NOTEBOOK 2: NEIGHBORHOOD CLUSTERS

In [1]:
```
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
from geopy import distance
def dist_from_center(lat, lon):
    buenos_aires_center = (-34.603722, -58.381592)
    return distance.distance(buenos_aires_center, (lat, lon)).km
```

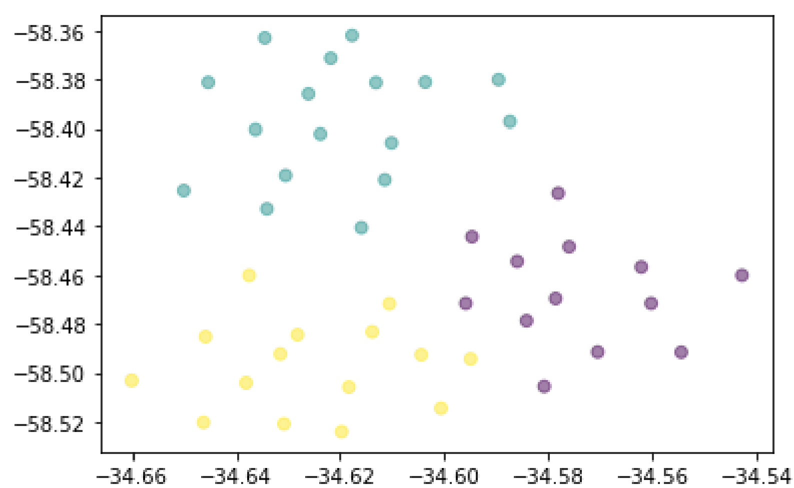
In [3]:
```
coords = {
    'Agronomía': (-34.5950, -58.4943),
    'Almagro': (-34.6114, -58.4210),
    'Balvanera': (-34.6101, -58.4059),
    'Barracas': (-34.6454, -58.3813),
    'Belgrano': (-34.5621, -58.4567),
    'Boedo': (-34.6305, -58.4192),
    'Caballito': (-34.6159, -58.4406),
    'Chacarita': (-34.5860, -58.4544),
    'Coghlan': (-34.5602, -58.4716),
    'Colegiales': (-34.5760, -58.4484),
    'Constitución': (-34.6261, -58.3860),
    'Flores': (-34.6375, -58.4601),
    'Floresta': (-34.6282, -58.4844),
    'La Boca': (-34.6345, -58.3631),
    'La Paternal': (-34.5959, -58.4716),
    'Liniers': (-34.6463, -58.5202),
    'Mataderos': (-34.6601, -58.5031),
    'Monserrat': (-34.6131, -58.3814),
    'Monte Castro': (-34.6183, -58.5057),
    'Nueva Pompeya': (-34.6501, -58.4254),
    'Núñez': (-34.5428, -58.4601),
    'Palermo': (-34.5781, -58.4265),
    'Parque Avellaneda': (-34.6459, -58.4852),
    'Parque Chacabuco': (-34.6341, -58.4329),
    'Parque Chas': (-34.5842, -58.4787),
    'Parque Patricios': (-34.6363, -58.4005),
    'Puerto Madero': (-34.6177, -58.3621),
    'Recoleta': (-34.5874, -58.3973),
    'Retiro': (-34.5896, -58.3802),
    'Saavedra': (-34.5545, -58.4916),
    'San Cristóbal': (-34.6238, -58.4023),
    'San Nicolás': (-34.6037, -58.3812),
    'San Telmo': (-34.6218, -58.3714),
    'Versalles': (-34.6308, -58.5208),
    'Villa Crespo': (-34.5947, -58.4443),
    'Villa Devoto': (-34.6007, -58.5144),
    'Villa General Mitre': (-34.6105, -58.4717),
    'Villa Luro': (-34.6381, -58.5040),
    'Villa Ortúzar': (-34.5786, -58.4696),
    'Villa Pueyrredón': (-34.5808, -58.5054),
    'Villa Real': (-34.6197, -58.5240),
    'Villa Santa Rita': (-34.6138, -58.4832),
    'Villa Urquiza': (-34.5705, -58.4915),
    'Villa del Parque': (-34.6045, -58.4926),
    'Vélez Sársfield': (-34.6315, -58.4923)
}
```

In [4]:
```
dists = {
    nhood: dist_from_center(*coords[nhood]) for nhood in coords
}
```

In [5]:
```
def out(action, bar, percent, done=False):
    done = f'\n' if done is True else ''
    sys.stdout.write('\r%s |%s| ' % (action.ljust(30), bar) + '{0:.0f}%'
.format(percent) + done)

def progress(i, total, action):
    i = i + 1
    ratio = i / total
    percent = 100 * ratio
    filled = int(round(20 * ratio))

    bar = '█' * filled + '-' * (20 - filled)

    done = (percent == 100)
    out(action, bar, percent, done)

    if i == total:
        sys.stdout.write('\n')

    sys.stdout.flush()
```

Clustering based on neighborhood

In [6]:
```
from sklearn.cluster import KMeans

X = pd.DataFrame.from_dict(coords, orient='index')

n_clusters = 3
neighborhood_kmeans = KMeans(n_clusters=n_clusters, random_state=0).fit(
X)

X_np = X.to_numpy()
plt.scatter(X_np[:,0], X_np[:,1], c=neighborhood_kmeans.labels_, alpha=
0.5)

plt.savefig('img/clusters/nhoods.pdf')
```



In [7]:
```
def cluster_from_nhood(nhood):
    c = coords[nhood]
    return neighborhood_kmeans.predict([list(c)]).item()
```

Import data

In [8]:
```
X_train = pd.read_csv('data/train.csv')
y_train = X_train['price']
X_test = pd.read_csv('data/test.csv')

X_train = X_train.drop(columns=['price'])

features = X_train.columns
labels = list(np.unique(y_train))

string_features = {
    'neighbourhood': ['Agronomía', 'Almagro', 'Balvanera', 'Barracas',
'Belgrano',
        'Boedo', 'Caballito', 'Chacarita', 'Coghlan', 'Colegiales',
        'Constitución', 'Flores', 'Floresta', 'La Boca', 'La Paternal',
        'Liniers', 'Mataderos', 'Monserrat', 'Monte Castro',
        'Nueva Pompeya', 'Núñez', 'Palermo', 'Parque Avellaneda',
        'Parque Chacabuco', 'Parque Chas', 'Parque Patricios',
        'Puerto Madero', 'Recoleta', 'Retiro', 'Saavedra', 'San Cristóba
l',
        'San Nicolás', 'San Telmo', 'Versalles', 'Villa Crespo',
        'Villa Devoto', 'Villa General Mitre', 'Villa Luro',
        'Villa Ortúzar', 'Villa Pueyrredón', 'Villa Real',
        'Villa Santa Rita', 'Villa Urquiza', 'Villa del Parque',
        'Vélez Sársfield'],
    'room_type': ['Entire home/apt', 'Hotel room', 'Private room', 'Shar
ed room'],
    'host_is_superhost': ['f', 't'],
    'bed_type': ['Airbed', 'Couch', 'Futon', 'Pull-out Sofa', 'Real Bed'
],
    'instant_bookable': ['f', 't'],
    'is_business_travel_ready': ['f'],
    'cancellation_policy': ['flexible', 'moderate', 'strict_14_with_grac
e_period', 'super_strict_30', 'super_strict_60'],
    'require_guest_profile_picture': ['f', 't'],
    'require_guest_phone_verification': ['f', 't'],
}
```

Add in distance from center of city

In [9]:
```
# set new "distance" column to be all 0s
X_train['distance'] = 0.0
X_test['distance'] = 0.0

# fill in the distances from the city center
for i in range(X_train.shape[0]):
    X_train.at[i, 'distance'] = dists[X_train.at[i, 'neighbourhood']]
for i in range(X_test.shape[0]):
    X_test.at[i, 'distance'] = dists[X_test.at[i, 'neighbourhood']]
```

save IDs

In [10]:
```
train_ids = X_train['id']
test_ids = X_test['id']
```

prediction saving function

In [11]:
```
def save_predictions(clf, filename):
    y_pred = clf.predict(X_test)
    assert(y_pred.shape[0] == 4149)

    y_pred = pd.DataFrame(data=y_pred, columns=['price'])
    y_pred['id'] = test_ids
    y_pred = y_pred.reindex(['id','price'], axis=1).astype(int)

    y_pred.to_csv(f'results/{filename}.csv', index=False)
```

Data Preprocessing

In [12]:
```
from datetime import date

def data2float(X):
    X = X.copy()

    # turn strings into integers
    for f in string_features:
        for i in range(X.shape[0]):
            X.at[i, f] = string_features[f].index(X.at[i, f])

    # turn dates into days relative to today
    today = date.today()
    for f in ['last_review', 'host_since']:
        for i in range(X.shape[0]):
            m, d, y = (int(x) for x in X.at[i, f].split('/'))
            X.at[i, f] = (today - date(y, m, d)).days

    return X.astype(float)
```

In [13]:
```
from sklearn.preprocessing import MinMaxScaler

def normalize_df(df):
    min_max_scaler = MinMaxScaler()
    return pd.DataFrame(min_max_scaler.fit_transform(df.values), columns
=df.columns)
```

In [14]:
```
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.ensemble import RandomForestClassifier


class HybridClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, max_depth):
        self.n_clusters = 3
        self.max_depth = max_depth
        self.cluster_models = None

    def _get_data_clusters(self, X, y):
        cluster_data = {i: () for i in range(self.n_clusters)}
        for c in range(self.n_clusters):
            idx = [i for i in range(len(X)) if cluster_from_nhood(X.at[i
, 'neighbourhood']) == c]
            X_c = normalize_df(data2float(X.loc[idx].reset_index(drop=Tr
ue)))
            y_c = y.loc[idx].reset_index(drop=True)
            cluster_data[c] = (X_c, y_c)
        return cluster_data

    def fit(self, X, y):
        # divide data into clusters based on neighborhood
        cluster_data = self._get_data_clusters(X, y)

        # fit a model for each cluster
        self.cluster_models = {i: None for i in range(self.n_clusters)}
        for c in range(self.n_clusters):
            X, y = cluster_data[c]
            clf = RandomForestClassifier(max_depth=self.max_depth, rando
m_state=0)
            clf.fit(X, y)
            self.cluster_models[c] = clf

    def predict(self, X):
        '''
        Assume X is **not** normalized or anything yet
        '''
        if self.cluster_models is None:
            raise AssertionError('You have to train the model first, du
h')

        pred_clusters = [cluster_from_nhood(n) for n in X['neighbourhoo
d']]
        X = normalize_df(data2float(X))

#         pred_y = [self.cluster_models[pred_clusters[i]].predict([X.loc
[i]]) for i in range(len(X))]
        pred_y = []
        for i in range(len(X)):
            x = X.loc[i]
            c = pred_clusters[i]
            pred_y.append(self.cluster_models[c].predict([x]))
        pred_y = np.array(pred_y).flatten()
        return pred_y
```

```
    def score(self, X, y):
        pred_y = self.predict(X)
        num_correct = sum([1 if pred_y[i] == y.loc[i] else 0 for i in ra
nge(len(X))])
        return num_correct / len(X)
```

In [15]:
```
# clf = HybridClassifier(max_depth=10)
# clf.fit(X_train, y_train)
```

In [16]:
```
# clusters_test = [cluster_from_nhood(n) for n in X_test['neighbourhoo
d']]
# X_test = normalize_df(data2float(X_test))
```

Cross-Validation

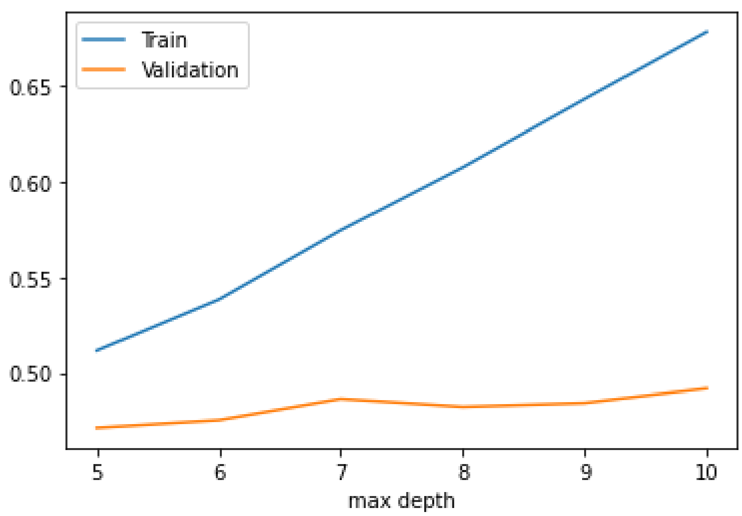
In [15]:
```
from sklearn.model_selection import KFold
from sklearn.base import clone

def cv(clf, X, y, k):
    X = X.copy()
    y = y.copy()
    base_clf = clone(clf)

    train_scores = []
    val_scores = []

    kf = KFold(n_splits=k)
    kf.get_n_splits(X)

    for train_index, val_index in kf.split(X):
        # split data into training and validation
        X_train, X_val = X.loc[train_index], X.loc[val_index]
        y_train, y_val = y.loc[train_index], y.loc[val_index]

        # reset indices for the data
        X_train = X_train.reset_index(drop=True)
        X_val = X_val.reset_index(drop=True)
        y_train = y_train.reset_index(drop=True)
        y_val = y_val.reset_index(drop=True)

        # train and score a classifier
        clf = clone(base_clf)
        clf.fit(X_train, y_train)
        train_scores.append(clf.score(X_train, y_train))
        val_scores.append(clf.score(X_val, y_val))

    train_mean = sum(train_scores) / len(train_scores)
    val_mean = sum(val_scores) / len(val_scores)
    return train_mean, val_mean
```

In [16]:
```
def vary_cv(clf_fn, prop, values, X, y, k):
    train_scores = []
    val_scores = []

    i = 0
    for v in values:
        progress(i, len(values), f'{prop}: {v}')
        i += 1

        clf = clf_fn(v)
        train_mean, val_mean = cv(clf, X, y, k=k)
        train_scores.append(train_mean)
        val_scores.append(val_mean)

    plt.plot(train_scores, label='Train')
    plt.plot(val_scores, label='Validation')
    plt.xticks(ticks=range(len(train_scores)), labels=values)
    plt.xlabel(prop)
    plt.legend()
    plt.show()

    return train_scores, val_scores
```

Training

In [17]:
```
depths = [5, 6, 7, 8, 9, 10]
hybrid_fn = lambda v: HybridClassifier(max_depth=v)
hybrid_train_scores, hybrid_val_scores = vary_cv(hybrid_fn, 'max depth', depths, X_train, y_train, k=3)
```

```
max depth: 10                          |██████████████████| 100%
```



In [ ]:
```
plt.plot(hybrid_train_scores, label='Train')
plt.plot(hybrid_val_scores, label='Validation')
plt.xticks(ticks=range(len(hybrid_train_scores)), labels=depths)
plt.xlabel('max depth')
plt.ylabel('Accuracy')
plt.title('Neighborhood Cluster Model')
plt.legend()
plt.savefig('img/training/nhood_removed.pdf')
```

In [21]:
```
clf = HybridClassifier(max_depth=10)
clf.fit(X_train, y_train)
save_predictions(clf, 'nhood_clusters_removed')
```