NOTEBOOK 1: MAIN

In [1]:

```
import sys
import csv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

# random seeds
np.random.seed(0)
```

Progress Bar

In [2]:

```
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()
```

Import Data

In [3]:
```
X_train = pd.read_csv('data/train.csv')
y_train = X_train['price']
X_test = pd.read_csv('data/test.csv')

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'],
}
```

Feature Engineering: Distance from city center

In [4]:
```
from geopy import distance
def dist_from_center(coords):
    buenos_aires_center = (-34.603722, -58.381592)
    return distance.distance(buenos_aires_center, coords).km

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)
}

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

In [5]:
```
# 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']]
```

Results Saving

In [6]:
```
train_ids = X_train['id']
test_ids = X_test['id']
```

In [7]:
```
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)
```

Normalize Data

In [8]:
```
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)

X_train = data2float(X_train)
X_test = data2float(X_test)
```

In [9]:
```
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)

X_train = normalize_df(X_train)
X_test = normalize_df(X_test)
```

Cross Validation code

In [10]:
```
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 [11]: 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
```

Data Inspection

Finding bad features

To start, we'll look at the distributions of the values of each feature. From this we can get a sense of which features are correlated with each price.

```
In [ ]: for f in features:
            sns.displot(data=X_train, x=f, hue='price', kind='kde', palette='vir
        idis', legend=False)
            plt.savefig(f'img/density/{f}.pdf')
            #### purple=1, yellow=4
```

```
In [13]: X_train = X_train.drop(columns=['price'])

         for f in ['id', 'is_business_travel_ready']:
             X_train = X_train.drop(columns=[f])
             X_test = X_test.drop(columns=[f])
```

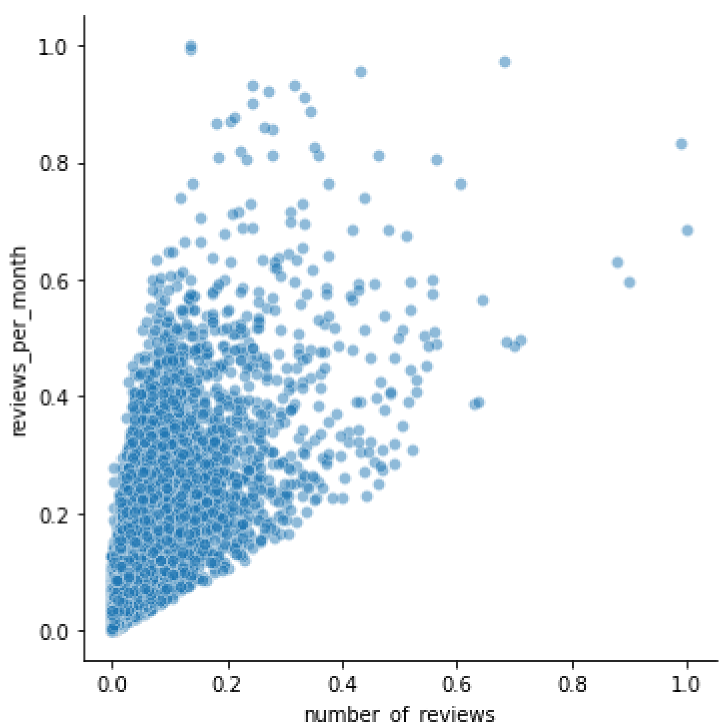
Remove outliers

In [14]:
```
# thresholds = [
#     'minimum_nights < 0.2',
#     'number_of_reviews < 0.4',
#     'last_review < 0.4',
#     'calculated_host_listings_count < 0.4',
#     'bathrooms < 0.4',
#     'bedrooms < 0.4',
#     'beds < 0.4',
#     'bed_type > 0.9',
#     'cleaning_fee < 0.4',
#     'guests_included < 0.4',
#     'extra_people < 0.2',
#     'maximum_nights < 0.1'
# ]

# def remove_outliers(X, y):
#     for t in thresholds:
#         X = X.query(t)
#     y = y.loc[X.index]
#     return X.reset_index(drop=True), y.reset_index(drop=True)

# X_train, y_train = remove_outliers(X_train, y_train)
```

Variable Correlation

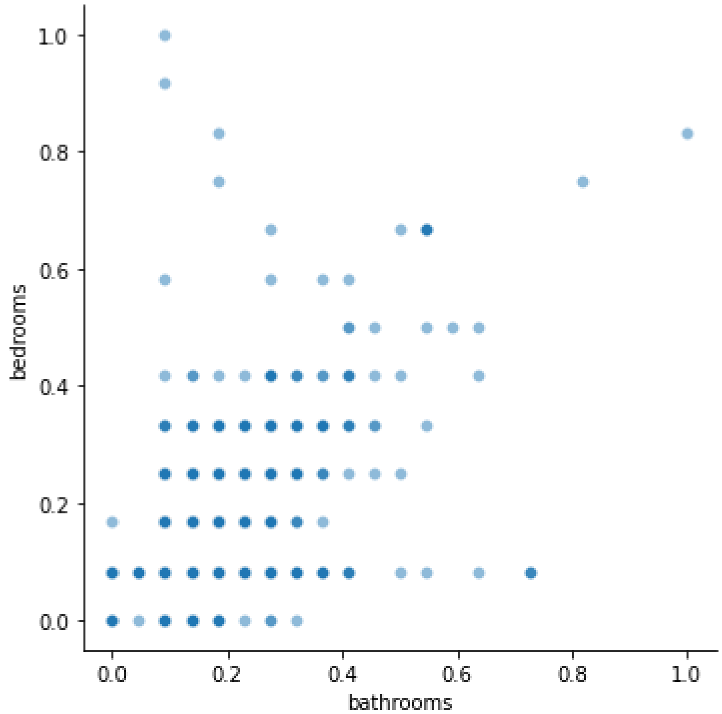
In [15]:
```
sns.relplot(x='number_of_reviews', y='reviews_per_month', data=X_train,
alpha=0.5)
# plt.savefig('img/corr/reviews.pdf')
```

Out[15]: <seaborn.axisgrid.FacetGrid at 0x1401fdcd0>

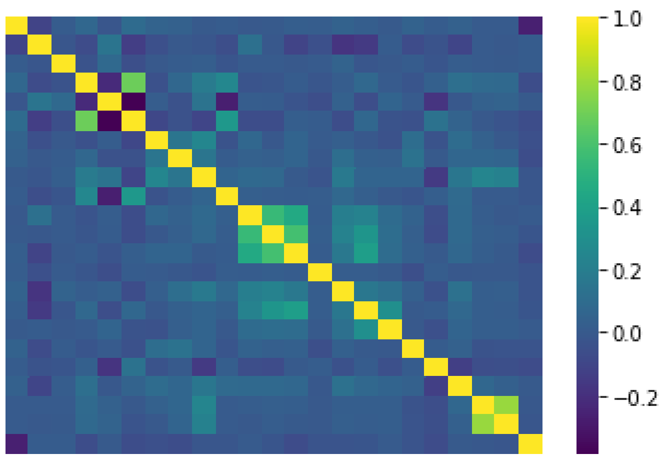


In [16]:
```
sns.relplot(x='bathrooms', y='bedrooms', data=X_train, alpha=0.5)
# plt.savefig('img/corr/reviews.pdf')
```

Out[16]: <seaborn.axisgrid.FacetGrid at 0x140280220>



In [17]:
```
corr_matrix = X_train.corr()
sns.heatmap(corr_matrix, cmap='viridis', xticklabels=False, yticklabels=
False)
# plt.show()
# plt.tight_layout()
# plt.gcf().subplots_adjust(bottom=0.3)
plt.savefig('img/corr/matrix.pdf')
```



Relationships

The more &'s, the stronger the correlation:

- reviews_per_month - number_of_reviews - last_review (&&&)
- bedrooms - bathrooms - beds (&&)
- guests_included - beds (&)
- host_is_superhost - reviews_per_month (&)
- require_guest_profile_picture - require_guest_phone_verification (&&&)

Note that 'distance' isn't strongly correlated with anything. That means we've added in new information

K-Means

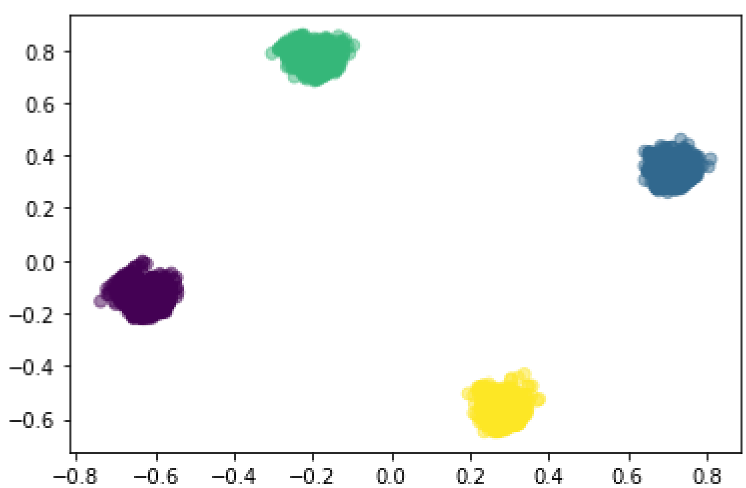
In [18]:
```
from sklearn.cluster import KMeans

n_clusters = 4
kmeans = KMeans(n_clusters=n_clusters, random_state=0).fit(X_train)
```

In [19]:
```
from sklearn.decomposition import PCA

X2 = PCA(n_components=2).fit_transform(X_train)
```

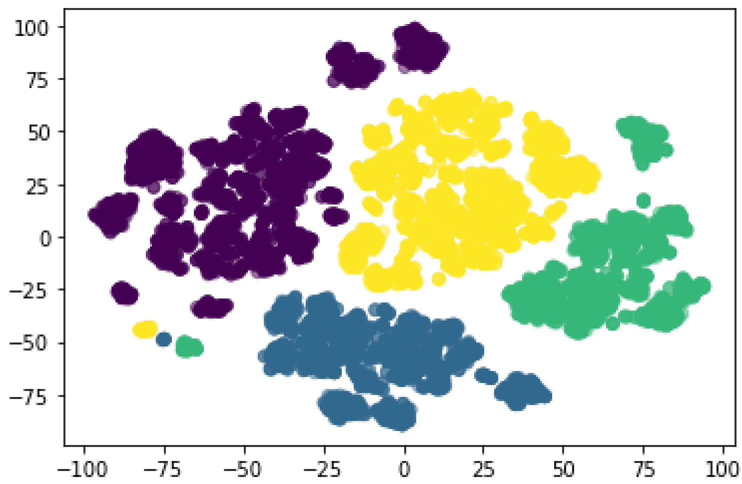
In [20]:
```
from sklearn.manifold import TSNE

X_embed = TSNE(n_components=2).fit_transform(X_train)
```

In [21]:
```
plt.scatter(X2[:,0], X2[:,1], c=kmeans.labels_, alpha=0.5)
# plt.savefig('img/clusters/clusters.pdf')
```

Out[21]: <matplotlib.collections.PathCollection at 0x1411c3400>

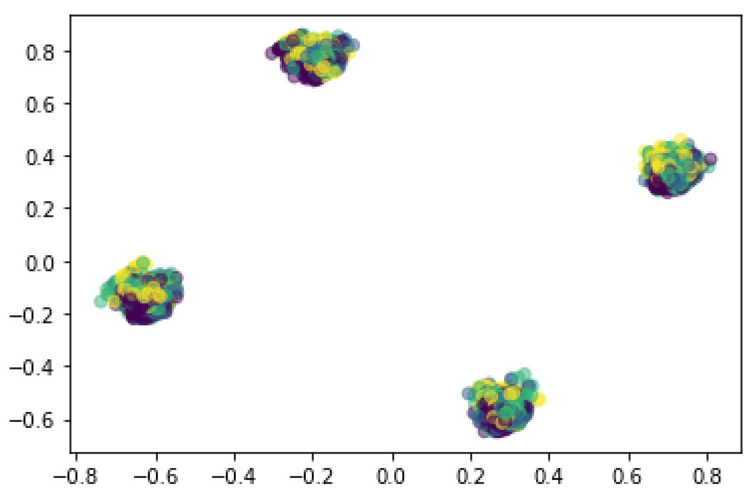


In [25]:
```
plt.scatter(X_embed[:,0], X_embed[:,1], c=kmeans.labels_, alpha=0.5)
# plt.savefig('img/clusters/tsne-clusters.pdf')
```

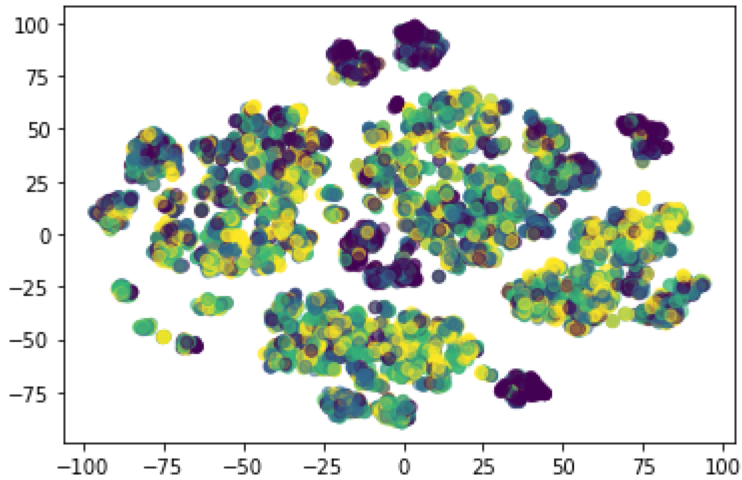


In [23]:
```
plt.scatter(X2[:,0], X2[:,1], c=y_train, alpha=0.5)
# plt.savefig('img/clusters/labels.pdf')
```

Out[23]: <matplotlib.collections.PathCollection at 0x140525820>



In [26]:
```
plt.scatter(X_embed[:,0], X_embed[:,1], c=y_train, alpha=0.5)
# plt.savefig('img/clusters/tsne-labels.pdf')
```



Experimental: K-Means -> Other Model

In [17]:
```
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.svm import SVC


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

    def _get_data_clusters(self, kmeans, 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 kmeans.labels_[i] == c]
            cluster_data[c] = (X.loc[idx], y.loc[idx])
        return cluster_data

    def fit(self, X, y):
        # divide data into clusters
        self.kmeans = KMeans(n_clusters=self.n_clusters, random_state=0)
.fit(X)
        cluster_data = self._get_data_clusters(self.kmeans, X, y)

        # fit 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 = AdaBoostClassifier(n_estimators=self.max_depth, rand
om_state=0)
#             clf = SVC(kernel='rbf', gamma=self.max_depth)
            clf.fit(X, y)
            self.cluster_models[c] = clf

    def predict(self, X):
        if self.cluster_models is None:
            raise AssertionError('You have to train the model first, du
h')

        pred_clusters = self.kmeans.predict(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
```

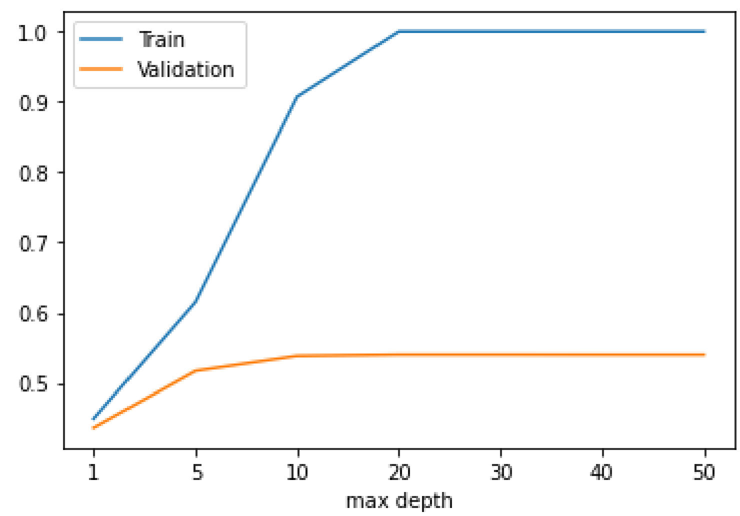
```python
nge(len(X))])
        return num_correct / len(X)
```

In [26]:
```python
# gammas = [0.1, 0.5, 1.0, 1.1, 1.2, 1.3, 1.5, 2.0]
# hybrid_fn = lambda v: HybridClassifier(n_clusters=4, max_depth=v)
# hybrid_train_scores, hybrid_val_scores = vary_cv(hybrid_fn, 'RBF gamm
a', gammas, X_train, y_train, k=3)
```

In [27]:
```python
# hybrid_val_scores
```

In [29]:
```python
depths = [1, 5, 10, 20, 30, 40, 50]
hybrid_fn = lambda v: HybridClassifier(n_clusters=4, max_depth=v)
hybrid_train_scores, hybrid_val_scores = vary_cv(hybrid_fn, 'max depth',
depths, X_train, y_train, k=3)
```

```
max depth: 50                    |██████████████████| 100%
```



In [30]:
```python
hybrid_val_scores
```

Out[30]:
```
[0.43631856213201115,
 0.5170953413903522,
 0.5381675446751368,
 0.5399235616155357,
 0.5399235616155356,
 0.5398202665013945,
 0.5398202665013945]
```

In [31]:
```python
# clusters = [1, 2, 3, 4, 5, 6]
# hybrid_fn = lambda v: HybridClassifier(n_clusters=v, max_depth=30)
# hybrid_train_scores, hybrid_val_scores = vary_cv(hybrid_fn, 'number of
clusters', clusters, X_train, y_train, k=3)
```

In [ ]:
```python
hybrid_val_scores
```

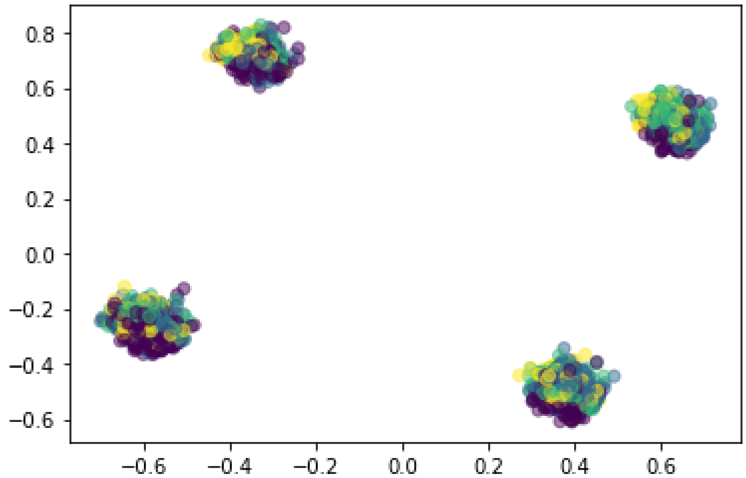
save results for best hybrid classifier

In [51]:
```
clf = HybridClassifier(n_clusters=4, max_depth=30)
clf.fit(X_train, y_train)
```

In [29]:
```
save_predictions(clf, 'hybrid')
```

In [29]:
```
X2_test = PCA(n_components=2).fit_transform(X_test)
plt.scatter(X2_test[:,0], X2_test[:,1], c=clf.predict(X_test), alpha=0.5
)
```

Out[29]: <matplotlib.collections.PathCollection at 0x15cb3b2e0>



SVM Model

In [25]:
```
from sklearn.svm import SVC
```

In [40]:
```
# kernels = ['linear', 'poly', 'rbf', 'sigmoid']
# svm_fn = lambda v: SVC(kernel=v)
# svm_train_scores, svm_val_scores = vary_cv(svm_fn, 'kernel', kernels,
 X_train, y_train, k=5)
```

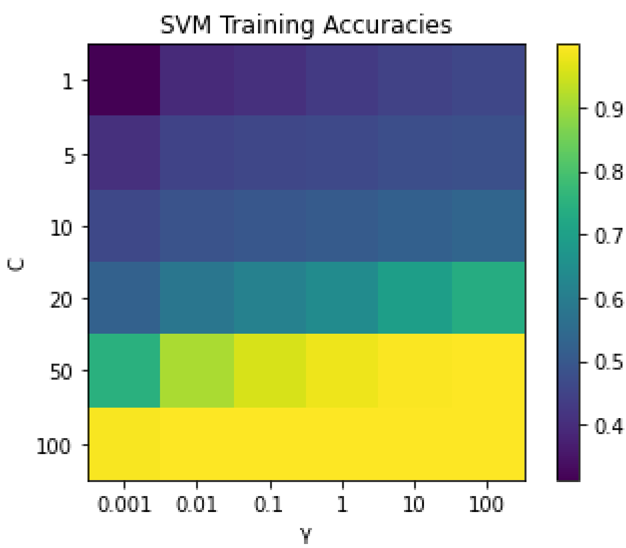
In [41]:
```
# gammas = [0.001, 0.01, 0.1, 1, 10, 100]
# rbf_fn = lambda v: SVC(kernel='rbf', gamma=v)
# rbf_train_scores, rbf_val_scores = vary_cv(rbf_fn, 'gamma', gammas, X_
train, y_train, k=5)
```

In [42]:
```
# C = [1, 5, 10, 20, 50, 100]
# rbf_fn = lambda v: SVC(kernel='rbf', C=v)
# rbf_train_scores, rbf_val_scores = vary_cv(rbf_fn, 'C', C, X_train, y_
train, k=5)
```

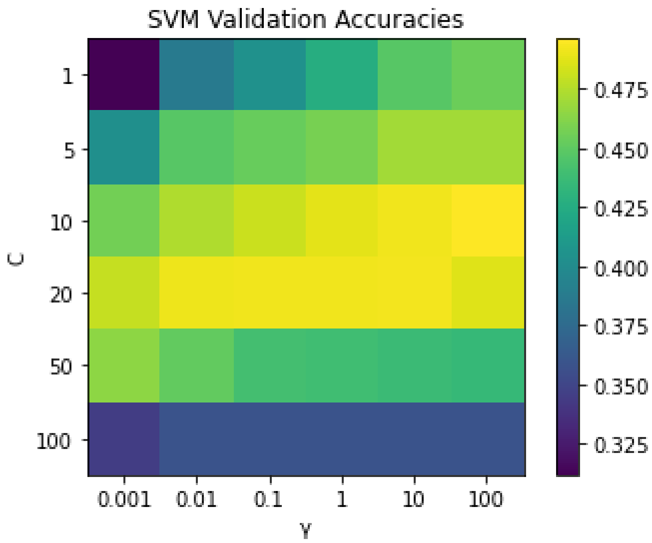
In [33]:
```
gammas = [0.001, 0.01, 0.1, 1, 10, 100]
C = [1, 5, 10, 20, 50, 100]

train_acc = np.zeros((len(gammas), len(C)))
val_acc = np.zeros((len(gammas), len(C)))

for i in range(len(gammas)):
    for j in range(len(C)):
        progress(i*len(C) + j, len(gammas) * len(C), f'γ={gammas[i]}, C={
C[j]}')
        clf = SVC(kernel='rbf', gamma=gammas[i], C=C[j])
        train_acc[i][j], val_acc[i][j] = cv(clf, X_train, y_train, k=3)
```

```
γ=100, C=100                          ||  100%
```

In [52]:
```
plt.imshow(train_acc.T)
plt.colorbar()
plt.title('SVM Training Accuracies')
plt.xlabel('γ')
plt.ylabel('C')
plt.xticks(ticks=range(len(gammas)), labels=gammas)
plt.yticks(ticks=range(len(C)), labels=C)
plt.savefig('img/training/svm_train.pdf')
```



In [55]:
```
plt.imshow(test_acc.T)
plt.colorbar()
plt.title('SVM Validation Accuracies')
plt.xlabel('γ')
plt.ylabel('C')
plt.xticks(ticks=range(len(gammas)), labels=gammas)
plt.yticks(ticks=range(len(C)), labels=C)
plt.savefig('img/training/svm_val.pdf')
```

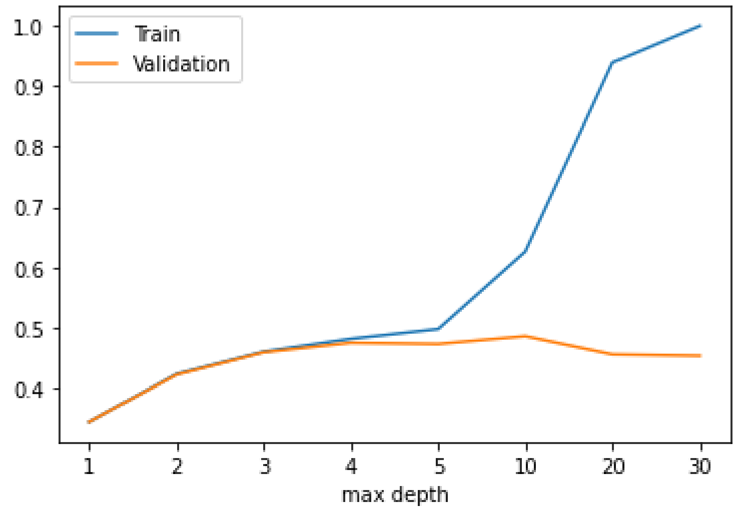


In [49]:
```
########## γ = 10, C = 20
```

Decision Tree

In [22]:
```
from sklearn.tree import DecisionTreeClassifier

depths = [1, 2, 3, 4, 5, 10, 20, 30]
tree_fn = lambda v: DecisionTreeClassifier(max_depth=v, random_state=0)
tree_train_scores, tree_val_scores = vary_cv(tree_fn, 'max depth', depth
s, X_train, y_train, k=10)
```

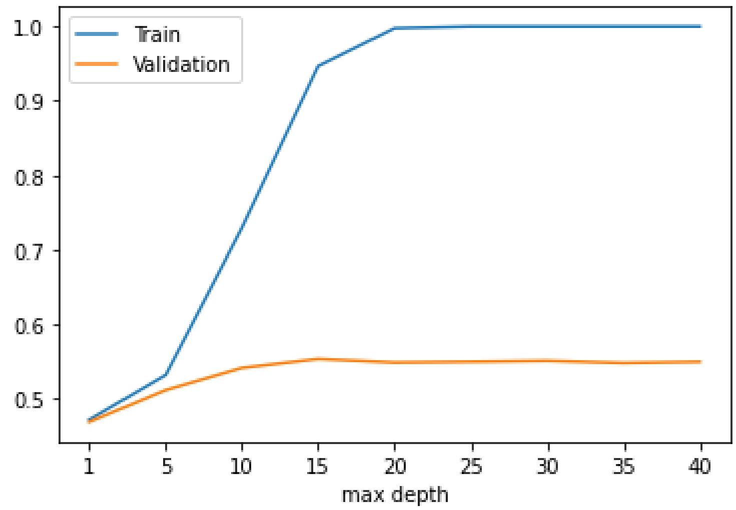
```
max depth: 30                    |██████████| 100%
```



Random Forest

In [57]:
```
from sklearn.ensemble import RandomForestClassifier

depths = [1, 5, 10, 15, 20, 25, 30, 35, 40]
forest_fn = lambda v: RandomForestClassifier(max_depth=v, random_state=0
)
forest_train_scores, forest_val_scores = vary_cv(forest_fn, 'max depth',
depths, X_train, y_train, k=10)
```

```
max depth: 40                    |████████████████████| 100%
```



In [ ]:
```
plt.plot(forest_train_scores, label='Train')
plt.plot(forest_val_scores, label='Validation')
plt.xticks(ticks=range(len(forest_train_scores)), labels=depths)
plt.xlabel('max depth')
plt.ylabel('Accuracy')
plt.title('Random Forest')
plt.legend()
plt.savefig('img/training/rf.pdf')
```

In [54]:
```
clf = RandomForestClassifier(max_depth=20, random_state=0)
clf.fit(X_train, y_train)
save_predictions(clf, 'random_forest')
```

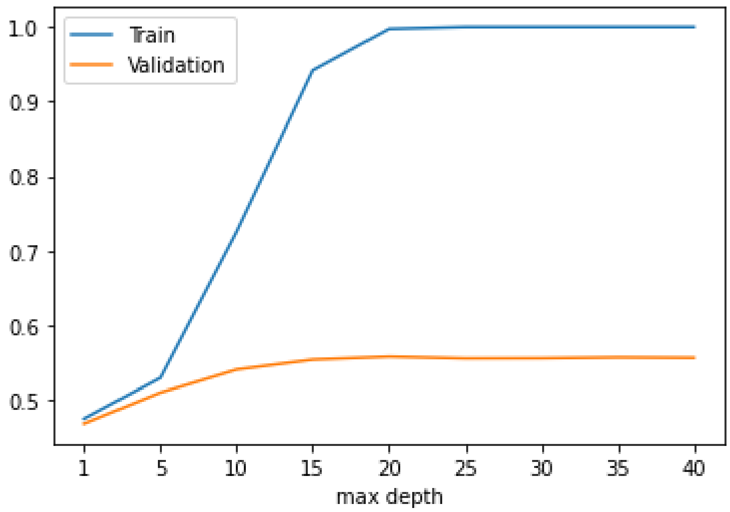
After Correlated Feature Removal

In [62]:
```
for f in ['number_of_reviews', 'require_guest_phone_verification']:
    X_train = X_train.drop(columns=[f])
    X_test = X_test.drop(columns=[f])
```

Random Forest

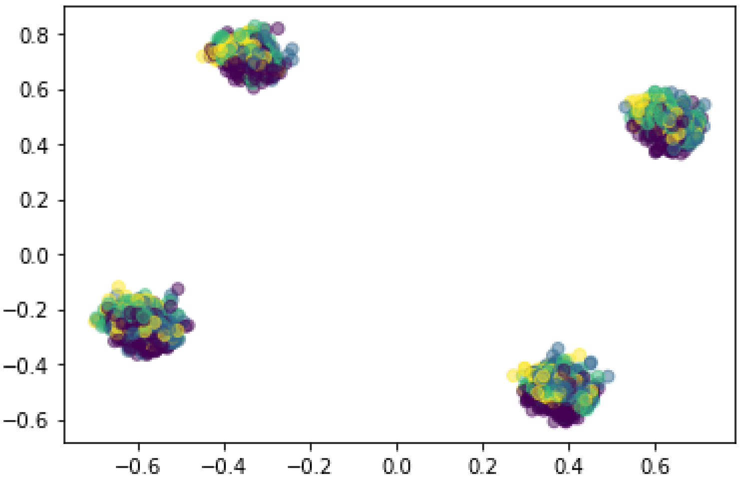
In [63]:
```
depths = [1, 5, 10, 15, 20, 25, 30, 35, 40]
forest_fn = lambda v: RandomForestClassifier(max_depth=v, random_state=0
)
forest_train_scores, forest_val_scores = vary_cv(forest_fn, 'max depth',
depths, X_train, y_train, k=10)
```

```
max depth: 40                    |██████████| 100%
```



In [ ]:
```
plt.plot(forest_train_scores, label='Train')
plt.plot(forest_val_scores, label='Validation')
plt.xticks(ticks=range(len(forest_train_scores)), labels=depths)
plt.xlabel('max depth')
plt.ylabel('Accuracy')
plt.title('Random Forest')
plt.legend()
plt.savefig('img/training/rf_removed.pdf')
```

In [33]:
```
clf = RandomForestClassifier(max_depth=20, random_state=0)
clf.fit(X_train, y_train)
save_predictions(clf, 'RF_feature_removal')

X = PCA(n_components=2).fit_transform(X_test)
plt.scatter(X[:,0], X[:,1], c=clf.predict(X_test), alpha=0.5)
plt.savefig('img/preds/forest.pdf')
```

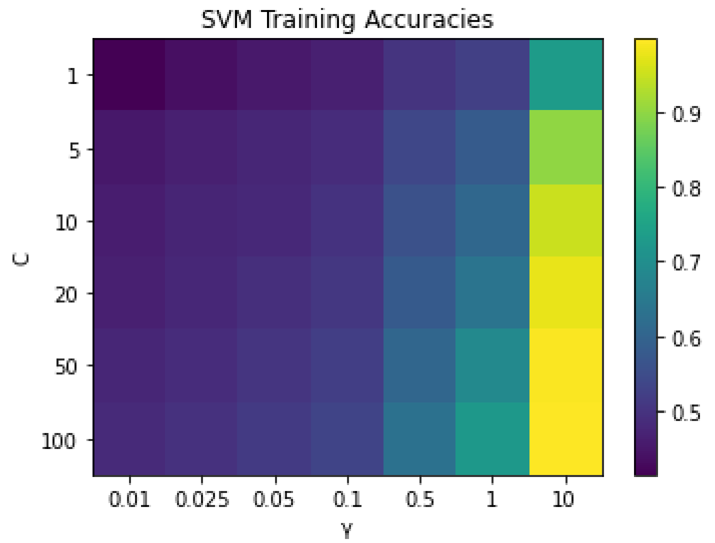


SVM

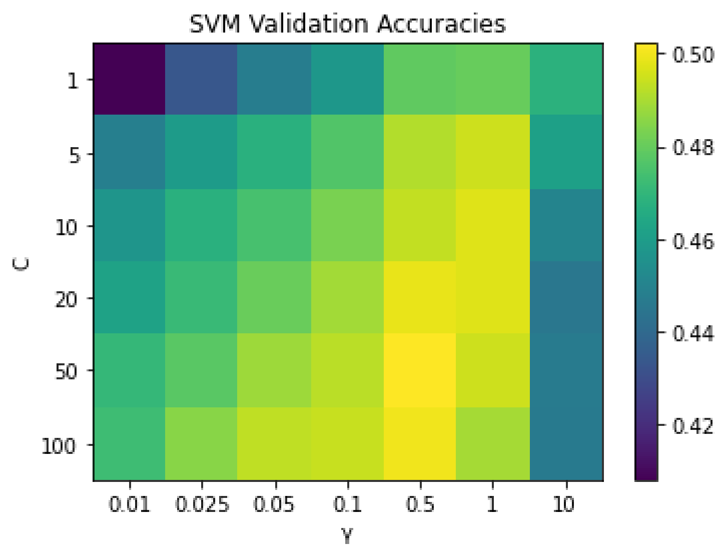
In [35]:
```
gammas = [0.01, 0.025, 0.05, 0.1, 0.5, 1, 10]
C = [1, 5, 10, 20, 50, 100]

train_acc = np.zeros((len(gammas), len(C)))
val_acc = np.zeros((len(gammas), len(C)))

for i in range(len(gammas)):
    for j in range(len(C)):
        progress(i*len(C) + j, len(gammas) * len(C), f'γ={gammas[i]}, C={
C[j]}')
        clf = SVC(kernel='rbf', gamma=gammas[i], C=C[j])
        train_acc[i][j], val_acc[i][j] = cv(clf, X_train, y_train, k=5)
```

```
γ=10, C=100                     |                    | 100%
```

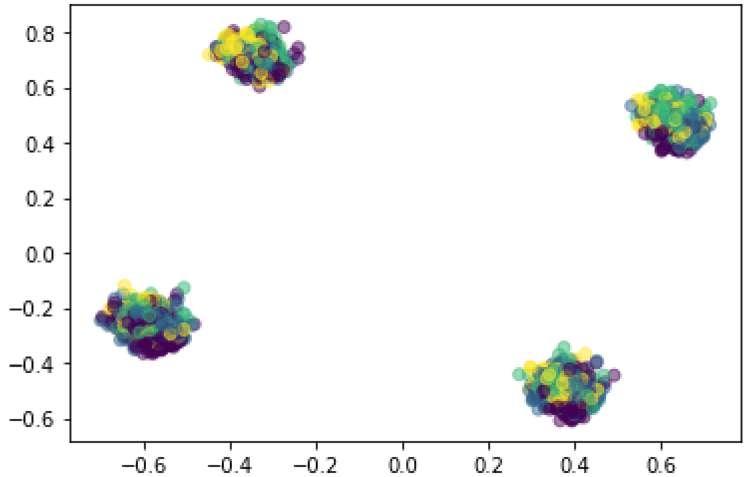
In [36]:
```
plt.imshow(train_acc.T)
plt.colorbar()
plt.title('SVM Training Accuracies')
plt.xlabel('γ')
plt.ylabel('C')
plt.xticks(ticks=range(len(gammas)), labels=gammas)
plt.yticks(ticks=range(len(C)), labels=C)
plt.savefig('img/training/svm_train_removed.pdf')
```



In [37]:
```
plt.imshow(val_acc.T)
plt.colorbar()
plt.title('SVM Validation Accuracies')
plt.xlabel('γ')
plt.ylabel('C')
plt.xticks(ticks=range(len(gammas)), labels=gammas)
plt.yticks(ticks=range(len(C)), labels=C)
plt.savefig('img/training/svm_val_removed.pdf')
```



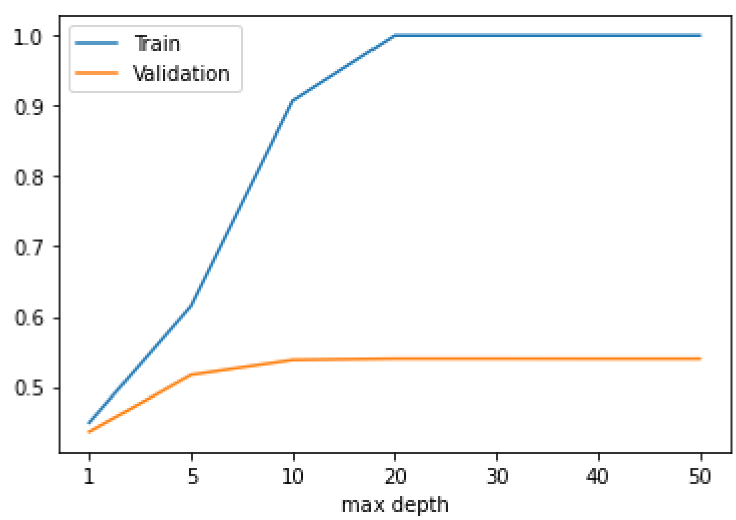
In [32]:
```
clf = SVC(kernel='rbf', gamma=0.5, C=50)
clf.fit(X_train, y_train)
save_predictions(clf, 'svm_removed')

X = PCA(n_components=2).fit_transform(X_test)
plt.scatter(X[:,0], X[:,1], c=clf.predict(X_test), alpha=0.5)
plt.savefig('img/preds/svm.pdf')
```



Cluster -> Random Forest

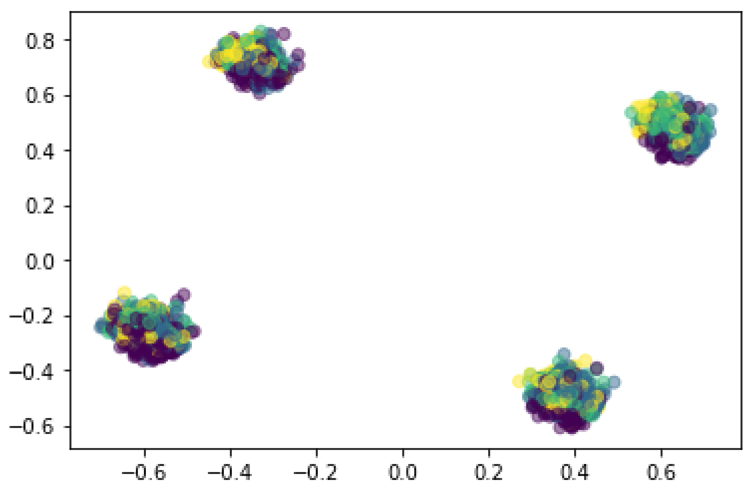
In [20]:
```
depths = [1, 5, 10, 20, 30, 40, 50]
hybrid_fn = lambda v: HybridClassifier(n_clusters=4, max_depth=v)
hybrid_train_scores, hybrid_val_scores = vary_cv(hybrid_fn, 'max depth',
depths, X_train, y_train, k=3)
```

```
max depth: 50                    |██████████| 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('General Cluster Model')
plt.legend()
plt.savefig('img/training/hybrid_removed.pdf')
```

In [31]:
```
clf = HybridClassifier(n_clusters=4, max_depth=20)
clf.fit(X_train, y_train)
save_predictions(clf, 'hybrid_removed')

X = PCA(n_components=2).fit_transform(X_test)
plt.scatter(X[:,0], X[:,1], c=clf.predict(X_test), alpha=0.5)
plt.savefig('img/preds/hybrid.pdf')
```



removing other features

In [21]:
```
for f in ['bedrooms']:
    X_train = X_train.drop(columns=[f])
    X_test = X_test.drop(columns=[f])
```

In [22]:
```
depths = [1, 5, 10, 15, 20, 25, 30, 35, 40]
forest_fn = lambda v: RandomForestClassifier(max_depth=v, random_state=0
)
forest_train_scores, forest_val_scores = vary_cv(forest_fn, 'max depth',
depths, X_train, y_train, k=10)
```

```
max depth: 40                |██████████████████████| 100%
```