My Learning Journey

Introductory Data Science

Cody Schellenberger


AMERICAN INTERNATIONA HOOL - RIYADH
First Day
of School
2017-2018

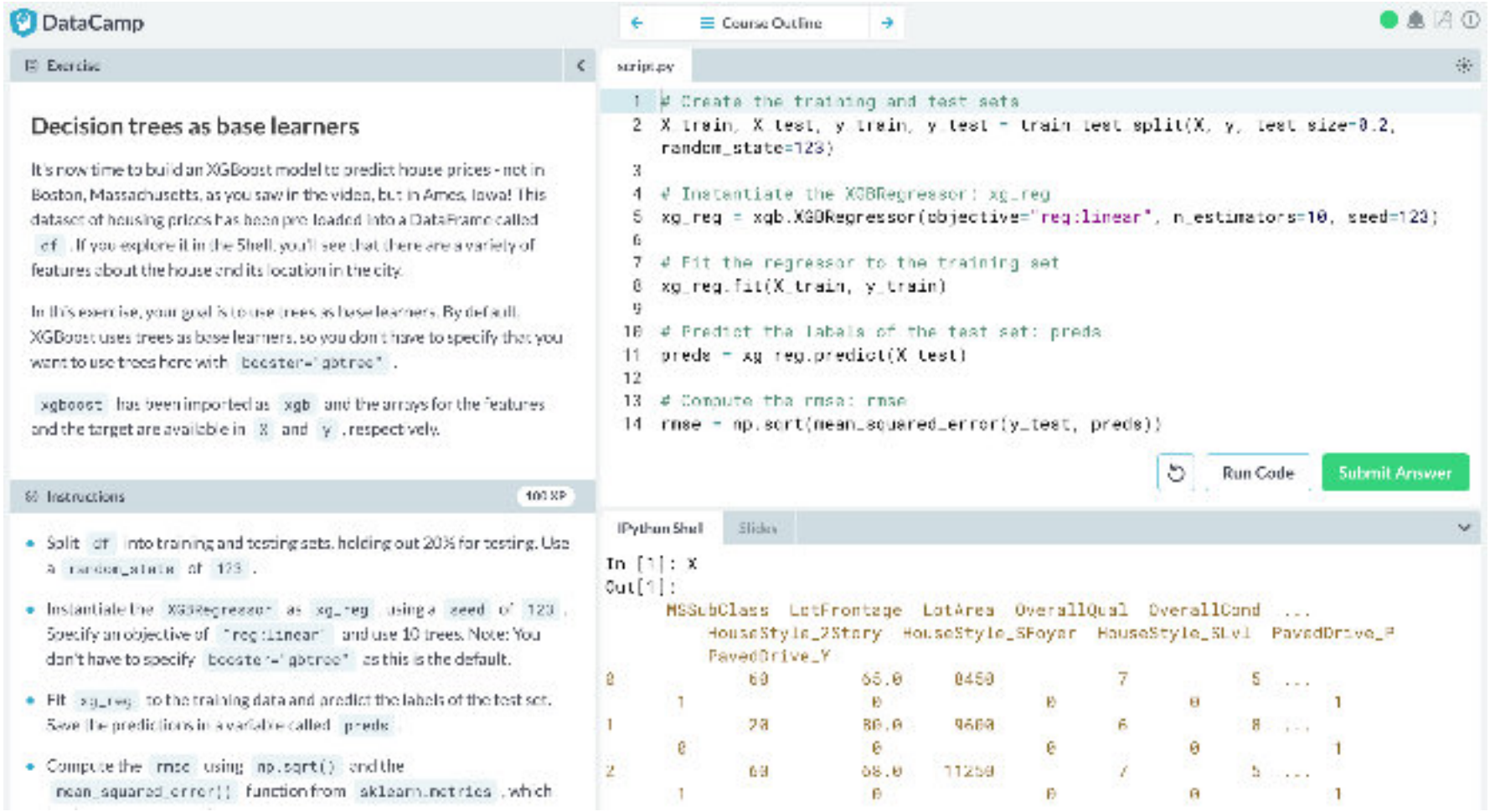
Al Bustan Village


Springboard


HARVARD
School of Engineering
and Applied Sciences


Khan Academy


DataCamp


DataCamp
Course Outline
Exercise
Decision trees as base learners
It's now time to build an XGBoost model to predict house prices - not in Boston, Massachusetts, as you saw in the video, but in Ames, Iowa! This dataset of housing prices has been pre-loaded into a DataFrame called df. If you explore it in the Shell, you'll see that there are a variety of features about the house and its location in the city.
In this exercise, your goal is to use trees as base learners. By default, XGBoost uses trees as base learners, so you don't have to specify that you want to use trees here with booster="gbtree".
xgboost has been imported as xgb and the arrays for the features and the target are available in X and y, respectively.
Instructions
100 XP
Split df into training and testing sets, holding out 20% for testing. Use a random_state of 123.
Instantiate the XGBRegressor as xg_reg using a seed of 123. Specify an objective of "reg:linear" and use 10 trees. Note: You don't have to specify booster="gbtree" as this is the default.
Fit xg_reg to the training data and predict the labels of the test set. Save the predictions in a variable called preds.
Compute the rmse using np.sqrt() and the mean_squared_error() function from sklearn.metrics, which
script.py
Create the training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
# Instantiate the XGBRegressor: xg_reg
xg_reg = xgb.XGBRegressor(objective="reg:linear", n_estimators=10, seed=123)
# Fit the regressor to the training set
xg_reg.fit(X_train, y_train)
# Predict the labels of the test set: preds
preds = xg_reg.predict(X_test)
# Compute the rmse: rmse
rmse = np.sort(mean_squared_error(y_test, preds))
Run Code
Submit Answer
IPython Shell
Slides
In [1]: X
Out[1]:
MSSubClass LotFrontage LotArea OverallQual OverallCond ...
HouseStyle_2Story HouseStyle_SFoyer HouseStyle_SLvl PavedDrive_P
PavedDrive_Y
0 60 65.0 8450 7 5 ...
1 0 0 0 1
1 20 80.0 9600 6 8 ...
0 0 0 0 1
2 60 68.0 11250 7 5 ...
1 0 0 0 1

jupyter My Learning Journey Last Checkpoint: 4 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Run Code

In [1]:
```
import pandas as pd
```

In [2]:
```
housing = pd.read_csv("ames_unprocessed_data.csv")
housing.head()
```

Out[2]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Neighborhood	BldgType	HouseStyle	Overal
0	60	RL	65.0	8450	CollgCr	1Fam	2Story	
1	20	RL	80.0	9600	Veenker	1Fam	1Story	

```
In [2]: X.columns
Out[2]:
Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
    'YearBuilt', 'Remodeled', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath',
    'FullBath', 'HalfBath', 'BedroomAbvGr',
       'Fireplaces', 'GarageArea', 'MSZoning_FV', 'MSZoning_RH', 'MSZoning_RL',
          'MSZoning_RM', 'Neighborhood_Blueste', 'Neighborhood_BrDale',
          'Neighborhood_BrkSide', 'Neighborhood_ClearCr',
       'Neighborhood_CollgCr', 'Neighborhood_Crawfor', 'Neighborhood_Edwards',
```

```
In [3]: len(X.columns)
Out[3]: 56
```

```
In [25]: len(housing.columns)
Out[25]: 21
```

```
In [ ]: pd.get_dummies()
```

```
Signature: pd.get_dummies(data, prefix=None, prefix_sep='_', dummy_na=Fa
columns=None, sparse=False, drop_first=False, dtype=None)
Docstring:
Convert categorical variable into dummy/indicator variables
```

```
In [4]: housing_encode = pd.get_dummies(housing)
```

```
In [5]: housing_encode.head()
```

Out[5]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond
0	60	65.0	8450	7	5
1	20	80.0	9600	6	8

```
In [6]: len(housing_encode.columns)

Out[6]: 62
```

```
In [7]: datacamp_X = ['MSSubClass', 'LotFrontage', 'LotArea', 'Ove
                      'Fireplaces', 'GarageArea', 'MSZoning_FV', 'MSZonir
                      'Neighborhood_CollgCr', 'Neighborhood_Crawfor', 'Ne
                      'Neighborhood_NPkVill', 'Neighborhood_NWAmes', 'Nei
                      'Neighborhood_Somerst', 'Neighborhood_StoneBr', 'Ne
                      'HouseStyle_1Story', 'HouseStyle_2.5Fin', 'HouseSty

In [9]: dc_set_X = set(datacamp_X)
        set_columns = set(housing_encode.columns)

In [13]: set_columns.difference(dc_set_X)

Out[13]: {'BldgType_1Fam',
          'HouseStyle_1.5Fin',
          'MSZoning_C (all)',
          'Neighborhood_Blmngtn',
          'PavedDrive_N',
          'SalePrice'}
```

In [14]:
```
housing_encode = pd.get_dummies(housing, drop_first=True)
```

```
In [19]: set_columns.difference(dc_set_X)
Out[19]: {'SalePrice'}
```

In [22]:
```
X = housing_encode.drop('SalePrice', axis=1)
len(X.columns)
```

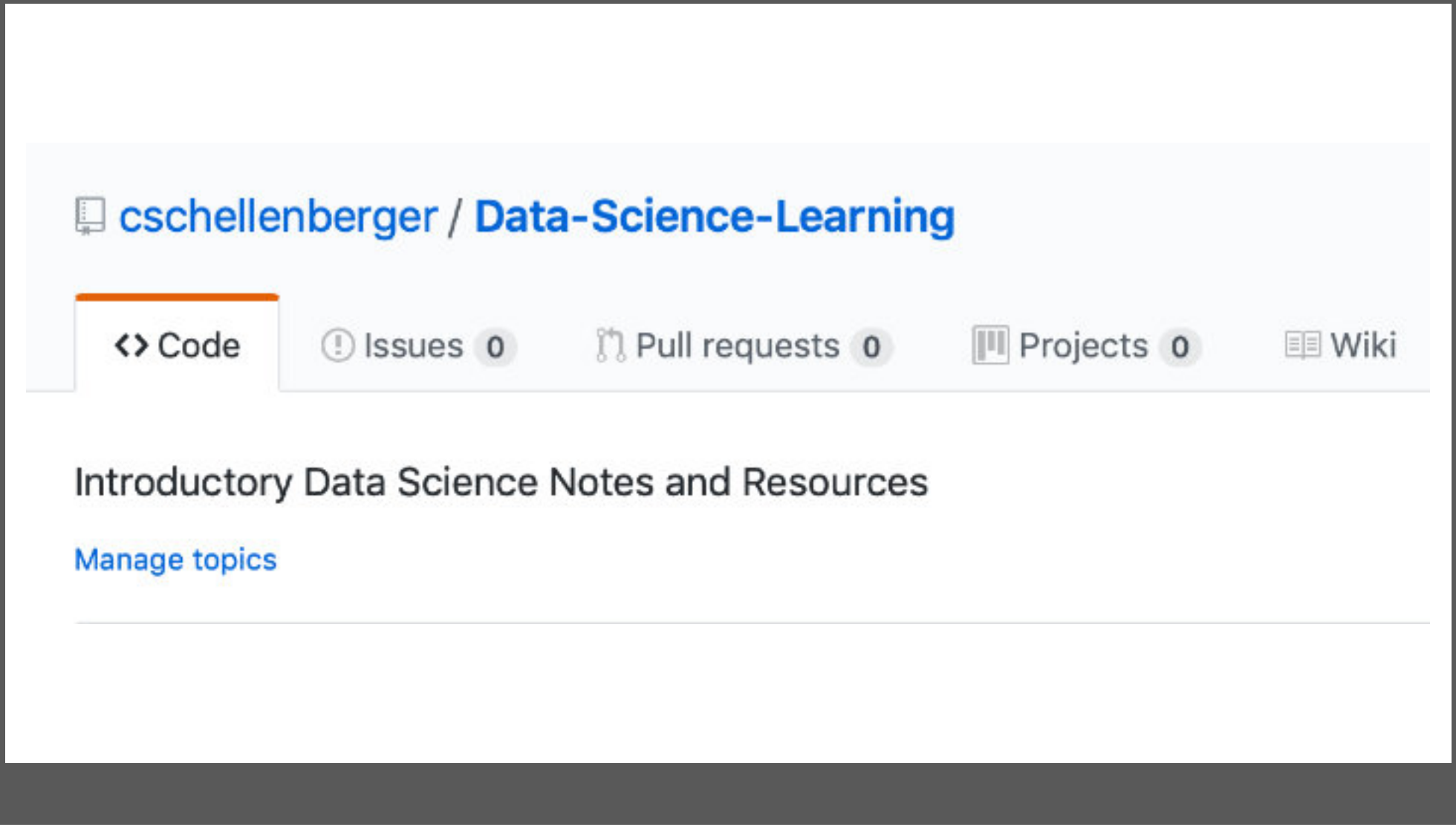
Out[22]:
```
56
```

In [24]:
```
y = housing_encode['SalePrice']
y.head(2)
```

Out[24]:
```
0    208500
1    181500
Name: SalePrice, dtype: int64
```

script.py

```
# Create the training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=123)

```


cschellenberger / Data-Science-Learning
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Introductory Data Science Notes and Resources
Manage topics

Data Science

Exploratory, training and resource code for many Data Science usage cases in python 3.6

Resources

Learning Sequence	Title	Link	Notes
1	Data Types for Data Science	iPython Notebook	General overview of datatypes in Python
2	Unix Shell Commands for Data Science	DataCamp Course	Fundamentals of using unix commands
3	Git Introduction	DataCamp Course	General commands for commiting, staging, deleting, and working with history

1 file 0 forks 0 comments 0 stars

cschellenberger / full_column_output.py

Created Dec 3, 2018

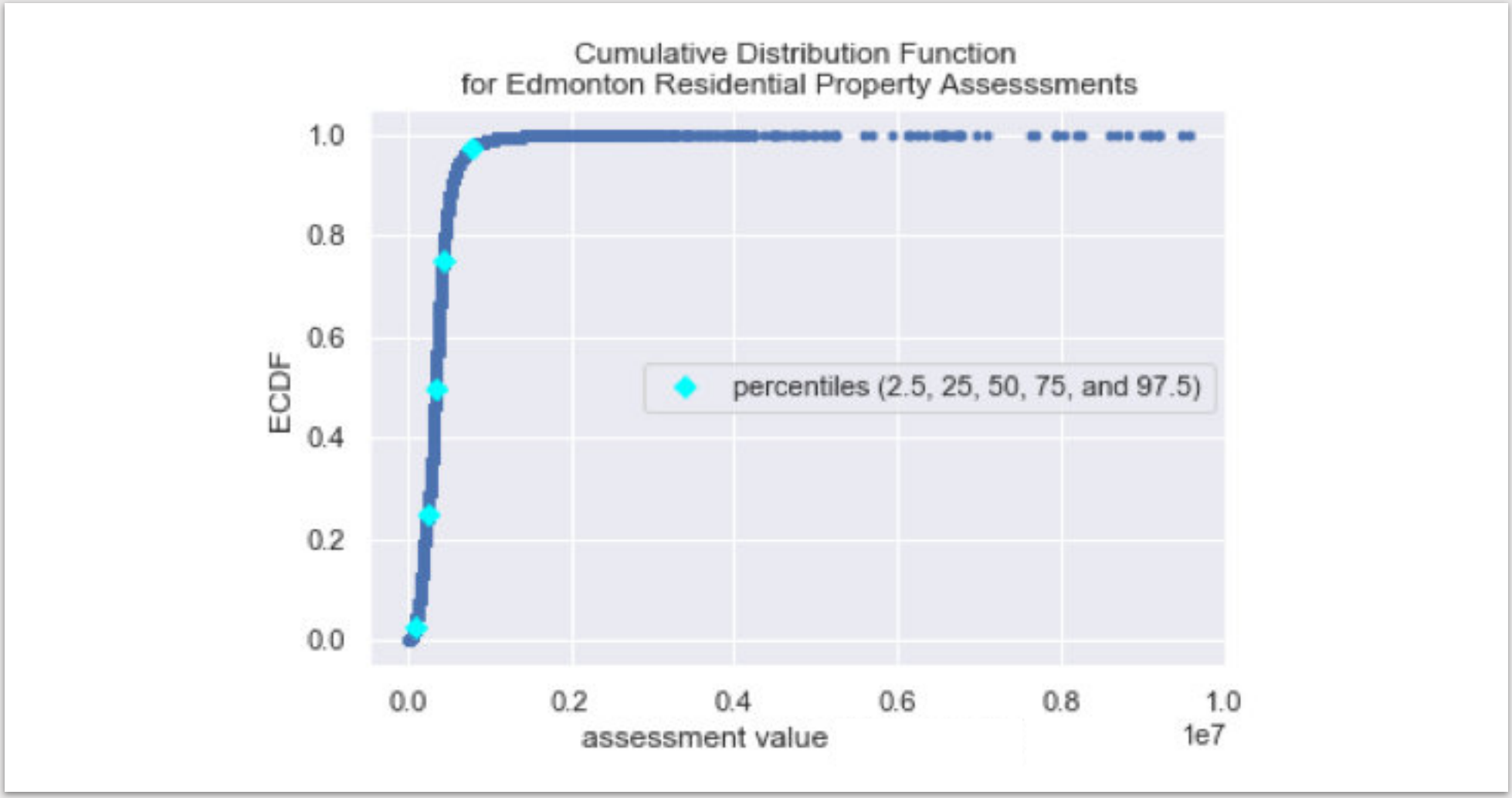
Displaying the full range of columns for pandas output with DataFrames > 15 features

```
1  import pandas as pd
2
3  # This will set the output display to 7 columns max
4  pd.set_option('display.max_columns', 7)
5
6  # This will force the display of any number of columns
7  pd.set_option('display.max_columns', None)
8
9  # This will change the default 80 pixel width to 200.
10 pd.set_option('display.width', 200)
```

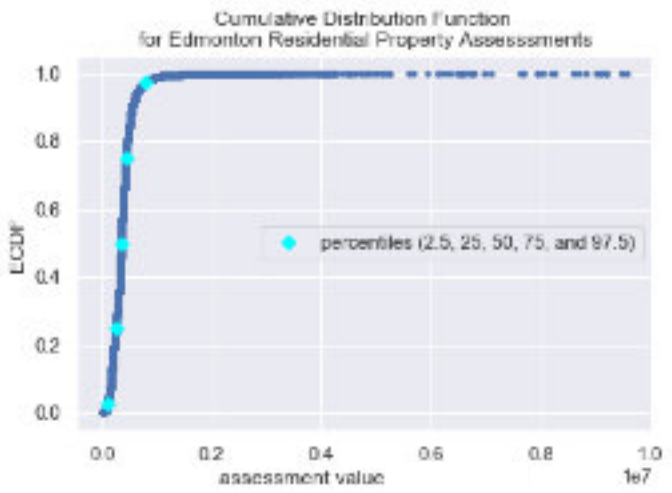

HackerRank

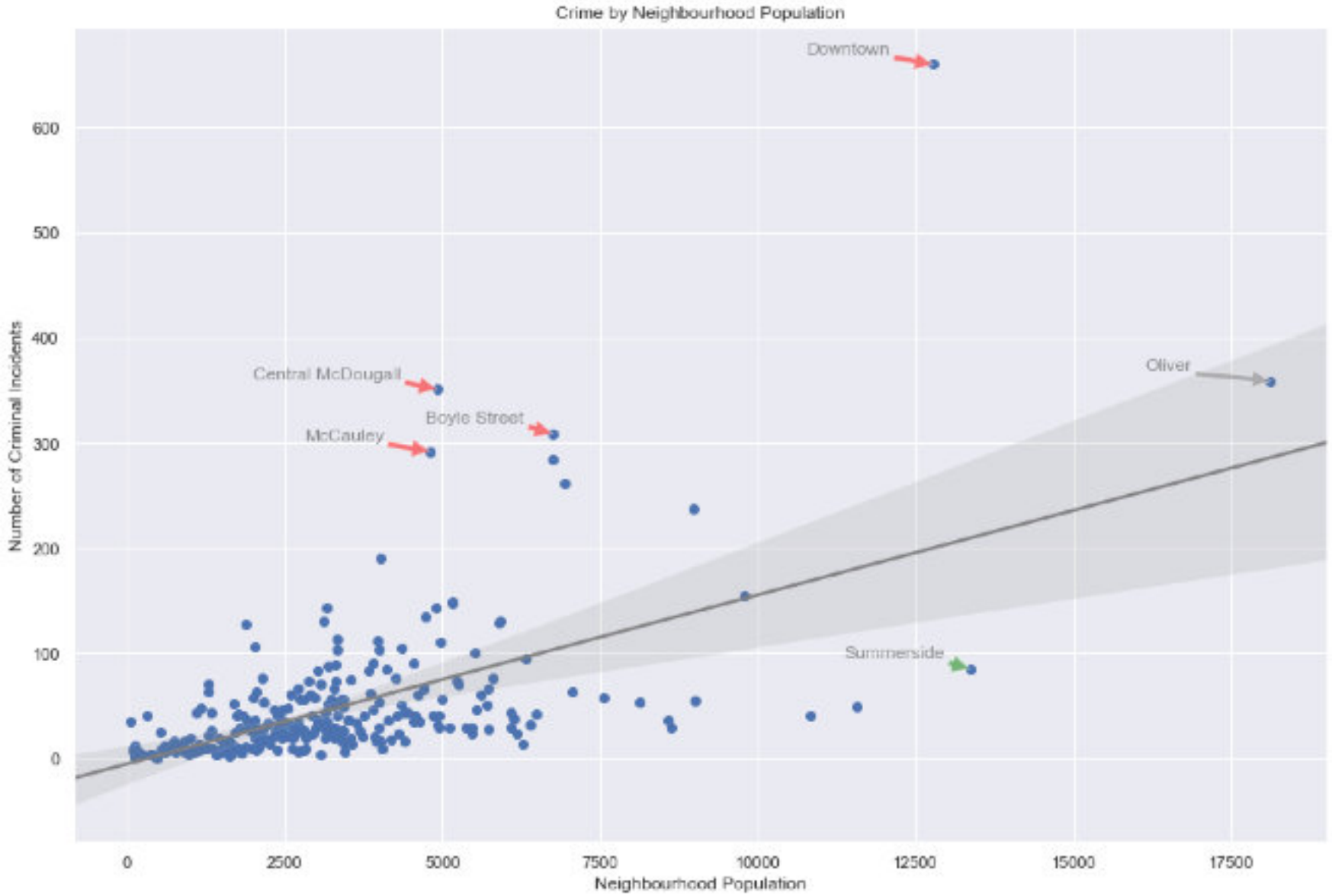

kaggle


stackoverflow


Cumulative Distribution Function
for Edmonton Residential Property Assesssments
ECDF
1.0
0.8
0.6
0.4
0.2
0.0
percentiles (2.5, 25, 50, 75, and 97.5)
0.0
0.2
0.4
0.6
0.8
1.0
assessment value
1e7

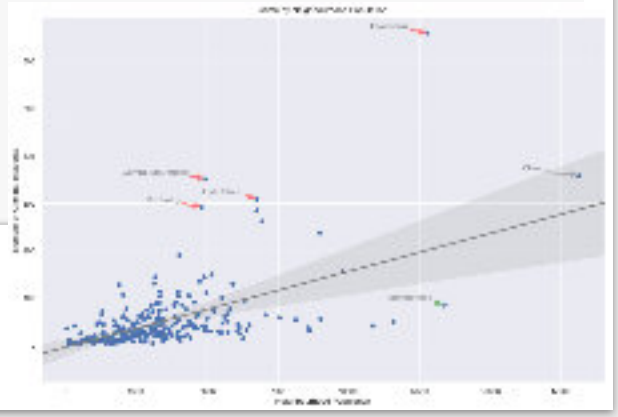
```
%run stats_func.py

value = property_assess['value']

# Specify array of percentiles: percentiles
percentiles = np.array([2.5, 25, 50, 75, 97.5])

# Compute percentiles
ptiles = np.percentile(value, percentiles)

x_res, y_res = ecdf(value)
_ = plt.plot(x_res, y_res, marker='.', linestyle='none')
_ = plt.xlabel('assessment value (Million CAD)')
_ = plt.ylabel('ECDF')
_ = plt.title('Cumulative Distribution Function \nfor Edmonton Residential Property Assesssments')

_ = plt.plot(ptiles, percentiles/100, marker='D', color='cyan',
             linestyle='none', label='percentiles (2.5, 25, 50, 75, and 97.5)')
_ = plt.legend(loc='center right')

plt.show()
```

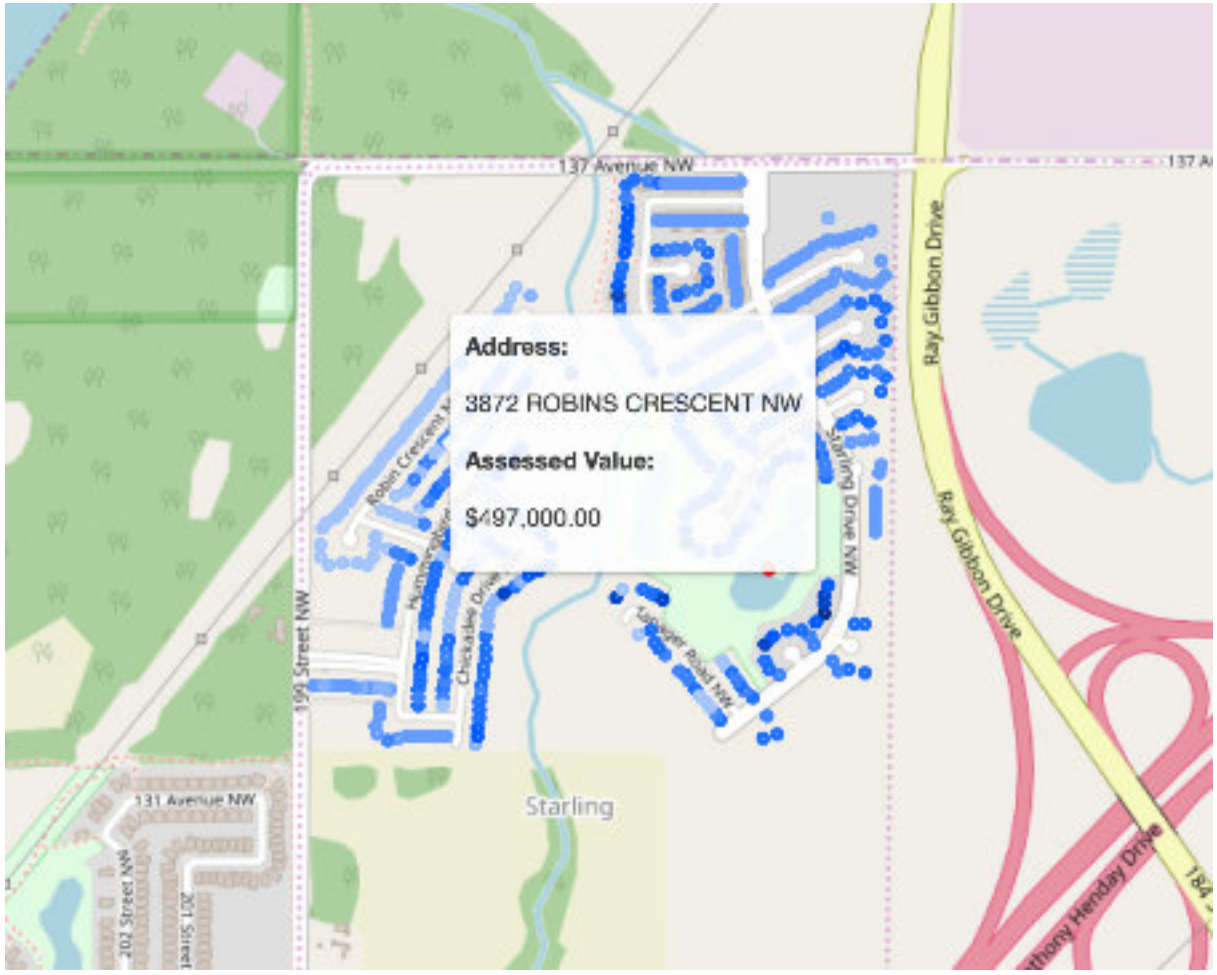



Crime by Neighbourhood Population
Downtown
Central McDougall
Boyle Street
McCauley
Oliver
Summerside
Number of Criminal Incidents
0
100
200
300
400
500
600
0
2500
5000
7500
10000
12500
15000
17500
Neighbourhood Population

```
pop = nb_pop_assess['population']
inc = nb_pop_assess['num_incidents']

plt.figure(figsize=(15,10))

plt.scatter(pop, inc)

# Add arrow annotation
plt.annotate('Downtown', xy=(12768, 660), xytext=(10768, 660+10),
             arrowprops=dict(facecolor='red', alpha=0.5), alpha=0.5)
plt.annotate('Oliver', xy=(18123, 359), xytext=(16123, 359+10),
             arrowprops=dict(facecolor='gray', alpha=0.5), alpha=0.5)
plt.annotate('Central McDougall', xy=(4911, 351), xytext=(2000, 351+10),
             arrowprops=dict(facecolor='red', alpha=0.5), alpha=0.5)
plt.annotate('Boyle Street', xy=(6740, 309), xytext=(4740, 309+10),
             arrowprops=dict(facecolor='red', alpha=0.5), alpha=0.5)
plt.annotate('McCauley', xy=(4799, 292), xytext=(2799, 292+10),
             arrowprops=dict(facecolor='red', alpha=0.5), alpha=0.5)
plt.annotate('Summerside', xy=(13360, 85), xytext=(11360, 85+10),
             arrowprops=dict(facecolor='green', alpha=0.5), alpha=0.5)
```

```
sns.regplot(x='population', y='num_incidents', data=nb_pop_assess,
            color='grey', scatter=None, order=1)

# Add axis labels and title
plt.xlabel('Neighbourhood Population')
plt.ylabel('Number of Criminal Incidents')
plt.title('Crime by Neighbourhood Population')

plt.savefig('crime_population.png')

plt.show()
```

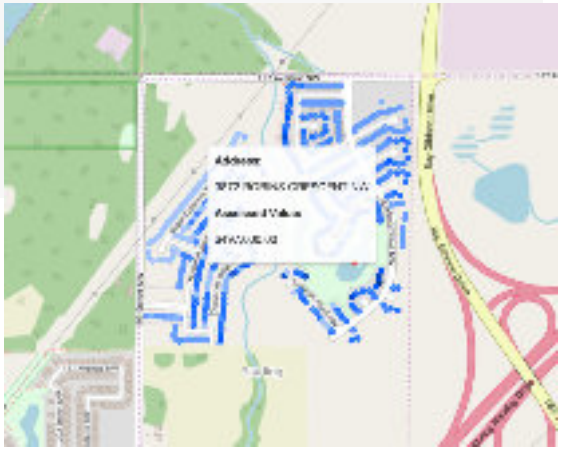



Address:
3872 ROBINS CRESCENT NW
Assessed Value:
$497,000.00
137 Avenue NW
Ray Gibbon Drive
199 Street NW
Starling Drive NW
Starling
131 Avenue NW
202 Street NW
201 Street

In [5]:

```
# define a function that will map the results for a particular neighbourhood
def nb_map(neighbourhood):
    ''' This function takes a neighbourhood as input and returns a folium map
    with a colour map associated with the property assessment value and a
    on-hover tooltip with assessment details.
    '''

    my_nb = property_assess[property_assess['nb'] == str(neighbourhood).upper()]

    m2 = folium.Map(location=[my_nb['lat'].mean(), my_nb['long'].mean()], zoom_start=15)

    def add_marker(row):
        if row['value'] < 100000:
            color = '#bfd7ff'
```

```
    row_cur = '${:0,.2f}'.format(row['value'])
    tooltip = '<p><b>Address:</b></p><p>' + str(row['number']) + " " + row['street'] \
              + '</p>' \
              + '<p><b>Assessed Value:</b></p><p>' + row_cur + '</p>'
    marker = folium.CircleMarker([row['lat'], row['long']], radius=2,
                                 color=color, tooltip=tooltip)
    marker.add_to(m2)

my_nb.apply(add_marker, axis=1)

return m2
```



Thank you

github.com/cschellenberger

linkedin.com/in/codyschellenberger

cody@schellenberger.me