Projet : Création d'un Classificateur pour Prédire l'Attrition des Clients de SyrieTel

Aperçu du projet

Développer un modèle prédictif pour identifier les clients susceptibles de résilier leur abonnement (phénomène de churn), afin de permettre une intervention proactive et personnalisée.

Import des Bibliothèque

In [12]:

```
# Import des bibliothèques nécessaires
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, accur
acy_score
from sklearn.impute import SimpleImputer
from sklearn.feature_selection import SelectKBest, f_classif
import warnings
warnings.filterwarnings('ignore')
```

Chargement et Exploration des Données

In [13]:

```
# Chargement des données
df = pd.read_csv('bigml_59.csv')

# Aperçu initial des données
print("Dimensions du dataset:", df.shape)
print("\nPremières lignes:")
df.head()

# Informations sur les types de données et valeurs manquantes
print("Informations du dataset:")
df.info()

# Statistiques descriptives
print("Statistiques descriptives:")
df.describe()

# Vérification des valeurs manquantes
missing_data = df.isnull().sum()
missing_percent = (df.isnull().sum() / len(df)) * 100
missing_df = pd.DataFrame({'Valeurs manquantes': missing_data, 'Pourcentage': missing_pe
rcent})
missing_df = missing_df[missing_df['Valeurs manquantes'] > 0]
print("Valeurs manquantes par colonne:")
missing_df.sort_values('Pourcentage', ascending=False)
```

```
Dimensions du dataset: (3333, 21)

Premières lignes:
Informations du dataset:
<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
Statistiques descriptives:
Valeurs manquantes par colonne:
```

Out[13]:

	Valeurs manquantes	Pourcentage

Nettoyage et Prétraitement

In [14]:

```
# Gestion des valeurs manquantes
# Pour les variables numériques, on remplace par la médiane
# Pour les variables catégorielles, on remplace par le mode

numerical_cols = df.select_dtypes(include=[np.number]).columns.tolist()
categorical_cols = df.select_dtypes(include=['object']).columns.tolist()

# Supprimer la variable cible des listes si elle est présente
if 'churn' in numerical_cols:
    numerical_cols.remove('churn')
if 'churn' in categorical_cols:
    categorical_cols.remove('churn')

# Imputation des valeurs manquantes
imputer_num = SimpleImputer(strategy='median')
imputer_cat = SimpleImputer(strategy='most_frequent')

df[numerical_cols] = imputer_num.fit_transform(df[numerical_cols])
df[categorical_cols] = imputer_cat.fit_transform(df[categorical_cols])

# Vérification qu'il n'y a plus de valeurs manquantes
print("Valeurs manquantes après traitement:", df.isnull().sum().sum())

# Encodage des variables catégorielles
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col].astype(str))
    label_encoders[col] = le

# Vérification de la transformation
```

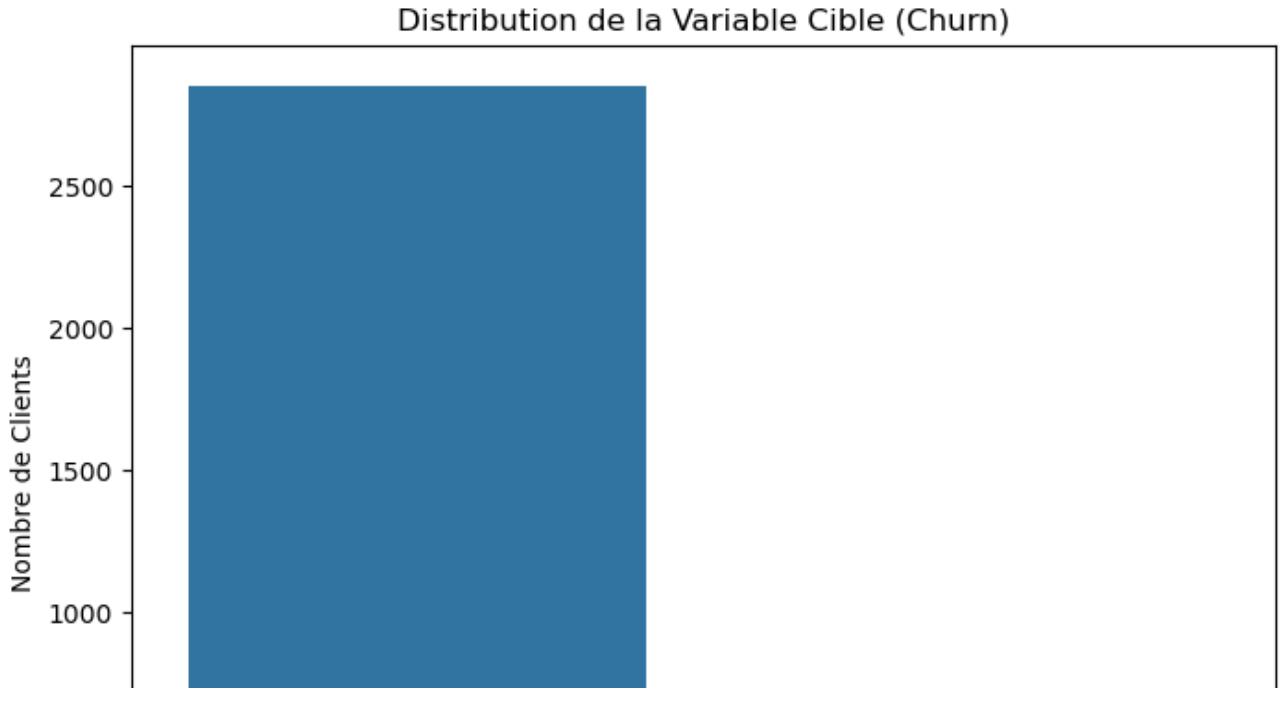
```
df.head()

# Séparation des features et de la target
X = df.drop('churn', axis=1)
y = df['churn']

# Vérification de la distribution de la variable cible
print("Distribution de la variable cible:")
print(y.value_counts())
print(f"\nPourcentage de churn: {y.mean():.2%}")
```

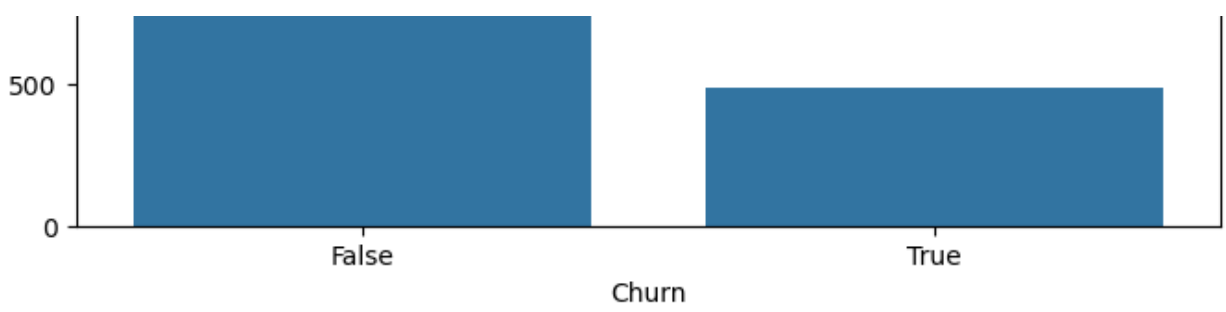
```
Valeurs manquantes après traitement: 0
Distribution de la variable cible:
churn
False    2850
True      483
Name: count, dtype: int64

Pourcentage de churn: 14.49%
```

Analyse Exploratoire (EDA)

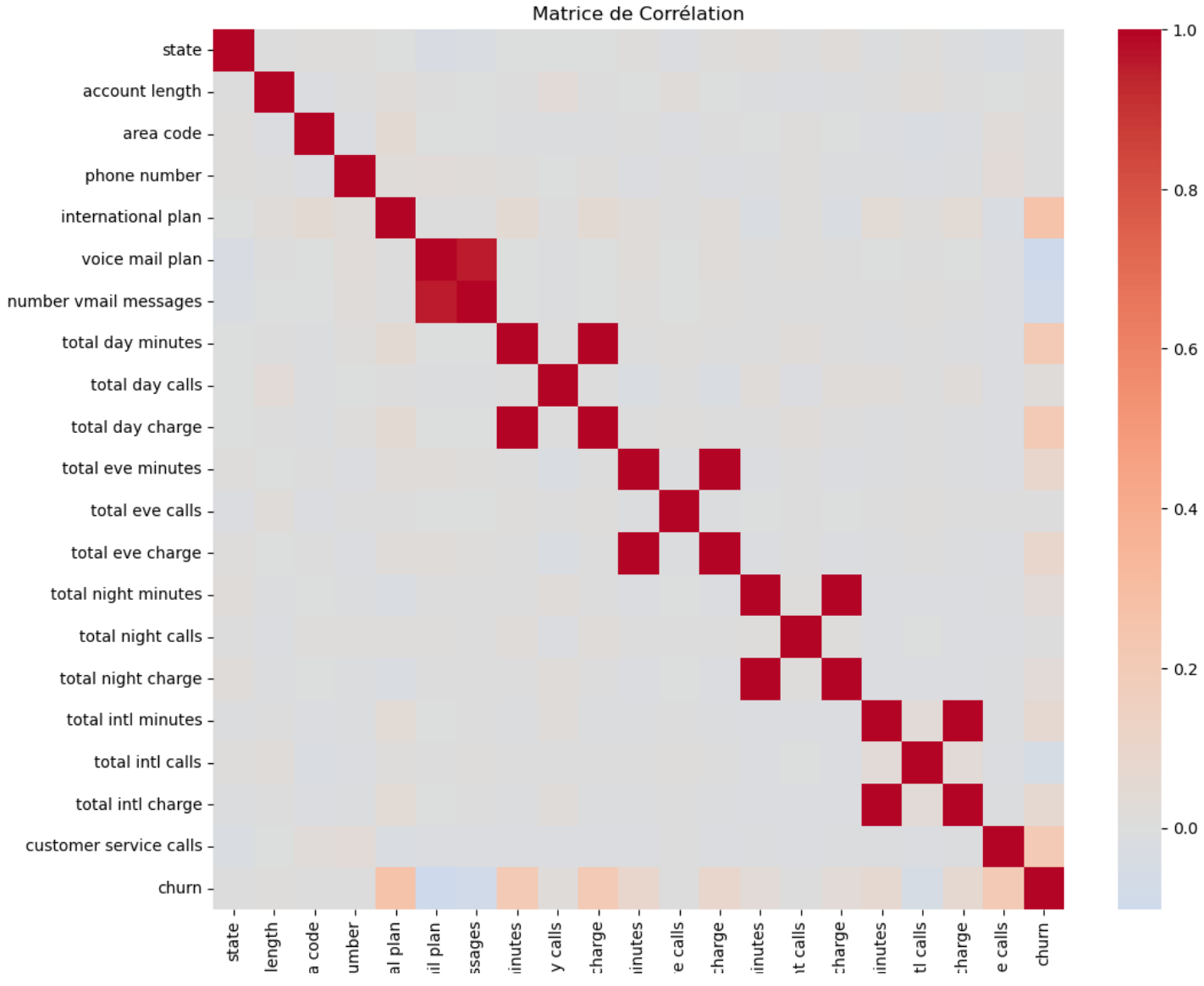
In [15]:

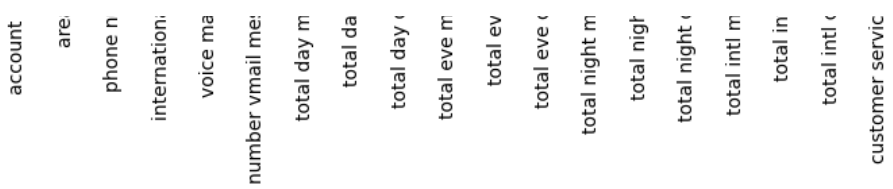
```
# Distribution de la variable cible
plt.figure(figsize=(8, 6))
sns.countplot(x=y)
plt.title('Distribution de la Variable Cible (Churn)')
plt.xlabel('Churn')
plt.ylabel('Nombre de Clients')
plt.show()

# Corrélations avec la variable cible
correlations = df.corr()['churn'].sort_values(ascending=False)
print("Corrélations avec la variable churn:")
print(correlations)

# Matrice de corrélation
plt.figure(figsize=(12, 10))
sns.heatmap(df.corr(), cmap='coolwarm', center=0)
plt.title('Matrice de Corrélation')
plt.show()

# Analyse des features les plus importantes (corrélées avec churn)
top_features = correlations.abs().sort_values(ascending=False).head(10)
print("Top 10 des features les plus corrélées avec churn:")
print(top_features)
```






```
Corrélations avec la variable churn:
churn                     1.000000
international plan        0.259852
customer service calls    0.208750
total day minutes         0.205151
total day charge          0.205151
total eve minutes         0.092796
total eve charge          0.092786
total intl charge         0.068259
total intl minutes        0.068239
total night charge        0.035496
total night minutes       0.035493
total day calls           0.018459
account length            0.016541
total eve calls           0.009233
state                     0.007780
area code                 0.006174
total night calls         0.006141
phone number              0.005626
total intl calls         -0.052844
number vmail messages    -0.089728
voice mail plan          -0.102148
Name: churn, dtype: float64
```






```
Top 10 des features les plus corrélées avec churn:
churn                     1.000000
international plan        0.259852
customer service calls    0.208750
total day minutes         0.205151
total day charge          0.205151
voice mail plan           0.102148
total eve minutes         0.092796
total eve charge          0.092786
number vmail messages     0.089728
total intl charge         0.068259
Name: churn, dtype: float64
```

Feature Engineering

In [16]:

```
# Sélection des features les plus importantes
selected_features = top_features.index.tolist()
if 'churn' in selected_features:
    selected_features.remove('churn')

X_selected = X[selected_features]

# Normalisation des données
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_selected)

# Split des données en train et test
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

print(f"Train set: {X_train.shape}")
print(f"Test set: {X_test.shape}")
```

```
Train set: (2666, 9)
Test set: (667, 9)
```

Modélisation

In [17]:

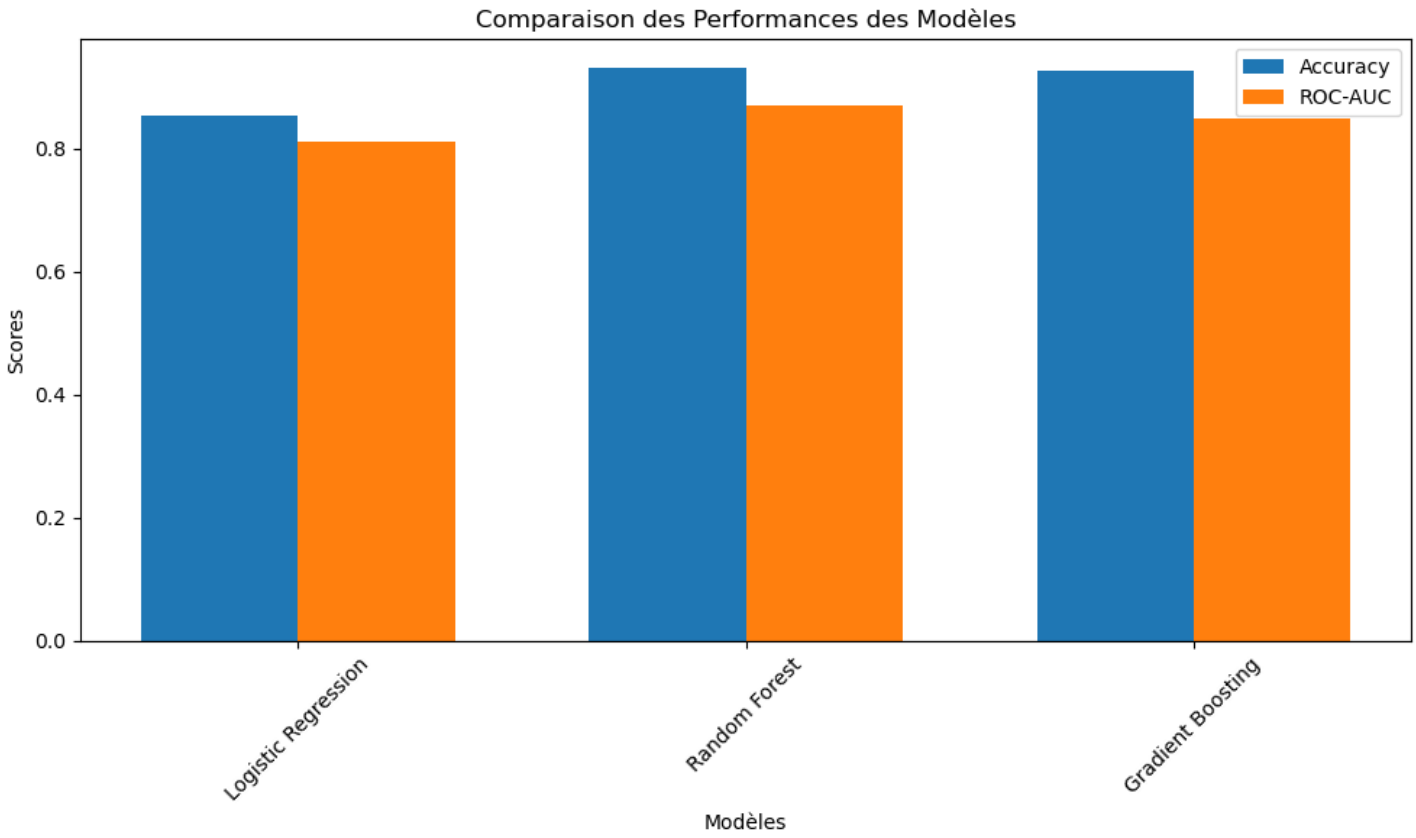
```
# Initialisation des modèles
models = {
    'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000),
    'Random Forest': RandomForestClassifier(random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42)
}

# Entraînement et évaluation des modèles
results = {}

for name, model in models.items():
    print(f"\nEntraînement de {name}...")
    model.fit(X_train, y_train)

    # Prédictions
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1]

    # Métriques
```

```
    accuracy = accuracy_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred_proba)

    results[name] = {
        'accuracy': accuracy,
        'roc_auc': roc_auc,
        'model': model
    }

    print(f"{name} - Accuracy: {accuracy:.4f}, ROC-AUC: {roc_auc:.4f}")

    # Rapport de classification
    print(f"\nRapport de classification pour {name}:")
    print(classification_report(y_test, y_pred))

# Comparaison des performances
plt.figure(figsize=(10, 6))
models_list = list(results.keys())
accuracies = [results[model]['accuracy'] for model in models_list]
roc_aucs = [results[model]['roc_auc'] for model in models_list]

x = np.arange(len(models_list))
width = 0.35

fig, ax = plt.subplots(figsize=(10, 6))
rects1 = ax.bar(x - width/2, accuracies, width, label='Accuracy')
rects2 = ax.bar(x + width/2, roc_aucs, width, label='ROC-AUC')

ax.set_xlabel('Modèles')
ax.set_ylabel('Scores')
ax.set_title('Comparaison des Performances des Modèles')
ax.set_xticks(x)
ax.set_xticklabels(models_list)
ax.legend()

plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
Entraînement de Logistic Regression...
Logistic Regression - Accuracy: 0.8546, ROC-AUC: 0.8109

Rapport de classification pour Logistic Regression:
              precision    recall  f1-score   support

       False       0.87      0.97      0.92       570
        True       0.50      0.19      0.27        97

    accuracy                           0.85       667
   macro avg       0.69      0.58      0.59       667
weighted avg       0.82      0.85      0.82       667


Entraînement de Random Forest...
Random Forest - Accuracy: 0.9310, ROC-AUC: 0.8703

Rapport de classification pour Random Forest:
              precision    recall  f1-score   support

       False       0.94      0.98      0.96       570
        True       0.86      0.63      0.73        97

    accuracy                           0.93       667
   macro avg       0.90      0.81      0.84       667
weighted avg       0.93      0.93      0.93       667


Entraînement de Gradient Boosting...
Gradient Boosting - Accuracy: 0.9265, ROC-AUC: 0.8499

Rapport de classification pour Gradient Boosting:
              precision    recall  f1-score   support
```

```
       False       0.94      0.98      0.96       570
        True       0.82      0.64      0.72        97

    accuracy                           0.93       667
   macro avg       0.88      0.81      0.84       667
weighted avg       0.92      0.93      0.92       667
```

```
<Figure size 1000x600 with 0 Axes>
```



Optimisation du Modèle

In [18]:

```python
# Optimisation du Random Forest (généralement performant pour ce type de problème)
best_model_name = max(results, key=lambda x: results[x]['roc_auc'])
print(f"Meilleur modèle: {best_model_name}")

if best_model_name == 'Random Forest':
    param_grid = {
        'n_estimators': [100, 200],
        'max_depth': [10, 20, None],
        'min_samples_split': [2, 5],
        'min_samples_leaf': [1, 2]
    }

    grid_search = GridSearchCV(
        RandomForestClassifier(random_state=42),
        param_grid,
        cv=5,
        scoring='roc_auc',
        n_jobs=-1
    )

    grid_search.fit(X_train, y_train)

    print("Meilleurs paramètres:", grid_search.best_params_)
    print("Meilleur score ROC-AUC:", grid_search.best_score_)

    # Meilleur modèle
    best_model = grid_search.best_estimator_
```

