Edward Almoite
COMP 4254
July 8, 2022

Factors Affecting Crime in Vancouver

Introduction

In this project, I wanted to look at factors that affect crime in Vancouver. To do this, I downloaded the Vancouver Crime Database from the Vancouver Police Department website, as well as the 2016 Canada census data.

Dissemination Areas

The Canada census divides Vancouver into dissemination areas - small areas of mostly between 400-700 citizens. There are approximately 400 dissemination areas in Vancouver. Each dissemination area has information on variables such as income, education, and number of immigrants. I wanted to see if these variables had an effect on the number of crimes in a dissemination area.

Preparing the Datasets

2016 Canada Census
I filtered the data to only include dissemination areas in Vancouver. Then, I transposed the data and saved it into a new csv file.

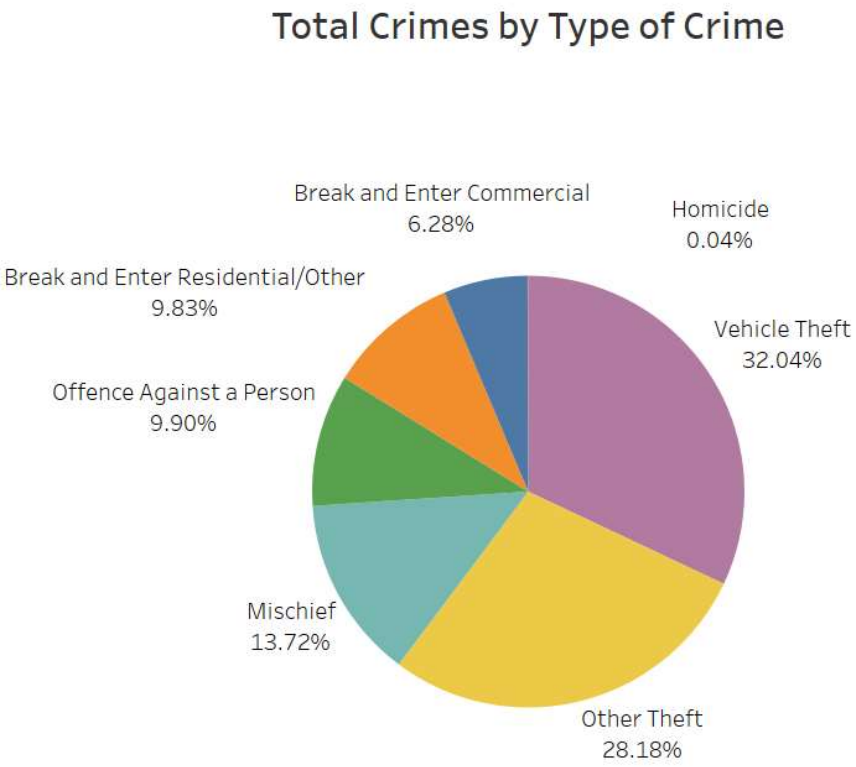
Vancouver Crime Dataset
The original dataset contained coordinates in UTM Zone 10 and were converted to latitude/longitude using Python. Then, I filtered the data to only include 2016 crimes. Next, in Python, I iterated through the crime dataset and selected the dissemination area closest to the crime.

Combining the Data
For each dissemination area, I counted the number of crimes that occurred and included this data in the census. My final dataset used the number of crimes in each area as the response variable and the variables in the census data as the predictor variables.

Exploratory Data Analysis

What types of crimes are reported in Vancouver?

Over 60% of crimes reported are thefts, while only 0.04% of crimes are homicides.



What is the average number of crimes that occur on a certain day in Vancouver?

According to the chart below, most crimes are reported:

- On the first day of the month
- In the middle of the month
- On New Year's Day, Halloween and June 15

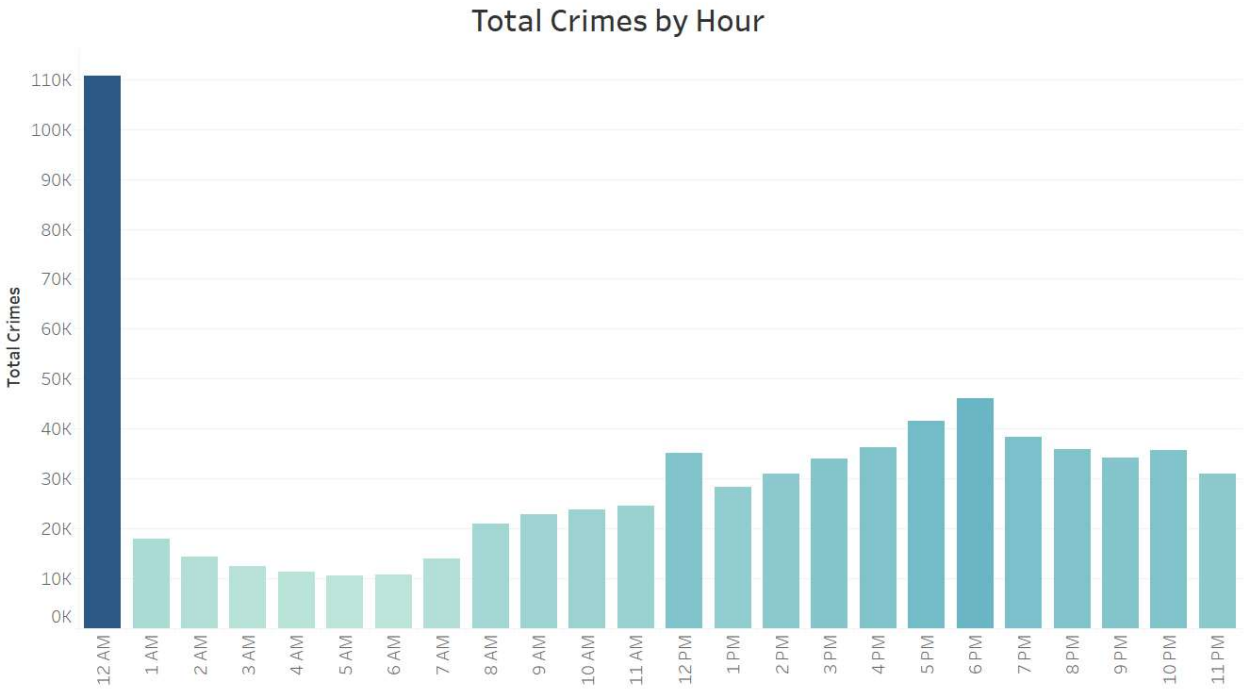
Additionally, less crimes are reported in July and December, possibly because people are on vacation.

Average Crimes Reported by Day/Month

Day of ..	Date											
	January	February	March	April	May	June	July	August	Septem..	October	Novemb..	December
1	139.85	113.45	111.20	112.15	112.55	110.80	109.45	113.95	115.15	108.00	116.95	103.70
2	97.70	105.55	96.35	102.10	92.70	99.05	97.65	100.50	99.60	94.60	95.95	91.40
3	103.15	104.00	97.30	104.45	95.40	99.05	98.90	100.30	103.00	95.30	98.15	94.75
4	98.10	102.10	97.75	100.75	94.35	99.30	97.80	98.70	101.75	95.10	100.25	96.05
5	100.95	105.50	102.10	99.15	100.05	103.10	98.15	99.60	99.25	98.75	104.70	100.40
6	98.55	104.95	99.45	102.35	103.10	107.25	95.55	101.80	99.35	94.05	101.50	100.90
7	108.65	107.10	102.60	102.05	100.65	102.05	98.45	101.55	101.50	101.50	101.00	100.80
8	105.00	105.75	102.10	109.85	103.60	103.20	94.05	102.80	101.80	96.40	100.50	97.30
9	100.55	106.60	97.10	107.00	104.60	102.20	99.00	101.70	102.40	100.25	101.40	100.65
10	105.00	101.55	101.30	101.30	106.65	104.45	92.60	103.55	107.05	100.75	107.30	96.10
11	104.05	102.45	102.25	107.00	104.75	106.15	96.00	100.60	103.15	100.60	107.30	99.60
12	100.30	102.60	111.05	103.90	101.95	107.30	98.05	102.70	108.20	103.15	109.20	97.40
13	105.80	108.35	106.85	104.20	106.55	103.85	98.45	103.90	102.70	97.10	102.20	98.50
14	104.15	107.35	98.45	109.10	108.60	104.45	99.10	103.75	102.40	101.10	108.10	95.55
15	105.80	110.10	108.90	104.85	109.05	136.95	103.35	109.15	104.40	108.85	109.75	96.30
16	104.90	103.10	102.80	102.65	105.00	104.70	99.60	99.90	106.15	104.30	105.75	92.95
17	109.55	98.20	107.85	101.10	104.75	105.40	100.25	105.00	106.40	103.95	100.25	85.75
18	101.30	94.95	105.70	99.80	101.90	105.65	100.60	104.80	103.55	99.55	94.95	84.25
19	100.50	95.70	104.35	101.75	97.70	98.55	101.85	100.65	97.00	96.85	94.35	81.00
20	99.55	87.70	103.95	98.80	103.85	109.45	100.25	104.60	99.60	101.10	93.55	86.25
21	94.00	88.65	94.55	94.65	96.85	101.50	95.20	100.55	94.00	97.90	85.90	77.75
22	89.80	89.75	100.60	95.90	95.80	95.20	100.50	99.65	95.20	96.85	89.15	85.35
23	91.30	87.45	90.95	93.10	94.60	94.90	93.35	95.75	96.40	94.50	85.75	85.05
24	86.90	92.80	93.30	92.70	97.85	92.10	89.65	92.10	90.50	89.80	83.20	85.40
25	91.00	89.30	91.50	88.90	89.40	86.25	93.75	92.80	88.85	92.25	82.05	64.05
26	95.75	89.15	94.15	90.85	89.60	86.00	83.60	94.15	90.70	90.40	87.85	83.80
27	100.15	87.40	89.95	87.90	92.20	85.30	84.85	84.35	92.25	88.65	87.50	89.70
28	101.65	97.55	94.90	88.45	92.00	88.20	88.05	90.35	87.55	94.00	88.35	91.65
29	96.30		94.50	90.10	92.30	90.70	89.70	94.90	92.40	98.35	86.65	93.95
30	99.50		96.15	95.95	90.50	93.20	95.20	96.80	90.80	98.90	95.85	90.85
31	102.50		99.80		96.90		92.00	97.10		120.35		103.50

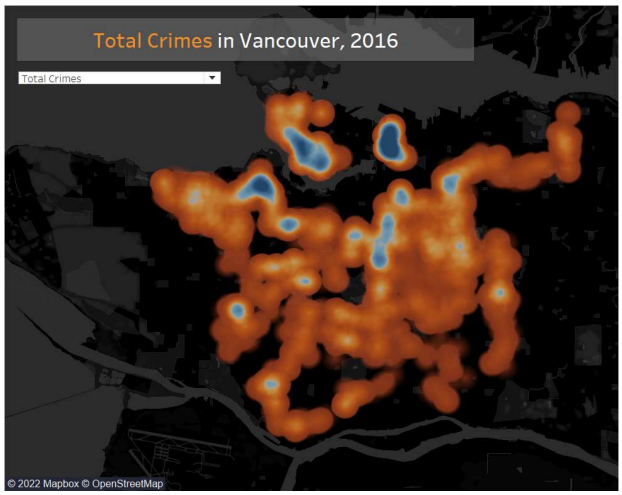
What are the total crimes reported per hour?

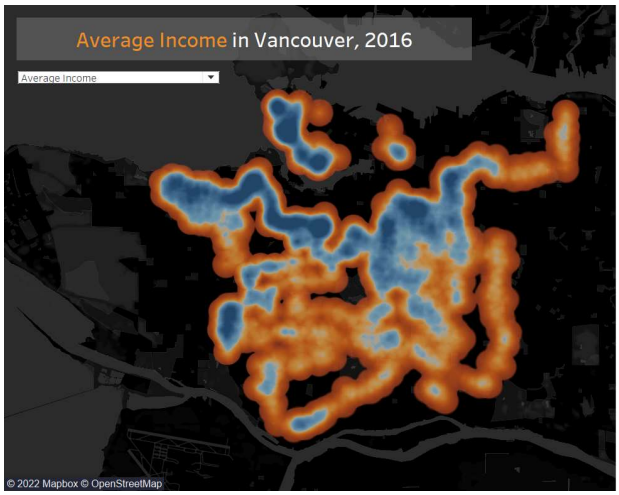
A majority of crimes reported occur between 12-1 AM.

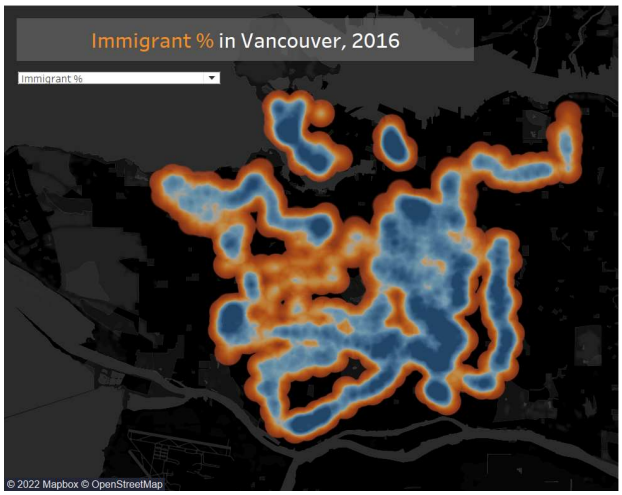


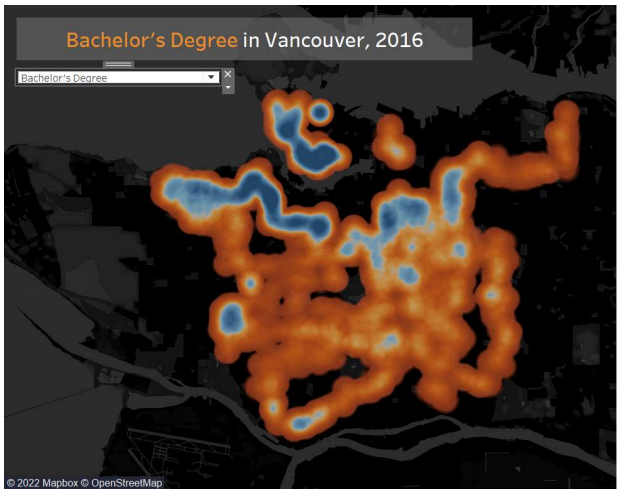
Census data in Vancouver, 2016

The following charts below are a density plot of total crimes, average income, percentage of immigrants, and total people with a bachelor's degree in Vancouver. Data was taken from the Canada 2016 census, using a density plot and parameters.







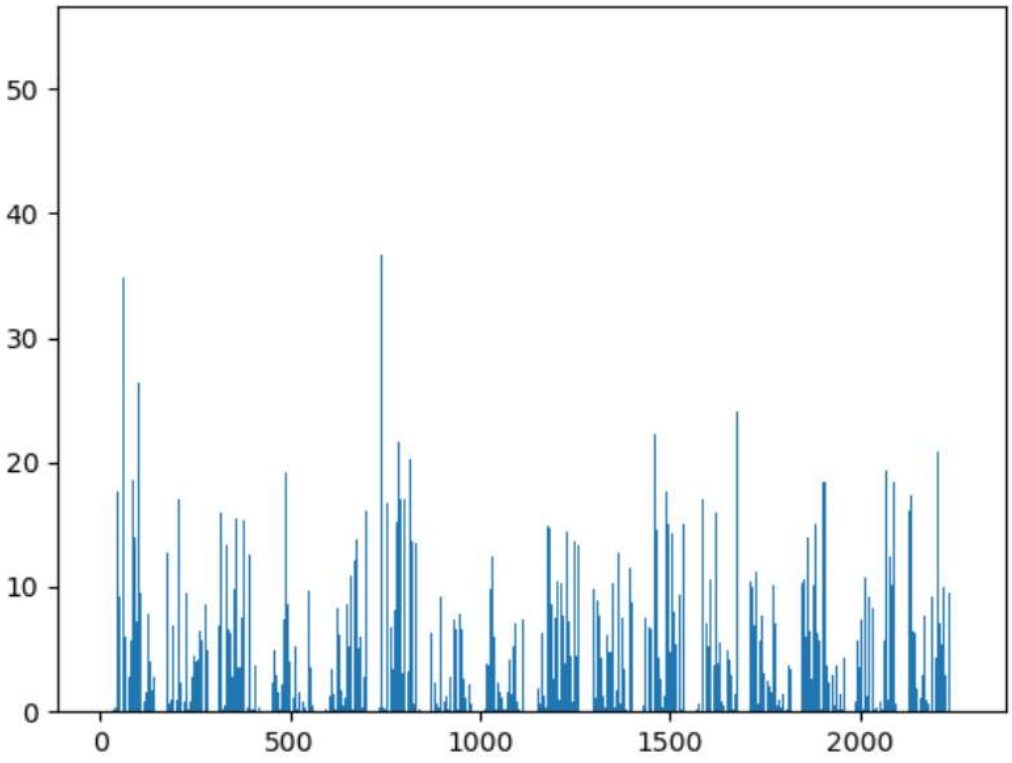


A few notes...

- The most crimes occurred in the Downtown, Strathcona and Kitsilano areas
- Average income and people with Bachelor's degrees are higher Downtown and in the northern areas of Vancouver
- Most immigrants live Downtown or in South Vancouver

Data-Preparation: Identifying Significant Features

First, I did a correlation feature selection to narrow down the variables. I chose variables with a score > 20.



Bar Chart of the Input Features (x) vs. Correlation Feature Importance (y)

Next, I performed 2 linear regressions and chose the variables with p-values < 0.05. These were the final variables that I chose.

- Multiple Aboriginal and non-Aboriginal ancestries
- No bedrooms
- 1991 to 2000.1
- Worked at home

I also performed a separate feature selection with a random forest. These were the most important features.

```
Variable: Separated            Importance: 0.02
Variable: Punjabi (Panjabi).2  Importance: 0.02
Variable: Government transfers (%) Importance: 0.01
Variable: $10,000 to $19,999   Importance: 0.01
Variable: In the seventh decile Importance: 0.01
Variable: Indo-Iranian languages.2 Importance: 0.01
Variable: Dutch.3              Importance: 0.01
Variable: Polish.3             Importance: 0.01
Variable: 1991 to 2000.1       Importance: 0.01
Variable: Major repairs needed Importance: 0.01
Variable: Average monthly shelter costs for rented dwellings ($) Importance: 0.01
```

After some experimentation, I added "Punjabi (Panjabi).2" and removed "Worked at home" from the model.

These are the final features that I chose for my model:

- Multiple Aboriginal and non-Aboriginal ancestries
- No bedrooms (Number of residences with no bedrooms)
- 1991 to 2000.1 (People who immigrated to Canada between 1991-2000)
- Punjabi (Panjabi).2

Data Modelling

I created different models using linear regression, linear regression with scaling, linear regression with Stacking, and linear regression with K-Fold validation.

Linear Regression Model

First, I performed a linear regression using the variables listed above.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:      total_crimes_2016   R-squared:                       0.187
Model:                            OLS   Adj. R-squared:                  0.178
Method:                 Least Squares   F-statistic:                     19.88
Date:                Wed, 10 Aug 2022   Prob (F-statistic):           9.65e-15
Time:                        21:23:56   Log-Likelihood:                -983.47
No. Observations:                 350   AIC:                             1977.
Df Residuals:                     345   BIC:                             1996.
Df Model:                           4
Covariance Type:            nonrobust
================================================================================================================
                                                   coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------------------------
const                                            3.5957      0.309     11.650      0.000       2.989       4.203
Multiple Aboriginal and non-Aboriginal ancestries  0.3874      0.087      4.443      0.000       0.216       0.559
No bedrooms                                      0.0306      0.012      2.467      0.014       0.006       0.055
1991 to 2000.1                                   0.0202      0.004      4.588      0.000       0.012       0.029
Punjabi (Panjabi).2                             -0.0240      0.006     -4.054      0.000      -0.036      -0.012
==============================================================================
Omnibus:                      191.651   Durbin-Watson:                   2.083
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1591.502
Skew:                           2.158   Prob(JB):                         0.00
Kurtosis:                      12.513   Cond. No.                         92.1
```

The Adjusted R2 is 17.8% and the Root Mean Square Error (RMSE) is 3.1929927632565462

Linear Regression Model with Scaling

Next, I performed a linear regression using 3 different scalers (Robust, Standard and MinMax scalers). While the StandardScaler has a lower RMSE, the RobustScaler seems to be the better performing model, as it has a higher adjusted R2 and lower AIC and BIC.

	Scaler	R2	R2_Adj	AIC	BIC	rmse
0	Robust	0.234768	0.225921	900.310980	915.742713	3.366475
1	Standard	0.187312	0.177916	928.664318	944.096050	3.192993
2	MinMax	0.438033	0.431536	-541.784879	-526.353146	4.369603

Linear Regression with Stacking

Compared to the other models, the main Linear Regression model is competitive.

```
** Evaluate Base Models **
 RMSE:4.114 ElasticNet
 RMSE:3.66 SVR
 RMSE:5.401 DecisionTreeRegressor
 RMSE:4.054 AdaBoostRegressor
 RMSE:4.184 RandomForestRegressor
 RMSE:4.608 ExtraTreesRegressor

** Evaluate Stacked Model **
 RMSE:3.797 LinearRegression
```

Linear Regression with K-Fold Validation

For this model, I used 8 folds and summarized the RMSE, BIC and R2 averages below.

```
Scores for all folds:
*****************************************
RMSE Average :   3.863353659124056
RMSE SD:         0.8623621739735355
BIC Average :    2282.833203107123
BIC SD:          28.503903373313427
RSQ Average :    0.20362982116424827
RSQ SD:          0.01385102669171399
```

Final Model: Linear Regression with scaling and K-Fold Validation

For my final model, I used K-Fold validation with 8 folds and scaling. The results are below.

```
Scores for all folds:
*****************************************
RMSE Average :   3.8567453623357597
RMSE SD:         0.596911494719496
BIC Average :    2284.084287001558
BIC SD:          19.655664848261363
RSQ Average :    0.20257165694894141
RSQ SD:          0.011780705241800499
```

Conclusion

Looking at the final variables for my model, factors that have an influence crime include:

- having citizens with Aboriginal and non-Aboriginal ancestries

- having immigrant citizens
- neighbourhoods with no-bedroom dwellings

In the end, I chose the final model using 8 k-folds and scaling using the RobustScaler. While the model without scaling had a lower RMSE, I felt that scaling the results with 8 folds produced a more accurate result.

One thing I did not attempt was a PCA. I wonder if this could have chosen more correlated variables, which could have produced a lower RMSE. Also, I feel the data would be more accurate with more data, specifically more data on crimes committed. Maybe I could have included crime data for more years than just 2016, or included crime data from all areas of Canada instead of just Vancouver

Appendix

The following code takes the 2016 census and filters/transposes the data into a new csv file.

In [ ]:
```
import numpy as np
import pandas as pd

# This script selects all the dissemination areas from the 2016 census that are in Greate

# This is the 2016 census
PATH = "C:\\datasets\\crime\\"
CSV_DATA = "census2016.csv"
df = pd.read_csv(PATH + CSV_DATA)

# Show all columns.
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)

# This creates a list of all dissemination areas in Greater Vancouver
dfVan = pd.read_csv("C:\\datasets\\crime\\dis_area_lat_long.csv")
dArea = dfVan[' DAuid/ADidu'].tolist()

# Filtering the 2016 census to include only Greater Vancouver areas
df2 = df[df['GEO_CODE (POR)'].isin(dArea)]

# Remove columns with 'xx'
# df2 = df2[(df2['Dim: Sex (3): Member ID: [1]: Total - Sex']!='xx')]
# df2.sort_index()
# df2.to_csv(path_or_buf="C:\\datasets\\crime\\census_van_area.csv")

# Pivoting the table
table = pd.pivot_table(df2
                       , values=["Dim: Sex (3): Member ID: [1]: Total - Sex"]
                       , index=['GEO_CODE (POR)']
                       , columns=["Member ID: Profile of Dissemination Areas (2247)", "DI
                       , aggfunc=np.sum
                       , sort=False
                      )
```

```
# table.to_csv(path_or_buf="C:\\datasets\\crime\\census_with_dis_area.csv")
```

This code converts UTM coordinates from the crime dataset into latitude and longitude. Then, it iterates through the crime dataset and chooses the dissemination area closest to the crime.

In [ ]:

```
import pandas as pd
from pyproj import Proj
import numpy as np
import math

# This code converts UTM coordinates from the crime dataset into latitude and longitude.
# Then, it iterates through the crime dataset and chooses the dissemination area closest

# Receives X_test values and finds closest cluster for each.
def predictCluster(centroids, X_test, clusterLabels):
    bestClusterList = []

    for i in range(0, len(X_test)):
        smallestDistance = None
        bestCluster = None
        print(i, len(X_test))

        # Compare each X value proximity with all centroids.
        for row in range(centroids.shape[0]):
            distance = 0

            # Get absolute distance between centroid and X.
            for col in range(centroids.shape[1]):
                distance += \
                    math.sqrt((centroids[row][col] - X_test.iloc[i][col]) ** 2)

            # Initialize bestCluster and smallestDistance during first iteration.
            # OR re-assign bestCluster if smaller distance to centroid found.
            if (bestCluster == None or distance < smallestDistance):
                bestCluster = clusterLabels[row]
                smallestDistance = distance

        bestClusterList.append(bestCluster)

    return bestClusterList


PATH = "C:\\datasets\\"
CSV_DATA = "crimedata.csv"
LATLONG_DATA = "C:\\datasets\\crime\\dis_area_lat_long.csv"

# Show all columns.
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)

df = pd.read_csv(PATH + CSV_DATA)
df = df.dropna(how='any', axis=0)

# This converts UTM coordinates into latitude and longitude
myProj = Proj("+proj=utm +zone=10 +north +ellps=WGS84 +datum=WGS84 +units=m +no_defs")
lon, lat = myProj(df['X'].values, df['Y'].values, inverse=True)
UTMx, UTMy = myProj(lon, lat)
df['UTMx'] = UTMx
```

```
df['UTMy'] = UTMy
df['lon'] = lon
df['lat'] = lat
df = df[(df['YEAR'] == 2016)]

# New dataframe with latitude and longitude
print(df)

# This dataframe shows the latitude and longitude of dissemination areas in Vancouver
dis_df = pd.read_csv(LATLONG_DATA)
dis_df = dis_df[[' DAuid/ADidu', ' DArplat/Adlat', ' DArplong/ADlong']]
print(dis_df.head(10))

geom = dis_df[[' DArplat/Adlat', ' DArplong/ADlong']]
print(geom)

# Assign centroids
centroids = np.zeros((len(dis_df), 2))
for row in range(0, len(dis_df)):
    for col in range(0, 2):
        centroids[row][col] = geom.iloc[row, col]
print(centroids)

clusterLabels = []
for i in range(0, len(dis_df)):
    clusterLabels.append(dis_df.loc[i, " DAuid/ADidu"])
print(clusterLabels)

X_test = df[['lon', 'lat']]
predictions = predictCluster(centroids, X_test, clusterLabels)
print('**********Predictions***********')
print(predictions)

df['dis_area'] = 0
for i in range(0, len(predictions)):
    df.at[i, 'dis_area'] = clusterLabels[i]
    print(i)
print(df)
```

```
1450 44126
1451 44126
1452 44126
1453 44126
1454 44126
1455 44126
1456 44126
1457 44126
1458 44126
1459 44126
1460 44126
1461 44126
1462 44126
1463 44126
```

This code performs a Correlation Feature Selection and selects the features with the largest scores. Then, it performs 3 linear regressions, each time selecting only the features with p-values < 0.05 from the previous iteration.

In [ ]:

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
```

```
from matplotlib import pyplot
from sklearn.feature_selection import f_regression
from sklearn.feature_selection import SelectKBest
from sklearn.impute import KNNImputer
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler, StandardScaler, MinMaxScaler


# feature selection
def select_features(X_train, y_train, X_test, k):
    # configure to select all features
    fs = SelectKBest(score_func=f_regression, k=k)
    # learn relationship from training data
    fs.fit(X_train, y_train)
    # transform train input data
    X_train_fs = fs.transform(X_train)
    # transform test input data
    X_test_fs = fs.transform(X_test)
    return X_train_fs, X_test_fs, fs


PATH2 = "C:\\datasets\\crime\\"
CSV_DATA = "vanCensus.csv"
sc_x = StandardScaler()
sc_y = StandardScaler()

# Show all columns.
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)

df = pd.read_csv(PATH2 + CSV_DATA)

# Remove columns with a sum of 0.0
col_name = list(df.columns)
for i in range(0, len(col_name)):
    if df[col_name[i]].sum() == 0.0:
        del df[col_name[i]]

# Remove columns with a sum < 200.0
col_name = list(df.columns)
for i in range(0, len(col_name)):
    if df[col_name[i]].sum() <= 200.0:
        del df[col_name[i]]

# Imputing null values
imputer = KNNImputer(n_neighbors=10)
df = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)

# Get X and y variables
columns = list(df.columns)
y = df["total_crimes_2016"]
X = df[columns]
del X["Dissemination Number"]
del X["total_crimes_2016"]

columns2 = []
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
# feature selection
```

```
X_train_fs, X_test_fs, fs = select_features(X_train, y_train, X_test, 'all')

# what are scores for the features
for i in range(len(fs.scores_)):
    print('Feature %d: %f' % (i, fs.scores_[i]))
    if fs.scores_[i] > 20.0:
        columns2.append(list(X.columns)[i])

# plot the scores
pyplot.bar([i for i in range(len(fs.scores_))], fs.scores_)
pyplot.show()

X = X[columns2]
print(columns2)

# MODEL 1
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

X_train = sc_x.fit_transform(X_train)
X_test = sc_x.transform(X_test)

y_train = sc_y.fit_transform(np.array(y_train).reshape(-1, 1))

model = sm.OLS(y_train, X_train, hasconst=True).fit()
unscaledPredictions = model.predict(X_test)  # make the predictions by the model
yhat = sc_y.inverse_transform(np.array(unscaledPredictions).reshape(-1, 1))
rmse = np.sqrt(mean_squared_error(y_test, yhat))
print(model.summary())
print(rmse)

# Choose the variables with p-value < 0.05
columns3 = []
for i in range(0, len(model.pvalues)):
    if model.pvalues[i] <= 0.05:
        columns3.append(columns2[i])
print(columns3)

# MODEL 2
X = X[columns3]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

X_train = sc_x.fit_transform(X_train)
X_test = sc_x.transform(X_test)

y_train = sc_y.fit_transform(np.array(y_train).reshape(-1, 1))

model = sm.OLS(y_train, X_train, hasconst=True).fit()
unscaledPredictions = model.predict(X_test)  # make the predictions by the model
yhat = sc_y.inverse_transform(np.array(unscaledPredictions).reshape(-1, 1))
rmse = np.sqrt(mean_squared_error(y_test, yhat))
print(model.summary())
print(rmse)

# Choose the variables with p-value < 0.05
columns4 = []
for i in range(0, len(model.pvalues)):
    if model.pvalues[i] <= 0.05:
        columns4.append(columns3[i])

print(columns4)
```

```
# FINAL MODEL
X = X[columns4]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

X_train = sc_x.fit_transform(X_train)
X_test = sc_x.transform(X_test)

y_train = sc_y.fit_transform(np.array(y_train).reshape(-1, 1))

model = sm.OLS(y_train, X_train, hasconst=True).fit()
unscaledPredictions = model.predict(X_test)  # make the predictions by the model
yhat = sc_y.inverse_transform(np.array(unscaledPredictions).reshape(-1, 1))
rmse = np.sqrt(mean_squared_error(y_test, yhat))
print(model.summary())
print(rmse)
```

The code below creates different linear regression models, using scaling, stacking, K-Fold validation, and a final model with scaling and K-fold.

In [ ]:

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from sklearn import metrics
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.impute import KNNImputer
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler, StandardScaler, MinMaxScaler
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor


def getUnfitModels():
    models = list()
    models.append(ElasticNet())
    models.append(SVR(gamma='scale'))
    models.append(DecisionTreeRegressor())
    models.append(AdaBoostRegressor())
    models.append(RandomForestRegressor(n_estimators=10))
    models.append(ExtraTreesRegressor(n_estimators=10))
    return models


def evaluateModel(y_test, predictions, model):
    mse = mean_squared_error(y_test, predictions)
    rmse = round(np.sqrt(mse), 3)
    print(" RMSE:" + str(rmse) + " " + model.__class__.__name__)


def fitBaseModels(X_train, y_train, X_test, models):
    dfPredictions = pd.DataFrame()

    # Fit base model and store its predictions in dataframe.
```

```
    for i in range(0, len(models)):
        models[i].fit(X_train, y_train)
        predictions = models[i].predict(X_test)
        colName = str(i)
        # Add base model predictions to column of data frame.
        dfPredictions[colName] = predictions
    return dfPredictions, models


def fitStackedModel(X, y):
    model = LinearRegression()
    model.fit(X, y)
    return model


PATH2 = "C:\\datasets\\crime\\"
CSV_DATA = "vanCensus.csv"
sc_x = StandardScaler()
sc_y = StandardScaler()
count = 0

# Show all columns.
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)

df = pd.read_csv(PATH2 + CSV_DATA)

# Imputing null values
imputer = KNNImputer(n_neighbors=10)
df = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)

# Get X and y variables
columns = list(df.columns)
y = df["total_crimes_2016"]
X = df[columns]
del X["Dissemination Number"]
del X["total_crimes_2016"]

columns = ['Multiple Aboriginal and non-Aboriginal ancestries', 'No bedrooms', '1991 to 2
X = df[columns]
X = sm.add_constant(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

#########################################################################################
# Original Model
#########################################################################################
model = sm.OLS(y_train, X_train, hasconst=True).fit()
predictions = model.predict(X_test)  # make the predictions by the model
rmse = np.sqrt(mean_squared_error(y_test, predictions))
print(model.summary())
print(rmse)

#########################################################################################
# Model with scaling
#########################################################################################
scalers = [RobustScaler(), StandardScaler(), MinMaxScaler()]
scaler_name = ["Robust", "Standard", "MinMax"]
r2 = []
adj_r2 = []
aic = []
```

```
bic = []
rmse_list = []

for s in range(0, len(scalers)):
    sc_x = scalers[s]
    sc_y = scalers[s]

    X = sm.add_constant(X)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_stat

    X_train = sc_x.fit_transform(X_train)
    X_test = sc_x.transform(X_test)
    y_train = sc_y.fit_transform(np.array(y_train).reshape(-1, 1))

    model = sm.OLS(y_train, X_train).fit()
    unscaledPredictions = model.predict(X_test)  # make the predictions by the model
    yhat = sc_y.inverse_transform(np.array(unscaledPredictions).reshape(-1, 1))
    rmse = np.sqrt(mean_squared_error(y_test, yhat))

    r2.append(model.rsquared)
    adj_r2.append(model.rsquared_adj)
    aic.append(model.aic)
    bic.append(model.bic)
    rmse_list.append(rmse)
    print(model.summary())
    print(rmse)

d = {'Scaler': scaler_name, 'R2': r2, 'R2_Adj': adj_r2, 'AIC': aic, 'BIC': bic, 'rmse': r
scaler_stats = pd.DataFrame(data=d)
print(scaler_stats)

##############################################################################################
# Model with Stacking
##############################################################################################

# Split data into train, test and validation sets.
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.70)
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.50)

# Get base models.
unfitModels = getUnfitModels()

# Fit base and stacked models.
dfPredictions, models = fitBaseModels(X_train, y_train, X_test, unfitModels)
stackedModel = fitStackedModel(dfPredictions, y_test)

# Evaluate base models with validation data.
print("\n** Evaluate Base Models **")
dfValidationPredictions = pd.DataFrame()
for i in range(0, len(models)):
    predictions = models[i].predict(X_val)
    colName = str(i)
    dfValidationPredictions[colName] = predictions
    evaluateModel(y_val, predictions, models[i])

# Evaluate stacked model with validation data.
stackedPredictions = stackedModel.predict(dfValidationPredictions)
print("\n** Evaluate Stacked Model **")
evaluateModel(y_val, stackedPredictions, stackedModel)
```

```
#########################################################################################
# Model with K-Fold Validation
#########################################################################################
# prepare cross validation with three folds.
kfold = KFold(n_splits=8, shuffle=True)
rmseList = []
bicList = []
rsquareLst = []
count = 1

for train_index, test_index in kfold.split(X):
    X_train = X.loc[X.index.isin(train_index)]
    X_test = X.loc[X.index.isin(test_index)]
    y_train = y.loc[y.index.isin(train_index)]
    y_test = y.loc[y.index.isin(test_index)]

    # Perform linear regression.
    model = sm.OLS(y_train, X_train).fit()
    print(model.summary())

    y_pred = model.predict(X_test)  # make the predictions by the model
    mse = metrics.mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    rmseList.append(rmse)
    bic = model.bic
    bicList.append(bic)
    rsqr = model.rsquared
    rsquareLst.append(rsqr)

    print("\n***K-fold: " + str(count))
    print("RMSE:     " + str(rmse))
    print("BIC:      " + str(bic))
    print("R^2:      " + str(rsqr))

    count += 1

# Show averages of scores over multiple runs.
print("*********************************************")
print("\nScores for all folds:")
print("*********************************************")
print("RMSE Average :   " + str(np.mean(rmseList)))
print("RMSE SD:         " + str(np.std(rmseList)))
print("BIC Average :    " + str(np.mean(bicList)))
print("BIC SD:          " + str(np.std(bicList)))
print("RSQ Average :    " + str(np.mean(rsquareLst)))
print("RSQ SD:          " + str(np.std(rsquareLst)))

#########################################################################################
# Final Model
#########################################################################################
scalers = RobustScaler()
scaler_name = ["Robust", "Standard", "MinMax"]

sc_x = scalers
sc_y = scalers

X = sm.add_constant(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

X_train = sc_x.fit_transform(X_train)
```

```
X_test = sc_x.transform(X_test)
y_train = sc_y.fit_transform(np.array(y_train).reshape(-1, 1))

# prepare cross validation with three folds.
kfold = KFold(n_splits=8, shuffle=True)
rmseList = []
bicList = []
rsquareLst = []
count = 1

for train_index, test_index in kfold.split(X):
    X_train = X.loc[X.index.isin(train_index)]
    X_test = X.loc[X.index.isin(test_index)]
    y_train = y.loc[y.index.isin(train_index)]
    y_test = y.loc[y.index.isin(test_index)]

    # Perform linear regression.
    model = sm.OLS(y_train, X_train).fit()
    print(model.summary())

    y_pred = model.predict(X_test)  # make the predictions by the model
    mse = metrics.mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    rmseList.append(rmse)
    bic = model.bic
    bicList.append(bic)
    rsqr = model.rsquared
    rsquareLst.append(rsqr)

    print("\n***K-fold: " + str(count))
    print("RMSE:     " + str(rmse))
    print("BIC:      " + str(bic))
    print("R^2:      " + str(rsqr))

    count += 1

# Show averages of scores over multiple runs.
print("*********************************************")
print("\nScores for all folds:")
print("*********************************************")
print("RMSE Average :   " + str(np.mean(rmseList)))
print("RMSE SD:         " + str(np.std(rmseList)))
print("BIC Average :    " + str(np.mean(bicList)))
print("BIC SD:          " + str(np.std(bicList)))
print("RSQ Average :    " + str(np.mean(rsquareLst)))
print("RSQ SD:          " + str(np.std(rsquareLst)))
```