Classification Model Comparison: NYC Shooting Incidents

DTSA 5509 Introduction to Machine Learning, University of Colorado, Boulder, April 2023

Required Packages

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import math
import seaborn as sns
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn import metrics
import time
```

Problem Description

This project will analyze the NYPD Shooting Incident Data data set in an attempt to classify shooting incidents as murders. Three different classification models will be built and the classification accuracy compared against various model parameters. The K-Nearest Neighbors (KNN), Random Forest (Forest) and Gradient Boosting (Boost) Classifiers from the sklearn package will be evaluated. The features of the data set will be converted from multi-value to binary and a feature importance analysis will be performed. A comparison of computation time will be included to assess the best balance between performance and time.

The dataset being used is the NYPD Shooting Incident Data (Historic). The source file is located at https://catalog.data.gov/dataset/nypd-shooting-incident-data-historic. The data contains information on every shooting incident that occurred in New York City from 1/1/2006 through 12/31/2021. The data includes features related to the location and time of incident, perpetrator and victim attributes and murder label. The murder label (SATISTICAL_MURDER_FLAG) indicates a shooting that resulted in the victim's death and is identified as a murder.

GITHUB Repository: https://github.com/jmskeet/DTSA-5509

EDA

Import Data and High Level Analysis

In [3]:
```
orginalDataSet = pd.read_csv('data/NYPD_Shooting_Incident_Data__Historic_.csv')
orginalDataSet.describe()
```

```
print("Shape: " + str(orginalDataSet.shape))
print("Row Count: " + str(len(orginalDataSet)))
print(orginalDataSet.head(5))
print(orginalDataSet.describe(percentiles = [], include='all'))
print(orginalDataSet.dtypes)
```

```
Shape: (25596, 19)
Row Count: 25596
   INCIDENT_KEY  OCCUR_DATE OCCUR_TIME      BORO  PRECINCT  JURISDICTION_CODE  \
0     236168668  11/11/2021   15:04:00  BROOKLYN        79                0.0
1     231008085  07/16/2021   22:05:00  BROOKLYN        72                0.0
2     230717903  07/11/2021   01:09:00  BROOKLYN        79                0.0
3     237712309  12/11/2021   13:42:00  BROOKLYN        81                0.0
4     224465521  02/16/2021   20:00:00    QUEENS       113                0.0

  LOCATION_DESC  STATISTICAL_MURDER_FLAG PERP_AGE_GROUP PERP_SEX  \
0           NaN                    False            NaN      NaN
1           NaN                    False          45-64        M
2           NaN                    False            <18        M
3           NaN                    False            NaN      NaN
4           NaN                    False            NaN      NaN

                  PERP_RACE VIC_AGE_GROUP VIC_SEX                  VIC_RACE  \
0                       NaN         18-24       M                     BLACK
1  ASIAN / PACIFIC ISLANDER         25-44       M  ASIAN / PACIFIC ISLANDER
2                     BLACK         25-44       M                     BLACK
3                       NaN         25-44       M                     BLACK
4                       NaN         25-44       M                     BLACK

   X_COORD_CD  Y_COORD_CD   Latitude  Longitude  \
0    996313.0    187499.0  40.681318 -73.956509
1    981845.0    171118.0  40.636364 -74.008667
2    996546.0    187436.0  40.681145 -73.955669
3   1001139.0    192775.0  40.695792 -73.939096
4   1050710.0    184826.0  40.673740 -73.760411

                                       Lon_Lat
0  POINT (-73.95650899099996 40.68131820000008)
1  POINT (-74.00866668999998 40.63636384100005)
2  POINT (-73.95566903799994 40.68114495900005)
3       POINT (-73.939095905 40.69579171600003)
4  POINT (-73.76041066999993 40.67374017600008)
        INCIDENT_KEY  OCCUR_DATE OCCUR_TIME      BORO      PRECINCT  \
count   2.559600e+04       25596      25596     25596  25596.000000
unique           NaN        5409       1411         5           NaN
top              NaN  07/05/2020   23:30:00  BROOKLYN           NaN
freq             NaN          47        171     10365           NaN
mean    1.123826e+08         NaN        NaN       NaN     65.869433
std     6.786117e+07         NaN        NaN       NaN     27.201904
min     9.953245e+06         NaN        NaN       NaN      1.000000
50%     8.643726e+07         NaN        NaN       NaN     69.000000
max     2.384901e+08         NaN        NaN       NaN    123.000000

        JURISDICTION_CODE             LOCATION_DESC STATISTICAL_MURDER_FLAG  \
count        25594.000000                     10619                   25596
unique                NaN                        39                       2
top                   NaN  MULTI DWELL - PUBLIC HOUS                   False
freq                  NaN                      4559                   20668
mean             0.331601                       NaN                     NaN
std              0.742266                       NaN                     NaN
min              0.000000                       NaN                     NaN
50%              0.000000                       NaN                     NaN
max              2.000000                       NaN                     NaN

       PERP_AGE_GROUP PERP_SEX PERP_RACE VIC_AGE_GROUP VIC_SEX VIC_RACE  \
count           16252    16286     16286         25596   25596    25596
```

```
unique             9        3       7             6       3       7
top            18-24        M   BLACK         25-44       M   BLACK
freq            5844    14416   10668         11386   23182   18281
mean             NaN      NaN     NaN           NaN     NaN     NaN
std              NaN      NaN     NaN           NaN     NaN     NaN
min              NaN      NaN     NaN           NaN     NaN     NaN
50%              NaN      NaN     NaN           NaN     NaN     NaN
max              NaN      NaN     NaN           NaN     NaN     NaN

          X_COORD_CD     Y_COORD_CD      Latitude     Longitude  \
count   2.559600e+04   25596.000000  25596.000000  25596.000000
unique           NaN            NaN           NaN           NaN
top              NaN            NaN           NaN           NaN
freq             NaN            NaN           NaN           NaN
mean    1.009455e+06  207893.776907     40.737250    -73.909039
std     1.842142e+04   31857.353942      0.087447      0.066427
min     9.149281e+05  125756.718750     40.511586    -74.249303
50%     1.007715e+06  194037.718750     40.699128    -73.915346
max     1.066815e+06  271127.687500     40.910818    -73.702046

                                            Lon_Lat
count                                         25596
unique                                        11472
top     POINT (-73.88151014499994 40.67141260500006)
freq                                             66
mean                                            NaN
std                                             NaN
min                                             NaN
50%                                             NaN
max                                             NaN
INCIDENT_KEY                 int64
OCCUR_DATE                  object
OCCUR_TIME                  object
BORO                        object
PRECINCT                     int64
JURISDICTION_CODE          float64
LOCATION_DESC               object
STATISTICAL_MURDER_FLAG       bool
PERP_AGE_GROUP              object
PERP_SEX                    object
PERP_RACE                   object
VIC_AGE_GROUP               object
VIC_SEX                     object
VIC_RACE                    object
X_COORD_CD                 float64
Y_COORD_CD                 float64
Latitude                   float64
Longitude                  float64
Lon_Lat                     object
dtype: object
```

High Level Observations

The following features contain NAs:

```
LOCATION_DESC
PERP_AGE_GROUP
PERP_SEX
```

```
PERP_RACE
```

The OCCUR_DATE and OCCUR_TIME are object types not datetime types.

The data will be assessed for the level of NA occurances.

In [4]:
```
nullValues = [np.nan, None, [], {}, 'NaN', 'Null','NULL','None','NA','?','-', '.','',
for c in orginalDataSet.columns:
    string_null = np.array([x in nullValues[2:] for x in orginalDataSet[c]])
    print(c, orginalDataSet[c].isnull().sum(), string_null.sum())
```

```
INCIDENT_KEY 0 0
OCCUR_DATE 0 0
OCCUR_TIME 0 0
BORO 0 0
PRECINCT 0 0
JURISDICTION_CODE 2 0
LOCATION_DESC 14977 0
STATISTICAL_MURDER_FLAG 0 0
PERP_AGE_GROUP 9344 0
PERP_SEX 9310 0
PERP_RACE 9310 0
VIC_AGE_GROUP 0 0
VIC_SEX 0 0
VIC_RACE 0 0
X_COORD_CD 0 0
Y_COORD_CD 0 0
Latitude 0 0
Longitude 0 0
Lon_Lat 0 0
```

Based on the NA assessment the following features can be thrown. (NA count > 5%)

```
LOCATION_DESC
PERP_AGE_GROUP
PERP_SEX
PERP_RACE
```

Addtionally, the following features are deememd either redundant or irrelevelvant to the analysis.

```
INCIDENT_KEY
PRECINCT
JURISDICTION_CODE
X_COORD_CD
Y_COORD_CD
Latitude
Longitude
Lon_Lat
```

In [5]:
```
featuresToThrow = ['INCIDENT_KEY', 'PRECINCT', 'JURISDICTION_CODE', 'X_COORD_CD', 'Y_(
orginalDataSet.drop(featuresToThrow, axis=1, inplace=True)
print("Shape: " + str(orginalDataSet.shape))
```

```
print("Row Count: " + str(len(orginalDataSet)))
print(orginalDataSet.dtypes)

print ('')
print('NA Removal Validation')
for c in orginalDataSet.columns:
    string_null = np.array([x in nullValues[2:] for x in orginalDataSet[c]])
    print(c, orginalDataSet[c].isnull().sum(), string_null.sum())
```

```
Shape: (25596, 7)
Row Count: 25596
OCCUR_DATE                 object
OCCUR_TIME                 object
BORO                       object
STATISTICAL_MURDER_FLAG      bool
VIC_AGE_GROUP              object
VIC_SEX                    object
VIC_RACE                   object
dtype: object

NA Removal Validation
OCCUR_DATE 0 0
OCCUR_TIME 0 0
BORO 0 0
STATISTICAL_MURDER_FLAG 0 0
VIC_AGE_GROUP 0 0
VIC_SEX 0 0
VIC_RACE 0 0
```

The OCCUR_DATE is converted to date time object and broken into month and day features. The OCCUR_TIME feature is broken out into hours of the day

In [6]:

```
orginalDataSet['OCCUR_DATE']= pd.to_datetime(orginalDataSet['OCCUR_DATE'])
orginalDataSet['hour'] = orginalDataSet['OCCUR_TIME'].astype(str).str[0]
orginalDataSet['hour'] = orginalDataSet['hour'].astype(int)
orginalDataSet['day'] = orginalDataSet['OCCUR_DATE'].dt.dayofweek
orginalDataSet['month'] = orginalDataSet['OCCUR_DATE'].dt.month
print(orginalDataSet.dtypes)
```

```
OCCUR_DATE                 datetime64[ns]
OCCUR_TIME                         object
BORO                               object
STATISTICAL_MURDER_FLAG              bool
VIC_AGE_GROUP                      object
VIC_SEX                            object
VIC_RACE                           object
hour                                int32
day                                 int64
month                               int64
dtype: object
```

Feature Value Assesment

Feature values will be evalauted via unique value and histogram assessment for completeness and correctness.

In [7]:

```
boroughs = orginalDataSet.BORO.unique()
murderFlag = orginalDataSet.STATISTICAL_MURDER_FLAG.unique()
```

```
victimAgeGroup = orginalDataSet.VIC_AGE_GROUP.unique()
victimSex = orginalDataSet.VIC_SEX.unique()
victimRace = orginalDataSet.VIC_RACE.unique()

print(boroughs)
print(murderFlag)
print(victimAgeGroup)
print(victimSex)
print(victimRace)
```

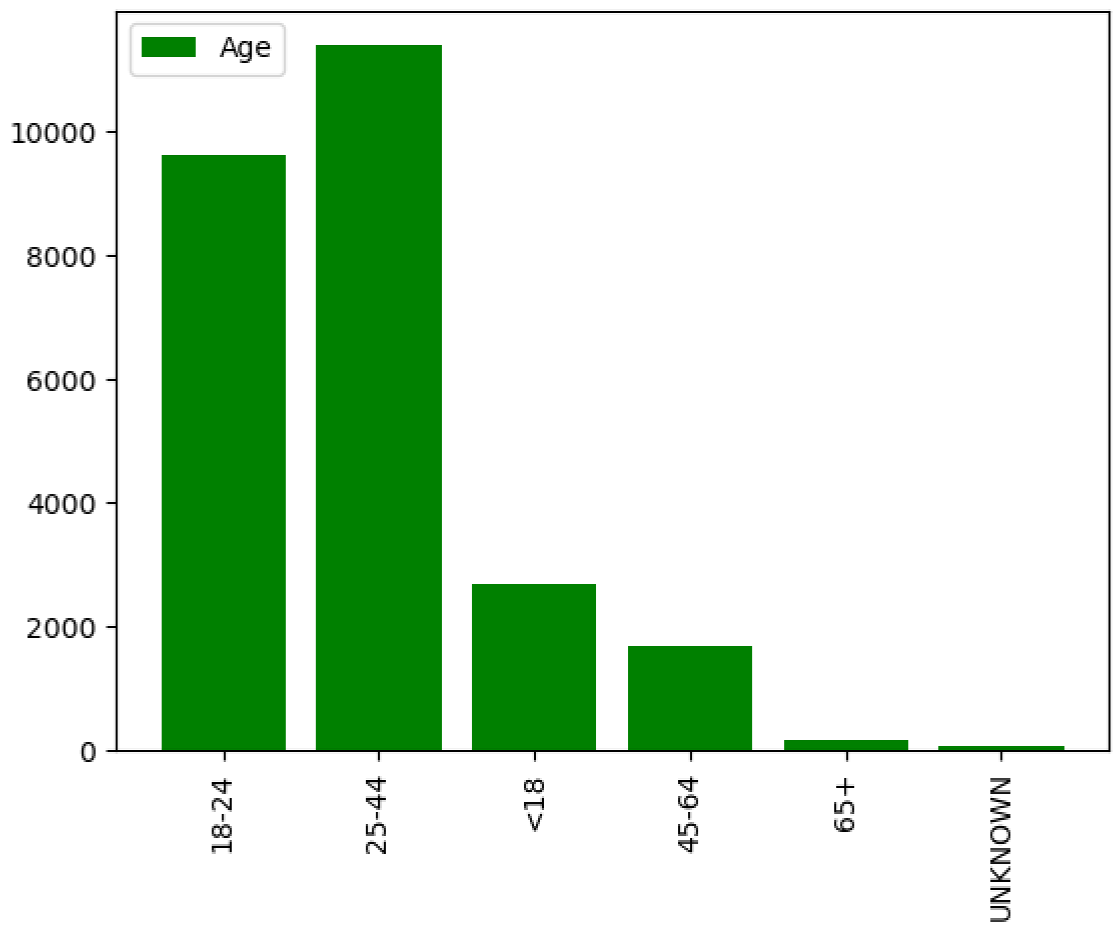
```
['BROOKLYN' 'QUEENS' 'BRONX' 'MANHATTAN' 'STATEN ISLAND']
[False  True]
['18-24' '25-44' '<18' '45-64' '65+' 'UNKNOWN']
['M' 'F' 'U']
['BLACK' 'ASIAN / PACIFIC ISLANDER' 'BLACK HISPANIC' 'WHITE HISPANIC'
 'WHITE' 'AMERICAN INDIAN/ALASKAN NATIVE' 'UNKNOWN']
```

In [8]:

```
plt.xticks(rotation='vertical')
ageHistogram = list(orginalDataSet['VIC_AGE_GROUP'])
ageDict = {x:ageHistogram.count(x) for x in ageHistogram}
plt.bar(ageDict.keys(), ageDict.values(), color='g')
plt.legend(['Age'], loc=2)
```

Out[8]:

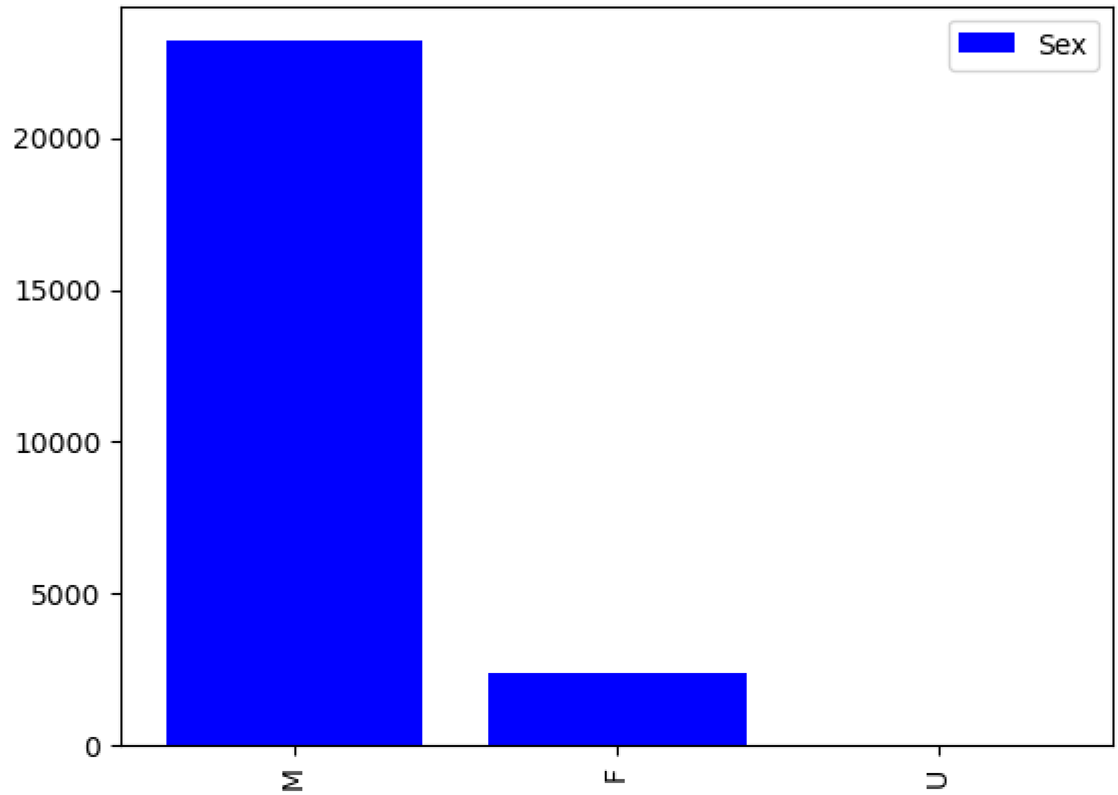
```
<matplotlib.legend.Legend at 0x1c98a1bf2b0>
```



In [9]:

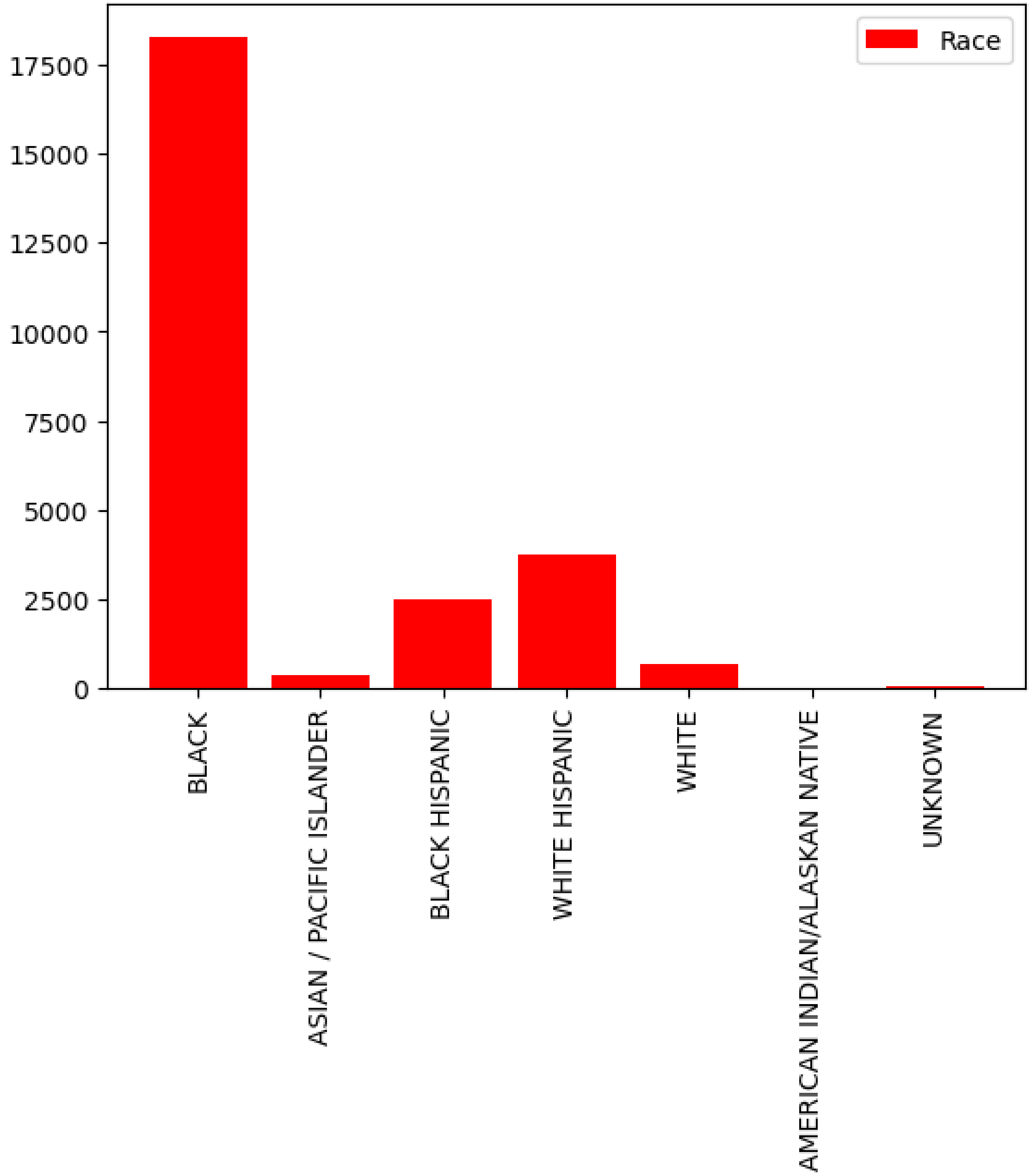
```
plt.xticks(rotation='vertical')
sexHistogram = list(orginalDataSet['VIC_SEX'])
sexDict = {x:sexHistogram.count(x) for x in sexHistogram}
plt.bar(sexDict.keys(), sexDict.values(), color='b')
plt.legend(["Sex"], loc=1)
```

Out[9]: <matplotlib.legend.Legend at 0x1c98be22ce0>



In [10]:
```
plt.xticks(rotation='vertical')
raceHistogram = list(orginalDataSet['VIC_RACE'])
raceDict = {x:raceHistogram.count(x) for x in raceHistogram}
plt.bar(raceDict.keys(), raceDict.values(), color='r')
plt.legend(["Race"], loc=1)
```

Out[10]: <matplotlib.legend.Legend at 0x1c99fee2d10>



The feature value analysis showed that VIC_AGE_GROUP, VIC_SEX, VIC_RACE contained U and UNKNOWN Values. The degree to which these values impact the data set was assessed via percent of observations and the above histograms.

In [8]:
```
unknownAgeRows = orginalDataSet.apply(lambda x: True if x['VIC_AGE_GROUP'] == 'UNKNOWN
print('Unknown Age Rows: ' + str(len(unknownAgeRows[unknownAgeRows == True].index)))
print('Percent of Rows: ' + str(len(unknownAgeRows[unknownAgeRows == True].index)/len(

unknownSexRows = orginalDataSet.apply(lambda x: True if x['VIC_SEX'] == 'U' else False
print('Unknown Sex Rows: ' + str(len(unknownSexRows[unknownSexRows == True].index)))
print('Percent of Rows: ' + str(len(unknownSexRows[unknownSexRows == True].index)/len(

unknownRaceRows = orginalDataSet.apply(lambda x: True if x['VIC_RACE'] == 'UNKNOWN' el
print('Unknown Age Rows: ' + str(len(unknownRaceRows[unknownRaceRows == True].index)))
print('Percent of Rows: ' + str(len(unknownRaceRows[unknownRaceRows == True].index)/le
```

```
Unknown Age Rows: 60
Percent of Rows: 0.0023441162681669013
Unknown Sex Rows: 11
Percent of Rows: 0.0004297546491639319
Unknown Age Rows: 65
Percent of Rows: 0.0025394592905141427
```

The results revealed that <.5% of the observations are impacted and can be removed. Removal was validated.

In [9]:
```
indexAge = orginalDataSet[ (orginalDataSet['VIC_AGE_GROUP'] == 'UNKNOWN')].index
orginalDataSet.drop(indexAge, inplace=True)

indexSex = orginalDataSet[ (orginalDataSet['VIC_SEX'] == 'U')].index
orginalDataSet.drop(indexSex, inplace=True)

indexRace = orginalDataSet[ (orginalDataSet['VIC_RACE'] == 'UNKNOWN')].index
orginalDataSet.drop(indexRace, inplace=True)

#Verfiy rows removed
victimAgeGroup = orginalDataSet.VIC_AGE_GROUP.unique()
victimSex = orginalDataSet.VIC_SEX.unique()
victimRace = orginalDataSet.VIC_RACE.unique()

print(victimAgeGroup)
print(victimSex)
print(victimRace)

print("Row Count: " + str(len(orginalDataSet)))
```

```
['18-24' '25-44' '<18' '45-64' '65+']
['M' 'F']
['BLACK' 'ASIAN / PACIFIC ISLANDER' 'BLACK HISPANIC' 'WHITE HISPANIC'
 'WHITE' 'AMERICAN INDIAN/ALASKAN NATIVE']
Row Count: 25482
```

Build Classification Dataframe

With the orginal data set cleansed, the dataset was converted from multi-value features to binary features. The result being an increase of features from 10 to 61. For illustration purposes the first 10 rows of the classification data set are provided.

In [10]:
```
classificationData = pd.DataFrame()

classificationData['murder'] = orginalDataSet['STATISTICAL_MURDER_FLAG']
#0 for female, 1 for male
classificationData['sex'] = [False if x == 'F' else True for x in orginalDataSet['day'

#Map multivalue features to boolean feature columns
#day of week
for d in range(7):
    header = 'd' + str(d)
    classificationData[header] = [True if x == d else False for x in orginalDataSet['c
#hour of day
for h in range(24):
    header = 'h' + str(h)
    classificationData[header] = [True if x == h else False for x in orginalDataSet['h
```

```
#month of year
for m in range(1,13):
    header = 'm' + str(m)
    classificationData[header] = [True if x == m else False for x in orginalDataSet['r

for b in boroughs:
    classificationData[b] = [True if x == b else False for x in orginalDataSet['BORO']

for r in victimRace:
    classificationData[r] = [True if x == r else False for x in orginalDataSet['VIC_RA

for a in victimAgeGroup :
    classificationData[a] = [True if x == a else False for x in orginalDataSet['VIC_AG


classificationData.info()
print(classificationData.head(10))
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25482 entries, 0 to 25595
Data columns (total 61 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   murder                          25482 non-null  bool
 1   sex                             25482 non-null  bool
 2   d0                              25482 non-null  bool
 3   d1                              25482 non-null  bool
 4   d2                              25482 non-null  bool
 5   d3                              25482 non-null  bool
 6   d4                              25482 non-null  bool
 7   d5                              25482 non-null  bool
 8   d6                              25482 non-null  bool
 9   h0                              25482 non-null  bool
 10  h1                              25482 non-null  bool
 11  h2                              25482 non-null  bool
 12  h3                              25482 non-null  bool
 13  h4                              25482 non-null  bool
 14  h5                              25482 non-null  bool
 15  h6                              25482 non-null  bool
 16  h7                              25482 non-null  bool
 17  h8                              25482 non-null  bool
 18  h9                              25482 non-null  bool
 19  h10                             25482 non-null  bool
 20  h11                             25482 non-null  bool
 21  h12                             25482 non-null  bool
 22  h13                             25482 non-null  bool
 23  h14                             25482 non-null  bool
 24  h15                             25482 non-null  bool
 25  h16                             25482 non-null  bool
 26  h17                             25482 non-null  bool
 27  h18                             25482 non-null  bool
 28  h19                             25482 non-null  bool
 29  h20                             25482 non-null  bool
 30  h21                             25482 non-null  bool
 31  h22                             25482 non-null  bool
 32  h23                             25482 non-null  bool
 33  m1                              25482 non-null  bool
 34  m2                              25482 non-null  bool
 35  m3                              25482 non-null  bool
 36  m4                              25482 non-null  bool
 37  m5                              25482 non-null  bool
 38  m6                              25482 non-null  bool
 39  m7                              25482 non-null  bool
 40  m8                              25482 non-null  bool
 41  m9                              25482 non-null  bool
 42  m10                             25482 non-null  bool
 43  m11                             25482 non-null  bool
 44  m12                             25482 non-null  bool
 45  BROOKLYN                        25482 non-null  bool
 46  QUEENS                          25482 non-null  bool
 47  BRONX                           25482 non-null  bool
 48  MANHATTAN                       25482 non-null  bool
 49  STATEN ISLAND                   25482 non-null  bool
 50  BLACK                           25482 non-null  bool
 51  ASIAN / PACIFIC ISLANDER        25482 non-null  bool
 52  BLACK HISPANIC                  25482 non-null  bool
 53  WHITE HISPANIC                  25482 non-null  bool
 54  WHITE                           25482 non-null  bool
```

```
 55  AMERICAN INDIAN/ALASKAN NATIVE  25482 non-null  bool
 56  18-24                           25482 non-null  bool
 57  25-44                           25482 non-null  bool
 58  <18                             25482 non-null  bool
 59  45-64                           25482 non-null  bool
 60  65+                             25482 non-null  bool
dtypes: bool(61)
memory usage: 1.7 MB
   murder   sex     d0     d1     d2     d3     d4     d5     d6     h0  ...  \
0   False  True  False  False  False   True  False  False  False  False  ...
1   False  True  False  False  False  False   True  False  False  False  ...
2   False  True  False  False  False  False  False  False   True   True  ...
3   False  True  False  False  False  False  False   True  False  False  ...
4   False  True  False   True  False  False  False  False  False  False  ...
5    True  True  False  False  False  False  False   True  False   True  ...
6    True  True  False  False   True  False  False  False  False  False  ...
7   False  True  False  False  False  False   True  False  False  False  ...
8   False  True   True  False  False  False  False  False  False   True  ...
9    True  True  False  False  False  False  False  False   True   True  ...

   ASIAN / PACIFIC ISLANDER  BLACK HISPANIC  WHITE HISPANIC  WHITE  \
0                     False           False           False  False
1                      True           False           False  False
2                     False           False           False  False
3                     False           False           False  False
4                     False           False           False  False
5                     False           False           False  False
6                     False           False           False  False
7                     False           False           False  False
8                     False            True           False  False
9                     False           False            True  False

   AMERICAN INDIAN/ALASKAN NATIVE  18-24  25-44    <18  45-64    65+
0                           False   True  False  False  False  False
1                           False  False   True  False  False  False
2                           False  False   True  False  False  False
3                           False  False   True  False  False  False
4                           False  False   True  False  False  False
5                           False  False   True  False  False  False
6                           False   True  False  False  False  False
7                           False  False   True  False  False  False
8                           False  False   True  False  False  False
9                           False  False   True  False  False  False

[10 rows x 61 columns]
```

Analysis

Models and Training

Train and test data sets were generated (75% train / 25% test) and the three models built with their respective outputs provided. The KNN model was assessed for 20 differen k values beginning at 2. The Random Forest and Gradient Boost models were evaluated for 20 different learner values ranging from 25 to 500, at intervals of 50. Accuracy and computation times were collected for each model at each of the conditions.

Each models was evaluated at 20 different condition sets to facilitate a side by side comparison of classification accurracy and runtime.

The results of the evaluations are shown below.

In [11]:

```
y = classificationData['murder']
X = classificationData.drop(['murder'], axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_sta

dfAccuracy = pd.DataFrame()
dfTimes = pd.DataFrame()

#KNN Evaluation
knnStartTime = time.time()

knnScores = [None]
knnTimes =[None]
for k in range(2, 22):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    knnPredictions = knn.predict(X_test)
    knnScores.append(metrics.accuracy_score(y_test, knnPredictions))
    knnTimes.append(time.time() - knnStartTime)

dfAccuracy['KNN'] = knnScores
dfTimes['KNN'] = knnTimes

#Random Forest
forestStartTime = time.time()

estimators = [25, 50, 75, 100, 125, 150, 175, 200, 225, 250, 275, 300, 325, 350, 375,
forestScores = [None]
forestTimes = [None]
for e in estimators:
    forest = RandomForestClassifier(n_estimators=e)
    forest.fit(X_train, y_train)
    forestPredictions = forest.predict(X_test)
    forestScores.append(metrics.accuracy_score(y_test, forestPredictions))
    forestTimes.append(time.time() - forestStartTime)

dfAccuracy['Forest'] = forestScores
dfTimes['Forest'] = forestTimes

#Gradient Boost
boostStartTime = time.time()

boostScores = [None]
boostTimes = [None]
for e in estimators:
    boost = GradientBoostingClassifier(n_estimators=e)
    boost.fit(X_train, y_train)
    boostPredictions = boost.predict(X_test)
    boostScores.append(metrics.accuracy_score(y_test, boostPredictions))
    boostTimes.append(time.time() - boostStartTime)

dfAccuracy['Boost'] = boostScores
dfTimes['Boost'] = boostTimes
```

```python
print("Table1: Accuracy Scores")
print(dfAccuracy)

print('')

print("Table2: Computation Times (seconds)")
print(dfTimes)
```

```
Table1: Accuracy Scores
         KNN    Forest     Boost
0        NaN       NaN       NaN
1   0.793753  0.779313  0.816041
2   0.753257  0.781667  0.815884
3   0.795636  0.778999  0.815571
4   0.777429  0.780568  0.815571
5   0.806938  0.780725  0.815728
6   0.798619  0.782452  0.815414
7   0.809606  0.782923  0.815414
8   0.805211  0.783550  0.814786
9   0.812431  0.782295  0.814943
10  0.807879  0.781824  0.814943
11  0.814315  0.784335  0.814315
12  0.812117  0.784963  0.815571
13  0.814158  0.781824  0.815414
14  0.813216  0.784963  0.814158
15  0.814786  0.782923  0.814315
16  0.813844  0.783864  0.814943
17  0.815414  0.784649  0.814315
18  0.814786  0.783707  0.814472
19  0.815414  0.786219  0.815257
20  0.815257  0.783237  0.814629

Table2: Computation Times (seconds)
          KNN     Forest      Boost
0         NaN        NaN        NaN
1    4.150854   0.453202   0.438048
2    6.338213   1.232533   1.254602
3    9.044696   2.414008   2.316445
4   11.897177   3.955795   3.690711
5   14.679410   5.905917   5.400847
6   17.361629   8.900021   7.465702
7   19.771153  11.916579   9.870031
8   22.277524  15.020473  12.607264
9   24.649970  18.526683  16.161250
10  27.080641  22.573222  20.194409
11  29.699369  27.721518  24.242798
12  32.423681  32.381730  28.524155
13  34.795897  37.425014  33.738252
14  37.240925  43.865817  38.676113
15  39.629192  49.807590  43.874922
16  42.235077  57.031243  50.426766
17  45.234476  65.078231  56.431689
18  47.940352  73.142348  63.171523
19  50.473495  80.695335  70.170099
20  52.857408  89.099107  77.346224
```

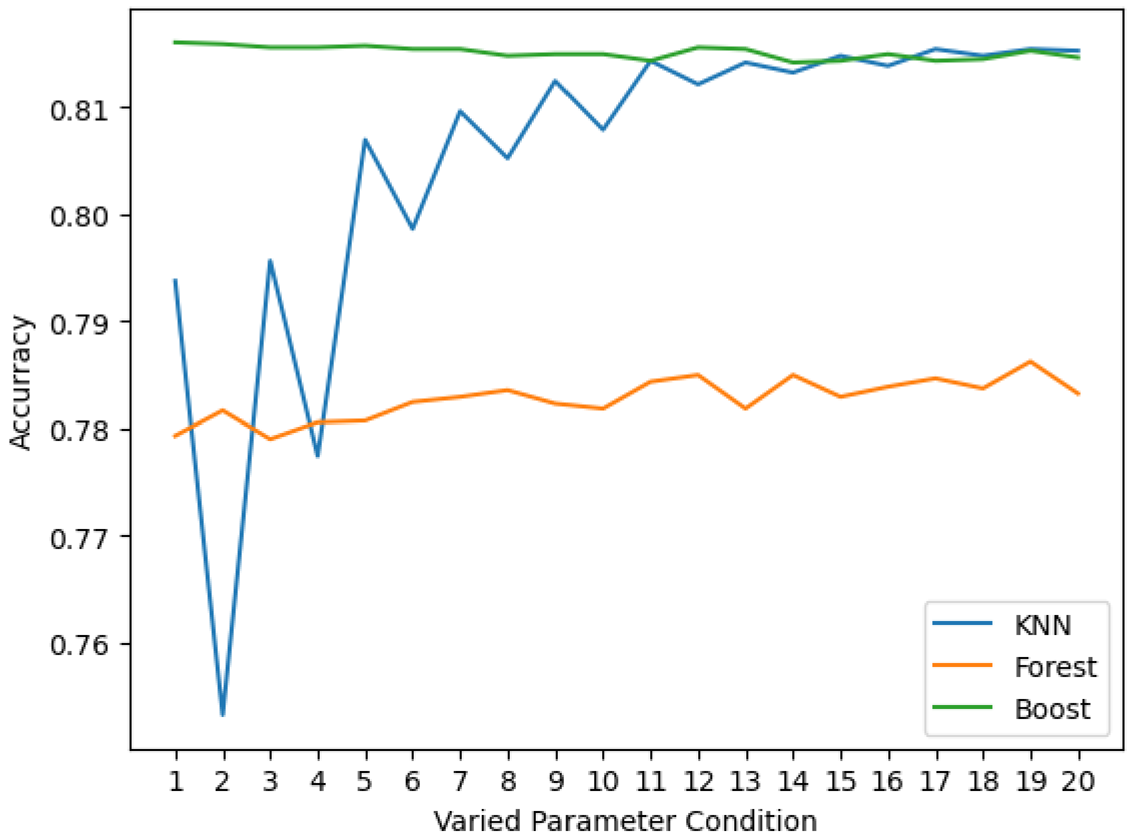
In [ ]:
```
Plots were generated to compare models.
```

In [12]:
```python
xLabels = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

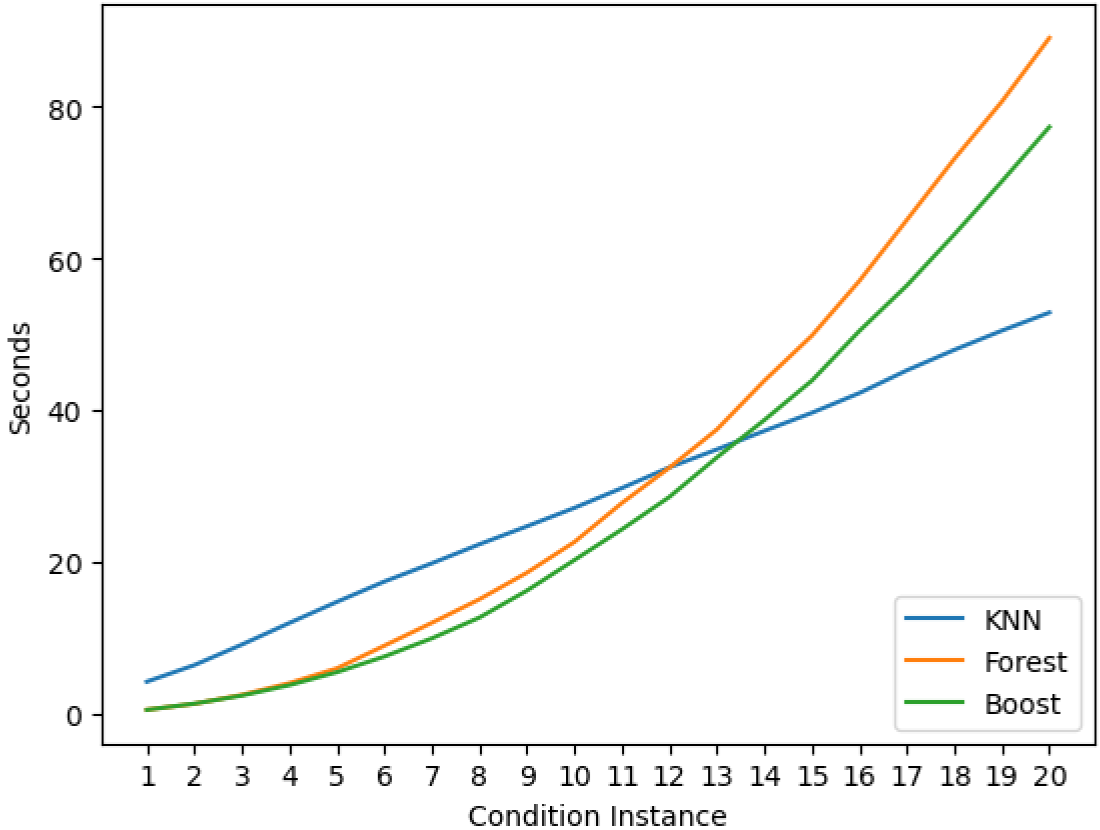
```python
print('Plot 1:  Accuracy Plot')
plt.plot(dfAccuracy['KNN'])
plt.plot(dfAccuracy['Forest'])
plt.plot(dfAccuracy['Boost'])
plt.xlabel('Varied Parameter Condition')
plt.ylabel('Accurracy')
plt.legend(['KNN', 'Forest', 'Boost'], loc=4)
plt.xticks(xLabels)
plt.show()


print('Plot 2:  Computation Time Plot')
plt.plot(dfTimes['KNN'])
plt.plot(dfTimes['Forest'])
plt.plot(dfTimes['Boost'])
plt.xlabel('Condition Instance')
plt.ylabel('Seconds')
plt.legend(['KNN', 'Forest', 'Boost'], loc=4)
plt.xticks(xLabels)
plt.show()
```

```
Plot 1:  Accuracy Plot
```




```
Plot 2:  Computation Time Plot
```



Feature Importance

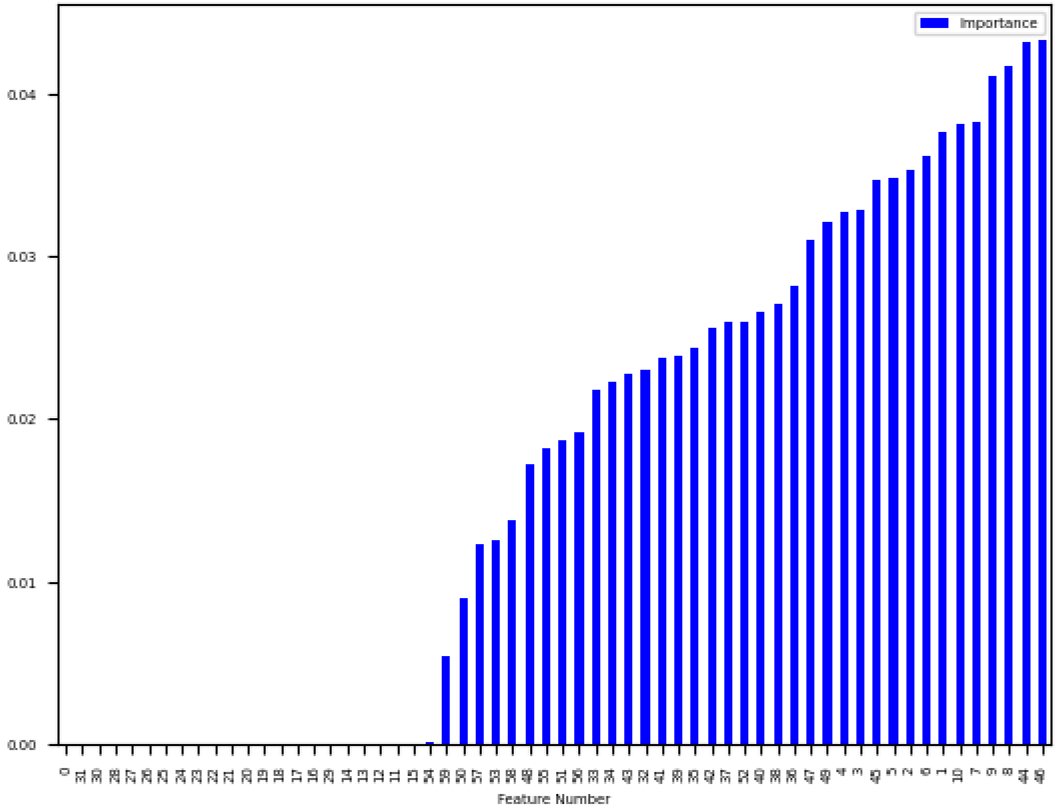
A features importance test was performed on the dataset to assess the relevence of the 61 features and to provide insight into model performance.

In [19]:

```
featureImportanceForest = RandomForestClassifier(n_estimators=300)
featureImportanceForest.fit(X_train, y_train)
featureImportance = featureImportanceForest.feature_importances_
importantanceDf = pd.DataFrame({"Feature": pd.DataFrame(X_train).columns, "Importance'
importantanceDf.set_index('Importance')
importantanceDf = importantanceDf.sort_values('Importance')
importantanceDf.plot.bar(color = 'blue')
plt.xlabel('Feature Number')
plt.rc('font', size=6)
plt.show
```

Out[19]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



The feature importance assessment showed that sex and most of the hours of the day have no impact on the model. These features will be removed and the models run with the reduced feature set. For illustration purposes the first 10 rows of the classification data set are provided.

In [17]:
```
featuresZeroImportance = importantanceDf[importantanceDf['Importance'] < .0001]
print(featuresZeroImportance)
featuresToRemove = list(featuresZeroImportance['Feature'])
print(featuresToRemove)

print('Create Important Feature Dataframe')
dfInportantFeatures = classificationData.drop(featuresToRemove, axis = 1)
dfInportantFeatures.info()
print(dfInportantFeatures.head(10))
```

```
   Feature  Importance
0      sex         0.0
31     h23         0.0
30     h22         0.0
28     h20         0.0
27     h19         0.0
26     h18         0.0
25     h17         0.0
24     h16         0.0
23     h15         0.0
22     h14         0.0
21     h13         0.0
20     h12         0.0
19     h11         0.0
18     h10         0.0
17      h9         0.0
16      h8         0.0
29     h21         0.0
14      h6         0.0
13      h5         0.0
12      h4         0.0
11      h3         0.0
15      h7         0.0
['sex', 'h23', 'h22', 'h20', 'h19', 'h18', 'h17', 'h16', 'h15', 'h14', 'h13', 'h12',
'h11', 'h10', 'h9', 'h8', 'h21', 'h6', 'h5', 'h4', 'h3', 'h7']
Create Important Feature Dataframe
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25482 entries, 0 to 25595
Data columns (total 39 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   murder                         25482 non-null  bool
 1   d0                             25482 non-null  bool
 2   d1                             25482 non-null  bool
 3   d2                             25482 non-null  bool
 4   d3                             25482 non-null  bool
 5   d4                             25482 non-null  bool
 6   d5                             25482 non-null  bool
 7   d6                             25482 non-null  bool
 8   h0                             25482 non-null  bool
 9   h1                             25482 non-null  bool
 10  h2                             25482 non-null  bool
 11  m1                             25482 non-null  bool
 12  m2                             25482 non-null  bool
 13  m3                             25482 non-null  bool
 14  m4                             25482 non-null  bool
 15  m5                             25482 non-null  bool
 16  m6                             25482 non-null  bool
 17  m7                             25482 non-null  bool
 18  m8                             25482 non-null  bool
 19  m9                             25482 non-null  bool
 20  m10                            25482 non-null  bool
 21  m11                            25482 non-null  bool
 22  m12                            25482 non-null  bool
 23  BROOKLYN                       25482 non-null  bool
 24  QUEENS                         25482 non-null  bool
 25  BRONX                          25482 non-null  bool
 26  MANHATTAN                      25482 non-null  bool
 27  STATEN ISLAND                  25482 non-null  bool
 28  BLACK                          25482 non-null  bool
```

```
 29  ASIAN / PACIFIC ISLANDER        25482 non-null  bool
 30  BLACK HISPANIC                  25482 non-null  bool
 31  WHITE HISPANIC                  25482 non-null  bool
 32  WHITE                           25482 non-null  bool
 33  AMERICAN INDIAN/ALASKAN NATIVE  25482 non-null  bool
 34  18-24                           25482 non-null  bool
 35  25-44                           25482 non-null  bool
 36  <18                             25482 non-null  bool
 37  45-64                           25482 non-null  bool
 38  65+                             25482 non-null  bool
dtypes: bool(39)
memory usage: 1.1 MB
   murder     d0     d1     d2     d3     d4     d5     d6     h0     h1  ...  \
0   False  False  False  False   True  False  False  False  False   True  ...
1   False  False  False  False  False   True  False  False  False  False  ...
2   False  False  False  False  False  False  False   True   True  False  ...
3   False  False  False  False  False  False   True  False  False   True  ...
4   False  False   True  False  False  False  False  False  False  False  ...
5    True  False  False  False  False  False   True  False   True  False  ...
6    True  False  False   True  False  False  False  False  False  False  ...
7   False  False  False  False  False   True  False  False  False   True  ...
8   False   True  False  False  False  False  False  False   True  False  ...
9    True  False  False  False  False  False  False   True   True  False  ...

   ASIAN / PACIFIC ISLANDER  BLACK HISPANIC  WHITE HISPANIC  WHITE  \
0                     False           False           False  False
1                      True           False           False  False
2                     False           False           False  False
3                     False           False           False  False
4                     False           False           False  False
5                     False           False           False  False
6                     False           False           False  False
7                     False           False           False  False
8                     False            True           False  False
9                     False           False            True  False

   AMERICAN INDIAN/ALASKAN NATIVE  18-24  25-44    <18  45-64    65+
0                           False   True  False  False  False  False
1                           False  False   True  False  False  False
2                           False  False   True  False  False  False
3                           False  False   True  False  False  False
4                           False  False   True  False  False  False
5                           False  False   True  False  False  False
6                           False   True  False  False  False  False
7                           False  False   True  False  False  False
8                           False  False   True  False  False  False
9                           False  False   True  False  False  False

[10 rows x 39 columns]
```

Reduced Feature Sets

Results of the models deployed with the reduced feature sets were compared.

In [18]:
```python
y = dfInportantFeatures['murder']
X = dfInportantFeatures.drop(['murder'], axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_sta

knnImportantFeatuesStartTime = time.time()
```

```
knnImportantFeatuesScores = [None]
knnImportantFeatuesTimes = [None]
for k in range(2, 22):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    knnImportantFeatuesPredictions = knn.predict(X_test)
    knnImportantFeatuesScores.append(metrics.accuracy_score(y_test, knnImportantFeatue
    knnImportantFeatuesTimes.append(time.time() - knnImportantFeatuesStartTime)

dfAccuracy['IF_KNN'] = knnImportantFeatuesScores
dfTimes['IF_KNN'] = knnImportantFeatuesTimes

forestImportantFeatuesStartTime = time.time()

forestImportantFeaturesScores = [None]
forestImportantFeaturesTimes = [None]
for e in estimators:
    forest = RandomForestClassifier(n_estimators=e)
    forest.fit(X_train, y_train)
    forestImportantFeatuesPredictions = forest.predict(X_test)
    forestImportantFeaturesScores.append(metrics.accuracy_score(y_test, forestImportar
    forestImportantFeaturesTimes.append(time.time() - forestImportantFeatuesStartTime)

dfAccuracy['IF_Forest'] = forestImportantFeaturesScores
dfTimes['IF_Forest'] = forestImportantFeaturesTimes


boostImportantFeatuesStartTime = time.time()

boostImportantFeaturesScores = [None]
boostImportantFeaturesTimes = [None]
for e in estimators:
    boost = GradientBoostingClassifier(n_estimators=e)
    boost.fit(X_train, y_train)
    boostImportantFeaturesPredictions = boost.predict(X_test)
    boostImportantFeaturesScores.append(metrics.accuracy_score(y_test, boostImportantF
    boostImportantFeaturesTimes.append(time.time() - boostImportantFeatuesStartTime)

dfAccuracy['IF_Boost'] = boostImportantFeaturesScores
dfTimes['IF_Boost'] = boostImportantFeaturesTimes

print("Table 3: Accuracy Scores")
print(dfAccuracy)
print("")
print("Table 4: Computation Times (seconds)")
print(dfTimes)
```

Table 3: Accuracy Scores

	KNN	Forest	Boost	IF_KNN	IF_Forest	IF_Boost
0	NaN	NaN	NaN	NaN	NaN	NaN
1	0.793753	0.774133	0.816041	0.793753	0.777429	0.816198
2	0.753257	0.779156	0.815884	0.753257	0.780882	0.816041
3	0.795636	0.780411	0.815571	0.795636	0.778214	0.815571
4	0.777429	0.780411	0.815571	0.777429	0.779469	0.815571
5	0.806938	0.779469	0.815728	0.806938	0.778528	0.815884
6	0.798619	0.782295	0.815414	0.798619	0.781039	0.815414
7	0.809606	0.783080	0.815257	0.809606	0.781510	0.815414
8	0.805211	0.781196	0.814786	0.805211	0.782452	0.815100
9	0.812431	0.782138	0.814943	0.812431	0.783394	0.814786
10	0.807879	0.781981	0.814472	0.807879	0.785748	0.814943
11	0.814315	0.781510	0.815257	0.814315	0.782452	0.814786
12	0.812117	0.781667	0.814472	0.812117	0.780882	0.815414
13	0.814158	0.784649	0.814472	0.814158	0.783864	0.815414
14	0.813216	0.783080	0.814158	0.813216	0.782923	0.814472
15	0.814786	0.783707	0.814943	0.814786	0.783237	0.814786
16	0.813844	0.782452	0.815257	0.813844	0.784649	0.814315
17	0.815414	0.785120	0.814943	0.815414	0.784963	0.815100
18	0.814786	0.784178	0.814943	0.814786	0.784806	0.814943
19	0.815414	0.783394	0.815257	0.815414	0.786062	0.814943
20	0.815257	0.785120	0.814629	0.815257	0.783550	0.814629

Table 4: Computation Times (seconds)

	KNN	Forest	Boost	IF_KNN	IF_Forest	IF_Boost
0	NaN	NaN	NaN	NaN	NaN	NaN
1	7.143708	0.494018	0.520737	2.190519	0.626532	0.516487
2	9.678371	1.529175	1.381484	4.592006	1.476287	1.293379
3	12.724357	3.270172	2.563129	7.257660	3.469318	2.381472
4	15.582071	5.605014	4.182911	10.135032	5.855485	3.905216
5	19.260036	7.912961	6.162511	13.744112	8.200517	6.322040
6	21.945208	10.599072	8.640114	16.808591	10.894904	8.654711
7	24.478692	13.663767	11.422848	19.499914	14.007270	11.419767
8	27.133781	17.325108	15.155066	22.140889	18.047600	15.206410
9	29.826973	22.468794	18.891805	24.749639	22.577338	18.911994
10	32.867790	26.957742	22.830912	27.777923	26.976693	22.770627
11	35.670117	31.757662	27.107825	30.945665	32.102499	26.881984
12	38.283276	38.166832	32.518307	34.230172	38.505718	32.193462
13	40.954995	44.071745	37.630870	36.888635	44.167713	37.261700
14	43.759811	50.467853	43.223540	39.675961	50.681957	42.646202
15	46.516842	57.868544	49.866562	42.316915	58.378937	49.264624
16	49.623553	65.021986	56.252465	45.573926	65.453766	55.416774
17	52.361191	74.006140	63.896537	48.750888	74.013451	62.156178
18	55.000636	82.433630	71.015882	51.402350	82.458524	69.616832
19	57.562840	91.629969	79.505557	54.225662	91.913945	76.916451
20	60.160380	101.581423	87.517169	57.163086	101.413854	85.863353

Reduced Feature Plots

In [20]:

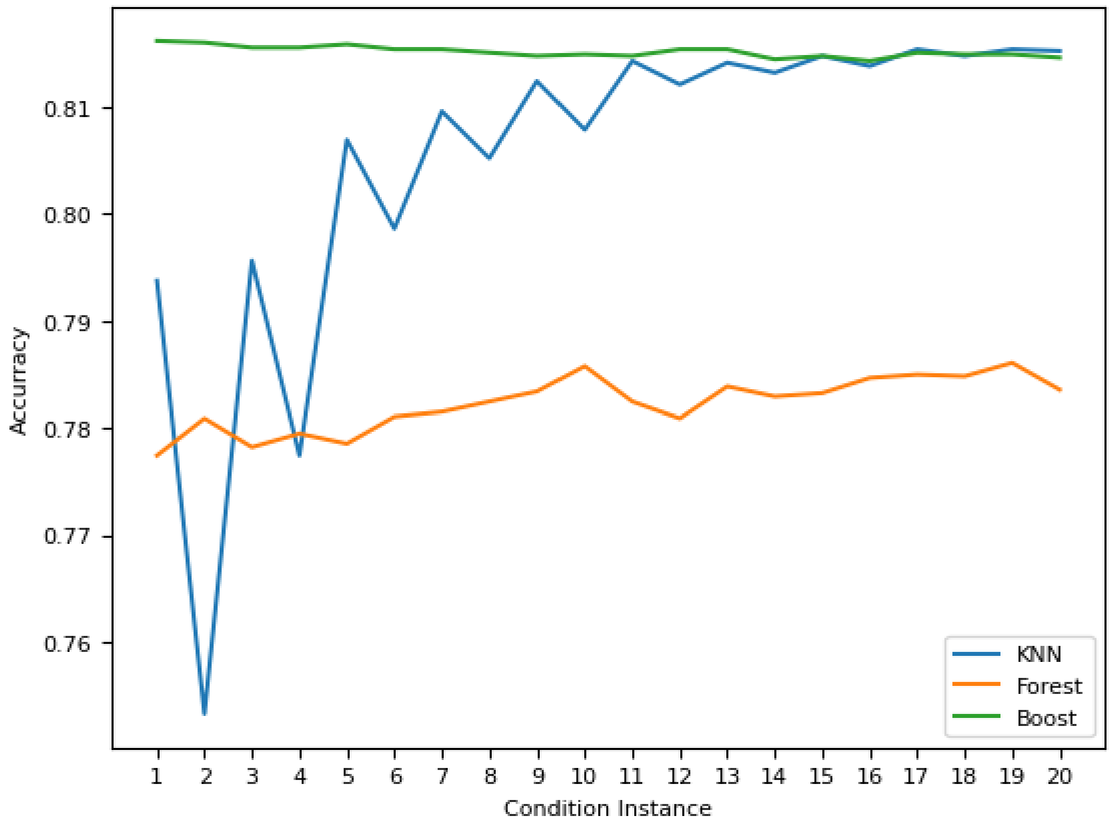
```
plt.rc('font', size=8)

print('Plot 3:  Reduced Feature Set Accuracy Plot')
plt.plot(dfAccuracy['IF_KNN'])
plt.plot(dfAccuracy['IF_Forest'])
plt.plot(dfAccuracy['IF_Boost'])
plt.xlabel('Condition Instance')
plt.ylabel('Accuracy')
plt.legend(['KNN', 'Forest', 'Boost'], loc=4)
```

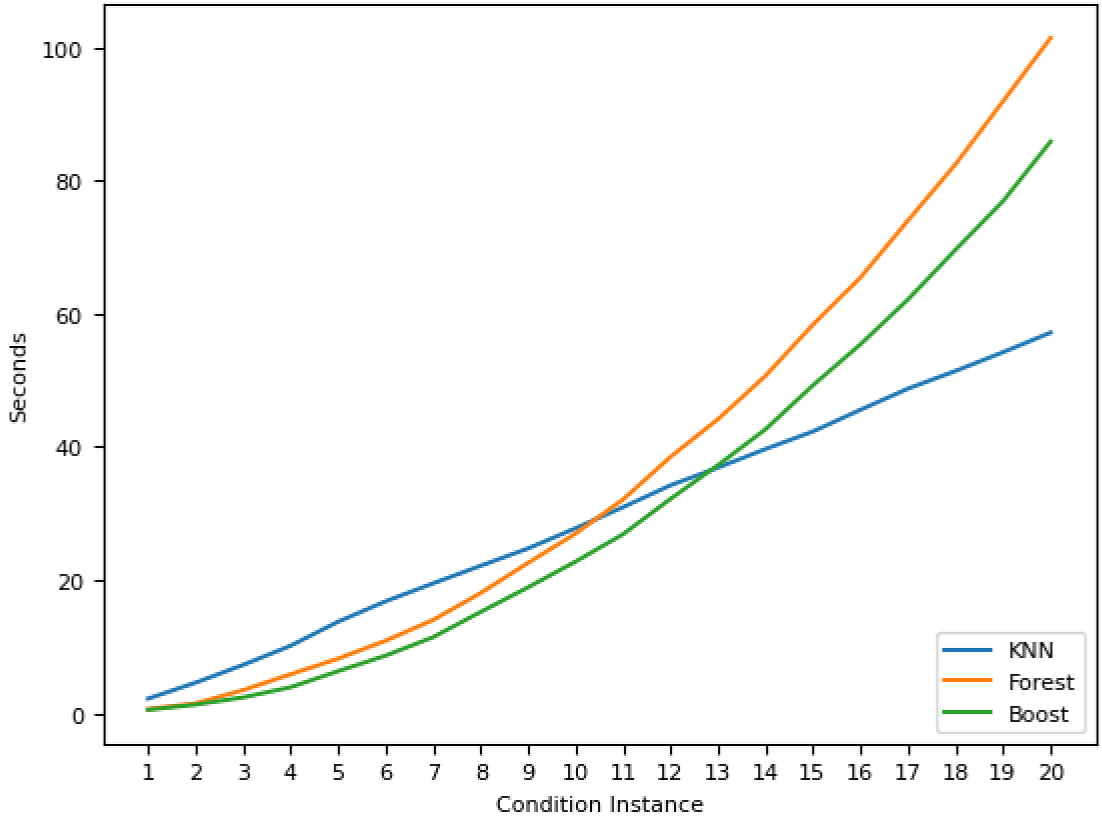
```
plt.xticks(xLabels)
plt.show()

print('Plot 4:  Reduced Feature Set Computation Time Plot')
plt.plot(dfTimes['IF_KNN'])
plt.plot(dfTimes['IF_Forest'])
plt.plot(dfTimes['IF_Boost'])
plt.xlabel('Condition Instance')
plt.ylabel('Seconds')
plt.legend(['KNN', 'Forest', 'Boost'], loc=4)
plt.xticks(xLabels)
plt.show()
```

```
Plot 3:  Reduced Feature Set Accuracy Plot
```




```
Plot 4:  Reduced Feature Set Computation Time Plot
```



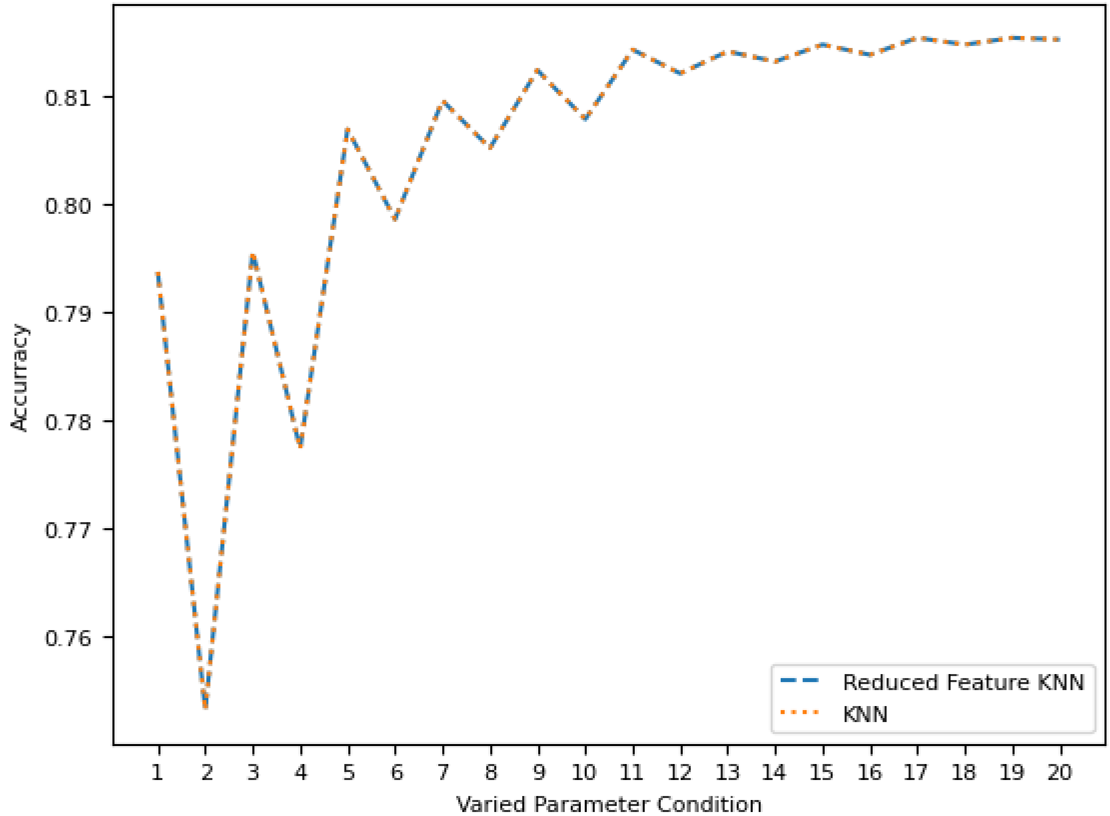
Plots were generated for each model comparing perfromance between the data sets

KNN Plots

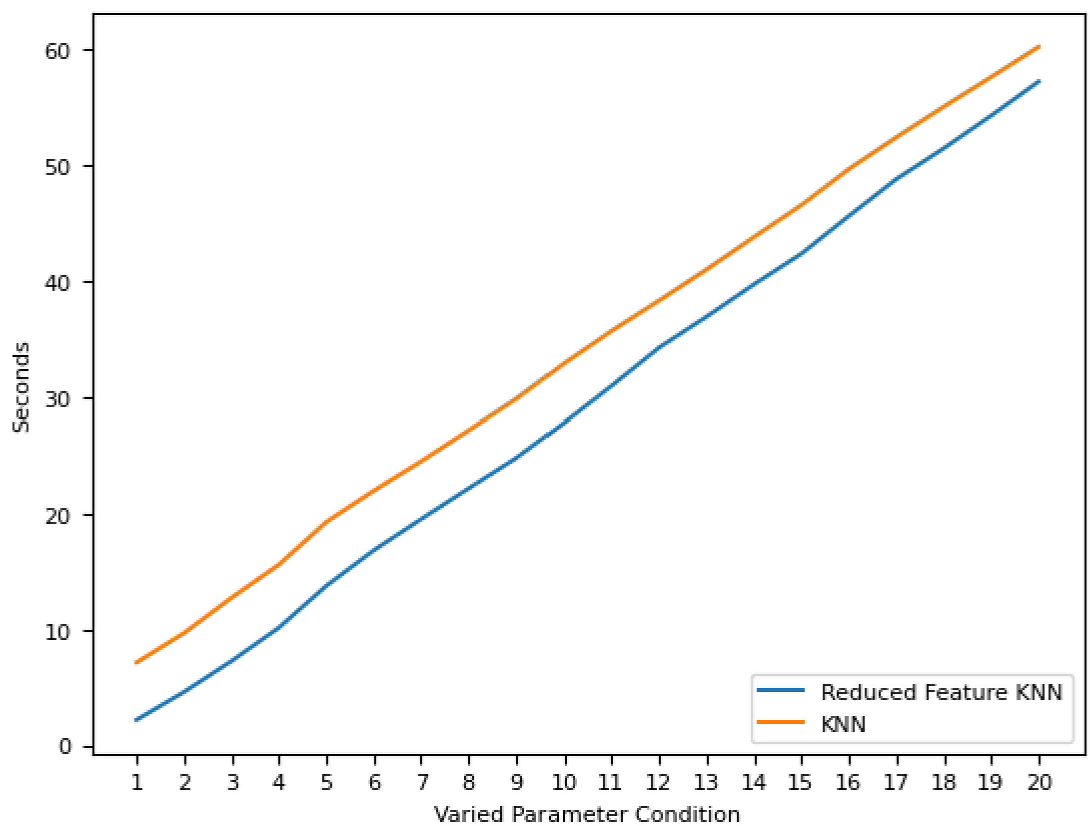
In [22]:
```
print('Plot 5:  KNN Comparison Accuracy Plot')
plt.plot(dfAccuracy['IF_KNN'], linestyle='dashed')
plt.plot(dfAccuracy['KNN'], linestyle='dotted')
plt.xlabel('Varied Parameter Condition')
plt.ylabel('Accurracy')
plt.legend(['Reduced Feature KNN','KNN'], loc=4)
plt.xticks(xLabels)
plt.show()

print('Plot 6:  Reduced Feature Set Computation Time Plot')
plt.plot(dfTimes['IF_KNN'])
plt.plot(dfTimes['KNN'])
plt.xlabel('Varied Parameter Condition')
plt.ylabel('Seconds')
plt.legend(['Reduced Feature KNN', 'KNN'], loc=4)
plt.xticks(xLabels)
plt.show()
```

```
Plot 5:  KNN Comparison Accuracy Plot
```




```
Plot 6:  Reduced Feature Set Computation Time Plot
```



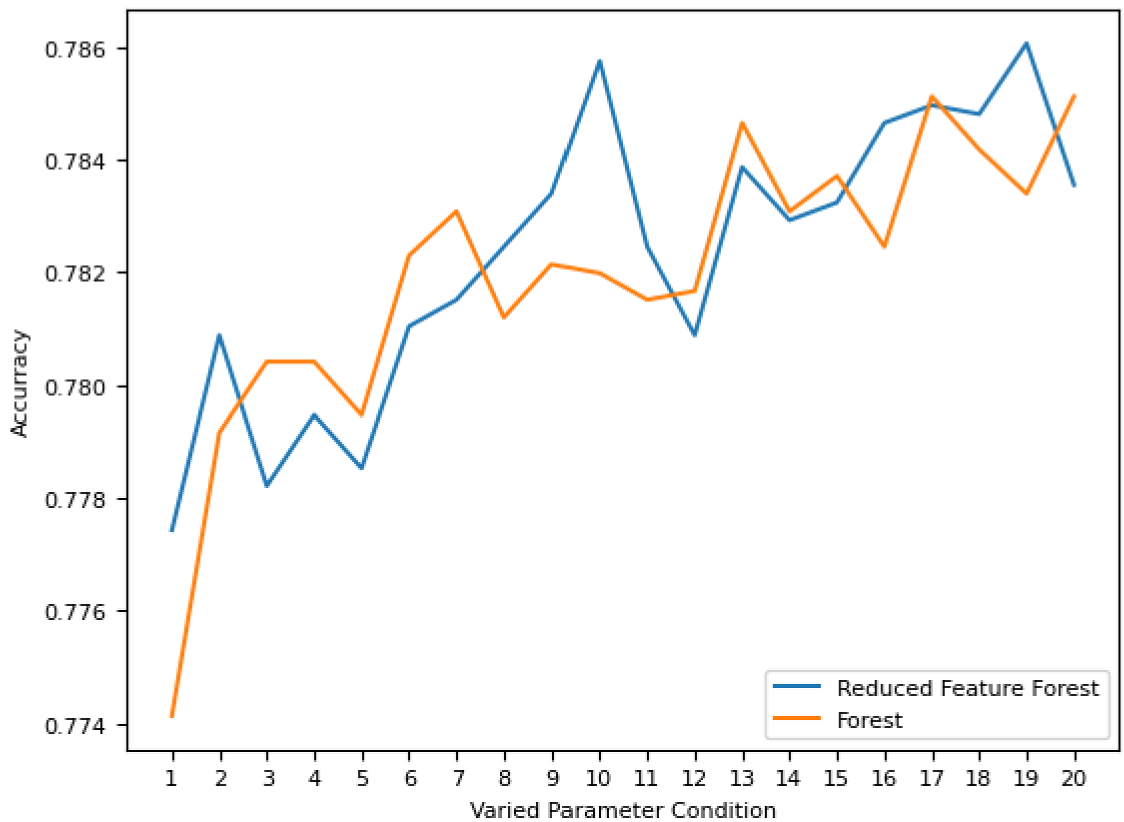
Random Forest Plots

In [23]:
```
print('Plot 7:  Reduced Feature Set Accuracy Plot')
plt.plot(dfAccuracy['IF_Forest'])
```

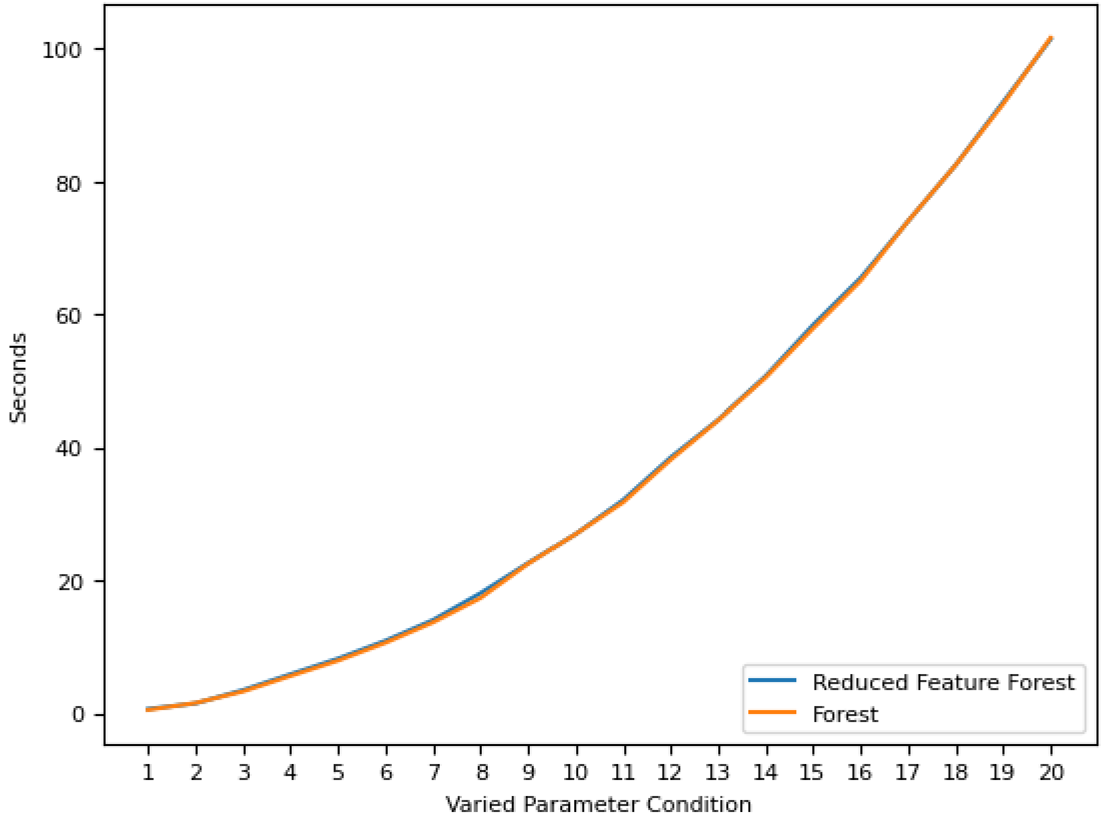
```python
plt.plot(dfAccuracy['Forest'])
plt.xlabel('Varied Parameter Condition')
plt.ylabel('Accurracy')
plt.legend(['Reduced Feature Forest', 'Forest'], loc=4)
plt.xticks(xLabels)
plt.show()

print('Plot 8:  Reduced Feature Set Computation Time Plot')
plt.plot(dfTimes['IF_Forest'])
plt.plot(dfTimes['Forest'])
plt.xlabel('Varied Parameter Condition')
plt.ylabel('Seconds')
plt.legend(['Reduced Feature Forest', 'Forest'], loc=4)
plt.xticks(xLabels)
plt.show()
```

```
Plot 7:  Reduced Feature Set Accuracy Plot
```




```
Plot 8:  Reduced Feature Set Computation Time Plot
```



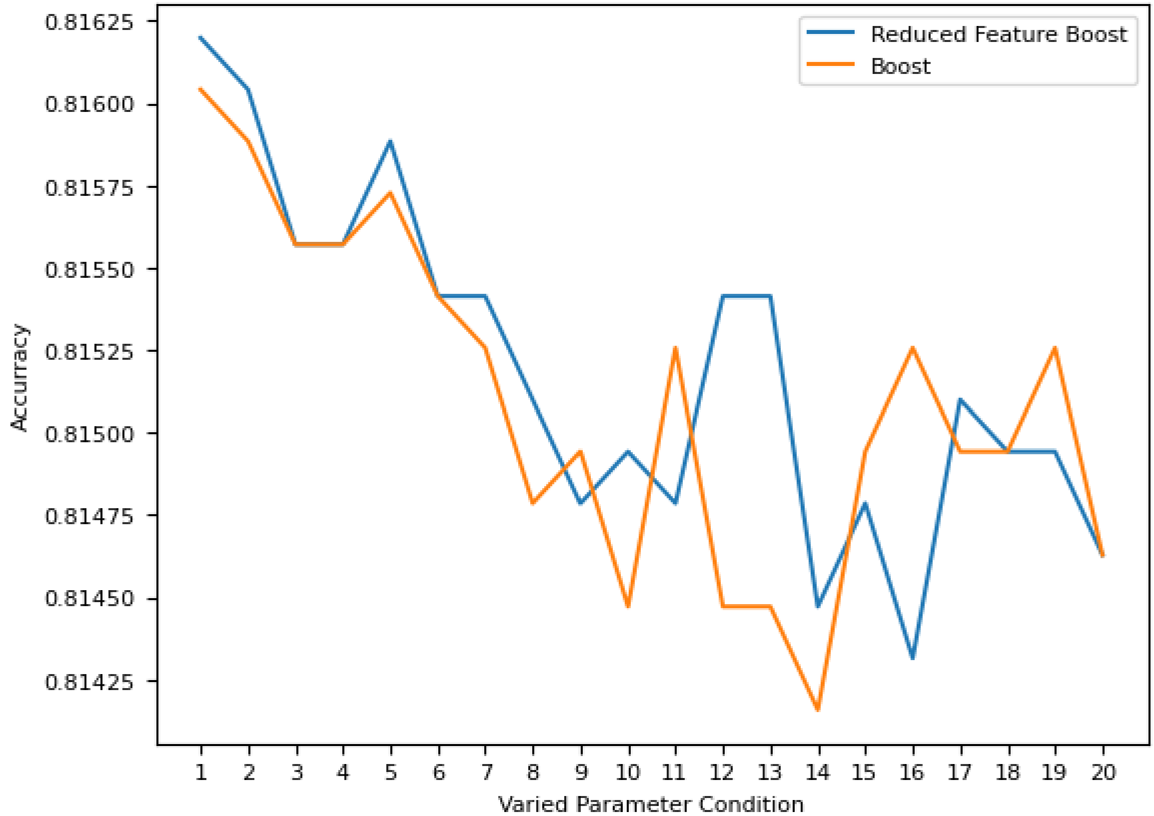
Gradient Decent Plots

In [24]:

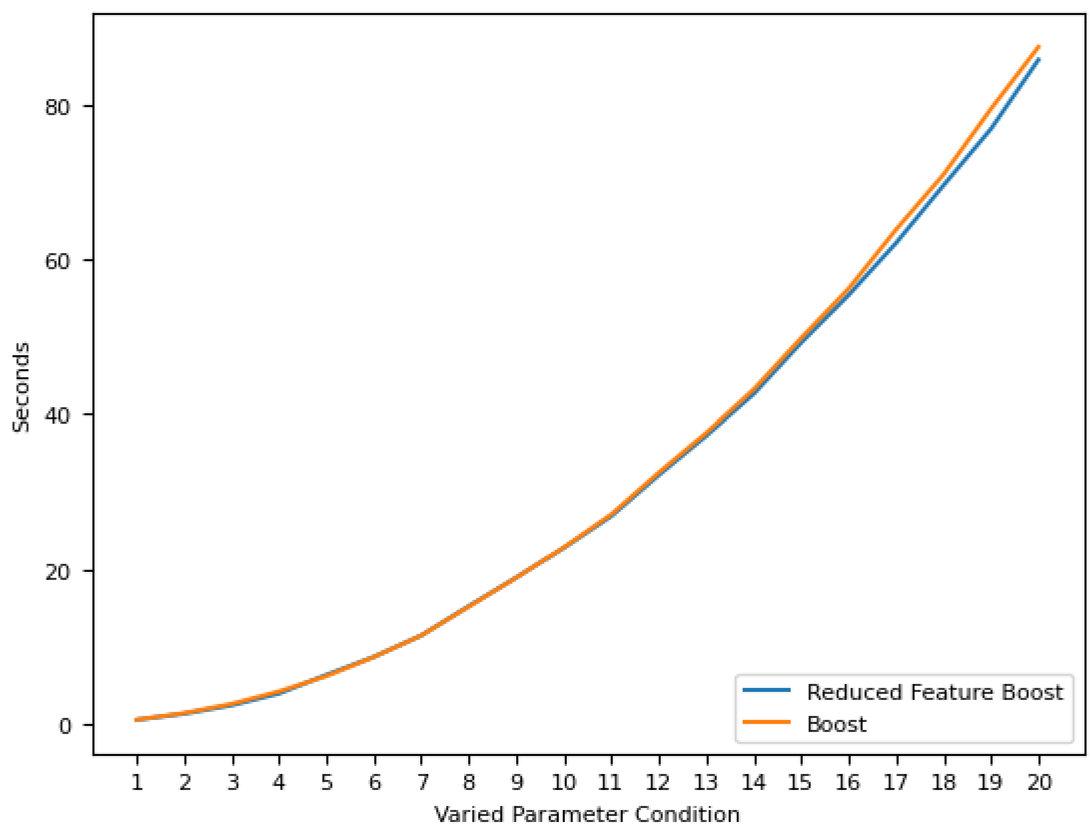
```
print('Plot 9:  Reduced Feature Set Accuracy Plot')
plt.plot(dfAccuracy['IF_Boost'])
plt.plot(dfAccuracy['Boost'])
plt.xlabel('Varied Parameter Condition')
plt.ylabel('Accurracy')
plt.legend(['Reduced Feature Boost', 'Boost'], loc=1)
plt.xticks(xLabels)
plt.show()

print('Plot 10:  Reduced Feature Set Computation Time Plot')
plt.plot(dfTimes['IF_Boost'])
plt.plot(dfTimes['Boost'])
plt.xlabel('Varied Parameter Condition')
plt.ylabel('Seconds')
plt.legend(['Reduced Feature Boost', 'Boost'], loc=4)
plt.xticks(xLabels)
plt.show()
```

```
Plot 9:  Reduced Feature Set Accuracy Plot
```



Plot 10: Reduced Feature Set Computation Time Plot



Discusion

Accuracy

Table 1 and Plot 1 provide a comparison of model accuracy based on the original data set of 61 features. The plot shows that the KNN and Boost models performed similarly. Both models resulted in an ultimate classification accuracy of 81%. The difference between the two models being that the Gradient Boost arrived at the 81% accuracy much more quickly than the KNN model. The Random Forest model performed the worst out of the three with its accuracy topping out at ~78%. Similar to the Gradient Boost model, the Random Forest the accuracy levels were arrived at more quickly than the KNN model.

The feature importance method from the Random Forest classifier showed that approximately 22 of the 61 features had no impact on the models. The features removed were the victim sex and the vast majority of the hours of the day. This makes sense since most of the murders occurred in the midnight and early morning hours and male victims far outweighed female victims. Expectations were that the models would perform better with the feature reduction. The models were rebuilt and run on the reduced feature set.

Table 3 and Plots 3, 5, 7 and 9 detail the accuracy comparisons between models, as well as individual model comparisons between feature set sizes. Expectations were that the accuracy would improve given the reduced feature set, however this was not the case. For all three models the accuracy between the feature set sizes were virtually identical. The KNN model resulted in an exact overlay and the tree models had minimal variations. These variations can be attributed to the randomness in the training and test sets used in the individual tree for each model.

Although not initially expected, the tracking of the accuracies between feature set sizes makes sense. For the KNN model, the unimportant features would have never fallen in the nearest neighbor group from the offset. They would reside outside the group as they had no influence on the model.

As for the tree models, the unimportant features would have been pruned out very early in the process, again with no influence on the model, removing them had no impact on the accuracy. Furthermore, the fact that the unimportant features were removed from the tree models in early in the classification, that explains why the variance in accuracy in those models was less than the KNN model. The KNN model had to progress through all ks looking at all points, whereas the tree models eliminate unimportant features from the offset.

Computational Time

Table 2 and Plot 2 provide insight into the computational resources, in terms of computation time, required for each model based on the original 61 features. The KNN model performed worst in the lower condition (lower k) state than both the Random Forest and Gradient Boost Models. However, at the k~12 and learner count of ~300 all three models required about the same computational resources. This was the crossover point where the KNN model began to outperform the tree models requiring less resources as the k value and number of learners increased.

Table 4 and plots 4, 6, 8 and 10 provide a view of the computational resources required across and within models based on the feature set sizes. The results of these comparisons were more in line with initial expectations. Plot 4 shows that the comparison between models at the reduced feature set size had very similar shapes. However, the computational resources, as expected, were reduced by 17%, 21% and 30% for the KNN, Random Forest and Gradient Boost models respectively.

Plots 6, 8 and 10 provide a side by side comparison for each of the models. These plots clearly show the reduction on computational resources required for the reduced feature set size. Additionally, the magnitude of the changes falls in line with expectations. The KNN model still had to calculate up to 20 nearest neighbors, just with a reduced feature set, still fairly expensive as k grows. On the other hand, the tree models benefitted more from the reduced feature set size because the pruning did not have to initially remove the unimportant features and was thus able to get to the optimal condition more quickly.

Enhancements

One question that arises is how the conversion from multi-variable features to binary features impacted the outcome of this study. The ideal model contains a minimal number of features required to arrive at optimal performance. In this case, the move to binary features was purely for educational and experimental purposes. It would be very interesting to compare the current results to results using the original multi-value feature set. Forward or backward stepwise regression could be employed to identify the optimal feature set and then compare classification metrics between the two different approaches.

Conclusion

In the end, this project provided both expected and unexpected results. Classification accuracy comparisons were within expectations between models. All three models arrived at accuracies right around 80%. What was interesting was that the tree models converged to their maximum accuracy much more quickly than the KNN model. This is not surprising given the pruning characteristics in the tree models versus the iterative nature of the KNN model.

The reduction in computational resources based on the reduced feature set also fell in line with expectations. Reduction in the number of features resulted in on average a 23% decrease in computational resources.

Unexpected outcomes based on initial assumptions were the impacts (or lack of impact) based on the removal of the unimportant features. Initial expectations were that removing unimportant features would potentially promote increased accuracy. This was not the case but easily explainable once the structure of the models was considered.

When comparing all three models based on accuracy and resource requirements combined, the Gradient Boost model provided the best performance. The Gradient Boost model reached the highest accuracy while requiring the least amount of resources to get there. The KNN and Random Forest models were comparable. The KNN model provided a bit more accuracy but required more resources than the Random Forest. On the other hand, the Random Forest arrived at its optimal accuracy more quickly than the KNN model, but suffered from a slightly degraded accuracy level.