Predicting bus arrival times using k Nearest Neighbors

In [1]:
```
import numpy as np
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error as mse
from matplotlib import pyplot as plt
import seaborn as sns
%config InlineBackend.figure_format = 'retina'
```

In [2]:
```
class FoobarExtrapolator:
    def __init__(self):
        pass

    def fit(self, X, y):
        pass #model doesn't depend on previous data

    def predict(self, X):
        return (X[:,-1] - X[:,0])*2
```

In [3]:
```
class MeanModel:
    def __init__(self):
        pass

    def fit(self, X, y):
        self.meany = y.mean()

    def predict(self, X):
        return np.ones(X.shape[0])*self.meany
```

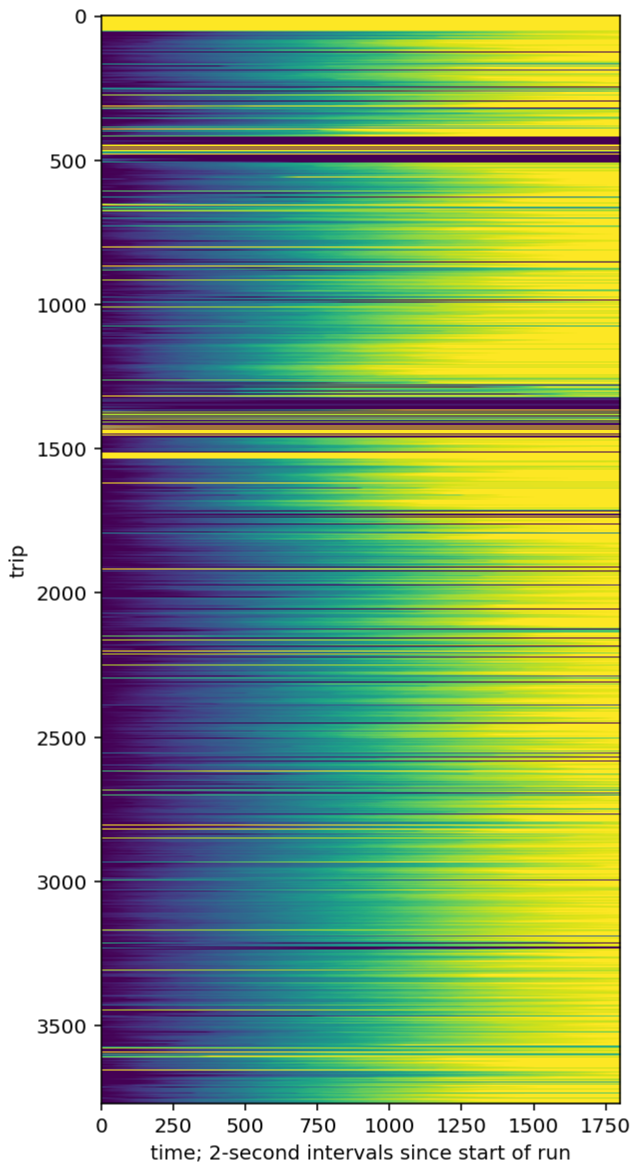
In [4]:
```
# row: trip; each row is a _single_ run of a bus along the route.
# col: time; each column is a two-second interval
# cell(i,j): the position of the trip i at time j.
traces = np.load("data/traces.npy")
```

In [5]:
```
traces.shape
```

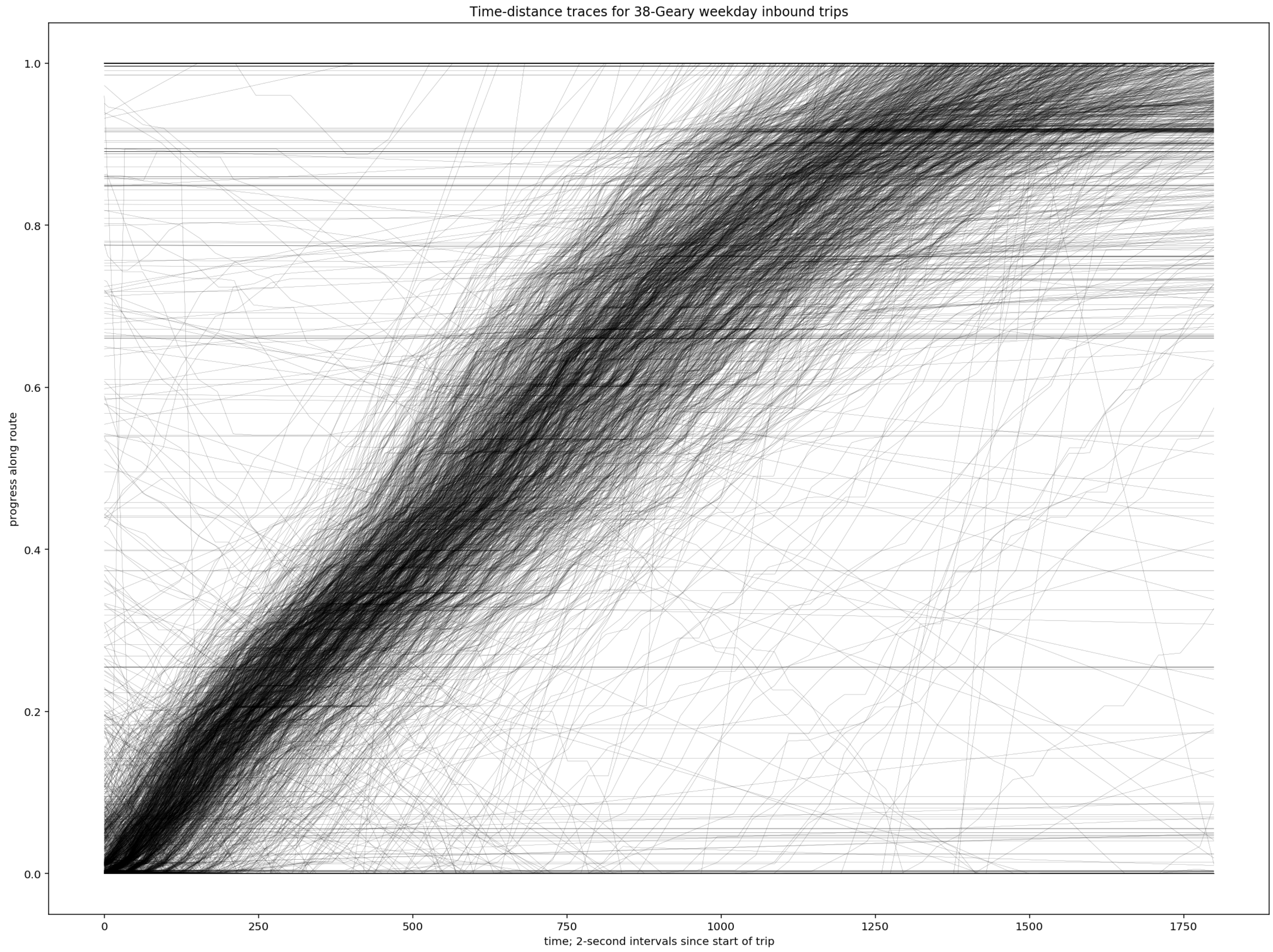
Out[5]:
```
(3769, 1800)
```

In [6]:
```
fig, ax = plt.subplots(figsize=(10,10))
ax.imshow( traces, vmin=0, vmax=1.0 )
ax.set_xlabel("time; 2-second intervals since start of run")
ax.set_ylabel("trip")
```

Out[6]:
```
Text(0,0.5,'trip')
```



In [8]:
```
fig, ax = plt.subplots(figsize=(20,15))
ax.set_title("Time-distance traces for 38-Geary weekday inbound trips")
ax.set_xlabel("time; 2-second intervals since start of trip")
ax.set_ylabel("progress along route")
for row in traces[0:2000]:
    ax.plot( row, lw=0.1, c="black" )
```



A prediction task

Let's use kNN to solve the holy grain of prediction tasks: transit vehicle arrival prediction.

Say there's a vehicle currently traveling the 38-Geary inbound route. We have access to several GPS location fixes since it started its run. Using that data, we want to predict where it will be at a certain time.

Our first step is to convert the GPS data into a "trace". A trace is a vector `x`, where the value `x_i` is the location of the bus along the route at time `i`.

For example, we can create a trace where each `i` is a 1-minute interval. In that case, `x[0]` is the location of the bus at sttart; `x[5]` is the location of the bus at 5 minutes since start, &c.

This trace vector `x` is a **feature vector**, where the feature `i` is the progress of the vehicle at time i.

This is almost a machine learning problem

For our training set, we can use the traces between `i=0` and `i=m` for all previously observed trips. For our target variable, we can use the progress of all previously observed trips at target time `i_target`.

Say we have access to an upcoming vehicle's trace up intil 10 minutes. The target is where it'll be at t=20 minutes.

In [9]:
```
t0 = 10
t1 = 20
X = traces[:, 0:int(t0*60/2)]
y = traces[:, int(t1*60/2)]
```

In [10]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
In [11]: model = KNeighborsRegressor(n_neighbors=4)
         model.fit( X_train, y_train )
         yhat = model.predict( X_test )
         mse(yhat, y_test)
```

```
Out[11]: 0.003767755516003575
```

```
In [12]: # compare against our naive extrapolator
         model = FoobarExtrapolator()
         model.fit( X_train, y_train )
         yhat = model.predict( X_test )
         mse(yhat, y_test)
```

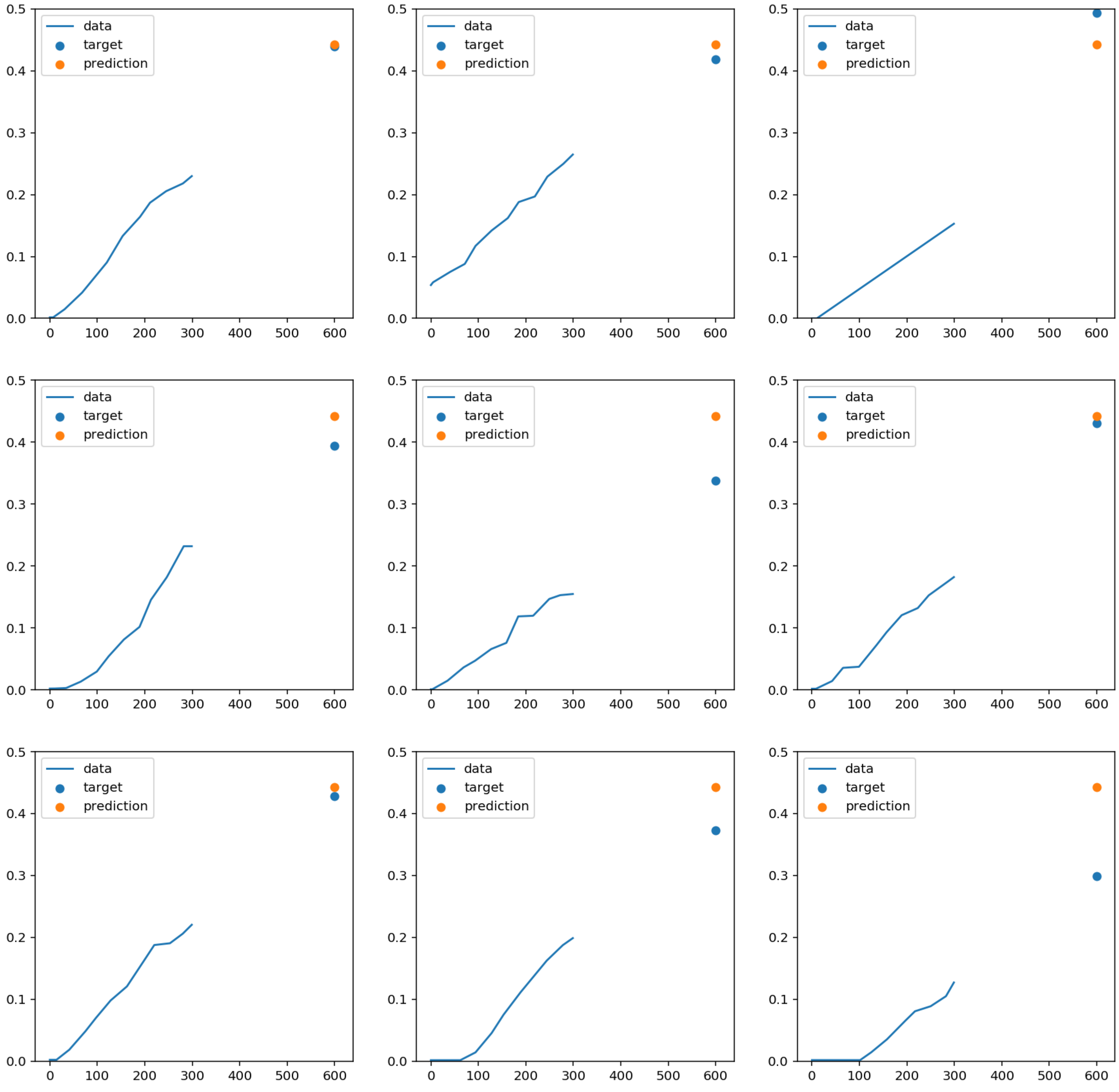
```
Out[12]: 0.09242003940461076
```

```
In [13]: #compare against our "mean model, which always returns the mean training label
         model = MeanModel()
         model.fit( X_train, y_train )
         yhat = model.predict( X_test )
         mse(yhat, y_test)
```

```
Out[13]: 0.04433015763917827
```

```
In [14]: fig,axs = plt.subplots(3,3, figsize=(15,15))

         for i, ax in enumerate(axs.ravel()):
             ax.plot( X_test[i], label="data" )
             ax.scatter( 600, y_test[i], label="target" )
             ax.scatter( 600, yhat[i], label="prediction" )
             ax.set_ylim(0,0.5)
             ax.legend()
```



Dimensionality reduction with Principle Component Analysis

In [15]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import normalize
```

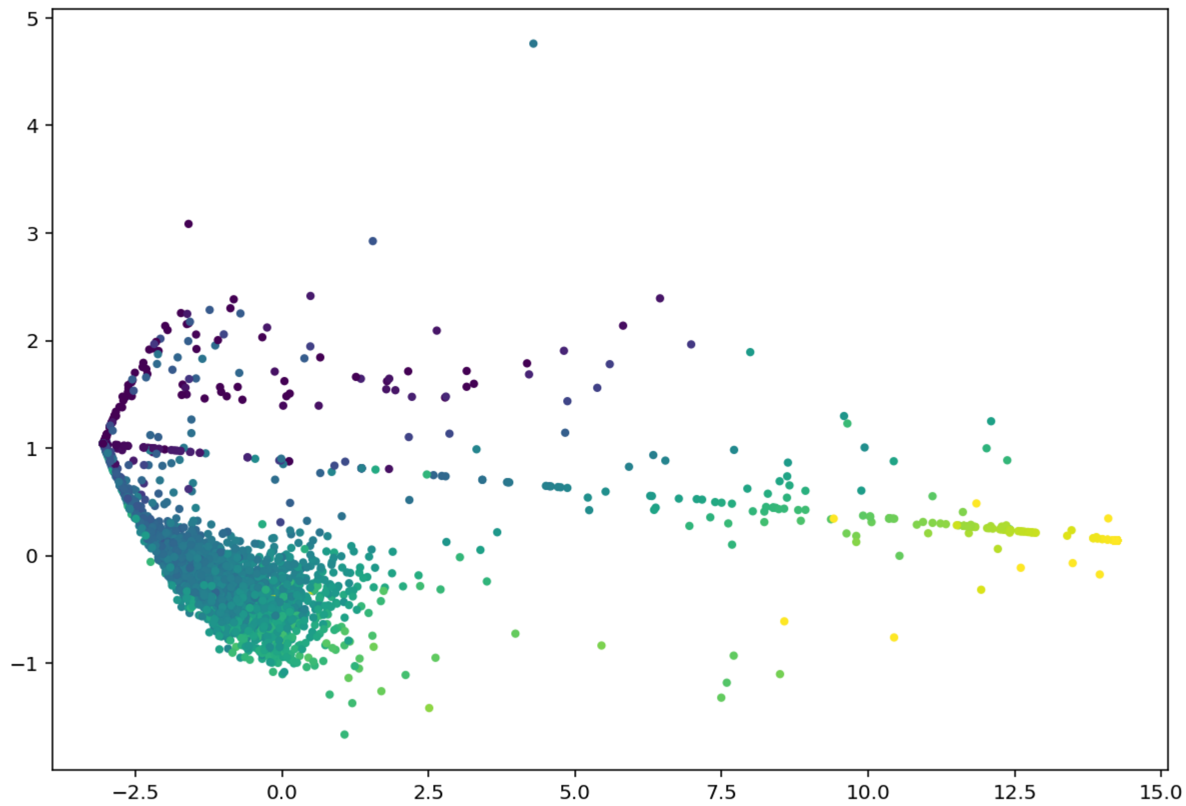
In [16]:
```
pca = PCA(n_components=20)
pca.fit(X_train)
X_pca = pca.transform(X)
```

In [17]:
```
#X_pca = normalize( X_pca )
```

In [18]:
```
from mpl_toolkits.mplot3d import Axes3D
```

In [19]:
```
fig,ax = plt.subplots(figsize=(10,7))
ax.scatter( X_pca.T[0], X_pca.T[1], s=10, c=y, label="foobar" )
```

Out[19]: <matplotlib.collections.PathCollection at 0x11d317e10>



In [22]:
```
X_pca_train, X_pca_test, y_pca_train, y_pca_test = train_test_split(X_pca, y)
```

In [23]:
```
for i in range(1,20):
    model = KNeighborsRegressor(n_neighbors=i)
    model.fit( X_pca_train, y_pca_train )
    yhat_pca = model.predict( X_pca_test )
    print( i, mse(yhat_pca, y_pca_test) )
```

```
1 0.006595273494313357
2 0.004466464462278115
3 0.00393521952764864
4 0.0037726363653018467
5 0.0036527829703182375
6 0.003630806979433291
7 0.003628522457791577
8 0.0036814893580253472
9 0.0037375782978718943
10 0.0037607275935388935
11 0.0037169650317509057
12 0.003722968855190793
13 0.00378946993272464
14 0.0038393325293772447
15 0.0038609394918785326
16 0.0038924715157603157
17 0.003906704277317209
18 0.003920416186433773
19 0.00393858038471648
```