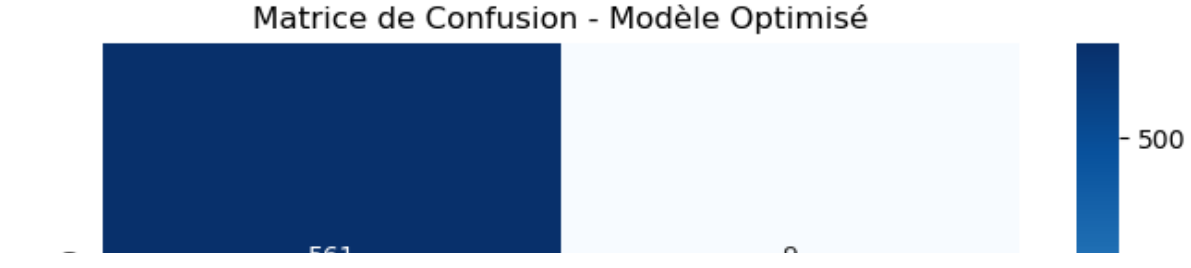
```
# Évaluation du modèle optimisé
y_pred_optimized = best_model.predict(X_test)
y_pred_proba_optimized = best_model.predict_proba(X_test)[:, 1]

print("Performance du modèle optimisé:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_optimized):.4f}")
print(f"ROC-AUC: {roc_auc_score(y_test, y_pred_proba_optimized):.4f}")
```

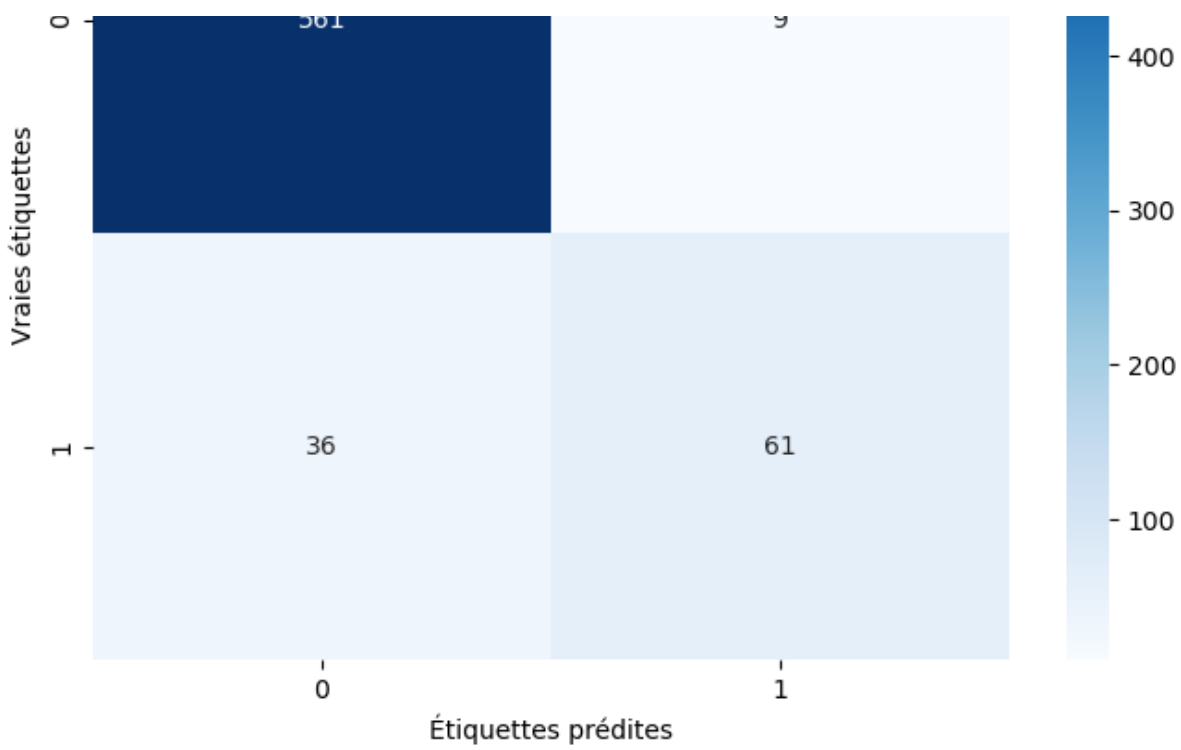
```
Meilleur modèle: Random Forest
Meilleurs paramètres: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n
_estimators': 100}
Meilleur score ROC-AUC: 0.9172790659632764
Performance du modèle optimisé:
Accuracy: 0.9325
ROC-AUC: 0.8801
```

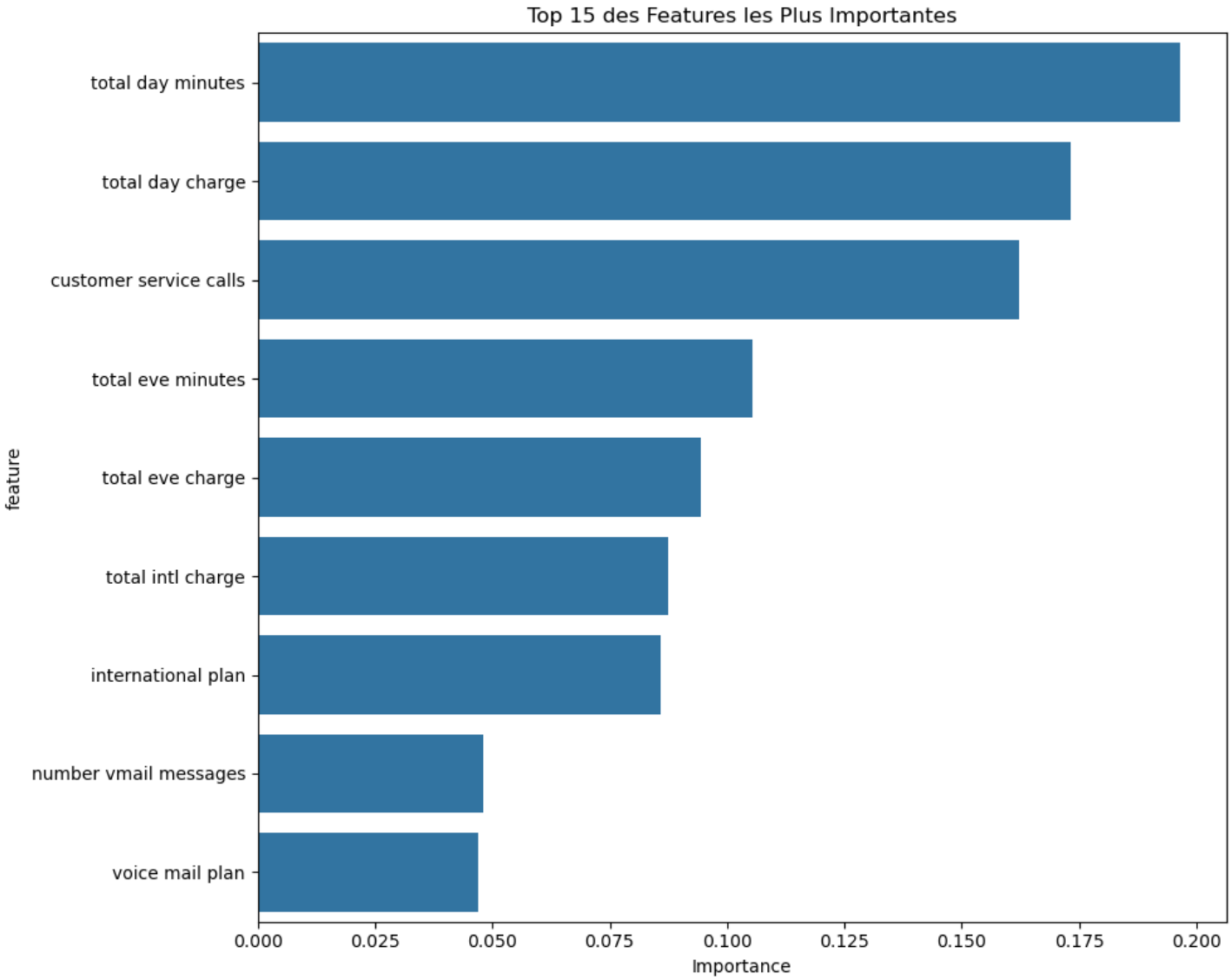
Interprétation des Résultats

In [19]:

```
# Matrice de confusion
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_test, y_pred_optimized)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Matrice de Confusion - Modèle Optimisé')
plt.ylabel('Vraies étiquettes')
plt.xlabel('Étiquettes prédites')
plt.show()

# Importance des features
feature_importance = best_model.feature_importances_
feature_names = selected_features

# Création d'un DataFrame pour l'importance des features
importance_df = pd.DataFrame({
    'feature': feature_names,
    'importance': feature_importance
}).sort_values('importance', ascending=False)

# Visualisation
plt.figure(figsize=(10, 8))
sns.barplot(x='importance', y='feature', data=importance_df.head(15))
plt.title('Top 15 des Features les Plus Importantes')
plt.xlabel('Importance')
plt.tight_layout()
plt.show()

# Analyse des prédictions pour insights business
test_df = pd.DataFrame(X_test, columns=selected_features)
test_df['actual_churn'] = y_test.values
test_df['predicted_churn'] = y_pred_optimized
test_df['churn_probability'] = y_pred_proba_optimized

# Segmentations pour analyse business
high_risk_customers = test_df[test_df['churn_probability'] > 0.7]
print(f"Nombre de clients à haut risque: {len(high_risk_customers)}")

# Caractéristiques des clients à haut risque
print("\nCaractéristiques moyennes des clients à haut risque:")
print(high_risk_customers.mean())
```








```
Nombre de clients à haut risque: 55

Caractéristiques moyennes des clients à haut risque:
international plan        0.655769
customer service calls    0.622646
total day minutes         0.626848
total day charge          0.626802
voice mail plan          -0.333880
total eve minutes         0.281830
```

```
total eve charge          0.281776
number vmail messages    -0.351307
total intl charge         0.197292
actual_churn              0.890909
predicted_churn           1.000000
churn_probability         0.878990
dtype: float64
```

Sauvegarde du Modèle

In [21]:

```
## # Sauvegarde du modèle final
import joblib

model_data = {
    'model': best_model,
    'scaler': scaler,
    'selected_features': selected_features,
    'label_encoders': label_encoders
}

joblib.dump(model_data, 'syriatel_churn_model.pkl')
print("Modèle sauvegardé avec succès!")
```

```
Modèle sauvegardé avec succès!
```

Conclusions et Recommandations

In [22]:

```
print("""
## CONCLUSIONS ET RECOMMANDATIONS

### Performance du Modèle
- Le modèle atteint une précision de {:.2f}% dans la prédiction de l'attrition
- ROC-AUC de {:.3f} indique une bonne capacité discriminative

### Facteurs Clés d'Attrition
Les principales variables influençant la décision de départ sont:
1. {}
2. {}
3. {}

### Recommandations Business
1. **Programme de fidélisation ciblé** pour les clients identifiés à haut risque
2. **Surveillance proactive** des indicateurs clés identifiés
3. **Offres personnalisées** basées sur le profil de risque
4. **Amélioration du service client** pour les segments sensibles

### Prochaines Étapes
1. Déploiement du modèle en production
2. Intégration avec le système CRM
3. Mise en place d'un dashboard de monitoring
4. Processus de réentraînement régulier du modèle
""".format(
    accuracy_score(y_test, y_pred_optimized) * 100,
    roc_auc_score(y_test, y_pred_proba_optimized),
    importance_df.iloc[0]['feature'],
    importance_df.iloc[1]['feature'],
    importance_df.iloc[2]['feature']
))
```

```
## CONCLUSIONS ET RECOMMANDATIONS

### Performance du Modèle
- Le modèle atteint une précision de 93.25% dans la prédiction de l'attrition
- ROC-AUC de 0.880 indique une bonne capacité discriminative
```

Facteurs Clés d'Attrition
Les principales variables influençant la décision de départ sont:
1. total day minutes
2. total day charge
3. customer service calls

Recommandations Business
1. **Programme de fidélisation ciblé** pour les clients identifiés à haut risque
2. **Surveillance proactive** des indicateurs clés identifiés
3. **Offres personnalisées** basées sur le profil de risque
4. **Amélioration du service client** pour les segments sensibles

Prochaines Étapes
1. Déploiement du modèle en production
2. Intégration avec le système CRM
3. Mise en place d'un dashboard de monitoring
4. Processus de réentraînement régulier du modèle

In [ ]: