In [ ]:

```
!jupyter nbconvert --to webpdf --allow-chromium-download churnpredictionT1.ipynb
```

In [1]:

```
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, KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import (accuracy_score, precision_score, recall_score, f1_scor
                             roc_auc_score, confusion_matrix, ConfusionMatrixDisplay
                             classification_report, roc_curve)
from imblearn.over_sampling import SMOTE
import joblib
import warnings
warnings.filterwarnings('ignore')


#  for plots
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette("Set2")

# random seed for reproducibility
np.random.seed(42)
```

In [2]:

```
# load  dataset
df = pd.read_csv('churn_clean.csv')

# select relevant features based on EDA and domain knowledge
selected_features = [
    'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year', 'Outage_sec_perweek',
    'Contract', 'InternetService', 'PaperlessBilling', 'PaymentMethod',
    'OnlineSecurity', 'TechSupport', 'StreamingTV', 'StreamingMovies'
]

#  new dataframe with only the selected features and target variable
df_selected = df[selected_features + ['Churn']]

# prepare features and target
X = df_selected.drop('Churn', axis=1)
y = (df_selected['Churn'] == 'Yes').astype(int)  # Convert to binary (1 for churn,

# split data into training, validation, and test sets
from sklearn.model_selection import train_test_split

# First split: training+validation and test (80/20)
X_train_val, X_test, y_train_val, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Second split: training and validation (75/25 of the 80% = 60/20 overall)
X_train, X_val, y_train, y_val = train_test_split(
```

```
    X_train_val, y_train_val, test_size=0.25, random_state=42, stratify=y_train_val
)

# checking sizxe of each set
print(f"Training set: {X_train.shape[0]} samples")
print(f"Validation set: {X_val.shape[0]} samples")
print(f"Test set: {X_test.shape[0]} samples")

# check class distribution in each set
print("\nClass distribution:")
print(f"Training set: {y_train.mean()*100:.2f}% churn")
print(f"Validation set: {y_val.mean()*100:.2f}% churn")
print(f"Test set: {y_test.mean()*100:.2f}% churn")

# identify categorical and numerical features
categorical_features = [
    'Contract', 'InternetService', 'PaperlessBilling', 'PaymentMethod',
    'OnlineSecurity', 'TechSupport', 'StreamingTV', 'StreamingMovies'
]

numerical_features = [
    'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year', 'Outage_sec_perweek'
]
```

```
Training set: 6000 samples
Validation set: 2000 samples
Test set: 2000 samples

Class distribution:
Training set: 26.50% churn
Validation set: 26.50% churn
Test set: 26.50% churn
```

In [3]:

```
#  preprocessing steps --pipelines
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(drop='first'), categorical_features)
    ]
)

# create a pipeline for the entire workflow
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))
])
```

In [4]:

```
# apply SMOTE to balance the training data -- handliong class imbalance SMOTE
smote = SMOTE(random_state=42)
X_train_processed = preprocessor.fit_transform(X_train)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_processed, y_trai




# Checking class distribution after SMOTE
```

```
print("\nClass distribution after SMOTE:")
print(f"Original training set: {pd.Series(y_train).value_counts()}")
print(f"Resampled training set: {pd.Series(y_train_resampled).value_counts()}")
```

```
Class distribution after SMOTE:
Original training set: Churn
0    4410
1    1590
Name: count, dtype: int64
Resampled training set: Churn
0    4410
1    4410
Name: count, dtype: int64
```

In [ ]:

```

```

In [5]:

```
# train the initial Random Forest model --
initial_clf = RandomForestClassifier(random_state=42)
initial_clf.fit(X_train_resampled, y_train_resampled)

# make predictions on the training set for initial evaluation
y_train_pred = initial_clf.predict(X_train_processed)
y_train_pred_proba = initial_clf.predict_proba(X_train_processed)[:, 1]


# calculate metrics for the initial model on training data
train_accuracy = accuracy_score(y_train, y_train_pred)
train_precision = precision_score(y_train, y_train_pred)
train_recall = recall_score(y_train, y_train_pred)
train_f1 = f1_score(y_train, y_train_pred)
train_auc_roc = roc_auc_score(y_train, y_train_pred_proba)
train_cm = confusion_matrix(y_train, y_train_pred)


# display metrics for the initial model
print("\nInitial Model Metrics (Training Data):")
print(f"Accuracy: {train_accuracy:.4f}")
print(f"Precision: {train_precision:.4f}")
print(f"Recall: {train_recall:.4f}")
print(f"F1 Score: {train_f1:.4f}")
print(f"AUC-ROC: {train_auc_roc:.4f}")
print("Confusion Matrix:")
print(train_cm)
```

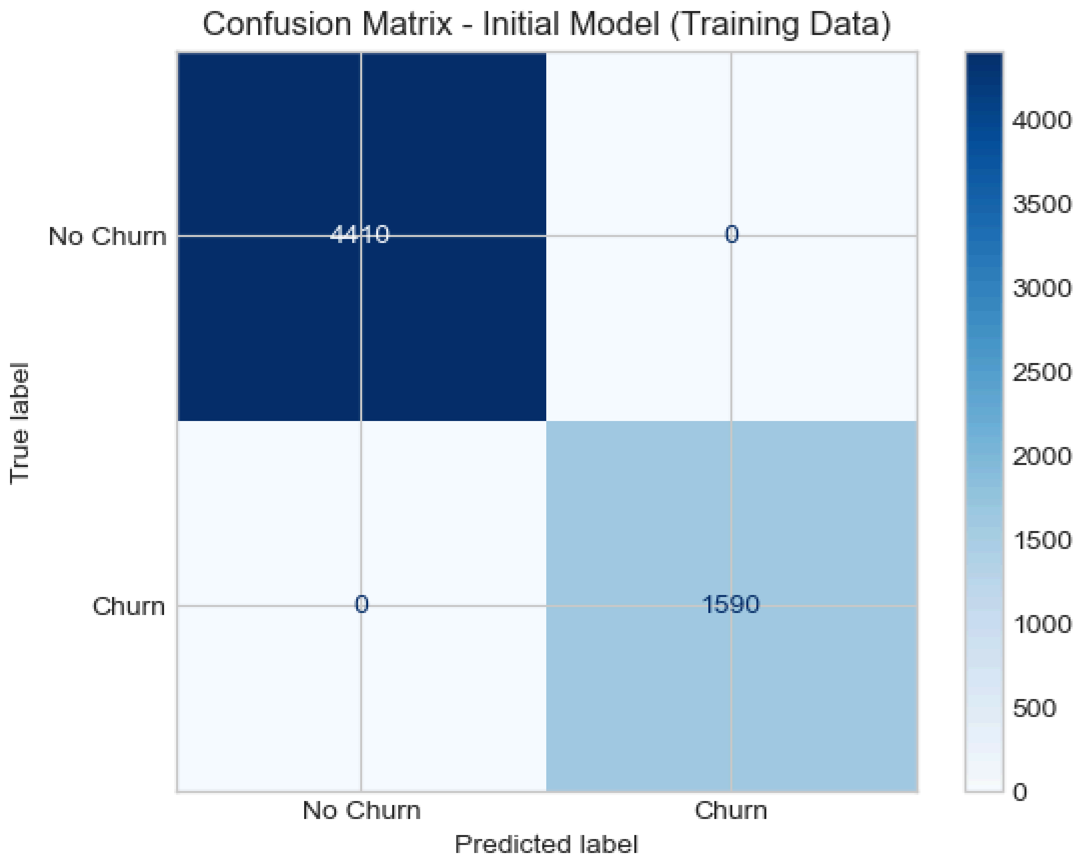
```
Initial Model Metrics (Training Data):
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1 Score: 1.0000
AUC-ROC: 1.0000
Confusion Matrix:
[[4410    0]
 [   0 1590]]
```

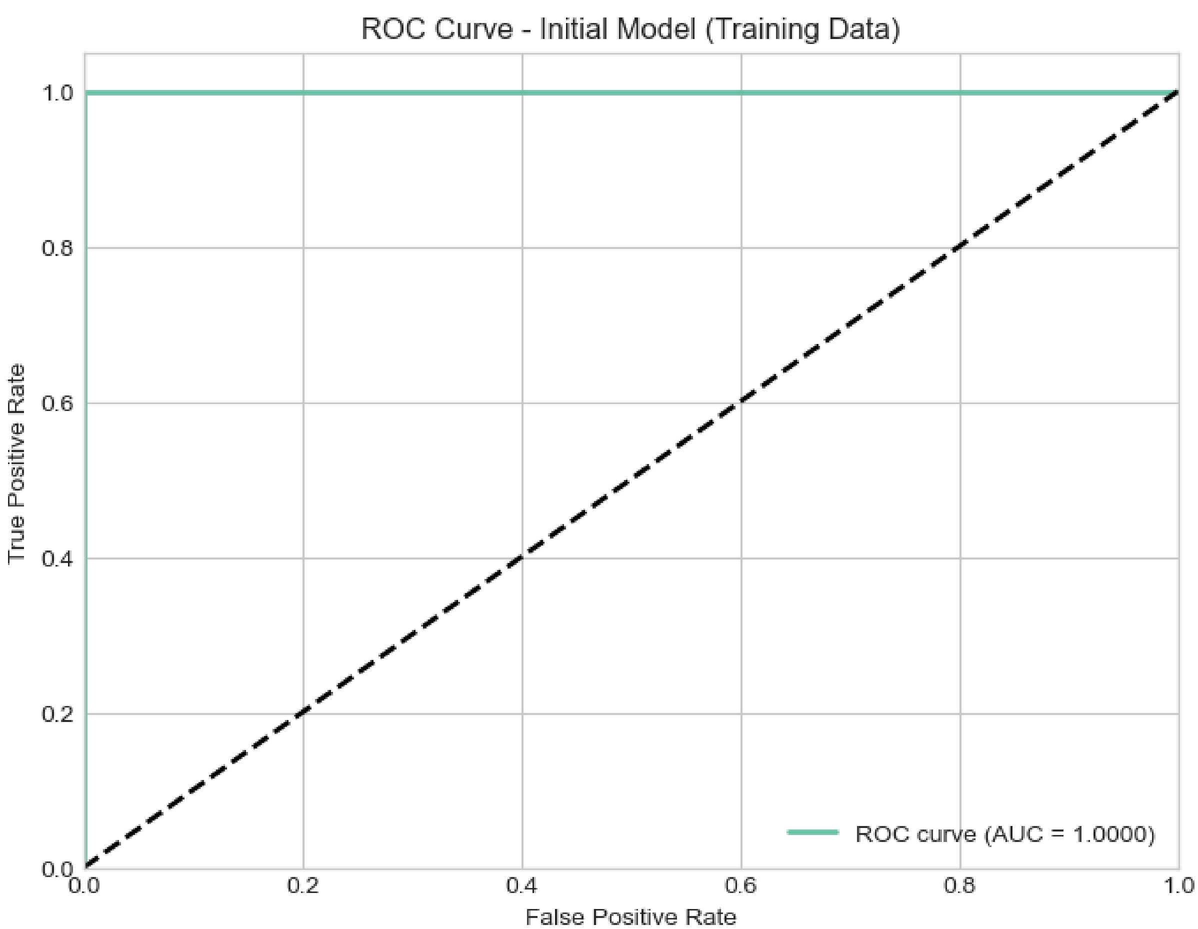
In [6]:

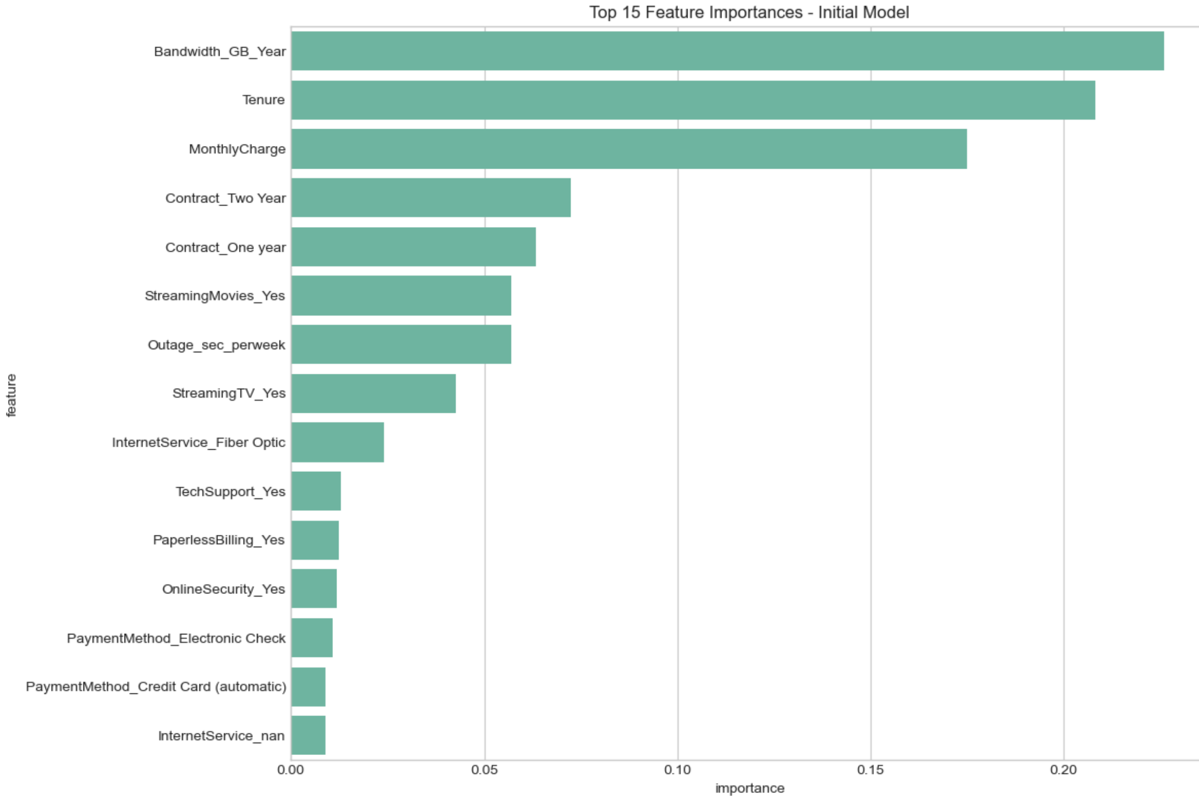
```
#visualize model results
# plot the confusion matrix
plt.figure(figsize=(8, 6))
```

```python
disp = ConfusionMatrixDisplay(confusion_matrix=train_cm, display_labels=['No Churn'
disp.plot(cmap='Blues')
plt.title('Confusion Matrix - Initial Model (Training Data)')
plt.savefig('initial_model_cm.png')
plt.show()


# plot the ROC curve
plt.figure(figsize=(8, 6))
fpr, tpr, _ = roc_curve(y_train, y_train_pred_proba)
plt.plot(fpr, tpr, lw=2, label=f'ROC curve (AUC = {train_auc_roc:.4f})')
plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Initial Model (Training Data)')
plt.legend(loc="lower right")
plt.savefig('initial_model_roc.png')
plt.show()


# get feature importances from the initial model
feature_names = (
    numerical_features +
    list(preprocessor.transformers_[1][1].get_feature_names_out(categorical_feature
)

feature_importances = pd.DataFrame({
    'feature': feature_names,
    'importance': initial_clf.feature_importances_
})

feature_importances = feature_importances.sort_values('importance', ascending=False



# plot feature importances
plt.figure(figsize=(12, 8))
sns.barplot(x='importance', y='feature', data=feature_importances.head(15))
plt.title('Top 15 Feature Importances - Initial Model')
plt.tight_layout()
plt.savefig('initial_model_feature_importance.png')
plt.show()

print("\nTop 10 Most Important Features:")
print(feature_importances.head(10))
```

```
<Figure size 800x600 with 0 Axes>
```


Confusion Matrix - Initial Model (Training Data)
True label
No Churn
Churn
4410
0
0
1590
No Churn
Churn
Predicted label
4000
3500
3000
2500
2000
1500
1000
500
0


ROC Curve - Initial Model (Training Data)
True Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
ROC curve (AUC = 1.0000)




```
Top 10 Most Important Features:
                        feature  importance
2             Bandwidth_GB_Year    0.226034
0                        Tenure    0.208050
1                 MonthlyCharge    0.175052
5             Contract_Two Year    0.072326
4             Contract_One year    0.063327
15          StreamingMovies_Yes    0.057118
3            Outage_sec_perweek    0.056940
14              StreamingTV_Yes    0.042662
6   InternetService_Fiber Optic    0.024157
13              TechSupport_Yes    0.012971
```

In [7]:
```python
# Chyperparameter tuning with grid Ssarch
# defineing hyperparameter grid for Random Forest
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

#  k-fold cross-validation
cv = KFold(n_splits=5, shuffle=True, random_state=42)

print("Starting hyperparameter tuning with Grid Search...")
print(f"Parameter grid: {param_grid}")
print("This may take some time...")

# Creating a new classifier for grid search
grid_clf = RandomForestClassifier(random_state=42)
```

```
# perform grid search
grid_search = GridSearchCV(
    grid_clf, param_grid, cv=cv,
    scoring='f1', n_jobs=-1, verbose=1
)

# process the validation data
X_val_processed = preprocessor.transform(X_val)

#fit grid search to the validation data
grid_search.fit(X_val_processed, y_val)

# getting the best hyperparameters
best_params = grid_search.best_params_
print(f"\nBest hyperparameters: {best_params}")
print(f"Best F1 score: {grid_search.best_score_:.4f}")

# save the best hyperparameters image
plt.figure(figsize=(10, 6))
plt.title('Best Hyperparameters for Random Forest', fontsize=15)
params_df = pd.DataFrame.from_dict(best_params, orient='index', columns=['Value'])
table = plt.table(
    cellText=params_df.values,
    rowLabels=params_df.index,
    colLabels=params_df.columns,
    cellLoc='center',
    loc='center'
)

table.auto_set_font_size(False)
table.set_fontsize(12)
table.scale(1, 1.5)
plt.axis('off')
plt.tight_layout()
plt.savefig('best_hyperparameters.png', bbox_inches='tight')
plt.show()
```

```
Starting hyperparameter tuning with Grid Search...
Parameter grid: {'n_estimators': [100, 200, 300], 'max_depth': [None, 10, 20, 30],
'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4]}
This may take some time...
Fitting 5 folds for each of 108 candidates, totalling 540 fits

Best hyperparameters: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split':
2, 'n_estimators': 200}
Best F1 score: 0.7232
```

Best Hyperparameters for Random Forest

	Value
max_depth	20
min_samples_leaf	1
min_samples_split	2
n_estimators	200

In [8]:
```
#train Optimized Model and Evaluate on Test Set
#train the optimized model with the best hyperparameters
optimized_clf = RandomForestClassifier(
    n_estimators=best_params['n_estimators'],
    max_depth=best_params['max_depth'],
    min_samples_split=best_params['min_samples_split'],
    min_samples_leaf=best_params['min_samples_leaf'],
    random_state=42
)


#combine training and validation sets for final model training
X_train_full = pd.concat([X_train, X_val])
y_train_full = pd.concat([y_train, y_val])

# Processing the combined data
X_train_full_processed = preprocessor.fit_transform(X_train_full)




# apply SMOTE to the combined training data
X_train_full_resampled, y_train_full_resampled = smote.fit_resample(X_train_full_pr

# train the optimiezed model
optimized_clf.fit(X_train_full_resampled, y_train_full_resampled)

# process the test data
X_test_processed = preprocessor.transform(X_test)


# make predictions on the tesst set
y_test_pred = optimized_clf.predict(X_test_processed)
```

```python
y_test_pred_proba = optimized_clf.predict_proba(X_test_processed)[:, 1]

# calculate metrics for the optimized model on test data
test_accuracy = accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred)
test_f1 = f1_score(y_test, y_test_pred)
test_auc_roc = roc_auc_score(y_test, y_test_pred_proba)
test_cm = confusion_matrix(y_test, y_test_pred)

# display metrics for the optimized model
print("\nOptimized Model Metrics (Test Data):")
print(f"Accuracy: {test_accuracy:.4f}")
print(f"Precision: {test_precision:.4f}")
print(f"Recall: {test_recall:.4f}")
print(f"F1 Score: {test_f1:.4f}")
print(f"AUC-ROC: {test_auc_roc:.4f}")
print("Confusion Matrix:")
print(test_cm)
```

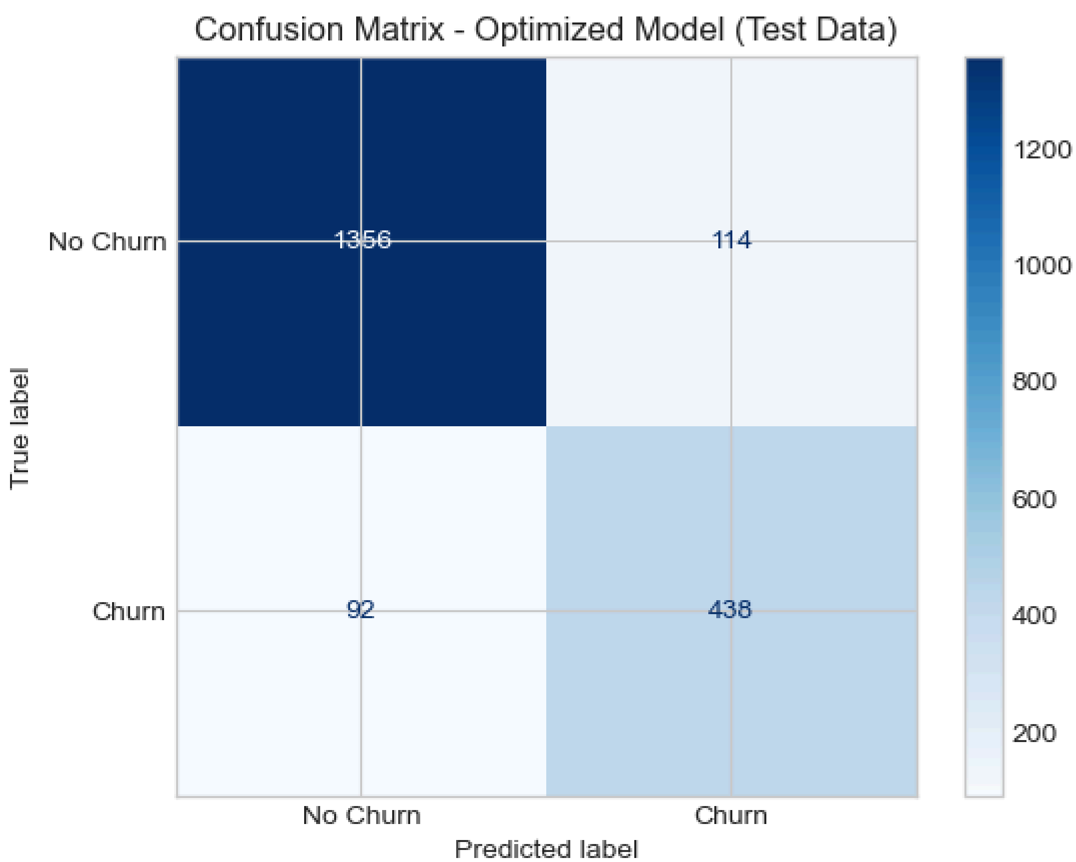
```
Optimized Model Metrics (Test Data):
Accuracy: 0.8970
Precision: 0.7935
Recall: 0.8264
F1 Score: 0.8096
AUC-ROC: 0.9517
Confusion Matrix:
[[1356  114]
 [  92  438]]
```

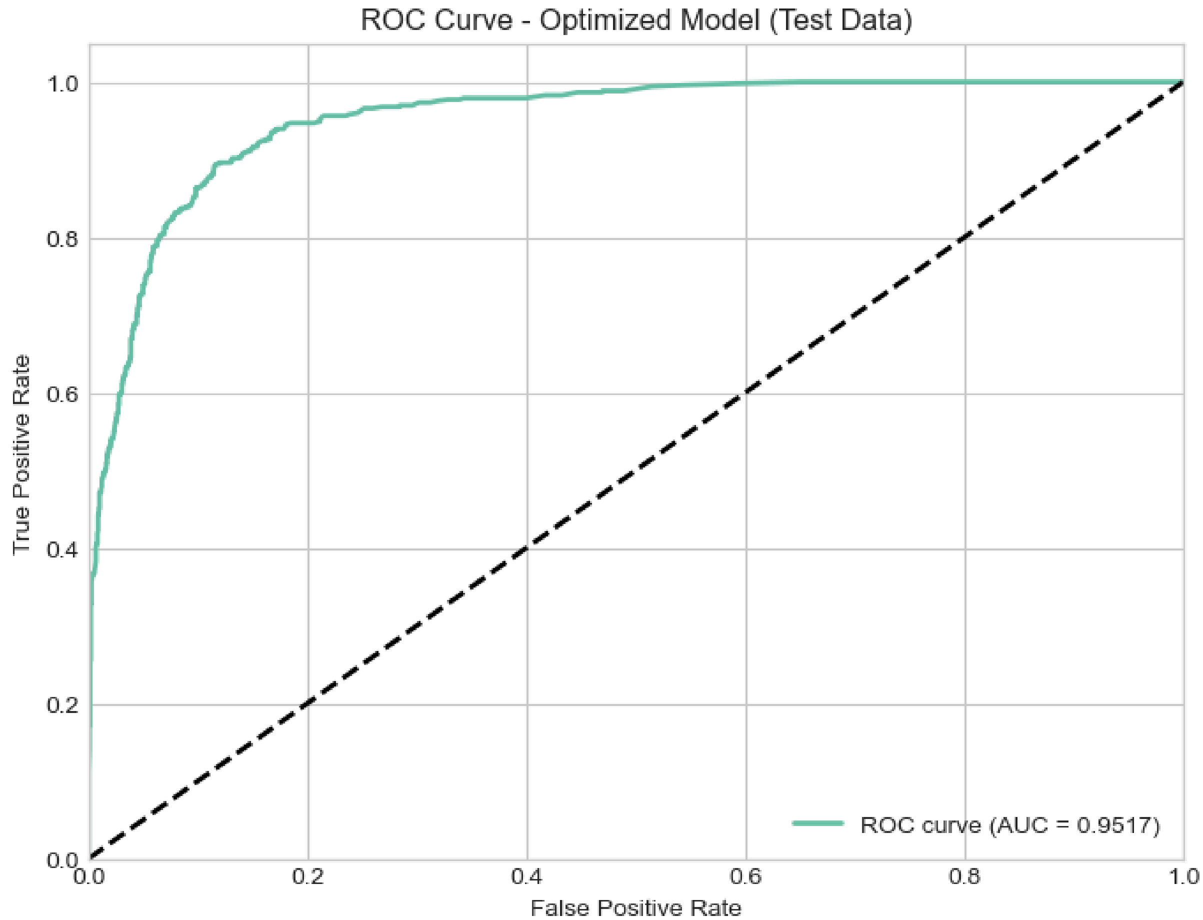
In [9]:

```python
# visualize  Model Results
# plot the confusion matrix for the optimized model
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=test_cm, display_labels=['No Churn',
disp.plot(cmap='Blues')
plt.title('Confusion Matrix - Optimized Model (Test Data)')
plt.savefig('optimized_model_cm.png')
plt.show()

# plot the ROC curve for the optimized model
plt.figure(figsize=(8, 6))
fpr, tpr, _ = roc_curve(y_test, y_test_pred_proba)
plt.plot(fpr, tpr, lw=2, label=f'ROC curve (AUC = {test_auc_roc:.4f})')
plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Optimized Model (Test Data)')
plt.legend(loc="lower right")
plt.savefig('optimized_model_roc.png')
plt.show()

# get feature importances from the optimized model
feature_importances_opt = pd.DataFrame({
    'feature': feature_names,
```

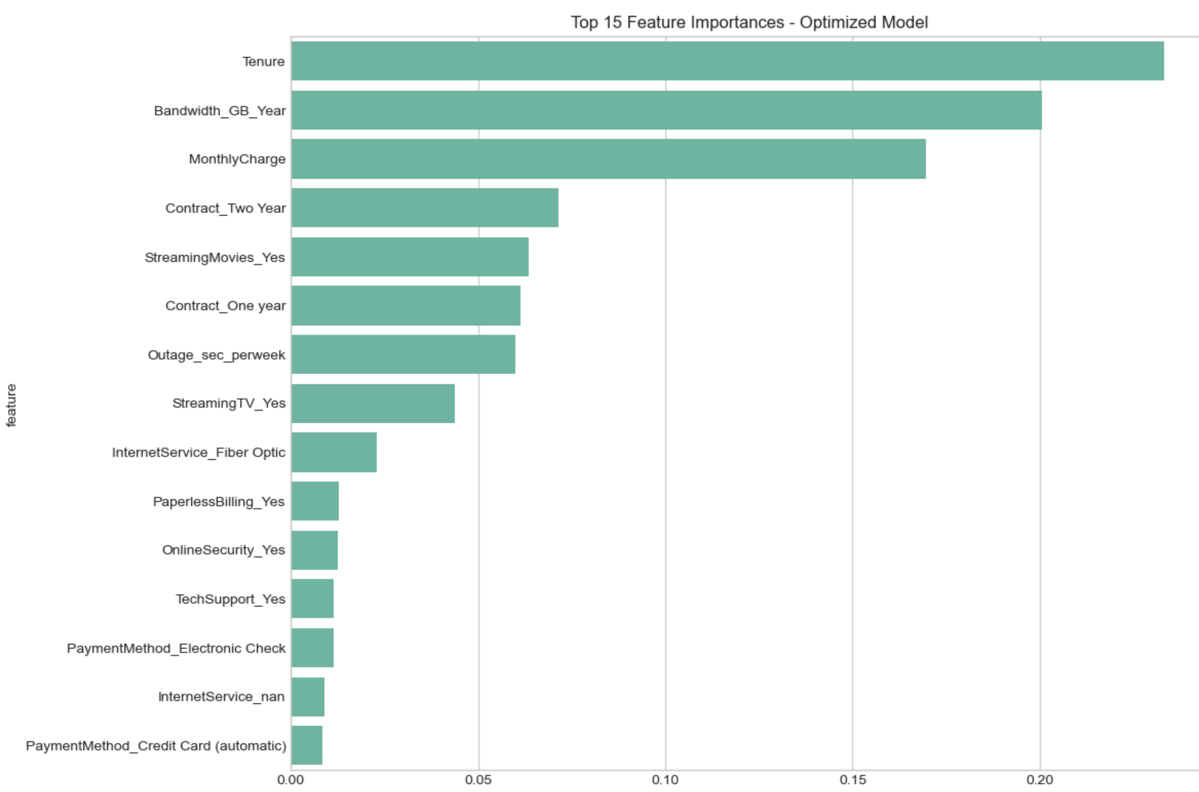
```
    'importance': optimized_clf.feature_importances_
})


feature_importances_opt = feature_importances_opt.sort_values('importance', ascendi

# plot feature importances for the optimeized model
plt.figure(figsize=(12, 8))
sns.barplot(x='importance', y='feature', data=feature_importances_opt.head(15))
plt.title('Top 15 Feature Importances - Optimized Model')
plt.tight_layout()
plt.savefig('optimized_model_feature_importance.png')
plt.show()

print("\nTop 10 Most Important Features (Optimized Model):")
print(feature_importances_opt.head(10))
```

```
<Figure size 800x600 with 0 Axes>
```





ROC Curve - Optimized Model (Test Data)
True Positive Rate
False Positive Rate
ROC curve (AUC = 0.9517)
0.0
0.2
0.4
0.6
0.8
1.0



Top 15 Feature Importances - Optimized Model
Tenure
Bandwidth_GB_Year
MonthlyCharge
Contract_Two Year
StreamingMovies_Yes
Contract_One year
Outage_sec_perweek
StreamingTV_Yes
InternetService_Fiber Optic
PaperlessBilling_Yes
OnlineSecurity_Yes
TechSupport_Yes
PaymentMethod_Electronic Check
InternetService_nan
PaymentMethod_Credit Card (automatic)
feature
importance
0.00
0.05
0.10
0.15
0.20

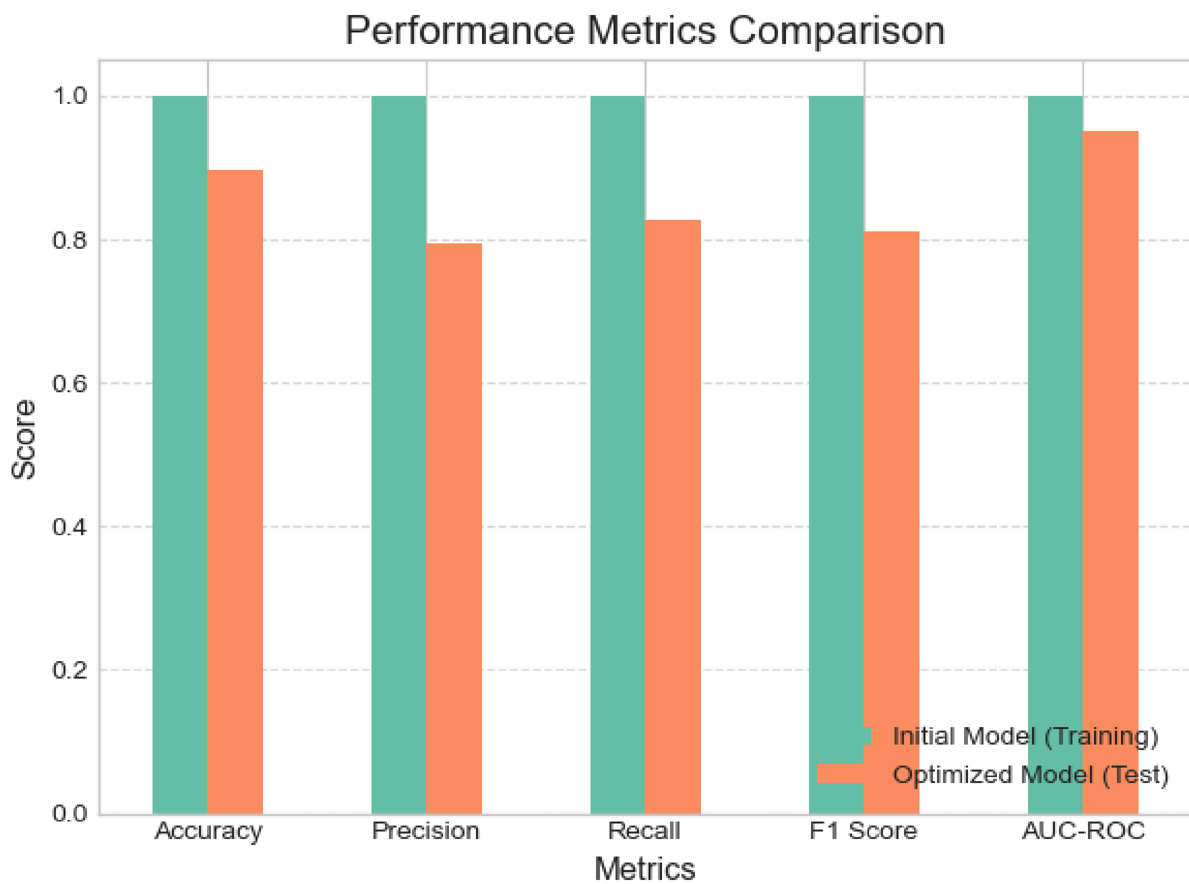
```
Top 10 Most Important Features (Optimized Model):
                       feature  importance
0                       Tenure    0.233316
2             Bandwidth_GB_Year    0.200548
1                 MonthlyCharge    0.169614
5             Contract_Two Year    0.071329
15          StreamingMovies_Yes    0.063379
4             Contract_One year    0.061320
3            Outage_sec_perweek    0.059955
14             StreamingTV_Yes    0.043694
6    InternetService_Fiber Optic    0.023050
8          PaperlessBilling_Yes    0.012760
```

In [ ]:

In [10]:

```
# compare both results
# create a comparative dataframe of metrics
model_comparison = pd.DataFrame({
    'Initial Model (Training)': [train_accuracy, train_precision, train_recall, tra
    'Optimized Model (Test)': [test_accuracy, test_precision, test_recall, test_f1,
}, index=['Accuracy', 'Precision', 'Recall', 'F1 Score', 'AUC-ROC'])

# calculate the difference
model_comparison['Difference'] = model_comparison['Optimized Model (Test)'] - model

# print comparison
print("\nModel Performance Comparison:")
print(model_comparison)

# plot the comparison
plt.figure(figsize=(12, 8))
model_comparison[['Initial Model (Training)', 'Optimized Model (Test)']].plot(kind=
plt.title('Performance Metrics Comparison', fontsize=15)
plt.ylabel('Score', fontsize=12)
plt.xlabel('Metrics', fontsize=12)
plt.xticks(rotation=0)
plt.legend(loc='lower right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.savefig('model_comparison.png')
plt.show()
```

```
Model Performance Comparison:
           Initial Model (Training)  Optimized Model (Test)  Difference
Accuracy                        1.0                0.897000   -0.103000
Precision                       1.0                0.793478   -0.206522
Recall                          1.0                0.826415   -0.173585
F1 Score                        1.0                0.809612   -0.190388
AUC-ROC                         1.0                0.951702   -0.048298
<Figure size 1200x800 with 0 Axes>
```



In [12]:

```
# save the preprocessing pipeline and the optimized model
joblib.dump(preprocessor, 'churn_preprocessor.joblib')
joblib.dump(optimized_clf, 'churn_model.joblib')

print("\nSaved model files:")
print("- churn_preprocessor.joblib - Preprocessing pipeline")
print("- churn_model.joblib - Optimized Random Forest model")


# sample new customer data
new_customers = pd.DataFrame({
    'Tenure': [7.5, 65.2, 24.3],
    'MonthlyCharge': [180.5, 155.2, 190.8],
    'Bandwidth_GB_Year': [2500, 3800, 4200],
    'Outage_sec_perweek': [12.5, 5.2, 9.7],
    'Contract': ['Month-to-month', 'Two Year', 'One year'],
    'InternetService': ['Fiber Optic', 'DSL', 'Fiber Optic'],
    'PaperlessBilling': ['Yes', 'No', 'Yes'],
    'PaymentMethod': ['Electronic Check', 'Credit Card (automatic)', 'Bank Transfer
    'OnlineSecurity': ['No', 'Yes', 'No'],
    'TechSupport': ['No', 'Yes', 'Yes'],
    'StreamingTV': ['Yes', 'No', 'Yes'],
    'StreamingMovies': ['Yes', 'No', 'Yes']
})

# process the new customer data
new_customers_processed = preprocessor.transform(new_customers)
```

```python
# predict churn probability
churn_probabilities = optimized_clf.predict_proba(new_customers_processed)

# extract the probability of churn (class 1)
churn_prob = churn_probabilities[:, 1]

# setup results dataframe
results = pd.DataFrame({
    'Customer': [1, 2, 3],
    'Tenure': new_customers['Tenure'],
    'Contract': new_customers['Contract'],
    'Monthly Charge': new_customers['MonthlyCharge'],
    'Churn Probability': churn_prob,
    'Churn Risk': ['High' if p > 0.5 else 'Low' for p in churn_prob]
})

print("\nChurn Prediction for New Customers:")
print(results)
```

```
Saved model files:
- churn_preprocessor.joblib - Preprocessing pipeline
- churn_model.joblib - Optimized Random Forest model

Churn Prediction for New Customers:
   Customer  Tenure        Contract  Monthly Charge  Churn Probability  \
0         1     7.5  Month-to-month           180.5            0.75144
1         2    65.2        Two Year           155.2            0.02000
2         3    24.3        One year           190.8            0.18000

  Churn Risk
0       High
1        Low
2        Low
```

In [ ]:

```python

```

In [ ]:

```python

```