**Predicting a Burglary and finding the best model with cost benefit analysis**

**Objective: Find the best classification model for finding burglaries**

Create a classification model using random forest to predict BurgStatus.

* Combine the prediction results with the Logistic Regression and Decision Tree Model results.

**Section 2: Import Libraries**

In [1]:

*# Code Block 1*

​

**import** pandas **as** pd

**import** numpy **as** np

**import** sklearn

**import** matplotlib

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

​

**from** sklearn.model\_selection **import** train\_test\_split, cross\_val\_score, GridSearchCV, RandomizedSearchCV

**from** sklearn **import** preprocessing

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.metrics **import** precision\_recall\_curve, f1\_score, make\_scorer, ConfusionMatrixDisplay, confusion\_matrix, classification\_report, precision\_score, recall\_score, accuracy\_score, roc\_auc\_score, roc\_curve

**from** sklearn.ensemble **import** RandomForestClassifier

**from** sklearn.feature\_selection **import** SelectFromModel

**from** sklearn.preprocessing **import** OneHotEncoder

**from** sklearn.compose **import** ColumnTransformer

**from** sklearn.pipeline **import** Pipeline

**from** sklearn.tree **import** DecisionTreeClassifier

​

**%**matplotlib inline

​

**import** warnings

warnings.filterwarnings('ignore')

​

pd.set\_option('display.max\_columns',1000)

plt.style.use('seaborn')

​

plt.style.use('seaborn-v0\_8-colorblind')

​

sns.set(style**=**"whitegrid")

**Section 3: Import Data**

In [2]:

*# Code Block 2*

​

*# Import the datasets*

df\_all **=** pd.read\_csv('data/CantonPoliceDept\_HW05.csv')

df\_all.head()

Out[2]:

|  | **Index** | **Subzone** | **YEAR\_WEEK** | **SUB\_YEAR\_WEEK** | **All\_comp** | **FalseAlarm\_comp** | **Arrest\_comp** | **Cleared\_comp** | **NoContact\_comp** | **NoReport\_comp** | **Resolved\_comp** | **Filed\_comp** | **Calls\_comp** | **BurgAlarm\_comp** | **Suspicious\_comp** | **Shots\_comp** | **Intox\_comp** | **Drugs\_comp** | **Assault\_comp** | **Armed\_comp** | **Disturb\_comp** | **Fireworks\_comp** | **Noise\_comp** | **Stalking\_comp** | **ActualBurg** | **BurgStatus** | **BurgStatus2** | **call\_ALL\_target** | **call\_Burglary\_target** | **call\_ALL** | **Friday** | **Monday** | **Saturday** | **Sunday** | **Thursday** | **Tuesday** | **Wednesday** | **month\_1** | **month\_2** | **month\_3** | **month\_4** | **month\_5** | **month\_6** | **month\_7** | **month\_8** | **month\_9** | **month\_10** | **month\_11** | **month\_12** | **call\_Armed subject** | **call\_Assault** | **call\_Burglar alarm** | **call\_Burglary** | **call\_Disturbance** | **call\_Domestic** | **call\_FW FIREWORKS** | **call\_Fight** | **call\_Loitering** | **call\_Message delivery** | **call\_Noise complaint** | **call\_Possible drugs present** | **call\_Prowler**call\_Public Intoxicationcall\_Shots firedcall\_Special residence patrolcall\_Special watchcall\_Stalkingcall\_Suspicious activitycall\_Trespassingcall\_Vandalismcall\_Welfare checkdisp\_15-ARRESTdisp\_66-CANCELdisp\_9-VERBALdisp\_ADV-ADVICE GIVENdisp\_AST-ASSISTdisp\_CI-CITY CITATIONdisp\_CLR-CLEARED BY DISPATCHdisp\_CM-CIVIL MATTERdisp\_COMPdisp\_DISRdisp\_Disregarddisp\_Domestic Violence Reportdisp\_FA-FALSE ALARMdisp\_FD-FOUNDED ALARMdisp\_FI-FIELD INTERVIEWdisp\_Gone on Arrivaldisp\_IRT Reportdisp\_K9 Reportdisp\_MC-STATE MISD CITATIONdisp\_MDT-BOLO'D MDTdisp\_MSG-MESSAGE DELIVEREDdisp\_N25-NO CONTACTdisp\_NR-NO REPORTdisp\_Otherdisp\_PR-PROPERTY RETURNEDdisp\_RES-RESOLVEDdisp\_RF-REPORT FILEDdisp\_Runaway juvenile (entered NCIC)disp\_SAT-SETTLED AMONG SELVESdisp\_TES-TESTdisp\_TI -TOW INdisp\_Truancydisp\_VA Hospital Alarm (Fire)SqFootageHousingUnitsPopulationPopulation\_MalePopulation\_FemaleWorkedWorkers who travel to workDrove alone to WorkCarpooled to WorkPopulation\_3andoverEnrolled in schoolEnrolled in nursery school, preschoolEnrolled in kindergartenEnrolled in college, undergraduate yearsGraduate or professional schoolNot enrolled in schoolHouseholds\_earningsHouseholds\_wageorsalaryincomeHouseholds\_selfemploymentincomeHouseholds\_interest\_dividendsHouseholds\_SSIHouseholds\_publicassistanceincomeMedianAge\_TotalMedianAge\_MaleMedianAge\_FemaleHouseholdIncome\_MedianHouseholdIncome\_Median\_25to44HouseholdIncome\_Median\_65andoverHouseholdIncome\_Median\_45to64Income\_PerCapita |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | ZONE1A | 2009\_1 | 2009\_1\_ZONE1A | 182 | 7 | 14 | 0 | 21 | 49 | 21 | 21 | 196 | 14 | 56 | 14 | 0 | 7 | 14 | 0 | 35 | 0 | 7 | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00000000000000000000000000000000000000000002.196943e+077728660.4811580.51884210970.9671830.8304470.05834127630.2642060.0209920.0000000.0318490.0275060.7357949050.9558010.1314920.1712710.1812150.0287293327402599431089.00000437462767916862 |
| **1** | 1 | ZONE1B | 2009\_1 | 2009\_1\_ZONE1B | 144 | 9 | 18 | 0 | 9 | 63 | 9 | 18 | 153 | 18 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00000000000000000000000000000000000000000002.330865e+076834170.5750660.42493420250.9841980.6809880.22518532950.3010620.0088010.0045520.1811840.0406680.69893812851.0000000.0622570.1782100.0217900.0132303738345205047114.00000813165582722271 |
| **2** | 2 | ZONE1C | 2009\_1 | 2009\_1\_ZONE1C | 230 | 30 | 10 | 0 | 10 | 80 | 10 | 20 | 260 | 10 | 40 | 20 | 0 | 10 | 0 | 0 | 30 | 0 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00000000000000000000000000000000000000000006.501904e+06306260.3226840.6773163171.0000000.8075710.1640386060.1452150.0445540.0000000.0231020.0000000.8547852291.0000000.2969430.0567690.1659390.0218344537462215065078.29167180683479214181 |
| **3** | 3 | ZONE1D | 2009\_1 | 2009\_1\_ZONE1D | 189 | 0 | 18 | 9 | 27 | 63 | 0 | 27 | 189 | 9 | 27 | 18 | 9 | 9 | 18 | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00000000000000000000000000000000000000000001.306158e+0818245220.5075190.49248123660.9885880.8837700.08748943580.2673240.0110140.0068840.1092240.0286830.73267616571.0000000.0422450.1665660.0645750.0506943128323322535873.00000320844443320857 |
| **4** | 4 | ZONE2A | 2009\_1 | 2009\_1\_ZONE2A | 36 | 4 | 0 | 0 | 4 | 12 | 12 | 4 | 36 | 0 | 16 | 0 | 0 | 0 | 0 | 0 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00000000000000000000000000000000000000000006.732244e+077528910.5786920.4213089731.0000000.8509760.12333028460.1658470.0000000.0144060.0319750.0042160.8341535811.0000000.1067130.0929430.1841650.1153183839374593750703.00000273335473322852 |

