

Customer Churn Prediction

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('darkgrid')
```

Data Preparation based on EDA

```
In [2]: def datapreparation(filepath):

    df = pd.read_csv(filepath)
    df.drop(["customerID"], inplace = True, axis = 1)

    df.TotalCharges = df.TotalCharges.replace(" ",np.nan)
    df.TotalCharges.fillna(0, inplace = True)
    df.TotalCharges = df.TotalCharges.astype(float)

    cols1 = ['Partner', 'Dependents', 'PaperlessBilling', 'Churn', 'PhoneService'
    for col in cols1:
        df[col] = df[col].apply(lambda x: 0 if x == "No" else 1)

    df.gender = df.gender.apply(lambda x: 0 if x == "Male" else 1)
    df.MultipleLines = df.MultipleLines.map({'No phone service': 0, 'No': 0, 'Yes': 1})

    cols2 = ['OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport'
    for col in cols2:
        df[col] = df[col].map({'No internet service': 0, 'No': 0, 'Yes': 1})

    df = pd.get_dummies(df, columns=['InternetService', 'Contract', 'PaymentMethod'])

    return df
```

```
In [3]: df = datapreparation(filepath = "C:/Data/Telco-Customer-Churn.csv")
df.head()
```

Out[3]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	O
0	1	0	1	0	1	0	0	0
1	0	0	0	0	34	1	0	0
2	0	0	0	0	2	1	0	0
3	0	0	0	0	45	0	0	0
4	1	0	0	0	2	1	0	0

5 rows × 24 columns

```
In [4]: df.isnull().any().any()
```

Out[4]: False

Model Building

```
In [84]: from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.metrics import roc_auc_score, roc_curve, precision_score, recall_score
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier

In [6]: train, test = train_test_split(df, test_size=0.2, random_state=111, stratify = df['Churn'])

In [7]: x = df.columns[df.columns!="Churn"]
y = "Churn"
train_x = train[x]
train_y = train[y]
test_x = test[x]
test_y = test[y]

In [8]: #function for model fitting
def churn_prediction(algo, training_x, training_y, testing_x, testing_y, cols, c):
    algo.fit(training_x,training_y)
    predictions = algo.predict(testing_x)
    probabilities = algo.predict_proba(testing_x)[:,1]

    #coeffs
    if cf == "coefficients":
        coefficients = pd.DataFrame(algo.coef_.ravel())
    elif cf == "features":
        coefficients = pd.DataFrame(algo.feature_importances_)

    column_df = pd.DataFrame(cols)
    coef_sumry = (pd.merge(coefficients, column_df, left_index= True,
                           right_index= True, how = "left"))
    coef_sumry.columns = ["coefficients","features"]
    coef_sumry = coef_sumry.sort_values(by = "coefficients", ascending = False)

    print (algo)
    print ("\n Classification report : \n",classification_report(testing_y,predictions))
    print ("Accuracy Score : ",accuracy_score(testing_y,predictions))

    #confusion matrix
    conf_matrix = confusion_matrix(testing_y,predictions)
    plt.figure(figsize=(12,12))
    plt.subplot(221)
    sns.heatmap(conf_matrix, fmt = "d", annot=True, cmap='Blues')
    plt.title('Confusion Matrix')
    plt.ylabel('True Values')
    plt.xlabel('Predicted Values')

    #roc_auc_score
    model_roc_auc = roc_auc_score(testing_y,probabilities)
    print ("Area under curve : ",model_roc_auc,"\\n")
    fpr,tpr,thresholds = roc_curve(testing_y,probabilities)

    plt.subplot(222)
    plt.plot(fpr, tpr, color='darkorange', lw=1, label = "Auc : %.3f" %model_roc_auc)
```

```

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")

plt.subplot(212)
sns.barplot(x = coef_sumry["features"] ,y = coef_sumry["coefficients"])
plt.title('Feature Importances')
plt.xticks(rotation="vertical")

plt.show()

```

Hyperparameters Tuning

Grid 1: Selecting class weight and estimators

```

In [32]: param_grid1 = {'max_features':['auto', 'sqrt', 'log2', None],
                     'n_estimators':[300, 500, 700, 900, 1100, 1300]
                    }

rf_model = RandomForestClassifier()
grid1 = GridSearchCV(estimator=rf_model, param_grid=param_grid1, n_jobs=-1, cv=3)
grid1.fit(train_x, train_y)

```

Fitting 3 folds for each of 24 candidates, totalling 72 fits

```

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   28.9s
[Parallel(n_jobs=-1)]: Done  72 out of  72 | elapsed:  1.4min finished

```

```

Out[32]: GridSearchCV(cv=3, error_score=nan,
                      estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                                      class_weight=None,
                                                      criterion='gini', max_depth=None,
                                                      max_features='auto',
                                                      max_leaf_nodes=None,
                                                      max_samples=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=None, verbose=0,
                                                      warm_start=False),
                      iid='deprecated', n_jobs=-1,
                      param_grid={'max_features': ['auto', 'sqrt', 'log2', None],
                                  'n_estimators': [300, 500, 700, 900, 1100, 1300]},
                      pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                      scoring='f1', verbose=1)

```

```
In [35]: grid1.best_estimator_
```

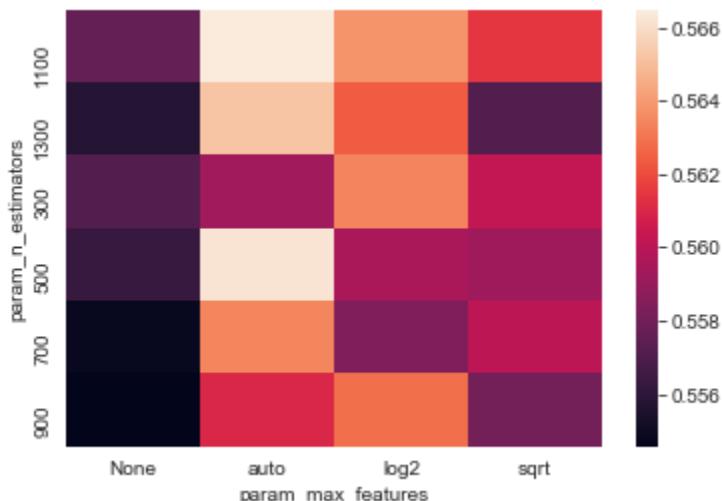
```
Out[35]: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                                criterion='gini', max_depth=None, max_features='auto',
                                max_leaf_nodes=None, max_samples=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, n_estimators=1100,
                                n_jobs=None, oob_score=False, random_state=None,
                                verbose=0, warm_start=False)
```

```
In [37]: dt = pd.DataFrame(grid1.cv_results_)
dt.param_max_features = dt.param_max_features.astype(str)
dt.param_n_estimators = dt.param_n_estimators.astype(str)

table = pd.pivot_table(dt, values='mean_test_score', index='param_n_estimators',
                      columns='param_max_features')

sns.heatmap(table)
```

```
Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x24e31749898>
```



```
In [38]: grid1.best_score_
```

```
Out[38]: 0.5664538956289195
```

Grid 2: Selecting max depth and split criterion

```
In [39]: param_grid2 = {'max_features':['auto'],
                     'n_estimators':[1000, 1100, 1200],
                     'criterion': ['entropy', 'gini'],
                     'max_depth': [7, 9, 11, 13, 15, None],
                     }

rf_model = RandomForestClassifier()
grid2 = GridSearchCV(estimator=rf_model, param_grid=param_grid2, n_jobs=-1, cv=3)
grid2.fit(train_x, train_y)
```

Fitting 3 folds for each of 36 candidates, totalling 108 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 34 tasks | elapsed: 38.8s
[Parallel(n_jobs=-1)]: Done 108 out of 108 | elapsed: 2.0min finished

```
Out[39]: GridSearchCV(cv=3, error_score=nan,
                     estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                                     class_weight=None,
                                                     criterion='gini', max_depth=None,
                                                     max_features='auto',
                                                     max_leaf_nodes=None,
                                                     max_samples=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=None, verbose=0,
                                                     warm_start=False),
                     iid='deprecated', n_jobs=-1,
                     param_grid={'criterion': ['entropy', 'gini'],
                                 'max_depth': [7, 9, 11, 13, 15, None],
                                 'max_features': ['auto'],
                                 'n_estimators': [1000, 1100, 1200]},
                     pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                     scoring='f1', verbose=1)
```

In [40]: `grid2.best_estimator_`

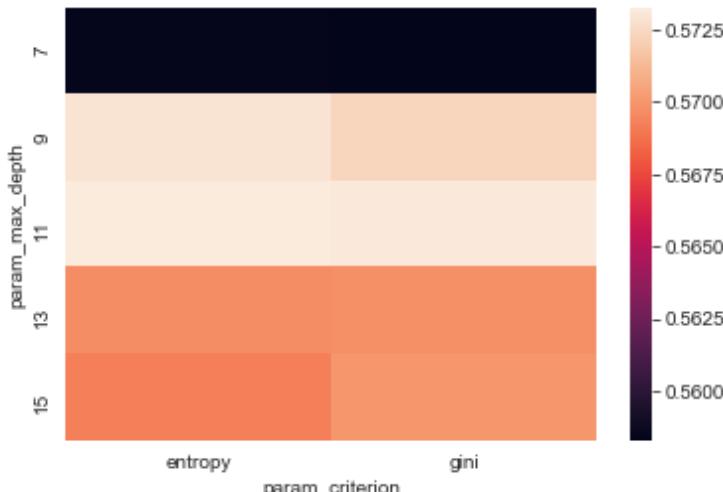
```
Out[40]: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                                 criterion='entropy', max_depth=11, max_features='auto',
                                 max_leaf_nodes=None, max_samples=None,
                                 min_impurity_decrease=0.0, min_impurity_split=None,
                                 min_samples_leaf=1, min_samples_split=2,
                                 min_weight_fraction_leaf=0.0, n_estimators=1000,
                                 n_jobs=None, oob_score=False, random_state=None,
                                 verbose=0, warm_start=False)
```

In [41]: `dt = pd.DataFrame(grid2.cv_results_)`

```
table = pd.pivot_table(dt, values='mean_test_score', index='param_max_depth',
                       columns='param_criterion')

sns.heatmap(table)
```

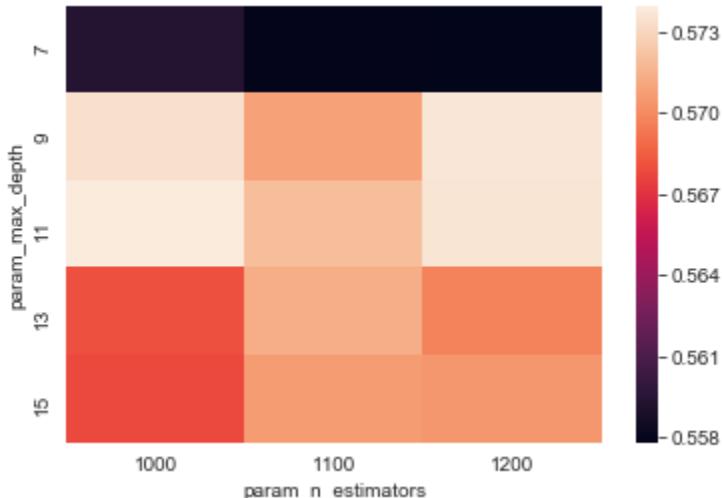
Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x24e26d720f0>



```
In [44]: table = pd.pivot_table(dt, values='mean_test_score', index='param_max_depth',
                               columns='param_n_estimators')

sns.heatmap(table)
```

Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x24e3e7c6550>



```
In [42]: grid2.best_score_
```

Out[42]: 0.5754368499916416

```
In [45]: param_grid2_2 = {'max_features':['auto'],
                      'n_estimators':[950, 1000, 1050],
                      'criterion': ['entropy'],
                      'max_depth': [10, 11, 12],
                      }

rf_model = RandomForestClassifier()
grid2_2 = GridSearchCV(estimator=rf_model, param_grid=param_grid2_2, n_jobs=-1,
grid2_2.fit(train_x, train_y)
```

Fitting 3 folds for each of 9 candidates, totalling 27 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.

[Parallel(n_jobs=-1)]: Done 27 out of 27 | elapsed: 23.4s finished

```
Out[45]: GridSearchCV(cv=3, error_score=nan,
                      estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                                      class_weight=None,
                                                      criterion='gini', max_depth=None,
                                                      max_features='auto',
                                                      max_leaf_nodes=None,
                                                      max_samples=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=None, verbose=0,
                                                      warm_start=False),
                      iid='deprecated', n_jobs=-1,
                      param_grid={'criterion': ['entropy'], 'max_depth': [10, 11, 12],
                                  'max_features': ['auto'],
                                  'n_estimators': [950, 1000, 1050]},
                      pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                      scoring='f1', verbose=1)
```

In [46]: grid2_2.best_estimator_

```
Out[46]: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                                 criterion='entropy', max_depth=10, max_features='auto',
                                 max_leaf_nodes=None, max_samples=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=1000,
                                 n_jobs=None, oob_score=False, random_state=None,
                                 verbose=0, warm_start=False)
```

In [47]: grid2_2.best_score_

Out[47]: 0.57924338341986

Grid 3: Selecting minimum samples leaf and split

```
In [52]: param_grid3 = {'max_features':['auto'],
                      'n_estimators':[1000],
                      'criterion': ['entropy'],
                      'max_depth': [10],
                      'min_samples_leaf': [1, 3, 5, 7],
                      'min_samples_split': [2, 4, 6, 8]
                     }

rf_model = RandomForestClassifier()
grid3 = GridSearchCV(estimator=rf_model, param_grid=param_grid3, n_jobs=-1, cv=3)
grid3.fit(train_x, train_y)
```

Fitting 3 folds for each of 16 candidates, totalling 48 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.

[Parallel(n_jobs=-1)]: Done 48 out of 48 | elapsed: 42.4s finished

```
Out[52]: GridSearchCV(cv=3, error_score=nan,
                      estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                                      class_weight=None,
                                                      criterion='gini', max_depth=None,
                                                      max_features='auto',
                                                      max_leaf_nodes=None,
                                                      max_samples=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=None, verbose=0,
                                                      warm_start=False),
                      iid='deprecated', n_jobs=-1,
                      param_grid={'criterion': ['entropy'], 'max_depth': [10],
                                  'max_features': ['auto'],
                                  'min_samples_leaf': [1, 3, 5, 7],
                                  'min_samples_split': [2, 4, 6, 8],
                                  'n_estimators': [1000]},
                      pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                      scoring='f1', verbose=1)
```

In [53]: `grid3.best_estimator_`

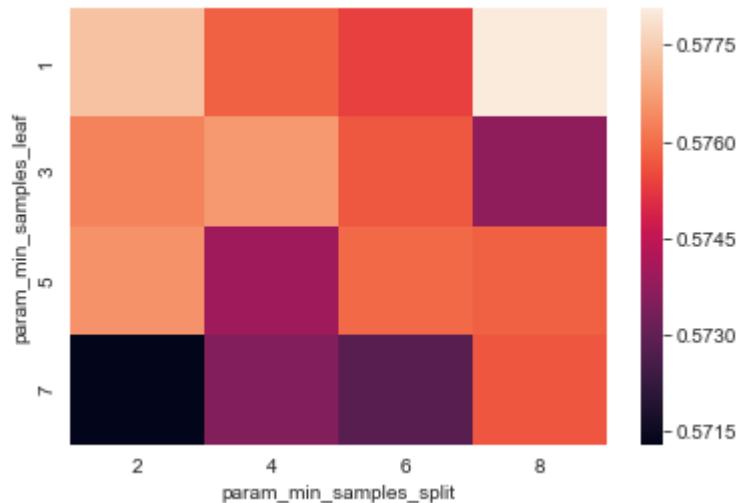
```
Out[53]: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                                 criterion='entropy', max_depth=10, max_features='auto',
                                 max_leaf_nodes=None, max_samples=None,
                                 min_impurity_decrease=0.0, min_impurity_split=None,
                                 min_samples_leaf=1, min_samples_split=8,
                                 min_weight_fraction_leaf=0.0, n_estimators=1000,
                                 n_jobs=None, oob_score=False, random_state=None,
                                 verbose=0, warm_start=False)
```

In [54]: `dt = pd.DataFrame(grid3.cv_results_)`

```
table = pd.pivot_table(dt, values='mean_test_score', index='param_min_samples_le
                           columns='param_min_samples_split')

sns.heatmap(table)
```

Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x24e3e9f2470>



In [55]: `grid3.best_score_`

Out[55]: 0.5780629261081371

Grid 4: Selecting class weight

```
In [66]: param_grid4 = {'class_weight': [{0:1, 1:1}, {0:1, 1:2}, {0:1, 1:3}],
                      'max_features': ['auto'],
                      'n_estimators': [1000],
                      'criterion': ['entropy'],
                      'max_depth': [10],
                      'min_samples_leaf': [1],
                      'min_samples_split': [8]
                     }

rf_model = RandomForestClassifier()
grid4 = GridSearchCV(estimator=rf_model, param_grid=param_grid4, n_jobs=-1, cv=3)
grid4.fit(train_x, train_y)
```

Fitting 3 folds for each of 3 candidates, totalling 9 fits

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done    4 out of    9 | elapsed:      5.2s remaining:     6.6s
[Parallel(n_jobs=-1)]: Done    9 out of    9 | elapsed:     8.3s finished
```

```
Out[66]: GridSearchCV(cv=3, error_score=nan,
                      estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                                      class_weight=None,
                                                      criterion='gini', max_depth=None,
                                                      max_features='auto',
                                                      max_leaf_nodes=None,
                                                      max_samples=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, ...
                                                      random_state=None, verbose=0,
                                                      warm_start=False),
                      iid='deprecated', n_jobs=-1,
                      param_grid={'class_weight': [{0: 1, 1: 1}, {0: 1, 1: 2},
                                                  {0: 1, 1: 3}],
                                  'criterion': ['entropy'], 'max_depth': [10],
                                  'max_features': ['auto'], 'min_samples_leaf': [1],
                                  'min_samples_split': [8], 'n_estimators': [1000]},
                      pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                      scoring='f1', verbose=1)
```

In [67]: `grid4.best_estimator_`

```
Out[67]: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight={0: 1, 1: 3},
                                 criterion='entropy', max_depth=10, max_features='auto',
                                 max_leaf_nodes=None, max_samples=None,
                                 min_impurity_decrease=0.0, min_impurity_split=None,
                                 min_samples_leaf=1, min_samples_split=8,
                                 min_weight_fraction_leaf=0.0, n_estimators=1000,
                                 n_jobs=None, oob_score=False, random_state=None,
                                 verbose=0, warm_start=False)
```

In [73]: `dt = pd.DataFrame(grid4.cv_results_)
dt.param_class_weight = dt.param_class_weight.astype(str)
table = pd.pivot_table(dt, values='mean_test_score', index='param_class_weight')
sns.heatmap(table)`

Out[73]: <matplotlib.axes._subplots.AxesSubplot at 0x24e527cf748>



```
In [65]: grid4.best_score_
```

```
Out[65]: 0.577445110448134
```

Final Model

```
In [76]: model = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight={0: 1
                                         criterion='entropy', max_depth=10, max_features='auto',
                                         max_leaf_nodes=None, max_samples=None,
                                         min_impurity_decrease=0.0, min_impurity_split=None,
                                         min_samples_leaf=1, min_samples_split=8,
                                         min_weight_fraction_leaf=0.0, n_estimators=1000,
                                         n_jobs=None, oob_score=False, random_state=None,
                                         verbose=0, warm_start=False})
```

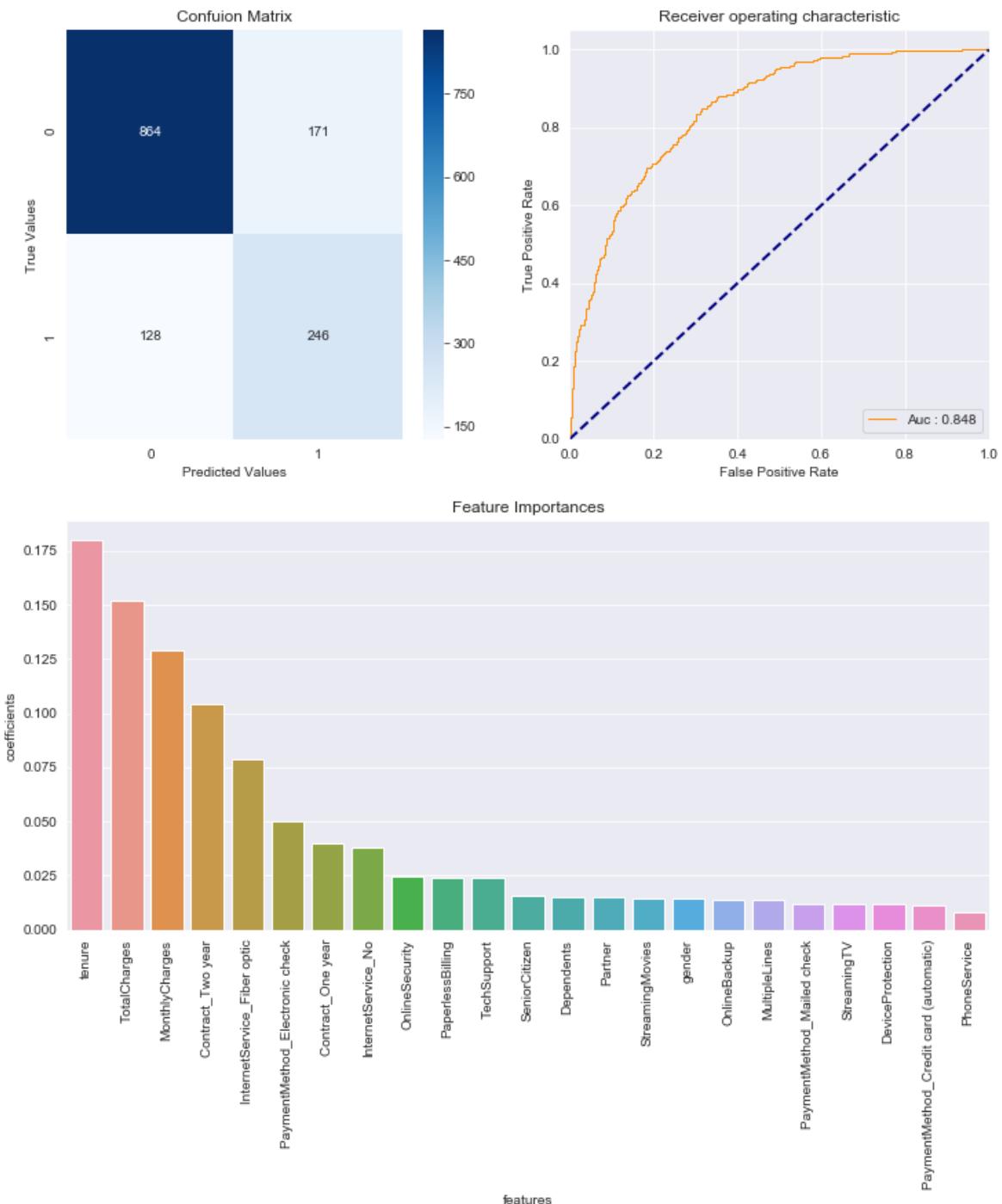
```
In [77]: churn_prediction(model, train_x, train_y, test_x, test_y, x,"features")
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight={0: 1, 1: 2},
                      criterion='entropy', max_depth=10, max_features='auto',
                      max_leaf_nodes=None, max_samples=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=8,
                      min_weight_fraction_leaf=0.0, n_estimators=1000,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
```

Classification report :				
	precision	recall	f1-score	support
0	0.87	0.83	0.85	1035
1	0.59	0.66	0.62	374
accuracy			0.79	1409
macro avg	0.73	0.75	0.74	1409
weighted avg	0.80	0.79	0.79	1409

Accuracy Score : 0.7877927608232789

Area under curve : 0.8479681727763569



Checking the model's performance on train data itself

```
In [87]: train_scores = cross_val_score(model, train_x, train_y, cv = 5, scoring='f1')
train_scores
```

```
Out[87]: array([0.60522273, 0.66346154, 0.62243286, 0.58139535, 0.63836478])
```

```
In [89]: np.mean(train_scores)
```

```
Out[89]: 0.6221754521655275
```

As we can see that the performance of the model on test data is same as training data. So, we can conclude that there is no overfitting and underfitting.

Saving model

```
In [78]: import pickle
pickle.dump(model, open('model.pkl', 'wb'))
```

Explaining the model

```
In [79]: import eli5
from eli5.sklearn import PermutationImportance

from pdpbox import pdp, info_plots
```

```
C:\Users\archd\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:144: FutureWarning: The sklearn.metrics.scorer module is deprecated in version 0.22 and will be removed in version 0.24. The corresponding classes / functions should instead be imported from sklearn.metrics. Anything that cannot be imported from sklearn.metrics is now part of the private API.
    warnings.warn(message, FutureWarning)
C:\Users\archd\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:144: FutureWarning: The sklearn.feature_selection.base module is deprecated in version 0.22 and will be removed in version 0.24. The corresponding classes / functions should instead be imported from sklearn.feature_selection. Anything that cannot be imported from sklearn.feature_selection is now part of the private API.
    warnings.warn(message, FutureWarning)
```

```
In [81]: perm = PermutationImportance(model, random_state=1).fit(test_x, test_y)
eli5.show_weights(perm, feature_names = test_x.columns.tolist())
```

Weight	Feature
0.0185 ± 0.0058	InternetService_Fiber optic
0.0064 ± 0.0088	Contract_Two year
0.0045 ± 0.0058	OnlineSecurity
0.0041 ± 0.0134	Contract_One year
0.0038 ± 0.0086	PaymentMethod_Electronic check
0.0037 ± 0.0071	InternetService_No
0.0028 ± 0.0094	tenure
0.0026 ± 0.0011	OnlineBackup
0.0020 ± 0.0078	MonthlyCharges
0.0010 ± 0.0014	DeviceProtection
0.0009 ± 0.0083	PaperlessBilling
0.0007 ± 0.0030	TechSupport
0.0004 ± 0.0032	StreamingMovies
0.0003 ± 0.0017	gender
0.0001 ± 0.0019	PhoneService
-0.0000 ± 0.0009	MultipleLines
-0.0001 ± 0.0006	StreamingTV
-0.0004 ± 0.0044	SeniorCitizen
-0.0009 ± 0.0033	Dependents
-0.0020 ± 0.0026	PaymentMethod_Credit card (automatic)
-0.0040 ± 0.0064	TotalCharges
-0.0040 ± 0.0039	Partner
-0.0075 ± 0.0033	PaymentMethod_Mailed check

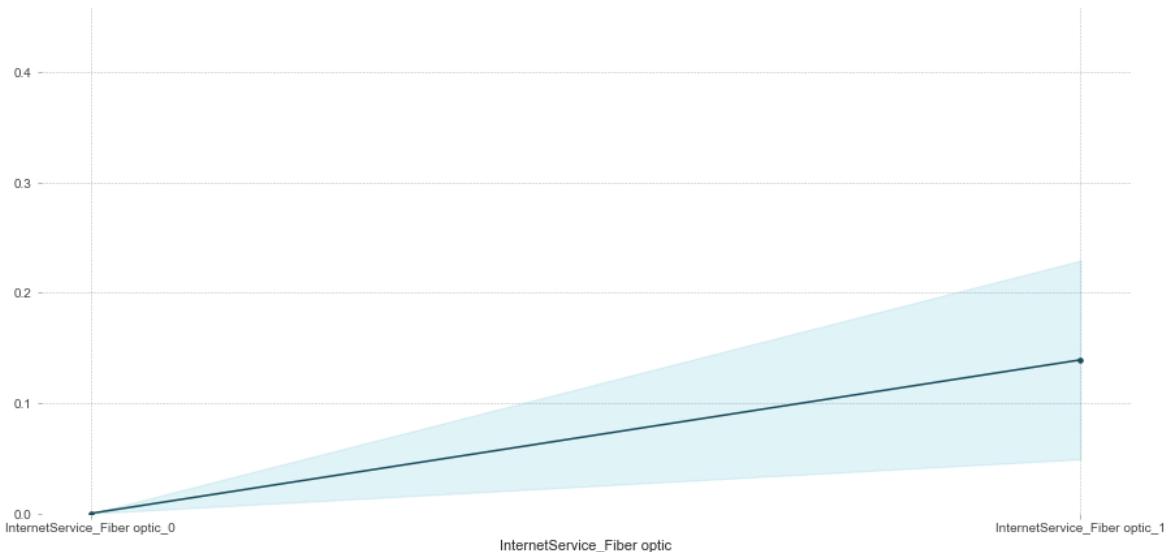
Visualizing how the partial dependence plots look for top features

Internet Service: Fiber Optic

```
In [95]: pdp_p = pdp.pdp_isolate(model=model, dataset=test_x, model_features=test_x.columns,
                             feature='InternetService_Fiber optic')
```

```
pdp.pdp_plot(pdp_p, 'InternetService_Fiber optic')
plt.show()
```

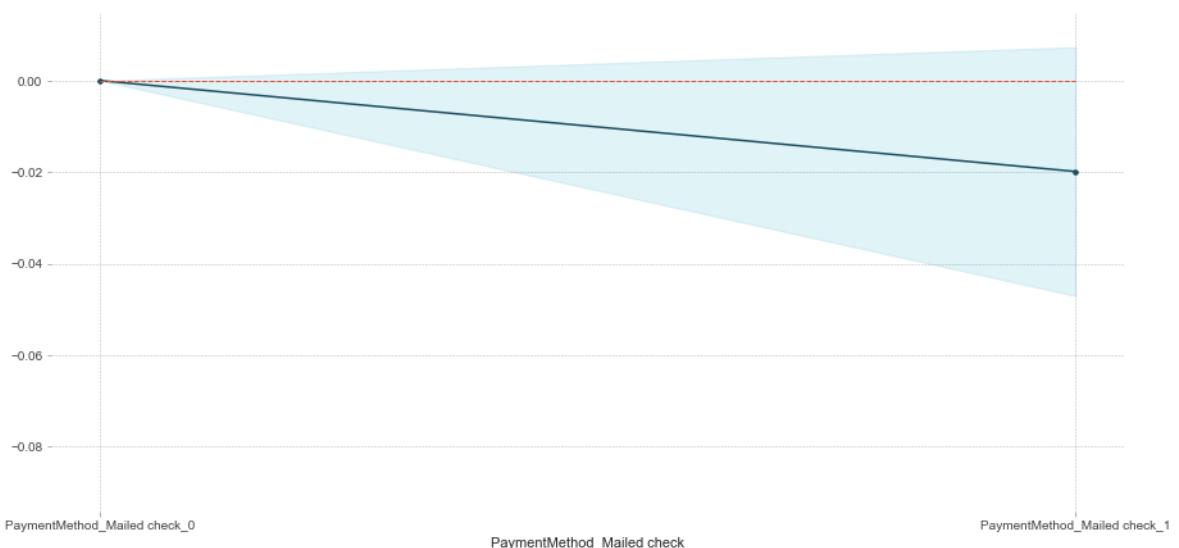
PDP for feature "InternetService_Fiber optic"
Number of unique grid points: 2



Payment Method: Mailed Check

```
In [97]: pdp_p = pdp.pdp_isolate(model=model, dataset=test_x, model_features=test_x.columns,
                               feature='PaymentMethod_Mailed check')
pdp.pdp_plot(pdp_p, 'PaymentMethod_Mailed check')
plt.show()
```

PDP for feature "PaymentMethod_Mailed check"
Number of unique grid points: 2



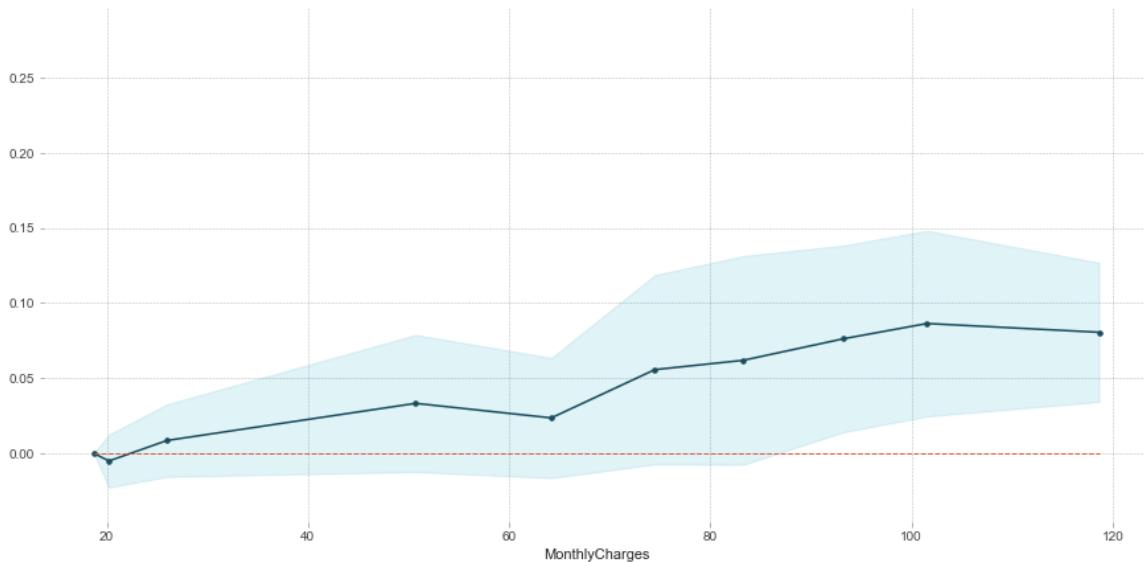
Monthly Charges

```
In [100...]: pdp_p = pdp.pdp_isolate(model=model, dataset=test_x, model_features=test_x.columns)
pdp.pdp_plot(pdp_p, 'MonthlyCharges')
```

```
plt.show()
```

PDP for feature "MonthlyCharges"

Number of unique grid points: 10

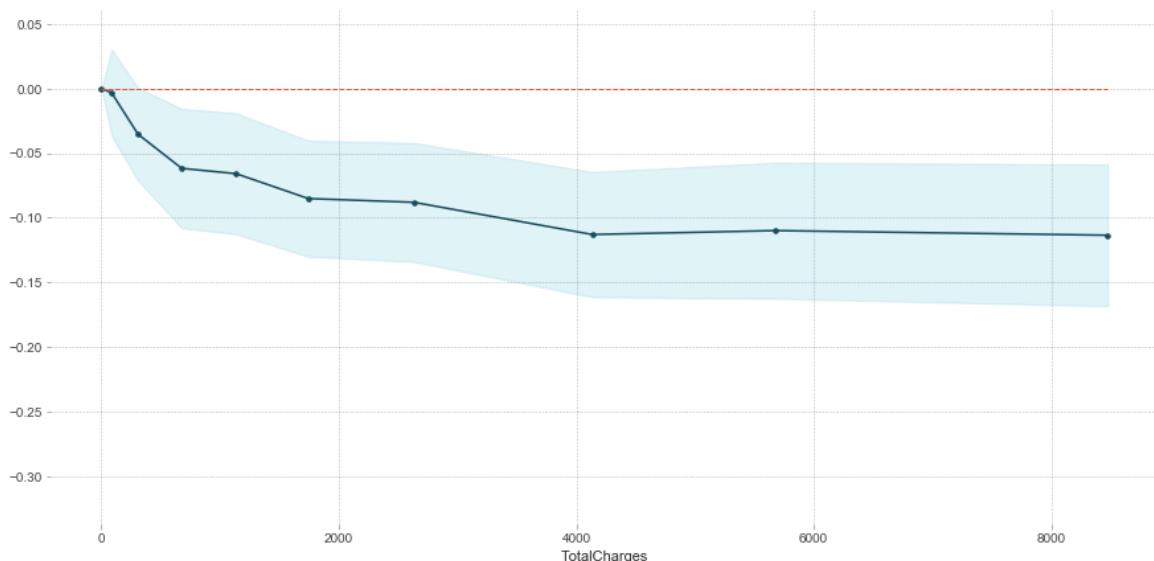


Total Charges

```
In [96]: pdp_p = pdp.pdp_isolate(model=model, dataset=test_x, model_features=test_x.columns)
pdp.pdp_plot(pdp_p, 'TotalCharges')
plt.show()
```

PDP for feature "TotalCharges"

Number of unique grid points: 10

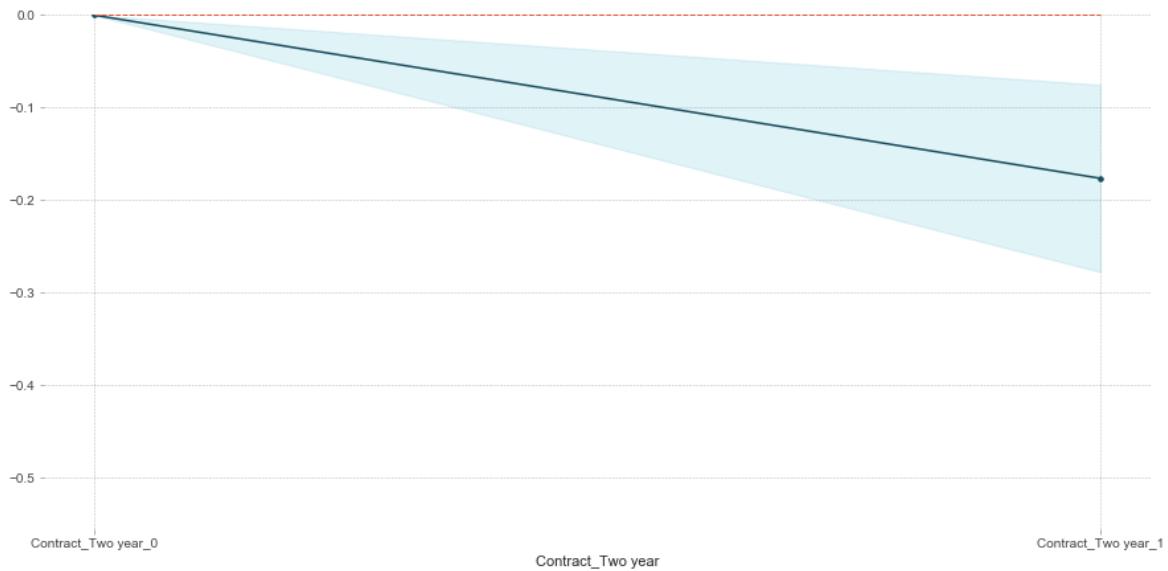


Contract - Two years

```
In [98]: pdp_p = pdp.pdp_isolate(model=model, dataset=test_x, model_features=test_x.columns,
                               feature='Contract_Two year')
```

```
pdp.pdp_plot(pdp_p, 'Contract_Two year')
plt.show()
```

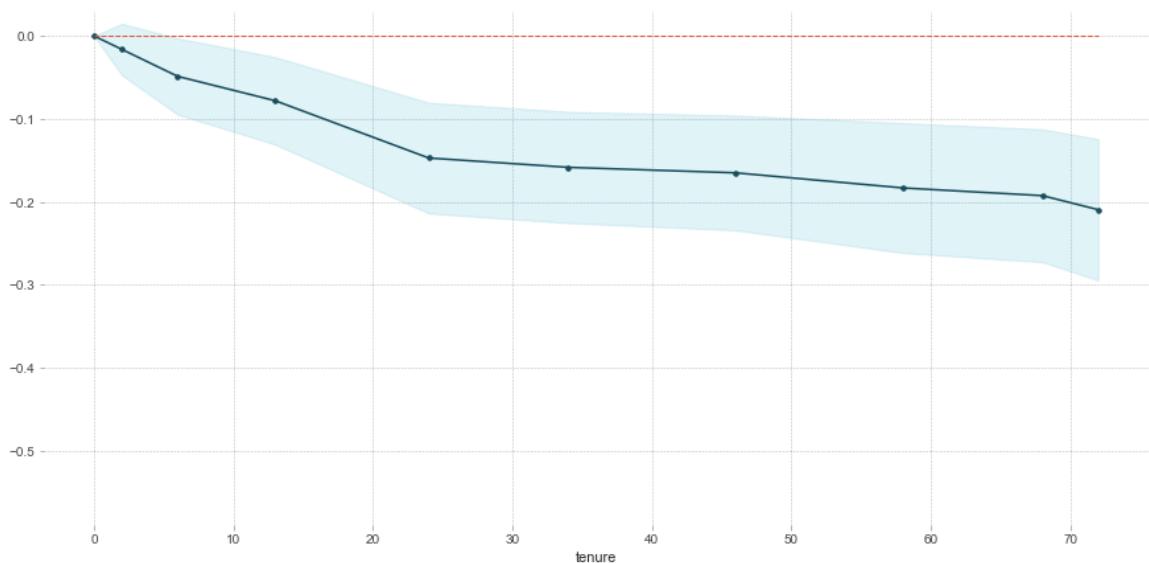
PDP for feature "Contract_Two year"
Number of unique grid points: 2



Tenure

```
In [99]: pdp_p = pdp.pdp_isolate(model=model, dataset=test_x, model_features=test_x.columns,
                               feature='tenure')
pdp.pdp_plot(pdp_p, 'tenure')
plt.show()
```

PDP for feature "tenure"
Number of unique grid points: 10



Shap Values

```
In [102...  
import shap  
shap.initjs()  
  
import joblib
```



```
In [104...  
explainer = shap.TreeExplainer(model)  
  
shap_values = explainer.shap_values(np.array(test_x.iloc[0]))  
shap.force_plot(explainer.expected_value[1], shap_values[1], test_x.iloc[0])
```



```
In [105...  
# Saving Explainer  
ex_filename = 'explainer.bz2'  
joblib.dump(explainer, filename=ex_filename, compress=('bz2', 9))
```

Out[105...]

```
In [106...  
explainer = joblib.load(filename="explainer.bz2")  
shap_values = explainer.shap_values(np.array(test_x.iloc[0]))  
shap.force_plot(explainer.expected_value[1], shap_values[1], list(test_x.columns))
```



Gauge Chart

```
In [107...  
from matplotlib.patches import Circle, Wedge, Rectangle  
  
def degree_range(n):  
    start = np.linspace(0, 180, n+1, endpoint=True)[0:-1]  
    end = np.linspace(0, 180, n+1, endpoint=True)[1::]  
    mid_points = start + ((end-start)/2.)  
    return np.c_[start, end], mid_points  
  
def rot_text(ang):  
    rotation = np.degrees(np.radians(ang) * np.pi / np.pi - np.radians(90))  
    return rotation  
  
def gauge(labels=['LOW', 'MEDIUM', 'HIGH', 'EXTREME'], \  
         colors=['#007A00', '#0063BF', '#FFCC00', '#ED1C24'], Probability=1, fname  
N = len(labels)
```

```

colors = colors[::-1]

"""
begins the plotting
"""

fig, ax = plt.subplots()

ang_range, mid_points = degree_range(4)

labels = labels[::-1]

"""
plots the sectors and the arcs
"""

patches = []
for ang, c in zip(ang_range, colors):
    # sectors
    patches.append(Wedge((0.,0.), .4, *ang, facecolor='w', lw=2))
    # arcs
    patches.append(Wedge((0.,0.), .4, *ang, width=0.10, facecolor=c, lw=2, a

[ax.add_patch(p) for p in patches]

"""
set the labels (e.g. 'LOW','MEDIUM',...
"""

for mid, lab in zip(mid_points, labels):

    ax.text(0.35 * np.cos(np.radians(mid)), 0.35 * np.sin(np.radians(mid)),
            horizontalalignment='center', verticalalignment='center', fontsize=1
            fontweight='bold', rotation = rot_text(mid))

"""
set the bottom banner and the title
"""

r = Rectangle((-0.4,-0.1),0.8,0.1, facecolor='w', lw=2)
ax.add_patch(r)

ax.text(0, -0.05, 'Churn Probability ' + np.round(Probability,2).astype(str)
        verticalalignment='center', fontsize=22, fontweight='bold')

"""
plots the arrow now
"""

pos = (1-Probability)*180
ax.arrow(0, 0, 0.225 * np.cos(np.radians(pos)), 0.225 * np.sin(np.radians(po
    width=0.04, head_width=0.09, head_length=0.1, fc='k', ec='k')

ax.add_patch(Circle((0, 0), radius=0.02, facecolor='k'))
ax.add_patch(Circle((0, 0), radius=0.01, facecolor='w', zorder=11))

"""
removes frame and ticks, and makes axis equal and tight
"""

ax.set_frame_on(False)

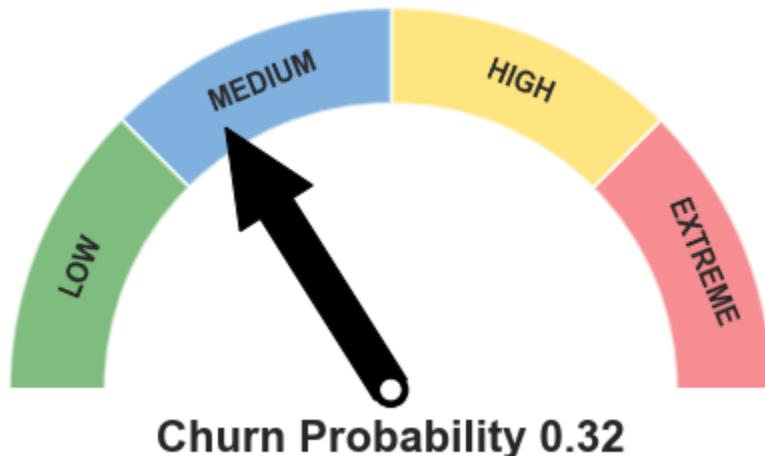
```

```

ax.axes.set_xticks([])
ax.axes.set_yticks([])
ax.axis('equal')
plt.tight_layout()
if fname:
    fig.savefig(fname, dpi=200)

```

In [109...]: gauge(Probability=model.predict_proba(test_x.iloc[0:1])[0,1])



Final Features

In [103...]: test_x.columns

Out[103...]: Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
 'PhoneService', 'MultipleLines', 'OnlineSecurity', 'OnlineBackup',
 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
 'PaperlessBilling', 'MonthlyCharges', 'TotalCharges',
 'InternetService_Fiber optic', 'InternetService_No',
 'Contract_One year', 'Contract_Two year',
 'PaymentMethod_Credit card (automatic)',
 'PaymentMethod_Electronic check', 'PaymentMethod_Mailed check'],
 dtype='object')

In []: