Introduction

Assignment

- **Course:** Higher Diploma in Data Analytics
- **Module:** NCG603

- **Assignment:** Machine Learning
- **Filename:** NCG603 - Python Machine Learning Assignment.ipynb

- **Student Name:** Sean O'Riogain
- **Student Number:** 18145624

Objective

The objective of this assignment is to use the provided code snippets to develop a model which predicts the (purchase/sales) price of a house in London based on its location (easting-northing coordinates) and floor area using an optimised K Nearest Neighbours (KNN) algorithm. That optimisation (using training & test cross-validation) will be done based on a sample of data from 1405 actual house purchases in that city. The resultant predictive capability will then be used to produce a 3D plot that 'maps' the predicted house prices across London for houses for each of the following floor area measurements:

1. Average (of all floor area measurements in the dataset);
2. $75m^2$;
3. $125m^2$.

Approach

The following is a list of the steps in the process that will be followed to achieve those objectives:

1. Import the required packages and functions;
2. Load the house price dataset;
3. Perform KNN with scaling & optimisation via cross validation (using a pipeline workflow);
4. Display the attributes of the selected (optimum) model;
5. Produce the required 3D plots.

Process Execution

The following subsections implement the approach outlined in the previous section of this document.

Import Packages and Functions

In [9]:
```python
# General Purpose:
import numpy as np
import pandas as pd

# For creating the data-scaling object:
from sklearn.preprocessing import StandardScaler
L
# For fitting the KNN model:
from sklearn.neighbors import KNeighborsRegressor as NN

# For assessing the optimum fit:
from sklearn.metrics import mean_absolute_error, make_scorer
from sklearn.model_selection import GridSearchCV

# For automating execution of the scaling, fitting and optimising steps:
from sklearn.pipeline import Pipeline

# For producing the required 3D plots:
import pylab as pl
from mpl_toolkits.mplot3d import Axes3D
```

Load Data

We will now load the data and display its structure as well as the contents of its first few rows.

In [41]:
```python
hp = pd.read_csv('hpdemo.csv',dtype=float)

print(hp.info())

print(hp[:6])
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1405 entries, 0 to 1404
Data columns (total 5 columns):
ID         1405 non-null float64
east       1405 non-null float64
north      1405 non-null float64
price      1405 non-null float64
fl_area    1405 non-null float64
dtypes: float64(5)
memory usage: 55.0 KB
None
    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
5  6.0  547900.0  189600.0   69995.0     95.0
```

The results above confirm that the required dataset has been imported and contains 1405 rows and 5 columns.

Perform KNN with Scaling & Optimisation

KNN is reliant on being able to accurately measure the distance between (the next nearest) predictors based on their coordinates in a p-dimensional space, where p is the number of predictors. In this case, p=3 because we have 3 predictors (east, north and fl_area). However, if the coordinate axes are not of the same scale, those distances become distorted. Although east and north are on the same scale (measured in meters), fl_area is on a different scale (measured in square meters). Therefore, we need to scale them, for which we will use the standard scaler (using on mean and standard deviation) to apply the same scale to all 3 predictors (whose values are then represented by their z-scores).

We will now extract the response variable (in thousands of pounds) and define the pipeline - i.e. the workflow (the steps that will need to be performed) - to fit one or more KNN models. This involves the scaling of the data followed by executing the KNN algorithm upon it. We will call the former step 'zscores' and the latter one 'NNreg'. Each step in the pipeline will implicitly invoke the workflow object's fit method upon instantiation and its transform method on exit - except for the final workflow object where its predict method will be invoked instead upon exit.

We will call the GridSearchCV function to drive execution of the specified workflow and to pass the selected value ranges for the specified workflow object's tuning parameter(s). In this case, we will pass it value ranges for the n_neigbors, weights and p tuning parameters for the NNReg (KNN execution) object and the scoring method (MAE) that we want it to use during cross validation. In effect, this means that GridSearchCV, as part of its workflow execution, repeatedly invokes the KNN execution object itself in order to identify its optimal tuning settings using cross validation.

We then apply the resultant optimum fit (called opt_nn2) to the response variable using its fit method.

In [32]:

```
price = hp["price"]/1000                                                      # Extract t
he response variable (price) in £000s

print(price[:6])                                                              # Display i
ts first few values

pipe = Pipeline([('zscores',StandardScaler()),('NNreg',NN())])                # Specify t
he pipe (workflow steps)

mae = make_scorer(mean_absolute_error, greater_is_better=False)               # Specify t
he scoring method (Mean Absolute Error)

opt_nn2 = GridSearchCV(                                                       # Use GridS
earchCV to perform
    estimator = pipe,                                                         #  - the sp
ecified workflow
    scoring = mae,                                                            #  - using
 the specified scoring method and
    param_grid = {                                                            #  - specif
ied combinations of (KNN) tuning parameters
        'NNreg__n_neighbors':range(1,35),                                     #     - k
 (number of neighbours) in the range 1 - 35
        'NNreg__weights':['uniform','distance'],                              #     - uni
form and distance weighting
        'NNreg__p':[1,2]})                                                    #     - dis
tance type (1=Euclidian and 2=City Block)

opt_nn2.fit(hp[['east','north','fl_area']],price)                             # Apply the
optimimum fit to response variable (price)
```

```
0    107.000
1     55.500
2    103.000
3    187.000
4     43.000
5     69.995
Name: price, dtype: float64

C:\Users\oriogain\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:205
3: FutureWarning: You should specify a value for 'cv' instead of relying on the defa
ult value. The default value will change from 3 to 5 in version 0.22.
  warnings.warn(CV_WARNING, FutureWarning)
```

Out[32]:

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=Pipeline(memory=None,
     steps=[('zscores', StandardScaler(copy=True, with_mean=True, with_std=True)),
('NNreg', KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=None, n_neighbors=5, p=2,
          weights='uniform'))]),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'NNreg__n_neighbors': range(1, 35), 'NNreg__weights': ['uniform',
'distance'], 'NNreg__p': [1, 2]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=make_scorer(mean_absolute_error, greater_is_better=False),
       verbose=0)
```

Display the Attributes of the Optimum Model

We now need to verify the success of the previous step's KNN workflow (pipeline) by displaying the details of the optimum model (i.e. its tuning parameter and score values) that it eventually selected.

In [33]:
```python
def print_summary2(opt_pipe_object):
    params = opt_pipe_object.best_estimator_.get_params()
    score = - opt_pipe_object.best_score_
    print("Nearest neighbours: %8d" % params['NNreg__n_neighbors'])
    print("Minkowski p       : %8d" % params['NNreg__p'])
    print("Weighting         : %8s" % params['NNreg__weights'])
    print("MAE Score         : %8.2f" % score)
    return

print_summary2(opt_nn2)
```

```
Nearest neighbours:       13
Minkowski p       :        1
Weighting         : distance
MAE Score         :    26.47
```

The above results show that the iterative cross-validation automation driven by the GridSearchCV function selected a KNN model with a k value (number of nearest neighbors) value of 13, a p value of 1 (for City Block distance) and a weighting type of distance (rather than uniform), and that those tuning parameter values yielded the (lowest) MAE scoring value of 26.47.

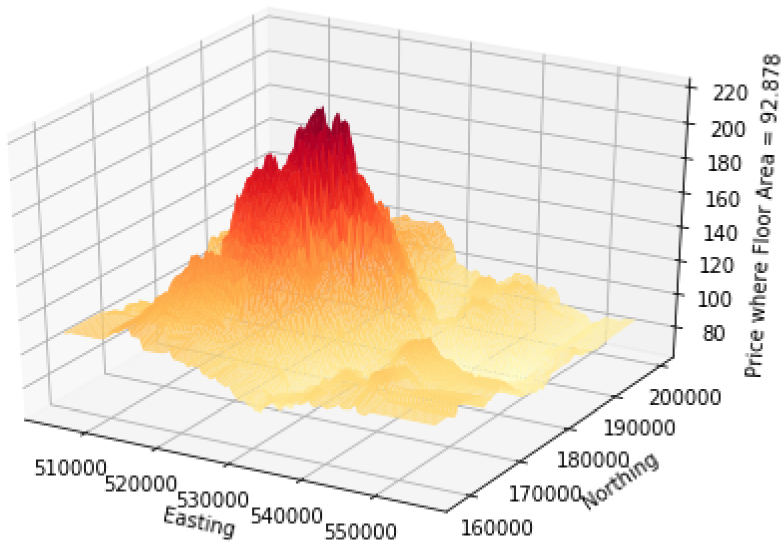
Produce the Required 3D Plots

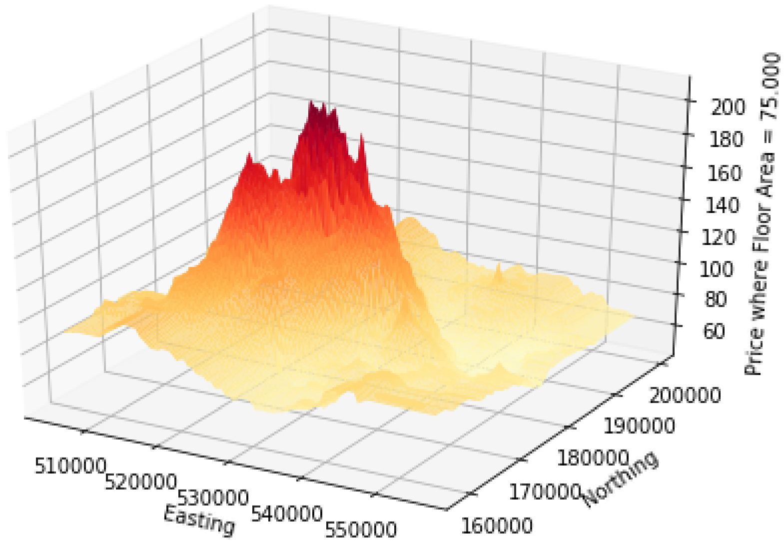
We now want to visualise what prices the optimum model predicts across London for houses of average floor area, and for floor areas of 75 and 125 square meters, respectively. To do this, we create a 100x100 grid (mesh) of points which span the area covered by the dataset's easting and northing ranges. Writing a function to produce the required type of 3D plot enables us to invoke it successively for the floor area values in which we are interested.

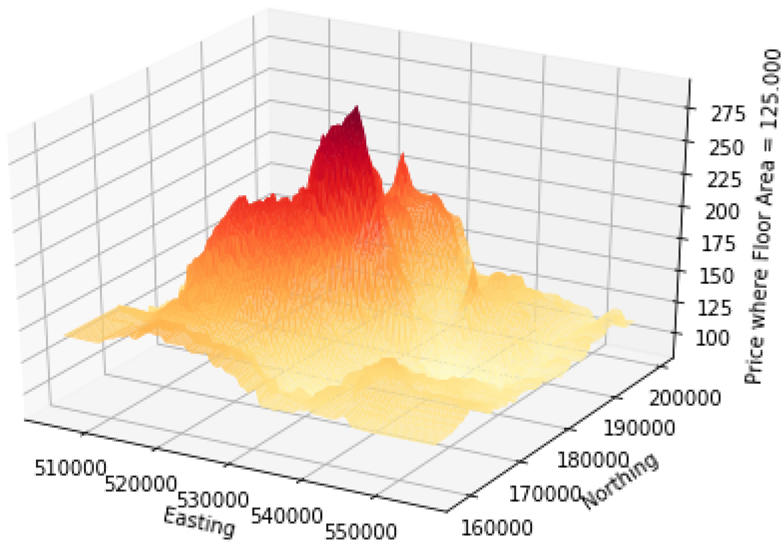
In [51]:

```
def surf3d(pipe_model,fl_area):                                # Accept the model and floo
r area as input parameters
    zlabel = 'Price where Floor Area = %6.3f' % fl_area        # Set the label text for th
e z-axis based on floor area
    east_mesh, north_mesh = np.meshgrid(                       # Create the 100 x 100 grid
(mesh) on the x-y plane using
        np.linspace(505000,555800,100),                        #   - easting value range a
nd
        np.linspace(158400,199900,100))                        #   - northingvalue range
    fl_mesh = np.zeros_like(east_mesh)                         # Initialise the z-axis gri
d intervals to zeros (based on easting axis)
    fl_mesh[:,:] = fl_area                                     # Set its values to that of
the relevant input parameter
    grid_predictor_vars = np.array([east_mesh.ravel(),         # Merge (ravel) the x-, y-
 and z-axis into a single vector to suit the
        north_mesh.ravel(),fl_mesh.ravel()]).T                 #    predict method of the
 specified model
    hp_pred = pipe_model.predict(grid_predictor_vars)          # Get the predicted house p
rices (as a vector)
    hp_mesh = hp_pred.reshape(east_mesh.shape)                 # Split (unravel) the predi
ction vector into its x- y- and z-axis grid
                                                               #    components
    pl.close()                                                 # Clear any previous plot
 (to ensure no overplotting)
    fig = pl.figure()                                          # Instantiate a plot object
    ax = Axes3D(fig)                                           # Draw the plot's 3 axes
    ax.plot_surface(east_mesh, north_mesh, hp_mesh,            # Draw the 3-D surface usin
g the 3 grid (mesh) vectors
        rstride=1, cstride=1, cmap='YlOrRd',lw=0.01)           #    and a Yellow-Orange-Re
d colour ramp
    ax.set_xlabel('Easting')                                   # Label the c-, y- and z-ax
es
    ax.set_ylabel('Northing')
    ax.set_zlabel(zlabel)
    pl.show()                                                  # Display the plot
    return

surf3d(opt_nn2,np.mean(hp["fl_area"]))                         # Produce the 3D surface pl
ot for the average floor area
surf3d(opt_nn2,75)                                             #    and for a floor area o
f 75 square meters
surf3d(opt_nn2,125)                                            #    and for a floor area o
f 125 square meters
```







The 3 plots above show that the highest house prices are predicted in approximately the same areas of London, where those prices peak at c. £160k, £170k and £230k for floor areas 75, 93 (average) and 125 square meters, respectively. (We can also see that coverage of the plot expands into the southeast end of London as the floor area increases.)