**Section 4: Create the datasets**

**4.1 Create the X and y datasets**

In [3]:

*# Code Block 3*

​

*# Using the df\_all dataframe, create the X and y datasets.*

​

*# Identify non-numeric columns in X\_train that could cause a conversion error*

non\_numeric\_columns **=** df\_all.select\_dtypes(include**=**['object']).columns

​

*# Display the non-numeric columns to understand which may need to be excluded*

non\_numeric\_columns

Out[3]:

Index(['Subzone', 'YEAR\_WEEK', 'SUB\_YEAR\_WEEK'], dtype='object')

In [4]:

*# Code Block 4*

​

*# Create the X dataset with some non-numeric columns excluded, along with other unnecessary columns.*

X **=** df\_all.drop(['Subzone', 'YEAR\_WEEK', 'SUB\_YEAR\_WEEK', 'call\_ALL\_target', 'call\_Burglary\_target', 'ActualBurg',

'BurgStatus', 'BurgStatus2'], axis**=**1)

In [5]:

*# Code Block 5*

​

*# Create the target variable 'y' using the 'BurgStatus' column.*

y **=** df\_all['BurgStatus']

​

(X.shape)

Out[5]:

(11845, 126)

**4.2 Create the training and test datasets**

In [6]:

*# Code Block 6*

​

*# Split the data for training and test datasets.*

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.30, random\_state**=**42)

​

*# Display the shapes of the training and test sets to ensure they are split correctly.*

(X\_train.shape, y\_train.shape), (X\_test.shape, y\_test.shape)

Out[6]:

(((8291, 126), (8291,)), ((3554, 126), (3554,)))

**4.3 Create the scaled versions of X\_train and X\_test datasets**

In [7]:

*# Code Block 7*

​

*# Scale the data using Standard Scaler.*

​

*# Initialize the scaler*

sc **=** preprocessing.StandardScaler()

​

*# Standardize the data*

sc.fit(X\_train)

X\_train\_sc **=** sc.transform(X\_train)

X\_train\_sc **=** pd.DataFrame(X\_train\_sc, columns**=**X\_train.columns)

X\_test\_sc **=** sc.transform(X\_test)

X\_test\_sc **=** pd.DataFrame(X\_test\_sc, columns**=**X\_test.columns)

**Section 5: Classification using Random Forest**

***Objective:***

Use Random Forest to create an OPTIMAL model (highest accuracy) and a DECISION model (based on precision and recall).

**5.1 Using Random Forest create a model using the raw data**

In [8]:

*# Code Block 8*

​

*# Create a Defined Function for building a model*

**def** modeltraintest(vartrain, vartest, y\_train, y\_test, model):

​

*#1) Set the properties for the model (model) - by setting vartrain, vartest, and model*

*#2) Fit the model with training data*

model.fit(vartrain, y\_train)

​

*#3) Predict the target variable with test data*

model\_pred **=** model.predict(vartest)

model\_prob **=** model.predict\_proba(vartest)

​

*#4) Assess the accuracy with the test data*

score **=** model.score(vartest, y\_test)

​

print('XXXXXXXXXXXXXXXX ACCURACY SCORE XXXXXXXXXXXXXXXXXX')

print(round(score, 6))

print("")

​

​

print('XXXXXXXXXXXXXXXX CONFUSION MATRIX XXXXXXXXXXXXXXXX')

print(confusion\_matrix(y\_test, model\_pred))

print("")

​

​

print('XXXXXXXXXXXXXX CLASSIFICATION REPORT XXXXXXXXXXXXXX')

print(classification\_report(y\_test, model\_pred))

print('')

​

​

print('XXXXXXXXXXXXXX ROC AUC SCORE AND CHART XXXXXXXXXXXXXXXXXX')

print('')

y\_pred\_prob **=** model.predict\_proba(vartest)[:,1]

​

fpr, tpr, thresholds **=** roc\_curve(y\_test, y\_pred\_prob)

​

plt.plot([0, 1], [0, 1],'k--')

plt.plot(fpr, tpr, label**=**'Classification Model')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.show();

​

*# calculate roc curve*

y\_pred\_prob **=** model.predict\_proba(vartest)[:,1]

fpr, tpr, thresholds **=** roc\_curve(y\_test, y\_pred\_prob)

roc\_auc **=** roc\_auc\_score(y\_test, y\_pred\_prob)

roc\_auc\_format **=** 'ROC AUC Score: {0:.4f}'.format(roc\_auc)

print(roc\_auc\_format)

print('')

​

​

print('XXXXXXXXXXXXXX CROSS VALIDATION XXXXXXXXXXXXXXXXXX')

print('')

cv\_scores **=** cross\_val\_score(model, vartrain, y\_train, cv**=**5,

scoring**=**'accuracy')

print('CV Accuracy Scores:')

print(cv\_scores)

print('')

cv\_rocauc **=** cross\_val\_score(model, vartrain, y\_train, cv**=**5,

scoring**=**'roc\_auc')

print('CV ROC AUC:')

print(cv\_rocauc)

​

print('')

print('XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX')

In [9]:

*# Code Block 9*

​

*# Create a short version of the modeling results*

**def** shorttraintest(vartrain, vartest, y\_train, y\_test, model):

​

*#Fit the model*

model.fit(vartrain, y\_train)

​

*#Predict with the model*

model\_pred **=** model.predict(vartest)

model\_prob **=** model.predict\_proba(vartest)

​

​

print('Confusion Matrix:')

print(confusion\_matrix(y\_test, model\_pred))

print("")

​

*#Assess with the model*

score **=** model.score(vartest, y\_test)

score\_format **=** 'Accuracy Score: {0:.4f}'.format(score)

print(score\_format)

​

recall **=** recall\_score(y\_test, model\_pred)

recall\_format **=** 'Recall Score: {0:.4f}'.format(recall)

print(recall\_format)

precision **=** precision\_score(y\_test, model\_pred)

precision\_format **=** 'Precision Score: {0:.4f}'.format(precision)

print(precision\_format)

*# calculate roc curve*

y\_pred\_prob **=** model.predict\_proba(vartest)[:,1]

fpr, tpr, thresholds **=** roc\_curve(y\_test, y\_pred\_prob)

roc\_auc **=** roc\_auc\_score(y\_test, y\_pred\_prob)

roc\_auc\_format **=** 'ROC AUC Score: {0:.4f}'.format(roc\_auc)

print(roc\_auc\_format)

print('')

In [10]:

*# Code Block 10*

​

*# Run a Random Forest Classifier with default parameters*

vartrain **=** X\_train

vartest **=** X\_test

model **=** RandomForestClassifier(random\_state**=**21)

​

modeltraintest(vartrain, vartest, y\_train, y\_test, model)

XXXXXXXXXXXXXXXX ACCURACY SCORE XXXXXXXXXXXXXXXXXX

0.814856

XXXXXXXXXXXXXXXX CONFUSION MATRIX XXXXXXXXXXXXXXXX

[[1969 270]

[ 388 927]]

XXXXXXXXXXXXXX CLASSIFICATION REPORT XXXXXXXXXXXXXX

precision recall f1-score support

0 0.84 0.88 0.86 2239

1 0.77 0.70 0.74 1315

accuracy 0.81 3554

macro avg 0.80 0.79 0.80 3554

weighted avg 0.81 0.81 0.81 3554

XXXXXXXXXXXXXX ROC AUC SCORE AND CHART XXXXXXXXXXXXXXXXXX

A graph with a line

Description automatically generated

ROC AUC Score: 0.8846

XXXXXXXXXXXXXX CROSS VALIDATION XXXXXXXXXXXXXXXXXX

CV Accuracy Scores:

[0.83001808 0.81845597 0.81544029 0.81182147 0.82086852]

CV ROC AUC:

[0.89030606 0.8716189 0.88491541 0.87422474 0.88502245]

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

In [11]:

*# Code Block 11*

​

vartrain **=** X\_train

vartest **=** X\_test

model **=** RandomForestClassifier(random\_state**=**21)

​

shorttraintest(vartrain, vartest, y\_train, y\_test, model)

Confusion Matrix:

[[1969 270]

[ 388 927]]

Accuracy Score: 0.8149

Recall Score: 0.7049

Precision Score: 0.7744

ROC AUC Score: 0.8846

In [12]:

*# Code Block 12*

​

*# Set the testing data to the training data (vartrain and y\_train) to see how much overfit.*

vartrain **=** X\_train

vartest **=** X\_test

model **=** RandomForestClassifier(random\_state**=**21)

​

shorttraintest(vartrain, vartrain, y\_train, y\_train, model)

Confusion Matrix:

[[5337 0]

[ 0 2954]]

Accuracy Score: 1.0000

Recall Score: 1.0000

Precision Score: 1.0000

ROC AUC Score: 1.0000

**5.2 Fine-tune the model to find the OPTIMAL model**

***Manually Setting the Properties for Random Forest***

In [13]:

*# Code Block 13*

*# With no set max\_depth it can overfit on the training data*

​

*# Use a list instead of a range*

depth **=** [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22]

​

*#Creates an empty list*

scores **=** []

​

**for** d **in** depth:

classifier**=**RandomForestClassifier(max\_depth **=** d, random\_state **=** 21)

classifier**=**classifier.fit(X\_train,y\_train)

score **=** classifier.score(X\_test, y\_test)

scores.append(classifier.score(X\_test, y\_test))

print("iteration {} done. Accuracy = ".format(d) **+** str(score))

​

​

plt.plot(depth, scores, '-o')

plt.xlabel('depth, d')

plt.ylabel('scores')

plt.xticks(depth)

plt.show()

iteration 2 done. Accuracy = 0.7608328643781654

iteration 4 done. Accuracy = 0.795160382667417

iteration 6 done. Accuracy = 0.810917276308385

iteration 8 done. Accuracy = 0.810917276308385

iteration 10 done. Accuracy = 0.8145751266178953

iteration 12 done. Accuracy = 0.8187957231288688

iteration 14 done. Accuracy = 0.8137310073157006

iteration 16 done. Accuracy = 0.8165447383230163

iteration 18 done. Accuracy = 0.8140123804164322

iteration 20 done. Accuracy = 0.814856499718627

iteration 22 done. Accuracy = 0.8159819921215532

A graph with a line going up

Description automatically generated

In [14]:

*# Code Block 14*

​

vartrain **=** X\_train

vartest **=** X\_test

model **=** RandomForestClassifier(max\_depth **=** 12, random\_state**=**21)

​

shorttraintest(vartrain, vartrain, y\_train, y\_train, model)

Confusion Matrix:

[[5195 142]

[ 325 2629]]

Accuracy Score: 0.9437

Recall Score: 0.8900

Precision Score: 0.9488

ROC AUC Score: 0.9870

In [15]:

*# Code block 15*

​

*# Manually set max depth of 12*

*# Find max\_features*

figsize**=**(20, 5)

maxf **=** range(1,25)

scores **=** []

​

**for** d **in** maxf:

classifier**=**RandomForestClassifier(max\_depth **=** 12, max\_features **=** d, random\_state**=**21)

classifier**=**classifier.fit(X\_train,y\_train)

score **=** classifier.score(X\_test, y\_test)

scores.append(classifier.score(X\_test, y\_test))

print("iteration {} done. Accuracy = ".format(d) **+** str(score))

​

​

plt.plot(maxf, scores, '-o')

plt.xlabel('maxf, d')

plt.ylabel('scores')

plt.xticks(maxf)

plt.show()

iteration 1 done. Accuracy = 0.7630838491840181

iteration 2 done. Accuracy = 0.7754642656162071

iteration 3 done. Accuracy = 0.7943162633652223

iteration 4 done. Accuracy = 0.8007878446820484

iteration 5 done. Accuracy = 0.7996623522791221

iteration 6 done. Accuracy = 0.8047270680922903

iteration 7 done. Accuracy = 0.806978052898143

iteration 8 done. Accuracy = 0.806978052898143

iteration 9 done. Accuracy = 0.812886888013506

iteration 10 done. Accuracy = 0.8131682611142375

iteration 11 done. Accuracy = 0.8187957231288688

iteration 12 done. Accuracy = 0.8165447383230163

iteration 13 done. Accuracy = 0.8179516038266742

iteration 14 done. Accuracy = 0.8159819921215532

iteration 15 done. Accuracy = 0.8176702307259426

iteration 16 done. Accuracy = 0.8193584693303321

iteration 17 done. Accuracy = 0.8173888576252111

iteration 18 done. Accuracy = 0.8190770962296005

iteration 19 done. Accuracy = 0.8171074845244795

iteration 20 done. Accuracy = 0.8196398424310636

iteration 21 done. Accuracy = 0.8159819921215532

iteration 22 done. Accuracy = 0.8190770962296005

iteration 23 done. Accuracy = 0.8179516038266742

iteration 24 done. Accuracy = 0.8171074845244795

A graph with a line going up

Description automatically generated

In [16]:

*# Code Block 16*

​

vartrain **=** X\_train

vartest **=** X\_test

model **=** RandomForestClassifier(max\_depth **=** 12, max\_features **=** 20, random\_state**=**21)

​

shorttraintest(vartrain, vartrain, y\_train, y\_train, model)

Confusion Matrix:

[[5194 143]

[ 341 2613]]

Accuracy Score: 0.9416

Recall Score: 0.8846

Precision Score: 0.9481

ROC AUC Score: 0.9851

In [17]:

*# Code block 17*

​

*# Select n\_estimators:*

figsize**=**(20, 5)

est **=** [1, 20, 50, 100, 150, 500]

scores **=** []

​

**for** d **in** est:

classifier**=**RandomForestClassifier(max\_depth **=** 12, max\_features **=** 20, n\_estimators **=** d, random\_state**=**21)

classifier**=**classifier.fit(X\_train,y\_train)

score **=** classifier.score(X\_test, y\_test)

scores.append(classifier.score(X\_test, y\_test))

print("iteration {} done. Accuracy = ".format(d) **+** str(score))

​

​

plt.plot(est, scores, '-o')

plt.xlabel('est, d')

plt.ylabel('scores')

plt.xticks(est)

plt.show()

iteration 1 done. Accuracy = 0.7779966235227912

iteration 20 done. Accuracy = 0.8168261114237478

iteration 50 done. Accuracy = 0.8187957231288688

iteration 100 done. Accuracy = 0.8196398424310636

iteration 150 done. Accuracy = 0.8182329769274057

iteration 500 done. Accuracy = 0.8179516038266742

A graph with a line going up

Description automatically generated

In [18]:

*# Code Block 18*

​

vartrain **=** X\_train

vartest **=** X\_test

model **=** RandomForestClassifier(max\_depth **=** 12, max\_features **=** 20, n\_estimators **=** 100, random\_state**=**21)

​

shorttraintest(vartrain, vartrain, y\_train, y\_train, model)

Confusion Matrix:

[[5194 143]

[ 341 2613]]

Accuracy Score: 0.9416

Recall Score: 0.8846

Precision Score: 0.9481

ROC AUC Score: 0.9851

**5.3 Fine-tune the model to find the DECISION model**

In [19]:

*# Code block 19*

​

*# Run the model using various class weights*

​

cw **=** [**None**, 'balanced', {0:1, 1:2}, {0:1, 1:3}, {0:1, 1:5}, {0:1, 1:10},

{0:1, 1:20}, {0:1, 1:25}, {0:1, 1:30}, {0:1, 1:50}, {0:1, 1:100}]

​

vartrain **=** X\_train

vartest **=** X\_test

​

**for** w **in** cw:

print('----------------------')

vartitle **=** "Model with Class Weight: " **+** str(w)

varcw **=** w

model **=** RandomForestClassifier(max\_depth **=** 12, max\_features **=** 20, n\_estimators **=** 100, class\_weight**=**varcw)

print(vartitle)

print('')

shorttraintest(vartrain, vartest, y\_train, y\_test, model)

​

​

print('----------------------')

----------------------

Model with Class Weight: None

Confusion Matrix:

[[1942 297]

[ 344 971]]

Accuracy Score: 0.8196

Recall Score: 0.7384

Precision Score: 0.7658

ROC AUC Score: 0.8874

----------------------

Model with Class Weight: balanced

Confusion Matrix:

[[1896 343]

[ 285 1030]]

Accuracy Score: 0.8233

Recall Score: 0.7833

Precision Score: 0.7502

ROC AUC Score: 0.8889

----------------------

Model with Class Weight: {0: 1, 1: 2}

Confusion Matrix:

[[1890 349]

[ 273 1042]]

Accuracy Score: 0.8250

Recall Score: 0.7924

Precision Score: 0.7491

ROC AUC Score: 0.8877

----------------------

Model with Class Weight: {0: 1, 1: 3}

Confusion Matrix:

[[1877 362]

[ 264 1051]]

Accuracy Score: 0.8239

Recall Score: 0.7992

Precision Score: 0.7438

ROC AUC Score: 0.8888

----------------------

Model with Class Weight: {0: 1, 1: 5}

Confusion Matrix:

[[1853 386]

[ 246 1069]]

Accuracy Score: 0.8222

Recall Score: 0.8129

Precision Score: 0.7347

ROC AUC Score: 0.8838

----------------------

Model with Class Weight: {0: 1, 1: 10}

Confusion Matrix:

[[1745 494]

[ 205 1110]]

Accuracy Score: 0.8033

Recall Score: 0.8441

Precision Score: 0.6920

ROC AUC Score: 0.8795

----------------------

Model with Class Weight: {0: 1, 1: 20}

Confusion Matrix:

[[1361 878]

[ 119 1196]]

Accuracy Score: 0.7195

Recall Score: 0.9095

Precision Score: 0.5767

ROC AUC Score: 0.8750

----------------------

Model with Class Weight: {0: 1, 1: 25}

Confusion Matrix:

[[1185 1054]

[ 95 1220]]

Accuracy Score: 0.6767

Recall Score: 0.9278

Precision Score: 0.5365

ROC AUC Score: 0.8719

----------------------

Model with Class Weight: {0: 1, 1: 30}

Confusion Matrix:

[[1095 1144]

[ 70 1245]]

Accuracy Score: 0.6584

Recall Score: 0.9468

Precision Score: 0.5211

ROC AUC Score: 0.8724

----------------------

Model with Class Weight: {0: 1, 1: 50}

Confusion Matrix:

[[ 865 1374]

[ 48 1267]]

Accuracy Score: 0.5999

Recall Score: 0.9635

Precision Score: 0.4797

ROC AUC Score: 0.8651

----------------------

Model with Class Weight: {0: 1, 1: 100}

Confusion Matrix:

[[ 699 1540]

[ 22 1293]]

Accuracy Score: 0.5605

Recall Score: 0.9833

Precision Score: 0.4564

ROC AUC Score: 0.8563

----------------------

In [20]:

*# Code Block 20*

​

vartrain **=** X\_train

vartest **=** X\_test

model **=** RandomForestClassifier(max\_depth **=** 12, max\_features **=** 9, n\_estimators **=** 20, class\_weight**=**{0: 1, 1: 3})

​

modeltraintest(vartrain, vartest, y\_train, y\_test, model)

XXXXXXXXXXXXXXXX ACCURACY SCORE XXXXXXXXXXXXXXXXXX

0.82386

XXXXXXXXXXXXXXXX CONFUSION MATRIX XXXXXXXXXXXXXXXX

[[1873 366]

[ 260 1055]]

XXXXXXXXXXXXXX CLASSIFICATION REPORT XXXXXXXXXXXXXX

precision recall f1-score support

0 0.88 0.84 0.86 2239

1 0.74 0.80 0.77 1315

accuracy 0.82 3554

macro avg 0.81 0.82 0.81 3554

weighted avg 0.83 0.82 0.83 3554

XXXXXXXXXXXXXX ROC AUC SCORE AND CHART XXXXXXXXXXXXXXXXXX

A graph with a line

Description automatically generated

ROC AUC Score: 0.8843

XXXXXXXXXXXXXX CROSS VALIDATION XXXXXXXXXXXXXXXXXX

CV Accuracy Scores:

[0.82218204 0.8106152 0.81966224 0.81182147 0.82086852]

CV ROC AUC:

[0.88312278 0.861743 0.87828122 0.87304253 0.8853959 ]

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

In [21]:

*# Code block 21*

​

grid**=**{"max\_depth" : [9, 11, 13, 15, 19, 21], "criterion": ['gini', 'entropy'],

"n\_estimators" : [20], "max\_features" : [11, 13, 17, 21],

"class\_weight": [**None**]}

model\_grid **=** RandomForestClassifier(random\_state**=**21, n\_jobs**=-**2)

model\_cv**=**GridSearchCV(model\_grid,grid,cv**=**5)

model\_cv.fit(vartrain,y\_train)

​

​

print("tuned hyperparameters :(best parameters) ",model\_cv.best\_params\_)

print("accuracy :",model\_cv.best\_score\_)

tuned hyperparameters :(best parameters) {'class\_weight': None, 'criterion': 'entropy', 'max\_depth': 9, 'max\_features': 17, 'n\_estimators': 20}

accuracy : 0.824869865797627

In [22]:

*# Code Block 22*

​

*# Set the X training and test datasets*

vartrain **=** X\_train

vartest **=** X\_test

​

*#Set the model properties*

model **=** RandomForestClassifier(max\_depth **=** 9, max\_features **=** 17, n\_estimators **=** 20, class\_weight**=None**, criterion **=** 'entropy', random\_state**=**21)

​

modeltraintest(vartrain, vartest, y\_train, y\_test, model)

XXXXXXXXXXXXXXXX ACCURACY SCORE XXXXXXXXXXXXXXXXXX

0.812887

XXXXXXXXXXXXXXXX CONFUSION MATRIX XXXXXXXXXXXXXXXX

[[1918 321]

[ 344 971]]

XXXXXXXXXXXXXX CLASSIFICATION REPORT XXXXXXXXXXXXXX

precision recall f1-score support

0 0.85 0.86 0.85 2239

1 0.75 0.74 0.74 1315

accuracy 0.81 3554

macro avg 0.80 0.80 0.80 3554

weighted avg 0.81 0.81 0.81 3554

XXXXXXXXXXXXXX ROC AUC SCORE AND CHART XXXXXXXXXXXXXXXXXX

A graph with a line

Description automatically generated

ROC AUC Score: 0.8822

XXXXXXXXXXXXXX CROSS VALIDATION XXXXXXXXXXXXXXXXXX

CV Accuracy Scores:

[0.82881254 0.81845597 0.82509047 0.8172497 0.83474065]

CV ROC AUC:

[0.89008267 0.87647035 0.88719499 0.87417638 0.89331776]

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

In [23]:

*# Code Block 23*

​

df\_fi **=** pd.DataFrame(model.feature\_importances\_)

df\_fi

Out[23]:

|  | **0** |
| --- | --- |
| **0** | 0.022846 |
| **1** | 0.032943 |
| **2** | 0.006295 |
| **3** | 0.004916 |
| **4** | 0.003388 |
| **...** | ... |
| **121** | 0.001552 |
| **122** | 0.024134 |
| **123** | 0.000836 |
| **124** | 0.011931 |
| **125** | 0.014692 |

126 rows × 1 columns

In [24]:

*# Code Block 24*

​

X\_names **=** pd.DataFrame(list(X.columns))

In [25]:

*# Code Block 25*

​

df\_feat\_imp **=** pd.concat([df\_fi, X\_names], axis **=** 1)

df\_feat\_imp.columns **=** ['Importance', 'Features']

df\_feat\_imp[df\_feat\_imp['Importance']**!=**0].sort\_values('Importance', ascending **=** **False**)

Out[25]:

|  | **Importance** | **Features** |
| --- | --- | --- |
| **44** | 0.204003 | call\_Burglary |
| **10** | 0.145120 | BurgAlarm\_comp |
| **89** | 0.067943 | disp\_RF-REPORT FILED |
| **9** | 0.046296 | Calls\_comp |
| **8** | 0.042989 | Filed\_comp |
| **...** | ... | ... |
| **73** | 0.000244 | disp\_Disregard |
| **93** | 0.000191 | disp\_TI -TOW IN |
| **80** | 0.000151 | disp\_K9 Report |
| **72** | 0.000054 | disp\_DISR |
| **18** | 0.000053 | Fireworks\_comp |

120 rows × 2 columns

**Section 6: Cost Benefit Analysis**

**6.1 Create a Summary Table**

In [35]:

*# Code Block 26*

​

model **=** [

*# Logistic Regression - 2 models*

(X\_train\_sc, X\_test\_sc, y\_train, 'log\_opt', 'Logistic',

LogisticRegression(random\_state**=**21, C**=**0.01, class\_weight**=**'balanced', penalty**=**'l2', solver**=**'lbfgs')),

(X\_train\_sc, X\_test\_sc, y\_train, 'log\_dec', 'Logistic',

LogisticRegression(C**=**0.1, class\_weight**=**{0: 1, 1: 4}, penalty**=**'l1', solver**=**'liblinear', random\_state**=**21)),

​

*# Decision Tree - 2 models*

(X\_train, X\_test, y\_train, 'dt\_opt', 'DecisionTree',

DecisionTreeClassifier(criterion**=**'entropy', max\_depth**=**10, max\_features**=None**, min\_samples\_leaf**=**4, min\_samples\_split**=**10)),

(X\_train, X\_test, y\_train, 'dt\_dec', 'DecisionTree',

DecisionTreeClassifier(criterion**=**'entropy', max\_depth**=**10, min\_samples\_leaf**=**4, random\_state**=**42)),

​

*# Random Forest - 2 models*

(X\_train, X\_test, y\_train, 'rf\_opt', 'RandomForest',

RandomForestClassifier(max\_depth**=**9, max\_features**=**17, n\_estimators**=**20, class\_weight**=None**, criterion**=**'entropy', random\_state**=**21)),

(X\_train, X\_test, y\_train, 'rf\_dec', 'RandomForest',

RandomForestClassifier(max\_depth**=**12, max\_features**=**20, n\_estimators**=**100, class\_weight**=**{0: 1, 1: 3}, random\_state**=**21))

]

​

cm\_all **=** pd.DataFrame(columns**=**['Actual', 'Model', 'Type', 'pred\_NoBurglary', 'pred\_Burglary', 'Score', 'Recall', 'Precision', 'F1'])

​

**for** tr, tst, yt, n, mod, m **in** model:

m.fit(tr, yt)

model\_pred **=** m.predict(tst)

model\_prob **=** m.predict\_proba(tst)

score **=** m.score(tst, y\_test)

score\_format **=** '{0:.4f}'.format(score)

​

recall **=** recall\_score(y\_test, model\_pred)

recall\_format **=** '{0:.4f}'.format(recall)

​

f1 **=** f1\_score(y\_test, model\_pred)

f1\_format **=** '{0:.4f}'.format(f1)

​

precision **=** precision\_score(y\_test, model\_pred)

precision\_format **=** '{0:.4f}'.format(precision)

​

y\_pred\_prob **=** m.predict\_proba(tst)[:,1]

fpr, tpr, thresholds **=** roc\_curve(y\_test, y\_pred\_prob)

​

*# Store these variables in a way that doesn't use exec() if possible, exec() is generally best avoided*

exec(f'fpr\_{n} = fpr')

exec(f'tpr\_{n} = tpr')

exec(f'thresholds\_{n} = thresholds')

exec(f'{n} = n')

​

*# Create confusion matrix and add 'Actual' column*

cm **=** pd.DataFrame(confusion\_matrix(y\_test, model\_pred), index**=**['no\_burglary', 'burglary'], columns**=**['pred\_NoBurglary', 'pred\_Burglary'])

cm.reset\_index(inplace**=True**)

cm.rename(columns**=**{'index': 'Actual'}, inplace**=True**)

​

cm['Model'] **=** mod

cm['Type'] **=** n

cm['Score'] **=** score\_format

cm['Recall'] **=** recall\_format

cm['Precision'] **=** precision\_format

cm['F1'] **=** f1\_format

​

*# Concatenate the current confusion matrix with the overall DataFrame*

cm\_all **=** pd.concat([cm\_all, cm], axis**=**0, ignore\_index**=True**)

​

print(n **+** " - Score: " **+** str(score\_format) **+**

" - Recall: " **+** str(recall\_format) **+**

" - Precision: " **+** str(precision\_format) **+**

" - F1: " **+** str(f1\_format))

print('------------------------------------------------------------------------')

​

*# Display the final DataFrame*

display(cm\_all)

print('------------------------------------------------------------------------')

log\_opt - Score: 0.8025 - Recall: 0.7110 - Precision: 0.7438 - F1: 0.7271

------------------------------------------------------------------------

log\_dec - Score: 0.7757 - Recall: 0.8829 - Precision: 0.6436 - F1: 0.7445

------------------------------------------------------------------------

dt\_opt - Score: 0.8044 - Recall: 0.7719 - Precision: 0.7199 - F1: 0.7450

------------------------------------------------------------------------

dt\_dec - Score: 0.8061 - Recall: 0.7741 - Precision: 0.7220 - F1: 0.7472

------------------------------------------------------------------------

rf\_opt - Score: 0.8129 - Recall: 0.7384 - Precision: 0.7515 - F1: 0.7449

------------------------------------------------------------------------

rf\_dec - Score: 0.8261 - Recall: 0.8061 - Precision: 0.7449 - F1: 0.7743

------------------------------------------------------------------------

|  | **Actual** | **Model** | **Type** | **pred\_NoBurglary** | **pred\_Burglary** | **Score** | **Recall** | **Precision** | **F1** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | no\_burglary | Logistic | log\_opt | 1917 | 322 | 0.8025 | 0.7110 | 0.7438 | 0.7271 |
| **1** | burglary | Logistic | log\_opt | 380 | 935 | 0.8025 | 0.7110 | 0.7438 | 0.7271 |
| **2** | no\_burglary | Logistic | log\_dec | 1596 | 643 | 0.7757 | 0.8829 | 0.6436 | 0.7445 |
| **3** | burglary | Logistic | log\_dec | 154 | 1161 | 0.7757 | 0.8829 | 0.6436 | 0.7445 |
| **4** | no\_burglary | DecisionTree | dt\_opt | 1844 | 395 | 0.8044 | 0.7719 | 0.7199 | 0.7450 |
| **5** | burglary | DecisionTree | dt\_opt | 300 | 1015 | 0.8044 | 0.7719 | 0.7199 | 0.7450 |
| **6** | no\_burglary | DecisionTree | dt\_dec | 1847 | 392 | 0.8061 | 0.7741 | 0.7220 | 0.7472 |
| **7** | burglary | DecisionTree | dt\_dec | 297 | 1018 | 0.8061 | 0.7741 | 0.7220 | 0.7472 |
| **8** | no\_burglary | RandomForest | rf\_opt | 1918 | 321 | 0.8129 | 0.7384 | 0.7515 | 0.7449 |
| **9** | burglary | RandomForest | rf\_opt | 344 | 971 | 0.8129 | 0.7384 | 0.7515 | 0.7449 |
| **10** | no\_burglary | RandomForest | rf\_dec | 1876 | 363 | 0.8261 | 0.8061 | 0.7449 | 0.7743 |
| **11** | burglary | RandomForest | rf\_dec | 255 | 1060 | 0.8261 | 0.8061 | 0.7449 | 0.7743 |

------------------------------------------------------------------------

Based on the summary table above, the best model in terms of performance metrics appears to be the Random Forest with 'rf\_dec' parameters. This model has the highest score (0.8261), which is a general measure of accuracy. It also has the highest recall (0.8061), indicating its ability to identify true positives effectively. Additionally, its precision (0.7449) is competitive, and it has the highest F1 score (0.7743), which is a harmonic mean of precision and recall and indicates a good balance between the two.

In [36]:

*# Code Block 27*

​

*# Set*

rf\_dec **=** RandomForestClassifier(max\_depth**=**12, max\_features**=**20, n\_estimators**=**100, class\_weight**=**{0: 1, 1: 3}, random\_state**=**21)

​

*# Fit*

rf\_dec.fit(X\_train, y\_train)

​

*# Predict*

y\_pred **=** rf\_dec.predict(X\_test)

​

*# Assess*

cm **=** confusion\_matrix(y\_test, y\_pred)

​

*# Plotting the confusion matrix*

plt.figure(figsize**=**(10, 7))

​

*# Using seaborn to create a more visually appealing confusion matrix*

sns.heatmap(cm, annot**=True**, fmt**=**'d', cmap**=**'Greens', cbar**=False**)

label\_font **=** {'size':'18'}

plt.xlabel('Predicted', fontdict**=**label\_font)

plt.ylabel('Actual', fontdict**=**label\_font)

plt.title('Confusion Matrix for RandomForest Model', fontdict**=**label\_font)

tick\_labels **=** ['No Burglary', 'Burglary']

plt.xticks([0.5, 1.5], tick\_labels, rotation**=**0, fontsize**=**14)

plt.yticks([0.5, 1.5], tick\_labels, rotation**=**0, fontsize**=**14)

plt.show()

A green and white squares with black numbers

Description automatically generated

**6.2 Understanding the additional officer hours for making a wrong decision**

For Random Forest DECISION:

* False Negative: 255 × 3 = 765 additional officer hours
* False Positive: 363 × 1=363 = 183 additional officer hours
* Total additional officer hours = 765 + 363 = 1,128
* Therefore, the Random Forest DECISION model will add an additional 1,128 officer hours.

**6.3 Creating the additional officer hours for making a wrong prediction**

In [52]:

*# Code Block 28*

​

*# Calculate the additional hours for false predictions*

​

*# noburglary\_hrs when actual is no burglary, otherwise 0*

cm\_all['noburglary\_hrs'] **=** np.where(cm\_all['Actual'] **==** 'no\_burglary', cm\_all['pred\_Burglary'] **\*** 1, 0).astype(int)

​

*# burglary\_hrs when actual is burglary, otherwise 0*

cm\_all['burglary\_hrs'] **=** np.where(cm\_all['Actual'] **==** 'burglary', cm\_all['pred\_NoBurglary'] **\*** 3, 0).astype(int)

​

cm\_all['pred\_NoBurglary'] **=** cm\_all['pred\_NoBurglary'].astype(float)

cm\_all['pred\_Burglary'] **=** cm\_all['pred\_Burglary'].astype(float)

cm\_all['Score'] **=** cm\_all['Score'].astype(float)

cm\_all['Recall'] **=** cm\_all['Recall'].astype(float)

cm\_all['Precision'] **=** cm\_all['Precision'].astype(float)

cm\_all['F1'] **=** cm\_all['F1'].astype(float)

cm\_all['noburglary\_hrs'] **=** cm\_all['noburglary\_hrs'].astype(int)

cm\_all['burglary\_hrs'] **=** cm\_all['burglary\_hrs'].astype(int)

​

cm\_all.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 12 entries, 0 to 11

Data columns (total 11 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Actual 12 non-null object

1 Model 12 non-null object

2 Type 12 non-null object

3 pred\_NoBurglary 12 non-null float64

4 pred\_Burglary 12 non-null float64

5 Score 12 non-null float64

6 Recall 12 non-null float64

7 Precision 12 non-null float64

8 F1 12 non-null float64

9 noburglary\_hrs 12 non-null int32

10 burglary\_hrs 12 non-null int32

dtypes: float64(6), int32(2), object(3)

memory usage: 1.1+ KB

**6.4 Group the additional officer hours for making a wrong prediction**

In [70]:

*# Code Block 29*

​

*# Calculate the sum noburglary\_hrs and burglary\_hrs (reset the index)*

cm\_all\_loss **=** cm\_all.groupby('Type')[['noburglary\_hrs', 'burglary\_hrs']].sum().reset\_index()

cm\_all\_loss

Out[70]:

|  | **Type** | **noburglary\_hrs** | **burglary\_hrs** |
| --- | --- | --- | --- |
| **0** | dt\_dec | 392 | 891 |
| **1** | dt\_opt | 395 | 900 |
| **2** | log\_dec | 643 | 462 |
| **3** | log\_opt | 322 | 1140 |
| **4** | rf\_dec | 363 | 765 |
| **5** | rf\_opt | 321 | 1032 |

In [71]:

*# Code Block 30*

​

*# Calculate the mean for Score, Recall, Precision, F1 (reset the index)*

cm\_all\_score **=** cm\_all.groupby('Type')[['Score', 'Recall', 'Precision', 'F1']].mean().reset\_index()

cm\_all\_score

Out[71]:

|  | **Type** | **Score** | **Recall** | **Precision** | **F1** |
| --- | --- | --- | --- | --- | --- |
| **0** | dt\_dec | 0.8061 | 0.7741 | 0.7220 | 0.7472 |
| **1** | dt\_opt | 0.8044 | 0.7719 | 0.7199 | 0.7450 |
| **2** | log\_dec | 0.7757 | 0.8829 | 0.6436 | 0.7445 |
| **3** | log\_opt | 0.8025 | 0.7110 | 0.7438 | 0.7271 |
| **4** | rf\_dec | 0.8261 | 0.8061 | 0.7449 | 0.7743 |
| **5** | rf\_opt | 0.8129 | 0.7384 | 0.7515 | 0.7449 |

In [72]:

*# Code Block 31*

​

*# Include the Model column by calculating the first record*

cm\_all\_model **=** cm\_all.groupby('Type')['Model'].first().reset\_index()

cm\_all\_model

Out[72]:

|  | **Type** | **Model** |
| --- | --- | --- |
| **0** | dt\_dec | DecisionTree |
| **1** | dt\_opt | DecisionTree |
| **2** | log\_dec | Logistic |
| **3** | log\_opt | Logistic |
| **4** | rf\_dec | RandomForest |
| **5** | rf\_opt | RandomForest |

In [73]:

*# Code Block 32*

​

*# Merge the three different calculated groupings together and name it model\_cost*

model\_cost **=** cm\_all\_loss.merge(cm\_all\_score, on**=**'Type').merge(cm\_all\_model, on**=**'Type')

​

*# For model\_cost, create a new column named pred\_hrs that adds noburglary\_hrs and burglary\_hrs together.*

model\_cost['pred\_hrs'] **=** model\_cost['noburglary\_hrs'] **+** model\_cost['burglary\_hrs']

model\_cost.head(1)

Out[73]:

|  | **Type** | **noburglary\_hrs** | **burglary\_hrs** | **Score** | **Recall** | **Precision** | **F1** | **Model** | **pred\_hrs** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | dt\_dec | 392 | 891 | 0.8061 | 0.7741 | 0.722 | 0.7472 | DecisionTree | 1283 |

**Section 7: Summarize and Visualize the Result**

In [76]:

*# Code Block 33*

​

model\_cost **=** model\_cost.sort\_values(by**=**'Model', ascending**=False**)

sns.set(style**=**"whitegrid")

plt.figure(figsize**=**(10,6))

ax **=** sns.barplot(y **=** "pred\_hrs", x **=** "Type", data **=** model\_cost, hue**=** 'Model', palette **=** 'deep', dodge**=False**)

ax.set\_xticklabels(ax.get\_xticklabels(), rotation**=**90, fontsize**=**'10')

plt.legend(loc**=**"upper right")

plt.title('Additional Officer Hours by Model Type')

plt.xlabel('Model Type')

plt.ylabel('Additional Officer Hours')

plt.xticks(rotation**=**45)

plt.tight\_layout()

A graph of different colored bars

Description automatically generated

In [77]:

*# Code Block 34*

​

model\_cost[model\_cost['Model']**==**'RandomForest']

Out[77]:

|  | **Type** | **noburglary\_hrs** | **burglary\_hrs** | **Score** | **Recall** | **Precision** | **F1** | **Model** | **pred\_hrs** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **5** | rf\_opt | 321 | 1032 | 0.8129 | 0.7384 | 0.7515 | 0.7449 | RandomForest | 1353 |
| **4** | rf\_dec | 363 | 765 | 0.8261 | 0.8061 | 0.7449 | 0.7743 | RandomForest | 1128 |

**Model Evaluation for Canton Police Department**

After performing an extensive analysis of various machine learning models, it has been determined that the Random Forest Decision model rf\_dec stands out as the best model for our needs.

**Why?**

* **Highest Overall Accuracy**: With a score of 0.8261, it has the best accuracy among the models tested.
* **Superior Recall**: At 0.8061, it is the most reliable model for identifying actual burglaries, which is crucial for public safety and resource allocation.
* **Balanced Precision and F1 Score**: Its precision and F1 score reflect a well-balanced model that correctly identifies burglaries without a high rate of false alarms.
* **Efficiency in Resource Utilization**: By reducing the additional officer hours needed for false predictions, rf\_dec allows the Canton Police Department to allocate their resources more effectively and potentially invest more in preventive measures.

In conclusion, adopting the rf\_dec model aligns with the department's goal of efficient and accurate burglary prediction, ensuring that officer hours are dedicated to genuine cases, ultimately enhancing the safety and security of the Canton community.

**Section 8: Predict Week 50**

In [84]:

*# Code Block 35*

​

*# Import the dataset named CantonPoliceDept\_HW05\_Week49.csv*

df\_canton\_new **=** pd.read\_csv('data/CantonPoliceDept\_HW05\_week49.csv')

df\_canton\_new.head()

Out[84]:

|  | **Index** | **Subzone** | **All\_comp** | **FalseAlarm\_comp** | **Arrest\_comp** | **Cleared\_comp** | **NoContact\_comp** | **NoReport\_comp** | **Resolved\_comp** | **Filed\_comp** | **Calls\_comp** | **BurgAlarm\_comp** | **Suspicious\_comp** | **Shots\_comp** | **Intox\_comp** | **Drugs\_comp** | **Assault\_comp** | **Armed\_comp** | **Disturb\_comp** | **Fireworks\_comp** | **Noise\_comp** | **Stalking\_comp** | **call\_ALL** | **Friday** | **Monday** | **Saturday** | **Sunday** | **Thursday** | **Tuesday** | **Wednesday** | **month\_1** | **month\_2** | **month\_3** | **month\_4** | **month\_5** | **month\_6** | **month\_7** | **month\_8** | **month\_9** | **month\_10** | **month\_11** | **month\_12** | **call\_Armed subject** | **call\_Assault** | **call\_Burglar alarm** | **call\_Burglary** | **call\_Disturbance** | **call\_Domestic** | **call\_FW FIREWORKS** | **call\_Fight** | **call\_Loitering** | **call\_Message delivery** | **call\_Noise complaint** | **call\_Possible drugs present** | **call\_Prowler** | **call\_Public Intoxication** | **call\_Shots fired** | **call\_Special residence patrol** | **call\_Special watch** | **call\_Stalking** | **call\_Suspicious activity** | **call\_Trespassing**call\_Vandalismcall\_Welfare checkdisp\_15-ARRESTdisp\_66-CANCELdisp\_9-VERBALdisp\_ADV-ADVICE GIVENdisp\_AST-ASSISTdisp\_CI-CITY CITATIONdisp\_CLR-CLEARED BY DISPATCHdisp\_CM-CIVIL MATTERdisp\_COMPdisp\_DISRdisp\_Disregarddisp\_Domestic Violence Reportdisp\_FA-FALSE ALARMdisp\_FD-FOUNDED ALARMdisp\_FI-FIELD INTERVIEWdisp\_Gone on Arrivaldisp\_IRT Reportdisp\_K9 Reportdisp\_MC-STATE MISD CITATIONdisp\_MDT-BOLO'D MDTdisp\_MSG-MESSAGE DELIVEREDdisp\_N25-NO CONTACTdisp\_NR-NO REPORTdisp\_Otherdisp\_PR-PROPERTY RETURNEDdisp\_RES-RESOLVEDdisp\_RF-REPORT FILEDdisp\_Runaway juvenile (entered NCIC)disp\_SAT-SETTLED AMONG SELVESdisp\_TES-TESTdisp\_TI -TOW INdisp\_Truancydisp\_VA Hospital Alarm (Fire)SqFootageHousingUnitsPopulationPopulation\_MalePopulation\_FemaleWorkedWorkers who travel to workDrove alone to WorkCarpooled to WorkPopulation\_3andoverEnrolled in schoolEnrolled in nursery school, preschoolEnrolled in kindergartenEnrolled in college, undergraduate yearsGraduate or professional schoolNot enrolled in schoolHouseholds\_earningsHouseholds\_wageorsalaryincomeHouseholds\_selfemploymentincomeHouseholds\_interest\_dividendsHouseholds\_SSIHouseholds\_publicassistanceincomeMedianAge\_TotalMedianAge\_MaleMedianAge\_FemaleHouseholdIncome\_MedianHouseholdIncome\_Median\_25to44HouseholdIncome\_Median\_65andoverHouseholdIncome\_Median\_45to64Income\_PerCapita |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | ZONE1A | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 154 | 0 | 21 | 0 | 0 | 0 | 0 | 0 | 21 | 0 | 0 | 0 | 22 | 6 | 2 | 1 | 6 | 0 | 5 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 21 | 0 | 0 | 5 | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 1183001001000005000000001600220000002.196943e+077728660.4811580.51884210970.9671830.8304470.05834127630.2642060.0209920.0000000.0318490.0275060.7357949050.9558010.1314920.1712710.1812150.0287293327402599431089.00000437462767916862 |
| **1** | 1 | ZONE1B | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 261 | 9 | 54 | 0 | 9 | 9 | 9 | 0 | 9 | 0 | 9 | 0 | 29 | 6 | 1 | 5 | 9 | 4 | 4 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 32 | 0 | 1 | 4 | 1 | 1 | 11 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 7 | 0118001021000004000000004200370000002.330865e+076834170.5750660.42493420250.9841980.6809880.22518532950.3010620.0088010.0045520.1811840.0406680.69893812851.0000000.0622570.1782100.0217900.0132303738345205047114.00000813165582722271 |
| **2** | 2 | ZONE1C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 300 | 0 | 100 | 10 | 0 | 0 | 10 | 0 | 50 | 0 | 10 | 0 | 30 | 2 | 9 | 2 | 3 | 3 | 6 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 29 | 0 | 1 | 3 | 0 | 1 | 2 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 11 | 10512004000000030000000021000520000006.501904e+06306260.3226840.6773163171.0000000.8075710.1640386060.1452150.0445540.0000000.0231020.0000000.8547852291.0000000.2969430.0567690.1659390.0218344537462215065078.29167180683479214181 |
| **3** | 3 | ZONE1D | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 333 | 0 | 54 | 18 | 27 | 0 | 9 | 0 | 9 | 0 | 0 | 9 | 37 | 5 | 2 | 5 | 5 | 6 | 5 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 37 | 0 | 1 | 9 | 0 | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 2 | 0 | 0 | 1 | 6 | 1162111201000009000000009400340000001.306158e+0818245220.5075190.49248123660.9885880.8837700.08748943580.2673240.0110140.0068840.1092240.0286830.73267616571.0000000.0422450.1665660.0645750.0506943128323322535873.00000320844443320857 |
| **4** | 4 | ZONE2A | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 132 | 0 | 44 | 4 | 4 | 4 | 0 | 0 | 16 | 0 | 4 | 0 | 33 | 8 | 3 | 2 | 3 | 7 | 6 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 33 | 0 | 0 | 2 | 0 | 2 | 5 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 11 | 21620000010000020000000021200950000006.732244e+077528910.5786920.4213089731.0000000.8509760.12333028460.1658470.0000000.0144060.0319750.0042160.8341535811.0000000.1067130.0929430.1841650.1153183839374593750703.00000273335473322852 |

In [96]:

*# Prepare the features for prediction*

X\_new **=** df\_canton\_new.drop(['Subzone', 'burgstatus'], axis**=**1)

y\_new **=** df\_canton\_new['burgstatus']

​

*# Initialize the RandomForestClassifier with the best parameters identified*

best\_model **=** RandomForestClassifier(max\_depth**=**12, max\_features**=**20, n\_estimators**=**100, class\_weight**=**{0: 1, 1: 3}, random\_state**=**21)

​

*# Fit the model on the previous data (not shown here) and predict on the new data*

best\_model.fit(X\_previous, y\_previous) *# X\_previous and y\_previous should be your training data used to train the model*

y\_pred **=** best\_model.predict(X\_new)

y\_pred\_prob **=** best\_model.predict\_proba(X\_new)

​

*# Display the confusion matrix*

cm **=** confusion\_matrix(y\_new, y\_pred)

print('Confusion Matrix:')

print(cm)

print("")

​

*# Display the classification report*

print('----------------------CLASSIFICATION REPORT----------------------')

print(classification\_report(y\_new, y\_pred))

print('-----------------------------------------------------------------')

​

*# Plot the confusion matrix using seaborn heatmap*

plt.figure(figsize**=**(8, 6))

sns.heatmap(cm, annot**=True**, cmap**=**"Blues", annot\_kws**=**{"size": 16}, fmt**=**"g")

plt.title('New Customers', fontweight**=**'bold', fontsize**=**'16', horizontalalignment**=**'center')

plt.xlabel('Predicted', fontsize**=**15)

plt.ylabel('True', fontsize**=**15)

plt.xticks([0.5, 1.5], ['Good', 'Bad'], fontsize**=**12)

plt.yticks([0.5, 1.5], ['Good', 'Bad'], fontsize**=**12, va**=**'center')

plt.show()

​

**---------------------------------------------------------------------------**

**KeyError** Traceback (most recent call last)

Cell **In[96], line 2**

1 # Prepare the features for prediction

**----> 2** X\_new = df\_canton\_new.drop(['Subzone', 'burgstatus'], axis=1)

3 y\_new = df\_canton\_new['burgstatus']

5 # Initialize the RandomForestClassifier with the best parameters identified

File **~\anaconda3\Lib\site-packages\pandas\core\frame.py:5258**, in DataFrame.drop**(self, labels, axis, index, columns, level, inplace, errors)**

5110 **def** drop(

5111 self,

5112 labels: IndexLabel = **None**,

**(...)**

5119 errors: IgnoreRaise = "raise",

5120 ) -> DataFrame | **None**:

5121 """

5122 Drop specified labels from rows or columns.

5123

**(...)**

5256 weight 1.0 0.8

5257 """

**-> 5258** **return** super().drop(

5259 labels=labels,

5260 axis=axis,

5261 index=index,

5262 columns=columns,

5263 level=level,

5264 inplace=inplace,

5265 errors=errors,

5266 )

File **~\anaconda3\Lib\site-packages\pandas\core\generic.py:4549**, in NDFrame.drop**(self, labels, axis, index, columns, level, inplace, errors)**

4547 **for** axis, labels **in** axes.items():

4548 **if** labels **is** **not** **None**:

**-> 4549** obj = obj.\_drop\_axis(labels, axis, level=level, errors=errors)

4551 **if** inplace:

4552 self.\_update\_inplace(obj)

File **~\anaconda3\Lib\site-packages\pandas\core\generic.py:4591**, in NDFrame.\_drop\_axis**(self, labels, axis, level, errors, only\_slice)**

4589 new\_axis = axis.drop(labels, level=level, errors=errors)

4590 **else**:

**-> 4591** new\_axis = axis.drop(labels, errors=errors)

4592 indexer = axis.get\_indexer(new\_axis)

4594 # Case for non-unique axis

4595 **else**:

File **~\anaconda3\Lib\site-packages\pandas\core\indexes\base.py:6699**, in Index.drop**(self, labels, errors)**

6697 **if** mask.any():

6698 **if** errors != "ignore":

**-> 6699** **raise** **KeyError**(f"**{**list(labels[mask])**}** not found in axis")

6700 indexer = indexer[~mask]

6701 **return** self.delete(indexer)

**KeyError**: "['burgstatus'] not found in axis"