Assignment 1: NCG613

Student Name: Susan Edgeworth

Student Number: 17704655

This report uses Python code that applies a cross-validated (k)-nearest neighbour model from the house price data, and produces the following:

· A 3-D plot of the house price for average floor area.

· A 3-D plot of the house price for floor areas of 75 square metres.

· A 3-D plot of the house price for floor areas of 125 square meters

A step by step approach is taken here, each code snippet has an explanation and method for each practice. This is a supervised learning algorithm used to predict house price in the London area, using the predictor variables, east, north and fl_area.

Here the data is being read in with the required packages and the csv file containing the house price data:

In [172]:

```
import numpy as np
```

In [173]:

```
import pandas as pd
```

In [174]:

```
hp = pd.read_csv('hpdemo.csv',dtype=float)
```

In [175]:

```
print (hp)
```

```
          ID      east     north     price  fl_area
0        1.0  523800.0  179700.0  107000.0     50.0
1        2.0  533200.0  170900.0   55500.0     66.0
2        3.0  514600.0  175800.0  103000.0     90.0
3        4.0  516000.0  171000.0  187000.0    125.0
4        5.0  533700.0  169200.0   43000.0     50.0
...      ...       ...       ...       ...      ...
1400  1401.0  515600.0  173100.0   68500.0     44.0
1401  1402.0  513200.0  186500.0   58500.0     59.0
1402  1403.0  542900.0  189500.0  247000.0    185.0
1403  1404.0  524900.0  185300.0  153000.0     96.0
1404  1405.0  522000.0  185400.0  146250.0    111.0

[1405 rows x 5 columns]
```

Using the sklearn package to scale the variables

In this section the data is being scaled using the sklearn package. The x_scaler here is a function of sorts, set up to implement the scaling for particular variables. Here the x_scaler.fit for the east north and fl_area works on scaling these predictor variables. The fit portion is used to calibrate the scale here. The reasons for scaling the data in this case is due to the varied units of measure ie fl_area is in square units and the east and north variables are in metres.

In [176]:

```
from sklearn.preprocessing import StandardScaler
x_scaler = StandardScaler()
```

In [177]:

```
x_scaler.fit(hp[['east','north','fl_area']])
```

Out[177]:

```
StandardScaler()
```

The data is then transformed to z-scores and then checked:

In [178]:

```
X = x_scaler.transform(hp[['east','north','fl_area']])
```

In [179]:

```
print(X[:6,:])
```

```
[[-0.46109525 -0.0036912  -1.16501944]
 [ 0.39051366 -1.02696462 -0.73029338]
 [-1.29458482 -0.45718737 -0.07820428]
 [-1.16774945 -1.01533651  0.87275899]
 [ 0.435812   -1.22464244 -1.16501944]
 [ 1.72228503  1.1474914   0.05764762]]
```

Choosing a Machine Learning algorithm and tuning the parameters

The next step is to choose what Machine Learning algorithm, for this assignment we will be using the k nearest neighbour model as we are seeking to predict the house prices for three distinct stipulations: price for the average floor area, price for the floor area being 75 square metres and price for floor area being 125 square metres. A tuning parameter will then be considered here, our predictor variables are east north and fl_area for the k nearest neighbour algorithm. Before choosing a tuning parameter we must;

· Determine the value of k, i.e., the nearest neighbours to look at.

· Calculate the distance, where to use a distance weighted mean.

· Determine which metric (distance) to use (Euclidean or Manhattan).

A note on Scoring Method

For predicted models where we have 'x' predictor variables and 'y' as the response variable, in this case x = east, north and fl_area and y = price. Our goal is to find an f(x) function that is as close as possible to our y. the f(x) relies on tuning parameters and some training data, assorting the tuning parameters modifies how close f(x) is to our y. The point is to identify which parameters, when combined reach the closest proximity to y. In this case we are combining our parameters east, north and fl_area to reach the closest price of house. The method to achieve this is through splitting our data into training and test form. We can then adjust f by using the training data along with a set of tuning parameters. From the test data we obtain x' and then compare this to y'. The scoring method then computes how near f(x') is to y'. Terms to describe the measurement of this score is with the 'root mean square error'(RMSE) or the mean absolute error (MAE). A small value for the RMSE and MAE imply a better prediction.

Optimisation of performance Score from tuning parameters

With our data a cross-validation grid search will be implemented through the sklearn package. This approach is a straightforward way of optimising the performance score.

Running the Algorithm

Here the kNeighborRegressors function is imported from the sklearn package. The reg_object gives an example of 6 nearest neighbors with a Euclidean distance metric and uniform mean weighting, it fits the regression and is used to predict the house price. The array 'price' is a data frame that is used with X to predict the house price given our scaled east, north and floor area. reg_object has now ‘learned’ to make predictions of the price of a house given its location and floor area.

In [180]:

```
from sklearn.neighbors import KNeighborsRegressor as NN
```

In [181]:

```
reg_object = NN(n_neighbors=6,weights='uniform',p=2)
```

In [182]:

```
price = hp['price']/1000.0
reg_object.fit(X,price)
```

Out[182]:

```
KNeighborsRegressor(n_neighbors=6)
```

Enhancing the Tuning Parameters

Having fit the model, we will now choose the tuning parameters for our model and use cross validation. The scoring method: mean absolute error (MEA) will be used here. The two imported functions from sklearn.metrics, make_scorer create the scoring object and mean_absolute_error designates the kind of scorer used. The scoring object created is called mae:

In [183]:

```
from sklearn.metrics import mean_absolute_error, make_scorer
mae = make_scorer(mean_absolute_error, greater_is_better=False)
```

GridSearchCV takes several arguments; estimator equals the specified ML algorithm, here it is the nearest neighbor method that we defined as NN earlier. Scoring is the mean absolute error and param_grid is a dictionary. The function works to combine items in the dictionary to be tested, and the combination with the best MAE will be selected. The results for both reg_object and GridSearchCV will be similar, GridSearchCV carries out a full cross-validation search rather than using a pre-specified set of tuning parameters.

In [184]:

```
from sklearn.model_selection import GridSearchCV
opt_nn = GridSearchCV(
    estimator = NN(),
    scoring = mae,
    param_grid = {
        'n_neighbors':range(1,35),
        'weights':['uniform','distance'],
        'p':[1,2]})
```

In [185]:

```
opt_nn.fit(X,price)
```

Out[185]:

```
GridSearchCV(estimator=KNeighborsRegressor(),
             param_grid={'n_neighbors': range(1, 35), 'p': [1, 2],
                         'weights': ['uniform', 'distance']},
             scoring=make_scorer(mean_absolute_error, greater_is_better=Fa
lse))
```

The Pipeline Function

Pipeline contains all the necessary components to implement a regression model and includes scaling to make predictions using the original data. The pipeline applies rescaling and then fits the model, so the original data is used in the pipeline approach, rather than the rescaled data.

In [186]:

```
pipe = Pipeline([('zscores',StandardScaler()),
    ('NNreg',NN(n_neighbors=6,weights='uniform',p=2))])
```

In [187]:

```
pipe.fit(hp[['east','north','fl_area']],price)
```

Out[187]:

```
Pipeline(steps=[('zscores', StandardScaler()),
                ('NNreg', KNeighborsRegressor(n_neighbors=6))])
```

Here we are setting up the nearest neighbors regression model:

In [188]:

```
pipe = Pipeline([('zscores',StandardScaler()),('NNreg',NN())])

opt_nn2 = GridSearchCV(
    estimator = pipe,
    scoring = mae,
    param_grid = {
        'NNreg__n_neighbors':range(1,35),
        'NNreg__weights':['uniform','distance'],
        'NNreg__p':[1,2]})
```

Here we are fitting the model with the parameters

In [189]:

```
opt_nn2.fit(hp[['east','north','fl_area']],price)
```

Out[189]:

```
GridSearchCV(estimator=Pipeline(steps=[('zscores', StandardScaler()),
                                       ('NNreg', KNeighborsRegressor())]),
             param_grid={'NNreg__n_neighbors': range(1, 35), 'NNreg__p':
[1, 2],
                         'NNreg__weights': ['uniform', 'distance']},
             scoring=make_scorer(mean_absolute_error, greater_is_better=Fa
lse))
```

Visualising the 3-D plots

To visualise the results a grid must be made. This code makes a grid of the eastings and northings within the data set and puts them into a 100 by 100 point grid. Using the numpty and meshgrid functions I implemented the code which gave me the grid over the required area:

In [190]:

```
east_mesh, north_mesh = np.meshgrid(
    np.linspace(505000,555800,100),
    np.linspace(158400,199900,100))
```

With there being a floor area, we must create an array that corresponds to the shape of our previous grid, this is an array with zeros with the average floor size.

In [191]:

```
fl_mesh = np.zeros_like(east_mesh)
```

In [192]:

```
fl_mesh[:,:] = np.mean(hp['fl_area'])
```

In [193]:

```
print(east_mesh.shape)
```

```
(100, 100)
```

In [194]:

```
print(north_mesh.shape)
```

```
(100, 100)
```

In [195]:

```
grid_predictor_vars = np.array([east_mesh.ravel(),
    north_mesh.ravel(),fl_mesh.ravel()]).T
```

In [196]:

```
hp_pred = opt_nn2.predict(grid_predictor_vars)
```

House prices here are unravelled so this snippet of code is putting it back to grid format done through the reshape function to make sure that it is the same shape as east_mesh:

In [197]:

```
hp_mesh = hp_pred.reshape(east_mesh.shape)
```

3-D graph for the Average Floor Area

For the average floor area this would constitute an average house size. In this 3-D plot the darkened peak is where the more expensive houses are located, the lighter areas are where the house prices are lower.

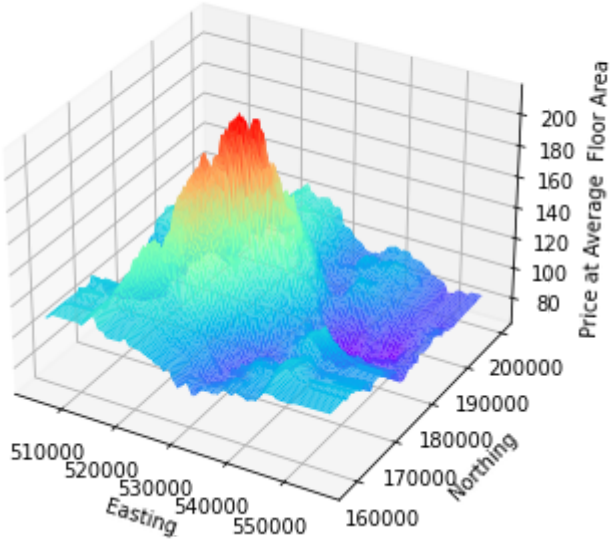
In [204]:

```
def surf3d(pipe_model,fl_area):
    east_mesh, north_mesh = np.meshgrid(
        np.linspace(505000,555800,100),
        np.linspace(158400,199900,100))
    fl_mesh = np.zeros_like(east_mesh)
    fl_mesh[:,:] = fl_area
    grid_predictor_vars = np.array([east_mesh.ravel(),
        north_mesh.ravel(),fl_mesh.ravel()]).T
    hp_pred = pipe_model.predict(grid_predictor_vars)
    hp_mesh = hp_pred.reshape(east_mesh.shape)
    fig = pl.figure()
    ax = Axes3D(fig)
    ax.plot_surface(east_mesh, north_mesh, hp_mesh,
        rstride=1, cstride=1, cmap='rainbow',lw=0.01)
    ax.set_title('3-D plot of the predicted house for Average Floor Area')
    ax.set_xlabel('Easting')
    ax.set_ylabel('Northing')
    ax.set_zlabel('Price at Average  Floor Area')
    pl.show()
    return
```

In [205]:

```
pl.close()
surf3d(opt_nn2,np.mean(hp['fl_area']))
pl.show()
```

3-D plot of the predicted house for Average Floor Area



3-D Graph for the Floor area at 75 Square Metres

Similar to the above graph the peaks here represent higher house prices. The peak is near the centre of the graph which means that the houses that are on the smaller scale towards the centre of London are more expensive which makes sense, as it is closer to the city.

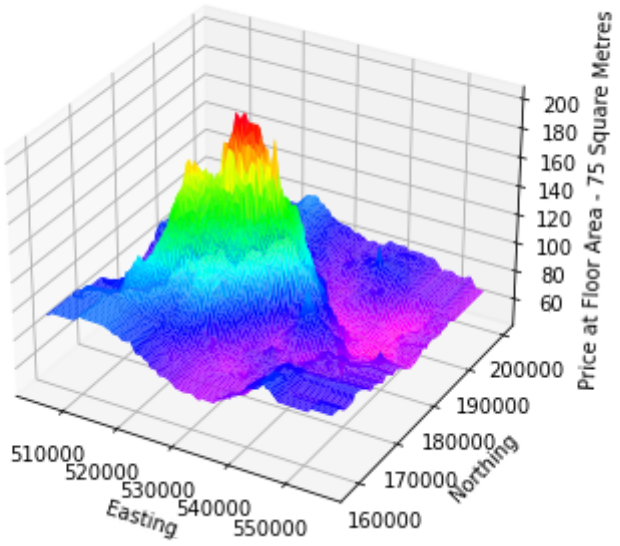
In [206]:

```
def surf3d(pipe_model,fl_area):
    east_mesh, north_mesh = np.meshgrid(
        np.linspace(505000,555800,100),
        np.linspace(158400,199900,100))
    fl_mesh = np.zeros_like(east_mesh)
    fl_mesh[:,:] = fl_area
    grid_predictor_vars = np.array([east_mesh.ravel(),
        north_mesh.ravel(),fl_mesh.ravel()]).T
    hp_pred = pipe_model.predict(grid_predictor_vars)
    hp_mesh = hp_pred.reshape(east_mesh.shape)
    fig = pl.figure()
    ax = Axes3D(fig)
    ax.plot_surface(east_mesh, north_mesh, hp_mesh,
        rstride=1, cstride=1, cmap='gist_rainbow_r',lw=0.01)
    ax.set_title('3-D plot of the predicted house price for floor areas of 75 Square Me
tres')
    ax.set_xlabel('Easting')
    ax.set_ylabel('Northing')
    ax.set_zlabel('Price at Floor Area - 75 Square Metres')
    pl.show()
    return
```

In [207]:

```
pl.close()
surf3d(opt_nn2,75.0)
pl.show()
```

3-D plot of the predicted house price for floor areas of 75 Square Metres



3-D Graph for Floor Area at 125 Square Metres

In this graph the peak is more defined, considering the larger size of houses, the prices are high in this area. Again the peak is in the centre of the grid which would mean that the more expensive houses are in the city centre.

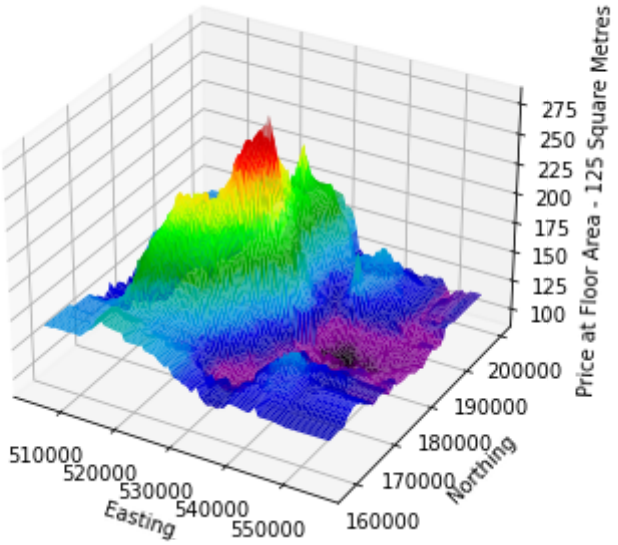
In [220]:

```
def surf3d(pipe_model,fl_area):
    east_mesh, north_mesh = np.meshgrid(
        np.linspace(505000,555800,100),
        np.linspace(158400,199900,100))
    fl_mesh = np.zeros_like(east_mesh)
    fl_mesh[:,:] = fl_area
    grid_predictor_vars = np.array([east_mesh.ravel(),
        north_mesh.ravel(),fl_mesh.ravel()]).T
    hp_pred = pipe_model.predict(grid_predictor_vars)
    hp_mesh = hp_pred.reshape(east_mesh.shape)
    fig = pl.figure()
    ax = Axes3D(fig)
    ax.plot_surface(east_mesh, north_mesh, hp_mesh,
        rstride=1, cstride=1, cmap='nipy_spectral',lw=0.01)
    ax.set_title('3-D plot of the predicted house price for floor areas of 125 Square M
etres')
    ax.set_xlabel('Easting')
    ax.set_ylabel('Northing')
    ax.set_zlabel('Price at Floor Area - 125 Square Metres')
    pl.show()
    return
```

In [221]:

```
pl.close()
surf3d(opt_nn2,125.0)
pl.show()
```

3-D plot of the predicted house price for floor areas of 125 Square Metres



Conclusion

This assignment has used the k nearest neighbours approach to look at how three predictor variables (east, north and floor area) have made predictions for the price of houses in the London area. Through 3-D graphing using different stipulations of floor area we can see that the most expensive houses are centred towards the London city area, while the cheaper houses are on the outskirts.