Assignment 3: Exploring Tree-Based Regression Methods for 3D Sinusoidal Data

DTSC 680: Applied Machine Learning

Name: Juliana Meirelles

Directions and Overview

The main purpose of this assignment is for you to gain experience using tree-based methods to solve simple regression problems. In this assignment, you will fit a `Gradient-Boosted Regression Tree` , a `Random Forest` , and a `Decision Tree` to a noisy 3D sinusoidal data set. Since these models can be trained very quickly on the supplied data, I want you to first manually adjust hyperparameter values and observe their influence on the model's predictions. That is, you should manually sweep the hyperparameter space and try to hone in on the optimal hyperparameter values, again, *manually*. (Yep, that means guess-and-check: pick some values, train the model, observe the prediction curve, repeat.)

But wait, there's more! Merely attempting to identify the optimal hyperparameter values is not enough. Be sure to really get a visceral understanding of how altering a hyperparameter in turn alters the model predictions (i.e. the prediction curve). This is how you will build your machine learning intuition!

So, play around and build some models. When you are done playing with hyperparameter values, you should try to set these values to the optimal values manually (you're likely going to be *way* off). Then, retrain the model. Next in this assignment, we will perform several grid searches, so you'll be able to compare your "optimal" hyperparameter values with those computed from the grid search.

We will visualize model predictions for the optimal `Gradient-Boosted Regression Tree` , a `Random Forest` , and `Decision Tree` models that were determined by the grid searches. Next, you will compute the generalization error on the test set for the three models.

Preliminaries

Let's import some common packages:

In [1]:

```python
# Common imports
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import pandas as pd
%matplotlib inline
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
import os

# Where to save the figures
PROJECT_ROOT_DIR = "."
FOLDER = "figures"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, FOLDER)
os.makedirs(IMAGES_PATH, exist_ok=True)


times_font = {'fontname': 'Times New Roman', 'size': 18}


def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)

def plot3Ddata(data_df):
    fig = plt.figure(figsize = (17, 17))

    scat_x = X_train[:, [0]]
    scat_y = X_train[:,[1]]
    scat_z = z_train

    ax = fig.add_subplot(2,2,1, projection = '3d')
    ax.scatter3D(scat_x, scat_y, scat_z)
    plt.xlim(0,14)
    plt.ylim(-6,6)
    ax.set_xlabel('x', color = 'maroon', **times_font)
    ax.set_ylabel('y', color = 'maroon', **times_font)
    ax.set_zlabel('z', color = 'maroon', **times_font)
    ax.view_init(0, 90)
```

```
    ax = fig.add_subplot(2,2,2, projection = '3d')
    ax.scatter3D(scat_x, scat_y, scat_z)
    plt.xlim(0,14)
    plt.ylim(-6,6)
    ax.set_xlabel('x', color = 'maroon', **times_font)
    ax.set_ylabel('y', color = 'maroon', **times_font)
    ax.set_zlabel('z', color = 'maroon', **times_font)
    ax.view_init(35, 0)

    ax = fig.add_subplot(2,2,3, projection = '3d')
    ax.scatter3D(scat_x, scat_y, scat_z)
    plt.xlim(0,14)
    plt.ylim(-6,6)
    ax.set_xlabel('x', color = 'maroon', **times_font)
    ax.set_ylabel('y', color = 'maroon', **times_font)
    ax.set_zlabel('z', color = 'maroon', **times_font)
    ax.view_init(35, 45)

    ax = fig.add_subplot(2,2,4, projection = '3d')
    ax.scatter3D(scat_x, scat_y, scat_z)
    plt.xlim(0,14)
    plt.ylim(-6,6)
    ax.set_xlabel('x', color = 'maroon', **times_font)
    ax.set_ylabel('y', color = 'maroon', **times_font)
    ax.set_zlabel('z', color = 'maroon', **times_font)
    ax.view_init(20, 20)

    plt.show()

def plotscatter3Ddata(fit_x, fit_y, fit_z, scat_x, scat_y, scat_z):
    scat_x = X_train[:, [0]]
    scat_x = scat_x.flatten()

    scat_y = X_train[:,[1]]
    scat_y = scat_y.flatten()

    scat_z = z_train.reshape(100,1)
    scat_z = scat_z.flatten()

    fit_x = scat_x

    fit_y = scat_y

    fit_z = fit_z
```

```
line = pd.DataFrame({'x': fit_x,'y': fit_y, 'z': fit_z}, columns = ['x', 'y', 'z'])

line = line.sort_values('x')

fig = plt.figure(figsize = (16, 16))

ax = fig.add_subplot(2,2,1, projection = '3d')
ax.scatter3D(scat_x, scat_y, scat_z)
plt.xlim(0,14)
plt.ylim(-6,6)
ax.set_xlabel('x', color = 'maroon', **times_font)
ax.set_ylabel('y', color = 'maroon', **times_font)
ax.set_zlabel('z', color = 'maroon', **times_font)
ax.plot3D(line['x'], line['y'], line['z'], color = 'black')
ax.view_init(0, 90)

ax = fig.add_subplot(2,2,2, projection = '3d')
ax.scatter3D(scat_x, scat_y, scat_z)
plt.xlim(0,14)
plt.ylim(-6,6)
ax.set_xlabel('x', color = 'maroon', **times_font)
ax.set_ylabel('y', color = 'maroon', **times_font)
ax.set_zlabel('z', color = 'maroon', **times_font)
ax.plot3D(line['x'], line['y'], line['z'], color = 'black')
ax.view_init(35, 0)

ax = fig.add_subplot(2,2,3, projection = '3d')
ax.scatter3D(scat_x, scat_y, scat_z)
plt.xlim(0,14)
plt.ylim(-6,6)
ax.set_xlabel('x', color = 'maroon', **times_font)
ax.set_ylabel('y', color = 'maroon', **times_font)
ax.set_zlabel('z', color = 'maroon', **times_font)
ax.plot3D(line['x'], line['y'], line['z'], color = 'black')
ax.view_init(35, 45)

ax = fig.add_subplot(2,2,4, projection = '3d')
ax.scatter3D(scat_x, scat_y, scat_z)
plt.xlim(0,14)
plt.ylim(-6,6)
ax.set_xlabel('x', color = 'maroon', **times_font)
ax.set_ylabel('y', color = 'maroon', **times_font)
ax.set_zlabel('z', color = 'maroon', **times_font)
ax.plot3D(line['x'], line['y'], line['z'], color = 'black')
```

```
    ax.view_init(20, 20)

    plt.show()
```

Import and Split Data

Complete the following:

1. Begin by importing the data from the file called `3DSinusoidal.csv`. Name the returned DataFrame `data`.
2. Call train_test_split() with a `test_size` of 20%. `x` and `y` will be your feature data and `z` will be your response data. Save the output into `X_train`, `X_test`, `z_train`, and `z_test`, respectively. Specify the `random_state` parameter to be `42` (do this throughout the entire note book).

In [2]:
```
from sklearn.model_selection import train_test_split

data = pd.read_csv("3DSinusoidal.csv")

X = data[['x', 'y']]

z = data[['z']]

X_train, X_test, z_train, z_test = train_test_split(X, z, test_size = 0.2, random_state = 42)

# Reshape X / Z Data and Make NumPy Arrays
X_train = np.array(X_train).reshape(-2,2)
X_test = np.array(X_test).reshape(-1,2)
z_train = np.array(z_train)
z_test = np.array(z_test)
```

In [25]:
```
X_train.shape
```

Out[25]:
```
(100, 2)
```

In [26]:
```
z_train.shape
```

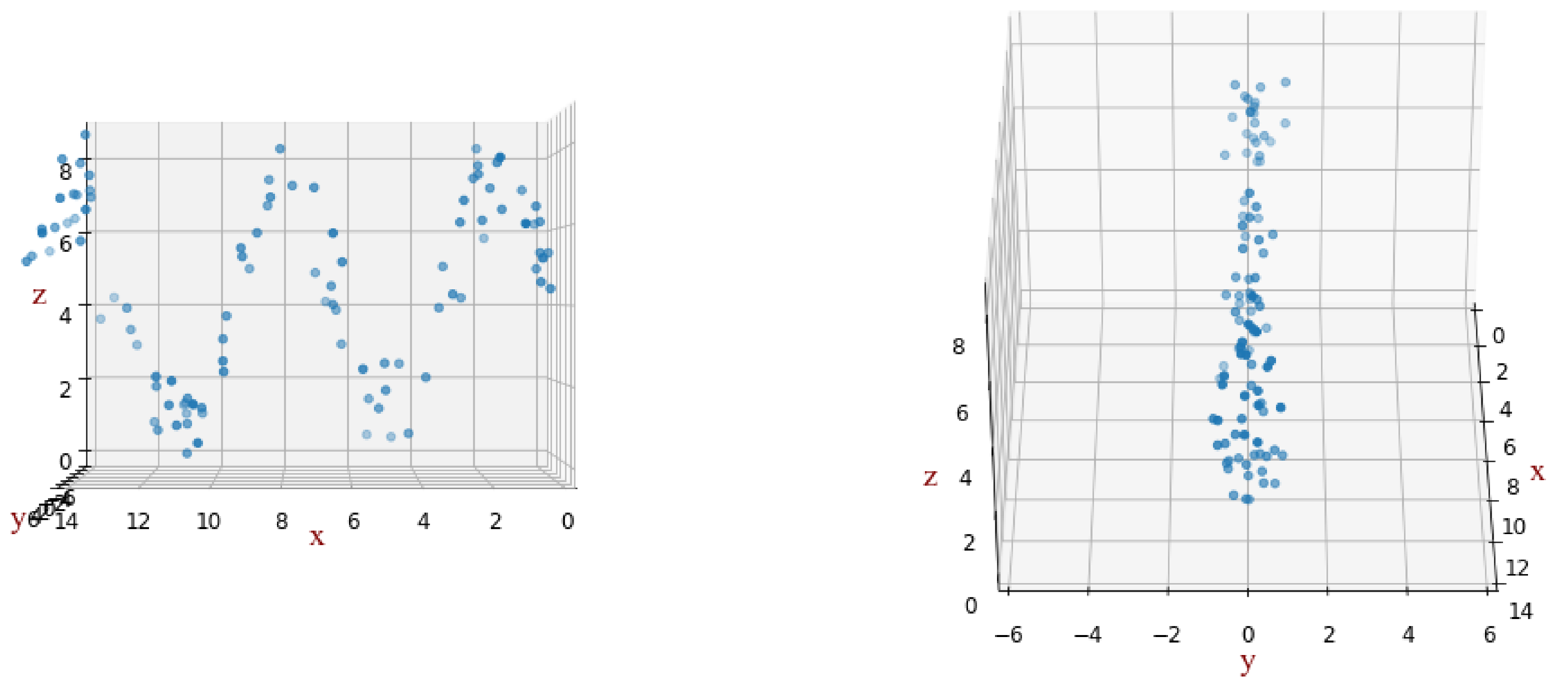
Out[26]:
```
(100, 1)
```

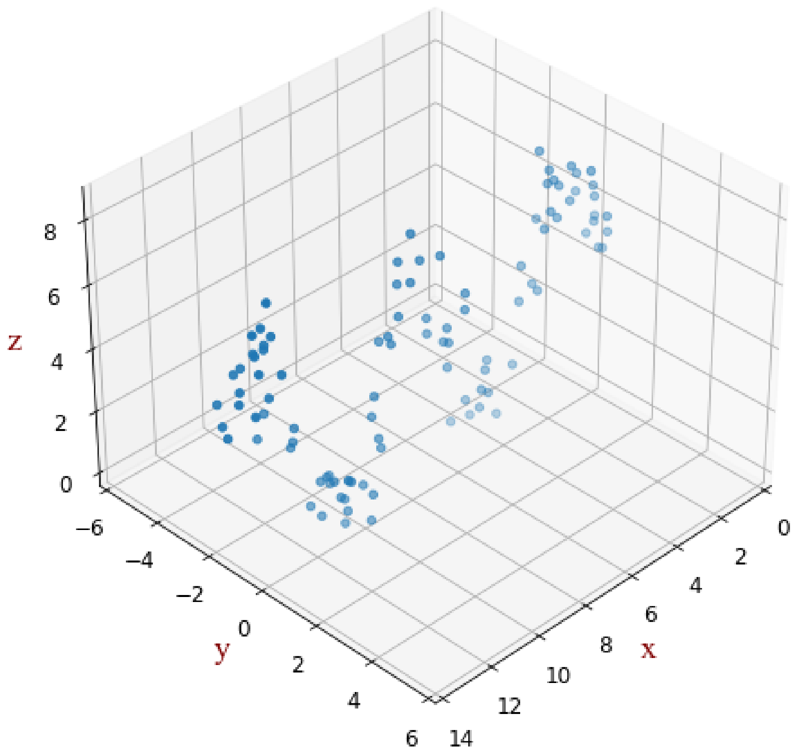
Plot Data

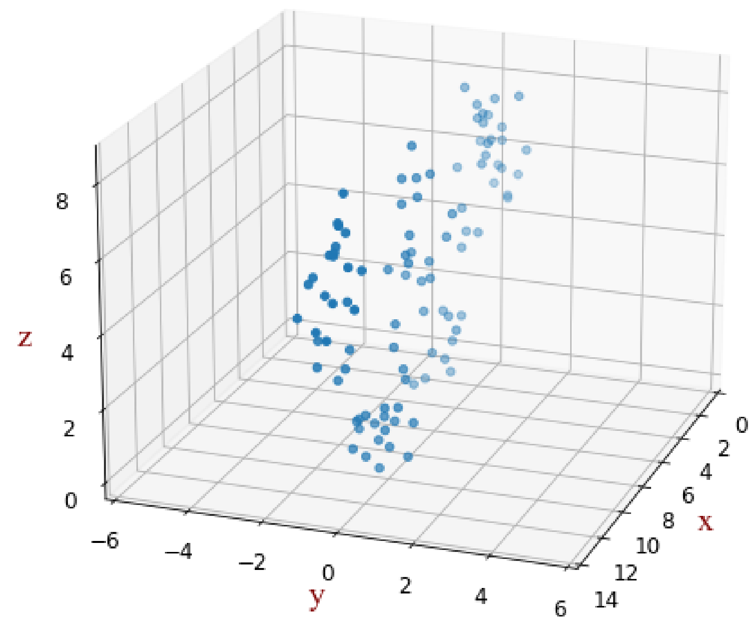
Simply plot your training data here, so that you know what you are working with. You must define a function called `plot3Ddata`, which accepts a Pandas DataFrame (composed of 3 spatial coordinates) and uses `scatter3D()` to plot the data. Use this function to plot only the training data (recall that you don't even want to look at the test set, until you are ready to calculate the generalization error). You must place the definition of this function in the existing code cell of the above **Preliminaries** section, and have nothing other than the function invocation in the below cell.

You must emulate the graphs shown in the respective sections below. Each of the graphs will have four subplots. Note the various viewing angles that each subplot presents - you can achieve this with the view_init() method. Be sure to label your axes as shown.

In [3]:
```
train_df = [X_train, z_train]

plot3Ddata(train_df)
```







A Quick Note

In the following sections you will be asked to plot the training data along with the model's predictions for that data superimposed on it. You must write a function called `plotscatter3Ddata(fit_x, fit_y, fit_z, scat_x, scat_y, scat_z)` that will plot this figure. The function accepts six parameters as input, shown in the function signature. All six input parameters must be NumPy arrays. The Numpy arrays called fit_x and fit_y represent the x and y coordinates from the training data and fit_z represents the model predictions from those coordinates (i.e. the prediction curve). The three Numpy arrays called `scat_x`, `scat_y`, and `scat_z` represent the x, y, and z coordinates of the training data.

You must place the definition of the `plotscatter3Ddata(fit_x, fit_y, fit_z, scat_x, scat_y, scat_z)` function in the existing code cell of the above **Preliminaries** section. (The function header is already there - you must complete the function definition.)

You will use the `plotscatter3Ddata()` function in each of the below **Plot Model Predictions for Training Set** portion of the three **Explore 3D Data** sections, as well as the **Visualize Optimal Model Predictions** section.

Important: Below, you will be asked to plot the model's prediction curve along with the training data. Even if you correctly train the model, you may find that your trendline is very ugly when you first plot it. If this happens to you, try plotting the model's predictions using a scatter plot rather than a connected line plot. You should be able to infer the problem and solution with the trendline from examining this new scatter plot of the model's predictions.

Explore 3D Data: GradientBoostingRegressor

Fit a `GradientBoostingRegressor` model to this data. You must manually assign values to the following hyperparameters. You should "play around" by using different combinations of hyperparameter values to really get a feel for how they affect the model's predictions. When you are done playing, set these to the best values you can for submission. (It is totally fine if you don't elucidate the optimal values here; however, you will want to make sure your model is not excessively overfitting or underfitting the data. Do this by examining the prediction curve generated by your model. You will be graded, more exactly, on the values that you calculate later from performing several rounds of grid searches.)

- `learning_rate = <value>`
- `max_depth = <value>`
- `n_estimators = <value>`
- `random_state = 42`

In [28]:
```
X_train.shape
```

Out[28]: (100, 2)

In [29]:
```
z_train.shape
```

Out[29]: (100, 1)

In [4]:
```
from sklearn.ensemble import GradientBoostingRegressor

gbrt = GradientBoostingRegressor(learning_rate = 0.2, max_depth = 1, n_estimators = 400, random_state = 42)

z_train = np.ravel(z_train)

gbrt.fit(X_train,z_train)
```

```
gbrt_predict = gbrt.predict(X_train)
```

Plot Model Predictions for Training Set

Use the `plotscatter3Ddata(fit_x, fit_y, fit_z, scat_x, scat_y, scat_z)` function to plot the data and the prediction curve.

In [59]:
```
# scat_x = X_train[:, [0]]
# scat_x = scat_x.flatten()

# scat_y = X_train[:,[1]]
# scat_y = scat_y.flatten()

# scat_z = z_train.reshape(100,1)
# scat_z = scat_z.flatten()

# fit_x = scat_x

# fit_y = scat_y

#fit_z = gbrt.predict(X_train)
#fit_z = fit_z.reshape(100,1)
#fit_z = fit_z.flatten()


#fit_x, fit_y, fit_z = zip(*sorted(zip(fit_x, fit_y, fit_z)))

#x_fit = np.linspace(0,21,1000)
#y_fit = x_fit

#x_fit = np.linspace(0,21,1000)
#y_fit = x_fit

#fit_z = gbrt.predict(X_train)

# fit_z = fit_z.predict(X_train)
# fit_z = fit_z.reshape(100,1)
# fit_z = fit_z.flatten()

# line = pd.DataFrame({'x': fit_x,'y': fit_y, 'z': fit_z}, columns = ['x', 'y', 'z'])
# line = line.sort_values('x')

#fit_z = np.sort(fit_z)
```

```
# X_train = X_train.reshape(100,)
# X_train = X_train.flatten()

# fit_z = np.sort(X_train[:, [0])

# zipped = zip(fit_x, fit_y, fit_z)
# sorted_zipped = np.sort(zipped)
# fit_x, fit_y, fit_z = zip(sorted_zipped)




# array = X_train, z_train
# sort_f = lambda x: (x[0][[:,0]], x[1])
# sorted_array = sorted(array, key = sort_f)



#s_array = np.sort(array[0])


#>>> l = [(0, 5, 1), (1, 3, 4), (0, -3, 1), (1, 3, 5)]
#>>> sorter = lambda x: (x[1], x[0], x[2])
#>>> sorted_l = sorted(l, key=sorter)


#s_array = zip(*sorted(zip(fit_x, fit_y, fit_z)))
#X_train, z_train = zip(*sorted(zip(fit_x, fit_y, fit_z)))
#fit_z = X_train[np.argsort(X_train)]
#fit_z = np.sort(fit_z)
#fit_z = np.sort(fit_z, axis = 0)
#fit_x = np.argsort(fit_x)
#fit_y = np.argsort(fit_y)
#fit_z = np.argsort(fit_z)
#fit_z = np.sort(fit_z)
```

In [10]:

```
#data = [scat_x, scat_y, scat_z]
#data
```

In [11]:

```
#x = data[:,0]
#x
```

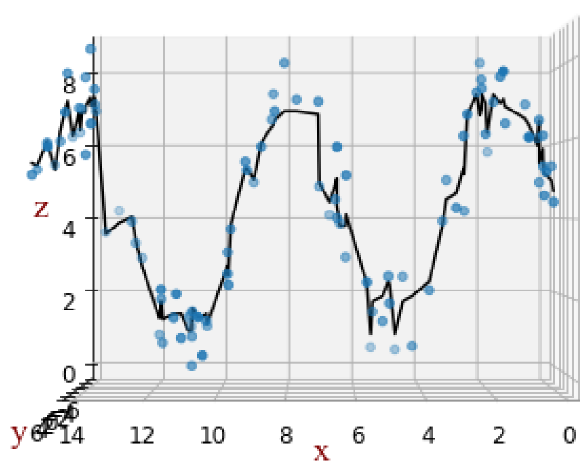
In [21]:
```
#fit_z = fit_z.reshape(100,1)
```

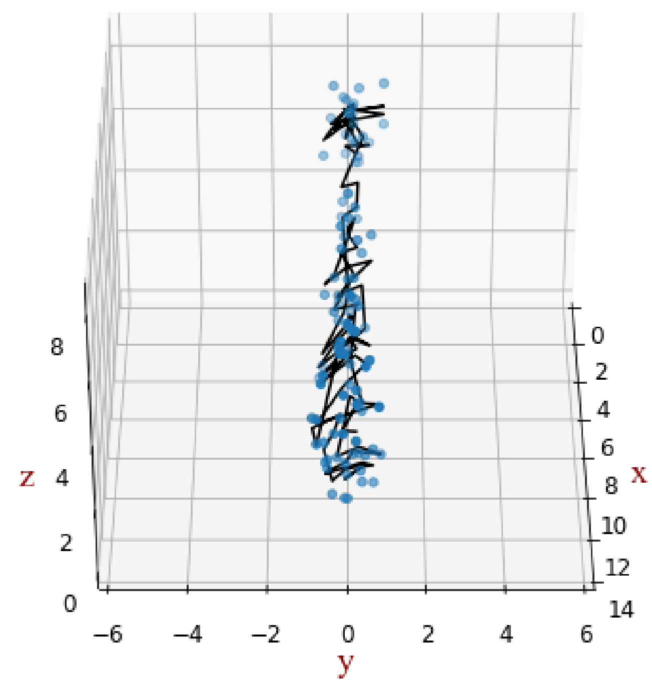
In [22]:
```
#fit_z.shape
```

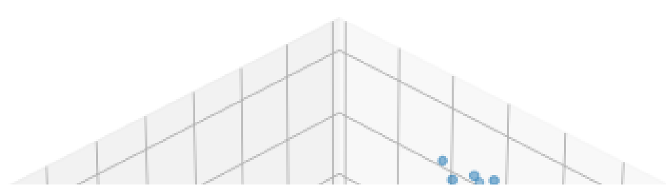
In [19]:
```
#scat_z = scat_z.reshape(100,1)
```

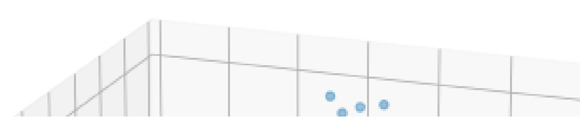
In [20]:
```
#scat_z.shape
```

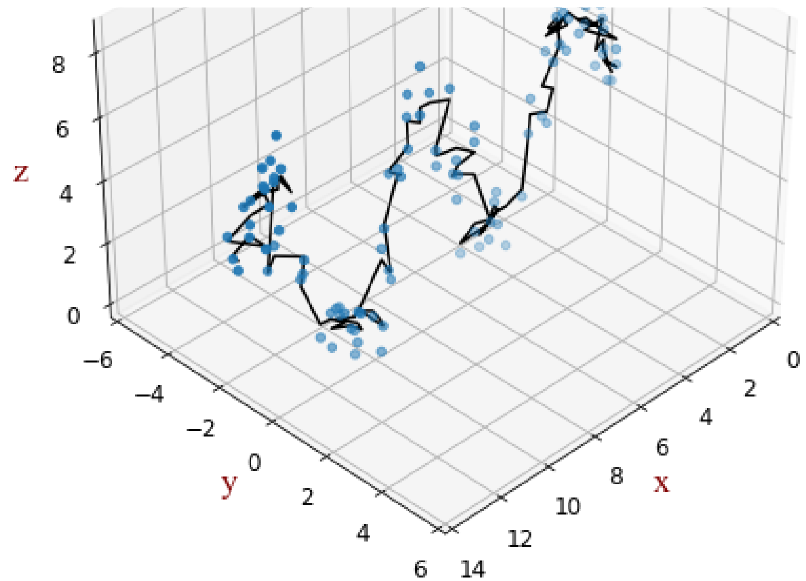
In [9]:
```
plotscatter3Ddata(X_train, X_train, gbrt_predict, X, X, z)
```

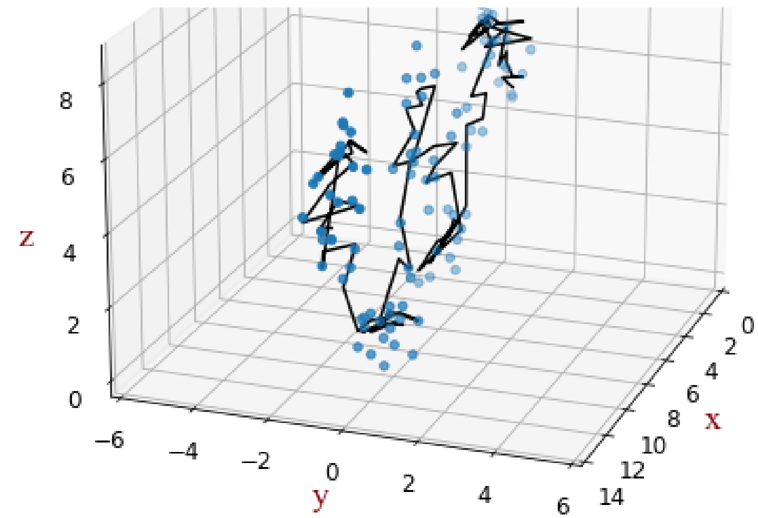












Explore 3D Data: RandomForestRegressor

Fit a `RandomForestRegressor` model to this data. You must manually assign values to the following hyperparameters. You should "play around" by using different combinations of hyperparameter values to really get a feel for how they affect the model's predictions. When you are done playing, set these to the best values you can for submission. (It is totally fine if you don't elucidate the optimal values here; however, you will want to make sure your model is not excessively overfitting or underfitting the data. Do this by examining the prediction curve generated by your model. You will be graded, more exactly, on the values that you calculate later from performing several rounds of grid searches.)

- `min_samples_split = <value>`
- `max_depth = <value>`
- `n_estimators = <value>`
- `random_state = 42`

In [10]:

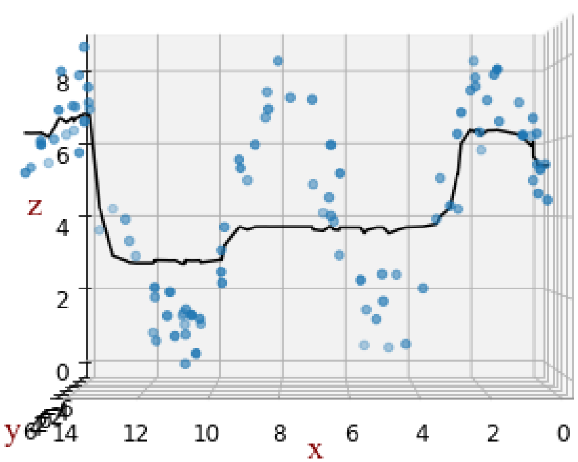
```
from sklearn.ensemble import RandomForestRegressor

rft_reg = RandomForestRegressor(min_samples_split = 4, max_depth = 2, n_estimators = 200, random_state = 42)

rft_reg.fit(X_train,z_train)
```

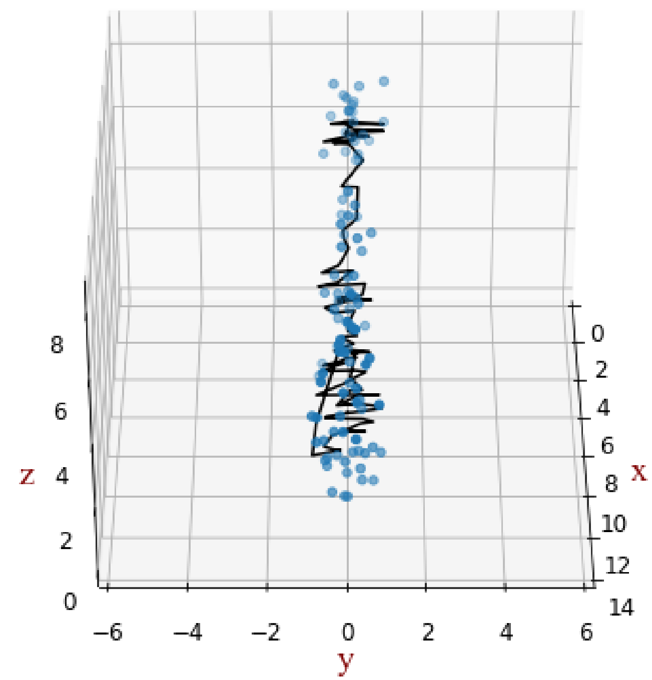
```
rft_predict = rft_reg.predict(X_train)
```

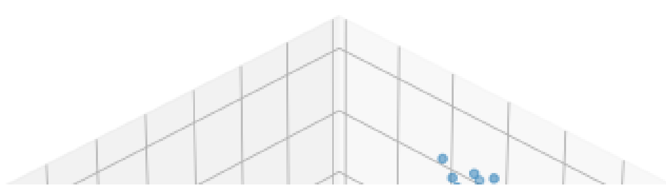
Plot Model Predictions for Training Set

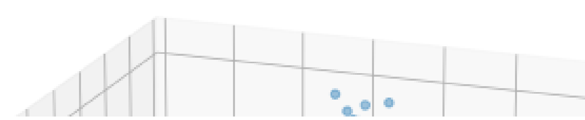
Use the `plotscatter3Ddata(fit_x, fit_y, fit_z, scat_x, scat_y, scat_z)` function to plot the data and the prediction curve.

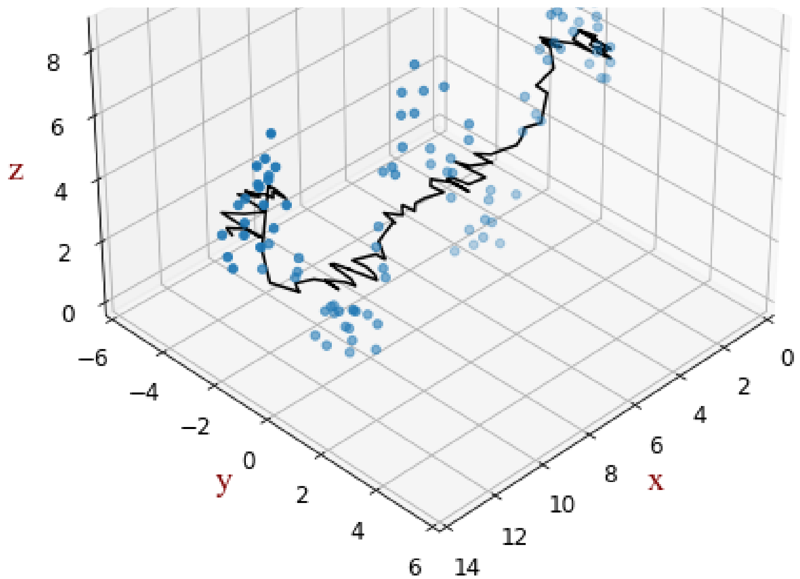
In [11]:
```
plotscatter3Ddata(X_train, X_train, rft_predict, X, X, z)
```

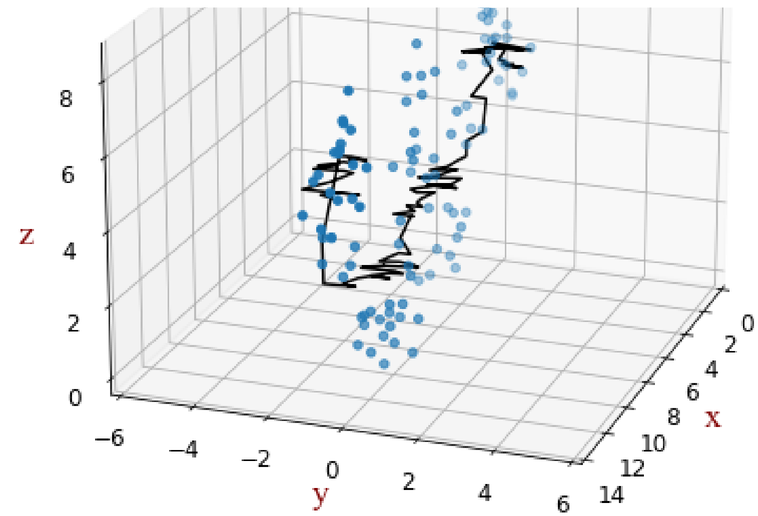












Explore 3D Data: DecisionTreeRegressor

Fit a `DecisionTreeRegressor` model to this data. You must manually assign values to the following hyperparameters. You should "play around" by using different combinations of hyperparameter values to really get a feel for how they affect the model's predictions. When you are done playing, set these to the best values you can for submission. (It is totally fine if you don't elucidate the optimal values here; however, you will want to make sure your model is not excessively overfitting or underfitting the data. Do this by examining the prediction curve generated by your model. You will be graded, more exactly, on the values that you calculate later from performing several rounds of grid searches.)

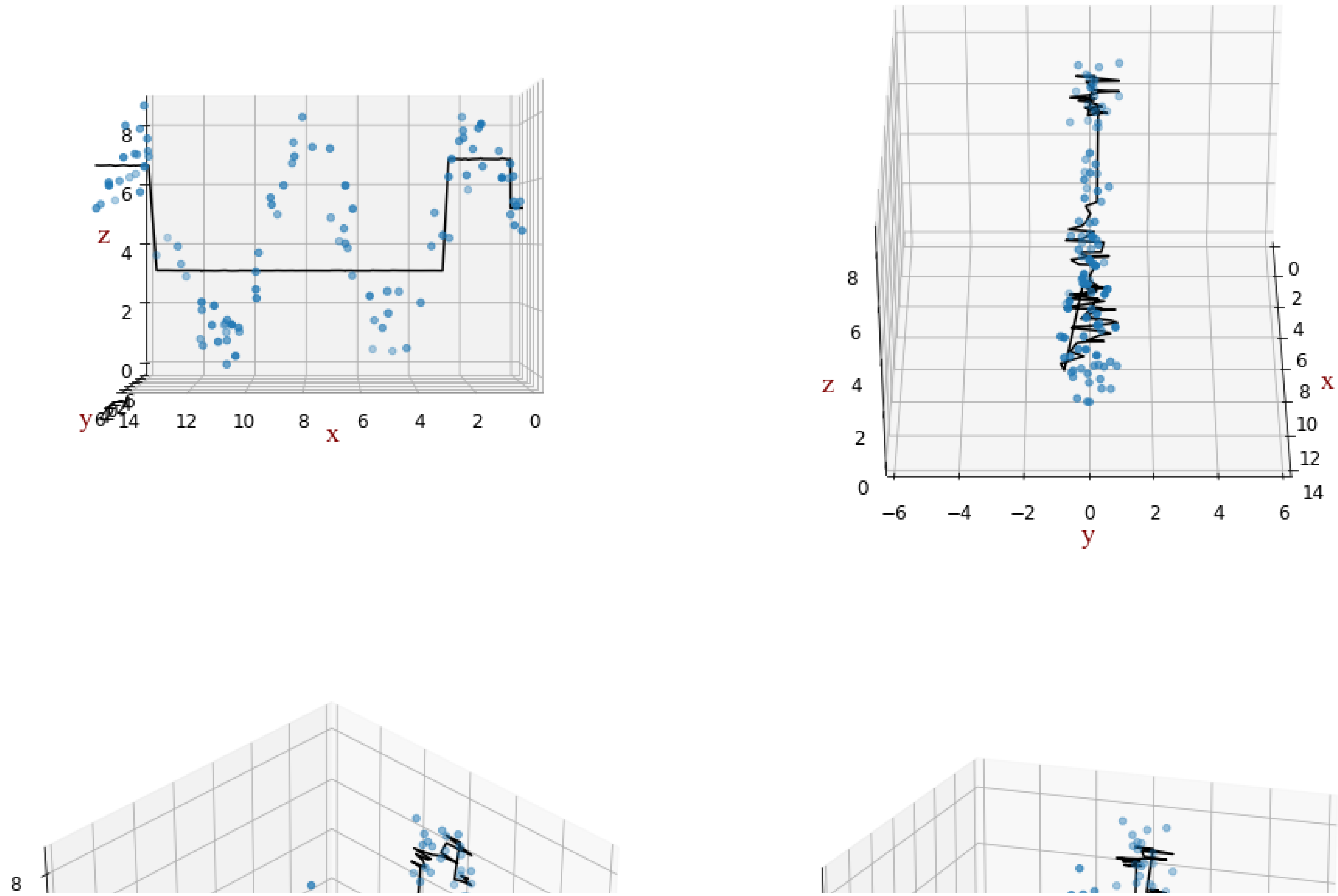
- `splitter = <value>`
- `max_depth = <value>`
- `min_samples_split = <value>`
- `random_state = 42`

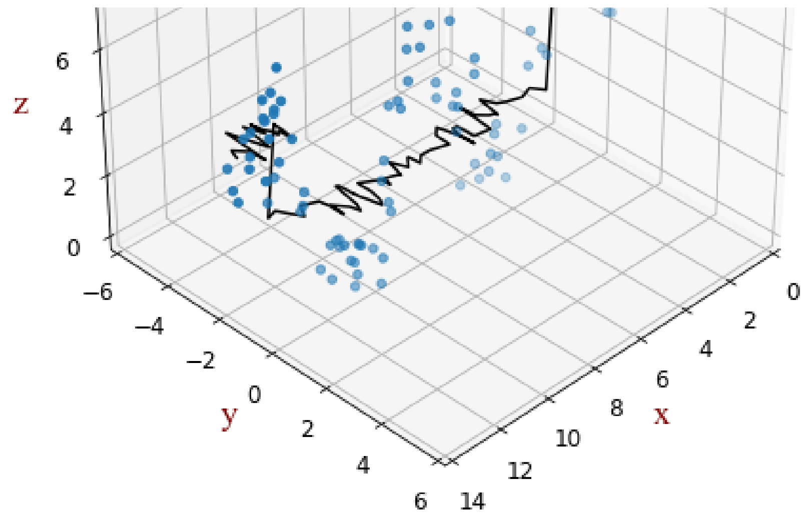
In [12]:
```
from sklearn.tree import DecisionTreeRegressor

dtree_reg = DecisionTreeRegressor(splitter = "best", max_depth = 2, min_samples_split = 2, random_state = 42)

dtree_reg.fit(X_train,z_train)

dtree_predict = dtree_reg.predict(X_train)
```

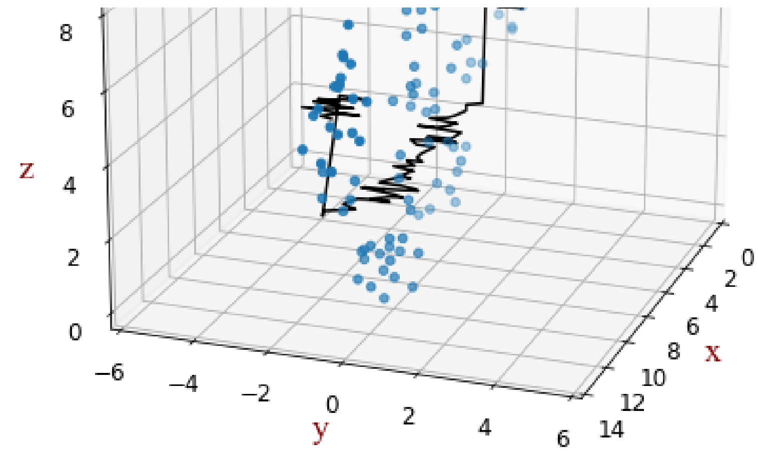
Plot Model Predictions for Training Set

Use the `plotscatter3Ddata(fit_x, fit_y, fit_z, scat_x, scat_y, scat_z)` function to plot the data and the prediction curve.

In [13]:
```
plotscatter3Ddata(X_train, X_train, dtree_predict, X, X, z)
```







Perform Grid Searches

You will perform a series of grid searches, which will yield the optimal hyperparamter values for each of the three model types. You can compare the values computed by the grid search with the values you manually found earlier. How do these compare?

You must perform a course-grained grid search, with a very broad range of values first. Then, you perform a second grid search using a tighter range of values centered on those identified in the first grid search. You may have to use another round of grid searching too (it took me at least three rounds of grid searches per model to ascertain the optimal hyperparameter values below).

Note the following:

1. Be sure to clearly report the optimal hyperparameters in the designated location after you calculate them!
2. You must use `random_state=42` everywhere that it is needed in this notebook.
3. You must use grid search to compute the following hyperparameters:

   GradientBoostingRegressor:

   - max_depth = <value>
   - n_estimators = <value>
   - learning_rate = <value>

RandomForestRegressor:

- `max_depth = <value>`
- `n_estimators = <value>`
- `min_samples_split = <value>`

DecisionTreeRegressor:

- `splitter = <value>`
- `max_depth = <value>`
- `min_samples_split = <value>`

1. `learning rate` should be rounded to two decimals.
2. The number of cross-folds. Specify `cv=3`

Perform Individual Model Grid Searches

In this section you will perform a series of grid searches to compute the optimal hyperparameter values for each of the three model types.

In [73]:
```python
from sklearn.model_selection import GridSearchCV

# -----
# Coarse-Grained GradientBoostingRegressor GridSearch
# -----


param_grid = {'learning_rate': [0.2, 0.4, 0.6, 0.8, 1.0],
              'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 20, 25, 32],
              'n_estimators': [100,200,300,400,500,600,700,800,900,1000]
              }

grid_search_cv = GridSearchCV(GradientBoostingRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, z_train)

print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 550 candidates, totalling 1650 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
The best parameters are:  {'learning_rate': 0.2, 'max_depth': 2, 'n_estimators': 100}
[Parallel(n_jobs=1)]: Done 1650 out of 1650 | elapsed:  3.9min finished
```

In [74]:

```python
# -----
# Refined GradientBoostingRegressor GridSearch
# -----

param_grid = {'learning_rate': [0.1, 0.2, 0.3],
              'max_depth': [1, 2, 3],
              'n_estimators': [60, 70, 80, 90, 100, 110, 120, 130, 140]
              }

grid_search_cv = GridSearchCV(GradientBoostingRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, z_train)

print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 81 candidates, totalling 243 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
The best parameters are:  {'learning_rate': 0.2, 'max_depth': 2, 'n_estimators': 70}
[Parallel(n_jobs=1)]: Done 243 out of 243 | elapsed:    5.3s finished
```

In [75]:

```python
# -----
# Refined GradientBoostingRegressor GridSearch (round 2)
# -----

param_grid = {'learning_rate': [0.16, 0.18, 0.2, 0.22, 0.24],
               'max_depth': [2],
               'n_estimators': [65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75]
              }

grid_search_cv = GridSearchCV(GradientBoostingRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, z_train)

print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 55 candidates, totalling 165 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
The best parameters are:  {'learning_rate': 0.22, 'max_depth': 2, 'n_estimators': 74}
[Parallel(n_jobs=1)]: Done 165 out of 165 | elapsed:    2.2s finished
```

In [77]:
```python
# -----
# Final GradientBoostingRegressor GridSearch
# -----

param_grid = {'learning_rate': [0.2, 0.21, 0.22, 0.23, 0.24],
              'max_depth': [2],
              'n_estimators': [71, 72, 73, 74, 75, 76, 77]
             }

grid_search_cv = GridSearchCV(GradientBoostingRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, z_train)

print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 35 candidates, totalling 105 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
The best parameters are:  {'learning_rate': 0.22, 'max_depth': 2, 'n_estimators': 74}
[Parallel(n_jobs=1)]: Done 105 out of 105 | elapsed:    1.6s finished
```

On this dataset, the optimal model parameters for the `GradientBoostingRegressor` class are:

- `learning_rate = 0.22`
- `max_depth = 2`
- `n_estimators = 74`

In [78]:
```python
# -----
# Coarse-Grained RandomForestRegressor GridSearch
# -----

param_grid = {'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 20, 25, 32],
              'n_estimators': [100,200,300,400,500,600,700,800,900,1000],
              'min_samples_split': [2, 4, 8, 10, 12, 14, 18, 20]
             }

grid_search_cv = GridSearchCV(RandomForestRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, z_train)

print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 880 candidates, totalling 2640 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[Parallel(n_jobs=1)]: Done 2640 out of 2640 | elapsed: 27.0min finished
The best parameters are:  {'max_depth': 12, 'min_samples_split': 2, 'n_estimators': 900}
```

In [83]:

```
# -----
# Refined RandomForestRegressor GridSearch
# -----

param_grid = {'max_depth': [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17],
              'n_estimators': [850, 860, 870, 880, 890, 900, 910, 920, 930, 940, 950],
              'min_samples_split': [2]
             }

grid_search_cv = GridSearchCV(RandomForestRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, z_train)

print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 121 candidates, totalling 363 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 363 out of 363 | elapsed:  5.6min finished
The best parameters are:  {'max_depth': 11, 'min_samples_split': 2, 'n_estimators': 890}
```

In [84]:

```
# -----
# Final RandomForestRegressor GridSearch
# -----

param_grid = {'max_depth': [10, 11, 12],
              'n_estimators': [881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893,
                               894, 895, 896, 897, 898, 899],
              'min_samples_split': [2]
             }

grid_search_cv = GridSearchCV(RandomForestRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, z_train)

print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 57 candidates, totalling 171 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 171 out of 171 | elapsed:  2.7min finished
The best parameters are:  {'max_depth': 11, 'min_samples_split': 2, 'n_estimators': 885}
```

On this dataset, the optimal model parameters for the `RandomForestRegressor` class are:

- `max_depth = 11`
- `n_estimators = 885`
- `min_samples_split = 2`

In [85]:

```python
# -----
# Coarse-Grained DecisionTreeRegressor GridSearch
# -----

param_grid = {'splitter': ['best', 'random'],
              'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 20, 25, 32],
              'min_samples_split': [2, 4, 8, 10, 12, 14, 18, 20]
              }

grid_search_cv = GridSearchCV(DecisionTreeRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, z_train)

print('The best parameters are: ', grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 176 candidates, totalling 528 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
The best parameters are:  {'max_depth': 8, 'min_samples_split': 10, 'splitter': 'best'}
[Parallel(n_jobs=1)]: Done 528 out of 528 | elapsed:    0.6s finished
```

In [86]:

```python
# -----
# Refined DecisionTreeRegressor GridSearch
# -----

param_grid = {'splitter': ['best'],
              'max_depth': [6, 7, 8, 9, 10],
              'min_samples_split': [8, 9, 10, 11, 12]
              }

grid_search_cv = GridSearchCV(DecisionTreeRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, z_train)

print('The best parameters are: ', grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 25 candidates, totalling 75 fits
```

```
The best parameters are:  {'max_depth': 7, 'min_samples_split': 10, 'splitter': 'best'}
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  75 out of  75 | elapsed:    0.1s finished
```

In [87]:

```
# -----
# Final-Grained DecisionTreeRegressor GridSearch
# -----

param_grid = {'splitter': ['best'],
              'max_depth': [6, 7, 8],
              'min_samples_split': [10]
              }

grid_search_cv = GridSearchCV(DecisionTreeRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, z_train)

print('The best parameters are: ', grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 3 candidates, totalling 9 fits
The best parameters are:  {'max_depth': 7, 'min_samples_split': 10, 'splitter': 'best'}
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    0.0s finished
```

On this dataset, the optimal model parameters for the `RandomForestRegressor` class are:

- `splitter = 'best'`
- `max_depth = 7`
- `min_samples_split = 10`

Visualize Optimal Model Predictions

In the previous section you performed a series of grid searches designed to identify the optimal hyperparameter values for all three models. Now, use the `best_params_` attribute of the grid search objects from above to create the three optimal models below. For each model, visualize the models predictions on the training set - this is what we mean by the "prediction curve" of the model.

Create Optimal GradientBoostingRegressor Model

In [ ]:

```
optimal_gbrt = GradientBoostingRegressor(learning_rate = 0.22, max_depth = 2, n_estimators = 74, random_state =
```

```
#optimal_gbrt.fit(X_train, y_train)
```

Plot Model Predictions for Training Set

Use the `plotscatter3Ddata(fit_x, fit_y, fit_z, scat_x, scat_y, scat_z)` function to plot the data and the prediction curve.

In [ ]:
```
plotscatter3Ddata(fit_x, fit_y, fit_z, scat_x, scat_y, scat_z)
```

Create Optimal RandomForestRegressor Model

In [ ]:
```
### ENTER CODE HERE ###
```

Plot Model Predictions for Training Set

Use the `plotscatter3Ddata(fit_x, fit_y, fit_z, scat_x, scat_y, scat_z)` function to plot the data and the prediction curve.

In [ ]:
```
### ENTER CODE HERE ###
```

Create Optimal DecisionTreeRegressor Model

In [ ]:
```
### ENTER CODE HERE ###
```

Plot Model Predictions for Training Set

Use the `plotscatter3Ddata(fit_x, fit_y, fit_z, scat_x, scat_y, scat_z)` function to plot the data and the prediction curve.

In [ ]:
```
### ENTER CODE HERE ###
```

Compute Generalization Error

Compute the generalization error for each of the optimal models computed above. Use MSE as the generalization error metric. Round your answers to four significant digits. Print the generalization error for all three models.

In [ ]:
```
# from sklearn.metrics import mean_squared_error

# z_pred1 = optimal_gbrt.predict(X_test)

# z_pred2 = optimal_rft_reg.predict(X_test)

# z_pred3 = optimal_dtree_reg.predict(X_test)

# gbrt_mse = mean_squared_error(z_test, z_pred1)

# rf_mse = mean_squared_error(z_test, z_pred2)

# dtree_mse = mean_squared_error(z_test, z_pred3)

# print('GradientBoostingRegressor Model MSE', round(gbrt_mse, 4))
# print('RandomForestRegressor Model MSE', round(rf_mse, 4))
# print('DecisionTreeRegressor Model MSE', round(dtree_mse, 4))
```