Project2

December 17, 2020

1 Project 2 San Francisco Crime Dataset

Author: Ryan Rosiak [rrosiak1@gulls.salisbury.edu] and Grant Dawson [gdawson1@gulls.salisbury.edu]

Date: 11/23/20

Description: Continuing work on San Francisco Crime Dataset

```
In [1]: import pandas as pd # Pandas library
        import numpy as np # Numpy library
        import matplotlib.pyplot as plt # Matplotlib library
        import numpy.linalg as la # Linear algebra functions
        import math # Math library
        import random # Random library
        import seaborn as sns # Seaborn library
```

2 The San Francisco Crime Dataset

```
In [2]: crime_data = pd.read_csv('./SanFranciscoCrimeDataset/crime.csv',
                                header=1,
                                skipinitialspace=False,
                                names=['IncidntNum', 'Category', 'Descript', 'DayOfWeek', 'Da
                                       'PdDistrict', 'Resolution', 'Address', 'X', 'Y', 'Locat
        crime_data = crime_data[ ~crime_data['PdDistrict'].isna() ]
        crime_data.head(10)
```

Out[2]:

	IncidntNum	Category	Descript
0	120058272	WEAPON LAWS	FIREARM, LOADED, IN VEHICLE, POSSESSION OR USE
1	141059263	WARRANTS	WARRANT ARREST
2	160013662	NON-CRIMINAL	LOST PROPERTY
3	160002740	NON-CRIMINAL	LOST PROPERTY
4	160002869	ASSAULT	BATTERY
5	160003130	OTHER OFFENSES	PAROLE VIOLATION
6	160003259	NON-CRIMINAL	FIRE REPORT
7	160003970	WARRANTS	WARRANT ARREST
8	160003641	MISSING PERSON	FOUND PERSON
9	160086863	LARCENY/THEFT	ATTEMPTED THEFT FROM LOCKED VEHICLE

```
  DayOfWeek                   Date   Time  PdDistrict      Resolution  \
0    Friday  01/29/2016 12:00:00 AM  11:00    SOUTHERN  ARREST, BOOKED
1    Monday  04/25/2016 12:00:00 AM  14:59     BAYVIEW  ARREST, BOOKED
2   Tuesday  01/05/2016 12:00:00 AM  23:50  TENDERLOIN            NONE
3    Friday  01/01/2016 12:00:00 AM  00:30     MISSION            NONE
4    Friday  01/01/2016 12:00:00 AM  21:35    NORTHERN            NONE
5  Saturday  01/02/2016 12:00:00 AM  00:04    SOUTHERN  ARREST, BOOKED
6  Saturday  01/02/2016 12:00:00 AM  01:02  TENDERLOIN            NONE
7  Saturday  01/02/2016 12:00:00 AM  12:21    SOUTHERN  ARREST, BOOKED
8    Friday  01/01/2016 12:00:00 AM  10:06     BAYVIEW            NONE
9    Friday  01/29/2016 12:00:00 AM  22:30     TARAVAL            NONE

                  Address           X          Y  \
0   800 Block of BRYANT ST -122.403405  37.775421
1    KEITH ST / SHAFTER AV -122.388856  37.729981
2   JONES ST / OFARRELL ST -122.412971  37.785788
3     16TH ST / MISSION ST -122.419672  37.765050
4    1700 Block of BUSH ST -122.426077  37.788019
5      MARY ST / HOWARD ST -122.405721  37.780879
6     200 Block of EDDY ST -122.411778  37.783981
7        4TH ST / BERRY ST -122.393357  37.775788
8  100 Block of CAMERON WY -122.387182  37.720967
9    1200 Block of 19TH AV -122.477377  37.764478

                                Location            PdId
0   (37.775420706711, -122.403404791479)  12005827212168
1  (37.7299809672996, -122.388856204292)  14105926363010
2  (37.7857883766888, -122.412970537591)  16001366271000
3  (37.7650501214668, -122.419671780296)  16000274071000
4   (37.788018555829, -122.426077177375)  16000286904134
5  (37.7808789360214, -122.405721454567)  16000313026150
6  (37.7839805592634, -122.411778295992)  16000325968000
7  (37.7757876218293, -122.393357241451)  16000397063010
8  (37.7209669615499, -122.387181635995)  16000364175000
9  (37.7644781578695, -122.477376524003)  16008686306240
```

3 What type of population is being sampled? What are the "things" getting measured – usually one per row of data.

The population of the data set is 150500 different crimes/incidents from San Francisco. Each data point gave extreme detail about where and when and some detail about what the specific incident was. We can look at this data and see many different things because we have many columns/details about one data point/incident.

4 What features does each sample have, i.e. what is being measured?

There are

IncidntNum Category Descript DayOfWeek Date Time PdDistrict Resolution Address X Y Location PdId

There are eight features to each sample. #### Feature: 1. Incident Number - THe incedent number associated to incident - This number is seperate form the police's ID number but ID's the number 2. Category - What type of incedent - This is a single to a few word that best describe he incedent that occured in the row 3. Descript - Description of the incident - This gives more detail than the category about the incident 4. Day Of Week/Date/Time - date and time when incident occured - These three columns give exact times, dates, and day of the weeks the row's incident occured 5. PdDistrict - Police District where incendent occured - This is the district the incident occured in but in the eyes of the police and their routing. 6. Resolution - Ending results - This is what happened to the party commiting the crime/causing the inccident 7. Address X/Y/Location - Location Longitude, Latitude - This is the exact geological location of the inccident 8. PdId - Police Department Identification Number - This is the ID number but for the police books

5 Are the features quantitative or qualitative? Ordinal or nominal? Continuous or discrete?

Feature: Nominal - catagorical - normally not a number

Ordinal - norminal data that can be ordered - can compare to each other but not base to compare numbers to/no abosolute zero

Interval - Not a full unit of messurement - not drawn to scale - ex tempature "it is not souble as hot outside, you say it is greater than" - 0 isn't you have 0 tempeture it is cold out

Radio - Unit of messurement

1. Incident Number -
    - Qualitiative
    - Ordinal
    - Discrete
2. Category -
    - Qualitative
    - Nominal
    - Ordinal
3. Descript -
    - Qualitative
    - Nominal
    - Ordinal
4. Day Of Week/Date/Time -
    - Day of week
        - Qualitative
        - Nominal

    - Ordinal
  - Data/Time
    - Quantitative
    - Interval
    - Discrete

5. PdDistrict -
   - Qualitative
   - Nominal
   - Ordinal
6. Resolution -
   - Qualitative
   - Nominal
   - Ordinal
7. Address X/Y/Location -
   - Quantitative
   - Interval
   - Continuous
8. PdId -
   - Qualitative
   - Nominal
   - Discrete

6 Is the data "complete" or do some of the samples have null or absent values for certain features? Why are these samples still useful? Why are they incomplete?

The data is filled out. It may seem like the Resolution column may have missing data when it says "NONE," But this is an actual case. This is possible that the police were not able to charge anyone for the crome/incident committed. If we were to consider "NONE" as null, then these points are still valid. These "null" points have meaning because these locations have potential criminals that have experience in evading the law.

7 Why are these features chosen to be part of the dataset?

Feature:

1. Incident Number - A unique id that is assigned to each incident (shortened version of the police id)
2. Category - Similar amoung most of the population - Broad desciption of incidents - Allows for search of a inccident
3. Descript - Less frequent amoung the population - Specific exsamples of incidents - Does the same as Categories but much moire specific exsamples

4. Day Of Week/Date/Time - Tells us when incidents occured
5. PdDistrict - Says stuff about location from police side of things, typically same group of police patrol the same aprts of town.
6. Resolution - Tells us what happened to the offenders in the end, if anything.
7. Address X/Y/Location - Geographic location of incident/Tells us where is happened.
8. PdId - Allows for future look up the inccident in police records.

8 What are some other features that are not included but that you think might make sense to include for this dataset?

Some other features that could have been added could have been the races of the perpetrators and victims. The addition of these new features adds some exciting relationships and statistics. We can show what race is causing the most considerable amount of crime. It can also show crime rates between races. It would also be interesting to see if the police knew it was gang violence or not. This may not be known for every incident, but it makes some fascinating data about gangs' locations for the ones that it is known. Disputes could be happening at HQ locations or on the edge of territories where we can then section the city species on a map into a gang "turfs."

9 Give at least one way that you can pivot the dataset to get a slightly different representation of some values. Explain what this is and how you would use it for a visualization.

We can pivot the data so that the time columns would be the x and the number of crimes would be the y. This can be either to show when the most incidents happen and to also show if you split the data up wha ttimes have mire of specific types of inccidents. In other words when is it the "safest" out side. You could also then sploit this into multiple graphs one for each police district to show where you livce when is it the safest. Or when you should be on the most lookout for a specific crime/all crime.

10 Identify any possible relationships between features included in the data: which ones are likely to affect others?

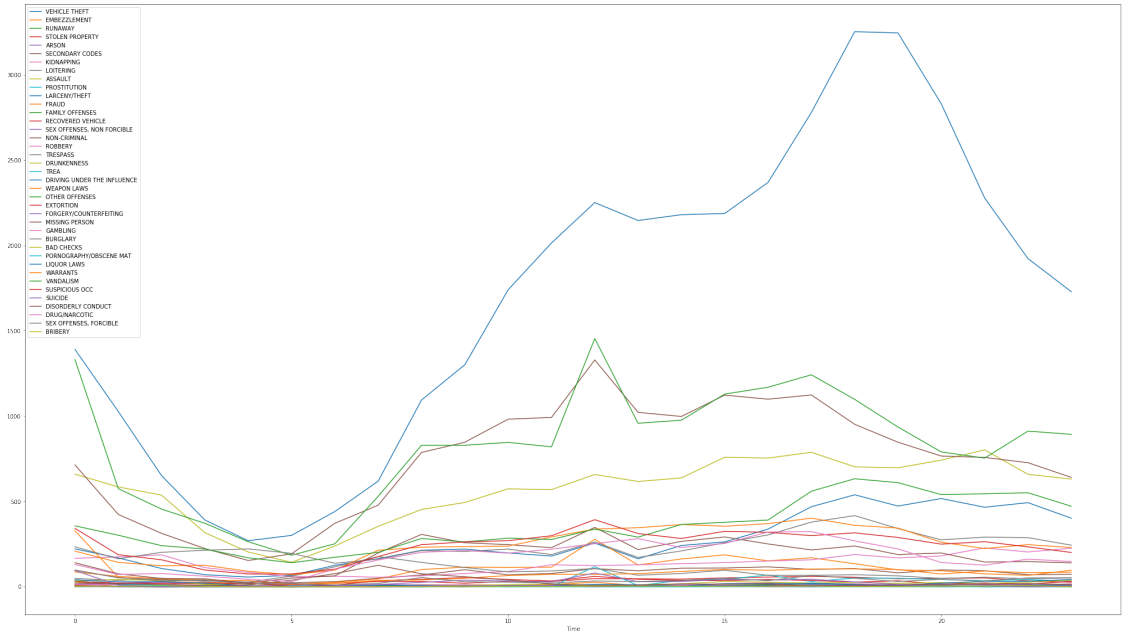
Times and type of Incident, typically robberies and car jacking ahppens at night and not while tohe sun is up. WHile we can look at the type of incidents and how frequent it is at hours in the day, we can also look into the fact that police may be tired and not want to deal with problems. In the actual graph we can see that between 4am - 6am there is a divit in all tpye of crime. We can either think people don't commit crimes at these hours, police are switching out the day and night shift, police are tired, and a few other things.

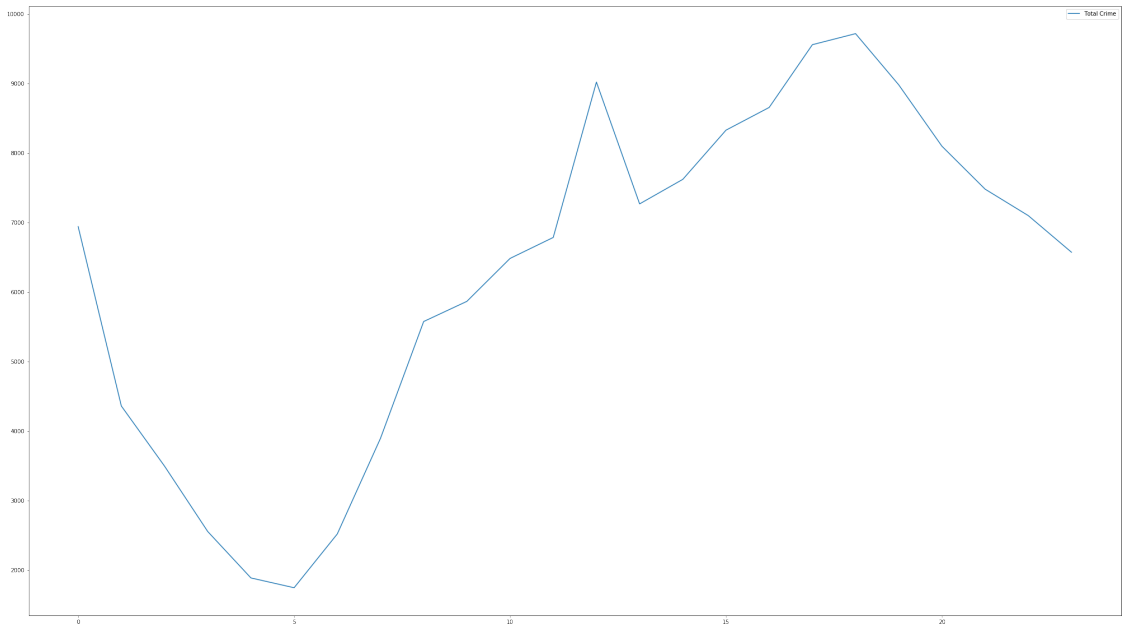
11 Show at least one plot or visualization to illustrate this (possible) relationship.

```
In [3]: TimeType = crime_data[['Time', 'Category']]
```

```
d = set(TimeType['Category'])

totalCrime = {}
categoryCrime = {}

for i in d:
    categoryCrime[i] = np.zeros(len(range(0,24)), dtype=int)

totalCrime['Total Crime'] = np.zeros(len(range(0,24)), dtype=int)

categoryCrimeData = pd.DataFrame(categoryCrime)
totalCrimeData = pd.DataFrame(totalCrime)
categoryCrimeData.index.name = 'Time'


for i in TimeType.values:
    hours,mins = i[0].split(':')
    hours = int(hours)
    categoryCrimeData.loc[hours][i[1]] += 1
    totalCrimeData.loc[hours]['Total Crime'] += 1


# print(crimeData)


categoryCrimeData.plot(figsize=(35,20))
totalCrimeData.plot(figsize=(35,20))
```

Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7b964f8b00>





12 What numerical or statistical techniques might you consider using to determine whether the relationship is reliable?

This is the coversation of if it is a said time then the chance of getting mugged is higher then at another time. We can prove that the diffrent imes yeild diffrent chance of a crime or a specific crime.

We are going to show the relationship between the probality that the time is T when you rcar was stollen.

T = The event there is a crime happening at time T

C = A car is jacked

P(T | C) = P(T n C) / P(C)

P(C) = Probability that your car is being stolen assuming a crime is going to happen to you = number of car thefts / total number of crimes = numberCJ / totalCrime

P(T) = Probability that a crime happened at time T = number of crimes at time T / total crimes = crimesTimes/totalCrimes

P(T n C) = Probability that at time T the crime is happening is car jacking = number of total car jackings at time T / total number of carjackings = totalCJTime/totalCJ

P(T | C) = (totalCJTime/totalCJ) / (numCJ/totalCrime)

13 Are there external inferences you think might be possible? For instance, can you hypothesize a relationship with data not included in the dataset? Why or why not?

For most data there is always some sort of external inference that can be made. THis is purely from the fact that a typical dataset never has all the data you can think of. People think diffrently which leads to data being collected you may hve not thought of and data not colleced that would have be interesting/nice to have. But for our dataset we could see more with a race of the person committing the crimes. We can see if there is relations between the race and the cimes a particullar race that was involved. Another external infrences could be if applicable the cost of damages, we could see if there was a higher probability of being caught if you do more than say $10,000 worth of damage.

14 What "extra" features can you perhaps compute from the data? For example, if you have data that includes product dates of purchase, you can "engineer" the data to construct the most popular products over various lengths of time (e.g. a particular holiday season). How might you use this information? Using the holiday example, you might try to correlate holiday sales of a product to some mainstream event that popularized it.

With the data we have we can make guesses about connections between crimes. Say if there is a yearly robbery in the same neighborhood orsomething to the extent of paterns of crimes we can take guesses that some crimes were related to one another. We can also make inferences about police officers biases/oruption. THis may be a bit of a stretch but if we notice that there is a sppecific type of crime that never has anyone getting caught/the result is none then we can chalk it up to the police being crupt or that there are experienced people in that area/PD district.

15 K-Nearest Neighbors Algorithm

```
In [4]: # KNN Class
        class KNN:
            def __init__(self, k):
                self.k = k
                self.trainarr = []
                self.classifier_list = []
                self.predicted_points = []

            def distance(self, point):
                """
                Finds the distance of every training set value from point, and then returns th
                by distance values paired with the corresponding "classifier"
                """
        #         distance_pairs_list = [(0,0) for _ in range(len(self.trainarr))]
```

```
#         for ind,i in enumerate(self.trainarr):
#             distance_pairs_list[ind] = (la.norm(np.array(point) - np.array(i[0])), i
        distance_pairs_list = [(la.norm(np.array(point) - np.array(i[0])), i[1]) for i
        distance_pairs_list.sort(key= lambda x: x[0])
        return distance_pairs_list

    def train(self, tlist):
        """
        Takes a list of training data => tuple containing a samples X ndims list and i
        that will be stored for later
        """
        self.trainarr = tlist[:]
        self.classifier_list = list(set([i[1] for i in self.trainarr]))

    def predict(self, point, graphing=False):
        """
        Point must be a list, same dimension as the training set
        graphing is default False (for efficiency), otherwise True adds point to list
        """
        classifier_counts = {i[1]: 0 for i in self.trainarr}
        distance_list = self.distance(point)
        key_select = ''
        for index,item in enumerate(distance_list):
            if index < self.k:
                classifier_counts[item[1]] += 1
            else:
                #max_dict_val = max([value for key,value in classifier_counts.items()]
                max_dict_val = max(classifier_counts.values())
                #max_dict_val = max(classifier_counts, key=classifier_counts.get)
                if list(classifier_counts.values()).count(max_dict_val) > 1:
                    classifier_counts[item[1]] += 1
                else:
                    for key, value in classifier_counts.items():
                        if value == max_dict_val:
                            key_select = key
                            break
                    break
        #print(f'I am predicting that the classifier for point {point} is... {key_sele
        if graphing:
            self.predicted_points.append((point, key_select))
        return key_select

    def plot_train(self):
        """
        Can only plot for 2d data
        """
        colors = ['Green', 'Red', 'Blue', 'Black', 'Yellow', 'Pink', 'Brown', 'Purple']
        for ind, item in enumerate(self.classifier_list):
```

```
                plotx = [j[0][0] for j in self.trainarr if j[1] == item]
                ploty = [j[0][1] for j in self.trainarr if j[1] == item]
                plt.scatter(x=plotx, y=ploty, c=colors[ind], label=item)
            plt.title('Plot of sample training data')
            plt.legend()
            plt.xlabel('X axis')
            plt.ylabel('Y axis')
            plt.show()

        def plot_train_and_predictions(self):
            """
            Can only plot for 2d data, plots the point that was predicted
            """
            colors = ['Green', 'Red', 'Blue', 'Black', 'Yellow', 'Pink', 'Brown', 'Purple']
            for ind, item in enumerate(self.classifier_list):
                plotxtrain = [j[0][0] for j in self.trainarr if j[1] == item]
                plotytrain = [j[0][1] for j in self.trainarr if j[1] == item]
                plotxpredict = [j[0][0] for j in self.predicted_points if j[1] == item]
                plotypredict = [j[0][1] for j in self.predicted_points if j[1] == item]
                plt.scatter(x=plotxtrain, y=plotytrain, c=colors[ind], label=f'Trained: {it
                plt.scatter(x=plotxpredict, y=plotypredict, c=colors[ind], label=f'Predicte
            plt.title(f'Plot of sample training data and predicted points (k = {self.k})')
            plt.legend()
            plt.xlabel('X axis')
            plt.ylabel('Y axis')
            plt.show()
```

15.1 To split the data:

```
In [5]: # split_sets Function
        def split_sets(dataframe, train_p):
            """
            Returns 2 lists of the form needed to use the KNN class using the San Francisco Cr
            -- The function uses x and y coordinate data for graphing and the type of crime as
            **The dataframe passed can be subset to go by location to see how KNN works on eac
            """
            assert(train_p < 1) # Make sure that p is a percentage
            assert(train_p > 0) # Make sure that p is non-negative

            # Split dataframe into training and test/prediction sets
            train_count = int(dataframe.shape[0] * train_p)
            predict_count = int(dataframe.shape[0] * (1 - train_p))

            #tmp_list = dataframe.values

            tmp_list = dataframe.values[:]
            np.random.shuffle(tmp_list) # Shuffle the rows
            train_list = tmp_list[0:train_count] # Take the first train_p percentage for the t
```

```
    predict_list = tmp_list[train_count:] # The rest go to predict data

    rtn_train = [([i[1], i[2]], i[0]) for i in train_list]
    rtn_predict = [([i[1], i[2]], i[0]) for i in predict_list]



    #train_list = [ dataframe.values[random.randint(0,train_count-1)][0:4] for i in ra
    #predict_list = [ dataframe.values[random.randint(0,train_count-1)][0:4] for i in
#     train_list = [(0,0) for _ in range(train_count)]
#     for i in range(0, train_count):
#         rand_row = train_list[random.randint(0, len(train_list) - 1)]
#         train_list[i] = ([rand_row[1], rand_row[2]], rand_row[0])


#     predict_list = [(0,0) for _ in range(predict_count)]
#     for i in range(0, predict_count):
#         rand_row = predict_list[random.randint(0, len(predict_list) - 1)]
#         predict_list[i] = ([rand_row[0], rand_row[1]], rand_row[2])

    # Some test runs
    #print(rtn_train)
    #print(rtn_predict)
    #print(len(dataframe))
    #print(len(train_list), len(predict_list))
    return rtn_train, rtn_predict
```

In [6]:
```
# # Test cases
# # No location subset tests:
# # Working on subset
# no_sub_data = crime_data[['Category', 'X', 'Y']]
# no_sub_data = no_sub_data[:20000]
# train_list, predict_list = split_sets(no_sub_data, .4)

# test1 = KNN(9)
# test1.train(train_list)
# correct = sum([1 for tupl ito yearsn predict_list if test1.predict(tupl[0]) == tupl[
# incorrect = len(predict_list) - correct
# correct_dict = {"Correct": correct, "Incorrect": incorrect}
# print(correct_dict)
```

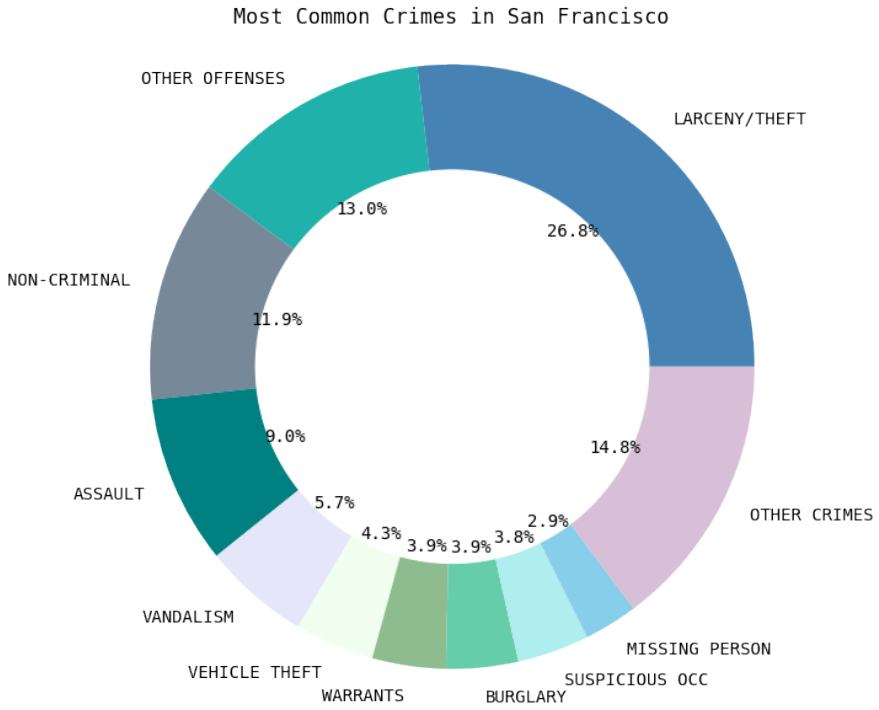
15.2 All districts to be looked at

In [7]:
```
# First subsetted Test case
# Print the various regions
parse_districts = list(set([i[0] for i in crime_data[['PdDistrict']].values]))
print('DISTRICTS')
print(parse_districts)
```

```
        # Print the various crimes
        parse_crimes = list(set([i[0] for i in crime_data[['Category']].values]))
        print('CRIME CATEGORIES')
        print(parse_crimes)
```

```
DISTRICTS
['CENTRAL', 'INGLESIDE', 'PARK', 'RICHMOND', 'TENDERLOIN', 'TARAVAL', 'BAYVIEW', 'NORTHERN', 'I
CRIME CATEGORIES
['VEHICLE THEFT', 'EMBEZZLEMENT', 'RUNAWAY', 'STOLEN PROPERTY', 'ARSON', 'SECONDARY CODES', 'KI
```

In [8]:
```
        # Pie chart for the percentage of each crime relative to the entire dataset
        count_crime_amount = {charge: 0 for charge in parse_crimes}
        crime_categories = crime_data[['Category']]
        for i in crime_categories.iterrows():
            count_crime_amount[i[1][0]] += 1
        #print(count_crime_amount)
        sort_crime_amount = sorted(count_crime_amount.items(), key=lambda x: x[1], reverse=True
        #print(sort_crime_amount)
        label_crime = [i[0] for i in sort_crime_amount][:10]
        crime_pie = [i[1] / crime_data.shape[0] for i in sort_crime_amount][:10]
        crime_pie.append(1 - sum(crime_pie))
        #print(crime_pie)

        fig = plt.figure(figsize=(16,10))
        plt.rcParams.update({'font.size': 14, 'font.family': 'monospace'})
        label_crime.append('OTHER CRIMES')
        labels = label_crime
        sizes = crime_pie
        colors = ['steelblue', 'lightseagreen', 'lightslategrey', 'teal', 'lavender', 'honeydew

        plt.pie(sizes, colors=colors, labels=labels, autopct='%1.1f%%', shadow=False)

        centre_circle = plt.Circle((0,0),0.65,color='white',fc='white',linewidth=1.25)
        fig = plt.gcf()
        fig.gca().add_artist(centre_circle)
        plt.axis('equal')
        plt.title('Most Common Crimes in San Francisco')
        plt.show()
```



15.2.1 Northern:

```
In [9]: northern_district = crime_data[crime_data['PdDistrict'] == 'NORTHERN'][['Category', 'X
        northern_district.head()
```

Out[9]:

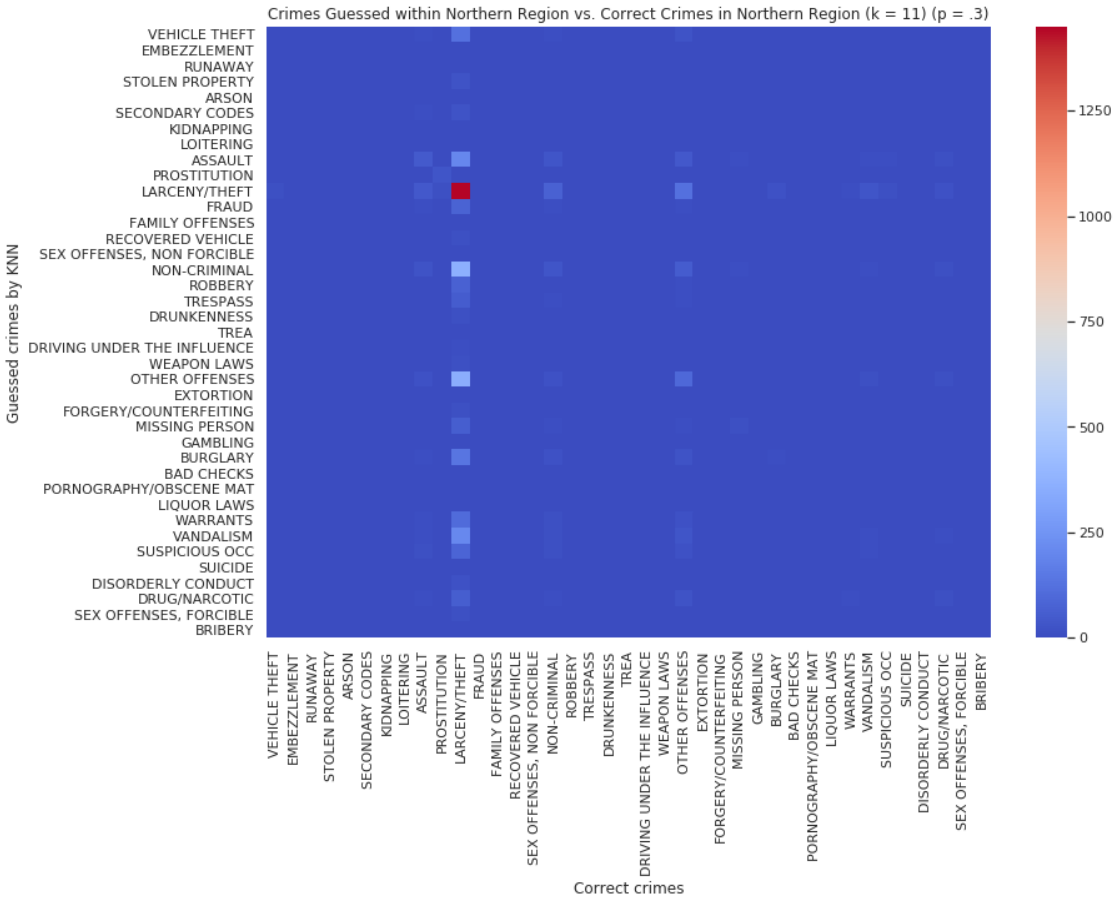
	Category	X	Y
4	ASSAULT	-122.426077	37.788019
19	FRAUD	-122.422316	37.773619
20	WARRANTS	-122.424520	37.792841
26	BURGLARY	-122.419203	37.787438
51	VEHICLE THEFT	-122.418045	37.777512

```
In [10]: train_northern1, predict_northern1 = split_sets(northern_district, .3)
         northern_test1 = KNN(11)
         northern_test1.train(train_northern1)
         subset_northern = predict_northern1[:5000]
         # Creates a list of tuples, (what the algorithm guessed, what is correct)
         sub_cf_northern = [(northern_test1.predict(tupl[0]), tupl[1]) for tupl in subset_north
```

```
In [11]: # confusion_matrix = [([0,0], district) for district in parse_districts]
         # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
         accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cri

         # INITIALIZE CONFUSION MATRIX FOR NORTHERN HERE!!!!!!!!!!!!!!!!!!!!!!!!
         northern_learning_curve = []
```

```
    for prediction in sub_cf_northern:
        accum_category_nums[prediction[1]][prediction[0]] += 1
    confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
    #print(confusion_matrix)
    sub_percentage = 0
    for ind, row in enumerate(confusion_matrix):
        sub_percentage += row[ind]
    sub_percentage /= 5000
    northern_learning_curve.append([11, sub_percentage])
    #print(northern_learning_curve)
```

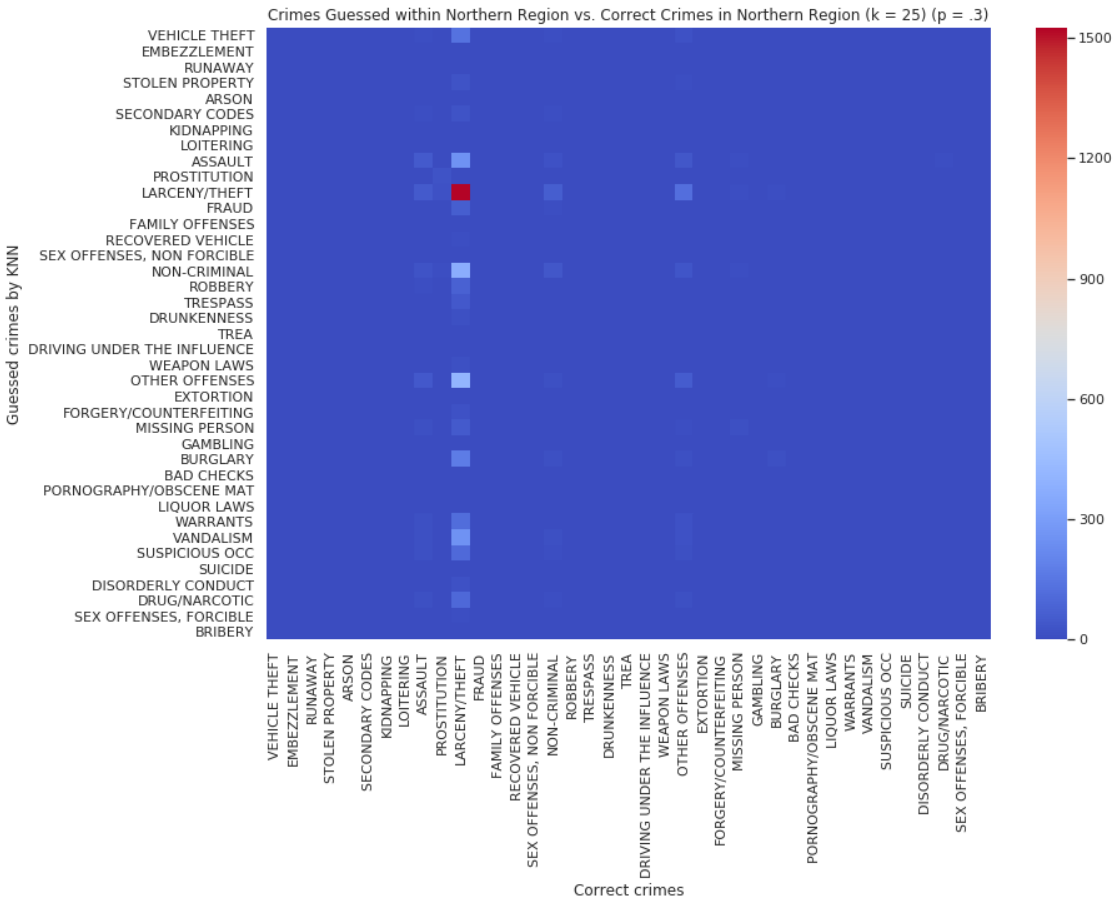
In [12]:
```
    # Plot the heatmap using seaborn
    sns.set(rc={'figure.figsize':(13,9)})
    ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
    ax.set(title='Crimes Guessed within Northern Region vs. Correct Crimes in Northern Reg
           xlabel='Correct crimes',
           ylabel='Guessed crimes by KNN')
    plt.show()
    print()
```



In [13]:
```
train_northern2, predict_northern2 = split_sets(northern_district, .3)
northern_test2 = KNN(25)
northern_test2.train(train_northern2)
subset_northern = predict_northern2[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_northern = [(northern_test2.predict(tupl[0]), tupl[1]) for tupl in subset_nortl
```

In [14]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cri

for prediction in sub_cf_northern:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
northern_learning_curve.append([25, sub_percentage])
#print(northern_learning_curve)
```

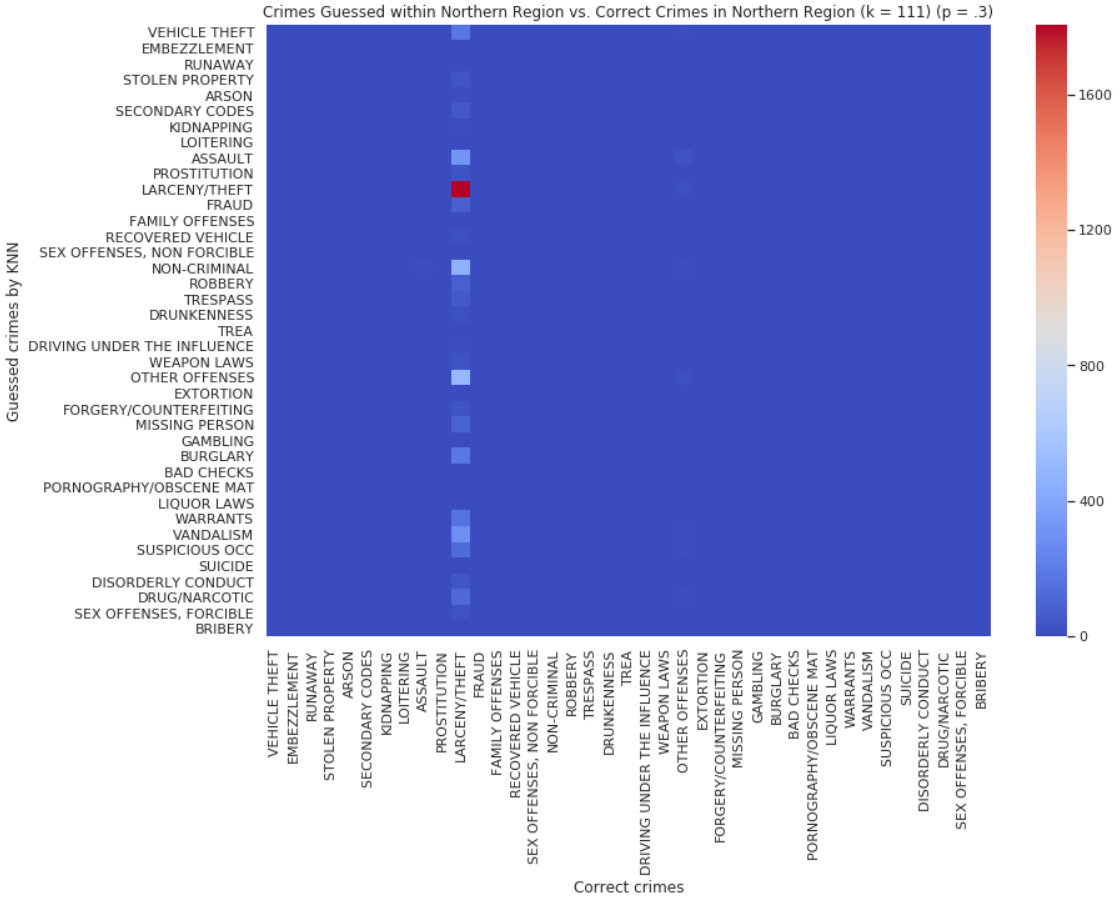
In [15]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklal
ax.set(title='Crimes Guessed within Northern Region vs. Correct Crimes in Northern Reg
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```




```
In [16]: train_northern3, predict_northern3 = split_sets(northern_district, .3)
         northern_test3 = KNN(111)
         northern_test3.train(train_northern3)
         subset_northern = predict_northern3[:5000]
         # Creates a list of tuples, (what the algorithm guessed, what is correct)
         sub_cf_northern = [(northern_test3.predict(tupl[0]), tupl[1]) for tupl in subset_northl

In [17]: # confusion_matrix = [([0,0], district) for district in parse_districts]
         # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
         accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cri

         for prediction in sub_cf_northern:
             accum_category_nums[prediction[1]][prediction[0]] += 1
         confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
         #print(confusion_matrix)
         sub_percentage = 0
```

```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        northern_learning_curve.append([111, sub_percentage])
        #print(northern_learning_curve)
```

In [18]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
        ax.set(title='Crimes Guessed within Northern Region vs. Correct Crimes in Northern Reg
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```

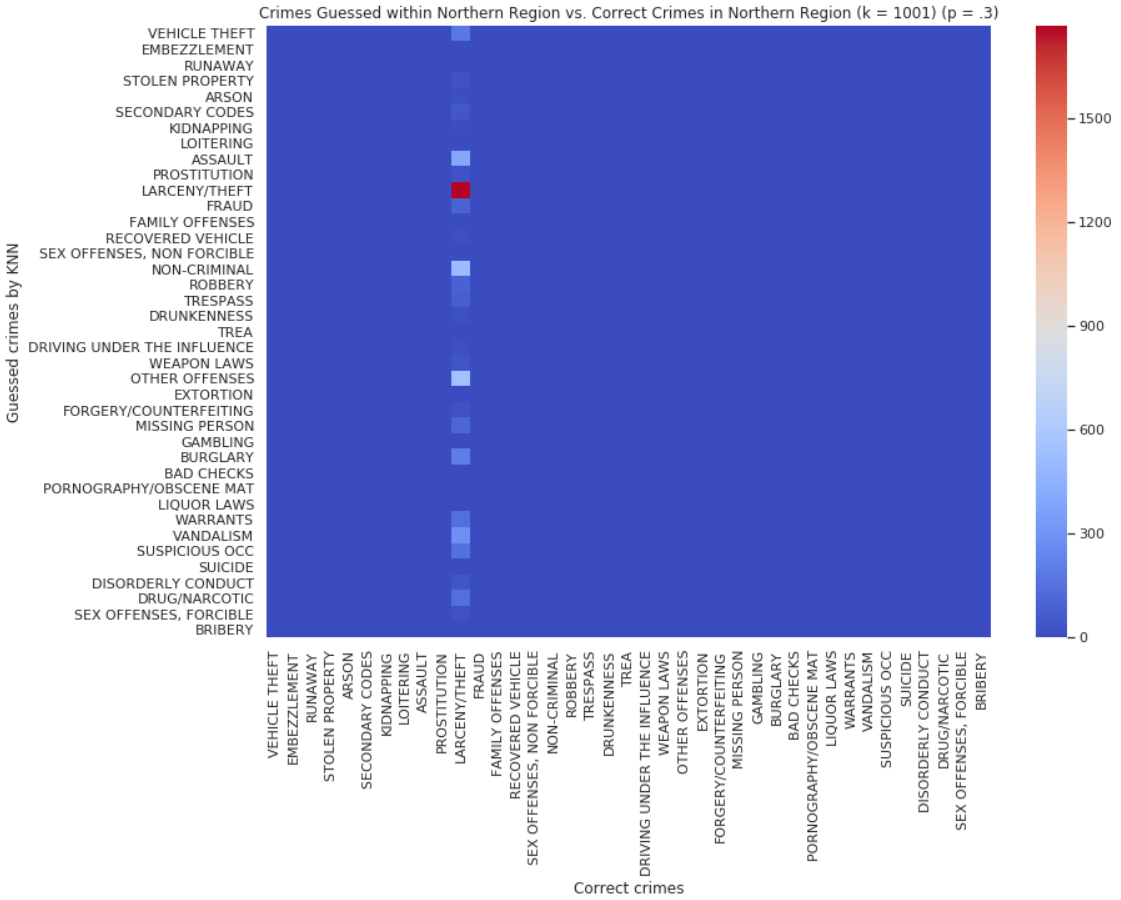


In [19]:
```
        train_northern4, predict_northern4 = split_sets(northern_district, .3)
        northern_test4 = KNN(1001)
```

```
northern_test4.train(train_northern4)
subset_northern = predict_northern4[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_northern = [(northern_test4.predict(tupl[0]), tupl[1]) for tupl in subset_north
```

In [20]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cri

for prediction in sub_cf_northern:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
northern_learning_curve.append([1001, sub_percentage])
#print(northern_learning_curve)
```

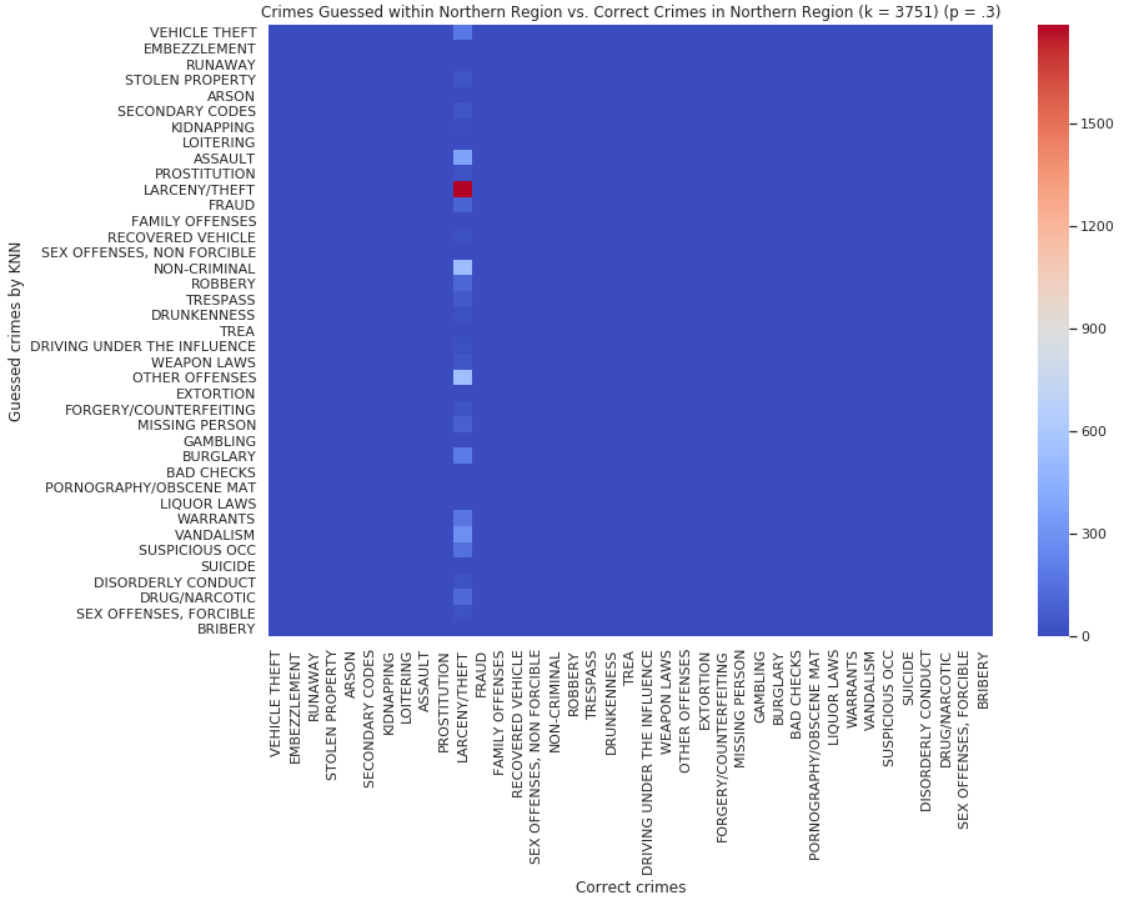
In [21]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
ax.set(title='Crimes Guessed within Northern Region vs. Correct Crimes in Northern Reg
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```




```
In [22]: train_northern5, predict_northern5 = split_sets(northern_district, .3)
         northern_test5 = KNN(3751)
         northern_test5.train(train_northern5)
         subset_northern = predict_northern5[:5000]
         # Creates a list of tuples, (what the algorithm guessed, what is correct)
         sub_cf_northern = [(northern_test5.predict(tupl[0]), tupl[1]) for tupl in subset_northl

In [23]: # confusion_matrix = [([0,0], district) for district in parse_districts]
         # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
         accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cri

         for prediction in sub_cf_northern:
             accum_category_nums[prediction[1]][prediction[0]] += 1
         confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
         #print(confusion_matrix)
         sub_percentage = 0
```

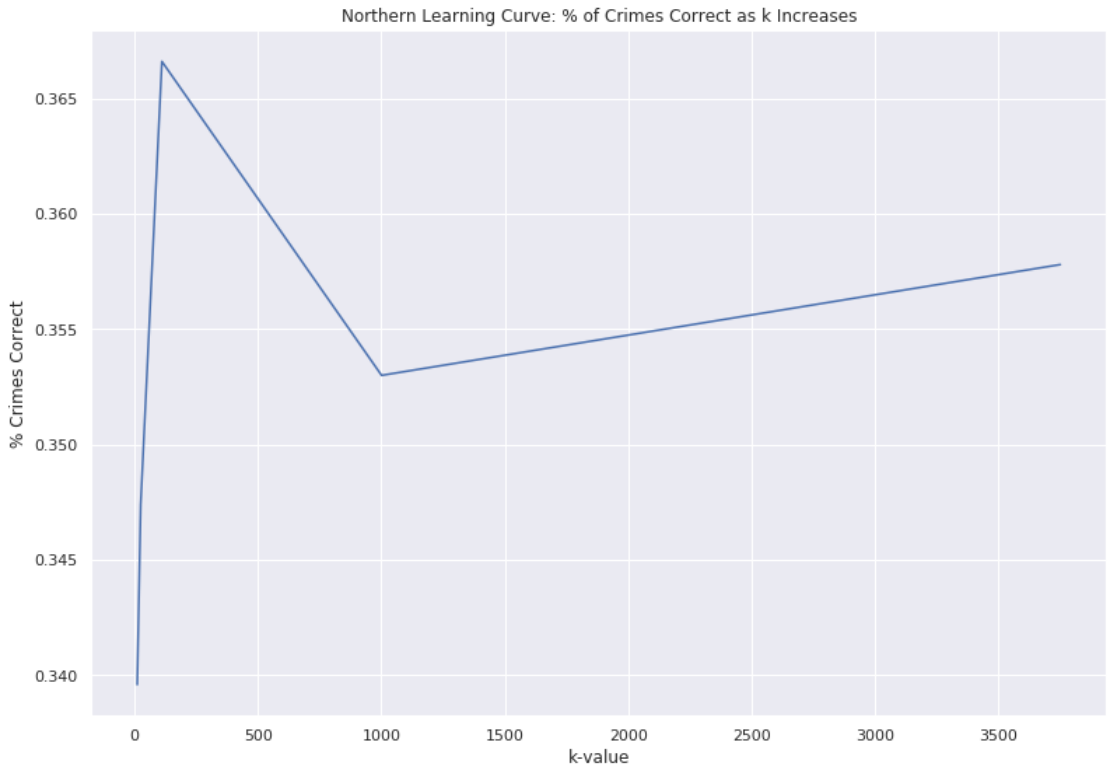
```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        northern_learning_curve.append([3751, sub_percentage])
        #print(northern_learning_curve)
```

In [24]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
        ax.set(title='Crimes Guessed within Northern Region vs. Correct Crimes in Northern Reg
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```



In [25]:
```
        # Printing the learning curve for northern
        # print(northern_learning_curve)
```

```
plt.plot([i[0] for i in northern_learning_curve], [i[1] for i in northern_learning_cu
plt.title('Northern Learning Curve: % of Crimes Correct as k Increases')
plt.xlabel('k-value')
plt.ylabel('% Crimes Correct')
plt.show()
```



15.2.2 Bayview:

```
In [26]: bayview_district = crime_data[crime_data['PdDistrict'] == 'BAYVIEW'][['Category', 'X'
         bayview_district.head()
```

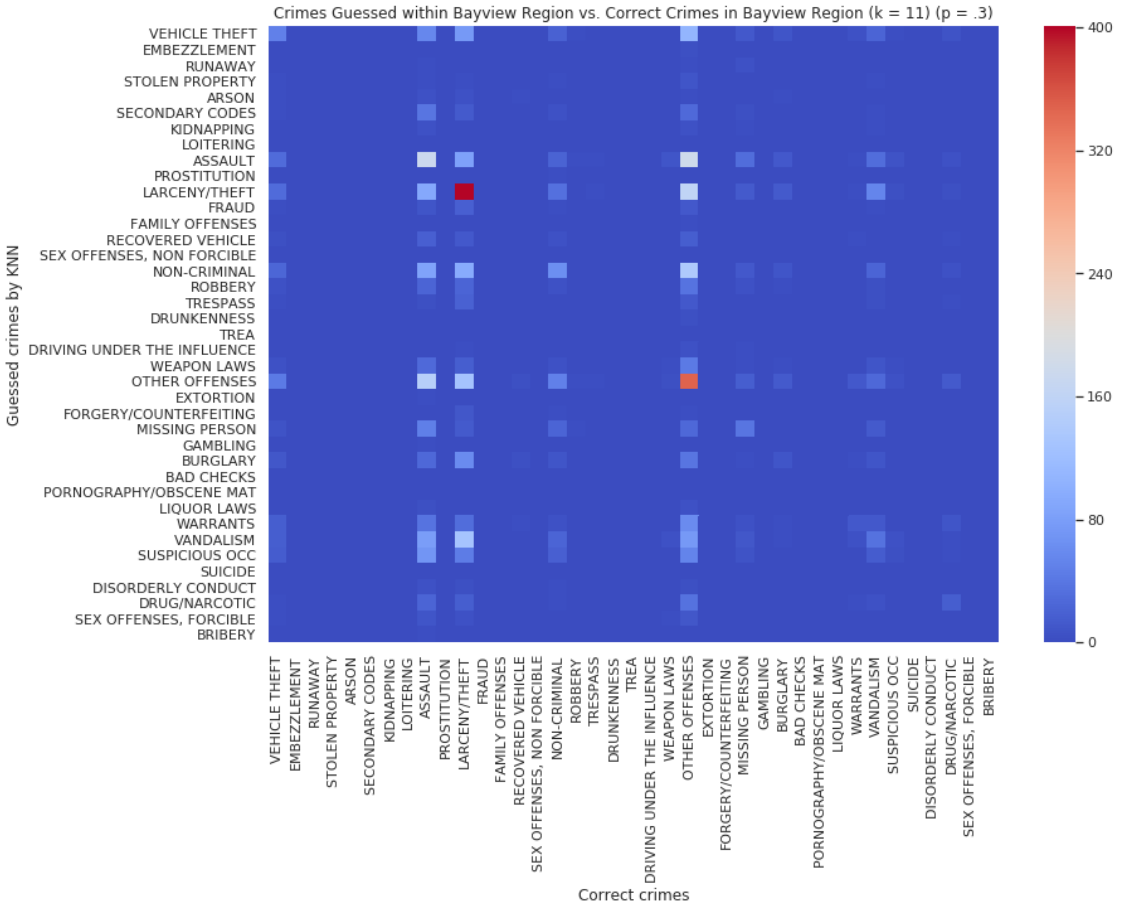
Out[26]:

	Category	X	Y
1	WARRANTS	-122.388856	37.729981
8	MISSING PERSON	-122.387182	37.720967
11	OTHER OFFENSES	-122.376758	37.735697
53	WARRANTS	-122.403409	37.727005
59	OTHER OFFENSES	-122.388753	37.740736

```
In [27]: train_bayview1, predict_bayview1 = split_sets(bayview_district, .3)
         bayview_test1 = KNN(11)
         bayview_test1.train(train_bayview1)
         subset_bayview = predict_bayview1[:5000]
         # Creates a list of tuples, (what the algorithm guessed, what is correct)
         sub_cf_bayview = [(bayview_test1.predict(tupl[0]), tupl[1]) for tupl in subset_bayvie
```

In [28]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cri

# INITIALIZE CONFUSION MATRIX FOR BAYVIEW HERE!!!!!!!!!!!!!!!!!!!!!!!!
bayview_learning_curve = []

for prediction in sub_cf_bayview:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
bayview_learning_curve.append([11, sub_percentage])
#print(bayview_learning_curve)
```

In [29]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
ax.set(title='Crimes Guessed within Bayview Region vs. Correct Crimes in Bayview Regio
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```

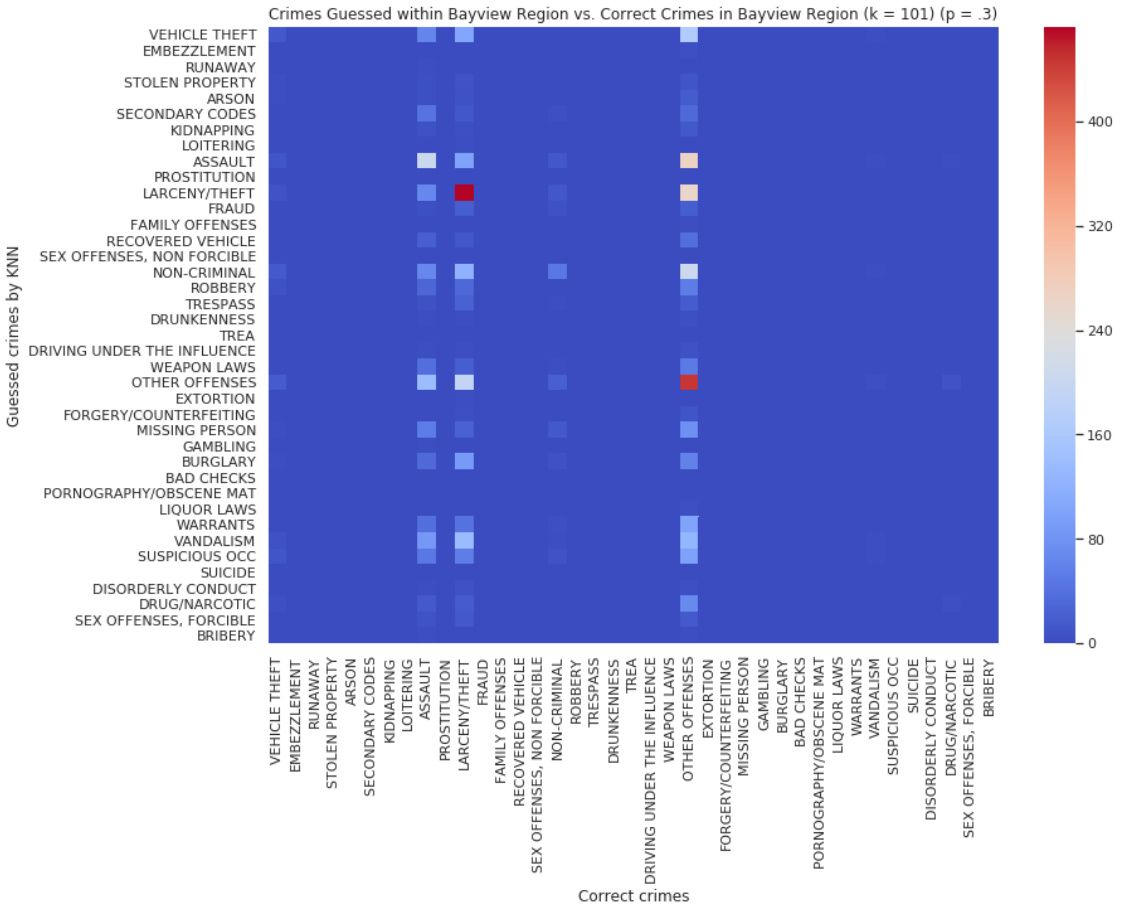



```
In [30]: train_bayview2, predict_bayview2 = split_sets(bayview_district, .3)
         bayview_test2 = KNN(101)
         bayview_test2.train(train_bayview2)
         subset_bayview = predict_bayview2[:5000]
         # Creates a list of tuples, (what the algorithm guessed, what is correct)
         sub_cf_bayview = [(bayview_test2.predict(tupl[0]), tupl[1]) for tupl in subset_bayview
```

```
In [31]: # confusion_matrix = [([0,0], district) for district in parse_districts]
         # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
         accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cri

         for prediction in sub_cf_bayview:
             accum_category_nums[prediction[1]][prediction[0]] += 1
         confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
         #print(confusion_matrix)
         sub_percentage = 0
```

```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        bayview_learning_curve.append([101, sub_percentage])
        #print(bayview_learning_curve)
```

In [32]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
        ax.set(title='Crimes Guessed within Bayview Region vs. Correct Crimes in Bayview Regio
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```

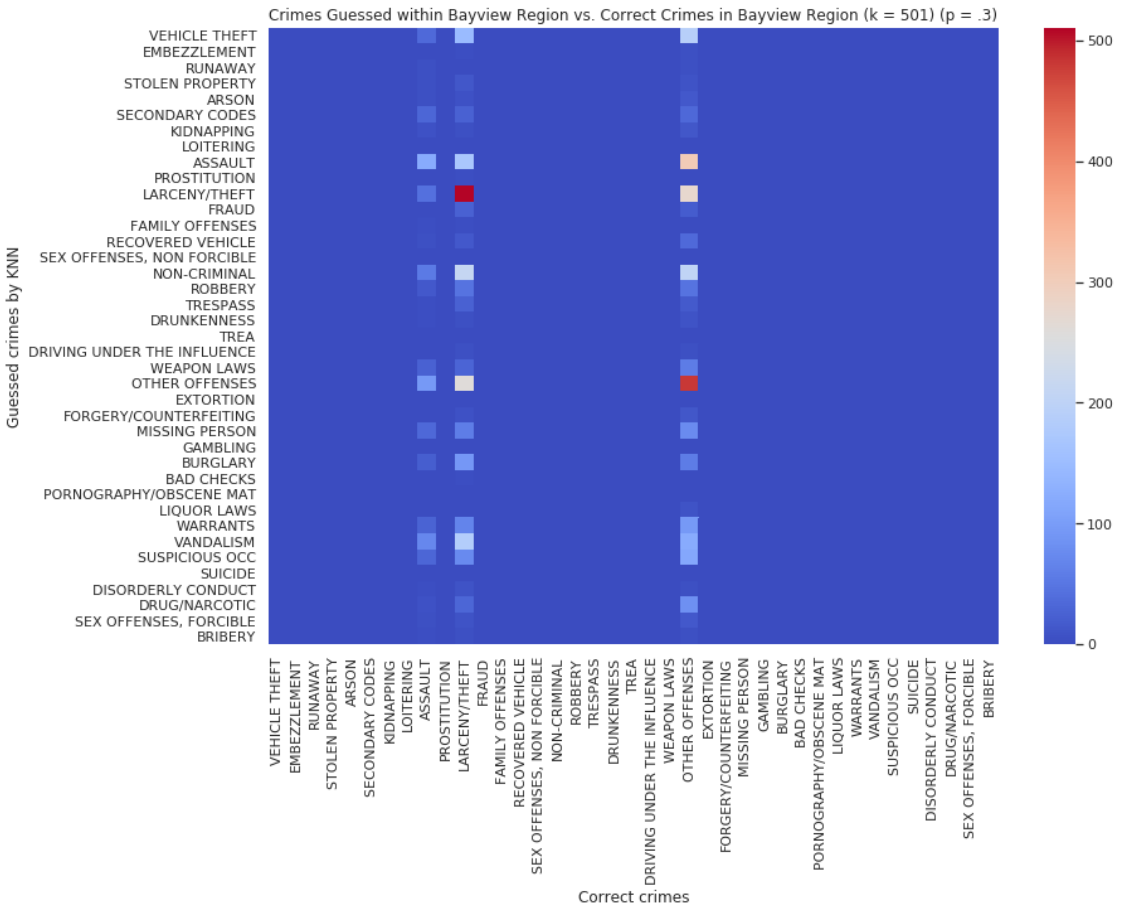


In [33]:
```
        train_bayview3, predict_bayview3 = split_sets(bayview_district, .3)
        bayview_test3 = KNN(501)
```

```
bayview_test3.train(train_bayview3)
subset_bayview = predict_bayview3[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_bayview = [(bayview_test3.predict(tupl[0]), tupl[1]) for tupl in subset_bayvie
```

In [34]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

for prediction in sub_cf_bayview:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
bayview_learning_curve.append([501, sub_percentage])
#print(bayview_learning_curve)
```

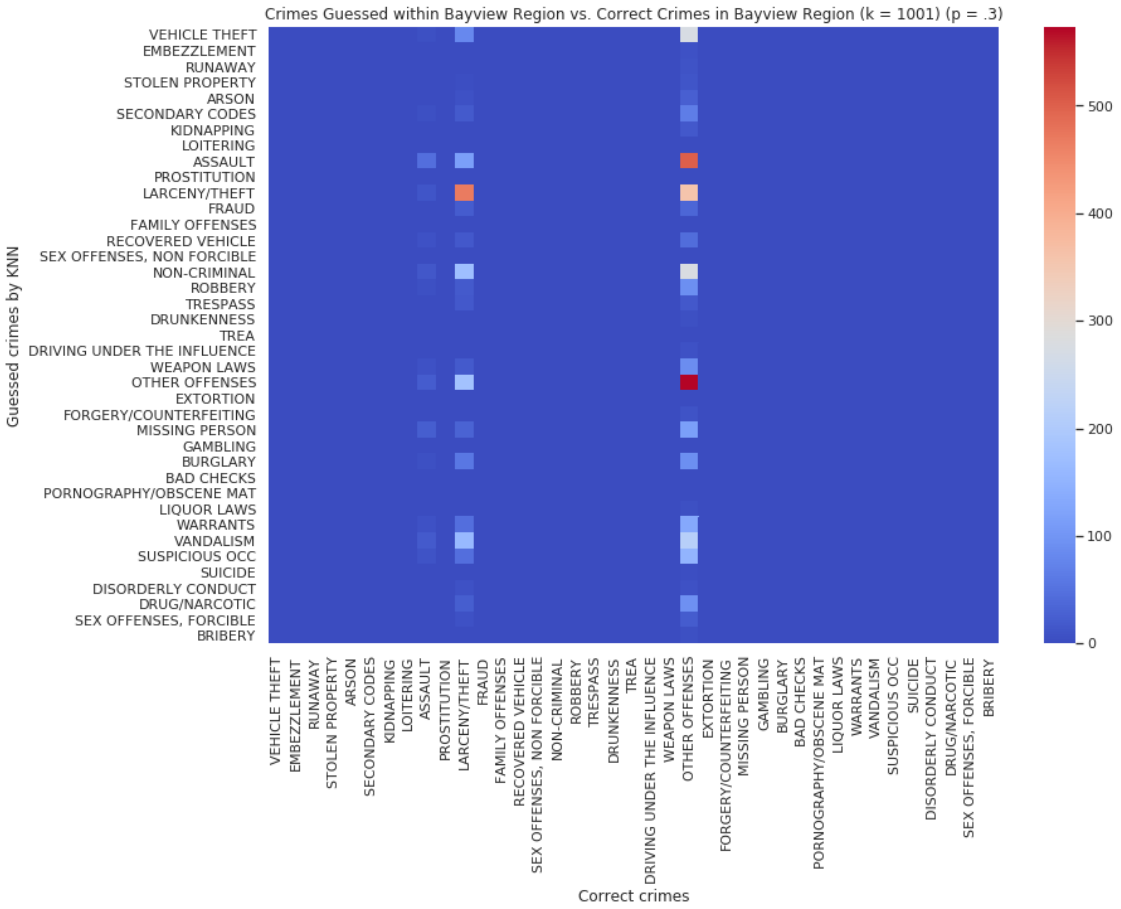
In [35]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
ax.set(title='Crimes Guessed within Bayview Region vs. Correct Crimes in Bayview Regi
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```




```
In [36]: train_bayview4, predict_bayview4 = split_sets(bayview_district, .3)
         bayview_test4 = KNN(1001)
         bayview_test4.train(train_bayview4)
         subset_bayview = predict_bayview4[:5000]
         # Creates a list of tuples, (what the algorithm guessed, what is correct)
         sub_cf_bayview = [(bayview_test4.predict(tupl[0]), tupl[1]) for tupl in subset_bayvie

In [37]: # confusion_matrix = [([0,0], district) for district in parse_districts]
         # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
         accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cri

         for prediction in sub_cf_bayview:
             accum_category_nums[prediction[1]][prediction[0]] += 1
         confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
         #print(confusion_matrix)
         sub_percentage = 0
```

```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        bayview_learning_curve.append([1001, sub_percentage])
        #print(bayview_learning_curve)
```

In [38]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
        ax.set(title='Crimes Guessed within Bayview Region vs. Correct Crimes in Bayview Regio
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```

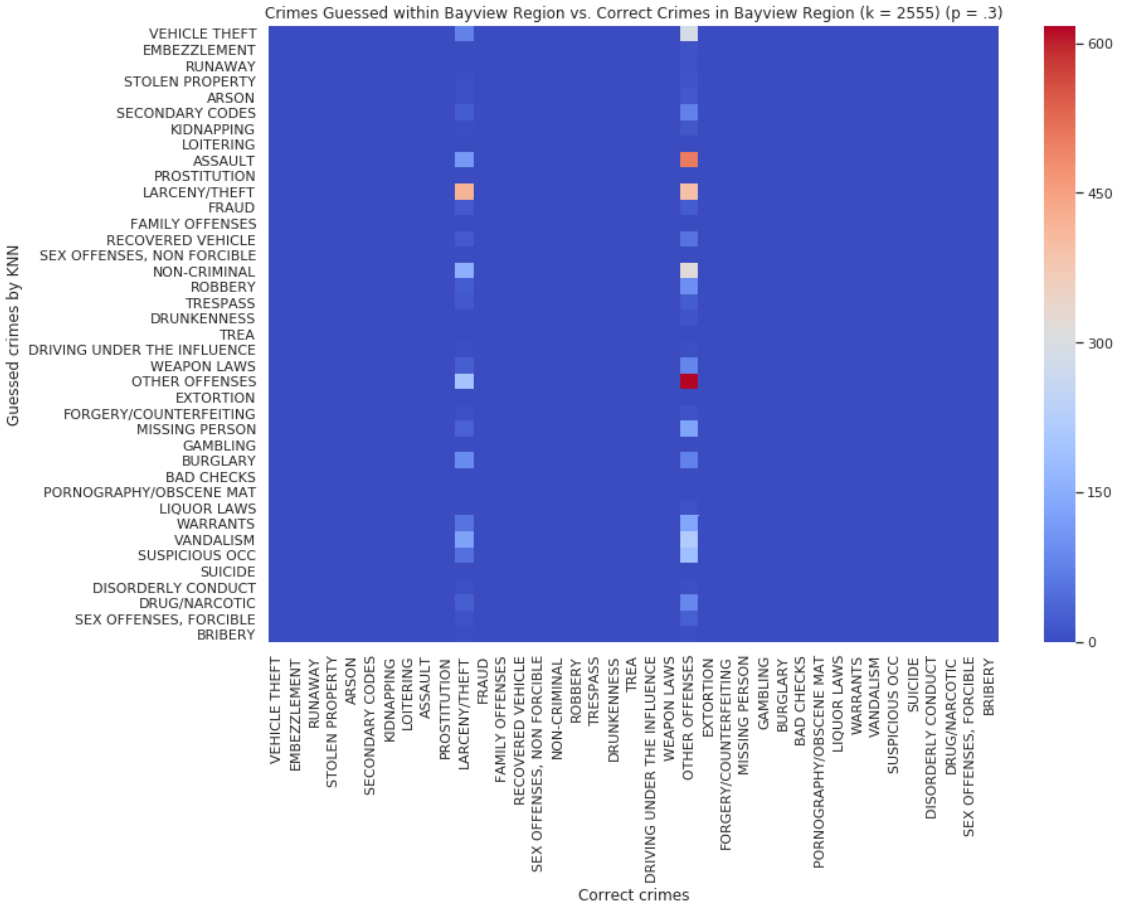


In [39]:
```
        train_bayview5, predict_bayview5 = split_sets(bayview_district, .3)
        bayview_test5 = KNN(2555)
```

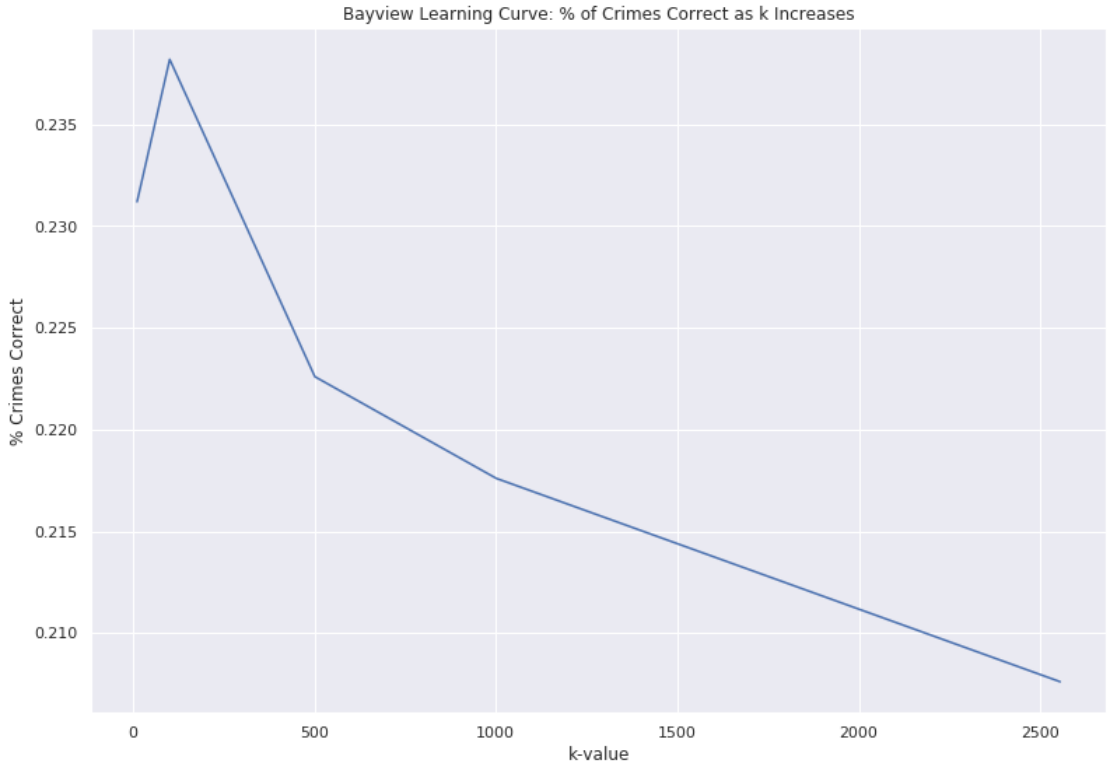
```
bayview_test5.train(train_bayview5)
subset_bayview = predict_bayview5[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_bayview = [(bayview_test5.predict(tupl[0]), tupl[1]) for tupl in subset_bayvie
```

In [40]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

for prediction in sub_cf_bayview:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
bayview_learning_curve.append([2555, sub_percentage])
#print(bayview_learning_curve)
```

In [41]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
ax.set(title='Crimes Guessed within Bayview Region vs. Correct Crimes in Bayview Regi
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```



In [42]:
```
# Printing the learning curve for bayview
# print(bayview_learning_curve)
plt.plot([i[0] for i in bayview_learning_curve], [i[1] for i in bayview_learning_curve
plt.title('Bayview Learning Curve: % of Crimes Correct as k Increases')
plt.xlabel('k-value')
plt.ylabel('% Crimes Correct')
plt.show()
```



15.2.3 Southern:

```
In [43]: southern_district = crime_data[crime_data['PdDistrict'] == 'SOUTHERN'][['Category', '
         southern_district.head()
```

Out[43]:

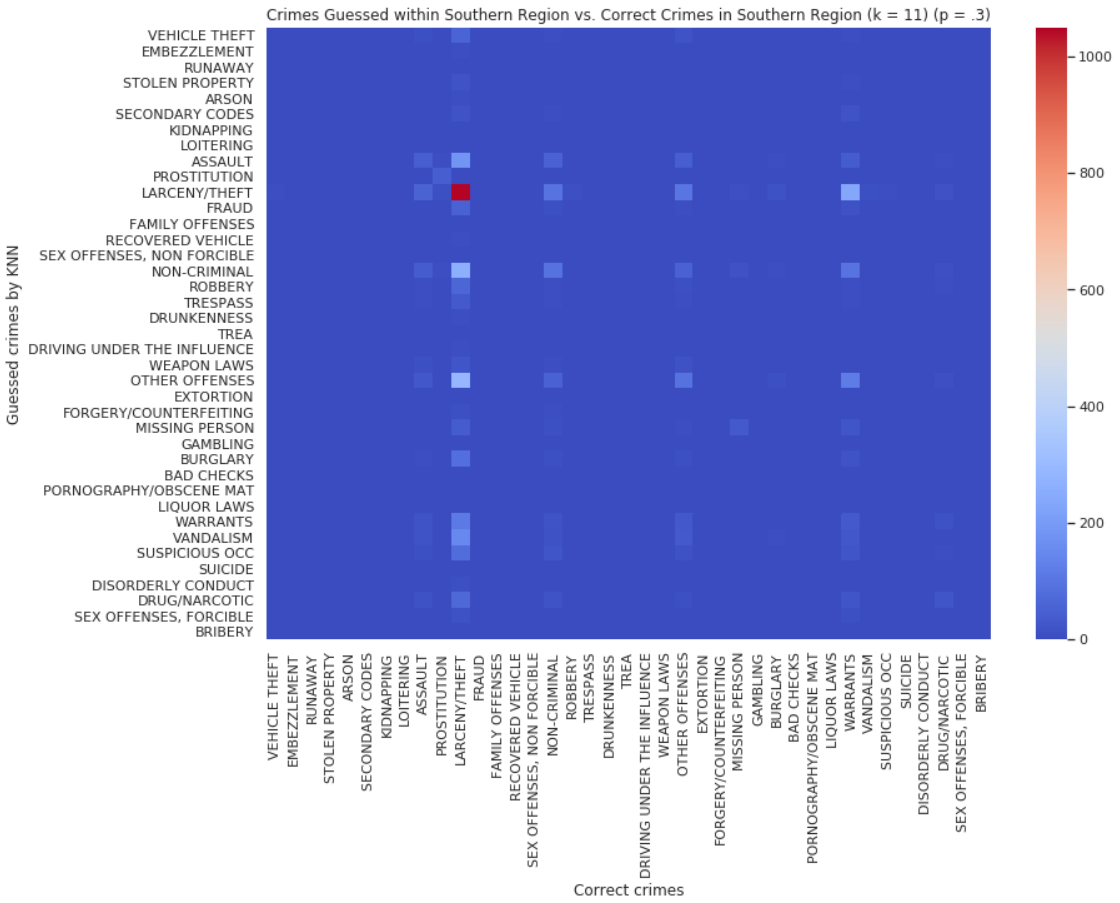
	Category	X	Y
0	WEAPON LAWS	-122.403405	37.775421
5	OTHER OFFENSES	-122.405721	37.780879
7	WARRANTS	-122.393357	37.775788
14	STOLEN PROPERTY	-122.408595	37.783707
18	LARCENY/THEFT	-122.408421	37.783570

```
In [44]: train_southern1, predict_southern1 = split_sets(southern_district, .3)
         southern_test1 = KNN(11)
         southern_test1.train(train_southern1)
         subset_southern = predict_southern1[:5000]
         # Creates a list of tuples, (what the algorithm guessed, what is correct)
         sub_cf_southern = [(southern_test1.predict(tupl[0]), tupl[1]) for tupl in subset_sout
```

```
In [45]: # confusion_matrix = [([0,0], district) for district in parse_districts]
         # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
         accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr
```

```
# INITIALIZE CONFUSION MATRIX FOR Southern HERE!!!!!!!!!!!!!!!!!!!!!!!!
southern_learning_curve = []

for prediction in sub_cf_southern:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
southern_learning_curve.append([11, sub_percentage])
#print(southern_learning_curve)
```

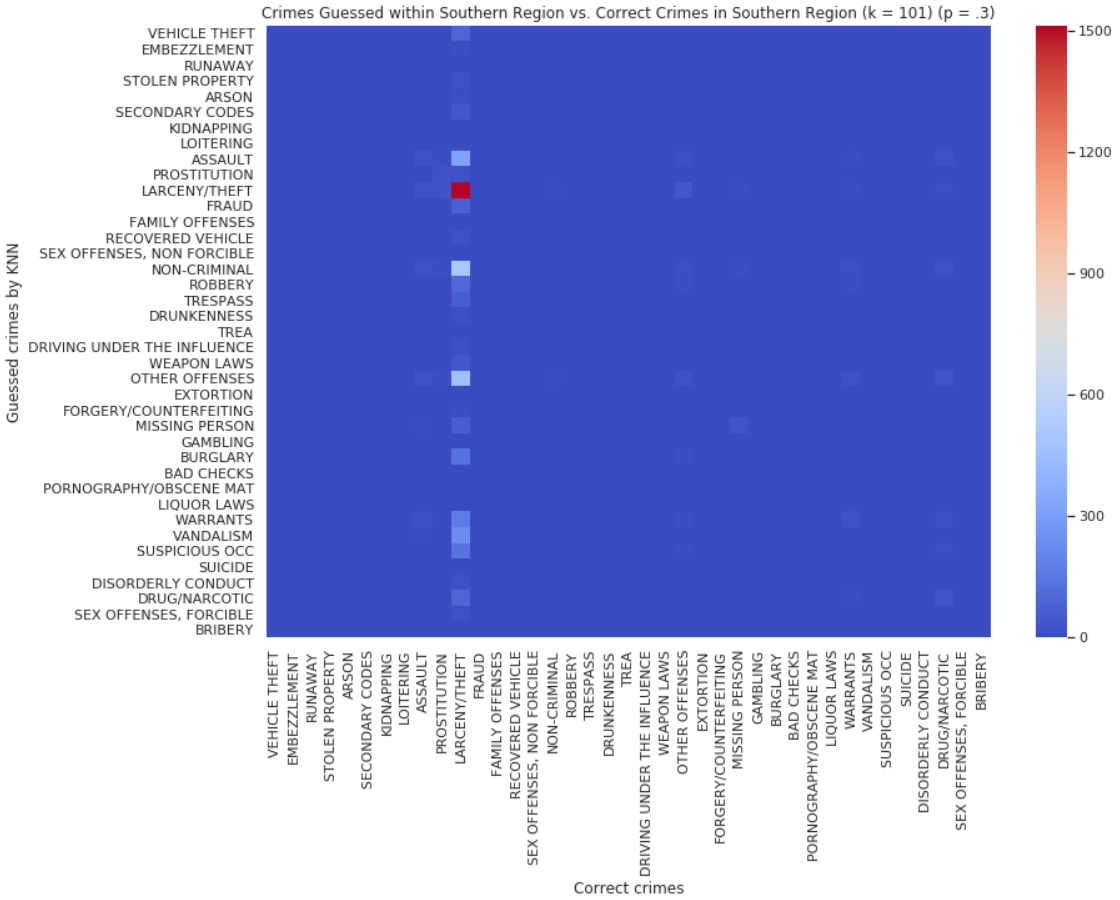
In [46]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
ax.set(title='Crimes Guessed within Southern Region vs. Correct Crimes in Southern Reg
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```



In [47]:
```
train_southern2, predict_southern2 = split_sets(southern_district, .3)
southern_test2 = KNN(101)
southern_test2.train(train_southern2)
subset_southern = predict_southern2[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_southern = [(southern_test2.predict(tupl[0]), tupl[1]) for tupl in subset_sout
```

In [48]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

for prediction in sub_cf_southern:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
southern_learning_curve.append([101, sub_percentage])
#print(southern_learning_curve)
```

In [49]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
ax.set(title='Crimes Guessed within Southern Region vs. Correct Crimes in Southern Reg
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```

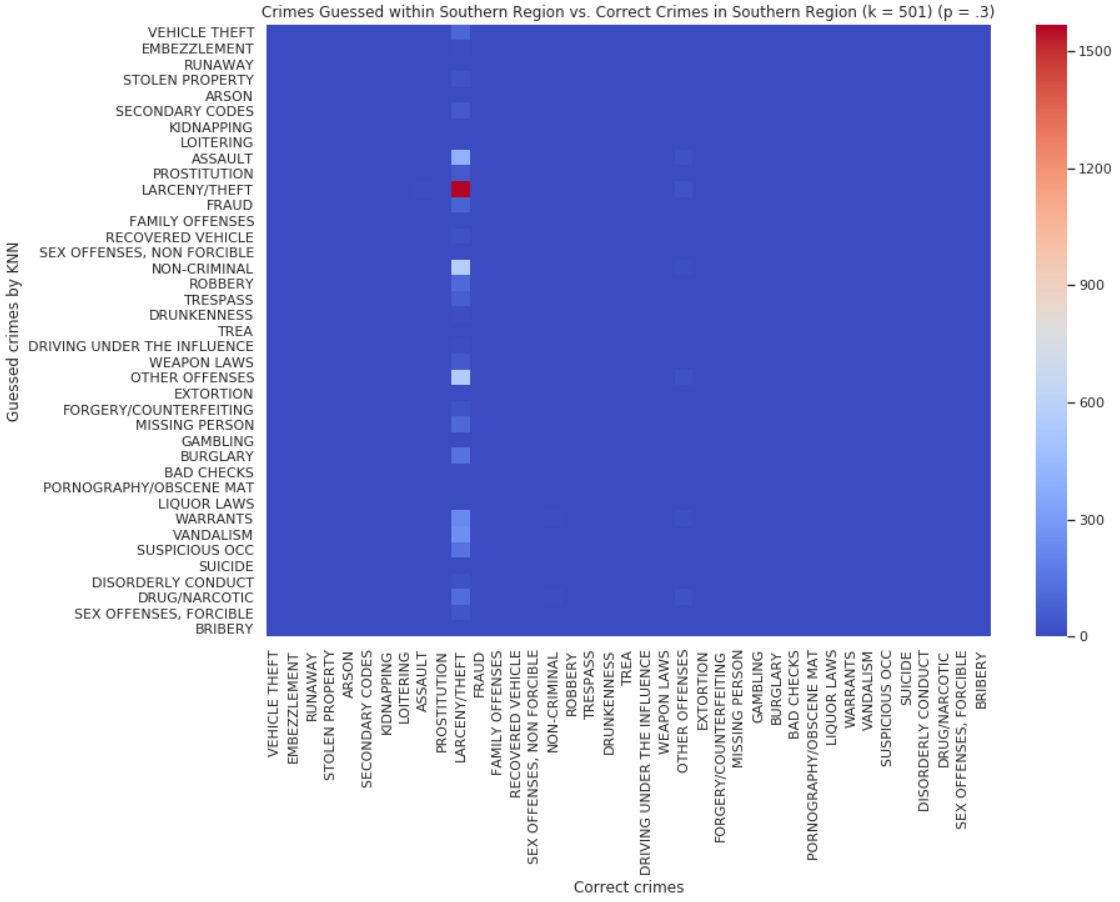



```
In [50]: train_southern3, predict_southern3 = split_sets(southern_district, .3)
         southern_test3 = KNN(501)
         southern_test3.train(train_southern3)
         subset_southern = predict_southern3[:5000]
         # Creates a list of tuples, (what the algorithm guessed, what is correct)
         sub_cf_southern = [(southern_test3.predict(tupl[0]), tupl[1]) for tupl in subset_southern
```

```
In [51]: # confusion_matrix = [([0,0], district) for district in parse_districts]
         # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
         accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cri

         for prediction in sub_cf_southern:
             accum_category_nums[prediction[1]][prediction[0]] += 1
         confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
         #print(confusion_matrix)
         sub_percentage = 0
```

```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        southern_learning_curve.append([501, sub_percentage])
        #print(southern_learning_curve)
```

In [52]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
        ax.set(title='Crimes Guessed within Southern Region vs. Correct Crimes in Southern Reg
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```

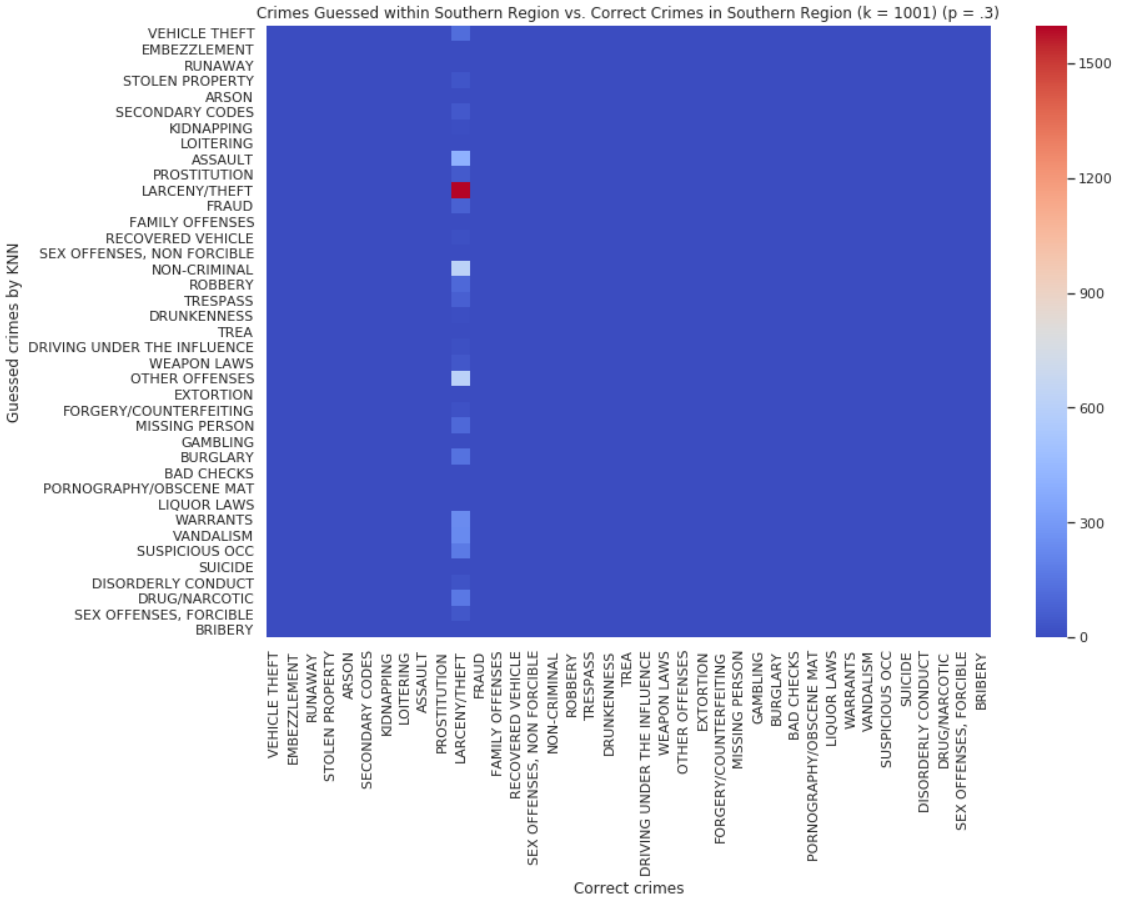


In [53]:
```
        train_southern4, predict_southern4 = split_sets(southern_district, .3)
        southern_test4 = KNN(1001)
```

```
southern_test4.train(train_southern4)
subset_southern = predict_southern4[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_southern = [(southern_test4.predict(tupl[0]), tupl[1]) for tupl in subset_south
```

In [54]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cri

for prediction in sub_cf_southern:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
southern_learning_curve.append([1001, sub_percentage])
#print(southern_learning_curve)
```

In [55]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
ax.set(title='Crimes Guessed within Southern Region vs. Correct Crimes in Southern Reg
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```

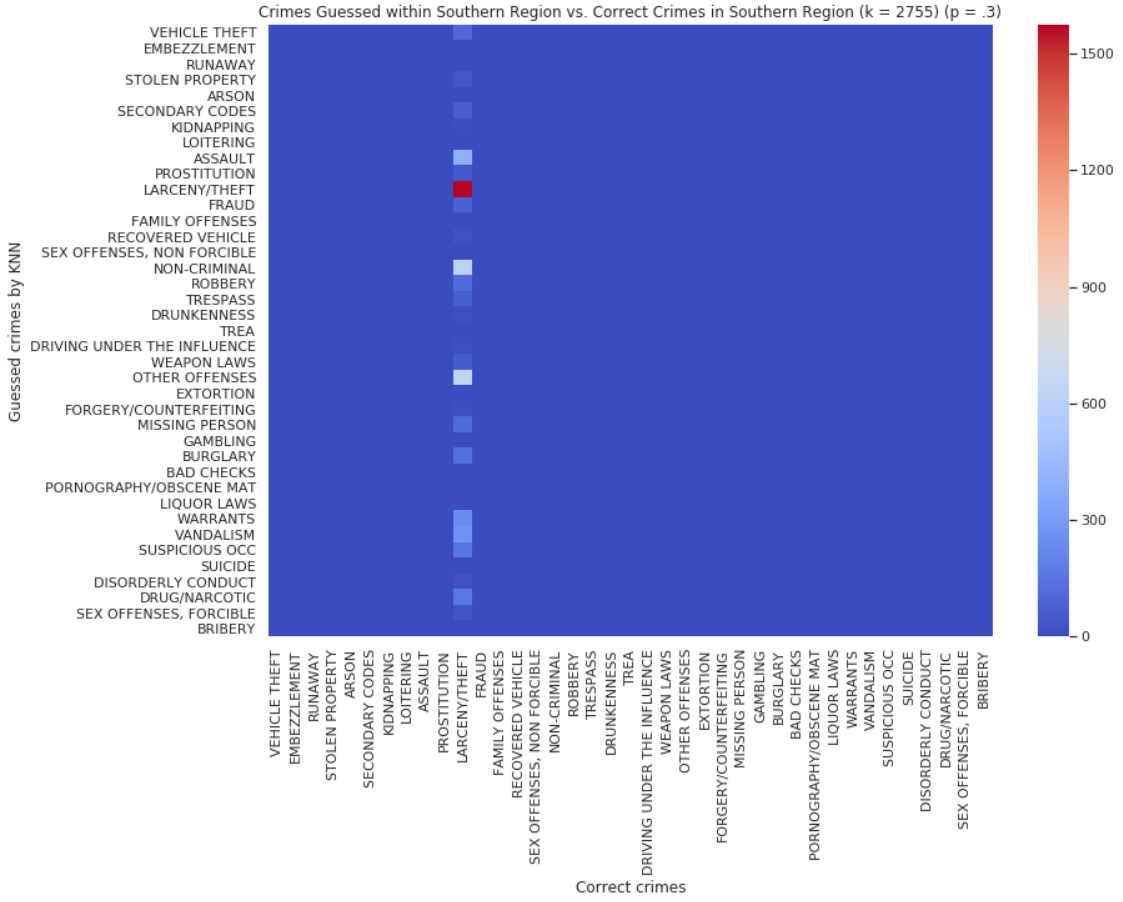



```
In [56]: train_southern5, predict_southern5 = split_sets(southern_district, .3)
         southern_test5 = KNN(2755)
         southern_test5.train(train_southern5)
         subset_southern = predict_southern5[:5000]
         # Creates a list of tuples, (what the algorithm guessed, what is correct)
         sub_cf_southern = [(southern_test5.predict(tupl[0]), tupl[1]) for tupl in subset_southern
```

```
In [57]: # confusion_matrix = [([0,0], district) for district in parse_districts]
         # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
         accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cri

         for prediction in sub_cf_southern:
             accum_category_nums[prediction[1]][prediction[0]] += 1
         confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
         #print(confusion_matrix)
         sub_percentage = 0
```

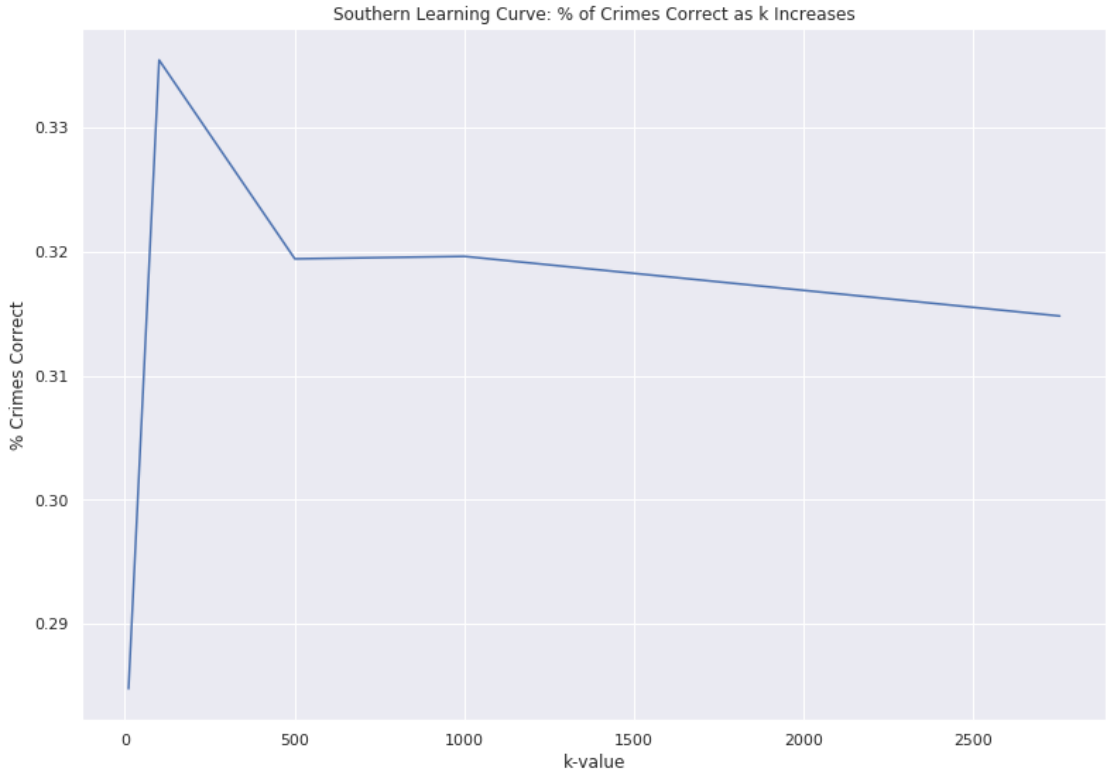
```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        southern_learning_curve.append([2755, sub_percentage])
        #print(southern_learning_curve)
```

In [58]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
        ax.set(title='Crimes Guessed within Southern Region vs. Correct Crimes in Southern Reg
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```



In [59]:
```
        # Printing the learning curve for southern
        # print(richmond_learning_curve)
```

```
plt.plot([i[0] for i in southern_learning_curve], [i[1] for i in southern_learning_cu
plt.title('Southern Learning Curve: % of Crimes Correct as k Increases')
plt.xlabel('k-value')
plt.ylabel('% Crimes Correct')
plt.show()
```



15.2.4 Tenderloin:

In [60]:
```
tenderloin_district = crime_data[crime_data['PdDistrict'] == 'TENDERLOIN'][['Category
tenderloin_district.head()
```

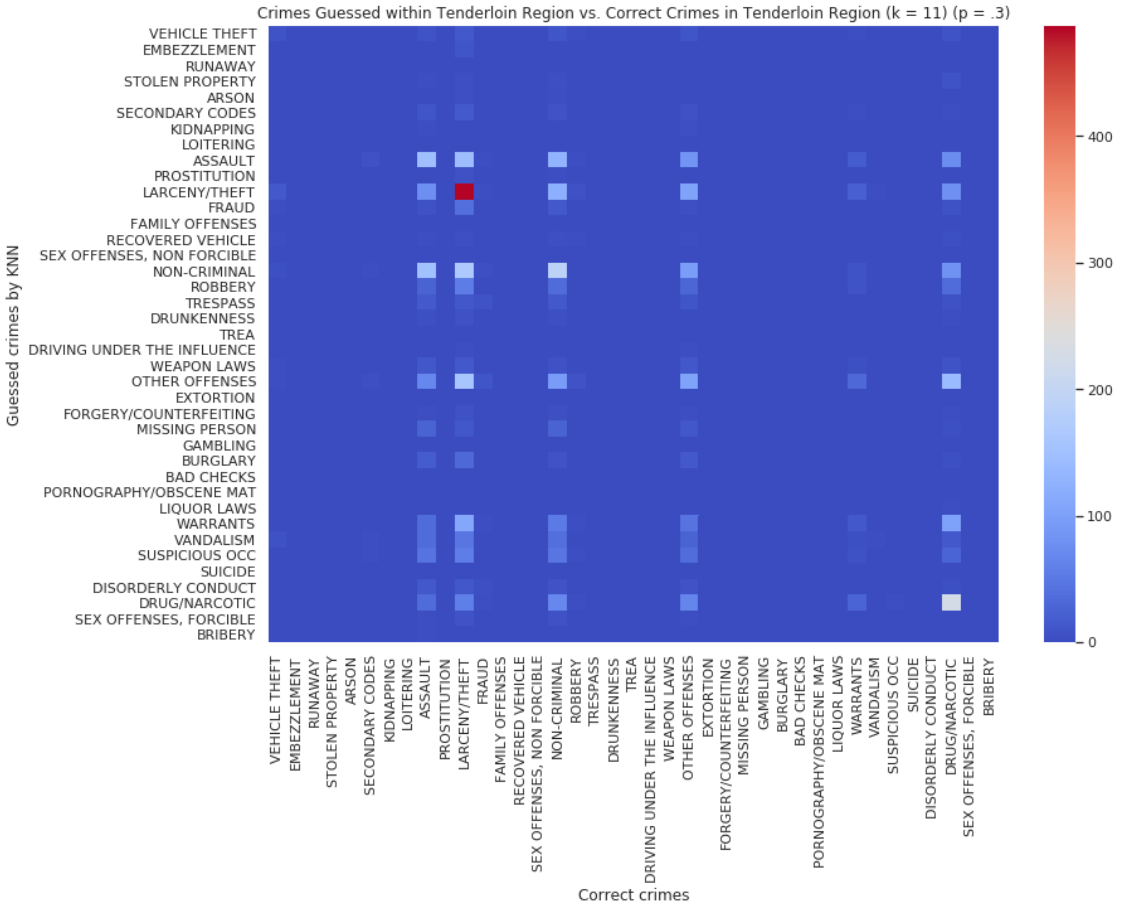
Out[60]:

	Category	X	Y
2	NON-CRIMINAL	-122.412971	37.785788
6	NON-CRIMINAL	-122.411778	37.783981
32	ASSAULT	-122.407244	37.786565
39	VANDALISM	-122.415670	37.782120
54	VANDALISM	-122.412931	37.783834

In [61]:
```
train_tenderloin1, predict_tenderloin1 = split_sets(tenderloin_district, .3)
tenderloin_test1 = KNN(11)
tenderloin_test1.train(train_tenderloin1)
subset_tenderloin = predict_tenderloin1[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_tenderloin = [(tenderloin_test1.predict(tupl[0]), tupl[1]) for tupl in subset_
```

In [62]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cri

# INITIALIZE CONFUSION MATRIX FOR BAYVIEW HERE!!!!!!!!!!!!!!!!!!!!!!!!
tenderloin_learning_curve = []

for prediction in sub_cf_tenderloin:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
tenderloin_learning_curve.append([11, sub_percentage])
#print(tenderloin_learning_curve)
```

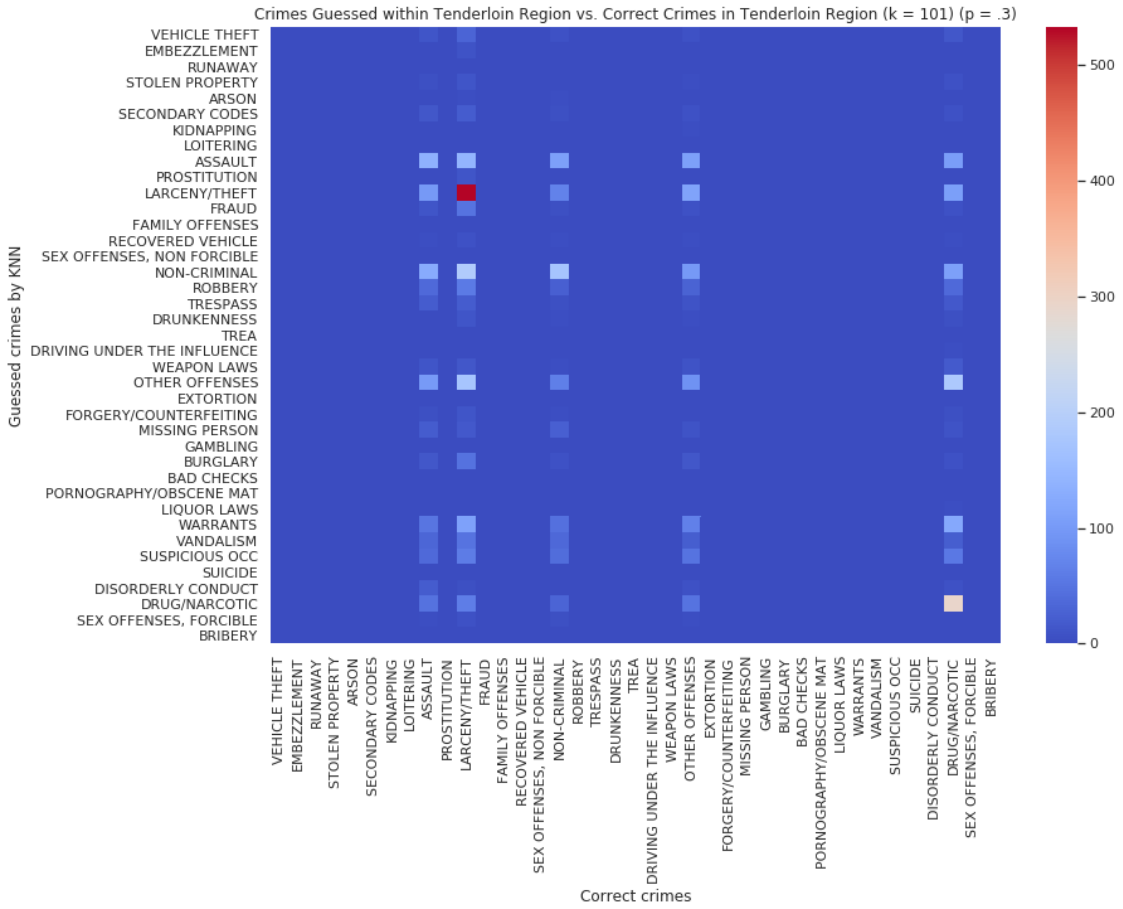
In [63]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
ax.set(title='Crimes Guessed within Tenderloin Region vs. Correct Crimes in Tenderloi
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```




```
In [64]: train_tenderloin2, predict_tenderloin2 = split_sets(tenderloin_district, .3)
         tenderloin_test2 = KNN(101)
         tenderloin_test2.train(train_tenderloin2)
         subset_tenderloin = predict_tenderloin2[:5000]
         # Creates a list of tuples, (what the algorithm guessed, what is correct)
         sub_cf_tenderloin = [(tenderloin_test2.predict(tupl[0]), tupl[1]) for tupl in subset_t

In [65]: # confusion_matrix = [([0,0], district) for district in parse_districts]
         # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
         accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cri

         for prediction in sub_cf_tenderloin:
             accum_category_nums[prediction[1]][prediction[0]] += 1
         confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
         #print(confusion_matrix)
         sub_percentage = 0
```

```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        tenderloin_learning_curve.append([101, sub_percentage])
        #print(tenderloin_learning_curve)
```

In [66]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
        ax.set(title='Crimes Guessed within Tenderloin Region vs. Correct Crimes in Tenderloi
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```

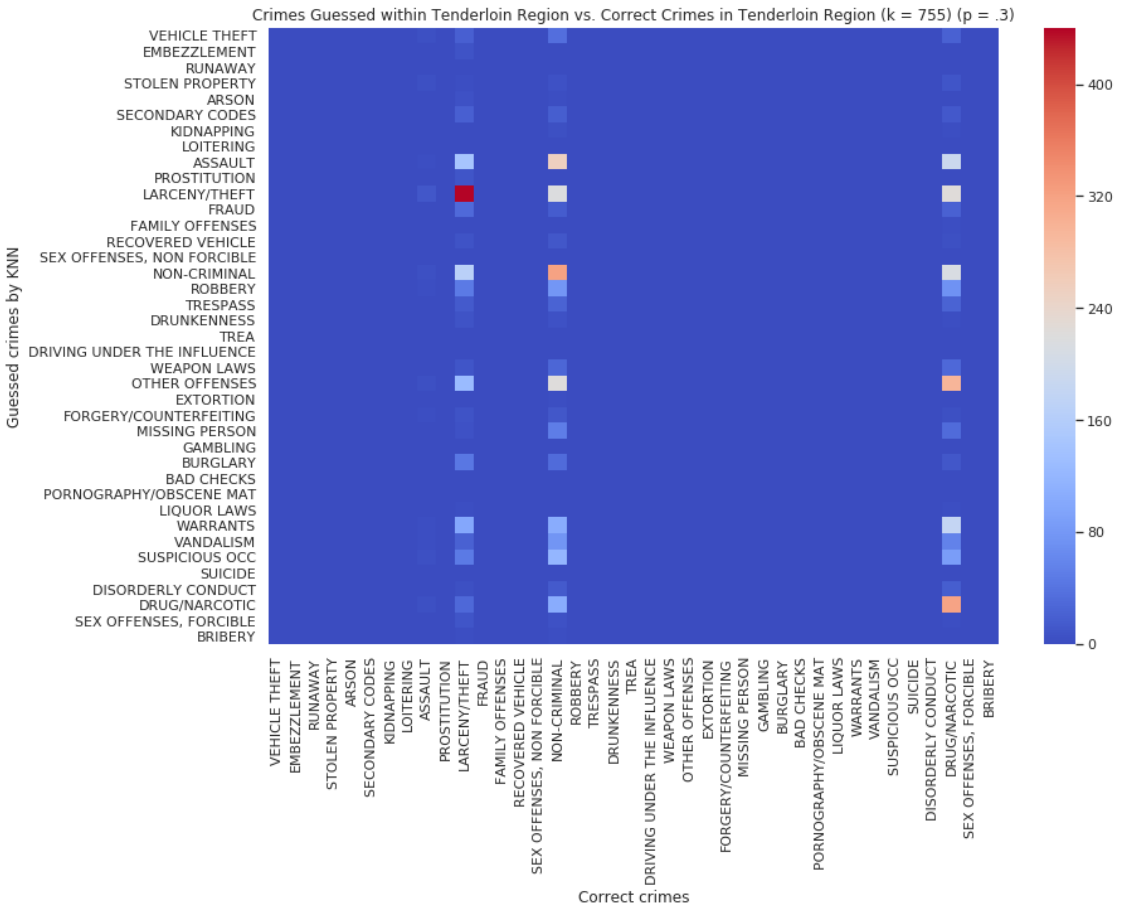


In [67]:
```
        train_tenderloin3, predict_tenderloin3 = split_sets(tenderloin_district, .3)
        tenderloin_test3 = KNN(755)
```

```
tenderloin_test3.train(train_tenderloin3)
subset_tenderloin = predict_tenderloin3[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_tenderloin = [(tenderloin_test3.predict(tupl[0]), tupl[1]) for tupl in subset_t
```

In [68]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cri

for prediction in sub_cf_tenderloin:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
tenderloin_learning_curve.append([755, sub_percentage])
#print(tenderloin_learning_curve)
```

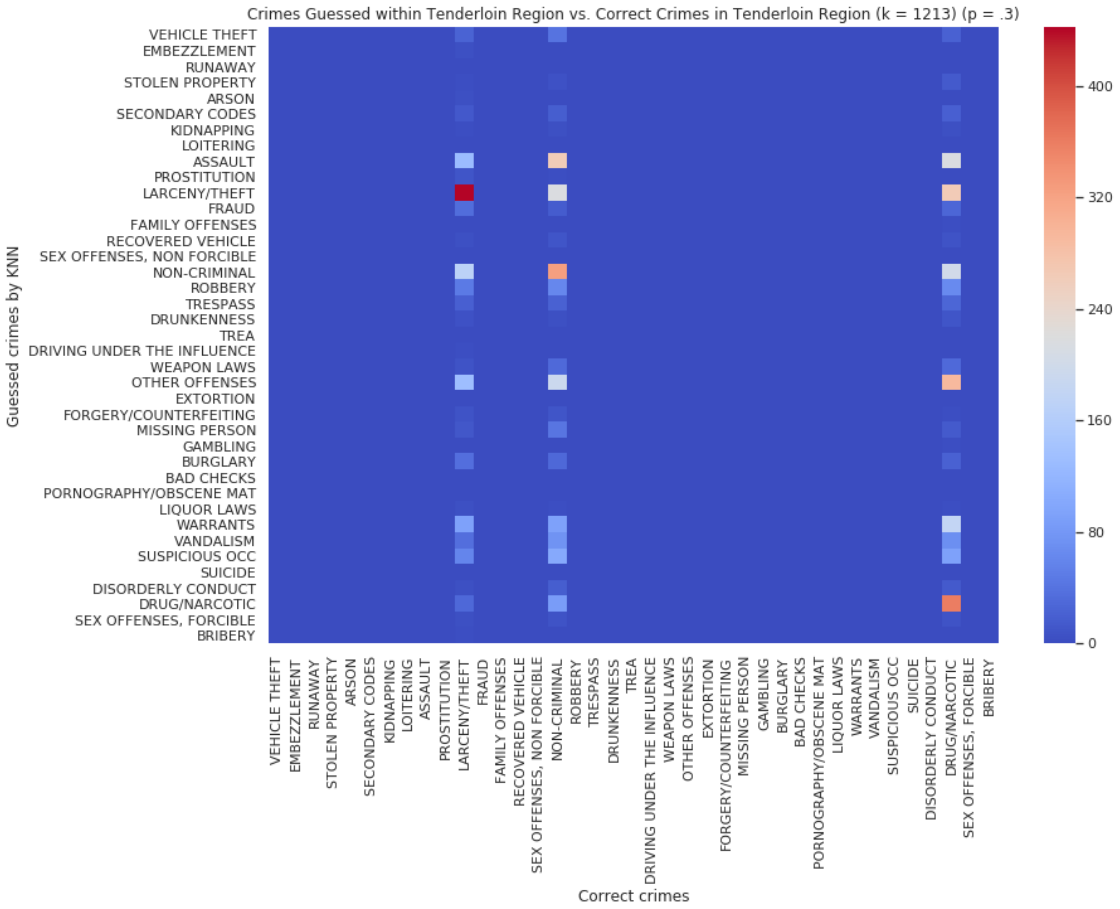
In [69]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
ax.set(title='Crimes Guessed within Tenderloin Region vs. Correct Crimes in Tenderloi
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```




```
In [70]: train_tenderloin4, predict_tenderloin4 = split_sets(tenderloin_district, .3)
         tenderloin_test4 = KNN(1213)
         tenderloin_test4.train(train_tenderloin4)
         subset_tenderloin = predict_tenderloin4[:5000]
         # Creates a list of tuples, (what the algorithm guessed, what is correct)
         sub_cf_tenderloin = [(tenderloin_test4.predict(tupl[0]), tupl[1]) for tupl in subset_t

In [71]: # confusion_matrix = [([0,0], district) for district in parse_districts]
         # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
         accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cri

         for prediction in sub_cf_tenderloin:
             accum_category_nums[prediction[1]][prediction[0]] += 1
         confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
         #print(confusion_matrix)
         sub_percentage = 0
```

```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        tenderloin_learning_curve.append([1213, sub_percentage])
        #print(tenderloin_learning_curve)
```

In [72]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
        ax.set(title='Crimes Guessed within Tenderloin Region vs. Correct Crimes in Tenderloi
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```

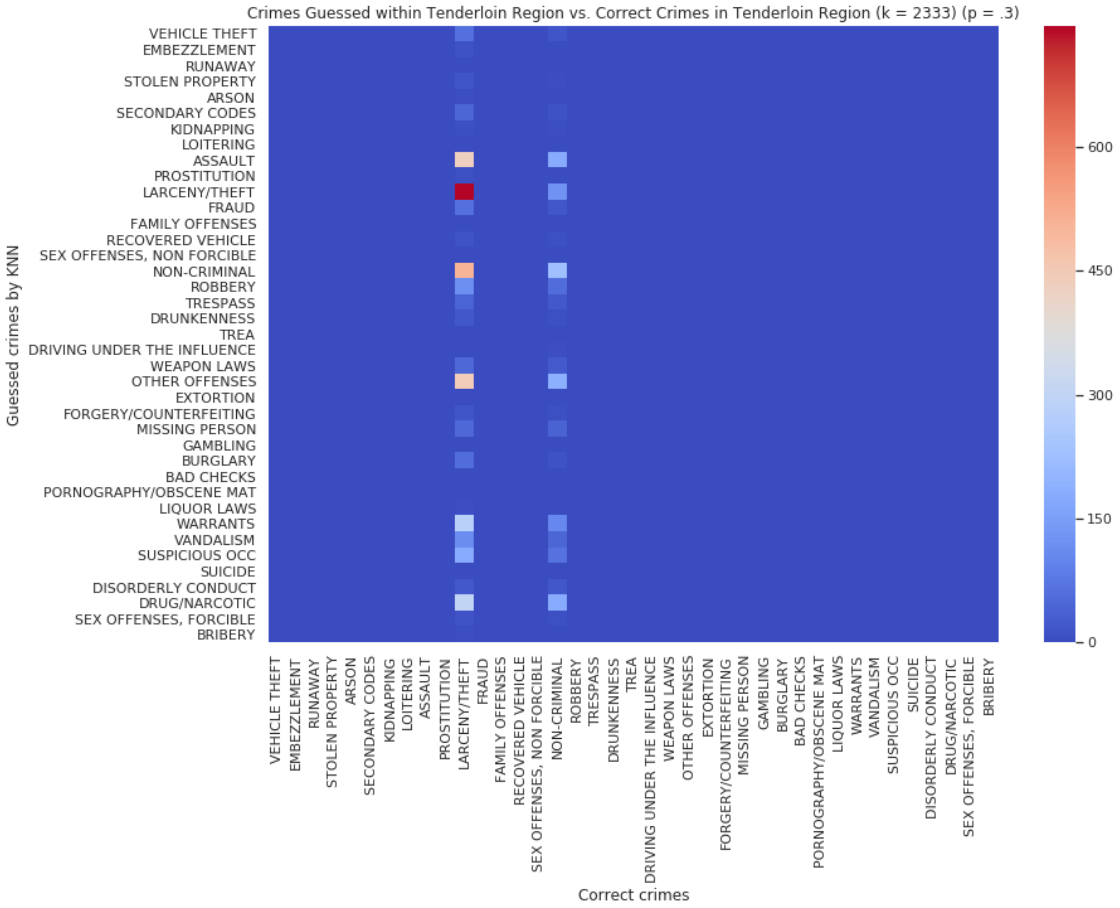


In [73]:
```
        train_tenderloin5, predict_tenderloin5 = split_sets(tenderloin_district, .3)
        tenderloin_test5 = KNN(2333)
```

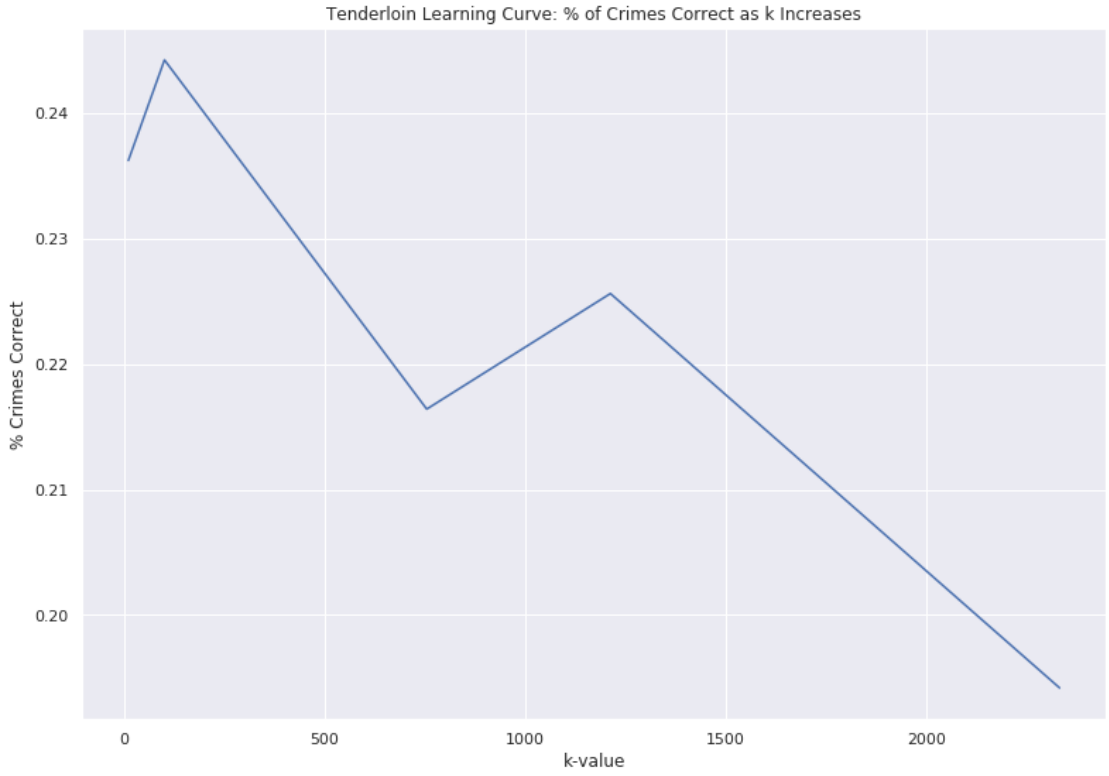
```
tenderloin_test5.train(train_tenderloin5)
subset_tenderloin = predict_tenderloin5[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_tenderloin = [(tenderloin_test5.predict(tupl[0]), tupl[1]) for tupl in subset_t
```

In [74]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cri

for prediction in sub_cf_tenderloin:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
tenderloin_learning_curve.append([2333, sub_percentage])
#print(tenderloin_learning_curve)
```

In [75]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
ax.set(title='Crimes Guessed within Tenderloin Region vs. Correct Crimes in Tenderloi
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```



In [76]:
```
# Printing the learning curve for tenderloin
# print(richmond_learning_curve)
plt.plot([i[0] for i in tenderloin_learning_curve], [i[1] for i in tenderloin_learning
plt.title('Tenderloin Learning Curve: % of Crimes Correct as k Increases')
plt.xlabel('k-value')
plt.ylabel('% Crimes Correct')
plt.show()
```



15.2.5 Mission:

```
In [77]: mission_district = crime_data[crime_data['PdDistrict'] == 'MISSION'][['Category', 'X'
         mission_district.head()
```

Out[77]:

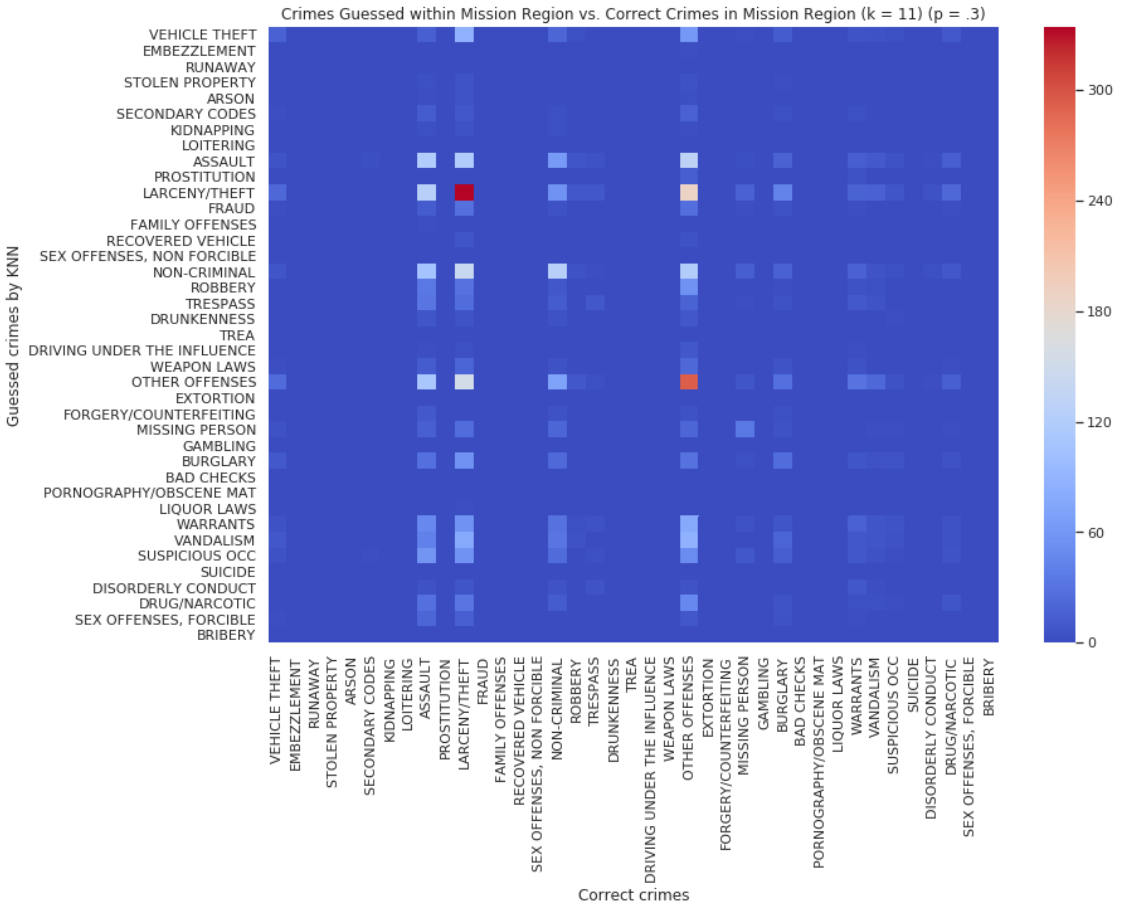
	Category	X	Y
3	NON-CRIMINAL	-122.419672	37.765050
15	ROBBERY	-122.406870	37.757290
16	ASSAULT	-122.420355	37.748906
21	DRUG/NARCOTIC	-122.414234	37.754099
22	OTHER OFFENSES	-122.414234	37.754099

```
In [78]: train_mission1, predict_mission1 = split_sets(mission_district, .3)
         mission_test1 = KNN(11)
         mission_test1.train(train_mission1)
         subset_mission = predict_mission1[:5000]
         # Creates a list of tuples, (what the algorithm guessed, what is correct)
         sub_cf_mission = [(mission_test1.predict(tupl[0]), tupl[1]) for tupl in subset_missio
```

```
In [79]: # confusion_matrix = [([0,0], district) for district in parse_districts]
         # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
         accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr
```

```
    # INITIALIZE CONFUSION MATRIX FOR MISSION HERE!!!!!!!!!!!!!!!!!!!!!!!!
    mission_learning_curve = []

    for prediction in sub_cf_mission:
        accum_category_nums[prediction[1]][prediction[0]] += 1
    confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
    #print(confusion_matrix)
    sub_percentage = 0
    for ind, row in enumerate(confusion_matrix):
        sub_percentage += row[ind]
    sub_percentage /= 5000
    mission_learning_curve.append([11, sub_percentage])
    #print(mission_learning_curve)
```

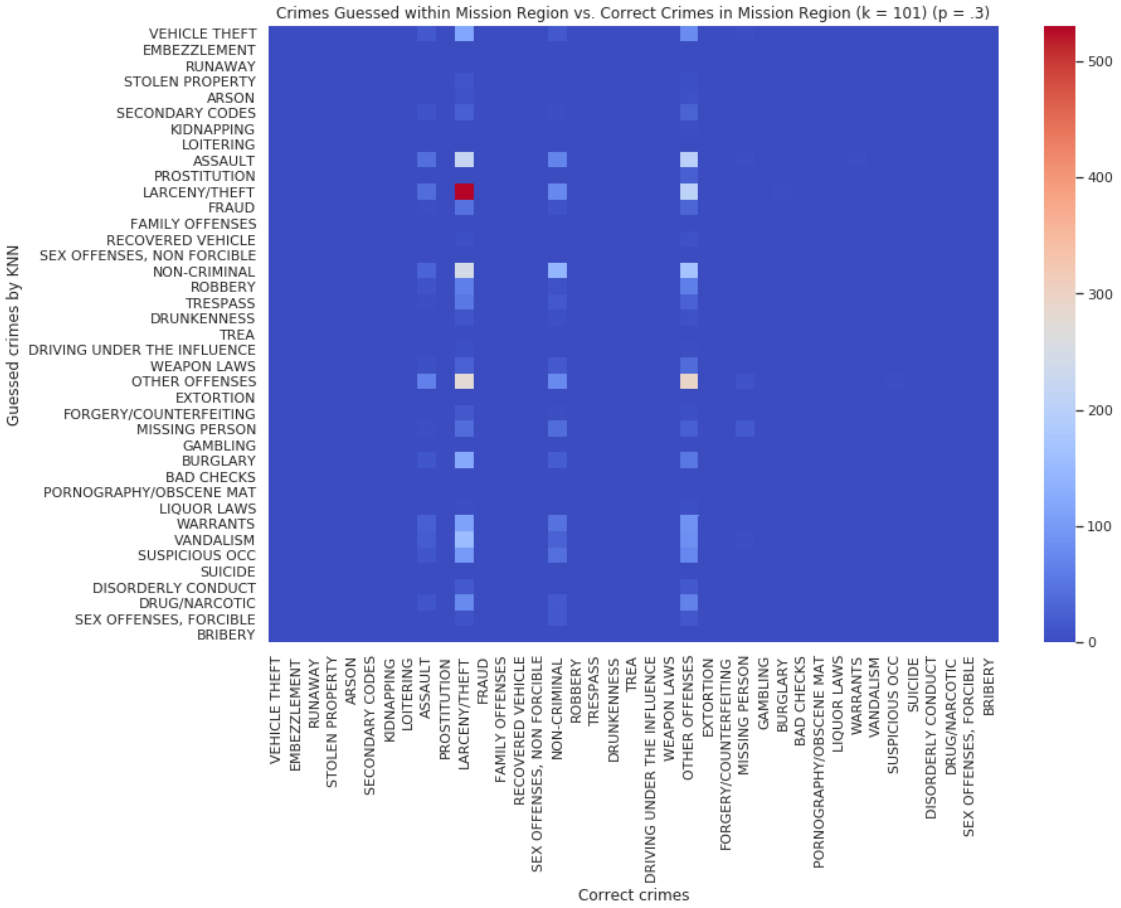
In [80]:
```
    # Plot the heatmap using seaborn
    sns.set(rc={'figure.figsize':(13,9)})
    ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklal
    ax.set(title='Crimes Guessed within Mission Region vs. Correct Crimes in Mission Regio
           xlabel='Correct crimes',
           ylabel='Guessed crimes by KNN')
    plt.show()
    print()
```



In [81]:
```
train_mission2, predict_mission2 = split_sets(mission_district, .3)
mission_test2 = KNN(101)
mission_test2.train(train_mission2)
subset_mission = predict_mission2[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_mission = [(mission_test2.predict(tupl[0]), tupl[1]) for tupl in subset_missio
```

In [82]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

for prediction in sub_cf_mission:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
mission_learning_curve.append([101, sub_percentage])
#print(mission_learning_curve)
```

In [83]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
ax.set(title='Crimes Guessed within Mission Region vs. Correct Crimes in Mission Regi
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```

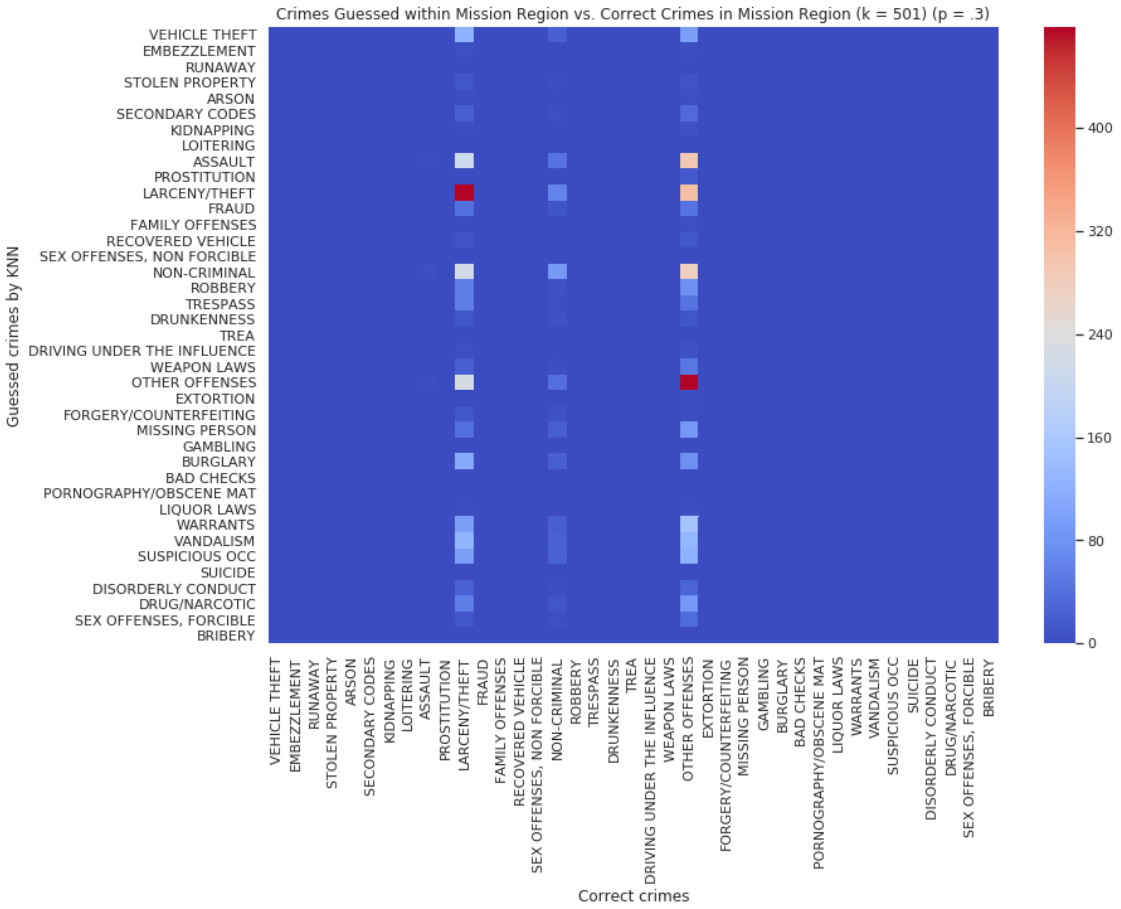



```
In [84]: train_mission3, predict_mission3 = split_sets(mission_district, .3)
         mission_test3 = KNN(501)
         mission_test3.train(train_mission3)
         subset_mission = predict_mission3[:5000]
         # Creates a list of tuples, (what the algorithm guessed, what is correct)
         sub_cf_mission = [(mission_test3.predict(tupl[0]), tupl[1]) for tupl in subset_missio
```

```
In [85]: # confusion_matrix = [([0,0], district) for district in parse_districts]
         # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
         accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

         for prediction in sub_cf_mission:
             accum_category_nums[prediction[1]][prediction[0]] += 1
         confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
         #print(confusion_matrix)
         sub_percentage = 0
```

```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        mission_learning_curve.append([501, sub_percentage])
        #print(mission_learning_curve)
```

In [86]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
        ax.set(title='Crimes Guessed within Mission Region vs. Correct Crimes in Mission Regio
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```

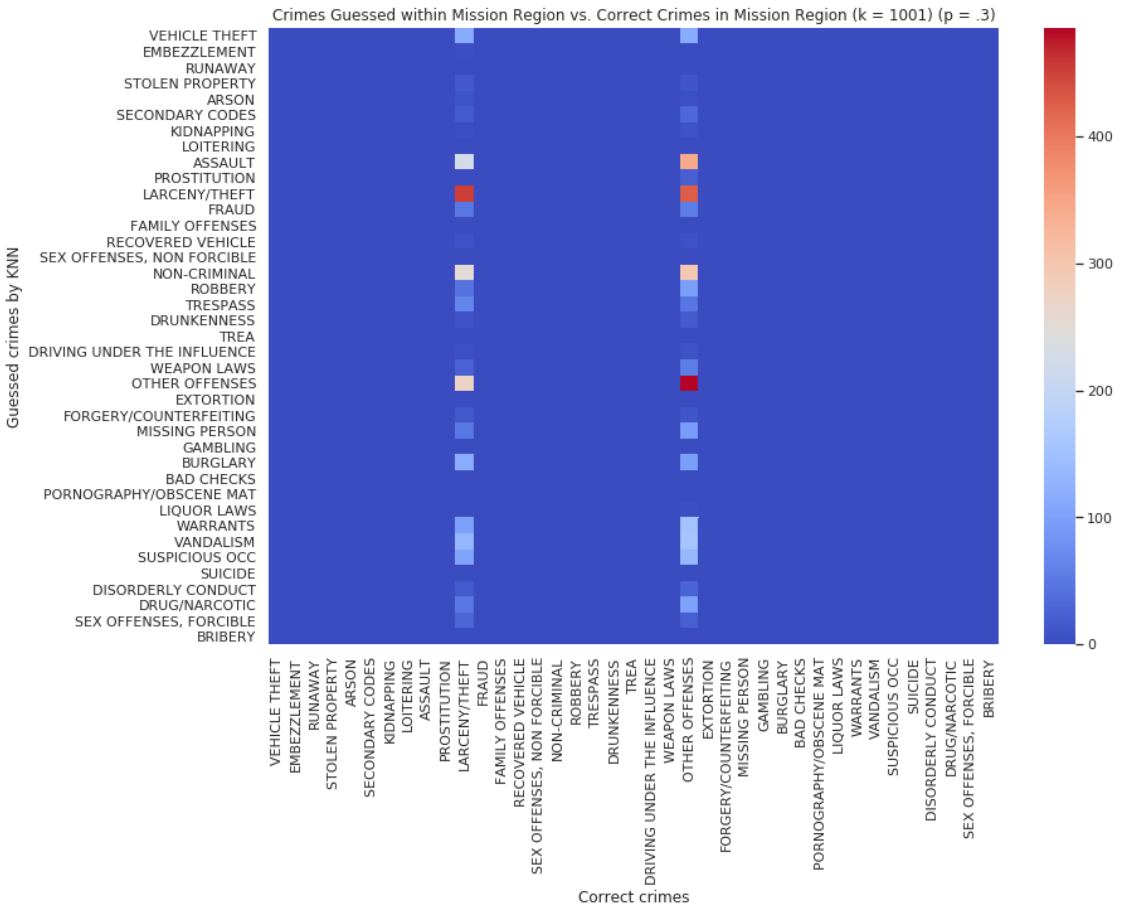


In [87]:
```
        train_mission4, predict_mission4 = split_sets(mission_district, .3)
        mission_test4 = KNN(1001)
```

```
mission_test4.train(train_mission4)
subset_mission = predict_mission4[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_mission = [(mission_test4.predict(tupl[0]), tupl[1]) for tupl in subset_missio
```

In [88]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

for prediction in sub_cf_mission:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
mission_learning_curve.append([1001, sub_percentage])
#print(mission_learning_curve)
```

In [89]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
ax.set(title='Crimes Guessed within Mission Region vs. Correct Crimes in Mission Regi
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```

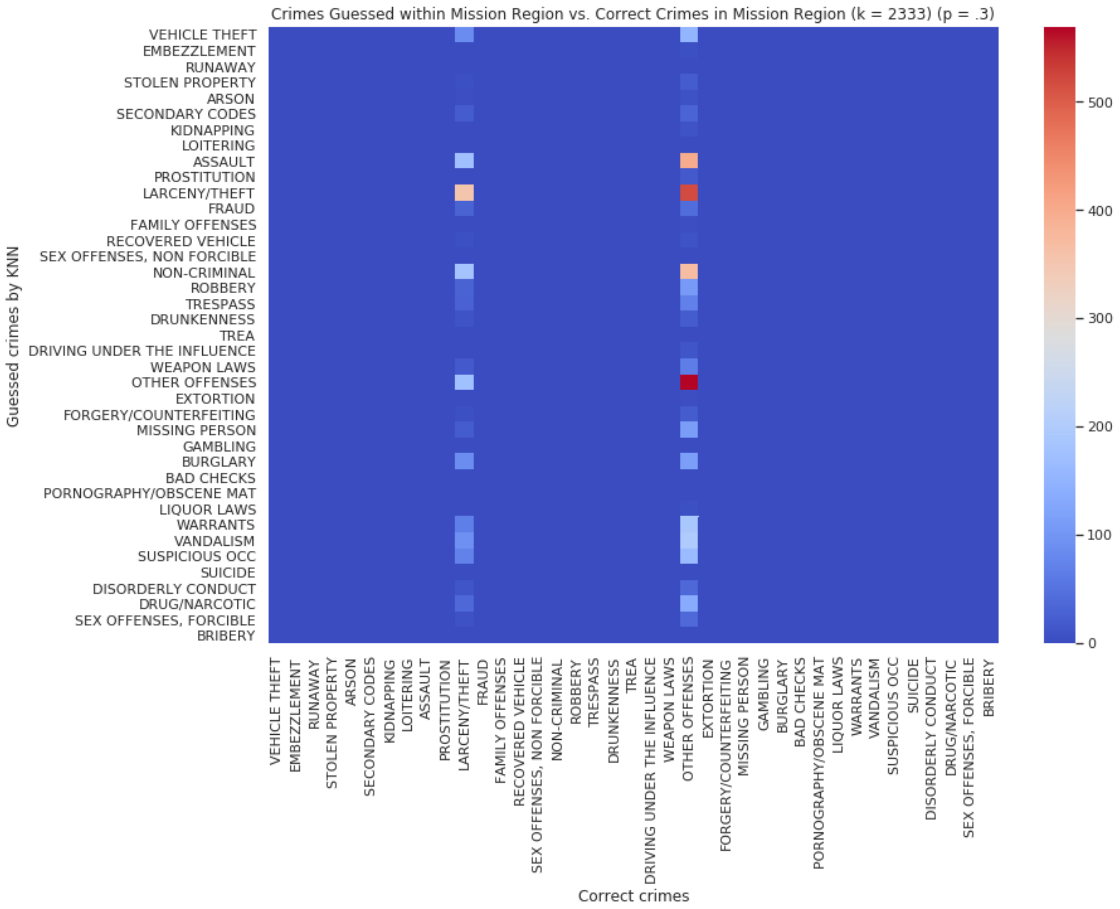



```
In [90]: train_mission5, predict_mission5 = split_sets(mission_district, .3)
         mission_test5 = KNN(2333)
         mission_test5.train(train_mission5)
         subset_mission = predict_mission5[:5000]
         # Creates a list of tuples, (what the algorithm guessed, what is correct)
         sub_cf_mission = [(mission_test5.predict(tupl[0]), tupl[1]) for tupl in subset_missio
```

```
In [91]: # confusion_matrix = [([0,0], district) for district in parse_districts]
         # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
         accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

         for prediction in sub_cf_mission:
             accum_category_nums[prediction[1]][prediction[0]] += 1
         confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
         #print(confusion_matrix)
         sub_percentage = 0
```

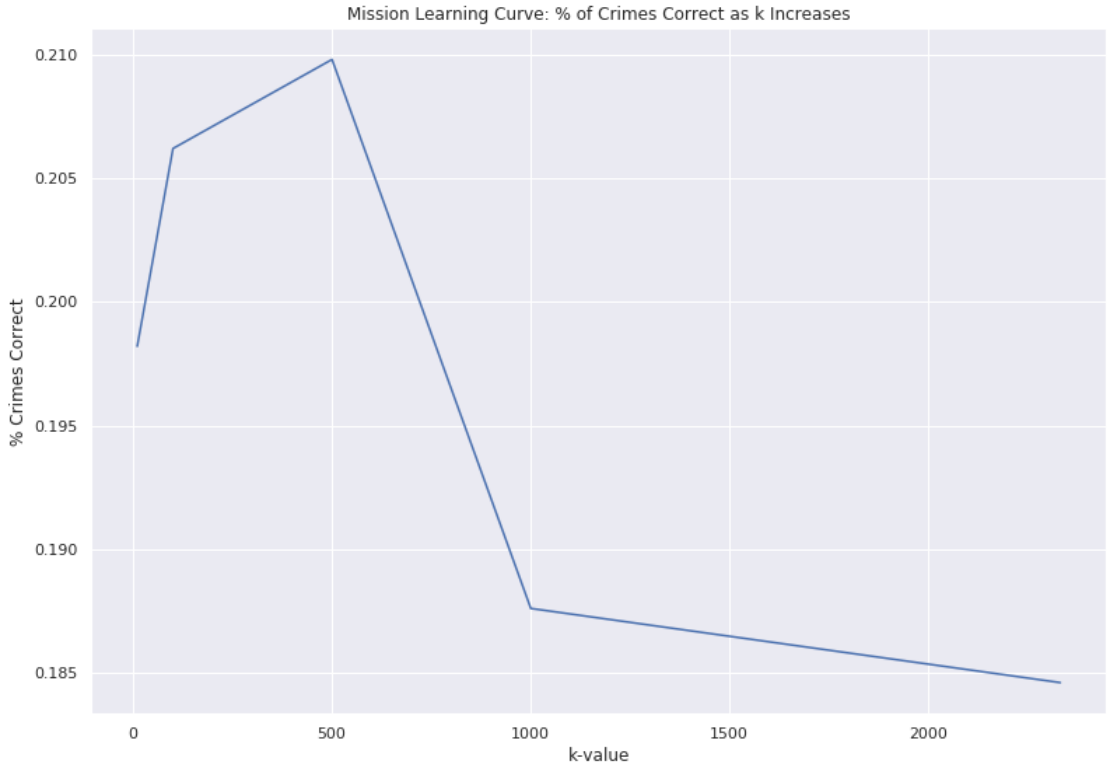
```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        mission_learning_curve.append([2333, sub_percentage])
        #print(mission_learning_curve)
```

In [92]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
        ax.set(title='Crimes Guessed within Mission Region vs. Correct Crimes in Mission Regio
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```



In [93]:
```
        # Printing the learning curve for mission
        # print(mission_learning_curve)
```

```
plt.plot([i[0] for i in mission_learning_curve], [i[1] for i in mission_learning_curve
plt.title('Mission Learning Curve: % of Crimes Correct as k Increases')
plt.xlabel('k-value')
plt.ylabel('% Crimes Correct')
plt.show()
```



15.2.6 Central:

```
In [94]: central_district = crime_data[crime_data['PdDistrict'] == 'CENTRAL'][['Category', 'X'
         central_district.head()
```

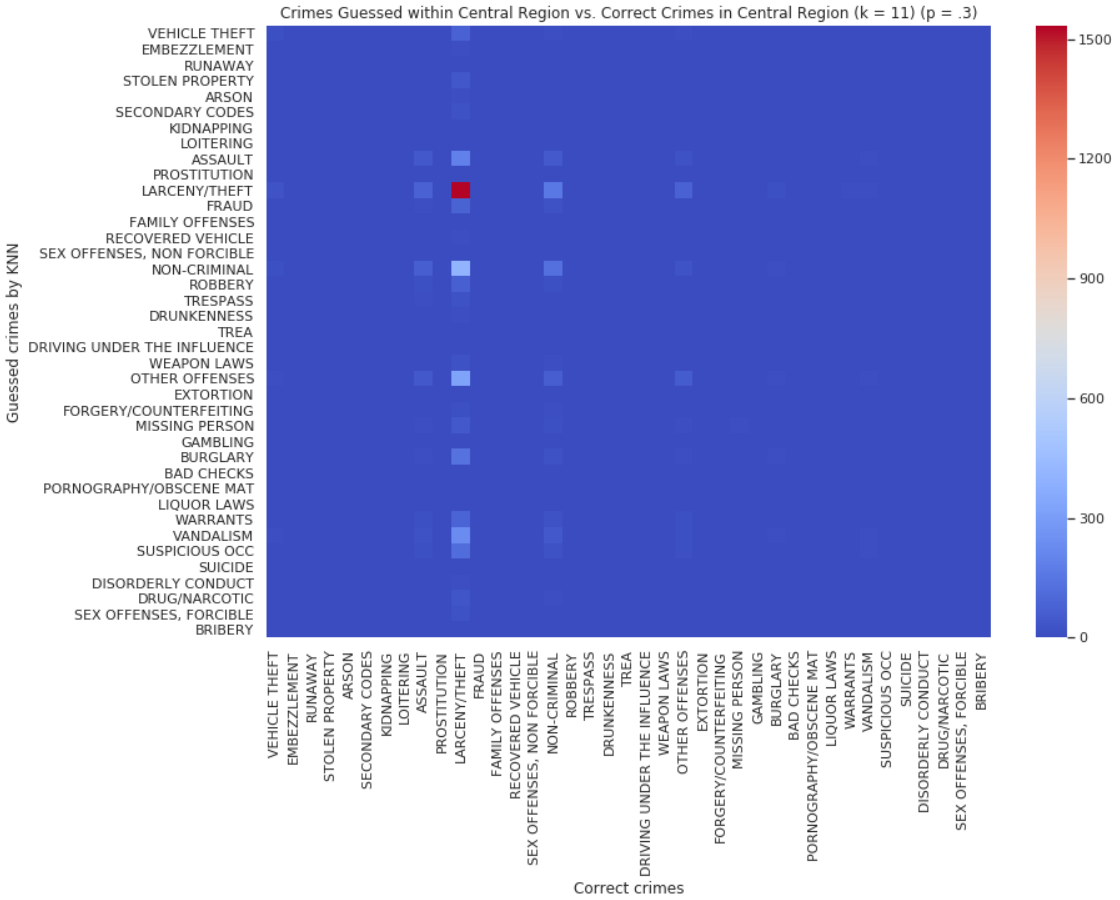
Out[94]:

	Category	X	Y
13	BURGLARY	-122.400909	37.791643
29	VEHICLE THEFT	-122.410883	37.787921
30	VEHICLE THEFT	-122.401433	37.800457
57	ASSAULT	-122.403916	37.790539
65	NON-CRIMINAL	-122.397374	37.795334

```
In [95]: train_central1, predict_central1 = split_sets(central_district, .3)
         central_test1 = KNN(11)
         central_test1.train(train_central1)
         subset_central = predict_central1[:5000]
         # Creates a list of tuples, (what the algorithm guessed, what is correct)
         sub_cf_central = [(central_test1.predict(tupl[0]), tupl[1]) for tupl in subset_central
```

In [96]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cri

# INITIALIZE CONFUSION MATRIX FOR CENTRAL HERE!!!!!!!!!!!!!!!!!!!!!!!!
central_learning_curve = []

for prediction in sub_cf_central:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
central_learning_curve.append([11, sub_percentage])
#print(central_learning_curve)
```

In [97]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xticklab
ax.set(title='Crimes Guessed within Central Region vs. Correct Crimes in Central Regio
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```

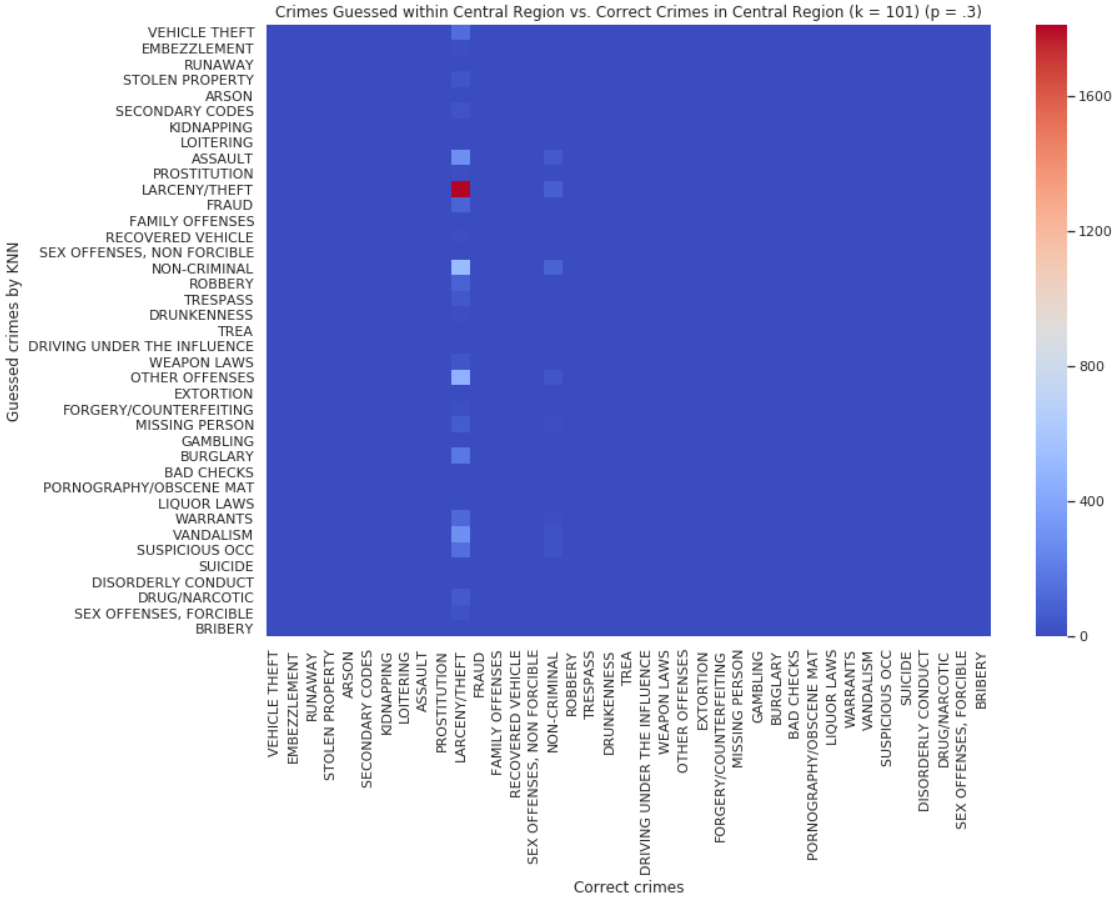


In [98]:
```
train_central2, predict_central2 = split_sets(central_district, .3)
central_test2 = KNN(101)
central_test2.train(train_central2)
subset_central = predict_central2[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_central = [(central_test2.predict(tupl[0]), tupl[1]) for tupl in subset_central
```

In [99]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cri

for prediction in sub_cf_central:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_nu
#print(confusion_matrix)
sub_percentage = 0
```

```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        central_learning_curve.append([101, sub_percentage])
        #print(central_learning_curve)
```

In [100]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
        ax.set(title='Crimes Guessed within Central Region vs. Correct Crimes in Central Reg
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```

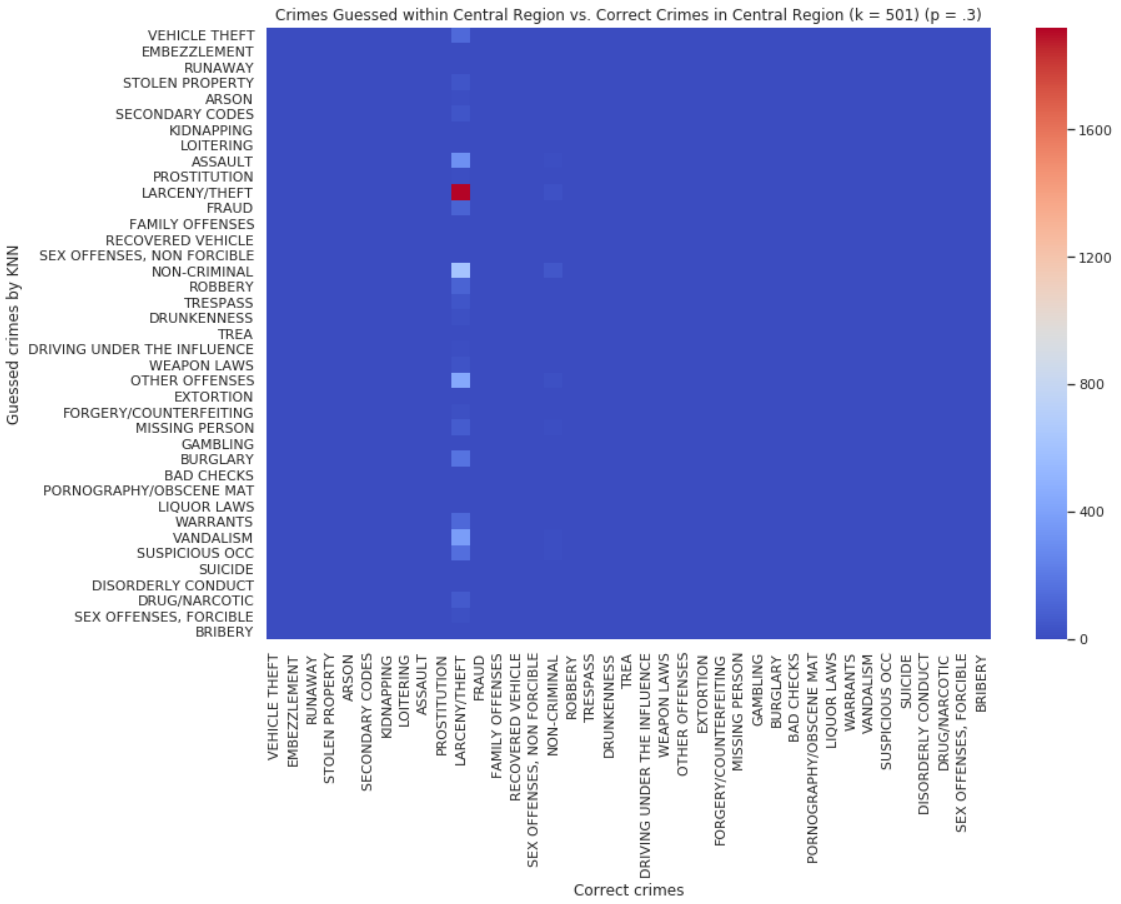


In [101]:
```
        train_central3, predict_central3 = split_sets(central_district, .3)
        central_test3 = KNN(501)
```

```
central_test3.train(train_central3)
subset_central = predict_central3[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_central = [(central_test3.predict(tupl[0]), tupl[1]) for tupl in subset_centra
```

In [102]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_norther
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

for prediction in sub_cf_central:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_n
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
central_learning_curve.append([501, sub_percentage])
#print(central_learning_curve)
```

In [103]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
ax.set(title='Crimes Guessed within Central Region vs. Correct Crimes in Central Regi
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```

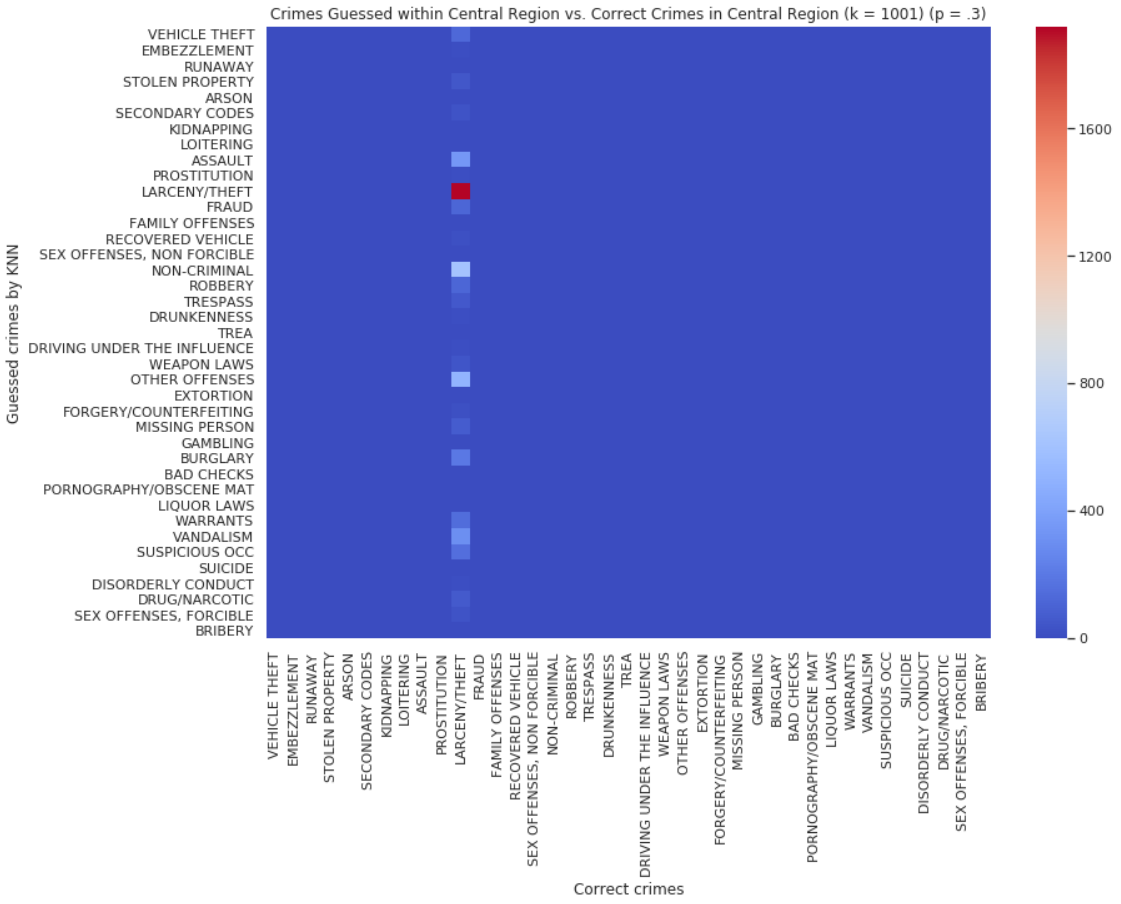


In [104]:
```
train_central4, predict_central4 = split_sets(central_district, .3)
central_test4 = KNN(1001)
central_test4.train(train_central4)
subset_central = predict_central4[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_central = [(central_test4.predict(tupl[0]), tupl[1]) for tupl in subset_centra
```

In [105]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_norther
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_c

for prediction in sub_cf_central:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_
#print(confusion_matrix)
sub_percentage = 0
```

```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        central_learning_curve.append([1001, sub_percentage])
        #print(central_learning_curve)
```

In [106]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
        ax.set(title='Crimes Guessed within Central Region vs. Correct Crimes in Central Reg
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```

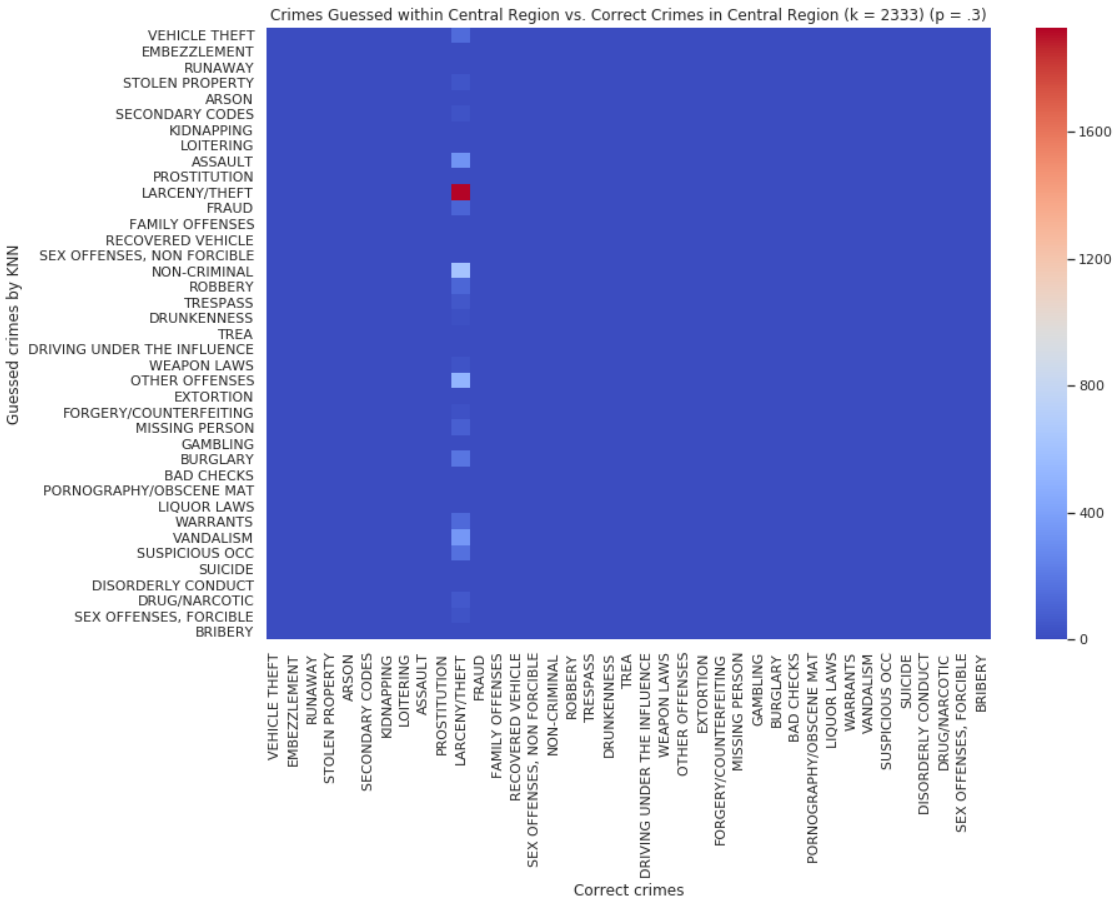


In [107]:
```
        train_central5, predict_central5 = split_sets(central_district, .3)
        central_test5 = KNN(2333)
```

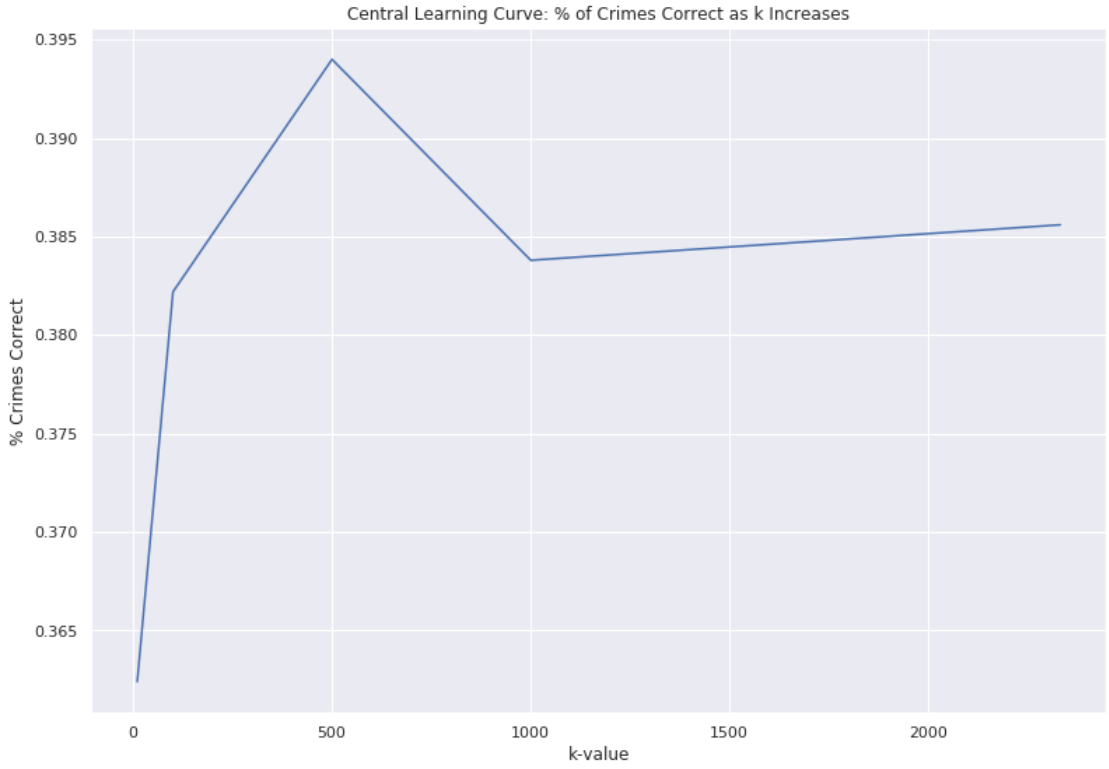
```
central_test5.train(train_central5)
subset_central = predict_central5[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_central = [(central_test5.predict(tupl[0]), tupl[1]) for tupl in subset_centra
```

In [108]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_norther
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_c

for prediction in sub_cf_central:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
central_learning_curve.append([2333, sub_percentage])
#print(central_learning_curve)
```

In [109]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
ax.set(title='Crimes Guessed within Central Region vs. Correct Crimes in Central Reg
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```



In [110]:
```
# Printing the learning curve for central
# print(central_learning_curve)
plt.plot([i[0] for i in central_learning_curve], [i[1] for i in central_learning_curv
plt.title('Central Learning Curve: % of Crimes Correct as k Increases')
plt.xlabel('k-value')
plt.ylabel('% Crimes Correct')
plt.show()
```



15.2.7 Ingleside:

```
In [111]: ingleside_district = crime_data[crime_data['PdDistrict'] == 'INGLESIDE'][['Category'
          ingleside_district.head()
```

Out[111]:

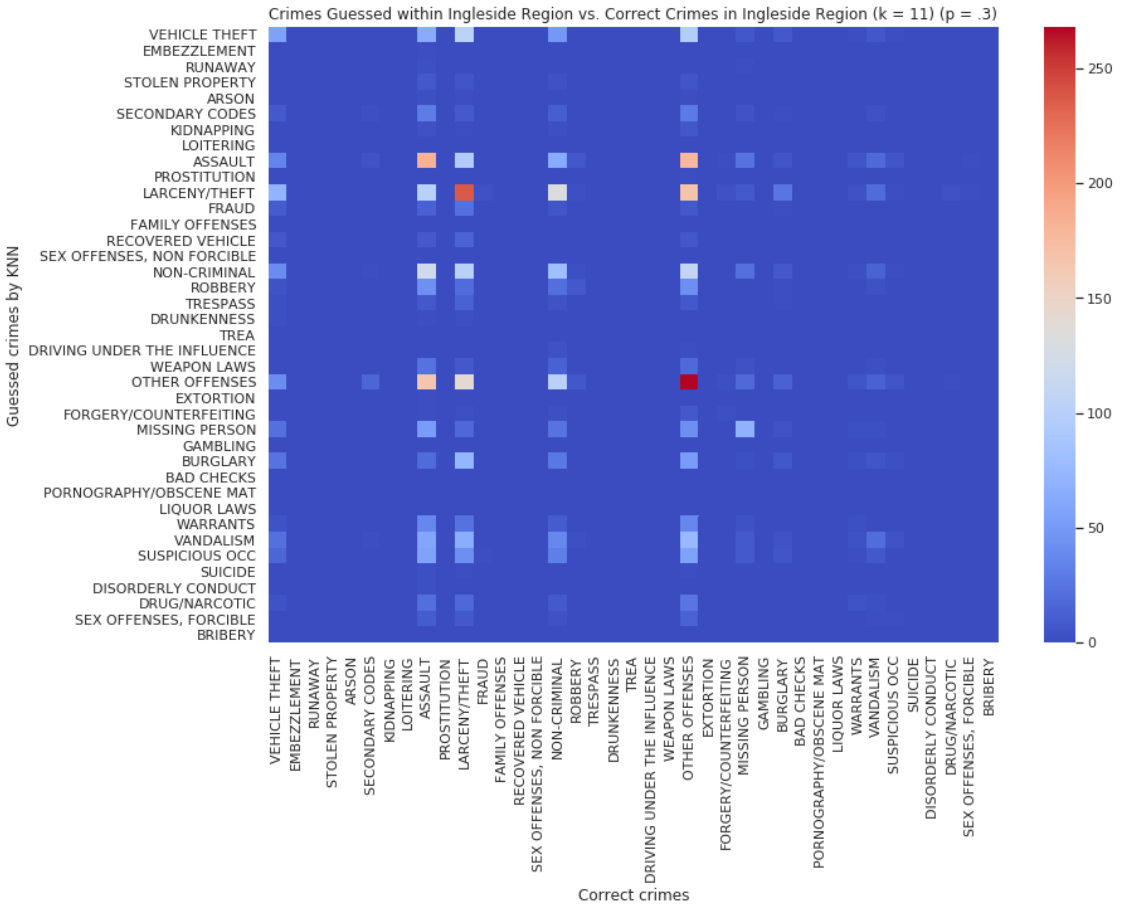
	Category	X	Y
12	ASSAULT	-122.432326	37.729271
17	ASSAULT	-122.439910	37.715765
28	NON-CRIMINAL	-122.434609	37.709201
31	LARCENY/THEFT	-122.421128	37.743555
35	NON-CRIMINAL	-122.444685	37.744193

```
In [112]: train_ingleside1, predict_ingleside1 = split_sets(ingleside_district, .3)
          ingleside_test1 = KNN(11)
          ingleside_test1.train(train_ingleside1)
          subset_ingleside = predict_ingleside1[:5000]
          # Creates a list of tuples, (what the algorithm guessed, what is correct)
          sub_cf_ingleside = [(ingleside_test1.predict(tupl[0]), tupl[1]) for tupl in subset_in
```

```
In [113]: # confusion_matrix = [([0,0], district) for district in parse_districts]
          # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
          accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr
```

```
# INITIALIZE CONFUSION MATRIX FOR INGLESIDE HERE!!!!!!!!!!!!!!!!!!!!!!!!
ingleside_learning_curve = []

for prediction in sub_cf_ingleside:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
ingleside_learning_curve.append([11, sub_percentage])
#print(ingleside_learning_curve)
```

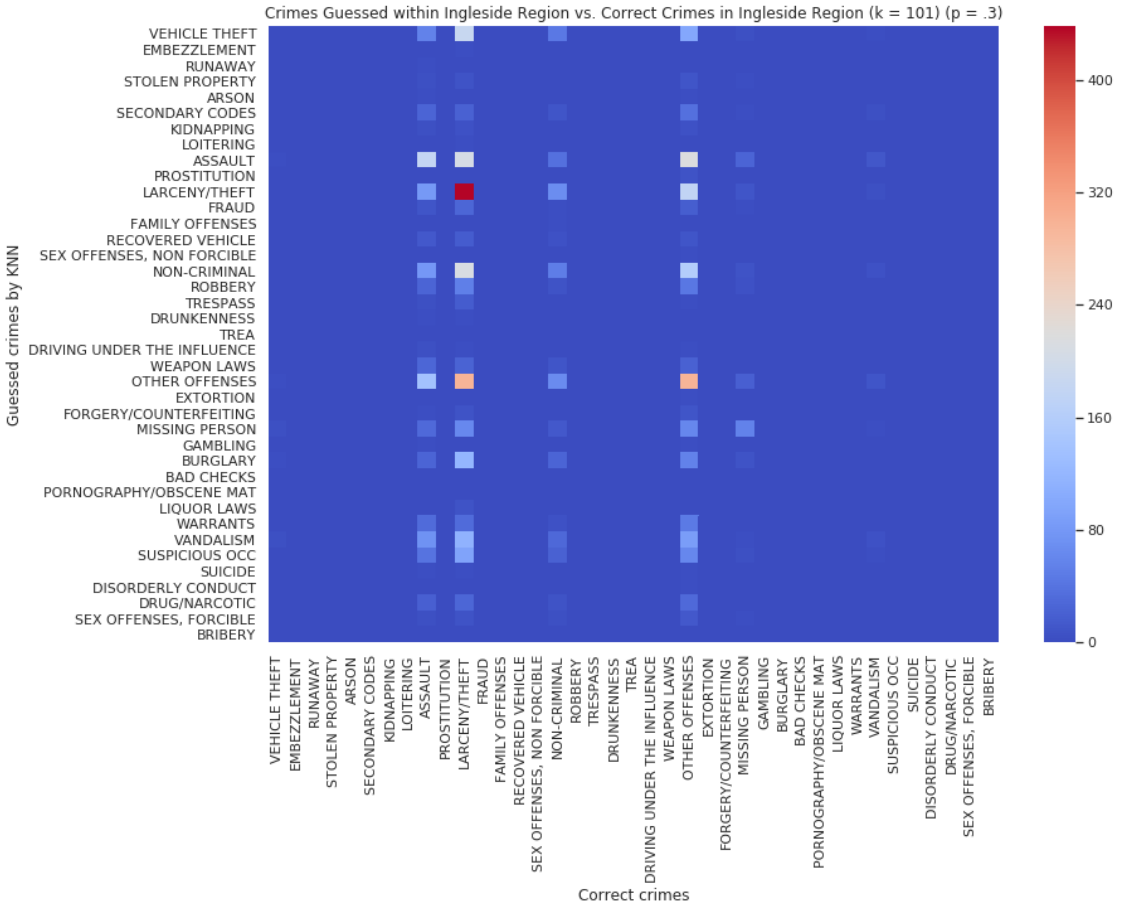
In [114]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
ax.set(title='Crimes Guessed within Ingleside Region vs. Correct Crimes in Ingleside
       xlabel='Correct crimes',
       ylabel='Guessed crimes by KNN')
plt.show()
print()
```



In [115]:
```
train_ingleside2, predict_ingleside2 = split_sets(ingleside_district, .3)
ingleside_test2 = KNN(101)
ingleside_test2.train(train_ingleside2)
subset_ingleside = predict_ingleside2[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_ingleside = [(ingleside_test2.predict(tupl[0]), tupl[1]) for tupl in subset_in
```

In [116]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

for prediction in sub_cf_ingleside:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_n
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
ingleside_learning_curve.append([101, sub_percentage])
#print(ingleside_learning_curve)
```

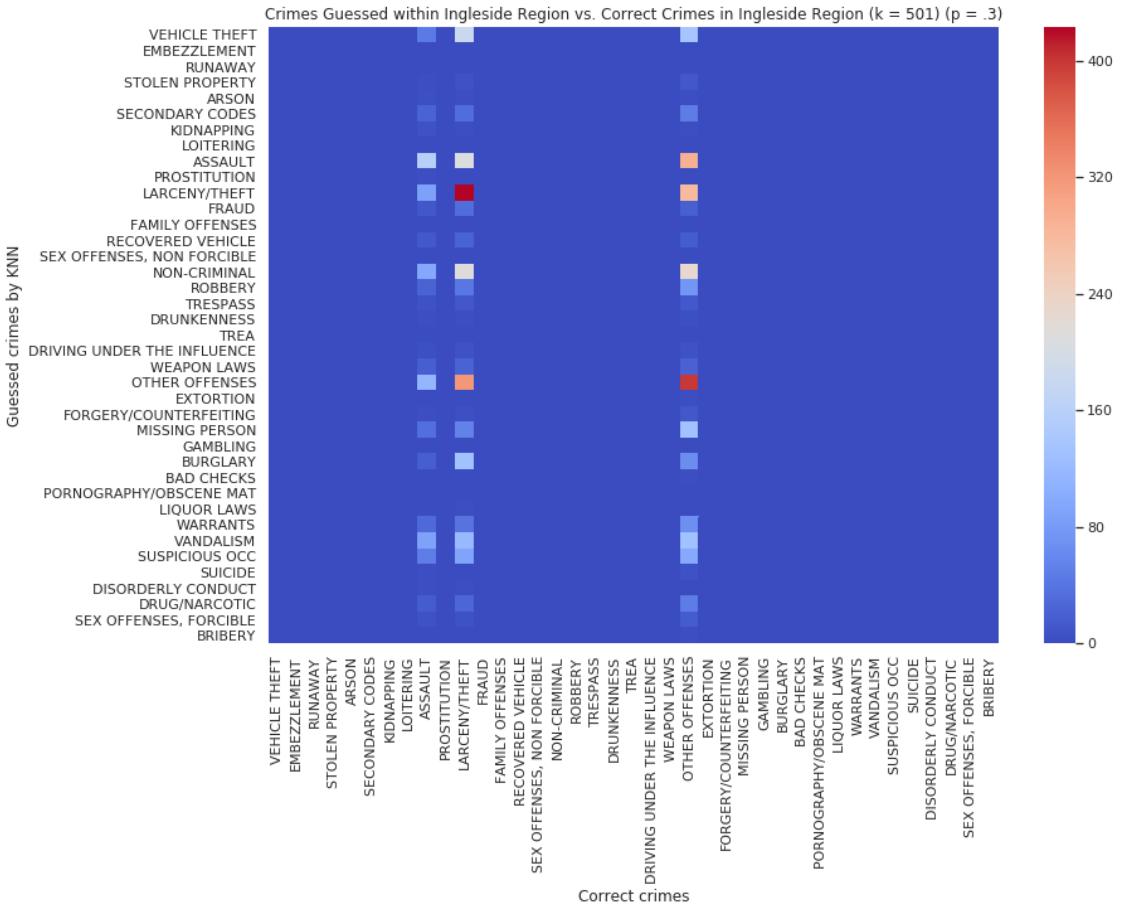
In [117]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
ax.set(title='Crimes Guessed within Ingleside Region vs. Correct Crimes in Ingleside
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```




```
In [118]: train_ingleside3, predict_ingleside3 = split_sets(ingleside_district, .3)
          ingleside_test3 = KNN(501)
          ingleside_test3.train(train_ingleside3)
          subset_ingleside = predict_ingleside3[:5000]
          # Creates a list of tuples, (what the algorithm guessed, what is correct)
          sub_cf_ingleside = [(ingleside_test3.predict(tupl[0]), tupl[1]) for tupl in subset_in

In [119]: # confusion_matrix = [([0,0], district) for district in parse_districts]
          # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
          accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

          for prediction in sub_cf_ingleside:
              accum_category_nums[prediction[1]][prediction[0]] += 1
          confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_n
          #print(confusion_matrix)
          sub_percentage = 0
```

```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        ingleside_learning_curve.append([501, sub_percentage])
        #print(ingleside_learning_curve)
```

In [120]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
        ax.set(title='Crimes Guessed within Ingleside Region vs. Correct Crimes in Ingleside
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```

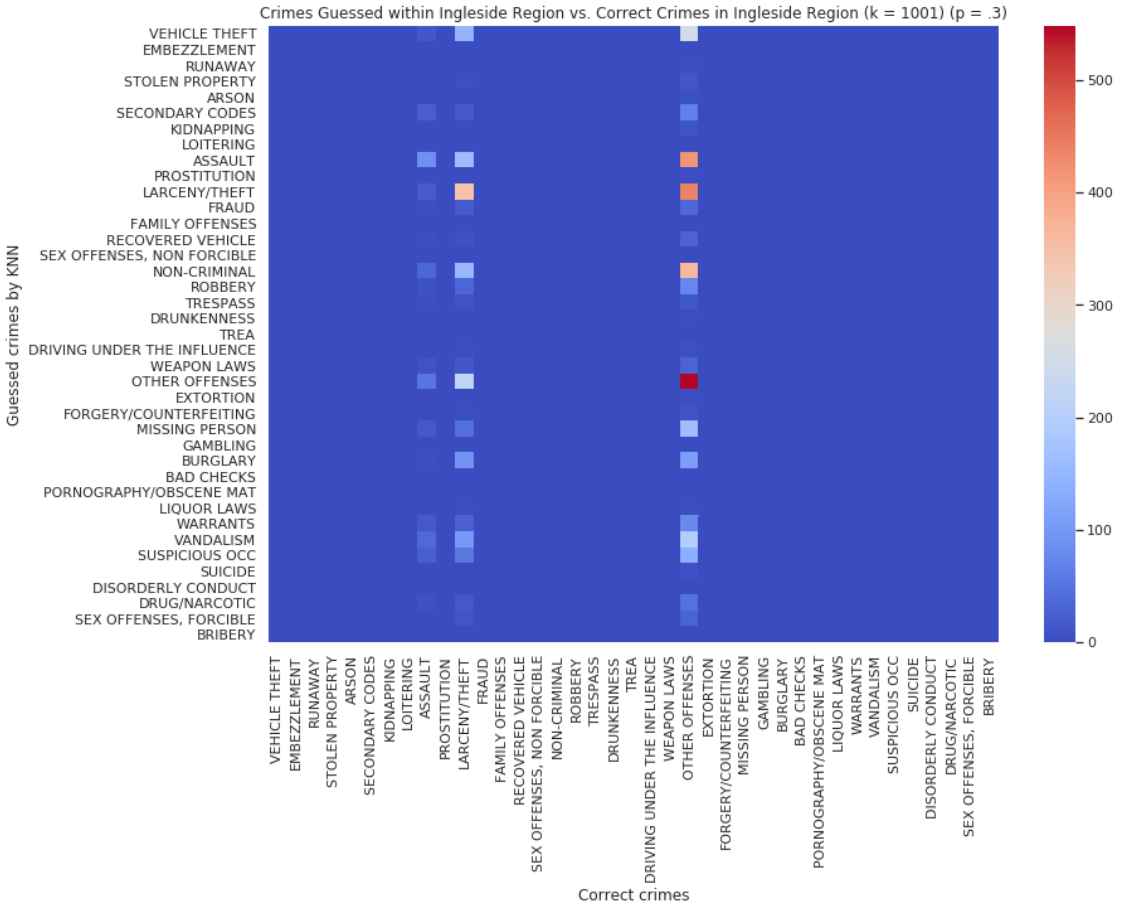


In [121]:
```
        train_ingleside4, predict_ingleside4 = split_sets(ingleside_district, .3)
        ingleside_test4 = KNN(1001)
```

```
ingleside_test4.train(train_ingleside4)
subset_ingleside = predict_ingleside4[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_ingleside = [(ingleside_test4.predict(tupl[0]), tupl[1]) for tupl in subset_in
```

In [122]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

for prediction in sub_cf_ingleside:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_n
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
ingleside_learning_curve.append([1001, sub_percentage])
#print(ingleside_learning_curve)
```

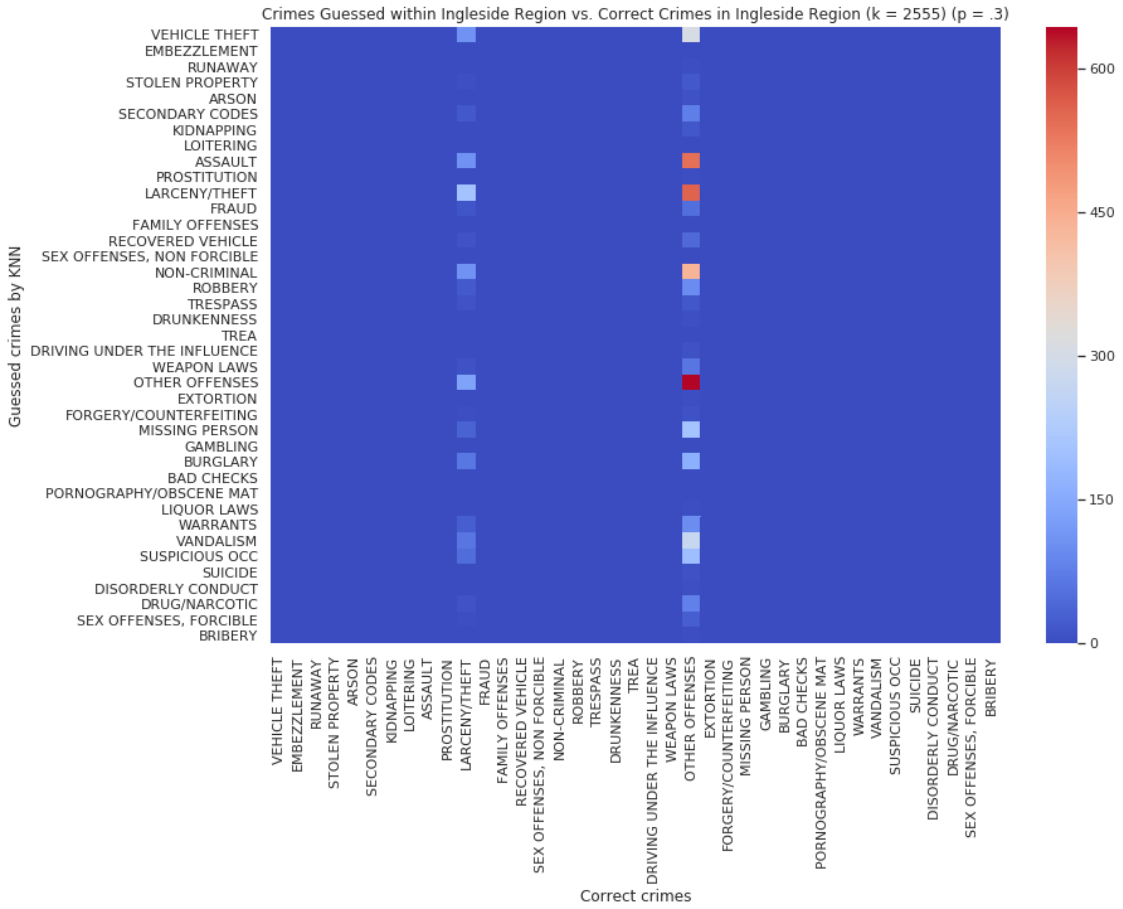
In [123]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
ax.set(title='Crimes Guessed within Ingleside Region vs. Correct Crimes in Ingleside
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```




```
In [124]: train_ingleside5, predict_ingleside5 = split_sets(ingleside_district, .3)
          ingleside_test5 = KNN(2555)
          ingleside_test5.train(train_ingleside5)
          subset_ingleside = predict_ingleside5[:5000]
          # Creates a list of tuples, (what the algorithm guessed, what is correct)
          sub_cf_ingleside = [(ingleside_test5.predict(tupl[0]), tupl[1]) for tupl in subset_in

In [125]: # confusion_matrix = [([0,0], district) for district in parse_districts]
          # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
          accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

          for prediction in sub_cf_ingleside:
              accum_category_nums[prediction[1]][prediction[0]] += 1
          confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_n
          #print(confusion_matrix)
          sub_percentage = 0
```

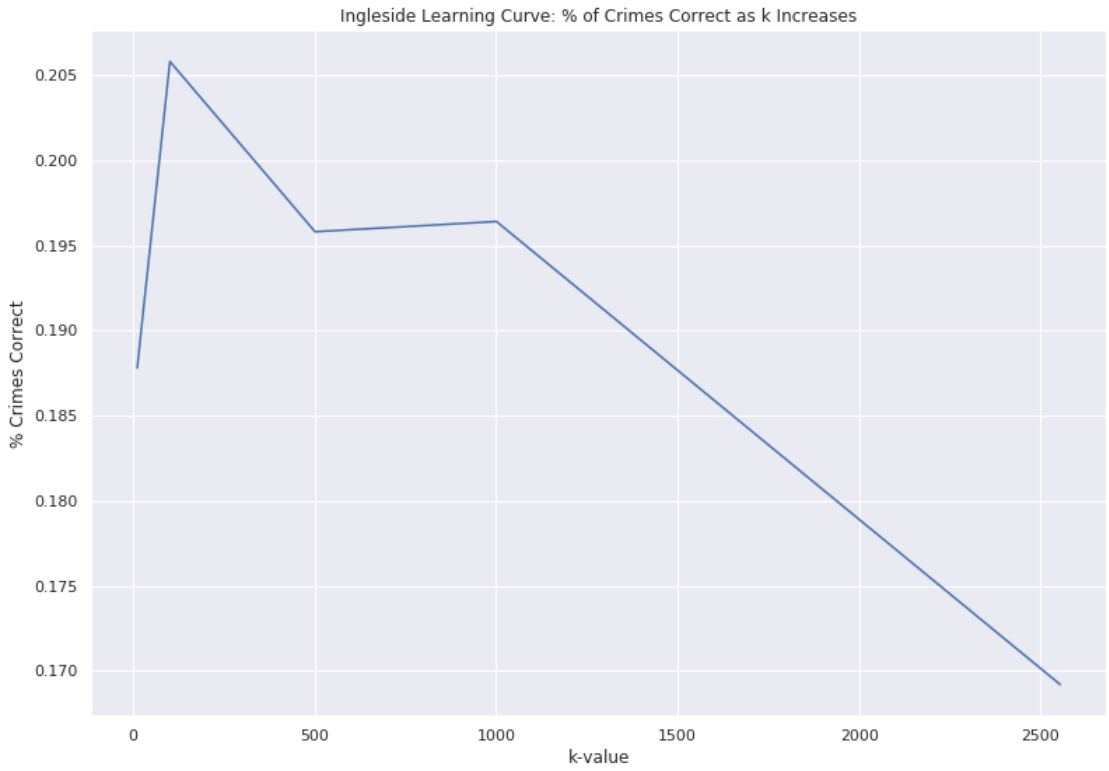
```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        ingleside_learning_curve.append([2555, sub_percentage])
        #print(ingleside_learning_curve)
```

In [126]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
        ax.set(title='Crimes Guessed within Ingleside Region vs. Correct Crimes in Ingleside
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```



In [127]:
```
        # Printing the learning curve for ingleside
        # print(central_learning_curve)
```

```
plt.plot([i[0] for i in ingleside_learning_curve], [i[1] for i in ingleside_learning_
plt.title('Ingleside Learning Curve: % of Crimes Correct as k Increases')
plt.xlabel('k-value')
plt.ylabel('% Crimes Correct')
plt.show()
```



15.2.8 Park:

In [128]:
```
park_district = crime_data[crime_data['PdDistrict'] == 'PARK'][['Category', 'X', 'Y']
park_district.head()
```

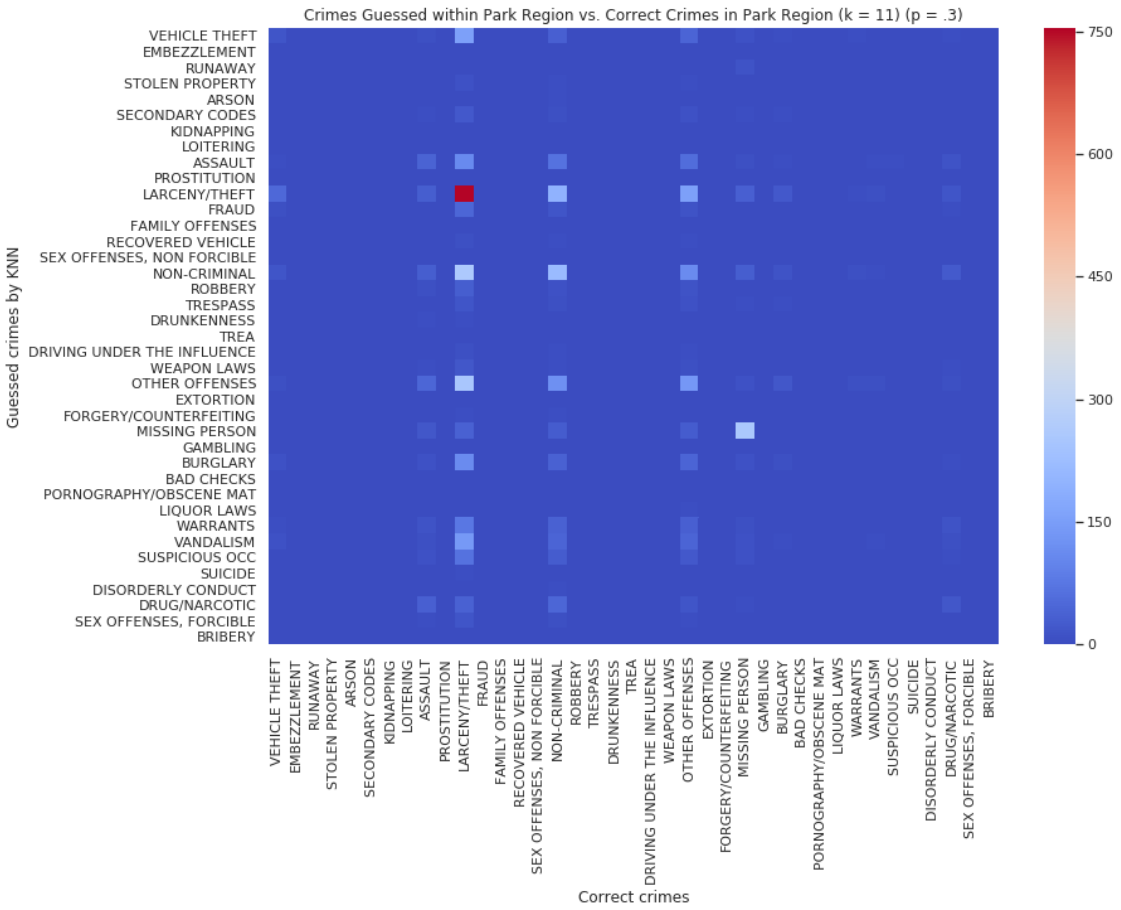
Out[128]:

	Category	X	Y
60	LARCENY/THEFT	-122.437800	37.774991
63	WARRANTS	-122.446613	37.782246
69	NON-CRIMINAL	-122.436614	37.773292
96	BURGLARY	-122.449752	37.764430
97	LARCENY/THEFT	-122.449752	37.764430

In [129]:
```
train_park1, predict_park1 = split_sets(park_district, .3)
park_test1 = KNN(11)
park_test1.train(train_park1)
subset_park = predict_park1[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_park = [(park_test1.predict(tupl[0]), tupl[1]) for tupl in subset_park]
```

In [130]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

# INITIALIZE CONFUSION MATRIX FOR PARK HERE!!!!!!!!!!!!!!!!!!!!!!!!
park_learning_curve = []

for prediction in sub_cf_park:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_n
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
park_learning_curve.append([11, sub_percentage])
#print(ingleside_learning_curve)
```

In [131]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
ax.set(title='Crimes Guessed within Park Region vs. Correct Crimes in Park Region (k
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```

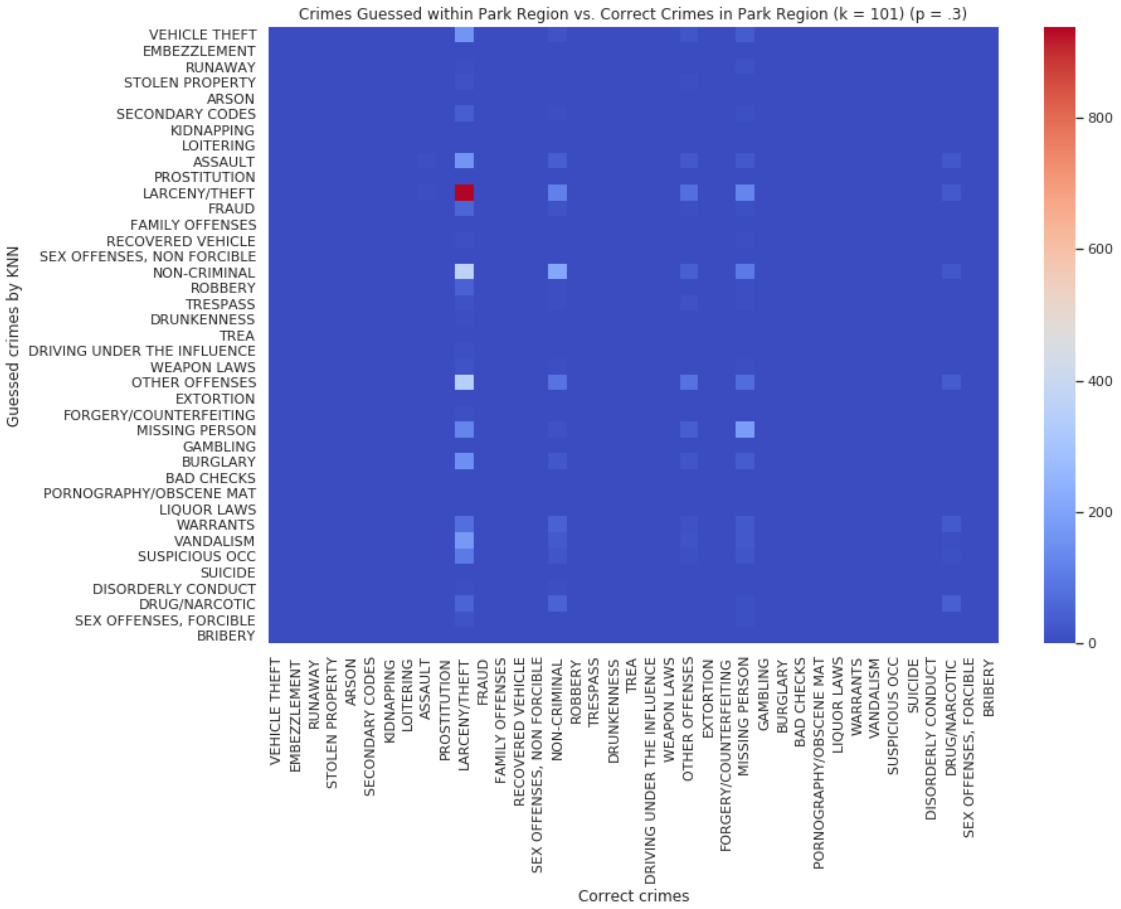


In [132]:
```
train_park2, predict_park2 = split_sets(park_district, .3)
park_test2 = KNN(101)
park_test2.train(train_park2)
subset_park = predict_park2[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_park = [(park_test2.predict(tupl[0]), tupl[1]) for tupl in subset_park]
```

In [133]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

for prediction in sub_cf_park:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_n
#print(confusion_matrix)
sub_percentage = 0
```

```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        park_learning_curve.append([101, sub_percentage])
        #print(ingleside_learning_curve)
```

In [134]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
        ax.set(title='Crimes Guessed within Park Region vs. Correct Crimes in Park Region (k
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```

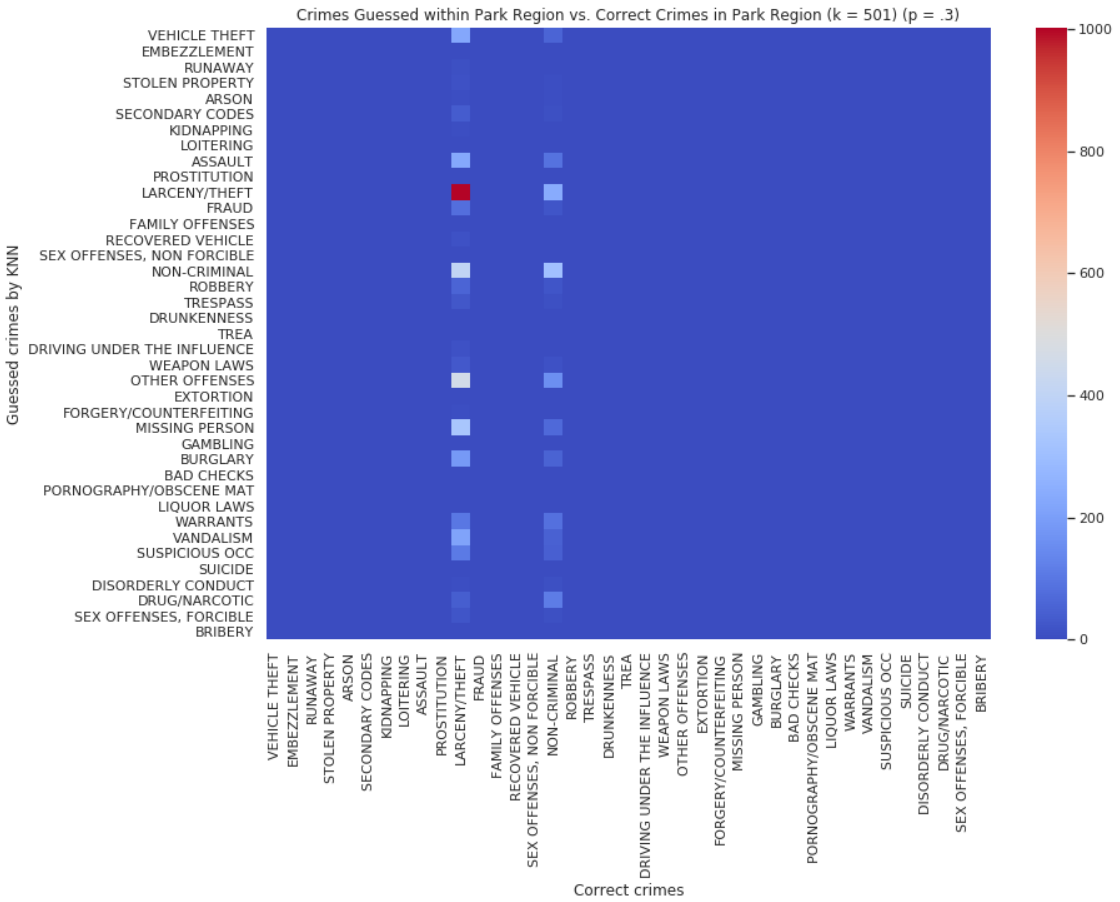


In [135]:
```
        train_park3, predict_park3 = split_sets(park_district, .3)
        park_test3 = KNN(501)
```

```
park_test3.train(train_park3)
subset_park = predict_park3[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_park = [(park_test3.predict(tupl[0]), tupl[1]) for tupl in subset_park]
```

In [136]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

for prediction in sub_cf_park:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_n
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
park_learning_curve.append([501, sub_percentage])
#print(ingleside_learning_curve)
```

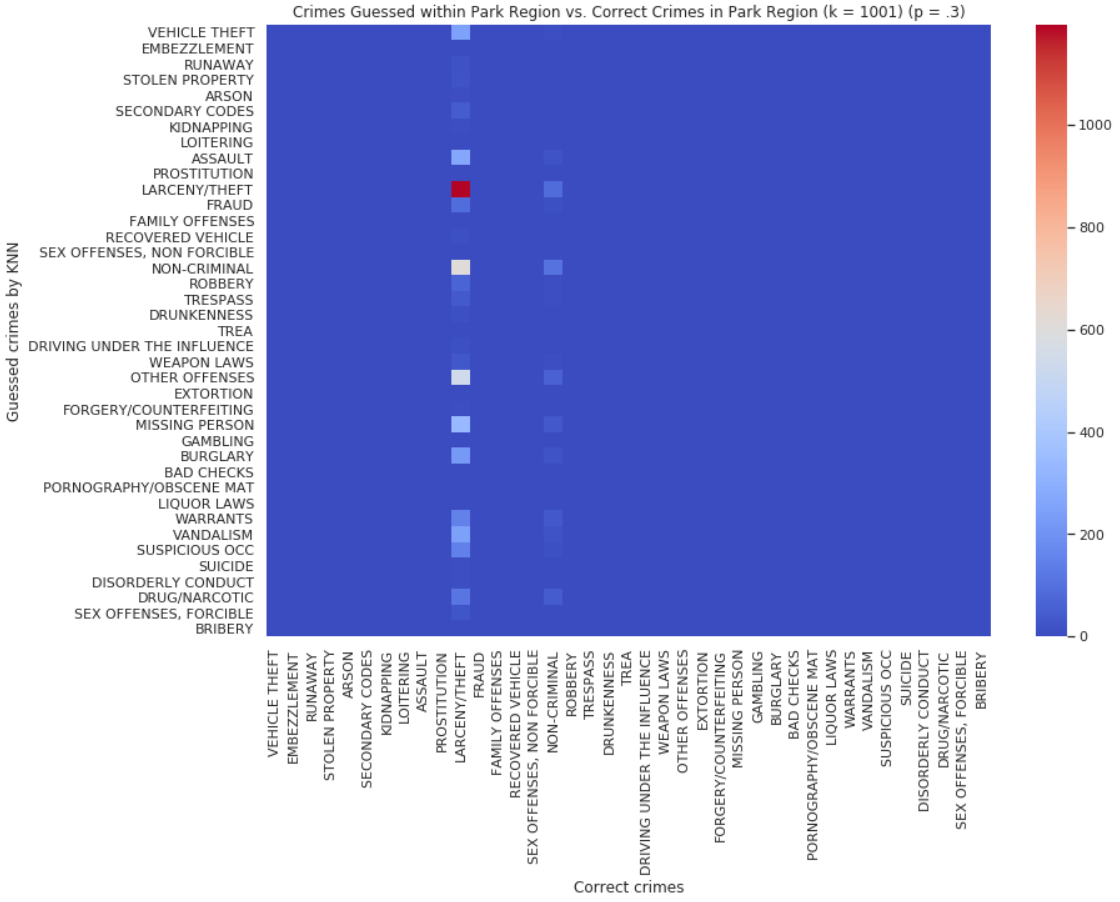
In [137]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
ax.set(title='Crimes Guessed within Park Region vs. Correct Crimes in Park Region (k
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```




```
In [138]: train_park4, predict_park4 = split_sets(park_district, .3)
          park_test4 = KNN(1001)
          park_test4.train(train_park4)
          subset_park = predict_park4[:5000]
          # Creates a list of tuples, (what the algorithm guessed, what is correct)
          sub_cf_park = [(park_test4.predict(tupl[0]), tupl[1]) for tupl in subset_park]

In [139]: # confusion_matrix = [([0,0], district) for district in parse_districts]
          # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
          accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

          for prediction in sub_cf_park:
              accum_category_nums[prediction[1]][prediction[0]] += 1
          confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_n
          #print(confusion_matrix)
          sub_percentage = 0
```

```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        park_learning_curve.append([1001, sub_percentage])
        #print(ingleside_learning_curve)
```

In [140]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
        ax.set(title='Crimes Guessed within Park Region vs. Correct Crimes in Park Region (k
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```

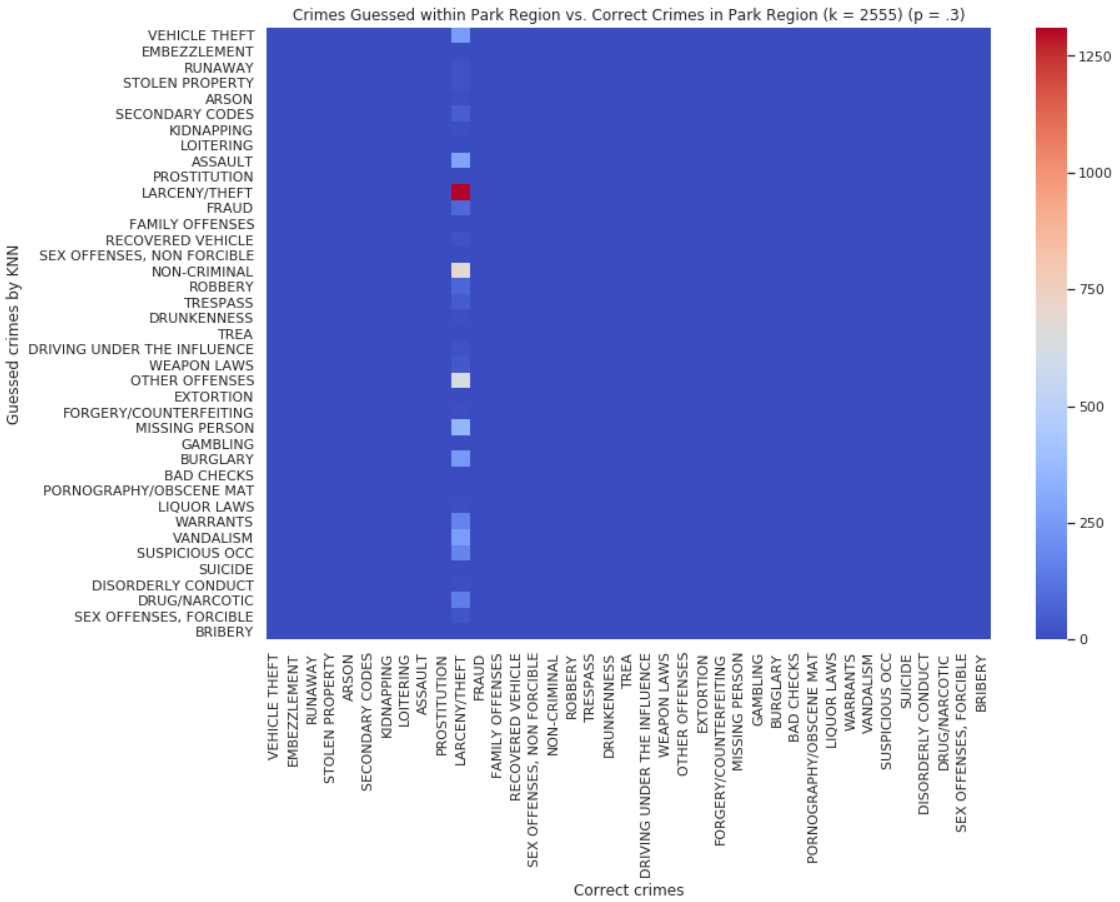


In [141]:
```
        train_park5, predict_park5 = split_sets(park_district, .3)
        park_test5 = KNN(2555)
```

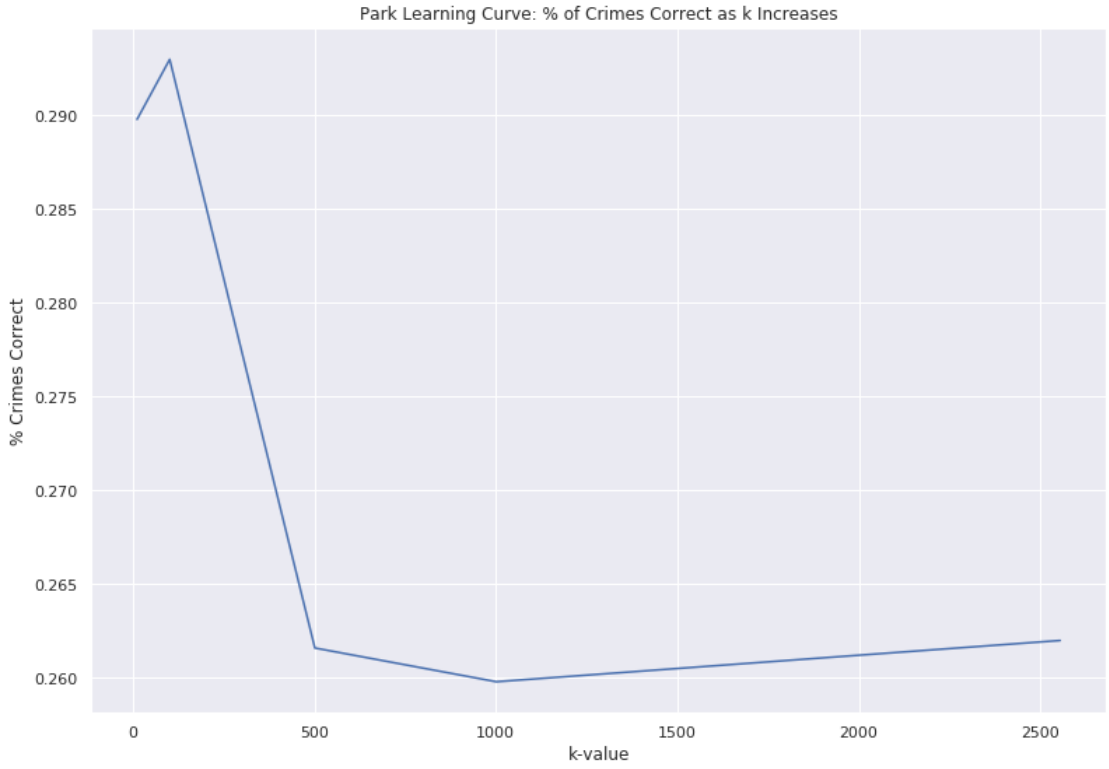
```
park_test5.train(train_park5)
subset_park = predict_park5[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_park = [(park_test5.predict(tupl[0]), tupl[1]) for tupl in subset_park]
```

In [142]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

for prediction in sub_cf_park:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_n
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
park_learning_curve.append([2555, sub_percentage])
#print(ingleside_learning_curve)
```

In [143]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
ax.set(title='Crimes Guessed within Park Region vs. Correct Crimes in Park Region (k
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```




```
In [144]: # Printing the learning curve for park
          # print(central_learning_curve)
          plt.plot([i[0] for i in park_learning_curve], [i[1] for i in park_learning_curve])
          plt.title('Park Learning Curve: % of Crimes Correct as k Increases')
          plt.xlabel('k-value')
          plt.ylabel('% Crimes Correct')
          plt.show()
```



15.2.9 Taraval:

```
In [145]: taraval_district = crime_data[crime_data['PdDistrict'] == 'TARAVAL'][['Category', 'X
          taraval_district.head()
```

Out[145]:

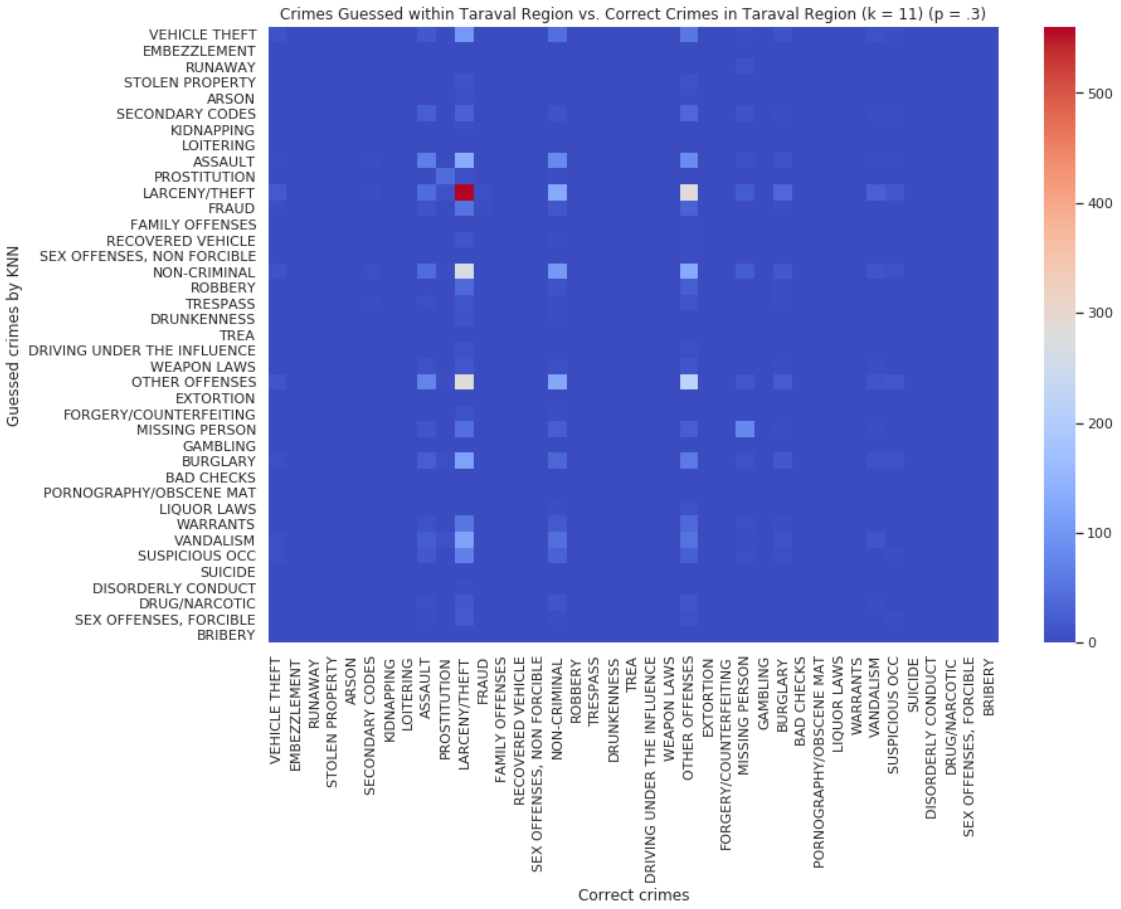
	Category	X	Y
9	LARCENY/THEFT	-122.477377	37.764478
10	NON-CRIMINAL	-122.477960	37.745739
24	OTHER OFFENSES	-122.474445	37.718302
37	ROBBERY	-122.475647	37.728528
94	OTHER OFFENSES	-122.486926	37.749084

```
In [146]: train_taraval1, predict_taraval1 = split_sets(taraval_district, .3)
          taraval_test1 = KNN(11)
          taraval_test1.train(train_taraval1)
          subset_taraval = predict_taraval1[:5000]
          # Creates a list of tuples, (what the algorithm guessed, what is correct)
          sub_cf_taraval = [(taraval_test1.predict(tupl[0]), tupl[1]) for tupl in subset_tarava
```

```
In [147]: # confusion_matrix = [([0,0], district) for district in parse_districts]
          # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_norther
          accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_c
```

```
        # INITIALIZE CONFUSION MATRIX FOR TARAVAL HERE!!!!!!!!!!!!!!!!!!!!!!!!
        taraval_learning_curve = []

        for prediction in sub_cf_taraval:
            accum_category_nums[prediction[1]][prediction[0]] += 1
        confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_
        #print(confusion_matrix)
        sub_percentage = 0
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        taraval_learning_curve.append([11, sub_percentage])
        #print(taraval_learning_curve)
```

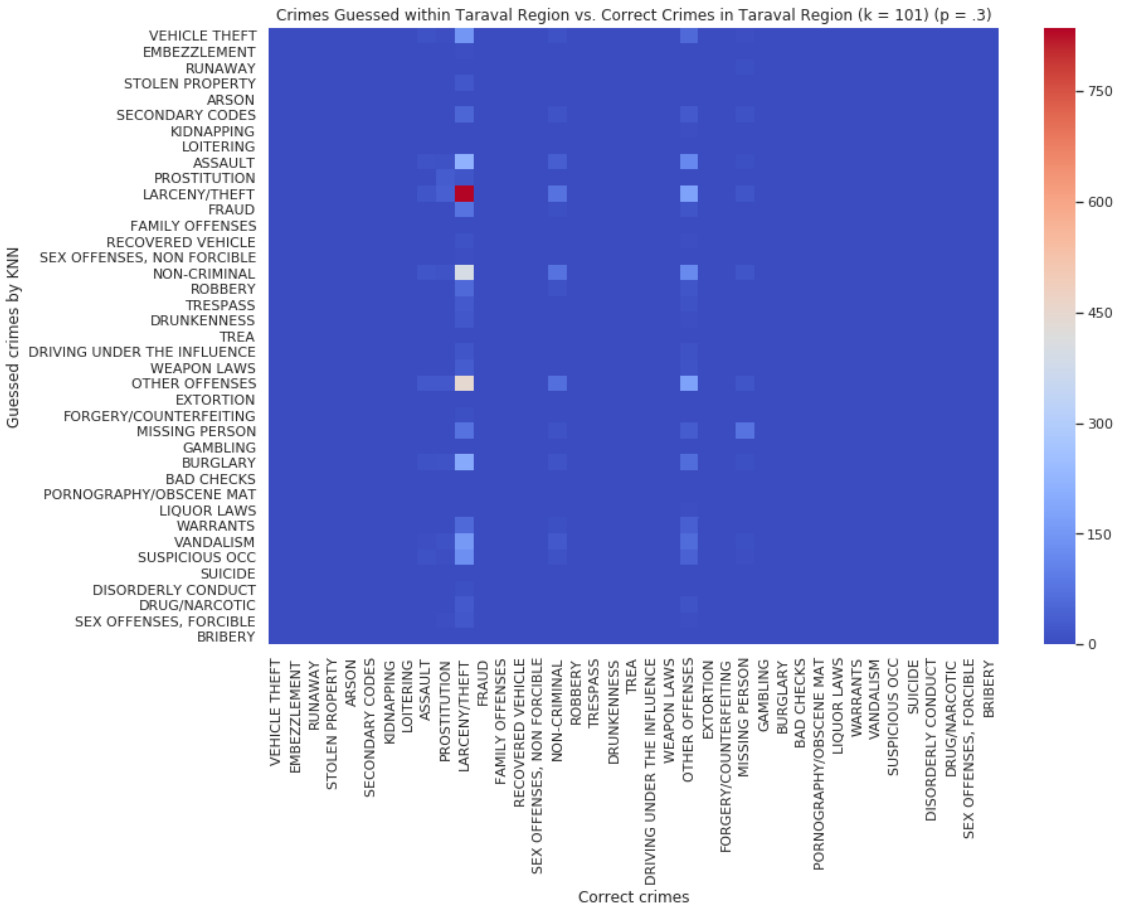
In [148]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
        ax.set(title='Crimes Guessed within Taraval Region vs. Correct Crimes in Taraval Reg
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```



In [149]:
```
train_taraval2, predict_taraval2 = split_sets(taraval_district, .3)
taraval_test2 = KNN(101)
taraval_test2.train(train_taraval2)
subset_taraval = predict_taraval2[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_taraval = [(taraval_test2.predict(tupl[0]), tupl[1]) for tupl in subset_tarava
```

In [150]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_norther
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

for prediction in sub_cf_taraval:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_n
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
taraval_learning_curve.append([101, sub_percentage])
#print(taraval_learning_curve)
```

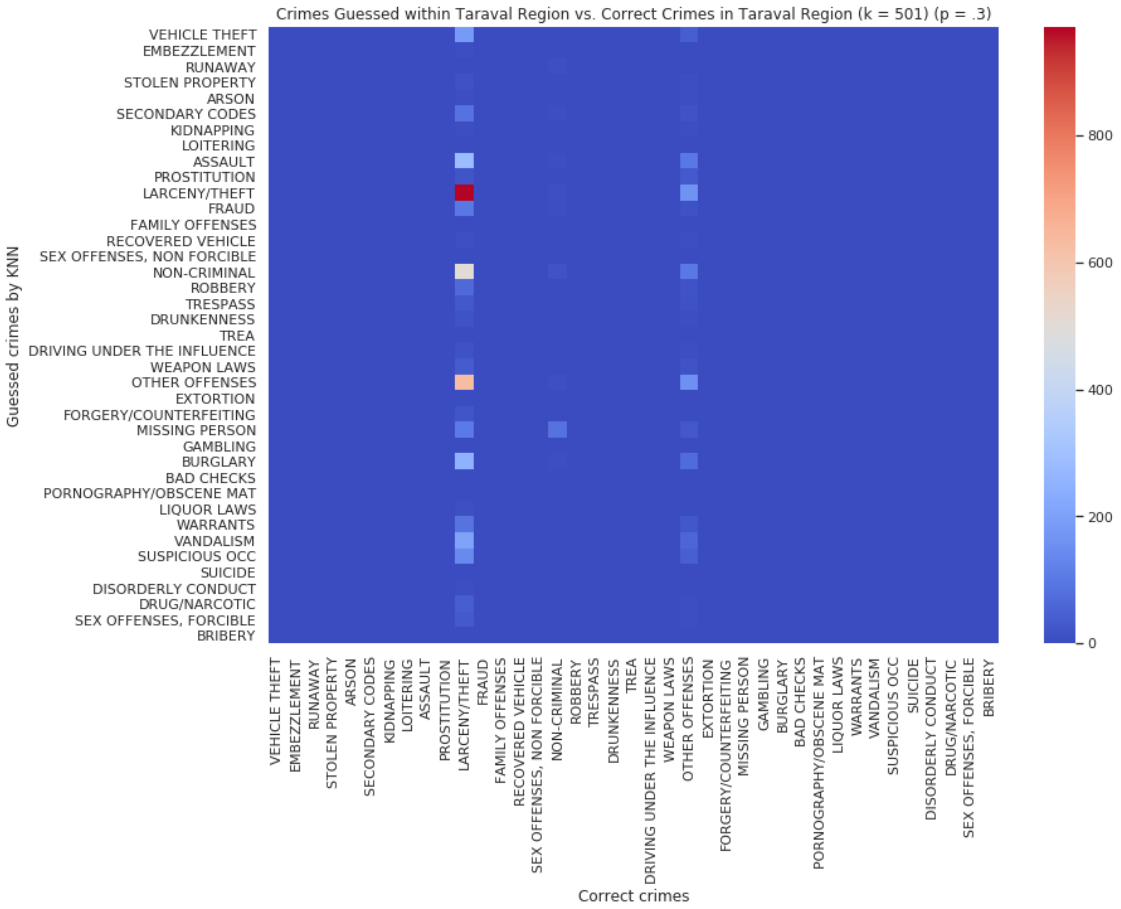
In [151]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
ax.set(title='Crimes Guessed within Taraval Region vs. Correct Crimes in Taraval Regi
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```




```
In [152]: train_taraval3, predict_taraval3 = split_sets(taraval_district, .3)
          taraval_test3 = KNN(501)
          taraval_test3.train(train_taraval3)
          subset_taraval = predict_taraval3[:5000]
          # Creates a list of tuples, (what the algorithm guessed, what is correct)
          sub_cf_taraval = [(taraval_test3.predict(tupl[0]), tupl[1]) for tupl in subset_tarava

In [153]: # confusion_matrix = [([0,0], district) for district in parse_districts]
          # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northers
          accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

          for prediction in sub_cf_taraval:
              accum_category_nums[prediction[1]][prediction[0]] += 1
          confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_n
          #print(confusion_matrix)
          sub_percentage = 0
```

```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        taraval_learning_curve.append([501, sub_percentage])
        #print(taraval_learning_curve)
```

In [154]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
        ax.set(title='Crimes Guessed within Taraval Region vs. Correct Crimes in Taraval Regi
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```

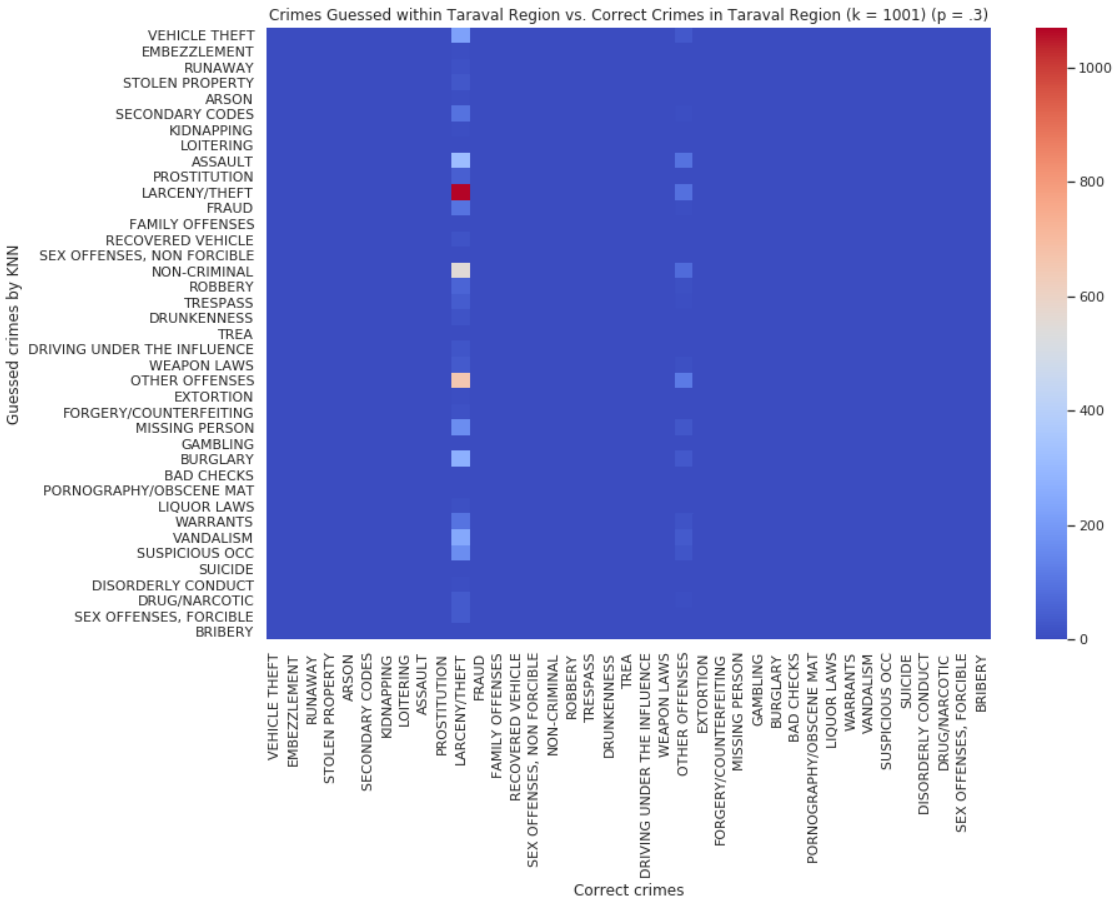


In [155]:
```
        train_taraval4, predict_taraval4 = split_sets(taraval_district, .3)
        taraval_test4 = KNN(1001)
```

```
taraval_test4.train(train_taraval4)
subset_taraval = predict_taraval4[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_taraval = [(taraval_test4.predict(tupl[0]), tupl[1]) for tupl in subset_tarava
```

In [156]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_norther
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_c

for prediction in sub_cf_taraval:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
taraval_learning_curve.append([1001, sub_percentage])
#print(taraval_learning_curve)
```

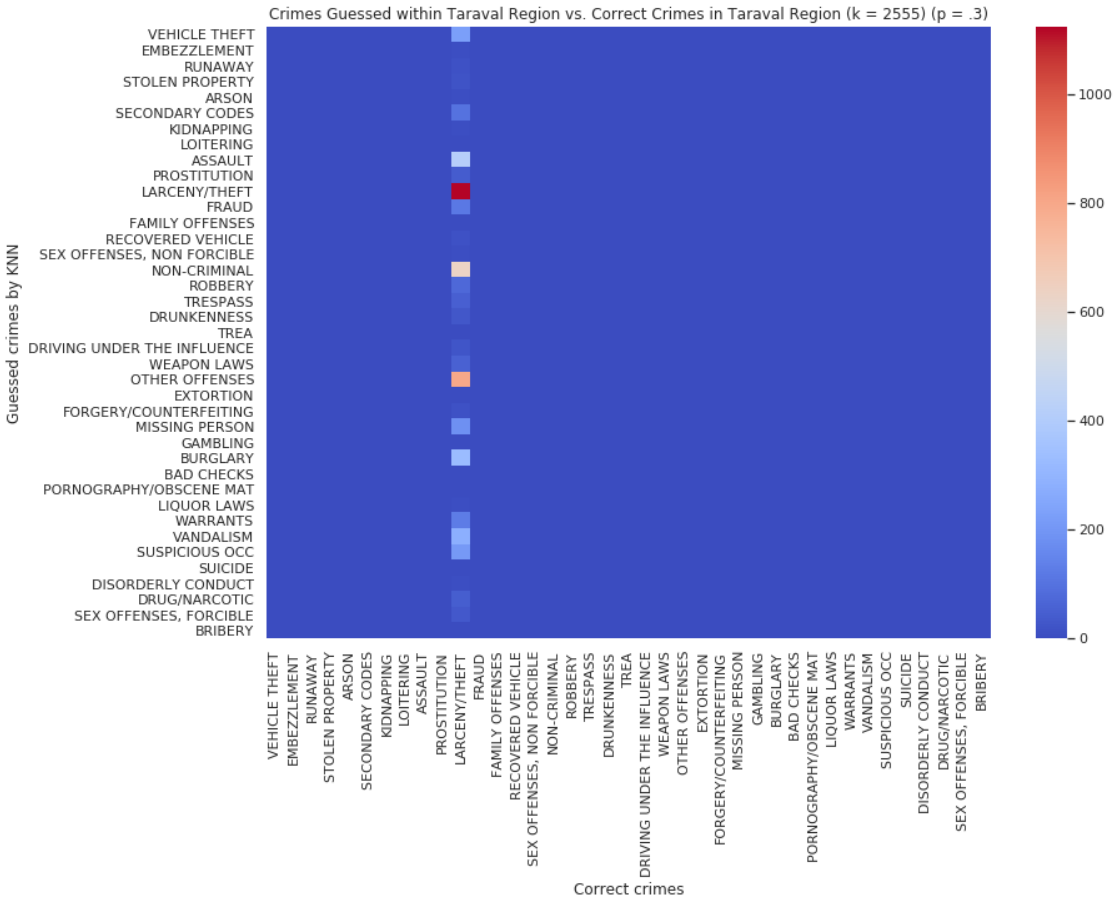
In [157]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
ax.set(title='Crimes Guessed within Taraval Region vs. Correct Crimes in Taraval Reg
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```




```
In [158]: train_taraval5, predict_taraval5 = split_sets(taraval_district, .3)
          taraval_test5 = KNN(2555)
          taraval_test5.train(train_taraval5)
          subset_taraval = predict_taraval5[:5000]
          # Creates a list of tuples, (what the algorithm guessed, what is correct)
          sub_cf_taraval = [(taraval_test5.predict(tupl[0]), tupl[1]) for tupl in subset_tarava

In [159]: # confusion_matrix = [([0,0], district) for district in parse_districts]
          # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_norther
          accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_c

          for prediction in sub_cf_taraval:
              accum_category_nums[prediction[1]][prediction[0]] += 1
          confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_
          #print(confusion_matrix)
          sub_percentage = 0
```

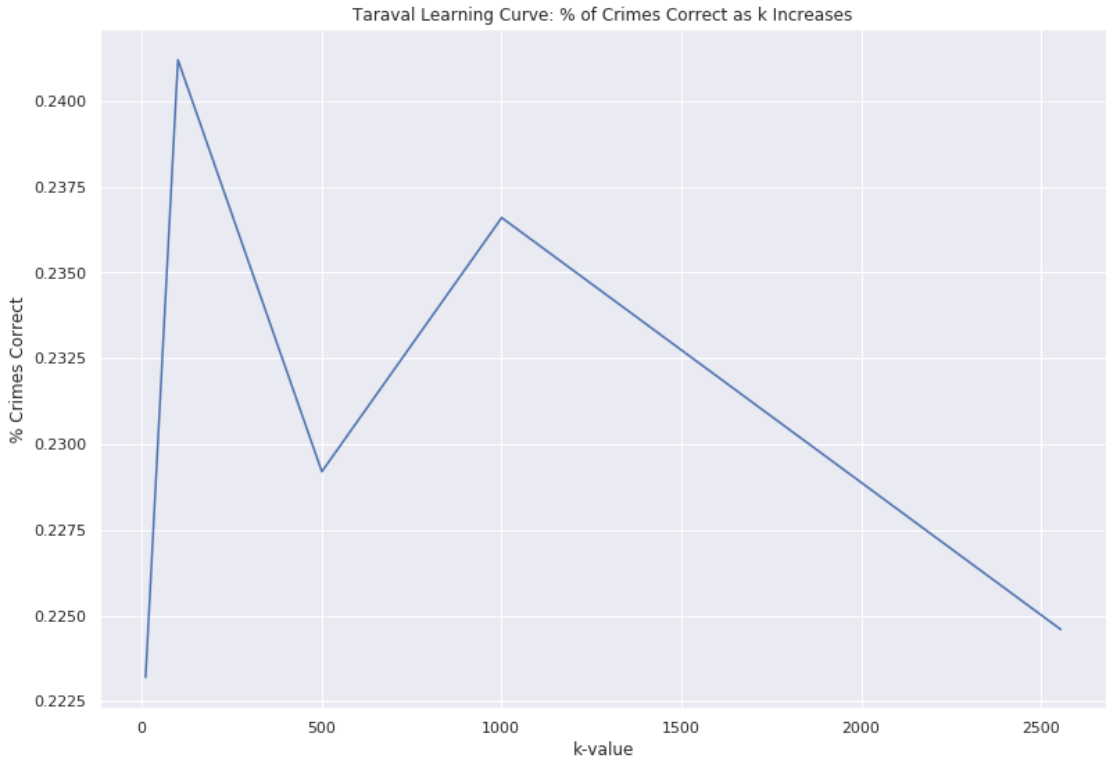
```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        taraval_learning_curve.append([2555, sub_percentage])
        #print(taraval_learning_curve)
```

In [160]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
        ax.set(title='Crimes Guessed within Taraval Region vs. Correct Crimes in Taraval Reg
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```



In [161]:
```
        # Printing the learning curve for bayview
        # print(taraval_learning_curve)
```

```
plt.plot([i[0] for i in taraval_learning_curve], [i[1] for i in taraval_learning_cur
plt.title('Taraval Learning Curve: % of Crimes Correct as k Increases')
plt.xlabel('k-value')
plt.ylabel('% Crimes Correct')
plt.show()
```



15.2.10 Richmond:

```
In [162]: richmond_district = crime_data[crime_data['PdDistrict'] == 'RICHMOND'][['Category',
          richmond_district.head()
```

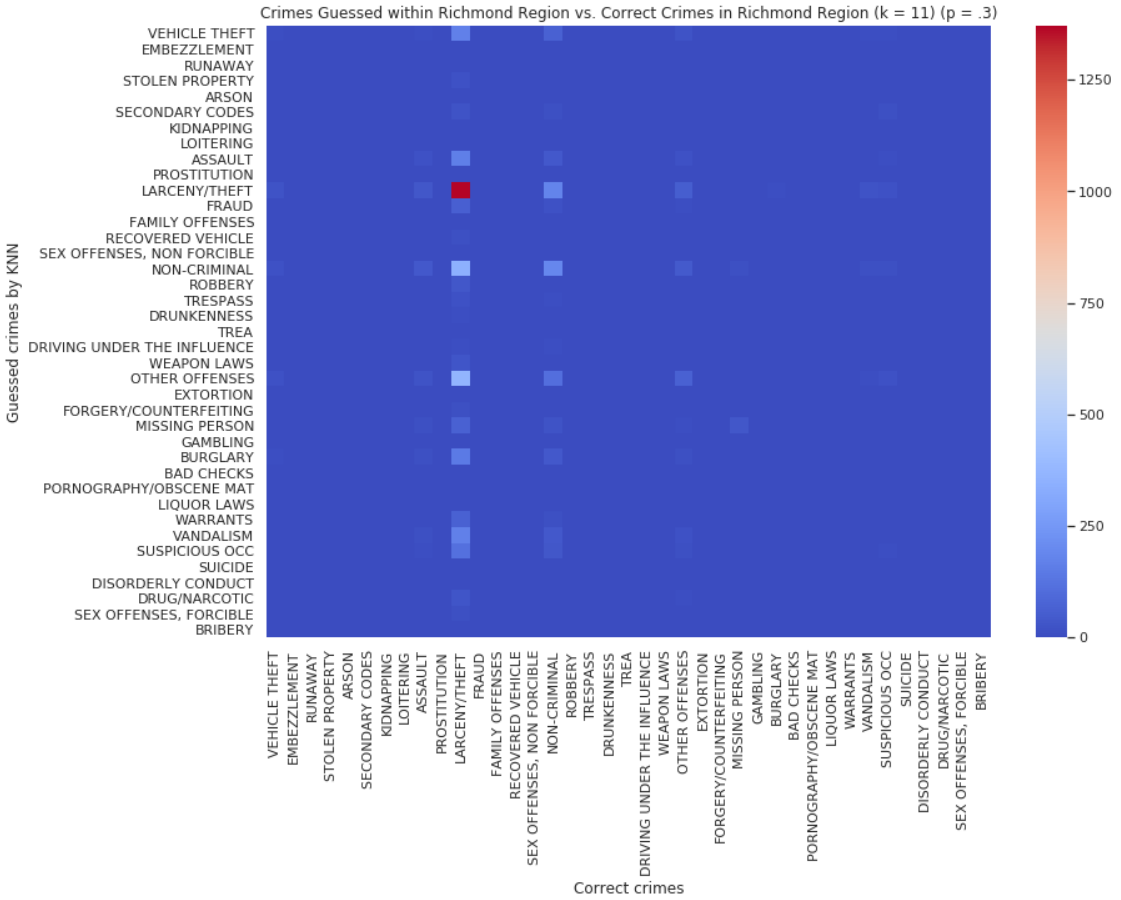
Out[162]:

	Category	X	Y
23	ASSAULT	-122.507750	37.771494
92	OTHER OFFENSES	-122.500380	37.774621
93	ROBBERY	-122.466559	37.775192
145	NON-CRIMINAL	-122.472982	37.781571
165	LARCENY/THEFT	-122.454819	37.780116

```
In [163]: train_richmond1, predict_richmond1 = split_sets(richmond_district, .3)
          richmond_test1 = KNN(11)
          richmond_test1.train(train_richmond1)
          subset_richmond = predict_richmond1[:5000]
          # Creates a list of tuples, (what the algorithm guessed, what is correct)
          sub_cf_richmond = [(richmond_test1.predict(tupl[0]), tupl[1]) for tupl in subset_ric
```

In [164]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_northern
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

# INITIALIZE CONFUSION MATRIX FOR RICHMOND HERE!!!!!!!!!!!!!!!!!!!!!!!!
richmond_learning_curve = []

for prediction in sub_cf_richmond:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_n
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
richmond_learning_curve.append([11, sub_percentage])
#print(richmond_learning_curve)
```

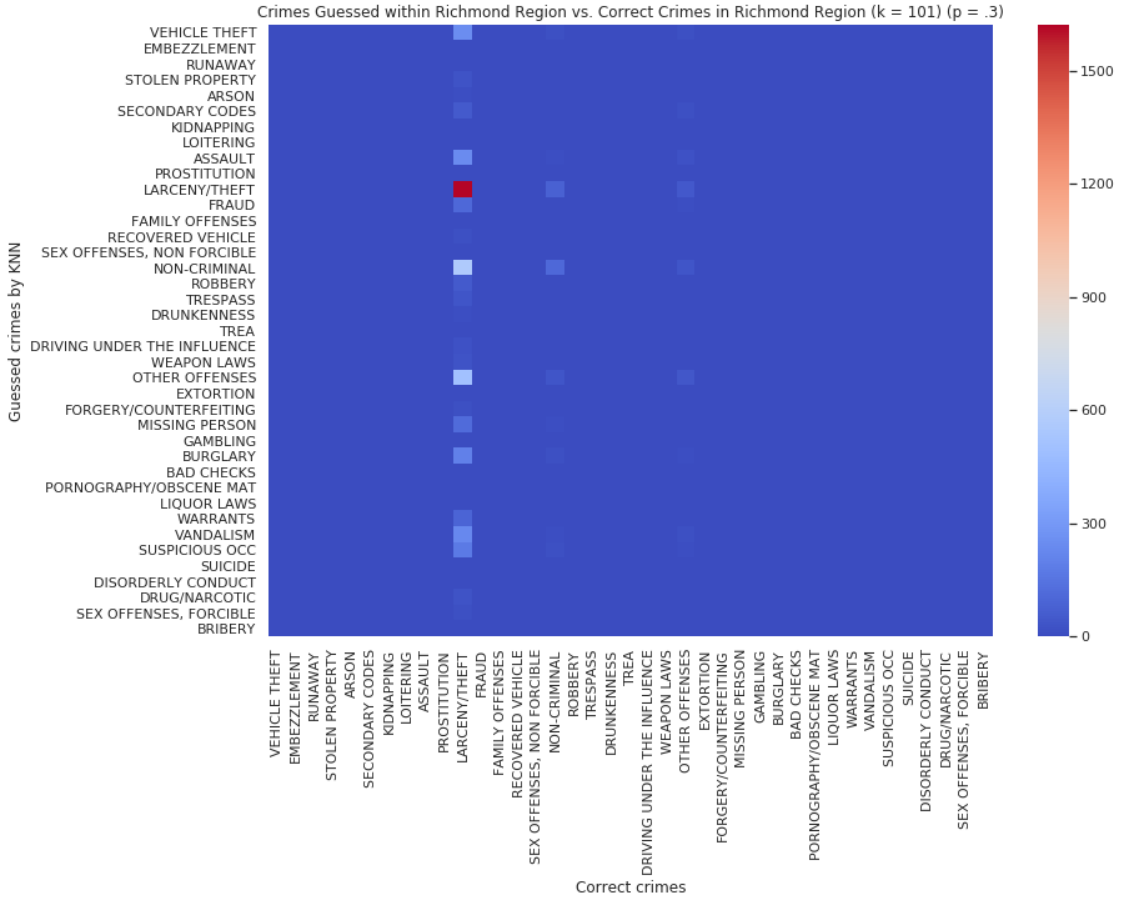
In [165]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
ax.set(title='Crimes Guessed within Richmond Region vs. Correct Crimes in Richmond Re
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```




```
In [166]: train_richmond2, predict_richmond2 = split_sets(richmond_district, .3)
          richmond_test2 = KNN(101)
          richmond_test2.train(train_richmond2)
          subset_richmond = predict_richmond2[:5000]
          # Creates a list of tuples, (what the algorithm guessed, what is correct)
          sub_cf_richmond = [(richmond_test2.predict(tupl[0]), tupl[1]) for tupl in subset_rich

In [167]: # confusion_matrix = [([0,0], district) for district in parse_districts]
          # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_norther
          accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_c

          for prediction in sub_cf_richmond:
              accum_category_nums[prediction[1]][prediction[0]] += 1
          confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_
          #print(confusion_matrix)
          sub_percentage = 0
```

```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        richmond_learning_curve.append([101, sub_percentage])
        #print(richmond_learning_curve)
```

In [168]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
        ax.set(title='Crimes Guessed within Richmond Region vs. Correct Crimes in Richmond Re
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```

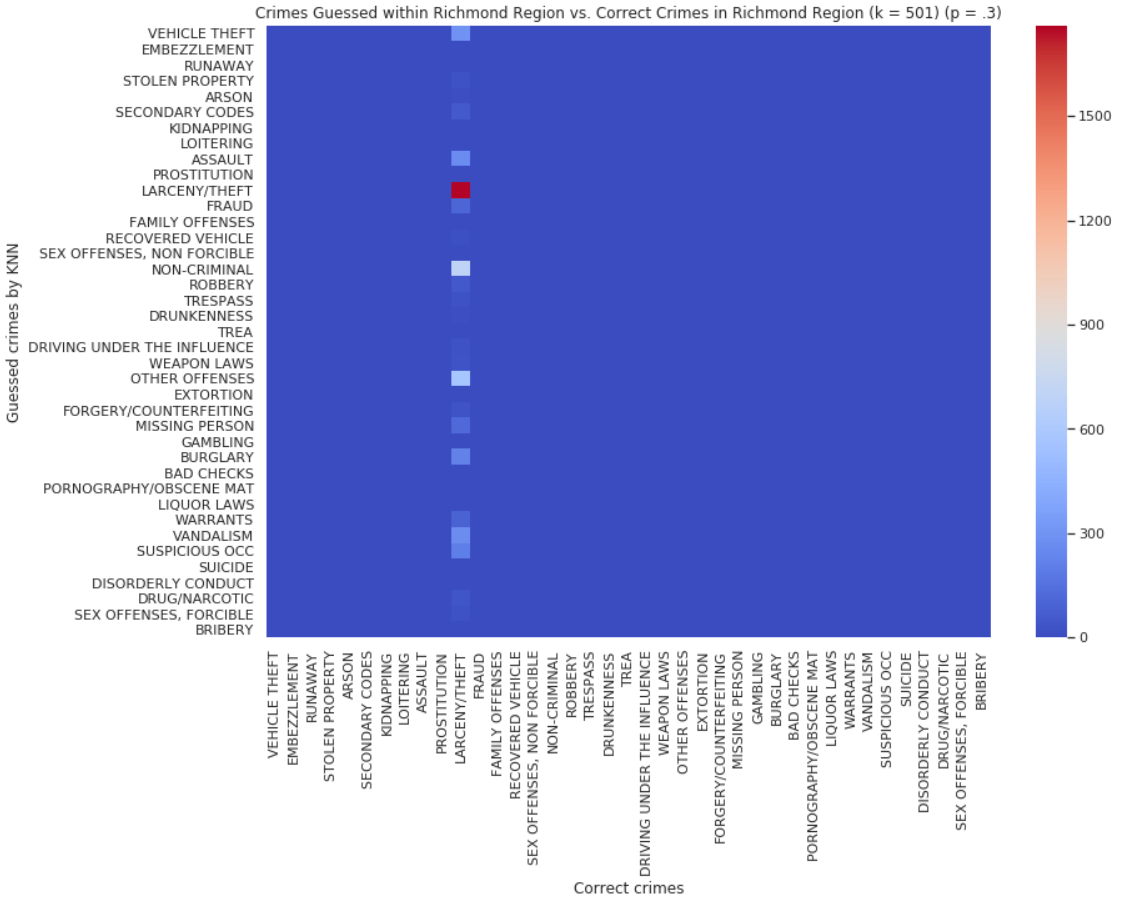


In [169]:
```
        train_richmond3, predict_richmond3 = split_sets(richmond_district, .3)
        richmond_test3 = KNN(501)
```

```
        richmond_test3.train(train_richmond3)
        subset_richmond = predict_richmond3[:5000]
        # Creates a list of tuples, (what the algorithm guessed, what is correct)
        sub_cf_richmond = [(richmond_test3.predict(tupl[0]), tupl[1]) for tupl in subset_rich
```

In [170]:
```
        # confusion_matrix = [([0,0], district) for district in parse_districts]
        # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_norther
        accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_cr

        for prediction in sub_cf_richmond:
            accum_category_nums[prediction[1]][prediction[0]] += 1
        confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_n
        #print(confusion_matrix)
        sub_percentage = 0
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        richmond_learning_curve.append([501, sub_percentage])
        #print(richmond_learning_curve)
```

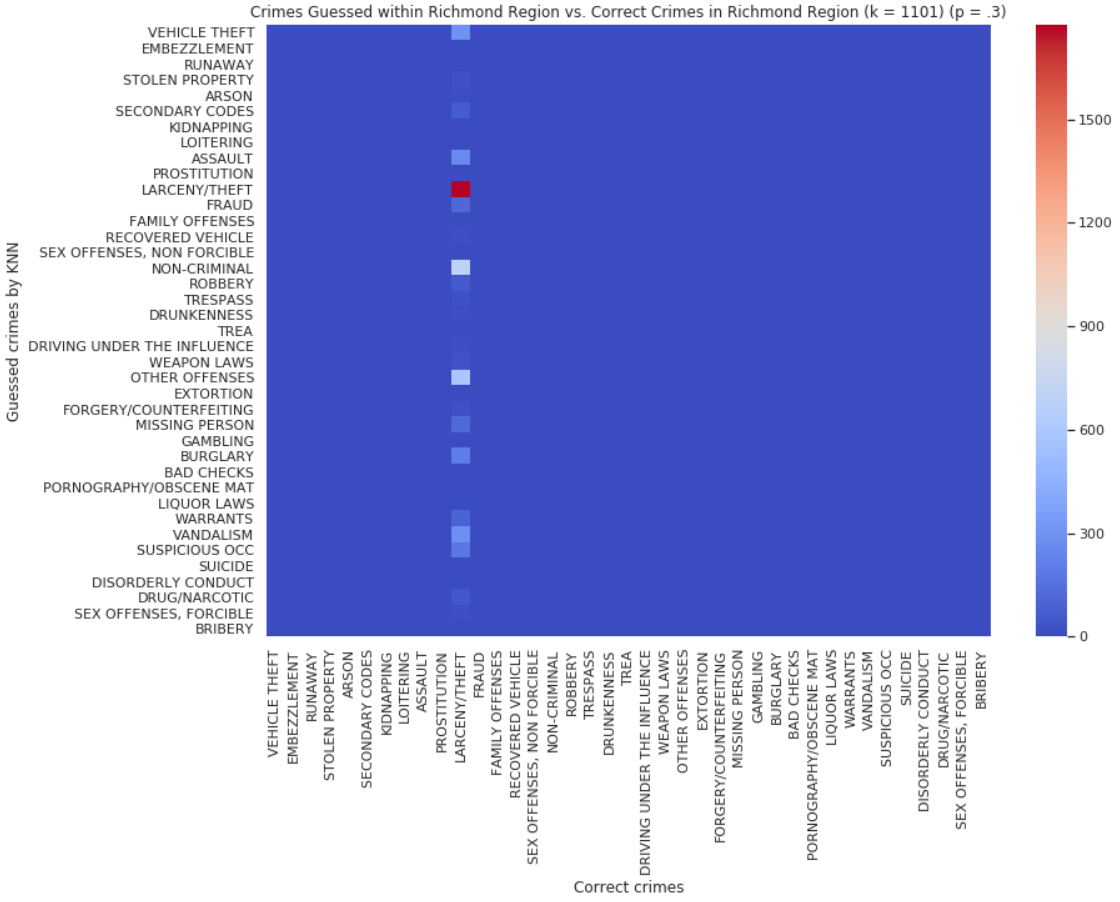
In [171]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
        ax.set(title='Crimes Guessed within Richmond Region vs. Correct Crimes in Richmond Re
              xlabel='Correct crimes',
              ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```




```
In [172]: train_richmond4, predict_richmond4 = split_sets(richmond_district, .3)
          richmond_test4 = KNN(1101)
          richmond_test4.train(train_richmond4)
          subset_richmond = predict_richmond4[:5000]
          # Creates a list of tuples, (what the algorithm guessed, what is correct)
          sub_cf_richmond = [(richmond_test4.predict(tupl[0]), tupl[1]) for tupl in subset_rich

In [173]: # confusion_matrix = [([0,0], district) for district in parse_districts]
          # confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_norther
          accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_c

          for prediction in sub_cf_richmond:
              accum_category_nums[prediction[1]][prediction[0]] += 1
          confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_
          #print(confusion_matrix)
          sub_percentage = 0
```

```
        for ind, row in enumerate(confusion_matrix):
            sub_percentage += row[ind]
        sub_percentage /= 5000
        richmond_learning_curve.append([1101, sub_percentage])
        #print(richmond_learning_curve)
```

In [174]:
```
        # Plot the heatmap using seaborn
        sns.set(rc={'figure.figsize':(13,9)})
        ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
        ax.set(title='Crimes Guessed within Richmond Region vs. Correct Crimes in Richmond Re
               xlabel='Correct crimes',
               ylabel='Guessed crimes by KNN')
        plt.show()
        print()
```

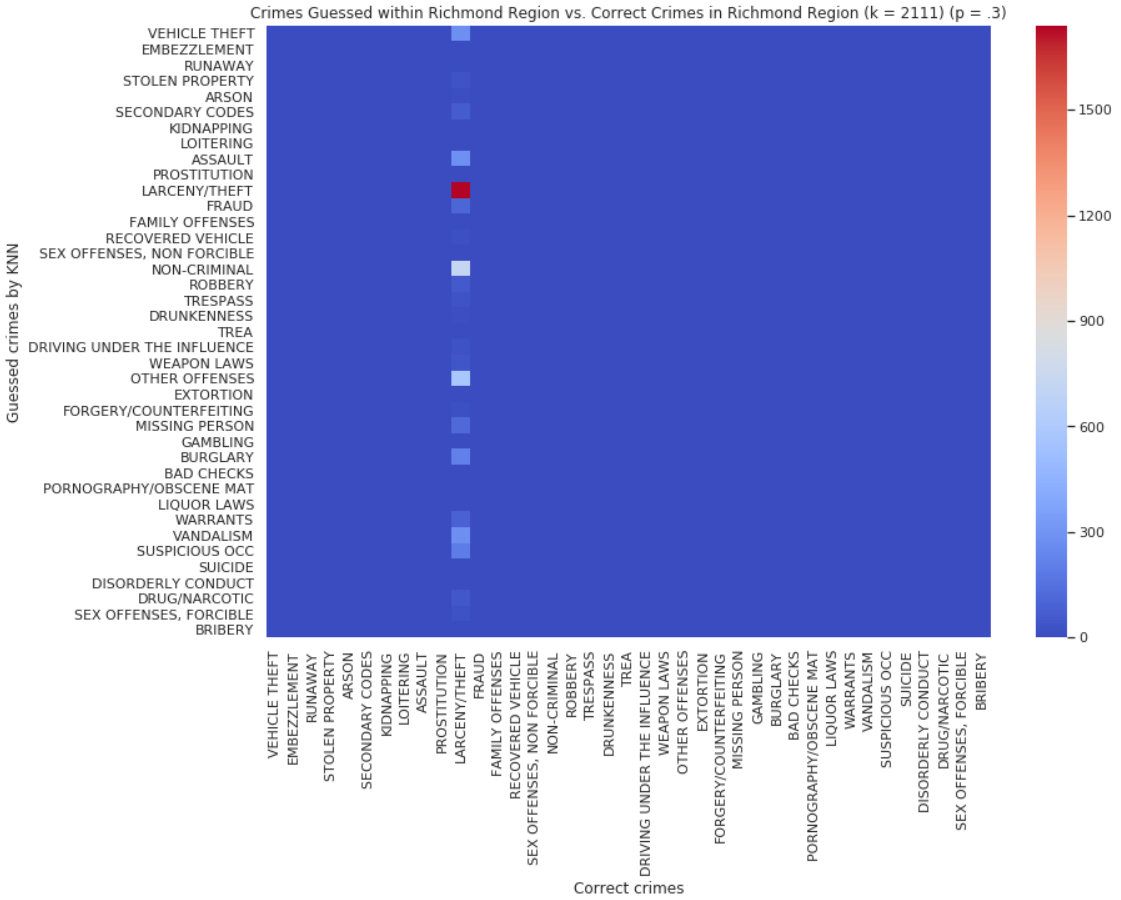


In [175]:
```
        train_richmond5, predict_richmond5 = split_sets(richmond_district, .3)
        richmond_test5 = KNN(2111)
```

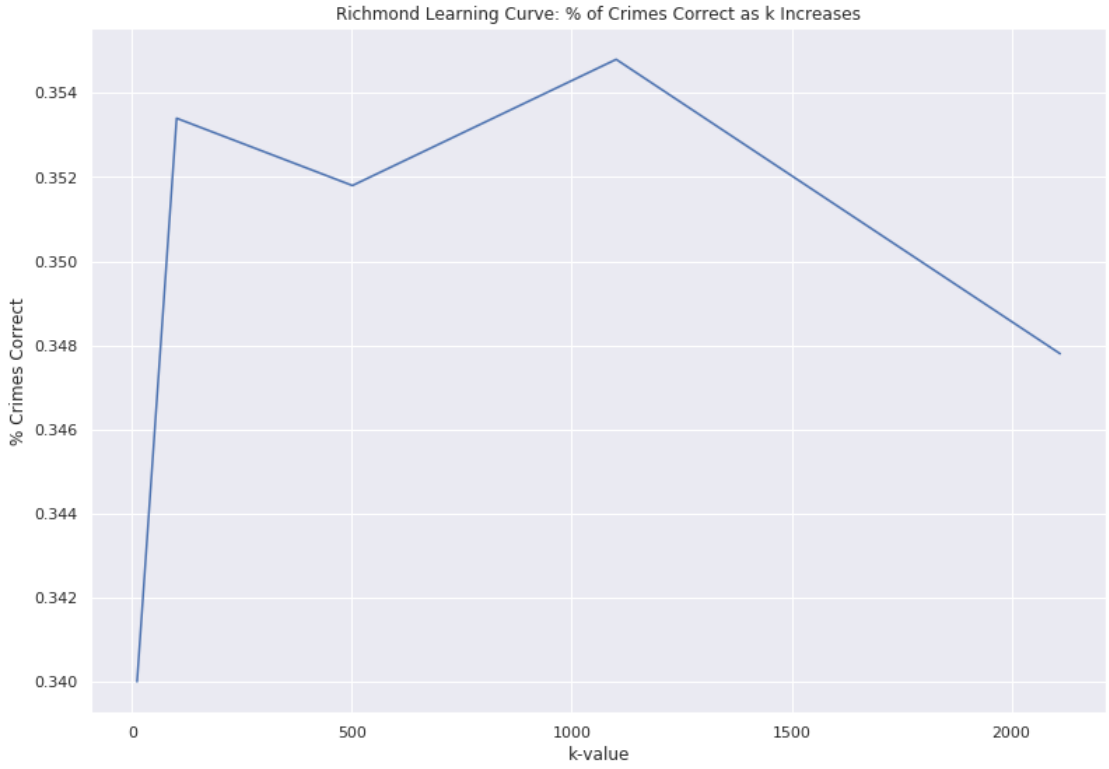
```
richmond_test5.train(train_richmond5)
subset_richmond = predict_richmond5[:5000]
# Creates a list of tuples, (what the algorithm guessed, what is correct)
sub_cf_richmond = [(richmond_test5.predict(tupl[0]), tupl[1]) for tupl in subset_rich
```

In [176]:
```
# confusion_matrix = [([0,0], district) for district in parse_districts]
# confusion_matrix = [ for pair in confusion_matrix for prediction in sub_cf_norther
accum_category_nums = {correct_crime: {guessed_crime: 0 for guessed_crime in parse_c

for prediction in sub_cf_richmond:
    accum_category_nums[prediction[1]][prediction[0]] += 1
confusion_matrix = [[value for value in dicts.values()] for dicts in accum_category_
#print(confusion_matrix)
sub_percentage = 0
for ind, row in enumerate(confusion_matrix):
    sub_percentage += row[ind]
sub_percentage /= 5000
richmond_learning_curve.append([2111, sub_percentage])
#print(richmond_learning_curve)
```

In [177]:
```
# Plot the heatmap using seaborn
sns.set(rc={'figure.figsize':(13,9)})
ax = sns.heatmap(data=np.array(confusion_matrix), cmap='coolwarm', cbar=True, xtickla
ax.set(title='Crimes Guessed within Richmond Region vs. Correct Crimes in Richmond R
      xlabel='Correct crimes',
      ylabel='Guessed crimes by KNN')
plt.show()
print()
```



In [178]:
```
# Printing the learning curve for richmond
# print(richmond_learning_curve)
plt.plot([i[0] for i in richmond_learning_curve], [i[1] for i in richmond_learning_cu
plt.title('Richmond Learning Curve: % of Crimes Correct as k Increases')
plt.xlabel('k-value')
plt.ylabel('% Crimes Correct')
plt.show()
```


Richmond Learning Curve: % of Crimes Correct as k Increases
0.354
0.352
0.350
0.348
0.346
0.344
0.342
0.340
% Crimes Correct
0
500
1000
1500
2000
k-value