Preprocessing

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import spearmanr
from dateutil import parser
from datetime import datetime
from sklearn.model_selection import train_test_split

# Import data
df = pd.read_csv('tripdata.csv')
df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')
df = df.sample(100000) #100,000 random sample

# Remove columns with missing birth year
df = df[df.birth_year != '\\N']
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-pa
ckages/pandas/core/ops/__init__.py:1115: FutureWarning: elementwise com
parison failed; returning scalar instead, but in the future will perfor
m elementwise comparison
  result = method(y)
```

In [2]:
```
for i in df.index:
    # Convert string datetimes to float timestamps
    for col in ['starttime', 'stoptime']:
        t = df.at[i, col]
        timestamp = datetime.timestamp(parser.parse( t ))
        df.at[i, col] = timestamp

    # Convert usertype from string to int
    u = df.at[i, 'usertype']
    if u == 'Subscriber':
        df.at[i, 'usertype'] = 0
    else:
        df.at[i, 'usertype'] = 1
```

In [ ]:
```

```

Clustering

In [3]:
```
# Put start and end locations into dataframes
start_long = df['start_station_longitude']
start_lat = df['start_station_latitude']
start_X = pd.DataFrame({'start_long': start_long, 'start_lat': start_lat
})

end_long = df['end_station_longitude']
end_lat = df['end_station_latitude']
end_X = pd.DataFrame({'end_long': end_long, 'end_lat': end_lat})

all_long = start_long.append(end_long)
all_lat = start_lat.append(end_lat)
all_X = pd.DataFrame({'all_long': all_long, 'all_lat': all_lat})
```

In [4]:
```
# Count unique stations
count = 0
seen = {}
for row in all_X.values:
    if row[0] not in seen or seen[row[0]] != row[1]:
        seen[row[0]] = row[1]
        count += 1
count
```

Out[4]: 793

In [5]:
```
from sklearn.cluster import KMeans

# Initialize kmeans and fit on all locations
n_clusters = 13
kmeans = KMeans(n_clusters=n_clusters, random_state=0)
kmeans.fit(all_X)

# Get clusters for start, end, and all locations
all_Y = kmeans.predict(all_X)
start_Y = kmeans.predict(start_X)
end_Y = kmeans.predict(end_X)

# Add start and end clusters to dataframe
df['start_cluster'] = start_Y
df['end_cluster'] = end_Y

# Dictionary of clusters and positions
cluster_coords = {}
for i in range(0, n_clusters):
    cluster_coords[i] = [kmeans.cluster_centers_[i,0], kmeans.cluster_ce
nters_[i,1]]
```

In [35]:
```
# Correlation

attributes = [i for i in df]
cor = pd.DataFrame(index=attributes, columns=attributes)

for i in df:
    for j in df:
        cor[i][j] = spearmanr(df[i], df[j])[0]

cor
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call l
ast)
<ipython-input-35-b78c21c17a38> in <module>
      6 for i in df:
      7     for j in df:
----> 8         cor[i][j] = spearmanr(df[i], df[j])[0]
      9
     10 cor

/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-pa
ckages/scipy/stats/stats.py in spearmanr(a, b, axis, nan_policy)
   3741                 variable_has_nan = np.isnan(a).sum(axis=axisout
)
   3742
-> 3743     a_ranked = np.apply_along_axis(rankdata, axisout, a)
   3744     rs = np.corrcoef(a_ranked, rowvar=axisout)
   3745     dof = n_obs - 2  # degrees of freedom

<__array_function__ internals> in apply_along_axis(*args, **kwargs)

/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-pa
ckages/numpy/lib/shape_base.py in apply_along_axis(func1d, axis, arr, *
args, **kwargs)
    377     except StopIteration:
    378         raise ValueError('Cannot apply_along_axis when any iter
ation dimensions are 0')
--> 379     res = asanyarray(func1d(inarr_view[ind0], *args, **kwargs))
    380
    381     # build a buffer for storing evaluations of func1d.

/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-pa
ckages/scipy/stats/stats.py in rankdata(a, method)
   6655     arr = np.ravel(np.asarray(a))
   6656     algo = 'mergesort' if method == 'ordinal' else 'quicksort'
-> 6657     sorter = np.argsort(arr, kind=algo)
   6658
   6659     inv = np.empty(sorter.size, dtype=np.intp)

<__array_function__ internals> in argsort(*args, **kwargs)

/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-pa
ckages/numpy/core/fromnumeric.py in argsort(a, axis, kind, order)
   1082
   1083     """
-> 1084     return _wrapfunc(a, 'argsort', axis=axis, kind=kind, order=
order)
   1085
   1086

/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-pa
ckages/numpy/core/fromnumeric.py in _wrapfunc(obj, method, *args, **kwd
s)
     59
     60     try:
---> 61         return bound(*args, **kwds)
```

```
     62     except TypeError:
     63         # A TypeError occurs if the object does have such a met
hod in its

KeyboardInterrupt:
```

In [29]:
```
cor.to_csv(r'/Users/tommygeiger/Jupyter/Data Mining/Citi Bike/correlatio
n.csv')
```

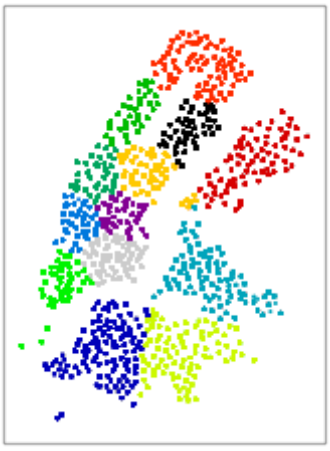
Plotting Clusters

In [62]:
```
# Plot stations and clusters
plt.scatter(all_long, all_lat, c=all_Y, s=1, cmap='nipy_spectral')

# Plot display settings
ax = plt.gca()
ax.set_aspect('equal')
ax.set_facecolor('white')
ax.grid(False)
ax.yaxis.set_major_locator(plt.NullLocator())
ax.xaxis.set_major_formatter(plt.NullFormatter())
plt.tick_params(bottom=False, top=False)
for s in ax.spines:
    ax.spines[s].set_color('black')
    ax.spines[s].set_linewidth(0.25)

# # Get graph bounds
# print(ax.get_xlim())
# print(ax.get_ylim())

# Save image to disk
plt.savefig('clusters.png', transparent=True, dpi=1000)
```



Distance

In [30]:
```
import math

# Function that returns average distance (miles) between true and predic
ted clusters
# (list, list) -> float
def avg_error (true, predict):
    error = []
    for i in range(0, true.size):
        true_coords = np.array(cluster_coords[true.values[i]])
        predict_coords = np.array(cluster_coords[predict[i]])

        # Haversize distance (km)
        distance = haversine_distance(true_coords, predict_coords)
        error.append(distance)

    # Return average error across predicted list
    return np.sum(error)/len(error)

# Haversine distance
# Distance (miles) between two gps points (long, lat)
def haversine_distance (true, predict):
    t_long, t_lat = true
    p_long, p_lat = predict
    R = 3959 #miles

    delta_long = math.radians(t_long - p_long)
    delta_lat = math.radians(t_lat - p_lat)

    A = math.sin(delta_lat/2) * math.sin(delta_lat/2) + math.cos(math.ra
dians(t_lat)) \
        * math.cos(math.radians(p_lat)) * math.sin(delta_long/2) * math.
sin(delta_long/2)
    C = 2 * math.atan2(math.sqrt(A), math.sqrt(1-A))
    D = R * C
    return D
```

In [31]:
```
# Seperate attributes (x) and classification (y)
x = df[['starttime', 'usertype', 'birth_year', 'gender', 'start_cluster'
, 'start_station_latitude', 'start_station_longitude']]
y = df.end_cluster

# Split into test and train sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=0)
```

In [32]:
```
haversine_distance(cluster_coords[1], cluster_coords[2])
```

Out[32]:
```
3.286509936575719
```

Format for mapping (deprecated)

In [33]:
```
# colors = []
# for c in df.start_cluster:
#     colors.append(s)

# # print(hex(1<<23))

# out = pd.DataFrame({'lat': df.start_station_latitude, \
#                     'long': df.start_station_longitude, \
#                     'name': None, 'color': colors})

# out = out.head(1000)
# out.to_csv(r'/Users/tommygeiger/Jupyter/Data Mining/Citi Bike/export.csv')
# out
```

Naive Bayes

In [34]:
```
from sklearn.naive_bayes import GaussianNB

# Fit and predict model
gnb = GaussianNB()
gnb.fit(x_train, y_train)
nbpredict = gnb.predict(x_test)

# Calculate error
nberror = avg_error(y_test, nbpredict)
print('average prediction error:', nberror, 'miles')
```

```
average prediction error: 1.9869838647467601 miles
```

In [ ]:
```

```

In [74]:
```
#DEMO

for i in df.index:
    if 'Barclay St & Church St' in df.at[i,'start_station_name']:
        cluster = df.at[i,'start_cluster']
        break
print(cluster)
```

```
7
```

Neural Network

In [37]:
```
from sklearn.neural_network import MLPClassifier

#Initialize model, train and predict
mlp = MLPClassifier(max_iter = 1000)
mlp.fit(x_train, y_train)
nnpredict = mlp.predict(x_test)

# Calculate error
nnerror = avg_error(y_test, nnpredict)
print('average prediction error:', nnerror, 'miles')
```

```
average prediction error: 1.9869838647467601 miles
```

SVM

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

# #Initialize model, train and predict
# svc = SVC(probability = True, gamma = 'auto')
# svc.fit(x_train, y_train)
# svmpredict = svc.predict(x_test)

# # Calculate error
# svmerror = avg_error(y_test, svmpredict)
# print('average prediction error:', svmerror, 'miles')
```

KNN

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier

ks = []
errors = []
best_k = 1
min_error = 1

for k in range(1, 500):
    # Initialize model
    knn = KNeighborsClassifier(n_neighbors = k)
    knn.fit(x_train, y_train)
    predict = knn.predict(x_test)

    # Calculate error
    error = avg_error(y_test, predict)

    # Add to lists
    ks.append(k)
    errors.append(error)

    if error < min_error:
        best_k = k
        min_error = error

    print (k/5,'%')

plt.plot(ks, errors)
plt.title('k vs. average error')
plt.xlabel('k')
plt.ylabel('average error (miles)')
plt.show()

plt.savefig('knn_error.png', transparent=True, dpi=1000)

print(min_error)
print(best_k)
```

In [ ]:
```
# Use best k
knn = KNeighborsClassifier(n_neighbors = 400)
knn.fit(x_train, y_train)
predict = knn.predict(x_test)
error = avg_error(y_test, predict)
print('average prediction error:', error, 'miles')
```

In [ ]:
```
# Find the best number of clusters to use
# RUN THIS CELL LAST
# otherwise it will mess up cluster_coords

import warnings
warnings.filterwarnings('ignore')

gnb = GaussianNB()

ns = []
errors = []

for n in range(2, 15):
    # Initialize and fit model
    kmeans = KMeans(n_clusters=n, random_state=0)
    kmeans.fit(all_X)

    # Get clusters for start, end, and all locations
    all_Y = kmeans.predict(all_X)
    start_Y = kmeans.predict(start_X)
    end_Y = kmeans.predict(end_X)

    # Add start and end clusters to dataframe
    df['start_cluster'] = start_Y
    df['end_cluster'] = end_Y

    # Dictionary of cluster positions
    cluster_coords = {}
    for i in range(0, n):
        cluster_coords[i] = [kmeans.cluster_centers_[i,0], kmeans.cluste
r_centers_[i,1]]

    # Split into train/test
    x = df[['starttime', 'usertype', 'birth_year', 'gender', 'start_clus
ter']]
    y = df.end_cluster
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=
0.2, random_state=0)

    # Fit and predict
    gnb.fit(x_train, y_train)
    predict = gnb.predict(x_test)

    # Calculate error
    error = avg_error(y_test, predict)

    # Add to lists
    ns.append(n)
    errors.append(error)

    print (n*4,'%')

plt.plot(ns, errors)
plt.title('n_clusters vs. average error')
plt.xlabel('n_clusters')
plt.ylabel('average error (miles)')
```

```
plt.show()

plt.savefig('kmeans_error.png', transparent=True, dpi=1000)
```

In [ ]:

In [ ]:
```
# Find the best number of clusters to use
# RUN THIS CELL LAST
# otherwise it will mess up cluster_coords


import warnings
warnings.filterwarnings('ignore')

mlp = MLPClassifier(max_iter = 5000)

ns = []
errors = []

for n in range(2, 10):

    # Initialize and fit model
    kmeans = KMeans(n_clusters=n, random_state=0)
    kmeans.fit(all_X)

    # Get clusters for start, end, and all locations
    all_Y = kmeans.predict(all_X)
    start_Y = kmeans.predict(start_X)
    end_Y = kmeans.predict(end_X)

    # Add start and end clusters to dataframe
    df['start_cluster'] = start_Y
    df['end_cluster'] = end_Y

    # Dictionary of cluster positions
    cluster_coords = {}
    for i in range(0, n):
        cluster_coords[i] = [kmeans.cluster_centers_[i,0], kmeans.cluste
r_centers_[i,1]]

    # Split into train/test
    x = df[['starttime', 'usertype', 'birth_year', 'gender', 'start_clus
ter']]
    y = df.end_cluster
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=
0.2, random_state=0)

    # Fit and predict
    mlp.fit(x_train, y_train)
    predict = mlp.predict(x_test)

    # Calculate error
    error = avg_error(y_test, predict)

    # Add to lists
    ns.append(n)
    errors.append(error)

    print (n*2,'%')

plt.plot(ns, errors)
plt.title('n_clusters vs. average error')
```

```
plt.xlabel('n_clusters')
plt.ylabel('average error (miles)')
plt.show()

plt.savefig('kmeans_error.png', transparent=True, dpi=1000)
```

In [ ]: