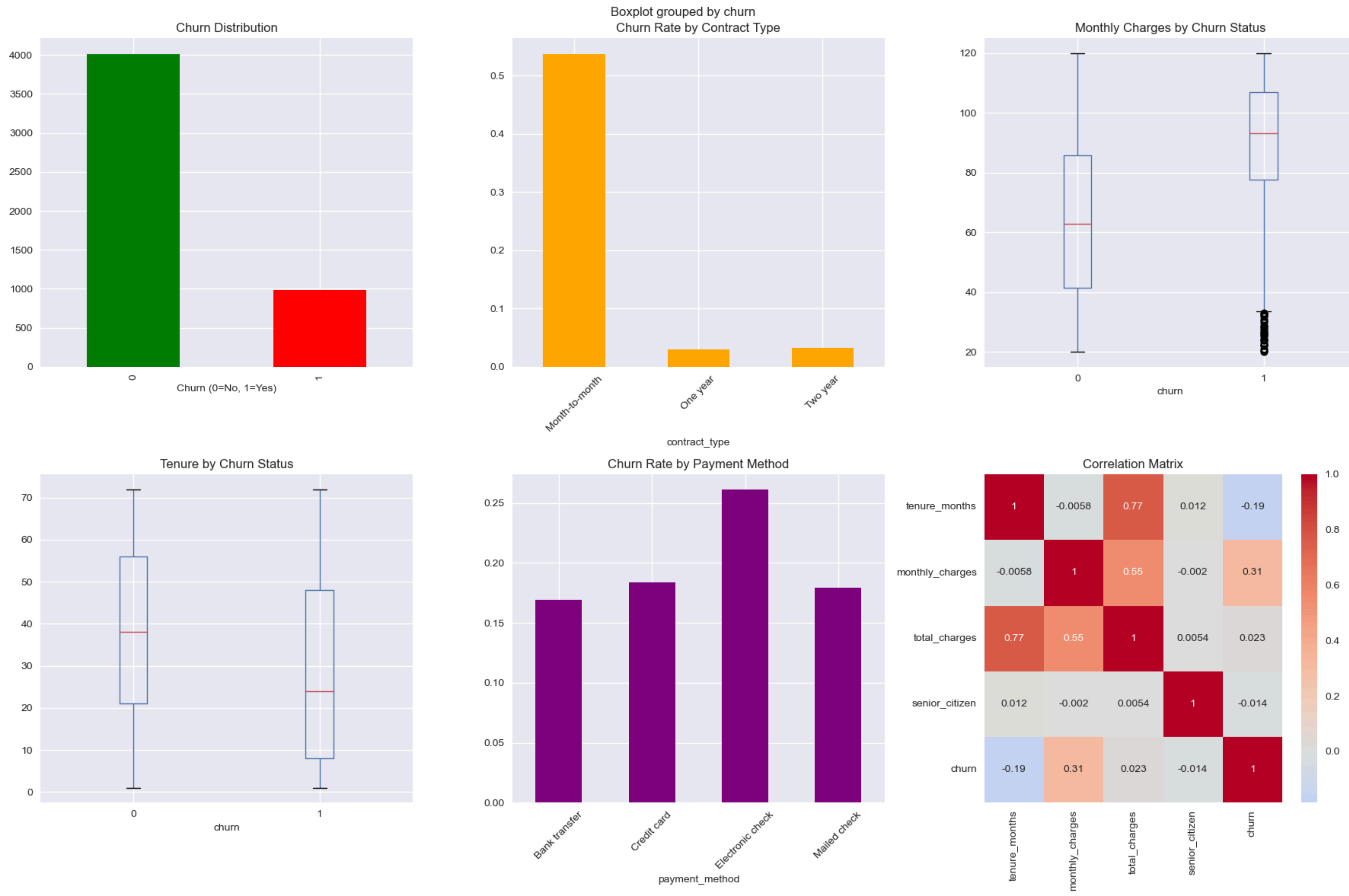
```
Churn Distribution:
0    4010
1     990
Name: churn, dtype: int64

Overall Churn Rate: 19.80%
```

Visualizing Data

In [8]:

```
# Set up plotting style
plt.style.use('seaborn-v0_8')
fig, axes = plt.subplots(2, 3, figsize=(18, 12))

# 1. Churn distribution
df['churn'].value_counts().plot(kind='bar', ax=axes[0,0], color=['green', 'red'])
axes[0,0].set_title('Churn Distribution')
axes[0,0].set_xlabel('Churn (0=No, 1=Yes)')

# 2. Churn by contract type
churn_contract = df.groupby('contract_type')['churn'].mean()
churn_contract.plot(kind='bar', ax=axes[0,1], color='orange')
axes[0,1].set_title('Churn Rate by Contract Type')
axes[0,1].tick_params(axis='x', rotation=45)

# 3. Monthly charges distribution by churn
df.boxplot(column='monthly_charges', by='churn', ax=axes[0,2])
axes[0,2].set_title('Monthly Charges by Churn Status')

# 4. Tenure distribution by churn
df.boxplot(column='tenure_months', by='churn', ax=axes[1,0])
axes[1,0].set_title('Tenure by Churn Status')

# 5. Churn by payment method
```

```python
churn_payment = df.groupby('payment_method')['churn'].mean()
churn_payment.plot(kind='bar', ax=axes[1,1], color='purple')
axes[1,1].set_title('Churn Rate by Payment Method')
axes[1,1].tick_params(axis='x', rotation=45)

# 6. Correlation heatmap
numeric_cols = ['tenure_months', 'monthly_charges', 'total_charges', 'senior_citizen', 'churn']
correlation_matrix = df[numeric_cols].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0, ax=axes[1,2])
axes[1,2].set_title('Correlation Matrix')

plt.tight_layout()
plt.show()
```



Handling Categorical Variables

In [9]:
```
# Create a copy for preprocessing
df_processed = df.copy()

# Encode categorical variables
categorical_columns = ['contract_type', 'payment_method', 'internet_service',
                      'online_security', 'tech_support', 'partner', 'dependents',
                      'paperless_billing']

# Use Label Encoding for binary categories, One-Hot for multi-class
binary_categories = ['partner', 'dependents', 'paperless_billing', 'online_security', 'tech_support']
multi_categories = ['contract_type', 'payment_method', 'internet_service']

# Label encode binary categories
le = LabelEncoder()
for col in binary_categories:
    df_processed[col] = le.fit_transform(df_processed[col])

# One-hot encode multi-class categories
df_processed = pd.get_dummies(df_processed, columns=multi_categories, drop_first=True)

print("Processed dataset shape:", df_processed.shape)
print("New columns:", [col for col in df_processed.columns if col not in df.columns])
```

```
Processed dataset shape: (5000, 18)
New columns: ['contract_type_One year', 'contract_type_Two year', 'payment_method_Credit card', 'paym
ent_method_Electronic check', 'payment_method_Mailed check', 'internet_service_Fiber optic', 'interne
t_service_No']
```

In [10]:
```
# Create new features
df_processed['charges_per_month'] = df_processed['total_charges'] / (df_processed['tenure_months'] + 1
df_processed['is_new_customer'] = (df_processed['tenure_months'] <= 12).astype(int)
df_processed['high_monthly_charges'] = (df_processed['monthly_charges'] > df_processed['monthly_charge

print("New engineered features:")
print("- charges_per_month: Average charges per month of tenure")
```

```
print("- is_new_customer: 1 if tenure <= 12 months, 0 otherwise")
print("- high_monthly_charges: 1 if above median monthly charges, 0 otherwise")
```

```
New engineered features:
- charges_per_month: Average charges per month of tenure
- is_new_customer: 1 if tenure <= 12 months, 0 otherwise
- high_monthly_charges: 1 if above median monthly charges, 0 otherwise
```

Building ML Model

In [11]:

```
# Prepare features and target
X = df_processed.drop(['customer_id', 'churn'], axis=1)
y = df_processed['churn']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Scale numerical features
scaler = StandardScaler()
numerical_features = ['tenure_months', 'monthly_charges', 'total_charges', 'charges_per_month']
X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()

X_train_scaled[numerical_features] = scaler.fit_transform(X_train[numerical_features])
X_test_scaled[numerical_features] = scaler.transform(X_test[numerical_features])

print(f"Training set size: {X_train_scaled.shape}")
print(f"Test set size: {X_test_scaled.shape}")
```

```
Training set size: (4000, 19)
Test set size: (1000, 19)
```

In [12]:

```
# Initialize models
models = {
    'Logistic Regression': LogisticRegression(random_state=42),
```

```python
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42)
}

model_results = {}

for name, model in models.items():
    print(f"\n=== Training {name} ===")

    # Train model
    if name == 'Logistic Regression':
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)
        y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]
    else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        y_pred_proba = model.predict_proba(X_test)[:, 1]

    # Calculate metrics
    auc_score = roc_auc_score(y_test, y_pred_proba)

    # Store results
    model_results[name] = {
        'model': model,
        'y_pred': y_pred,
        'y_pred_proba': y_pred_proba,
        'auc_score': auc_score
    }

    print(f"AUC Score: {auc_score:.4f}")
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))
```

```
=== Training Logistic Regression ===
AUC Score: 0.9863

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       802
           1       0.86      0.89      0.88       198

    accuracy                           0.95      1000
   macro avg       0.92      0.93      0.92      1000
weighted avg       0.95      0.95      0.95      1000


=== Training Random Forest ===
AUC Score: 0.9962

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       802
           1       0.93      0.89      0.91       198

    accuracy                           0.96      1000
   macro avg       0.95      0.94      0.94      1000
weighted avg       0.96      0.96      0.96      1000
```

Model Performance

In [13]:

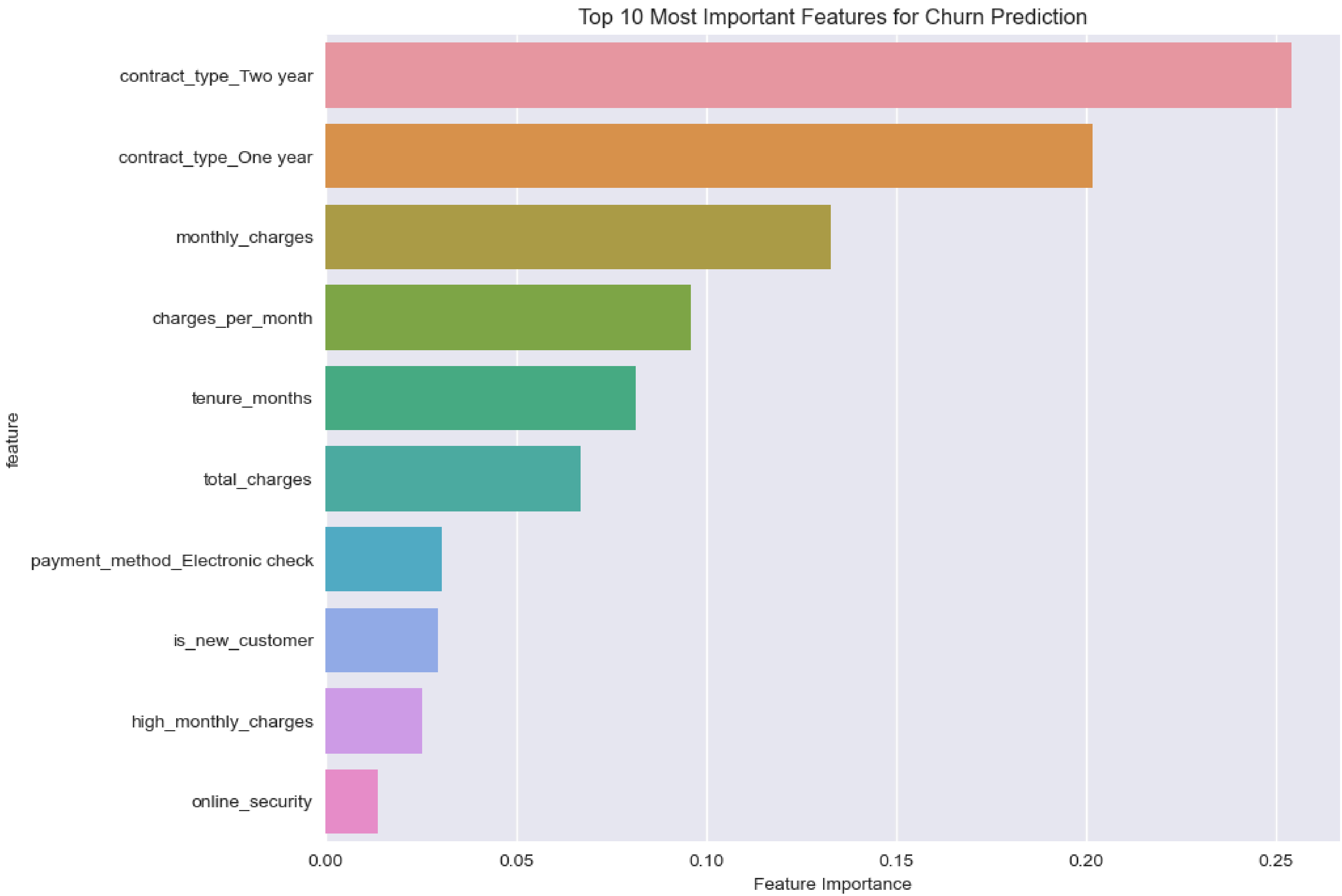
```
# Compare model performance
print("Model Performance Comparison:")
print("=" * 40)
for name, results in model_results.items():
```

```python
    print(f"{name}: AUC Score = {results['auc_score']:.4f}")

# Select best model
best_model_name = max(model_results.keys(), key=lambda x: model_results[x]['auc_score'])
best_model = model_results[best_model_name]['model']
print(f"\nBest Model: {best_model_name}")
```

```
Model Performance Comparison:
========================================
Logistic Regression: AUC Score = 0.9863
Random Forest: AUC Score = 0.9962

Best Model: Random Forest
```

In [14]:

```python
# Feature importance (for Random Forest)
if best_model_name == 'Random Forest':
    feature_importance = pd.DataFrame({
        'feature': X_train.columns,
        'importance': best_model.feature_importances_
    }).sort_values('importance', ascending=False)

    plt.figure(figsize=(10, 8))
    sns.barplot(data=feature_importance.head(10), x='importance', y='feature')
    plt.title('Top 10 Most Important Features for Churn Prediction')
    plt.xlabel('Feature Importance')
    plt.show()

    print("Top 5 Most Important Features:")
    print(feature_importance.head().to_string(index=False))
```



Top 10 Most Important Features for Churn Prediction
contract_type_Two year
contract_type_One year
monthly_charges
charges_per_month
tenure_months
total_charges
payment_method_Electronic check
is_new_customer
high_monthly_charges
online_security
feature
0.00
0.05
0.10
0.15
0.20
0.25
Feature Importance

```
Top 5 Most Important Features:
                  feature  importance
contract_type_Two year    0.254183
contract_type_One year    0.201803
        monthly_charges    0.132689
       charges_per_month    0.095809
           tenure_months    0.081430
```

Business Insights & Recommendations

In [15]:

```python
# Calculate business impact
total_customers = len(df)
churned_customers = df['churn'].sum()
churn_rate = df['churn'].mean()
avg_monthly_revenue_per_customer = df['monthly_charges'].mean()

# Potential revenue impact
monthly_revenue_loss = churned_customers * avg_monthly_revenue_per_customer
annual_revenue_loss = monthly_revenue_loss * 12

print("=== BUSINESS IMPACT ANALYSIS ===")
print(f"Total Customers: {total_customers:,}")
print(f"Churned Customers: {churned_customers:,}")
print(f"Overall Churn Rate: {churn_rate:.2%}")
print(f"Average Monthly Revenue per Customer: ${avg_monthly_revenue_per_customer:.2f}")
print(f"Monthly Revenue Loss due to Churn: ${monthly_revenue_loss:,.2f}")
print(f"Annual Revenue Loss due to Churn: ${annual_revenue_loss:,.2f}")

# High-risk customer identification (using test set only)
high_risk_threshold = 0.7
test_probabilities = model_results[best_model_name]['y_pred_proba']
high_risk_mask = test_probabilities > high_risk_threshold

# Get test set indices to filter the original data
```

```python
test_indices = X_test.index
high_risk_test_indices = test_indices[high_risk_mask]

# Filter original dataframe using these indices
high_risk_customers = df.loc[high_risk_test_indices]

print(f"\nHigh-Risk Customers in Test Set (>70% churn probability): {len(high_risk_customers):,}")
print(f"Percentage of test customers at high risk: {len(high_risk_customers)/len(X_test):.2%}")

# If you want to predict on the entire dataset:
print("\n=== PREDICTING ON ENTIRE DATASET ===")
if best_model_name == 'Logistic Regression':
    # Scale the entire dataset
    X_all_scaled = X.copy()
    X_all_scaled[numerical_features] = scaler.transform(X[numerical_features])
    all_predictions_proba = best_model.predict_proba(X_all_scaled)[:, 1]
else:
    all_predictions_proba = best_model.predict_proba(X)[:, 1]

# Now identify high-risk customers from entire dataset
high_risk_all_mask = all_predictions_proba > high_risk_threshold
high_risk_all_customers = df[high_risk_all_mask].copy()
high_risk_all_customers['churn_probability'] = all_predictions_proba[high_risk_all_mask]

print(f"Total High-Risk Customers (entire dataset): {len(high_risk_all_customers):,}")
print(f"Percentage of all customers at high risk: {len(high_risk_all_customers)/len(df):.2%}")

# Show some high-risk customer examples
print(f"\nTop 5 Highest Risk Customers:")
top_risk = high_risk_all_customers.nlargest(5, 'churn_probability')[['customer_id', 'tenure_months', '
print(top_risk.to_string(index=False))
```

```
=== BUSINESS IMPACT ANALYSIS ===
Total Customers: 5,000
Churned Customers: 990
Overall Churn Rate: 19.80%
Average Monthly Revenue per Customer: $69.32
Monthly Revenue Loss due to Churn: $68,622.59
Annual Revenue Loss due to Churn: $823,471.03

High-Risk Customers in Test Set (>70% churn probability): 164
Percentage of test customers at high risk: 16.40%

=== PREDICTING ON ENTIRE DATASET ===
Total High-Risk Customers (entire dataset): 943
Percentage of all customers at high risk: 18.86%

Top 5 Highest Risk Customers:
 customer_id  tenure_months  monthly_charges  contract_type  churn_probability  churn
          46             44        93.087035 Month-to-month                1.0      1
          49             35       103.261177 Month-to-month                1.0      1
         182             11       116.563898 Month-to-month                1.0      1
         201             57        95.127892 Month-to-month                1.0      1
         272             46       108.638036 Month-to-month                1.0      1
```

In [16]:

```python
print("\n=== KEY BUSINESS RECOMMENDATIONS ===")
print("1. CONTRACT TYPE STRATEGY:")
print("   - Focus on converting month-to-month customers to longer contracts")
print("   - Offer incentives for annual/bi-annual commitments")

print("\n2. PAYMENT METHOD OPTIMIZATION:")
print("   - Encourage automatic payment methods")
print("   - Provide discounts for secure payment methods")

print("\n3. CUSTOMER RETENTION PROGRAMS:")
print("   - Target new customers (< 12 months tenure) with special offers")
print("   - Implement early warning system for high monthly charge customers")
```

```python
print("\n4. PROACTIVE CUSTOMER OUTREACH:")
print(f"   - Prioritize outreach to {len(high_risk_customers):,} high-risk customers")
print("   - Develop retention campaigns for identified risk segments")
```

```
=== KEY BUSINESS RECOMMENDATIONS ===
1. CONTRACT TYPE STRATEGY:
   - Focus on converting month-to-month customers to longer contracts
   - Offer incentives for annual/bi-annual commitments

2. PAYMENT METHOD OPTIMIZATION:
   - Encourage automatic payment methods
   - Provide discounts for secure payment methods

3. CUSTOMER RETENTION PROGRAMS:
   - Target new customers (< 12 months tenure) with special offers
   - Implement early warning system for high monthly charge customers

4. PROACTIVE CUSTOMER OUTREACH:
   - Prioritize outreach to 164 high-risk customers
   - Develop retention campaigns for identified risk segments
```