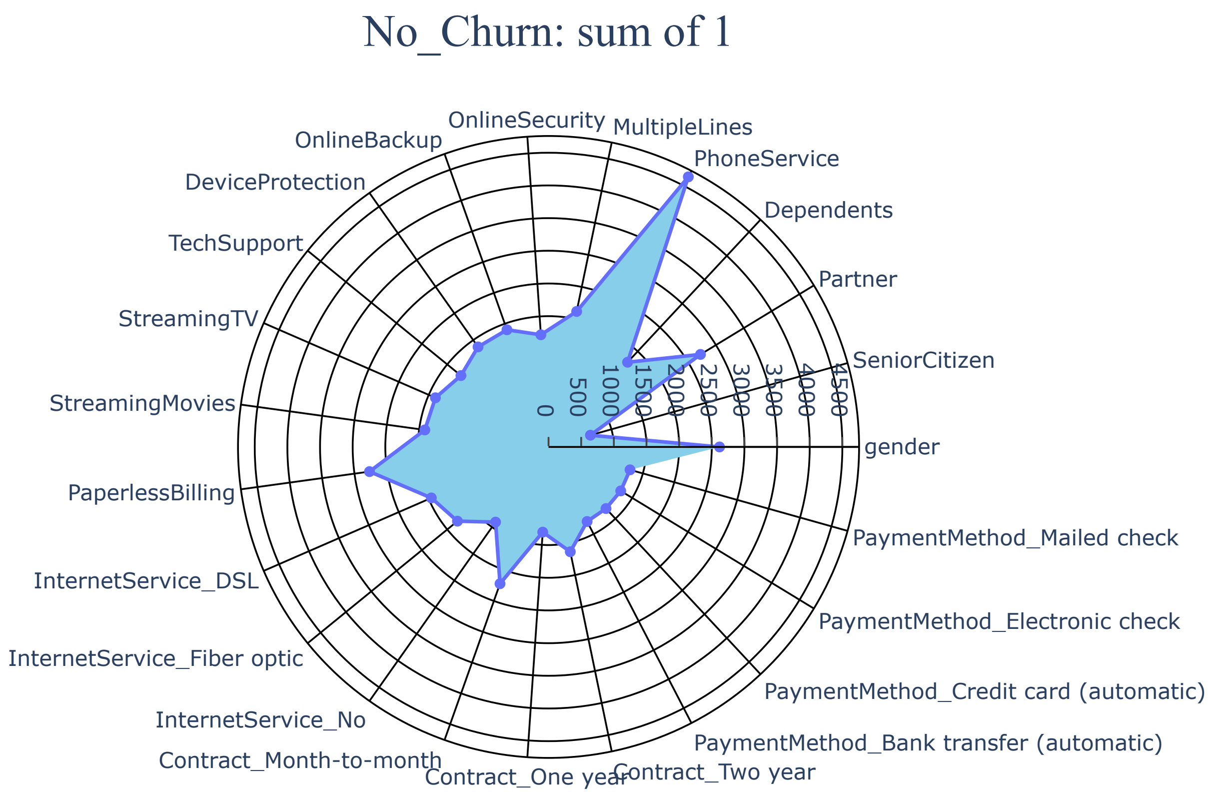
In [33]:
```
categories = bifeat_no_churn_radar.keys().tolist()
values = bifeat_no_churn_radar.values.tolist()
radar_chart(categories, values, 'No_Churn: sum of 1')
```



2. Model Training Using Grid Search to Find Optimal Parameters and Result Evaluation

In [34]:
```
target_col = [LABEL]
df_data = churn_feat_space_and_target
X = df_data.drop(labels = target_col, axis = 1) # pandas dataframe
y = np.where(df_data[LABEL] == LABEL_ONE, 1, 0) # numpy array
```

In [35]:
```python
def run_ml(X, y, algorithm, algorithm_name, n_folds = None, feature_importance_attr = None):
    """
    Run the machine learning model and show the results.

    Parameters:
    X: input features, dataframe
    y: input labels, numpy array
    algorithm: an object of the model class, e.g., sklearn.linear_model.LogisticRegression
    algorithm_name: name of the model
    n_folds: if not None, use k-fold cross validation, and n_folds is the number of folds.
    feature_importance_attr: if not None, it is the attribute name of the model class to get the feature importance

    Returns:
    algorithm: the trained model
    performance: the model performance dataframe, with columns of 'Model', 'Test Data Size', 'Accurary', 'Precision (1)',
                 'Recall (1)', 'F1 Score', 'AUC'
    df_feat_importance: the dataframe showing the importance of all features
    """

    # 1. Model training

    if n_folds is not None:
        y_test_pred = y.copy()
        y_test_prob = np.zeros(y.shape)
        # Construct a kfolds object
        kf = KFold(n_splits = n_folds, shuffle = True)
        # Iterate through folds
        for train_index, test_index in kf.split(X):
            X_train, X_test = X.iloc[train_index], X.iloc[test_index]
            y_train = y[train_index]
            # Fit the model according to the given training data.
            algorithm.fit(X_train, y_train)
            # Predict class labels for samples in X.
            # Note that the threshold is 0.5
            y_test_pred[test_index] = algorithm.predict(X_test)
            # Return estimates for all classes.
            # predict() will give 0 or 1 as output; predict_proba() will give the probability of 0 (in column 0) and 1 (in column 1).
            y_test_prob[test_index] = algorithm.predict_proba(X_test)[:, 1]
        y_test = y # all data is used as test set
        X_test = X
    else:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 0)
        # Fit the model according to the given training data.
        algorithm.fit(X_train, y_train)
        # Predict class labels for samples in X.
        # Note that the threshold is 0.5
```

```
        y_test_pred = algorithm.predict(X_test)
        # Return estimates for all classes.
        # predict() will give 0 or 1 as output; predict_proba() will give the probability of 0 (in column 0) and 1 (in column 1).
        y_test_prob = algorithm.predict_proba(X_test)[:, 1]

    # 2. Show the results

    if n_folds is not None:
        print('\nSummary of ' + str(algorithm_name) + ' Model with ' + str(n_folds) + '-Fold Cross Validation:')
    else:
        print('\nSummary of ' + str(algorithm_name) + ' Model:')

    # 2.1 Print the model details

    print('\nAlgorithm:\n')
    print(algorithm)

    # 2.2 Print the test accuracy

    print('\nTest Accuracy:', accuracy_score(y_true = y_test, y_pred = y_test_pred))

    # 2.3 Print the classification report

    print('\nClassification Report:\n', classification_report(y_true = y_test, y_pred = y_test_pred))

    # 2.4 Print and plot the receiver operating characteristic (ROC)

    auc = roc_auc_score(y_true = y_test, y_score = y_test_prob)
    fpr, tpr, thresholds = roc_curve(y_true = y_test, y_score = y_test_prob)
    trace_roc = go.Scatter(x = fpr, y = tpr)
    trace_roc_diag = go.Scatter(x =[0, 1], y = [0, 1], line = dict(dash = "dash"))

    # 2.5 Calculate and plot the confusion matrix

    conf_matrix = confusion_matrix(y_true = y_test, y_pred = y_test_pred)
    trace_conf_matrix = go.Heatmap(z = conf_matrix / len(y_test), x = ['Predicted: ' + str(algorithm.classes_[0]),
                                                                        'Predicted: ' + str(algorithm.classes_[1])],
                                   y = ['True: ' + str(algorithm.classes_[0]), 'True: ' + str(algorithm.classes_[1])])

    # 2.6 Calculate and plot accurary, recall and precision according to different thresholds

    if thresholds[0] > 1: # the library code adds a large at index 0 for some specific reason; we don't need it.
        thresholds = thresholds[1:]
    accuracy = np.zeros(thresholds.shape)
    precision = np.zeros(thresholds.shape)
    recall = np.zeros(thresholds.shape)
```

```
    for i in range(len(thresholds)):
        y_test_pred_i = (algorithm.predict_proba(X_test)[:, 1] >= thresholds[i])
        cm_i = confusion_matrix(y_true = y_test, y_pred = y_test_pred_i)
        tp = cm_i[1][1]
        tn = cm_i[0][0]
        fp = cm_i[0][1]
        fn = cm_i[1][0]
        accuracy[i] = (tp + tn) / (tp + fp + fn + tn + 0.0)
        recall[i] = tp / (tp + fn + 0.0)
        precision[i] = tp / (tp + fp + 0.0)
    trace_acc = go.Scatter(x = thresholds, y = accuracy, name = 'Accuracy')
    trace_recall = go.Scatter(x = thresholds, y = recall, name = 'Recall (Positive)')
    trace_precision = go.Scatter(x = thresholds, y = precision, name = 'Precision (Positive)')

    # 2.7 Calculate and plot feature importance

    if feature_importance_attr is not None:
        feature_importance = np.squeeze(getattr(algorithm, feature_importance_attr))
        dict_feat_importance = {'cols' : X.columns, 'importance' : feature_importance}
        df_feat_importance = pd.DataFrame(data = dict_feat_importance)
        df_feat_importance = df_feat_importance.iloc[df_feat_importance.importance.abs().argsort()[::-1]] # [::-1] to get decreasing
order
        trace_feat_importance = go.Bar(x = df_feat_importance['cols'], y = df_feat_importance['importance']) # bar plot

    # 3. Show the figures

    fig1 = py.subplots.make_subplots(rows = 1, cols = 2, subplot_titles = ('ROC', 'Confusion Matrix'),
                                     print_grid = False) # Remove the "This is the format of your plot grid...�?
    fig1.add_trace(trace_roc, row = 1, col = 1)
    fig1.add_trace(trace_roc_diag, row = 1, col = 1)
    fig1.add_trace(trace_conf_matrix, row = 1, col = 2)
    if n_folds is not None:
        title = str(algorithm_name) + ' (' + str(n_folds) + '-Fold Cross Validation)'
    else:
        title = str(algorithm_name)
    fig1['layout'].update(title = dict(text = '<b>Performance of ' + title + ' Model</b>', x = 0.5,
                                       font = dict(family = 'Times New Roman', size = 25)),
                          showlegend = False,
                          annotations = [dict(x = 0.3, y = 0.1, font = dict(size = 25, color = 'white'),
                                              text = 'AUC: ' + str(round(auc, 4)))],
                          plot_bgcolor = 'black')
    fig1["layout"]["xaxis1"].update(dict(title = "False Positive Rate", gridcolor = 'grey'))
    fig1["layout"]["yaxis1"].update(dict(title = "True Positive Rate", gridcolor = 'grey'))
    pyo.iplot(fig1)

    fig2 = go.Figure()
```

```
    fig2.add_trace(trace_acc)
    fig2.add_trace(trace_recall)
    fig2.add_trace(trace_precision)
    fig2['layout'].update(title = dict(text = '<b>Threshold Plot</b>', x = 0.5), titlefont = dict(size = 20), showlegend = True,
                          plot_bgcolor = "rgb(200, 200, 200)")
    fig2["layout"]["xaxis"].update(dict(title = "Threshold"))
    fig2["layout"]["yaxis"].update(dict(title = "Score"))
    pyo.iplot(fig2)

    if feature_importance_attr is not None:
        fig3 = go.Figure()
        fig3.add_trace(trace_feat_importance)
        fig3['layout'].update(title = dict(text = '<b>Feature Importance</b>', x = 0.5), titlefont = dict(size = 20),
                              plot_bgcolor = "rgb(223, 237, 245)")
        fig3["layout"]["xaxis"].update(dict(tickangle = 90, tickfont = dict(size = 10)))
        pyo.iplot(fig3)

    # contruct the performance dataframe
    test_size = X_test.shape[0]
    tp = conf_matrix[1][1]
    tn = conf_matrix[0][0]
    fp = conf_matrix[0][1]
    fn = conf_matrix[1][0]
    acc = (tp + tn) / (tp + fp + fn + tn + 0.0) # accuracy
    rec = tp / (tp + fn + 0.0) # recall
    prec = tp / (tp + fp + 0.0) # precision
    f1 = 2 * (prec * rec) / (prec + rec)
    performance = {'Model' : [algorithm_name + '<br>(N_Folds: ' + str(n_folds) + ')'], 'Test Data Size' : [test_size],
                   'Accuracy' : [acc], 'Precision (1)' : [prec], 'Recall (1)' : [rec], 'F1 Score' : [f1], 'AUC' : [auc]}
    performance = pd.DataFrame(data = performance)

    # construct the feature importance dataframe
    feat_importance = None
    if feature_importance_attr is not None:
        feat_importance = {'Features' : X.columns, algorithm_name + '<br>(N_Folds: ' + str(n_folds) + ')' : feature_importance}
        feat_importance = pd.DataFrame(data = feat_importance)
        feat_importance = feat_importance.set_index(keys = 'Features')

    return algorithm, performance, feat_importance
```

```
In [36]: def grid_search(X, y, algorithm, parameters):
    """
    Use grid search to find the optimal paramters for the model
    Parameters:
    X: input features, dataframe
    y: input labels, numpy array
    algorithm: an object of the model class, e.g., sklearn.linear_model.LogisticRegression
    paramters: dict or list of dictionaries. Dictionary with parameters names (string) as keys and lists of parameter settings to
               try as values, or a list of such dictionaries, in which case the grids spanned by each dictionary in the list are
               explored. This enables searching over any sequence of parameter settings.
    Returns:
    dictionary of best paramters.
    """

    gs = GridSearchCV(estimator = algorithm, param_grid = parameters, cv = 5, verbose = 1, refit = False)
    gs.fit(X, y)
    print('\nBest Score:', gs.best_score_)
    print('\nBest Parameter Set:', gs.best_params_)
    return gs.best_params_
```

2.1 Logistic Regression

```
In [37]: logis = LogisticRegression(max_iter = 100, multi_class = 'ovr', n_jobs = 1, solver = 'liblinear', verbose = 0, warm_start = False)
```

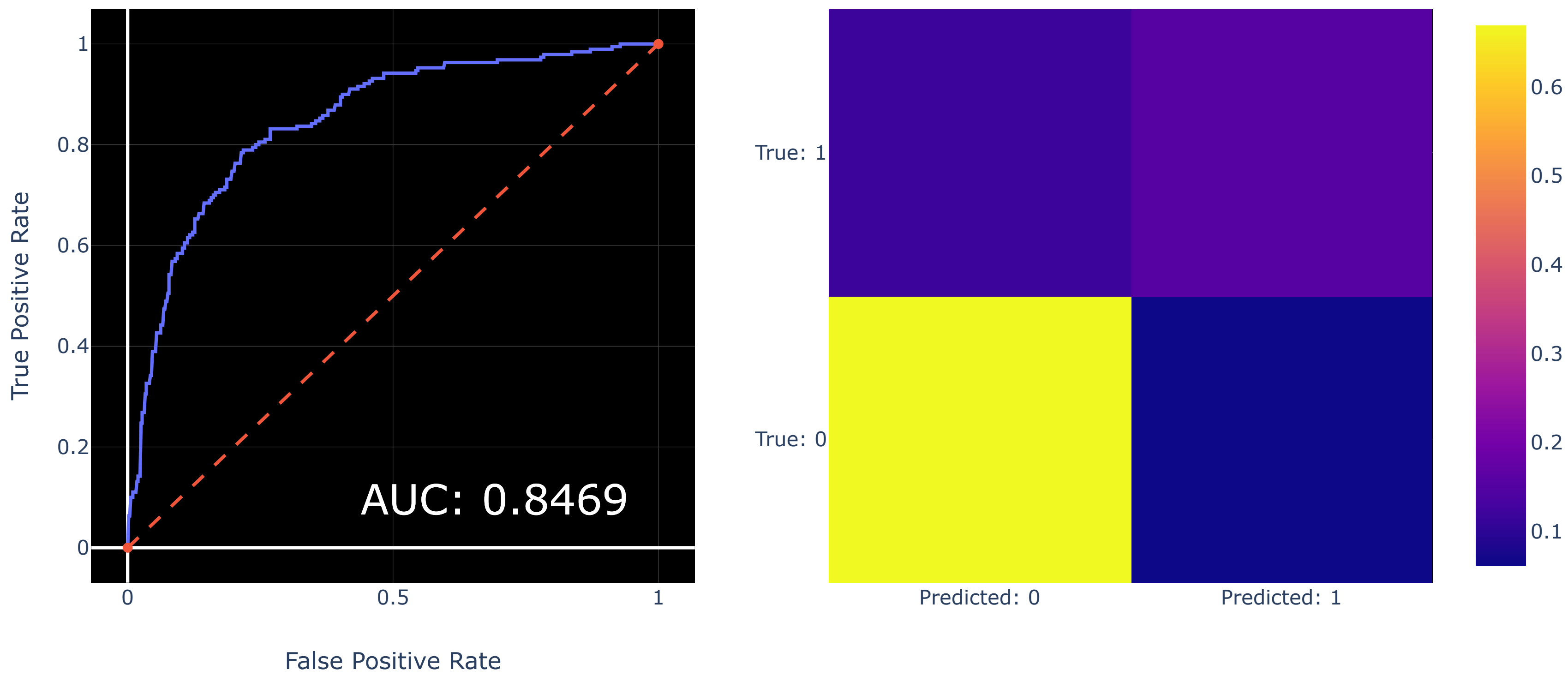
```
In [38]: parameters = {
    'penalty' : ['l1', 'l2'],
    'C' : 0.001 * 10 ** (np.arange(0, 1.02, 0.02) * 4) # 0.001 to 10
}
logis_params = grid_search(X, y, logis, parameters)
```

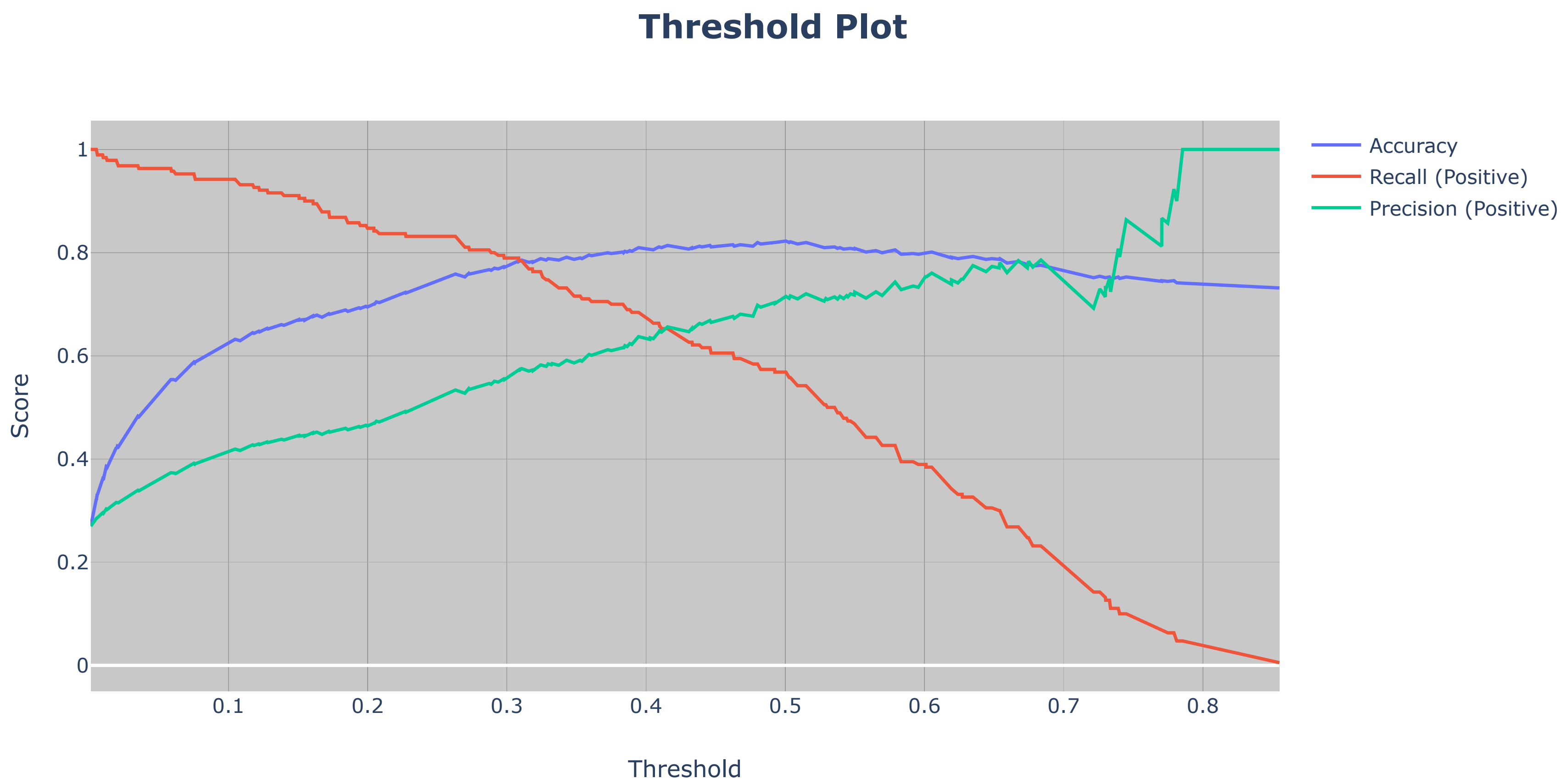
```
Fitting 5 folds for each of 102 candidates, totalling 510 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

Best Score: 0.8041808873720137


Best Parameter Set: {'C': 4.786300923226385, 'penalty': 'l2'}

[Parallel(n_jobs=1)]: Done 510 out of 510 | elapsed:   39.4s finished
```

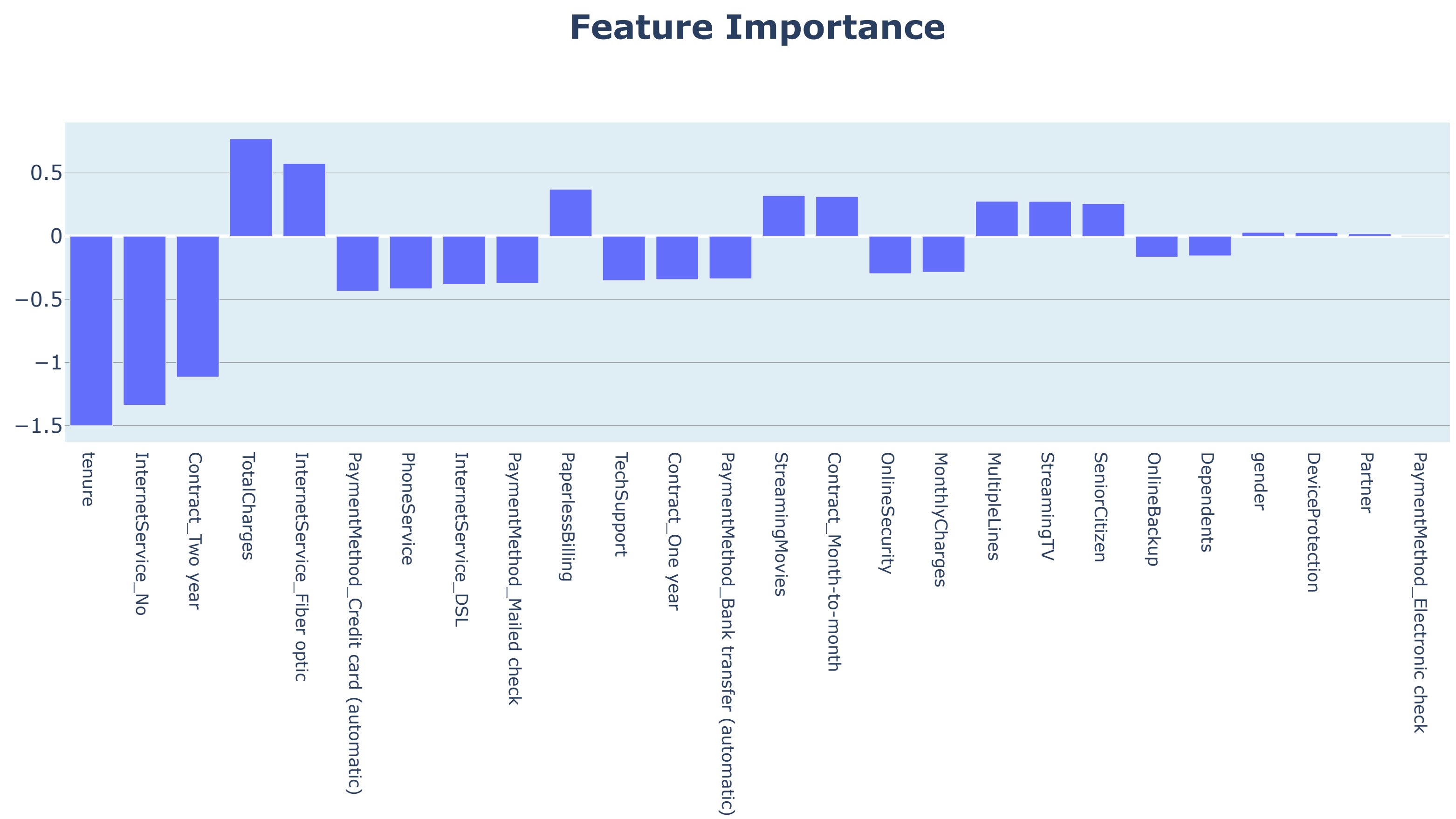
In [39]:
```
logis = LogisticRegression(max_iter = 100, multi_class = 'ovr', n_jobs = 1, C = logis_params['C'],
                            penalty = logis_params['penalty'], solver = 'liblinear', verbose = 0, warm_start = False)
_, performance_lr, feat_importance_lr = run_ml(X, y, logis, 'Logistic Regression', n_folds = None, feature_importance_attr = 'coef_')
```

```
Summary of Logistic Regression Model:

Algorithm:

LogisticRegression(C=4.786300923226385, class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='ovr', n_jobs=1, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)

Test Accuracy: 0.8224431818181818

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.92      0.88       514
           1       0.72      0.57      0.63       190

    accuracy                           0.82       704
   macro avg       0.78      0.74      0.76       704
weighted avg       0.81      0.82      0.82       704
```

Performance of Logistic Regression Model




Threshold Plot
Score
Threshold
Accuracy
Recall (Positive)
Precision (Positive)
0
0.2
0.4
0.6
0.8
1
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8



2.2 Logistic Regression with K-Fold Cross Validation

In [40]:
```
_, performance_lr_kfolds, feat_importance_lr_kfolds = run_ml(X, y, logis, 'Logistic Regression', n_folds = 5,
                                                             feature_importance_attr = 'coef_')
```

Summary of Logistic Regression Model with 5-Fold Cross Validation:

Algorithm:

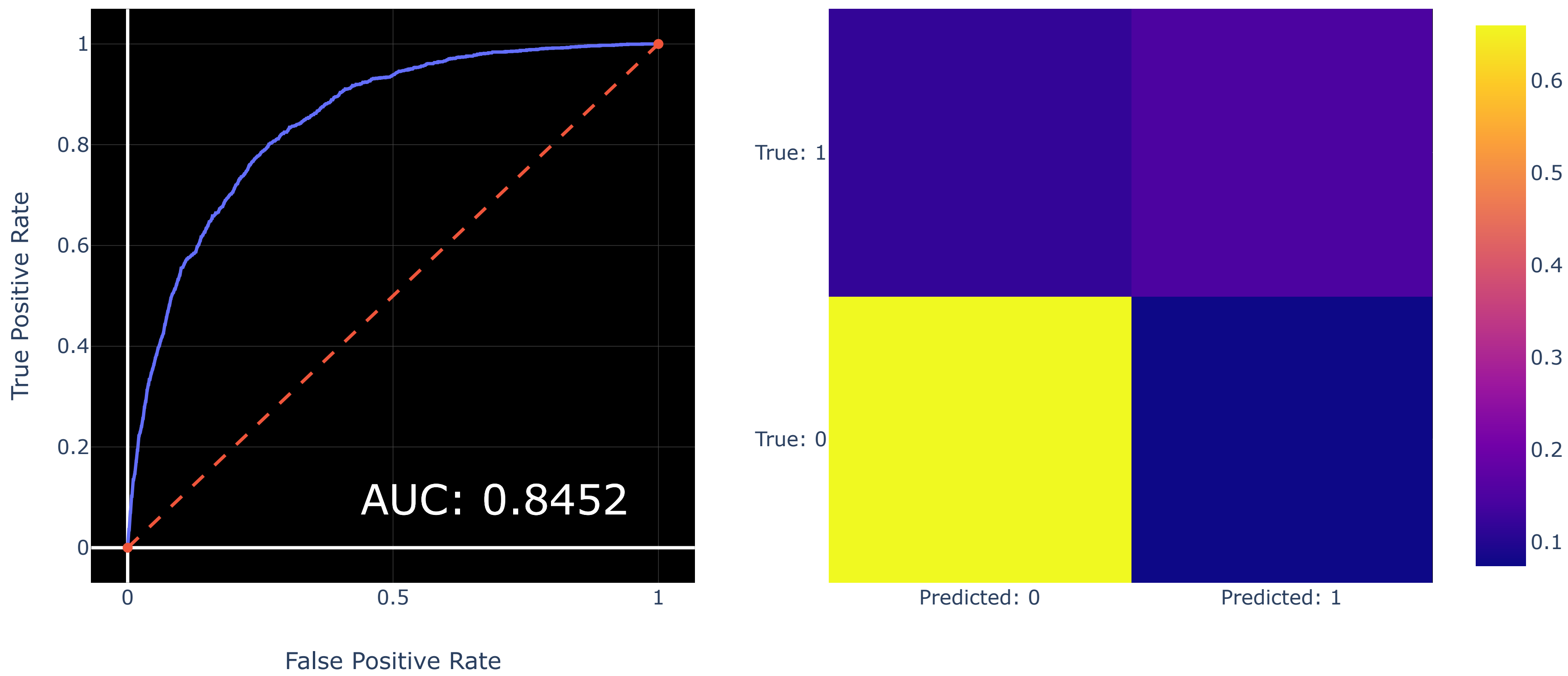
```
LogisticRegression(C=4.786300923226385, class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='ovr', n_jobs=1, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

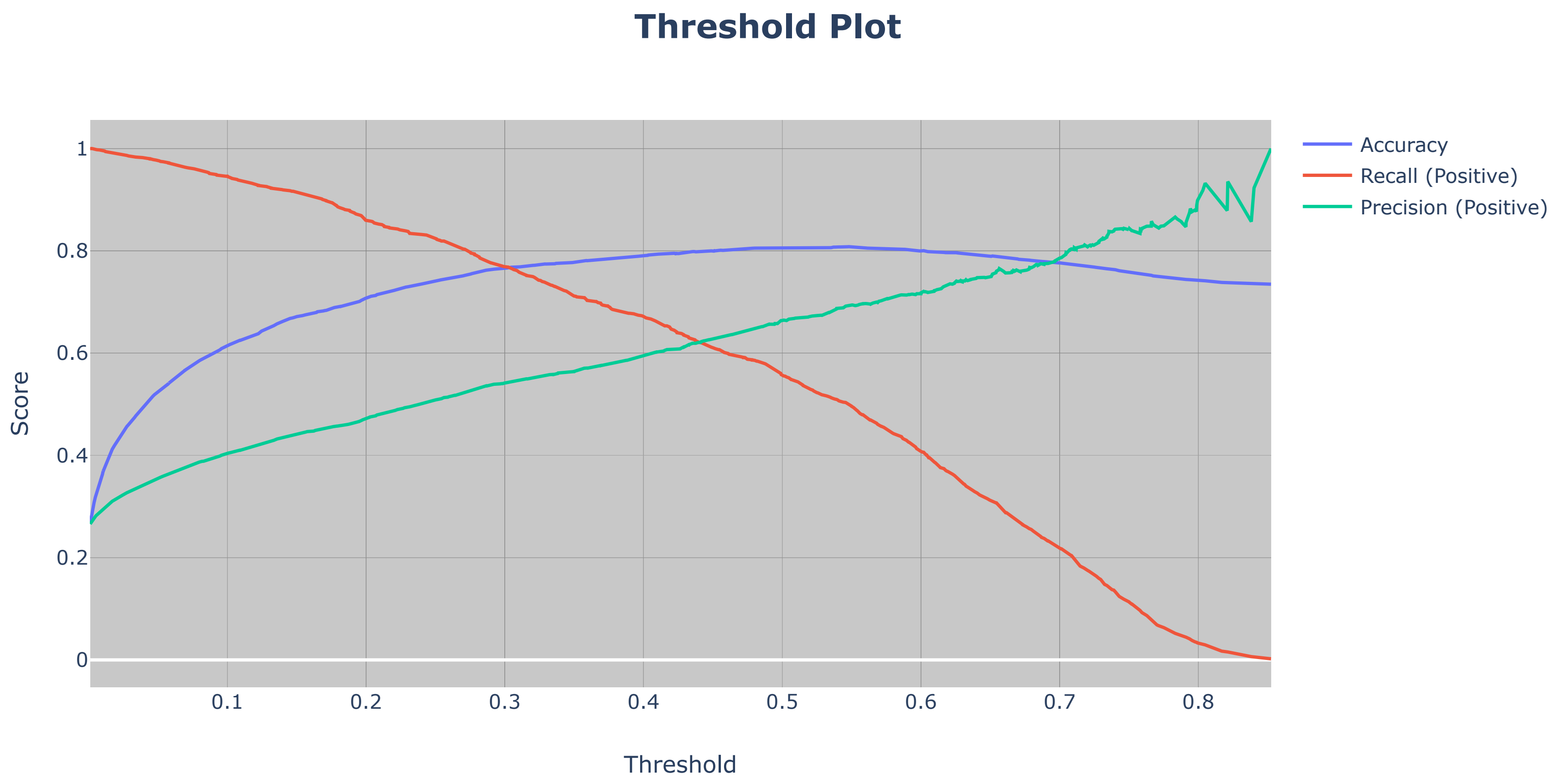
Test Accuracy: 0.8075938566552902

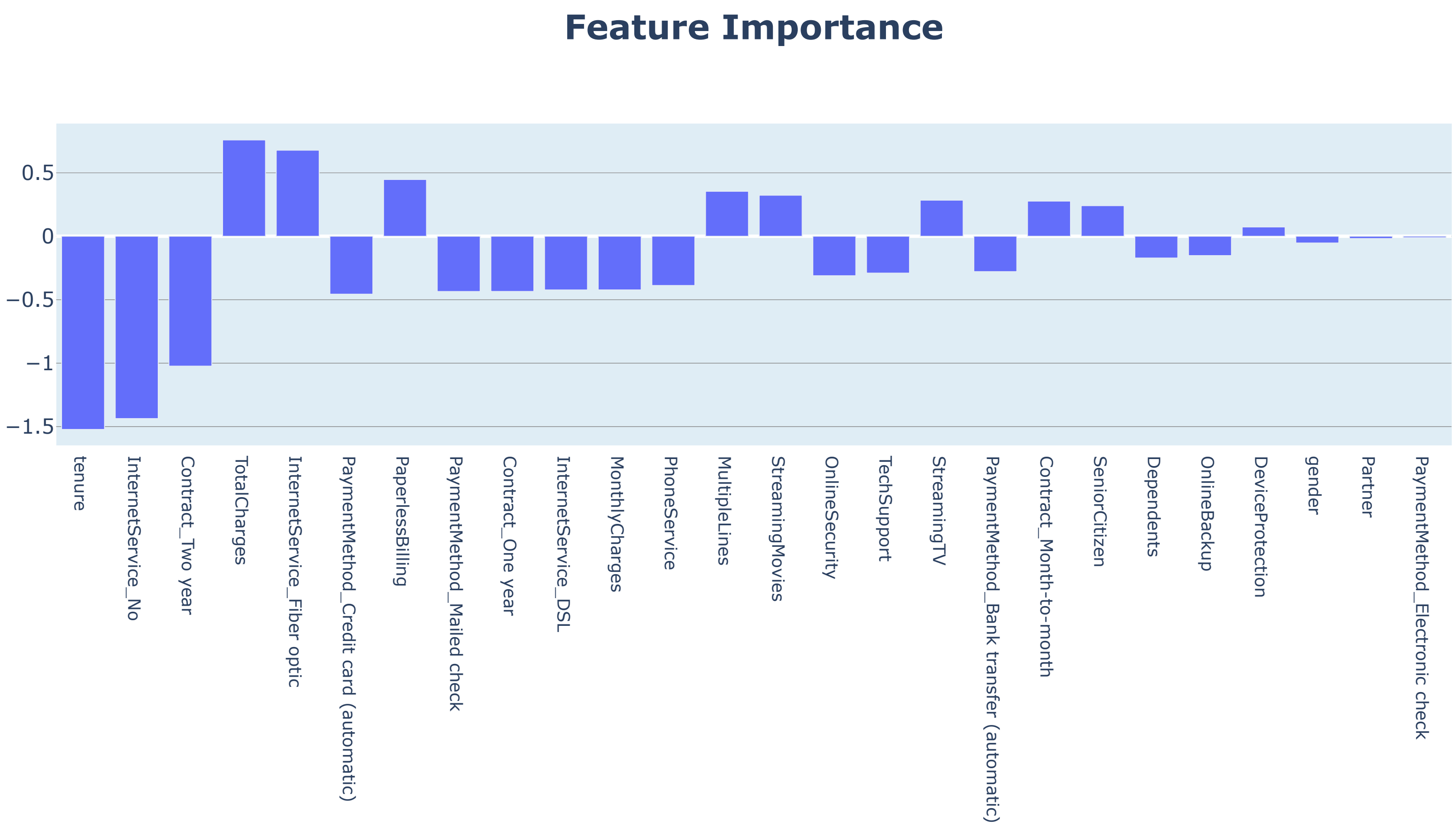
Classification Report:

	precision	recall	f1-score	support
0	0.85	0.90	0.87	5163
1	0.67	0.56	0.61	1869
accuracy			0.81	7032
macro avg	0.76	0.73	0.74	7032
weighted avg	0.80	0.81	0.80	7032

Performance of Logistic Regression (5-Fold Cross Validation) Model




Threshold Plot
Score
Threshold
1
0.8
0.6
0.4
0.2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Accuracy
Recall (Positive)
Precision (Positive)



2.3 Logistic Regression with Recursive Feature Elimination (RFE)

The goal of RFE is to select features by recursively considering smaller and smaller sets of features.

In [41]:
```
n_features_to_select = 10
logis_rfe = LogisticRegression(max_iter = 100, penalty = 'l2', C = 1, multi_class = 'ovr', n_jobs = 1, solver = 'liblinear',
                               verbose = 0, warm_start = False)
```

In [42]:
```python
rfe_logis = RFE(logis_rfe, n_features_to_select = n_features_to_select)

rfe_logis.fit(X, y)

print("Logistic Regression RFE Result:")
for k, v in sorted(zip(rfe_logis.ranking_, X.columns)):
    print(k, ':', v)

# get the selected columns
cols_selected = np.array(X.columns)[rfe_logis.support_].tolist()
```

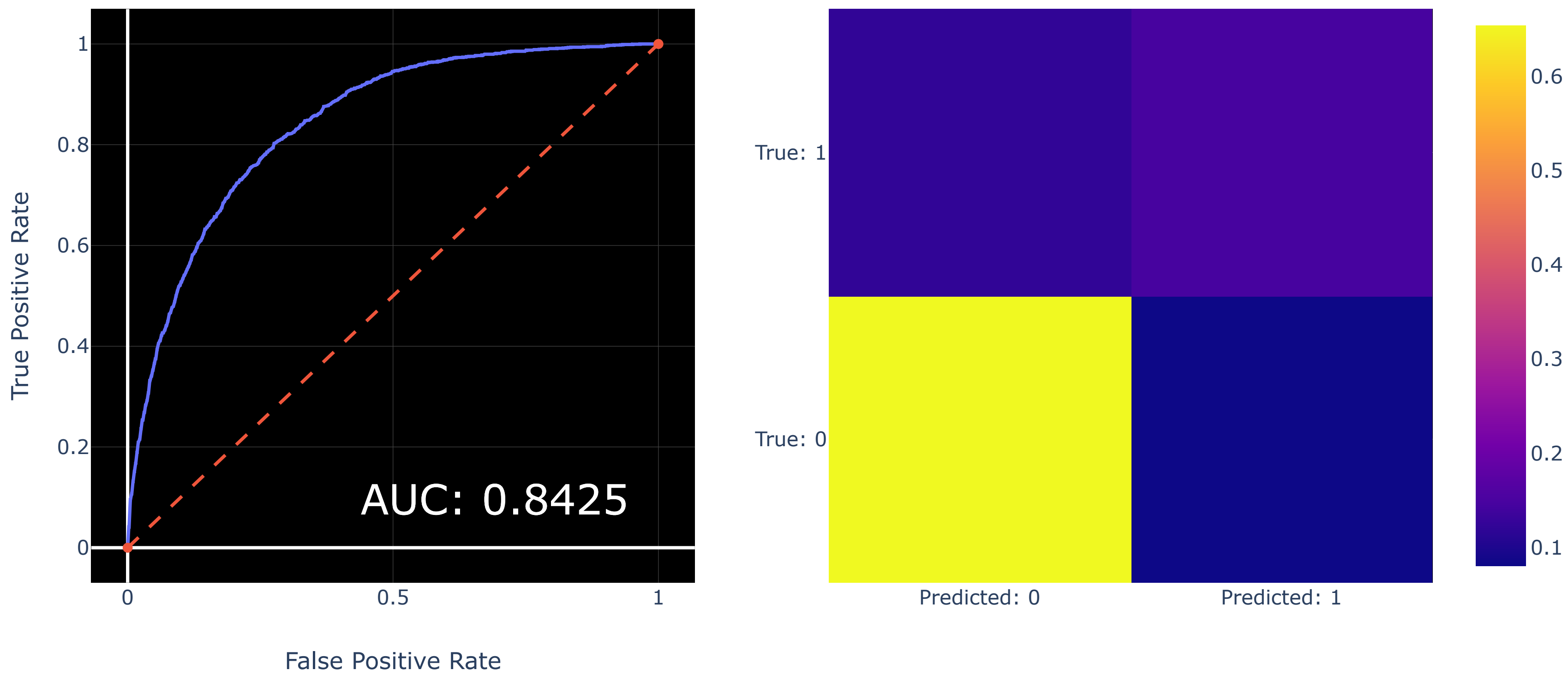
```
Logistic Regression RFE Result:
1 : Contract_Month-to-month
1 : Contract_Two year
1 : InternetService_Fiber optic
1 : InternetService_No
1 : OnlineSecurity
1 : PaperlessBilling
1 : PhoneService
1 : TechSupport
1 : TotalCharges
1 : tenure
2 : PaymentMethod_Mailed check
3 : PaymentMethod_Credit card (automatic)
4 : PaymentMethod_Bank transfer (automatic)
5 : InternetService_DSL
6 : MultipleLines
7 : Contract_One year
8 : StreamingMovies
9 : SeniorCitizen
10 : StreamingTV
11 : OnlineBackup
12 : Dependents
13 : MonthlyCharges
14 : DeviceProtection
15 : PaymentMethod_Electronic check
16 : gender
17 : Partner
```

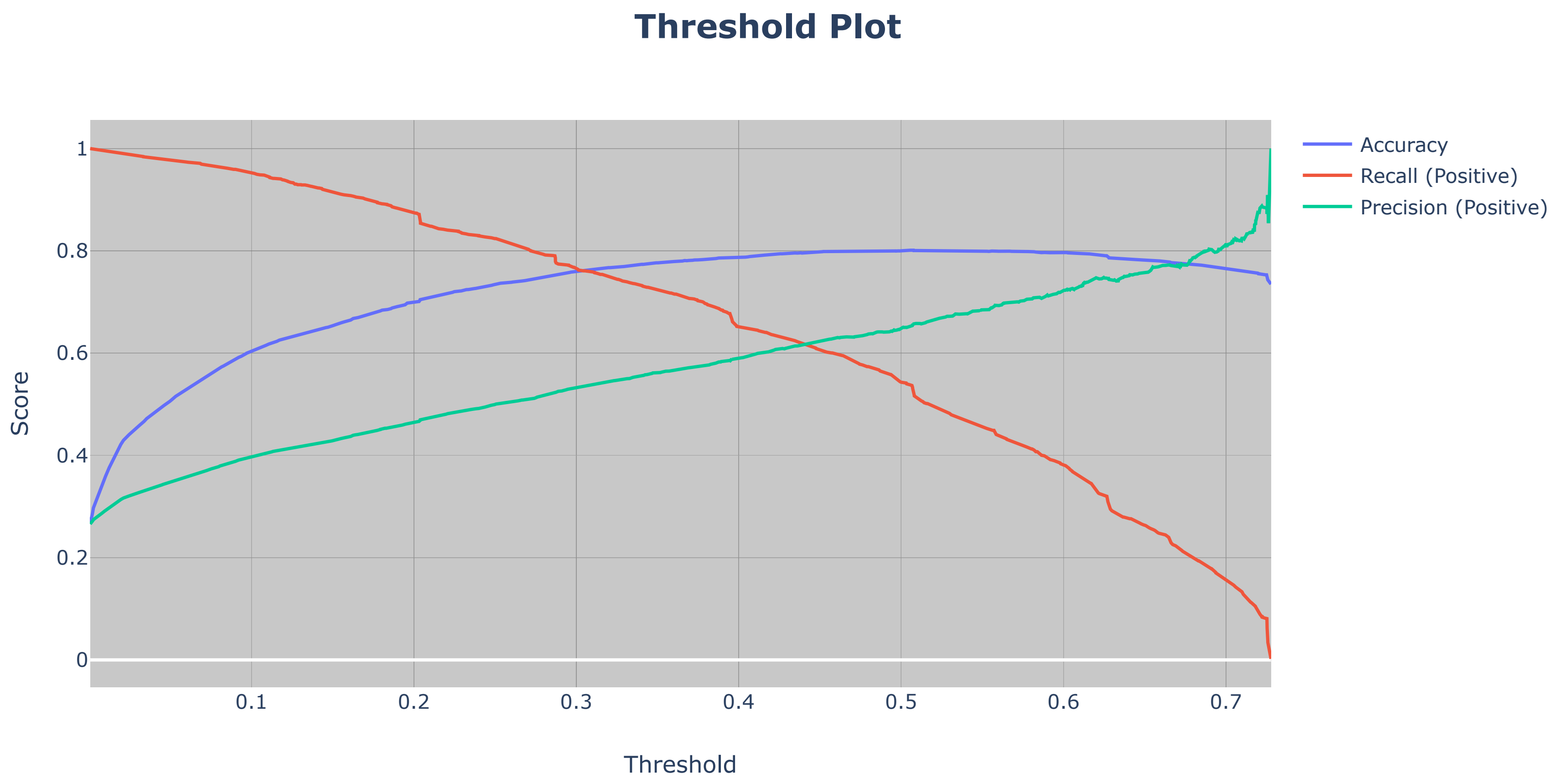
In [43]:

```
X_rfe = X[cols_selected]

_, performance_lr_rfe_kfolds, feat_importance_lr_rfe_kfolds = run_ml(X_rfe, y, logis_rfe, 'Logistic Regression RFE',
                                                                     n_folds = 5, feature_importance_attr = 'coef_')
```

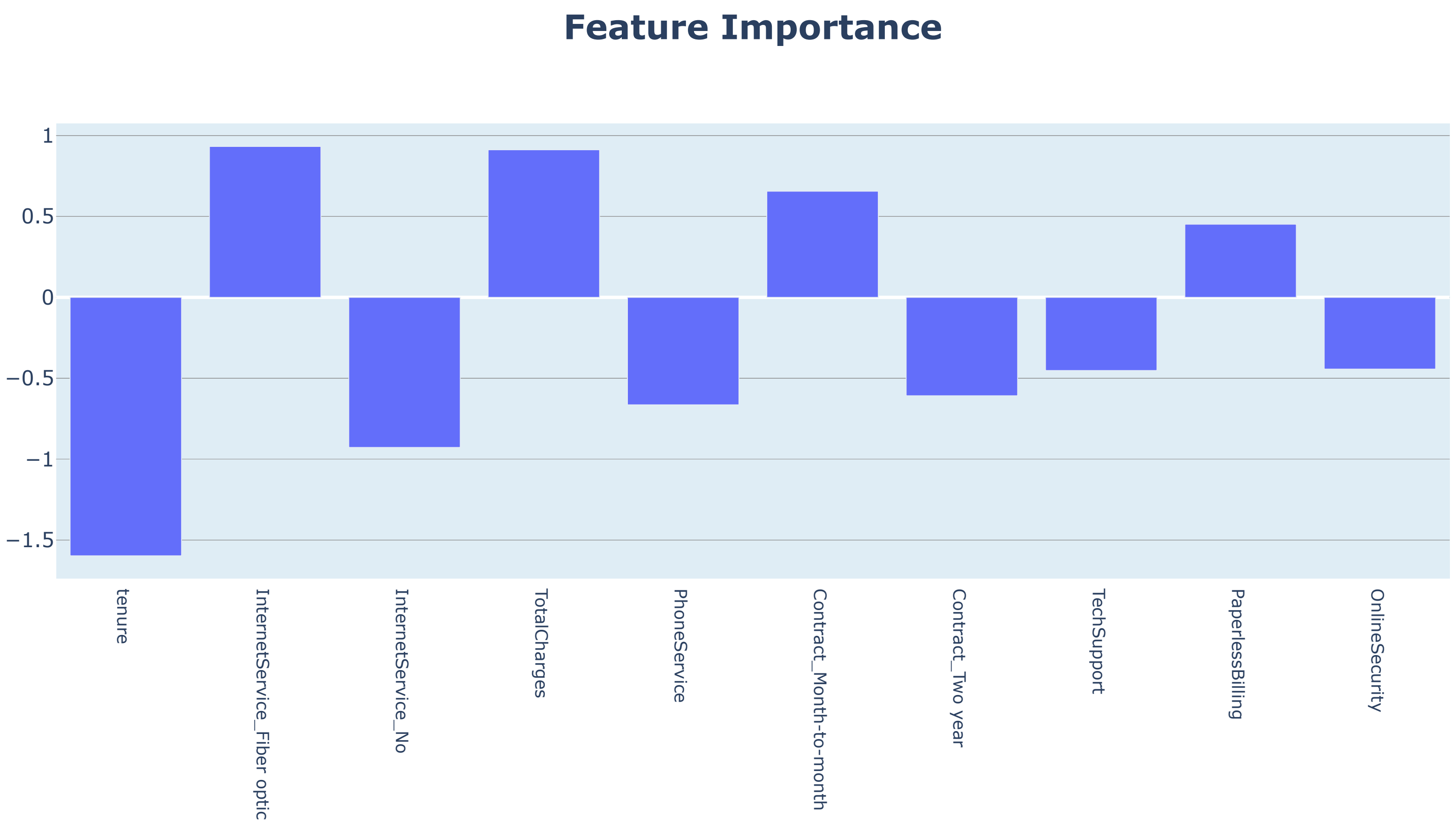
```
Summary of Logistic Regression RFE Model with 5-Fold Cross Validation:

Algorithm:

LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
                   solver='liblinear', tol=0.0001, verbose=0, warm_start=False)

Test Accuracy: 0.7987770193401593

Classification Report:
               precision    recall  f1-score   support

           0       0.84      0.89      0.87      5163
           1       0.64      0.55      0.59      1869

    accuracy                           0.80      7032
   macro avg       0.74      0.72      0.73      7032
weighted avg       0.79      0.80      0.79      7032
```

Performance of Logistic Regression RFE (5-Fold Cross Validation) Model





Threshold Plot
Accuracy
Recall (Positive)
Precision (Positive)
Score
Threshold
0
0.2
0.4
0.6
0.8
1
0.1
0.2
0.3
0.4
0.5
0.6
0.7



2.4 K Nearest Neighbors with K-Fold Cross Validation

In [44]:
```
knn = KNeighborsClassifier()
```

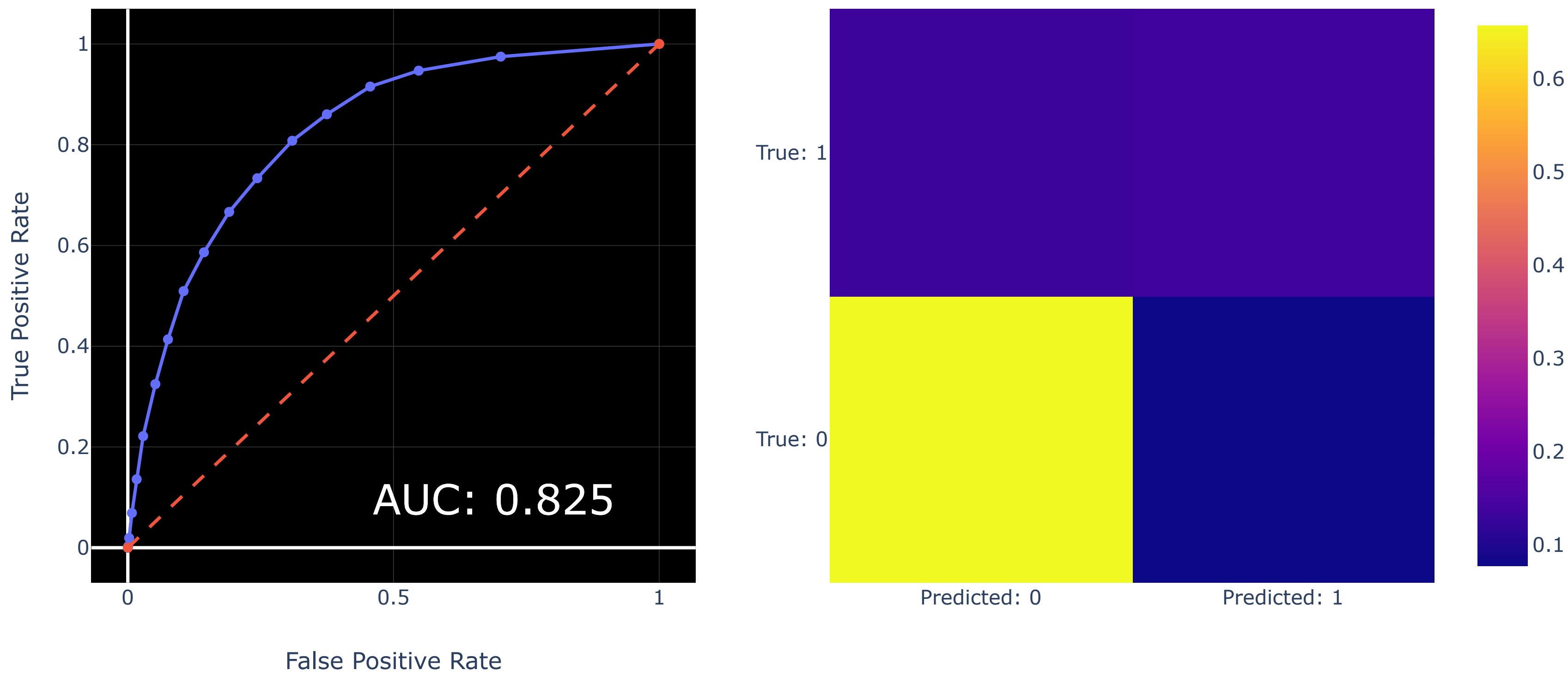
In [45]:
```python
parameters = {
    'n_neighbors' : np.arange(4, 20, 4)
}
knn_params = grid_search(X, y, knn, parameters)
```

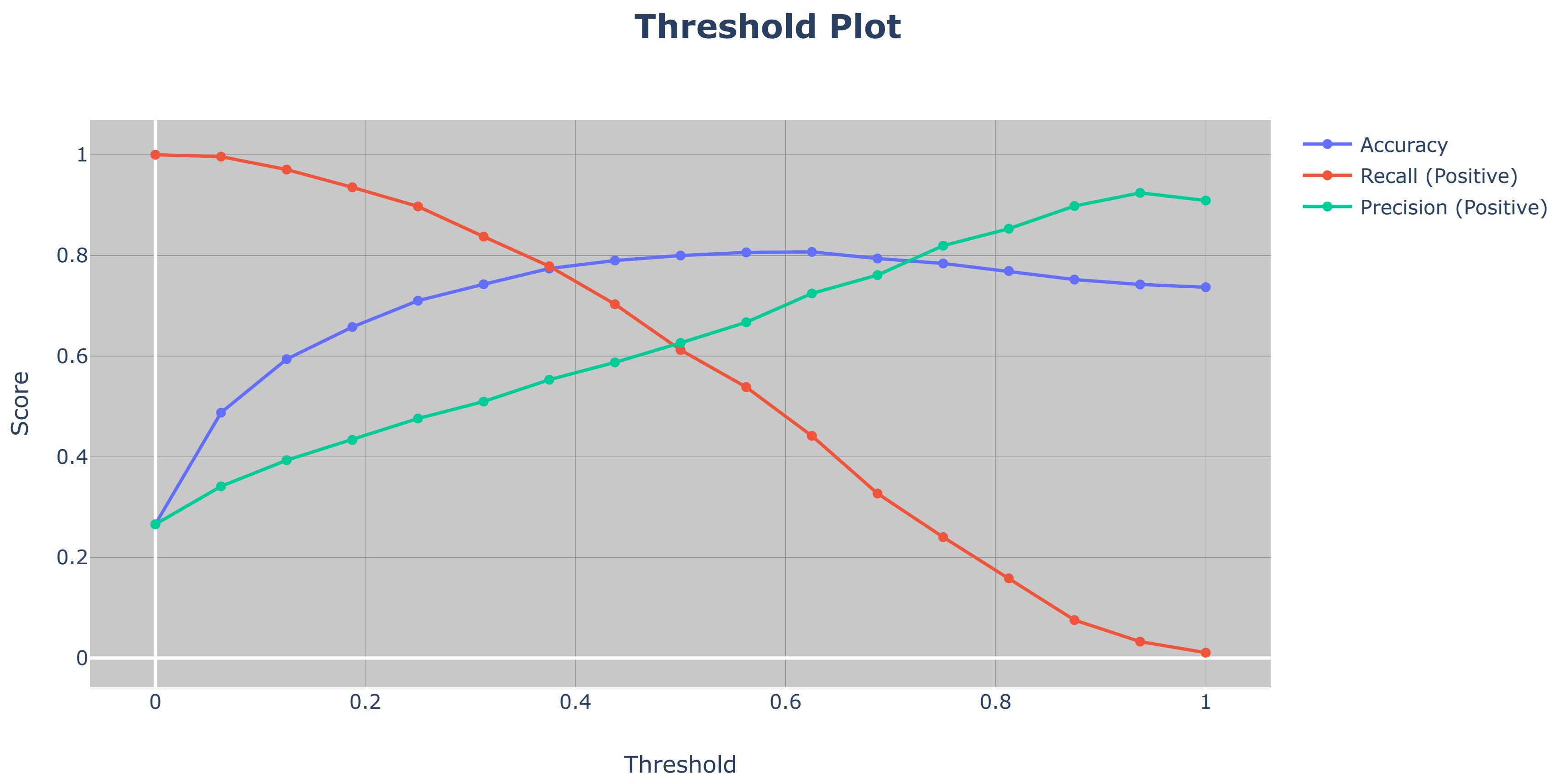
```
Fitting 5 folds for each of 4 candidates, totalling 20 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

Best Score: 0.7922354948805461

Best Parameter Set: {'n_neighbors': 16}

[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:    5.3s finished
```

In [46]:
```
knn = KNeighborsClassifier(n_neighbors = knn_params['n_neighbors'])
_, performance_knn_kfolds, feat_importance_knn_kfolds = run_ml(X, y, knn, 'K-Nearest-Neighbors', n_folds = 5,
                                                                feature_importance_attr = None)
```

```
Summary of K-Nearest-Neighbors Model with 5-Fold Cross Validation:

Algorithm:

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=16, p=2,
                     weights='uniform')

Test Accuracy: 0.7925199089874858

Classification Report:
               precision    recall  f1-score   support

           0       0.83      0.90      0.86      5163
           1       0.64      0.51      0.57      1869

    accuracy                           0.79      7032
   macro avg       0.74      0.70      0.71      7032
weighted avg       0.78      0.79      0.78      7032
```

Performance of K-Nearest-Neighbors (5-Fold Cross Validation) Model





2.5 Random Forest with K-Fold Cross Validation

In [47]:
```
rand_forest = RandomForestClassifier(n_estimators = 100, criterion = 'entropy', random_state = 0)
```

In [48]:
```
rand_forest, performance_rf_kfolds, feat_importance_rf_kfolds = run_ml(X, y, rand_forest, 'Random Forest', n_folds = 5,
                                                                       feature_importance_attr = 'feature_importances_')
```

Summary of Random Forest Model with 5-Fold Cross Validation:

Algorithm:

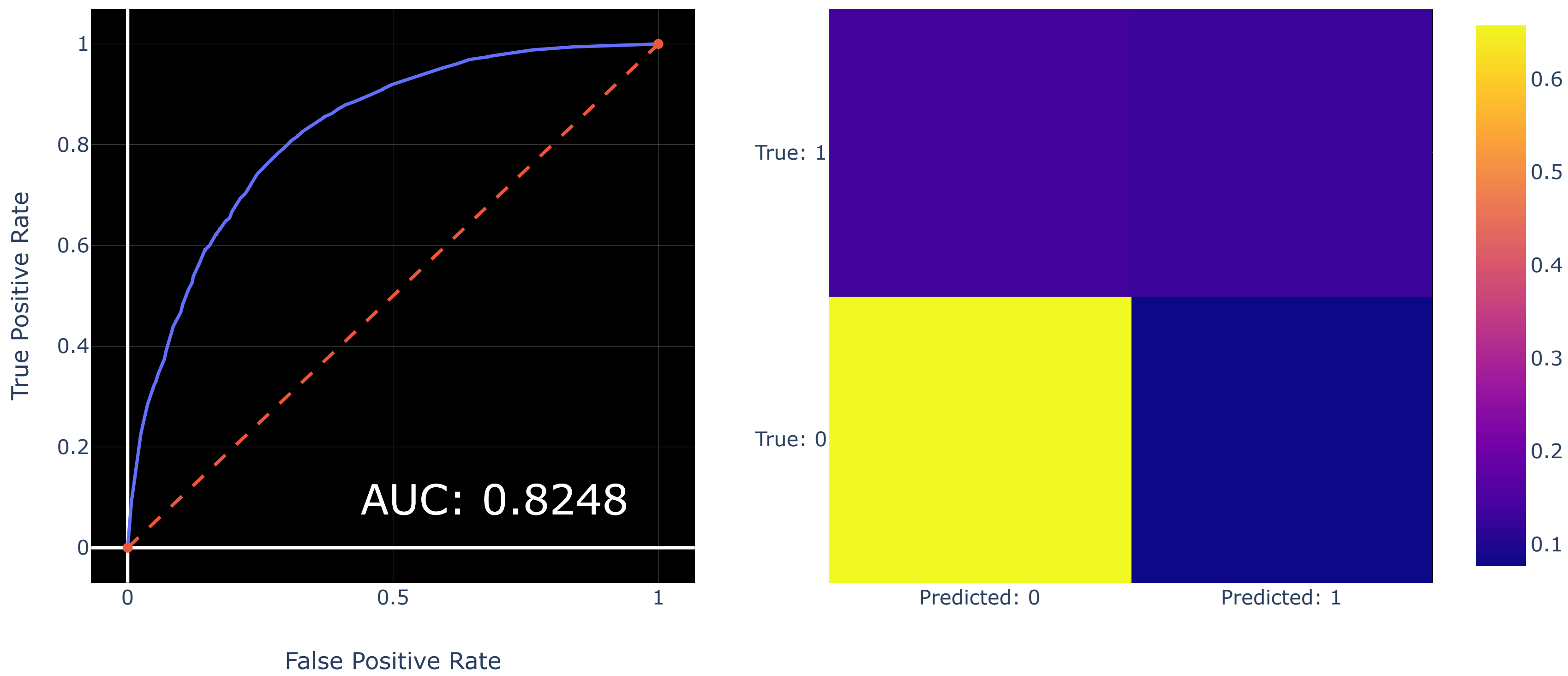
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

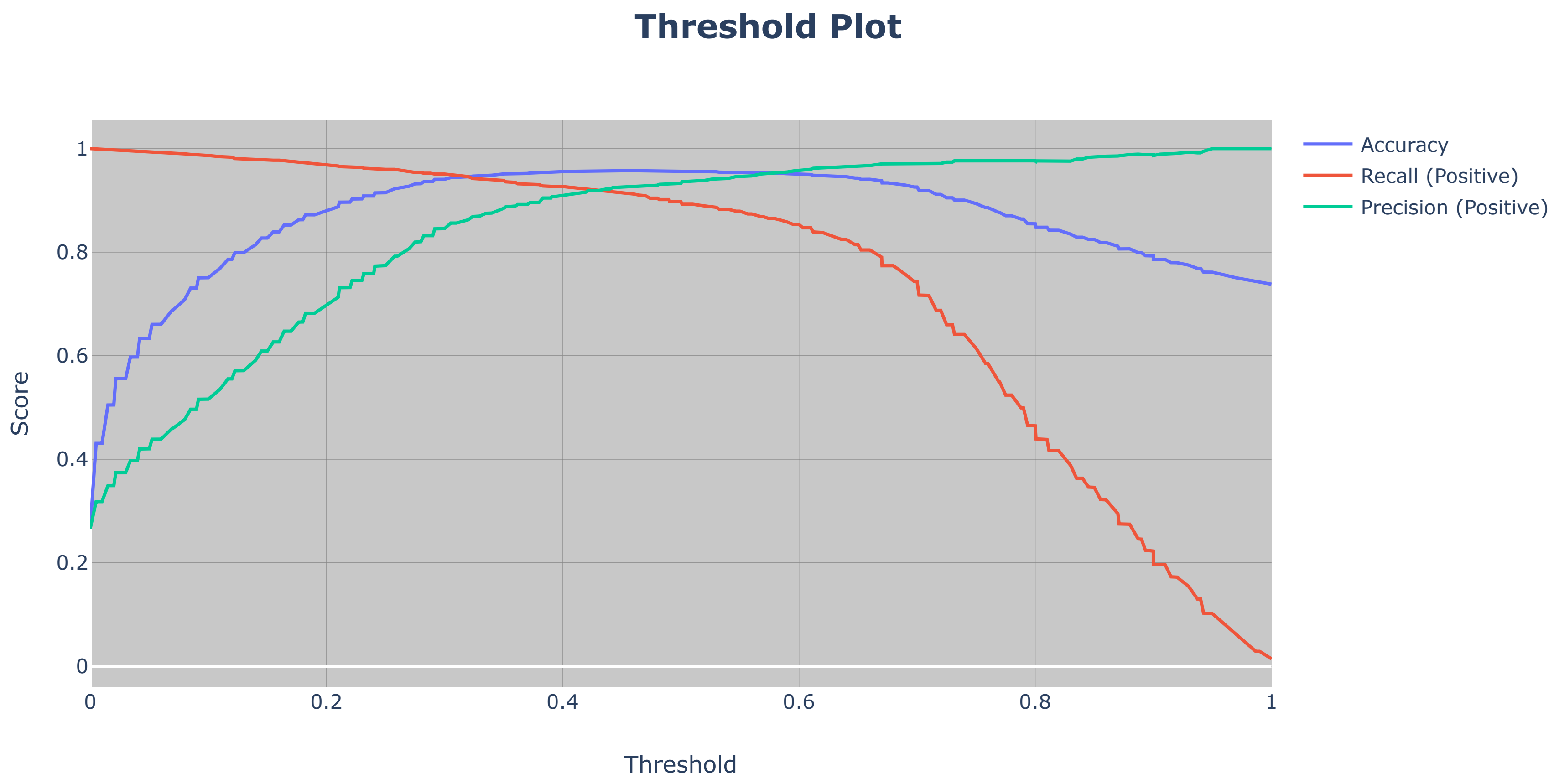
Test Accuracy: 0.7864050056882821

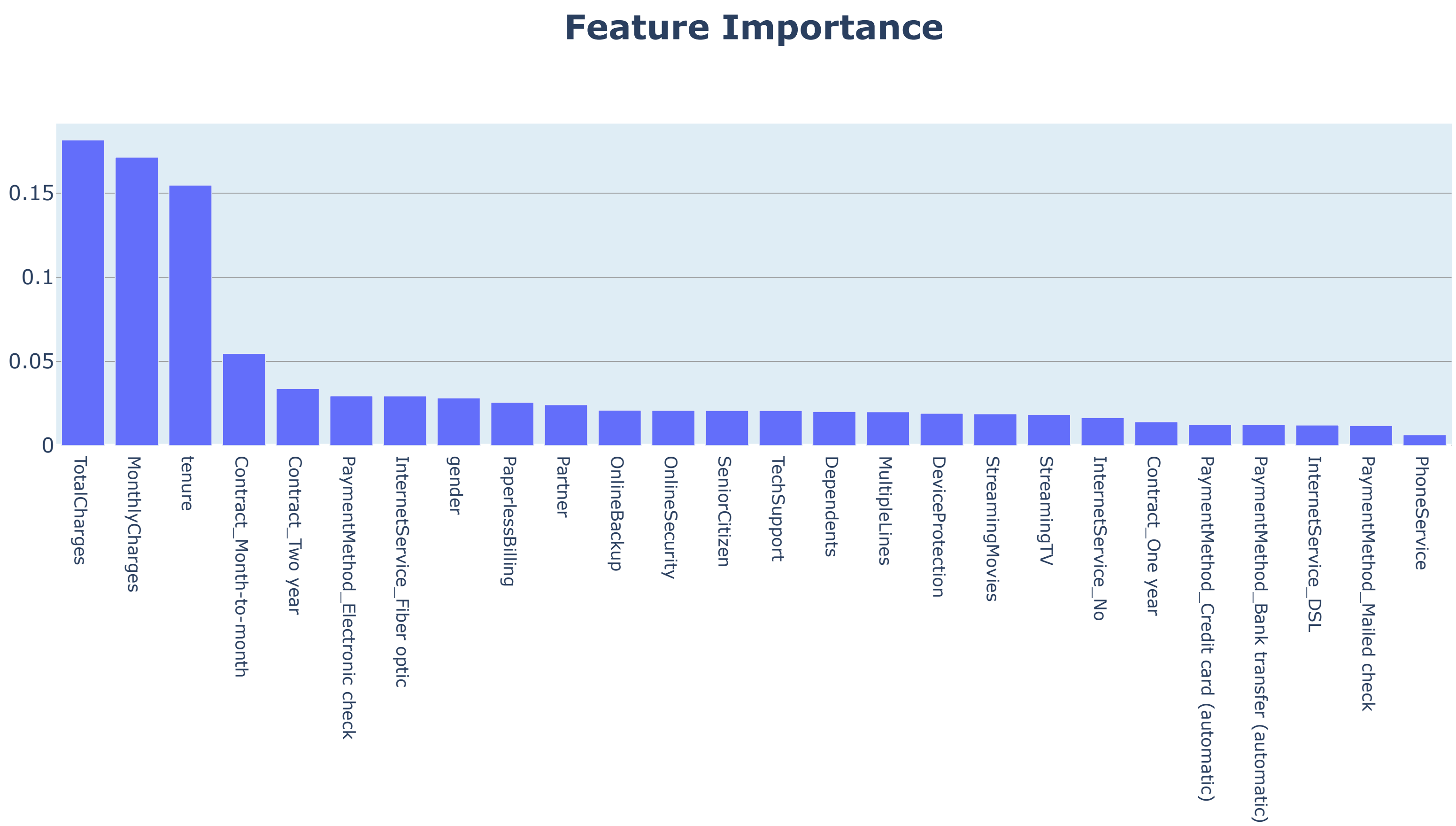
Classification Report:

	precision	recall	f1-score	support
0	0.83	0.90	0.86	5163
1	0.63	0.49	0.55	1869
accuracy			0.79	7032
macro avg	0.73	0.69	0.70	7032
weighted avg	0.77	0.79	0.78	7032

Performance of Random Forest (5-Fold Cross Validation) Model

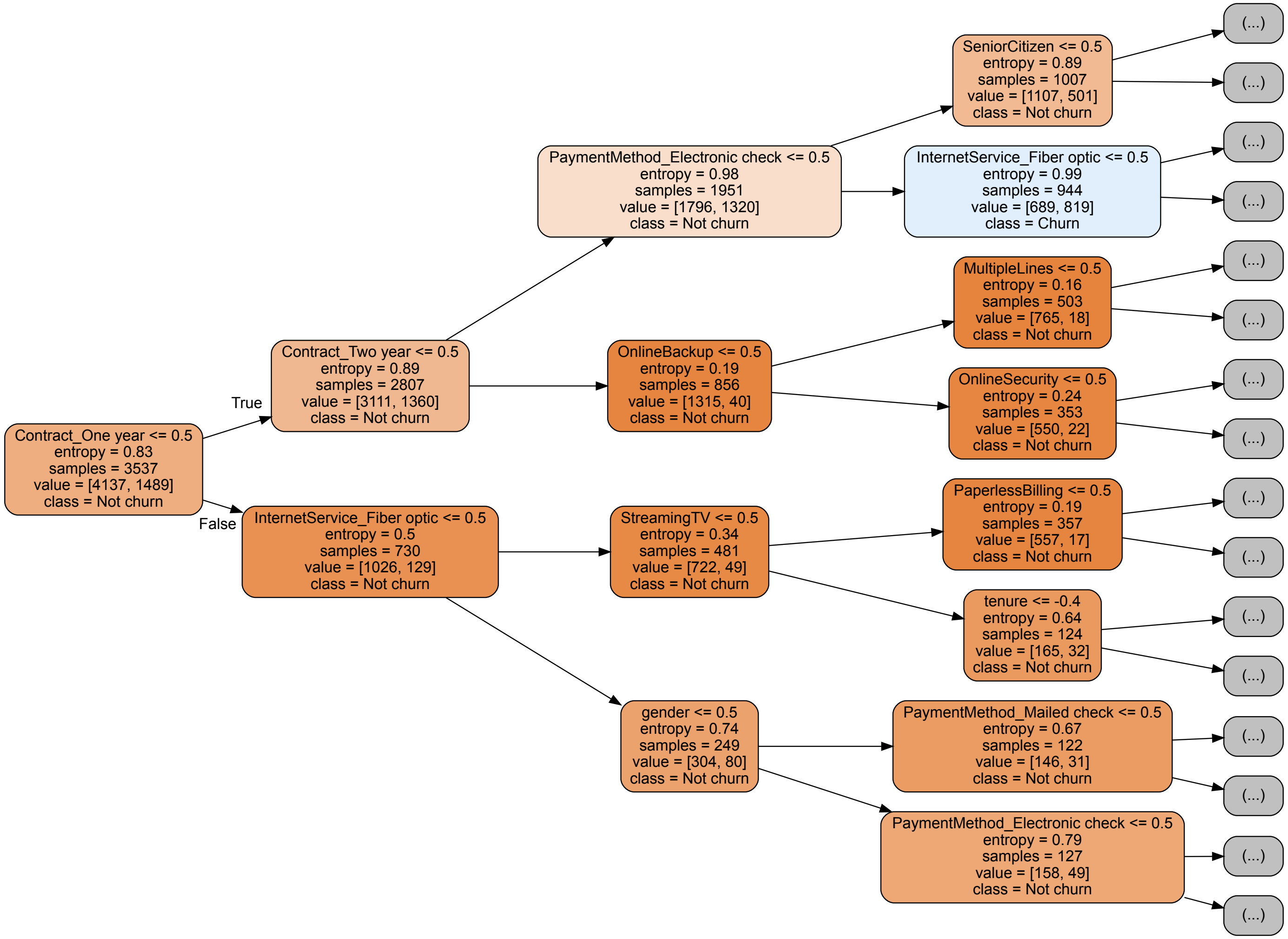



Threshold Plot
Accuracy
Recall (Positive)
Precision (Positive)
Score
Threshold
0
0.2
0.4
0.6
0.8
1

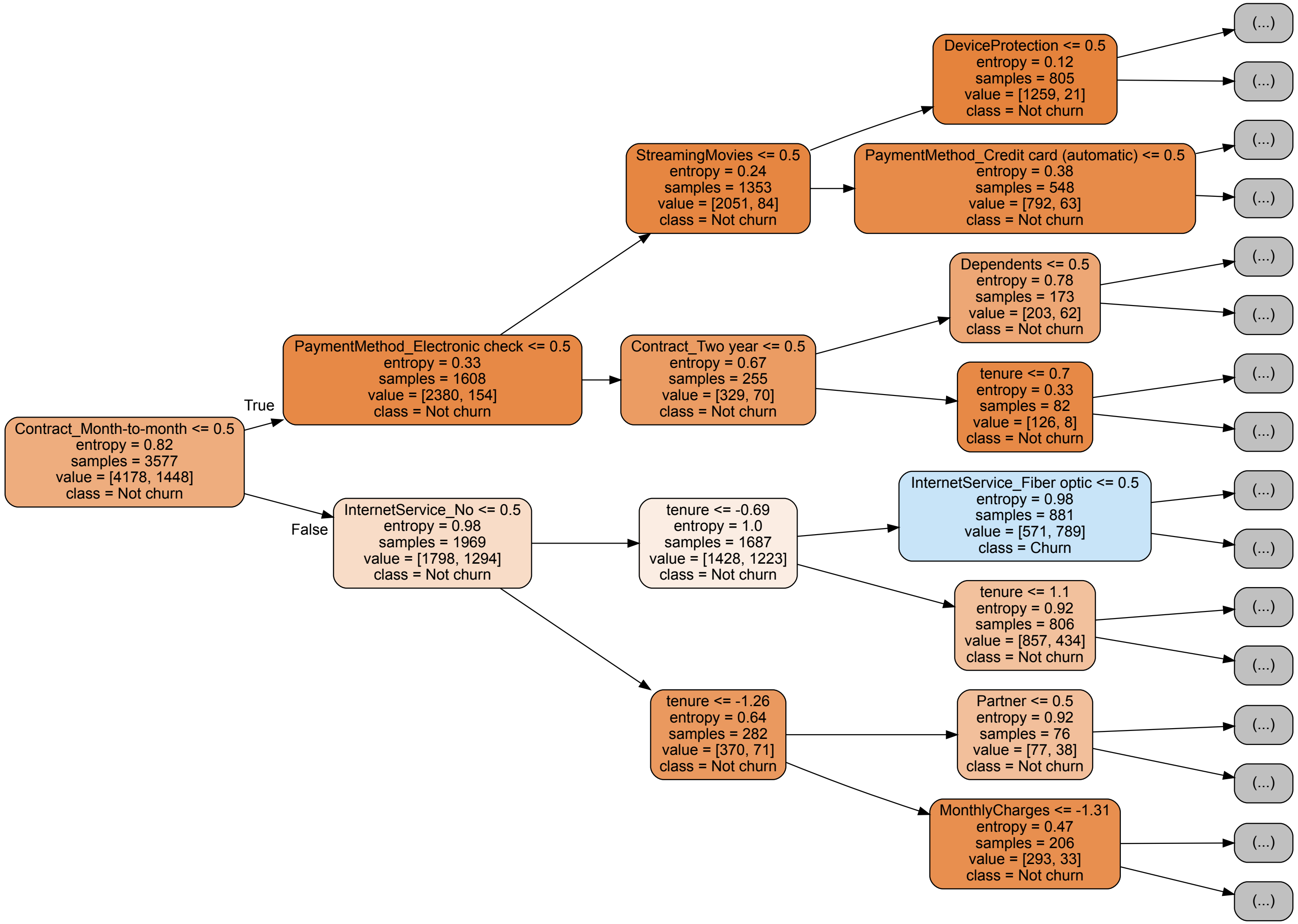

Feature Importance
0.15
0.1
0.05
0
TotalCharges
MonthlyCharges
tenure
Contract_Month-to-month
Contract_Two year
PaymentMethod_Electronic check
InternetService_Fiber optic
gender
PaperlessBilling
Partner
OnlineBackup
OnlineSecurity
SeniorCitizen
TechSupport
Dependents
MultipleLines
DeviceProtection
StreamingMovies
StreamingTV
InternetService_No
Contract_One year
PaymentMethod_Credit card (automatic)
PaymentMethod_Bank transfer (automatic)
InternetService_DSL
PaymentMethod_Mailed check
PhoneService

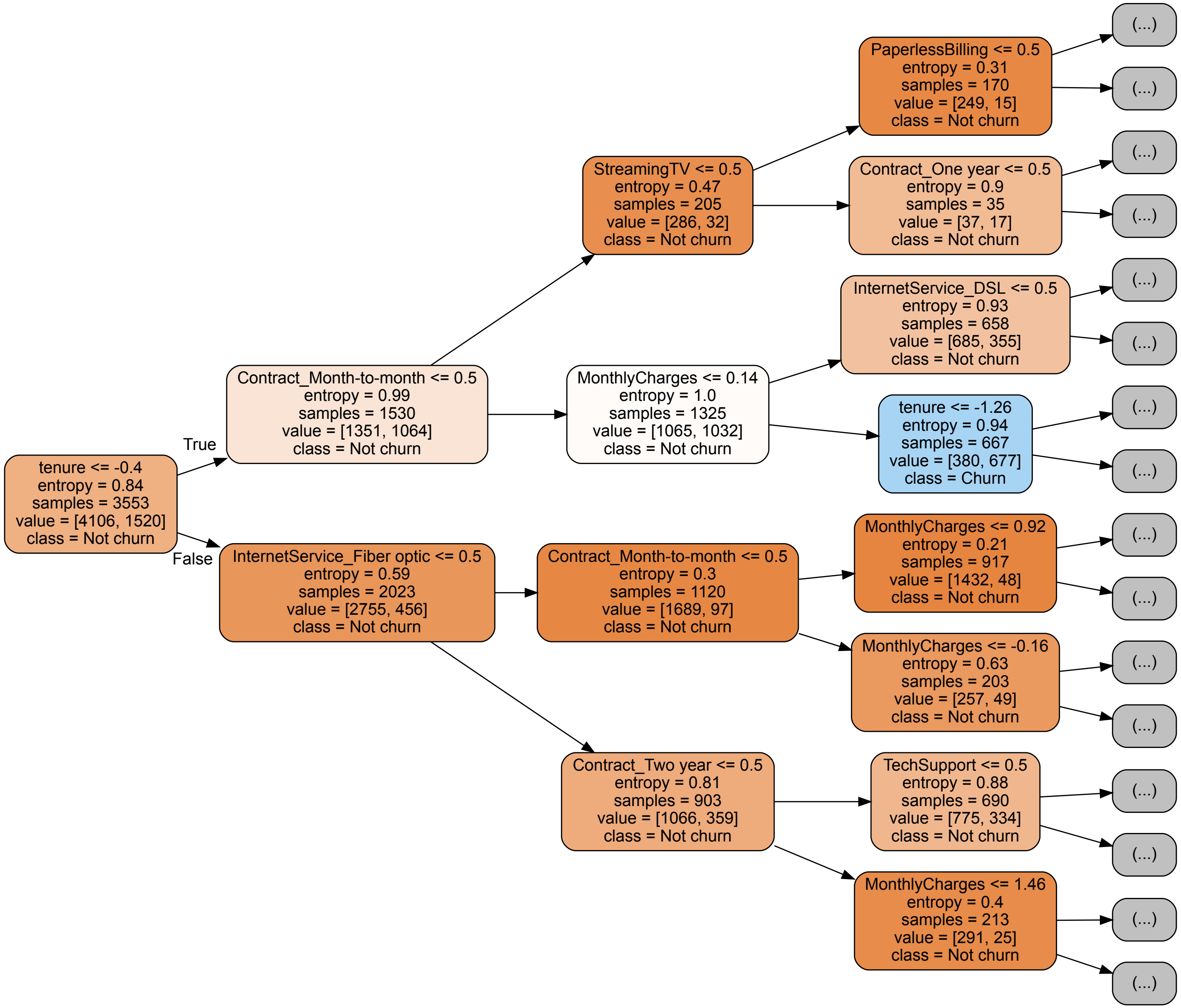
In [49]:

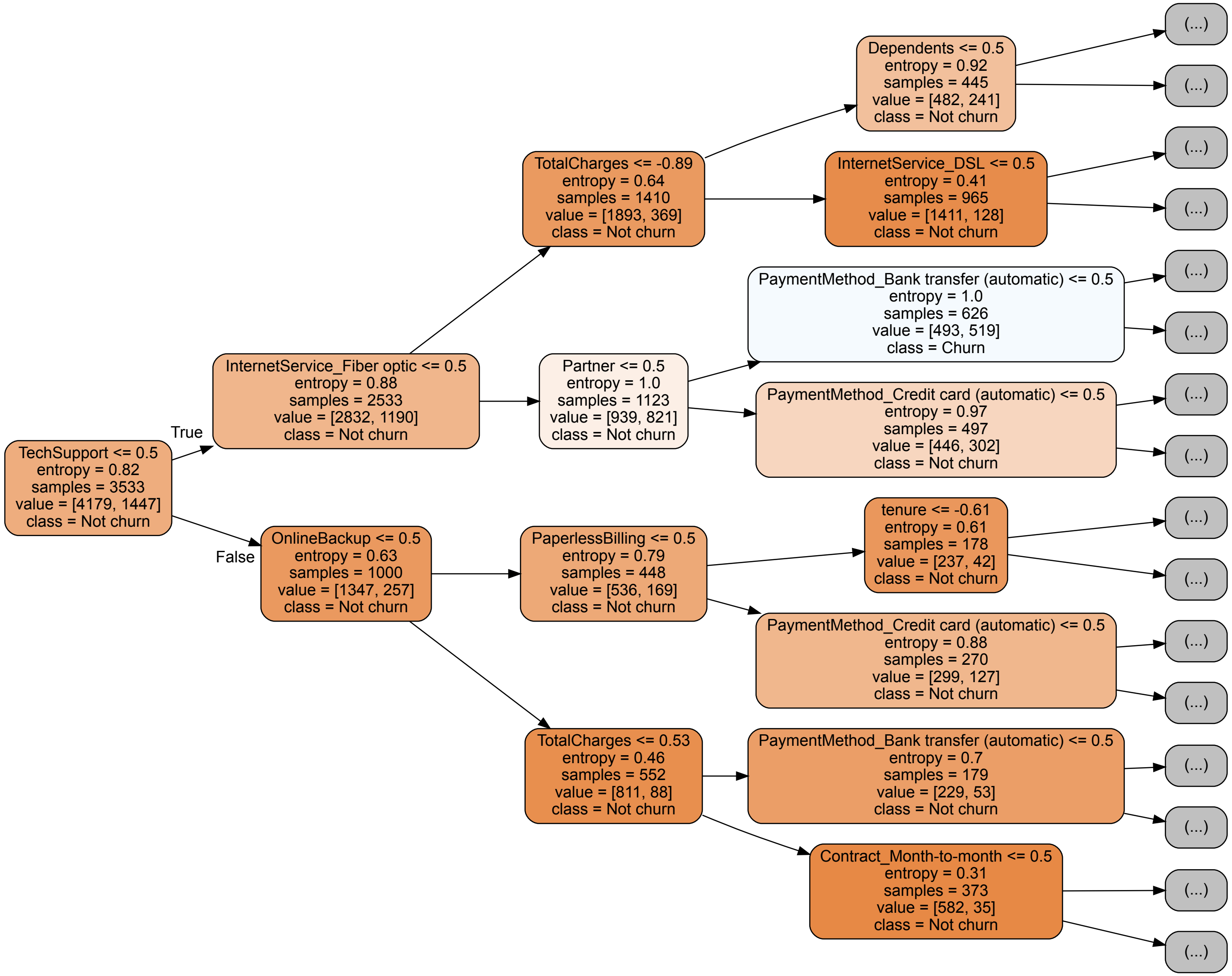
```
# Plot some of the decision trees

for i in range(5):
    dot_data = export_graphviz(rand_forest.estimators_[i], out_file = None, max_depth = 3, feature_names = X.columns.tolist(),
                               rounded = True, proportion = False, precision = 2, class_names = ["Not churn","Churn"],
                               filled = True, rotate = True)
    pdot = pydot.graph_from_dot_data(dot_data)
    # Access element [0] because graph_from_dot_data actually returns a list of DOT elements.
    pdot[0].set_graph_defaults(size = "\"8,8\"")
    graph = Source(pdot[0].to_string())
    display(graph)
```

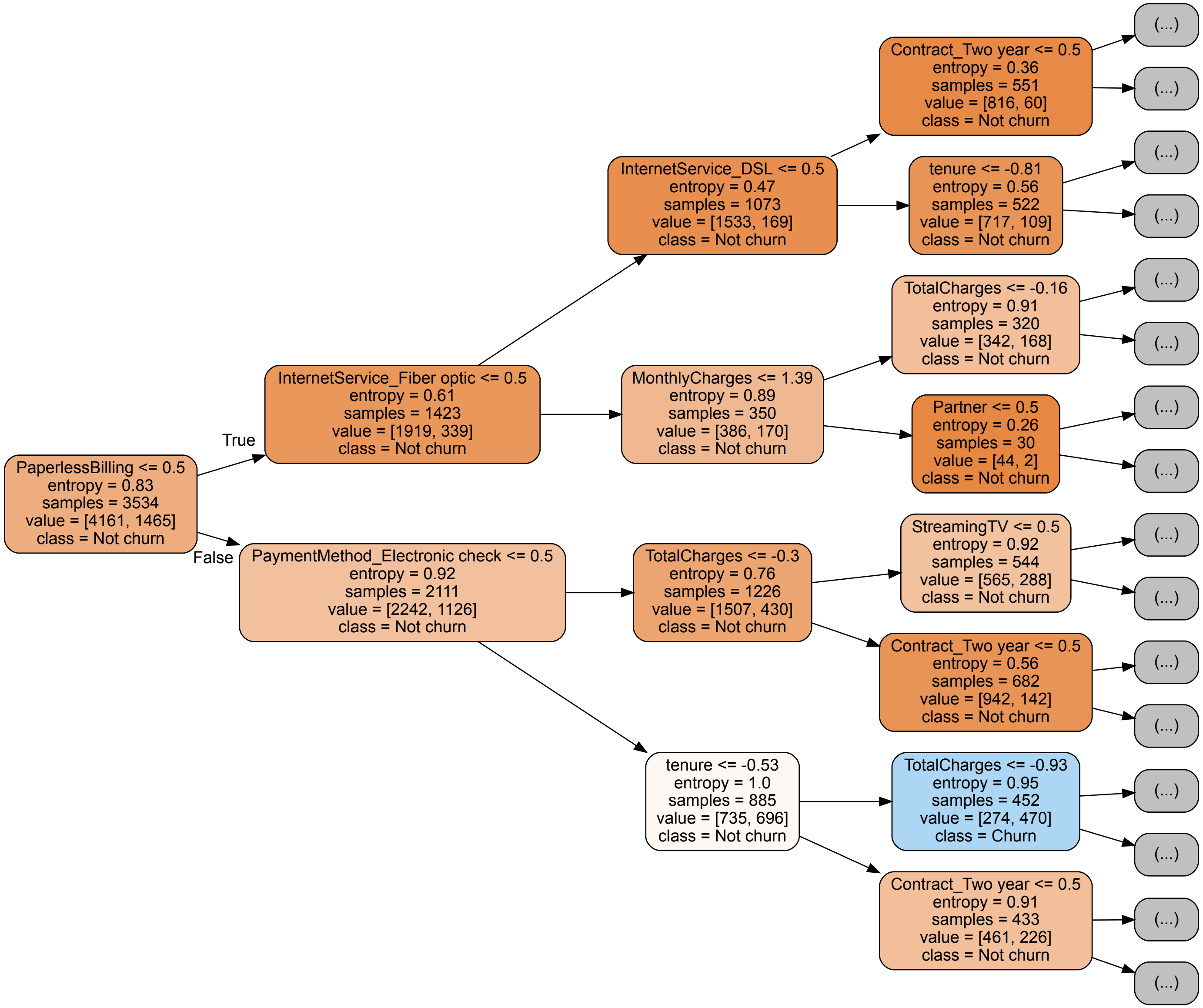


Contract_One year <= 0.5
entropy = 0.83
samples = 3537
value = [4137, 1489]
class = Not churn
True
False
Contract_Two year <= 0.5
entropy = 0.89
samples = 2807
value = [3111, 1360]
class = Not churn
InternetService_Fiber optic <= 0.5
entropy = 0.5
samples = 730
value = [1026, 129]
class = Not churn
PaymentMethod_Electronic check <= 0.5
entropy = 0.98
samples = 1951
value = [1796, 1320]
class = Not churn
OnlineBackup <= 0.5
entropy = 0.19
samples = 856
value = [1315, 40]
class = Not churn
StreamingTV <= 0.5
entropy = 0.34
samples = 481
value = [722, 49]
class = Not churn
gender <= 0.5
entropy = 0.74
samples = 249
value = [304, 80]
class = Not churn
SeniorCitizen <= 0.5
entropy = 0.89
samples = 1007
value = [1107, 501]
class = Not churn
InternetService_Fiber optic <= 0.5
entropy = 0.99
samples = 944
value = [689, 819]
class = Churn
MultipleLines <= 0.5
entropy = 0.16
samples = 503
value = [765, 18]
class = Not churn
OnlineSecurity <= 0.5
entropy = 0.24
samples = 353
value = [550, 22]
class = Not churn
PaperlessBilling <= 0.5
entropy = 0.19
samples = 357
value = [557, 17]
class = Not churn
tenure <= -0.4
entropy = 0.64
samples = 124
value = [165, 32]
class = Not churn
PaymentMethod_Mailed check <= 0.5
entropy = 0.67
samples = 122
value = [146, 31]
class = Not churn
PaymentMethod_Electronic check <= 0.5
entropy = 0.79
samples = 127
value = [158, 49]
class = Not churn
(...)


Contract_Month-to-month <= 0.5
entropy = 0.82
samples = 3577
value = [4178, 1448]
class = Not churn
True
False
PaymentMethod_Electronic check <= 0.5
entropy = 0.33
samples = 1608
value = [2380, 154]
class = Not churn
InternetService_No <= 0.5
entropy = 0.98
samples = 1969
value = [1798, 1294]
class = Not churn
StreamingMovies <= 0.5
entropy = 0.24
samples = 1353
value = [2051, 84]
class = Not churn
Contract_Two year <= 0.5
entropy = 0.67
samples = 255
value = [329, 70]
class = Not churn
tenure <= -0.69
entropy = 1.0
samples = 1687
value = [1428, 1223]
class = Not churn
tenure <= -1.26
entropy = 0.64
samples = 282
value = [370, 71]
class = Not churn
DeviceProtection <= 0.5
entropy = 0.12
samples = 805
value = [1259, 21]
class = Not churn
PaymentMethod_Credit card (automatic) <= 0.5
entropy = 0.38
samples = 548
value = [792, 63]
class = Not churn
Dependents <= 0.5
entropy = 0.78
samples = 173
value = [203, 62]
class = Not churn
tenure <= 0.7
entropy = 0.33
samples = 82
value = [126, 8]
class = Not churn
InternetService_Fiber optic <= 0.5
entropy = 0.98
samples = 881
value = [571, 789]
class = Churn
tenure <= 1.1
entropy = 0.92
samples = 806
value = [857, 434]
class = Not churn
Partner <= 0.5
entropy = 0.92
samples = 76
value = [77, 38]
class = Not churn
MonthlyCharges <= -1.31
entropy = 0.47
samples = 206
value = [293, 33]
class = Not churn
(...)


tenure <= -0.4
entropy = 0.84
samples = 3553
value = [4106, 1520]
class = Not churn
True
False
Contract_Month-to-month <= 0.5
entropy = 0.99
samples = 1530
value = [1351, 1064]
class = Not churn
InternetService_Fiber optic <= 0.5
entropy = 0.59
samples = 2023
value = [2755, 456]
class = Not churn
StreamingTV <= 0.5
entropy = 0.47
samples = 205
value = [286, 32]
class = Not churn
MonthlyCharges <= 0.14
entropy = 1.0
samples = 1325
value = [1065, 1032]
class = Not churn
Contract_Month-to-month <= 0.5
entropy = 0.3
samples = 1120
value = [1689, 97]
class = Not churn
Contract_Two year <= 0.5
entropy = 0.81
samples = 903
value = [1066, 359]
class = Not churn
PaperlessBilling <= 0.5
entropy = 0.31
samples = 170
value = [249, 15]
class = Not churn
Contract_One year <= 0.5
entropy = 0.9
samples = 35
value = [37, 17]
class = Not churn
InternetService_DSL <= 0.5
entropy = 0.93
samples = 658
value = [685, 355]
class = Not churn
tenure <= -1.26
entropy = 0.94
samples = 667
value = [380, 677]
class = Churn
MonthlyCharges <= 0.92
entropy = 0.21
samples = 917
value = [1432, 48]
class = Not churn
MonthlyCharges <= -0.16
entropy = 0.63
samples = 203
value = [257, 49]
class = Not churn
TechSupport <= 0.5
entropy = 0.88
samples = 690
value = [775, 334]
class = Not churn
MonthlyCharges <= 1.46
entropy = 0.4
samples = 213
value = [291, 25]
class = Not churn
(...)


TechSupport <= 0.5
entropy = 0.82
samples = 3533
value = [4179, 1447]
class = Not churn
True
False
InternetService_Fiber optic <= 0.5
entropy = 0.88
samples = 2533
value = [2832, 1190]
class = Not churn
OnlineBackup <= 0.5
entropy = 0.63
samples = 1000
value = [1347, 257]
class = Not churn
TotalCharges <= -0.89
entropy = 0.64
samples = 1410
value = [1893, 369]
class = Not churn
Partner <= 0.5
entropy = 1.0
samples = 1123
value = [939, 821]
class = Not churn
PaperlessBilling <= 0.5
entropy = 0.79
samples = 448
value = [536, 169]
class = Not churn
TotalCharges <= 0.53
entropy = 0.46
samples = 552
value = [811, 88]
class = Not churn
Dependents <= 0.5
entropy = 0.92
samples = 445
value = [482, 241]
class = Not churn
InternetService_DSL <= 0.5
entropy = 0.41
samples = 965
value = [1411, 128]
class = Not churn
PaymentMethod_Bank transfer (automatic) <= 0.5
entropy = 1.0
samples = 626
value = [493, 519]
class = Churn
PaymentMethod_Credit card (automatic) <= 0.5
entropy = 0.97
samples = 497
value = [446, 302]
class = Not churn
tenure <= -0.61
entropy = 0.61
samples = 178
value = [237, 42]
class = Not churn
PaymentMethod_Credit card (automatic) <= 0.5
entropy = 0.88
samples = 270
value = [299, 127]
class = Not churn
PaymentMethod_Bank transfer (automatic) <= 0.5
entropy = 0.7
samples = 179
value = [229, 53]
class = Not churn
Contract_Month-to-month <= 0.5
entropy = 0.31
samples = 373
value = [582, 35]
class = Not churn
(...)



value: the list represents the count of samples in each class that have reached that node.
sample: if using bootstrap, this number is not equal to the sum of the numbers in 'value'.

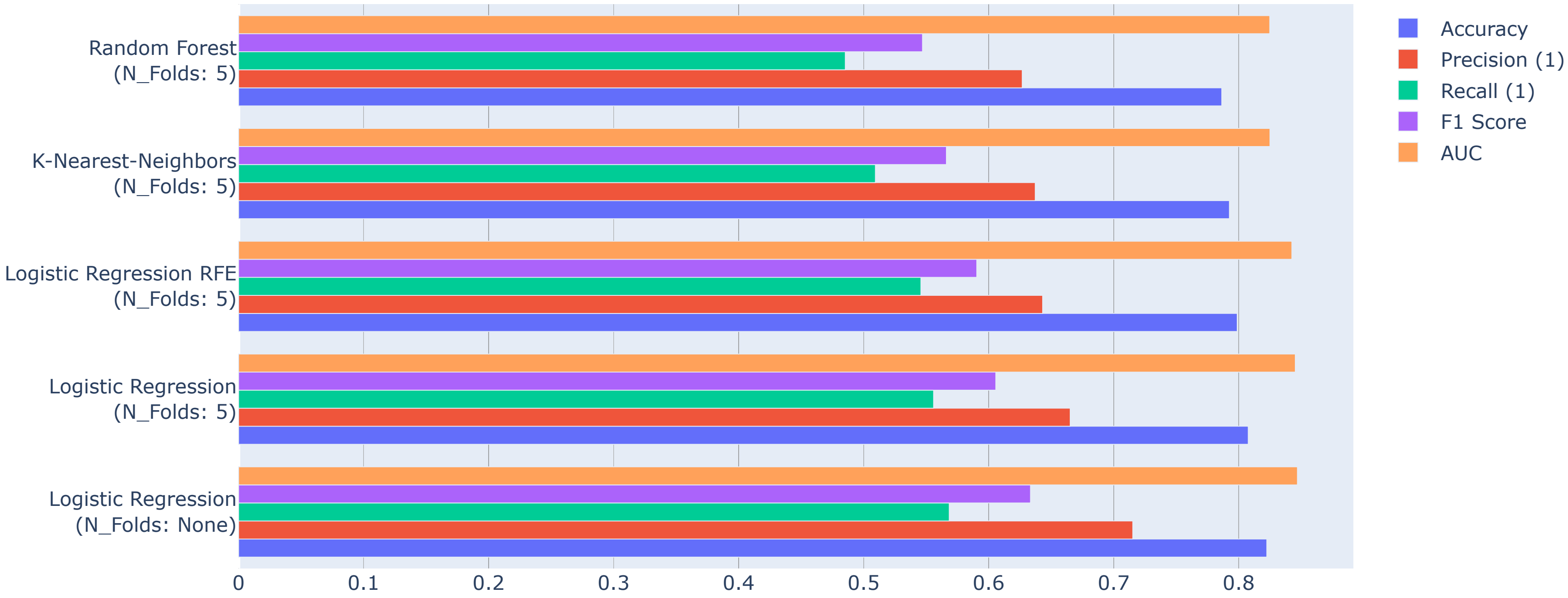
3. Summary

3.1 Model Performance

In [50]:
```
# contruct the dataframe table including the performance of all models

performances_df = pd.DataFrame()
performance_df = [performance_lr, performance_lr_kfolds, performance_lr_rfe_kfolds, performance_knn_kfolds, performance_rf_kfolds]
for i in performance_df:
    performances_df = performances_df.append(i, sort = False)
```

In [51]:
```
# bar chart comparing the performance of different models

metrices = ['Accuracy', 'Precision (1)', 'Recall (1)', 'F1 Score', 'AUC']
traces_performance = []
for metric in metrices:
    trace = go.Bar(y = performances_df['Model'], x = performances_df[metric], orientation = 'h', name = metric)
    traces_performance.append(trace)
layout_performance = go.Layout(title = dict(text = '<b>Model Performance Summary</b>', font = dict(family = 'Times New Roman', size = 
25),
                                            x = 0.5),
                               margin = dict(l = 150)) # 'l' in 'margin' sets the left margin in px
fig_performance = go.Figure(data = traces_performance, layout = layout_performance)
pyo.iplot(fig_performance)

# summary table

table_performance = ff.create_table(np.round(performances_df, 4))
pyo.iplot(table_performance)
```

Model Performance Summary

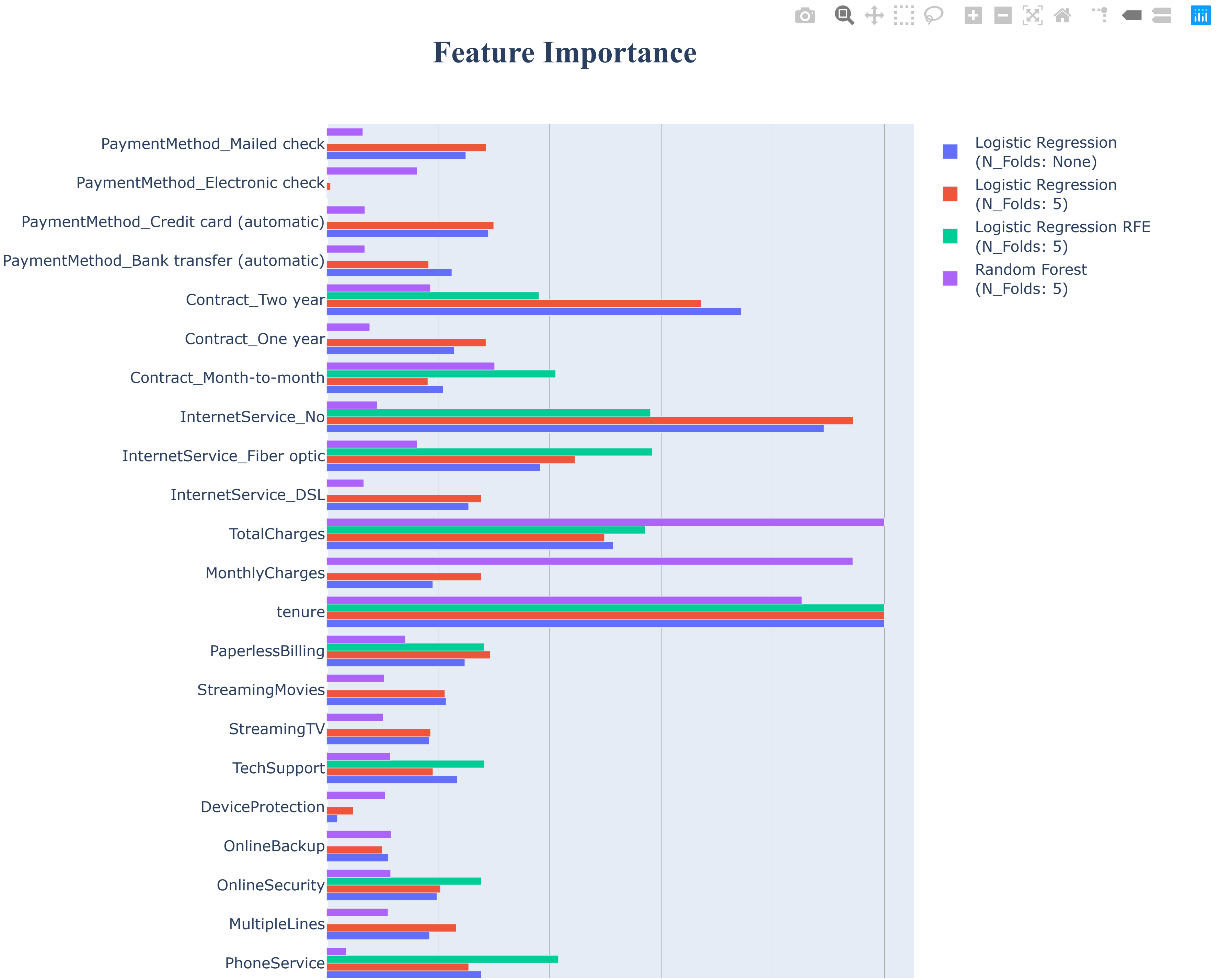


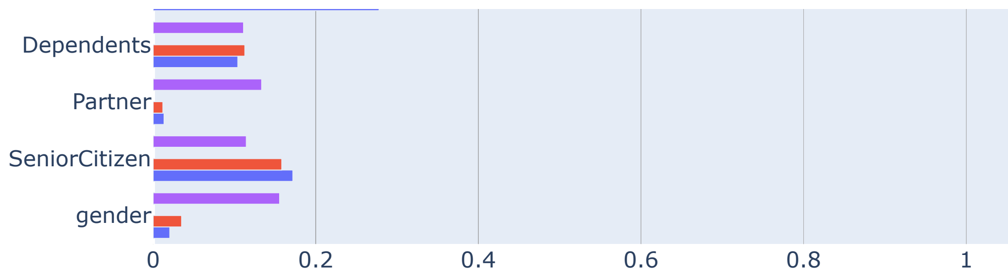
Model	Test Data Size	Accuracy	Precision (1)	Recall (1)	F1 Score	AUC
Logistic Regression (N_Folds: None)	704	0.8224	0.7152	0.5684	0.6334	0.8469
Logistic Regression (N_Folds: 5)	7032	0.8076	0.6652	0.5559	0.6057	0.8452
Logistic Regression RFE (N_Folds: 5)	7032	0.7988	0.6431	0.5457	0.5904	0.8425
K-Nearest-Neighbors (N_Folds: 5)	7032	0.7925	0.6372	0.5094	0.5662	0.825
Random Forest (N_Folds: 5)	7032	0.7864	0.6268	0.4853	0.547	0.8248

3.2 Feature Importance

In [52]:
```
feats_importance_df = pd.DataFrame()
feat_importance_df = [feat_importance_lr, feat_importance_lr_kfolds, feat_importance_lr_rfe_kfolds, feat_importance_knn_kfolds,
                      feat_importance_rf_kfolds]
for i in feat_importance_df:
    if i is not None:
        i = i.abs()
        max = i.max() # for normalization, set the max to be 1
        feats_importance_df = pd.concat([feats_importance_df, i / max], axis = 1, sort = False)
```

In [53]:
```
traces_feat_importance = []
for i in feats_importance_df.columns:
    trace = go.Bar(y = feats_importance_df.index, x = feats_importance_df[i], orientation = 'h', name = i)
    traces_feat_importance.append(trace)
layout_feat_importance = go.Layout(title = dict(text = '<b>Feature Importance</b>', font = dict(family = 'Times New Roman', size = 25
),
                                                x = 0.5),
                                   yaxis = dict(dtick = 1),
                                   margin = dict(l = 300),
                                   height = 1000)
fig_feat_importance = go.Figure(data = traces_feat_importance, layout = layout_feat_importance)
pyo.iplot(fig_feat_importance)
```


Feature Importance
PaymentMethod_Mailed check
PaymentMethod_Electronic check
PaymentMethod_Credit card (automatic)
PaymentMethod_Bank transfer (automatic)
Contract_Two year
Contract_One year
Contract_Month-to-month
InternetService_No
InternetService_Fiber optic
InternetService_DSL
TotalCharges
MonthlyCharges
tenure
PaperlessBilling
StreamingMovies
StreamingTV
TechSupport
DeviceProtection
OnlineBackup
OnlineSecurity
MultipleLines
PhoneService
Logistic Regression (N_Folds: None)
Logistic Regression (N_Folds: 5)
Logistic Regression RFE (N_Folds: 5)
Random Forest (N_Folds: 5)


Dependents
Partner
SeniorCitizen
gender
0
0.2
0.4
0.6
0.8
1

In [54]:

```
# summary table

print('Feature Importance Summary:')
table_feat_importance_mean = ff.create_table(pd.DataFrame(feats_importance_df.mean(axis = 1).sort_values(axis = 0)[::-1],
                                      columns = ['Mean Importance']).reset_index())
pyo.iplot(table_feat_importance_mean)
```

Feature Importance Summary:

Features	Mean Importance
tenure	0.9630173596922257
TotalCharges	0.6457857584345441
InternetService_No	0.6267839269666561
Contract_Two year	0.4957349744170466
MonthlyCharges	0.4705673303586739
InternetService_Fiber optic	0.39369312772910836
Contract_Month-to-month	0.2758324173686368
PhoneService	0.24596280269354162
PaperlessBilling	0.2415529248475717
PaymentMethod_Credit card (automatic)	0.21965729919730578
TechSupport	0.20568534139028977
PaymentMethod_Mailed check	0.2003460678487564
InternetService_DSL	0.19989215054796086
OnlineSecurity	0.19872298622421242
Contract_One year	0.19750966997494915
StreamingMovies	0.17672114393745927
MultipleLines	0.1758751654808676
PaymentMethod_Bank transfer (automatic)	0.1588764197893863
StreamingTV	0.15754563756482862
SeniorCitizen	0.14818334945641423
Dependents	0.10935270769604277
OnlineBackup	0.10888267047331805
gender	0.07031909228841961
DeviceProtection	0.05763341863163266

PaymentMethod_Electronic check	0.057132621248535076
Partner	0.05291432629360901