



**ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ**  
**UNIVERSITY OF PIRAEUS**

**ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ**

**ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ**

**Αναγνώριση προτύπων**

**Υπολογιστική εργασία 2021-2022**

**Εξάμηνο 5<sup>ο</sup>**

**Ιωάννης-Ιάσων Μποϊδάνης π19217**

**Ευθύμιος-Πάτροκλος Γεωργιάδης π19031**

**Καθηγητής Δ. Σωτηρόπουλος**

**Επισυνάπτονται: ένα αρχείο matlab για ευκολία στην αναπαράσταση δεδομένων και ένα αρχείο  
pythοn για την υπολογιστική διαδικασία(το οποίο διαθέτει τα απαραίτητα γραφήματα, απλά  
είναι πιο δύσχρηστο)**

**Περιεχόμενα**

<b>Αρχεία κώδικα.....σελ 3</b>
<b>Ερώτημα 1.....σελ 7</b>
<b>Ερώτημα 2.....σελ 7</b>
<b>Ερώτημα 3.....σελ 8</b>
<b>Ερώτημα 4.....σελ 8</b>
<b>Ερώτημα 5.....σελ 8</b>
<b>Ερώτημα 6.....σελ 11</b>
<b>Ερώτημα 7.....σελ 14</b>
<b>Ερώτημα 9.....σελ 17</b>



## Κώδικας

```
import tensorflow as tf

from keras.models import Sequential
from keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from keras.callbacks import EarlyStopping

import pandas as pd
import numpy as np
import padasip as pad
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
import sklearn
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

from matplotlib import pyplot as plt

def normalizedata(dt):
    return (dt - np.min(dt)) / (np.max(dt) - np.min(dt))

plt.rcParams["figure.figsize"] = [7.00, 3.50]
plt.rcParams["figure.autolayout"] = True
columns = ["longitude", "latitude", "housing_median_age", "total_rooms",
           "total_bedrooms", "population", "households", "median_income",
           "median_house_value", "ocean_proximity"]
data = pd.read_csv("housing.csv", usecols=columns)
housing = pd.read_csv("housing.csv")
data.fillna(data.total_bedrooms.mean(), inplace=True)
scaled_long = normalizedata(data.longitude)
plt.show()
scaled_lat = normalizedata(data.latitude)
scaled_age = normalizedata(data.housing_median_age)
scaled_room = normalizedata(data.total_rooms)
scaled_bed = normalizedata(data.total_bedrooms)
scaled_pop = normalizedata(data.population)
scaled_houses = normalizedata(data.households)
scaled_income = normalizedata(data.median_income)
scaled_value = normalizedata(data.median_house_value)

from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()
housing[['longitude', 'latitude', 'housing_median_age', 'total_rooms',
         'total_bedrooms', 'population', 'households', 'median_income',
         'median_house_value']] = min_max_scaler.fit_transform(housing[['longitude', 'latitude', 'housing_median_age',
         'total_bedrooms', 'population', 'households', 'median_income',
         'median_house_value']])

one_hot = pd.get_dummies(housing['ocean_proximity'])
housing = housing.drop('ocean_proximity', axis=1)
housing = housing.join(one_hot)
'''label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(data.ocean_proximity)
# print(integer_encoded)
onehot_encoder = OneHotEncoder(sparse=False)
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
onehot_encoded_ocean = onehot_encoder.fit_transform(integer_encoded)
# print(onehot_encoded_ocean)

scaled_ocean = normalizedata(onehot_encoded_ocean)'''
#housing = np.c_[scaled_long, scaled_lat, scaled_age, scaled_room, scaled_bed, scaled_pop, scaled_houses, scaled_income, scaled_ocean]
# print(scaled_ocean)
#print(housing)
'''plt.scatter(scaled_long, scaled_lat)
plt.show()

plt.scatter(scaled_value, scaled_income)
plt.show()'''

plt.hist(data.longitude, bins=16, density=True, stacked=True)
# plt.legend()
plt.title('Histogram 1')
plt.xlabel('Longitude')
plt.ylabel('Prob density')
plt.show()
```

```

plt.hist(data.latitude, bins=16, density=True, stacked=True)
plt.title('Histogram 2')
plt.xlabel('latitude')
plt.ylabel('Prob dencity')
plt.show()

plt.hist(data.housing_median_age, bins=16, density=True, stacked=True)
plt.title('Histogram 3')
plt.xlabel('age')
plt.ylabel('Prob dencity')
plt.show()

plt.hist(data.total_rooms, bins=64, density=True, stacked=True)
plt.title('Histogram 4')
plt.xlabel('rooms')
plt.ylabel('Prob dencity')
plt.show()

plt.hist(data.total_bedrooms, bins=64, density=True, stacked=True)
plt.title('Histogram 5')
plt.xlabel('beds')
plt.ylabel('Prob dencity')
plt.show()

plt.hist(data.population, bins=64, density=True, stacked=True)
plt.title('Histogram 6')
plt.xlabel('population')
plt.ylabel('Prob dencity')
plt.show()

plt.hist(data.households, bins=64, density=True, stacked=True)
plt.title('Histogram 7')
plt.xlabel('houses')
plt.ylabel('Prob dencity')
plt.show()

plt.hist(data.median_income, bins=16, density=True, stacked=True)
plt.title('Histogram 8')
plt.xlabel('income')
plt.ylabel('Prob dencity')
plt.show()

plt.hist(data.ocean_proximity, bins=16, density=True, stacked=True)
plt.title('Histogram 9')
plt.xlabel('ocean')
plt.ylabel('Prob dencity')
plt.show()

...
print(data.ocean_proximity)
print(data.median_house_value)
print(data.median_income)
print(data.households)
print(data.population)
print(data.total_bedrooms)
print(data.total_rooms)
print(data.housing_median_age)
print(data.latitude)
print(data.longitude)
...

```

```

plt.title('Histogram 9')
plt.xlabel('ocean')
plt.ylabel('Prob dencity')
plt.show()

plt.hist(data.median_house_value, bins=16, density=True, stacked=True)
plt.title('Histogram 10')
plt.xlabel('value')
plt.ylabel('Prob dencity')
plt.show()

...

```

```

#Y_true = np.c_[
# scaled_long, scaled_lat, scaled_bed, scaled_age, scaled_room, scaled_income, scaled_pop, scaled_houses, scaled_o
# print(Y_true)
#Y_pred = np.c_[scaled_value]
# print(Y_pred)
#MSE = np.square(np.subtract(Y_true.transpose(), Y_pred.transpose())).mean()
#print(MSE)
#Y_Pred=Y_pred.transpose()
df2 = housing['median_house_value']

df1 = housing.drop('median_house_value',axis=1)
X_train, X_test, y_train, y_test = train_test_split(df1, df2, test_size=0.2)
# Plots the results of a learning rate of 100, 1000, and 10000 respectively, with all other parameters constant

```

```
In [3]: #X_train = preprocessing.scale(X_train)
print(df1)
#X_test = preprocessing.scale(X_test)
```

	longitude	latitude	housing	median age	total rooms	total bedrooms	\
0	0.211155	0.567481		0.784314	0.022331	0.019863	
1	0.212151	0.565356		0.392157	0.180503	0.171477	
2	0.210159	0.564293		1.000000	0.037260	0.029330	
3	0.209163	0.564293		1.000000	0.032352	0.036313	
4	0.209163	0.564293		1.000000	0.041330	0.043296	
...	...	...		...	...	...	
20635	0.324701	0.737513		0.470588	0.042296	0.057883	
20636	0.312749	0.738576		0.333333	0.017676	0.023122	
20637	0.311753	0.732200		0.313725	0.057277	0.075109	
20638	0.301793	0.732200		0.333333	0.047256	0.063315	
20639	0.309761	0.725824		0.294118	0.070782	0.095438	

	population	households	median income	<1H OCEAN	INLAND	ISLAND	\
0	0.008941	0.020556	0.539668	0	0	0	
1	0.067210	0.186976	0.538027	0	0	0	
2	0.013818	0.028943	0.466028	0	0	0	
3	0.015555	0.035849	0.354699	0	0	0	
4	0.015752	0.042427	0.230776	0	0	0	
...	...	...	...	...	...	...	
20635	0.023599	0.054103	0.073130	0	1	0	
20636	0.009094	0.018502	0.141853	0	1	0	
20637	0.028140	0.071041	0.082764	0	1	0	
20638	0.020684	0.057227	0.094295	0	1	0	
20639	0.038790	0.086992	0.130253	0	1	0	

	NEAR BAY	NEAR OCEAN
0	1	0
1	1	0
2	1	0
3	1	0
4	1	0
...	...	...
20635	0	0
20636	0	0
20637	0	0
20638	0	0
20639	0	0

[20640 rows x 13 columns]

```
In [4]: print(df2)
```

0	0.902266
1	0.708247
2	0.695051
3	0.672783
4	0.674638
...	...
20635	0.130185
20636	0.128043
20637	0.159383
20638	0.143713
20639	0.153403

Name: median\_house\_value, Length: 20640, dtype: float64

```
In [7]: LR = [100,1000,10000]

for i in LR:
    #Defines linear regression model and its structure
    model = Sequential()
    model.add(Dense(1, input_shape=(13,)))

    #Compiles model
    model.compile(Adam(learning_rate=i), 'mean_squared_error')

    #Fits model
    history = model.fit(X_train, y_train, epochs = 500, validation_split = 0.1, verbose = 0)
    history_dict=history.history

    #Plots model's training cost/loss and model's validation split cost/loss
    loss_values = history_dict['loss']
    val_loss_values=history_dict['val_loss']
    plt.figure()
    plt.plot(loss_values,'bo',label='training loss')
    plt.plot(val_loss_values,'r',label='val training loss')
```

```
In [8]: # Runs and plots the performance of a model with the same parameters from before (and a learning rate of 10000
# but now with an activation function (Relu)

model = Sequential()
model.add(Dense(1, input_shape=(13,), activation = 'relu'))
model.compile(Adam(learning_rate=10000), 'mean_squared_error')
history = model.fit(X_train, y_train, epochs = 500, validation_split = 0.1, verbose = 0)

history_dict=history.history
loss_values = history_dict['loss']
val_loss_values=history_dict['val_loss']
plt.plot(loss_values,'bo',label='training loss')
plt.plot(val_loss_values,'r',label='training loss val')
```

```

: #Runs model (the one with the activation function, although this doesn't really matter as they perform the same)
# with its current weights on the training and testing data
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

```

```

# Calculates and prints r2 score of training and testing data
print("The R2 score on the Train set is:\t{:0.3f}".format(r2_score(y_train, y_train_pred)))
print("The R2 score on the Test set is:\t{:0.3f}".format(r2_score(y_test, y_test_pred)))

```

```

The R2 score on the Train set is:      -2.764
The R2 score on the Test set is:      -2.766

```

```

: # Defines "deep" model and its structure
model = Sequential()
model.add(Dense(13, input_shape=(13,), activation='relu'))
model.add(Dense(13, activation='relu'))
model.add(Dense(13, activation='relu'))
model.add(Dense(13, activation='relu'))
model.add(Dense(13, activation='relu'))
model.add(Dense(1,))
model.compile(Adam(learning_rate=0.003), 'mean_squared_error')

# Runs model for 2000 iterations and assigns this to 'history'
history = model.fit(X_train, y_train, epochs = 6000, validation_split = 0.2, verbose = 0)

# Plots 'history'
history_dict=history.history
loss_values = history_dict['loss']
val_loss_values=history_dict['val_loss']
plt.plot(loss_values,'bo',label='training loss')
plt.plot(val_loss_values,'r',label='training loss val')

```

## Αρχικά

1) αποδίδουμε τιμές στα χαρακτηριστικά

(με δύο τρόπους για να τα χρησιμοποιήσουμε σε διαφορετικά σημεία)

```
plt.rcParams["figure.figsize"] = [7.00, 3.50]
plt.rcParams["figure.autolayout"] = True
columns = ["longitude", "latitude", "housing_median_age", "total_rooms",
           "total_bedrooms", "population", "households", "median_income",
           "median_house_value", "ocean_proximity"]
data = pd.read_csv("housing.csv", usecols=columns)
housing = pd.read_csv("housing.csv")
```

```
M= readmatrix('housing.csv');
long = M(:,1);
lat = M(:,2);
age =M(:,3);
room = M(:,4);
bed = M(:,5);
pop = M(:,6);
houses = M(:,7);
income = M(:,8);
value = M(:,9);
ocean = M(:,10);
```

## 2) κάνουμε scaling των δεδομένων

```
scaled_long = normalizedata(data.longitude)
plt.show()
scaled_lat = normalizedata(data.latitude)
scaled_age = normalizedata(data.housing_median_age)
scaled_room = normalizedata(data.total_rooms)
scaled_bed = normalizedata(data.total_bedrooms)
scaled_pop = normalizedata(data.population)
scaled_houses = normalizedata(data.households)
scaled_income = normalizedata(data.median_income)
scaled_value = normalizedata(data.median_house_value)

from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()
housing[['longitude','latitude','housing_median_age', 'total_rooms',
         'total_bedrooms', 'population', 'households', 'median_income',
         'median_house_value']] = min_max_scaler.fit_transform(housing[['longitude','latitude','housing_median_age',
         'total_bedrooms', 'population', 'households', 'median_income',
         'median_house_value']])
```

```
long = rescale(long); %rescaling to [0,1]
```

```
pop = rescale(pop);
```

```
%mean_lat = mean(lat);
%lat = fillmissing(lat,"constant",mean_lat);
lat = rescale(lat);
```

```
%mean_houses = mean(houses);
%houses = fillmissing(houses,"constant",mean_ho
houses = rescale(houses);
```

```
%mean_age= mean(age);
%age = fillmissing(age,"constant",mean_age);
age = rescale(age);
```

```
%mean_in = mean(income);
%income = fillmissing(income,"constant",mean_in
income = rescale(income);
```

```
%mean_room = mean(room);
%room = fillmissing(room,"constant",mean_room);
room= rescale(room);
```

```
value = rescale(value);
```

## 3) one hot vector



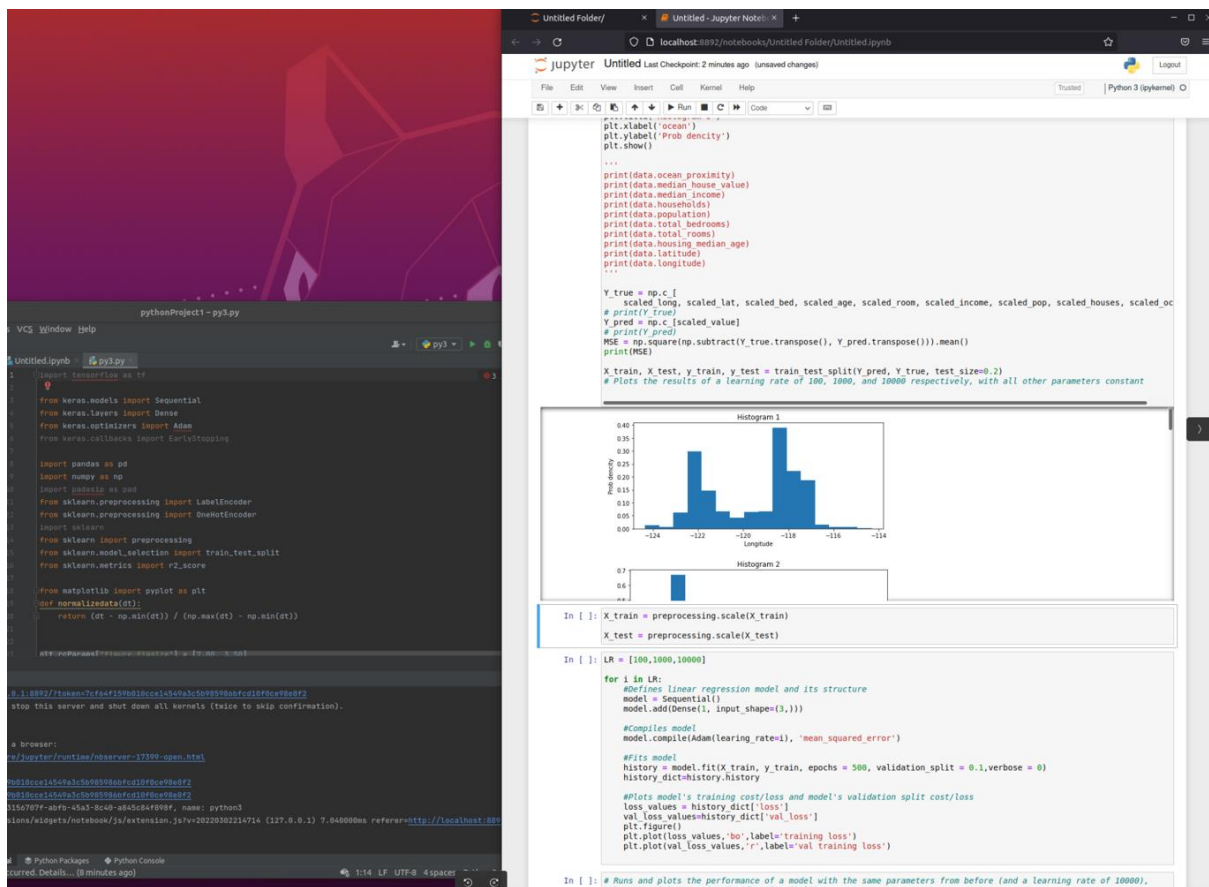
```
one_hot= pd.get_dummies(housing['ocean_proximity'])
housing = housing.drop('ocean_proximity',axis=1)
housing=housing.join(one_hot)
```

#### 4)για τις ελλειπείς τιμές των υπονοματιών

```
data.fillna(data.total_bedrooms.mean(),inplace=True)
```

```
mean_bed = nanmean.bed);
bed = fillmissing.bed, "constant", mean_bed);
```

#### 5)αναπαράσταση ιστογραμμάτων των συναρτήσεων πυκνότητας πιθανότητας για τα 10 χαρακτηριστικά



Untitled Folder/ x Untitled - Jupyter Note: x +

localhost:8892/notebooks/Untitled Folder/Untitled.ipynb

jupyter Untitled Last Checkpoint: 2 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Help Trusted Python 3 (ipykernel)

```

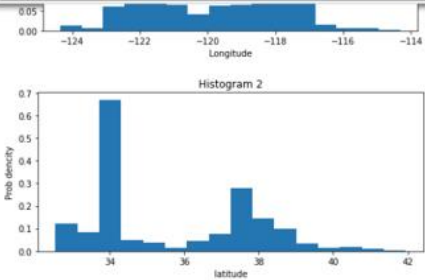
plt.xlabel('ocean')
plt.ylabel('Prob density')
plt.show()

...
print(data.ocean_proximity)
print(data.median_house_value)
print(data.median_income)
print(data.households)
print(data.population)
print(data.total_bedrooms)
print(data.total_rooms)
print(data.housing_median_age)
print(data.latitude)
print(data.longitude)
...

Y_true = np.c_[
    scaled_long, scaled_lat, scaled_bed, scaled_age, scaled_room, scaled_income, scaled_pop, scaled_houses, scaled_oc
# print(Y_true)
Y_pred = np.c_[scaled_value]
# print(Y_pred)
MSE = np.square(np.subtract(Y_true.transpose(), Y_pred.transpose())).mean()
print(MSE)

X_train, X_test, y_train, y_test = train_test_split(Y_pred, Y_true, test_size=0.2)
# Plots the results of a learning rate of 100, 1000, and 10000 respectively, with all other parameters constant

```



```

In [ ]: X_train = preprocessing.scale(X_train)
        X_test = preprocessing.scale(X_test)

In [ ]: LR = [100,1000,10000]

for i in LR:
    #Defines linear regression model and its structure
    model = Sequential()
    model.add(Dense(1, input_shape=(3,)))

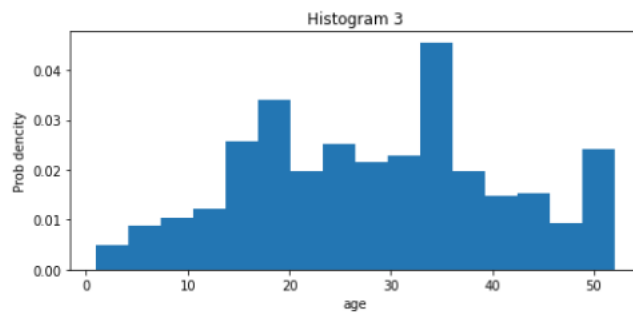
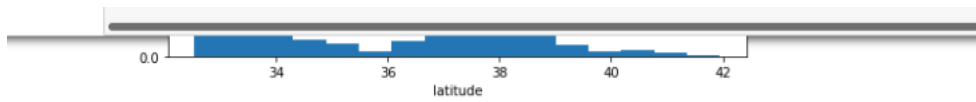
    #Compiles model
    model.compile(Adam(learning_rate=i), 'mean_squared_error')

    #Fits model
    history = model.fit(X_train, y_train, epochs = 500, validation_split = 0.1, verbose = 0)
    history_dict=history.history

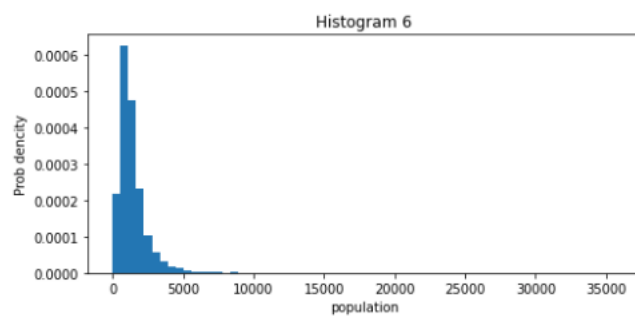
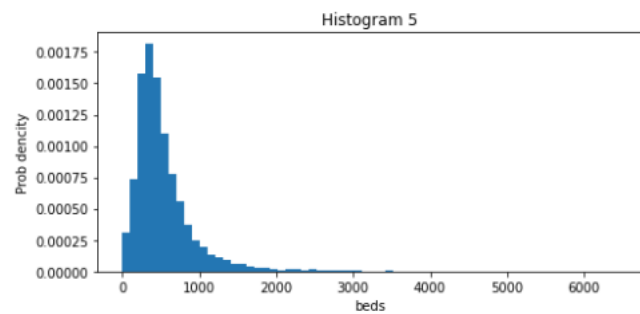
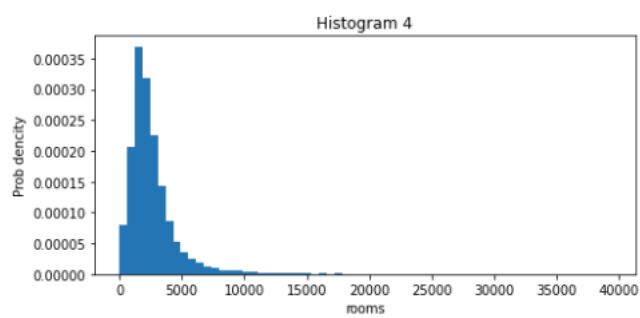
    #Plots model's training cost/loss and model's validation split cost/loss
    loss_values = history_dict['loss']
    val_loss_values=history_dict['val_loss']
    plt.figure()
    plt.plot(loss_values,'bo',label='training loss')
    plt.plot(val_loss_values,'r',label='val training loss')

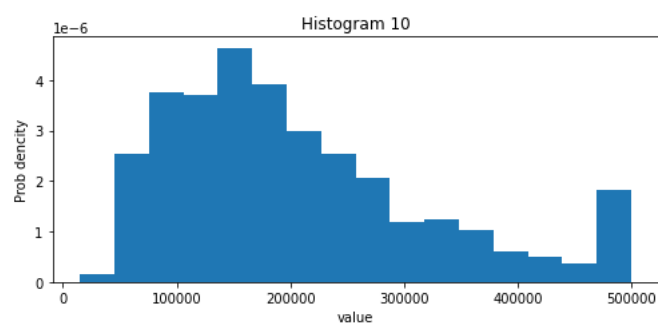
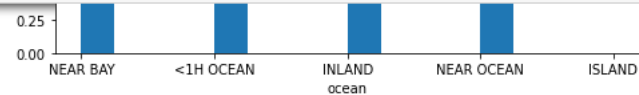
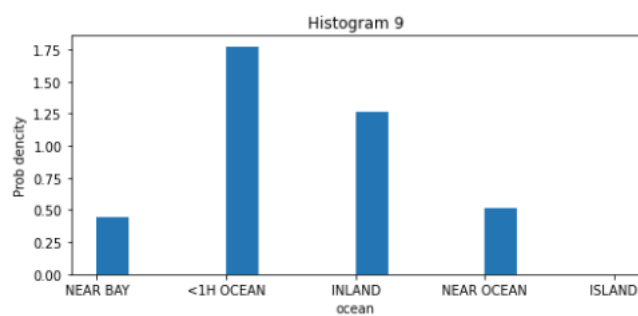
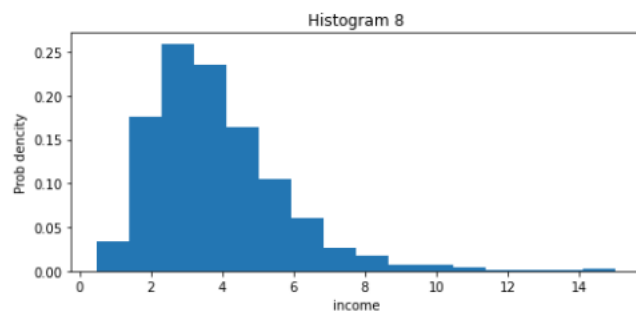
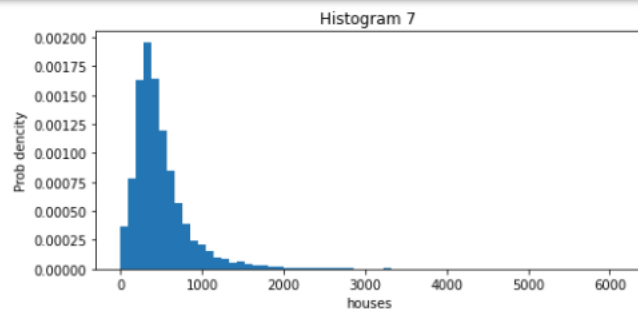
In [ ]: # Runs and plots the performance of a model with the same parameters from before (and a learning rate of 10000),

```

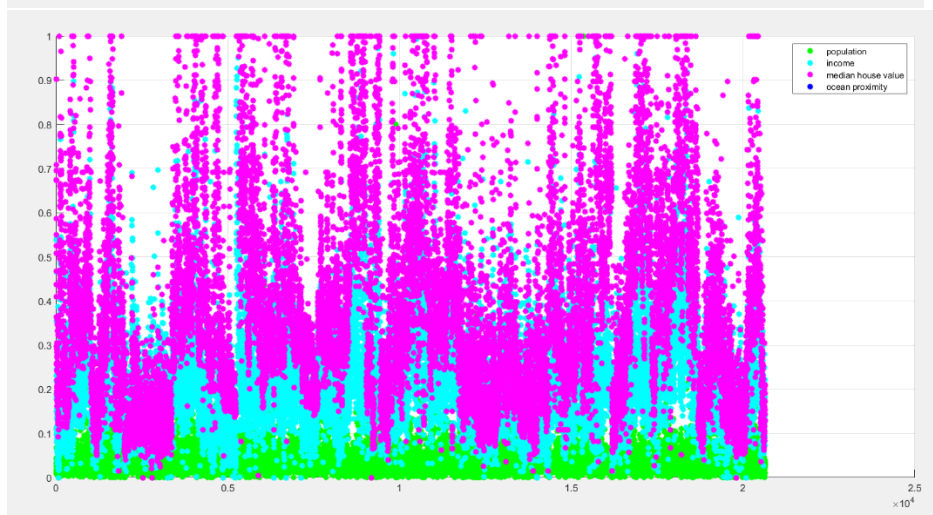
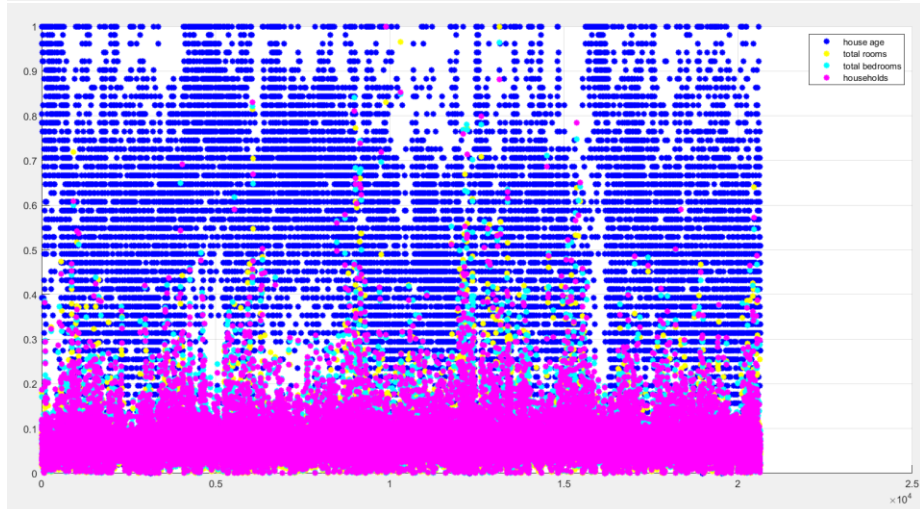
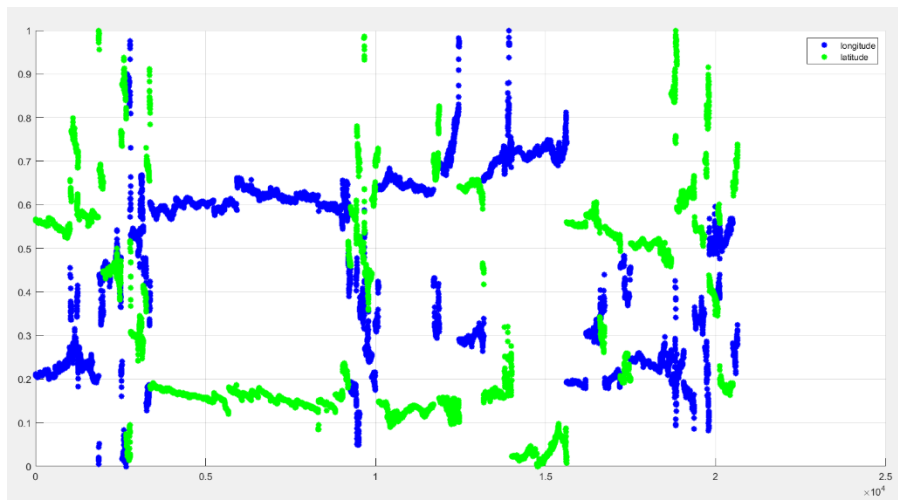


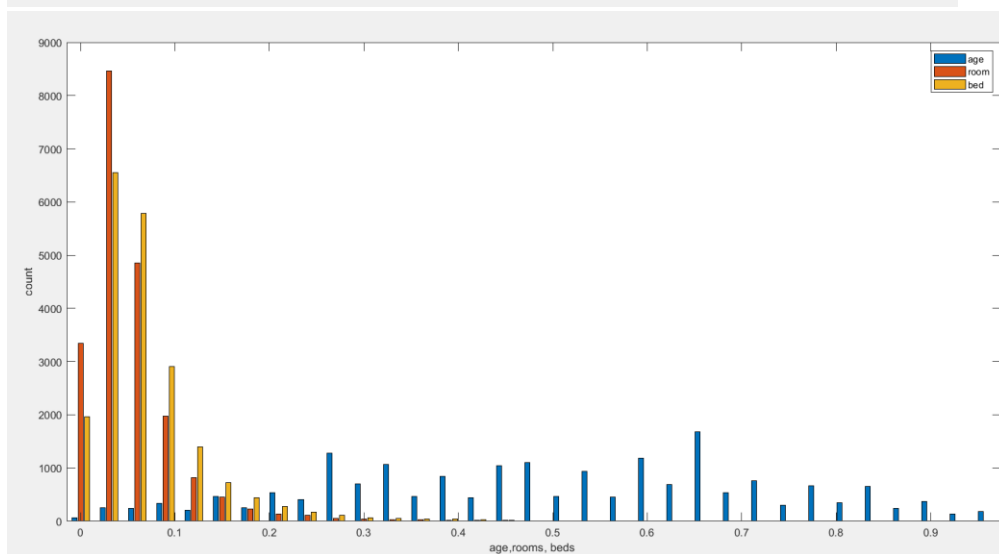
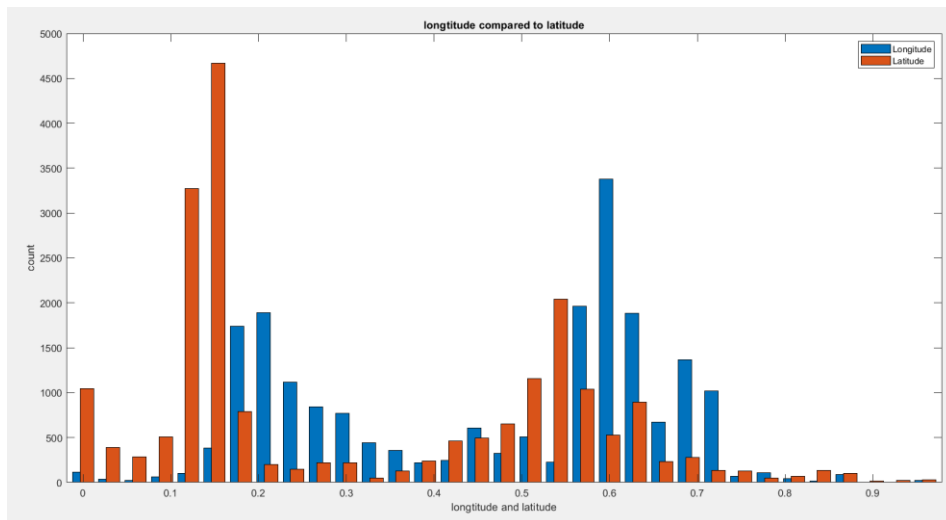
Histogram 4

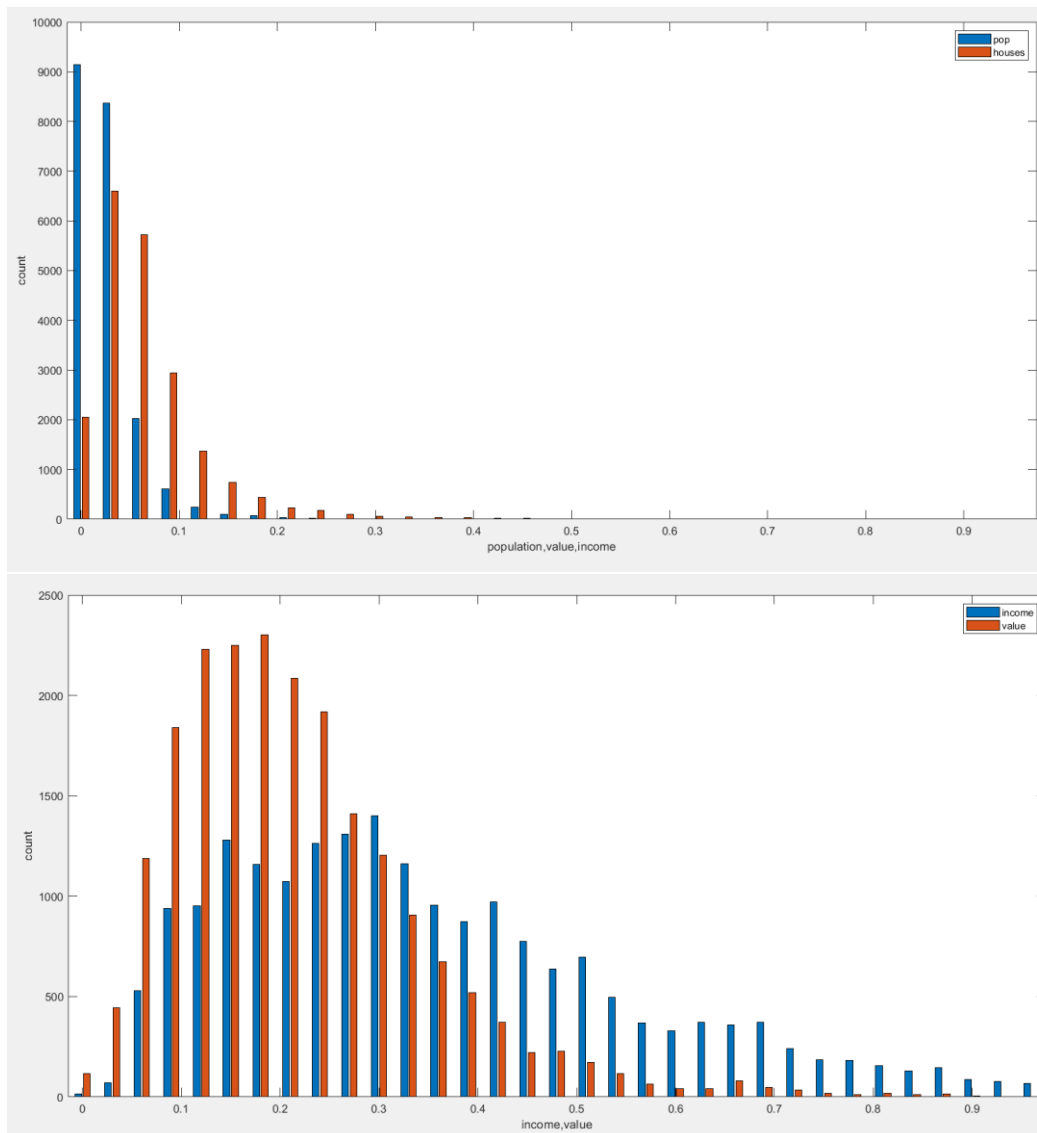




6) συνδυασμοί μεταβλητών σε μορφή σημείων(scaled εκδοχής) και ιστογραμμάτων πυκνότητας εμφάνισης των πραγματικών τιμών







## 7)αλγόριθμος mse

Ετοιμάζουμε τα δεδομένα μας, και ορίζουμε πλήθος προτύπων για εκπαίδευση

```
#Y_true = np.c [
# scaled_long, scaled_lat, scaled_bed, scaled_age, scaled_room, scaled_income, scaled_pop, scaled_houses, scaled_o
# print(Y_true)
#Y_pred = np.c [scaled_value]
# print(Y_pred)
#MSE = np.square(np.subtract(Y_true.transpose(), Y_pred.transpose()))
# print(MSE)
#Y_Pred=Y_pred.transpose()
df2 = housing['median_house_value']

df1 = housing.drop('median_house_value',axis=1)
X_train, X_test, y_train, y_test = train_test_split(df1, df2, test_size=0.2)
# Plots the results of a learning rate of 100, 1000, and 10000 respectively, with all other parameters constant
```

Τα δεδομένα μας είναι σε μορφή έτοιμη για παλινδρόμηση και στις σωστές διαστάσεις

In [3]:

```
#X_train = preprocessing.scale(X_train)
print(df1)
#X_test = preprocessing.scale(X_test)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	\
0	0.211155	0.567481	0.784314	0.022331	0.019863	
1	0.212151	0.565356	0.392157	0.180503	0.171477	
2	0.210159	0.564293	1.000000	0.037260	0.029330	
3	0.209163	0.564293	1.000000	0.032352	0.036313	
4	0.209163	0.564293	1.000000	0.041330	0.043296	
...	...	...	...	...	...	...
20635	0.324701	0.737513	0.470588	0.042296	0.057883	
20636	0.312749	0.738576	0.333333	0.017676	0.023122	
20637	0.311753	0.732200	0.313725	0.057277	0.075109	
20638	0.301793	0.732200	0.333333	0.047256	0.063315	
20639	0.309761	0.725824	0.294118	0.070782	0.095438	

	population	households	median_income	<1H	OCEAN	INLAND	ISLAND	\
0	0.008941	0.020556	0.539668	0	0	0		
1	0.067210	0.186976	0.538027	0	0	0		
2	0.013818	0.028943	0.466028	0	0	0		
3	0.015555	0.035849	0.354699	0	0	0		
4	0.015752	0.042427	0.230776	0	0	0		
...	...	...	...	...	...	...	...	
20635	0.023599	0.054103	0.073130	0	1	0		
20636	0.009894	0.018502	0.141853	0	1	0		
20637	0.028140	0.071041	0.082764	0	1	0		
20638	0.020684	0.057227	0.094295	0	1	0		
20639	0.038790	0.086992	0.130253	0	1	0		

	NEAR_BAY	NEAR_OCEAN
0	1	0
1	1	0
2	1	0
3	1	0
4	1	0
...	...	...
20635	0	0
20636	0	0
20637	0	0
20638	0	0
20639	0	0

[20640 rows x 13 columns]

In [4]: print(df2)

0	0.902266
1	0.708247
2	0.695051
3	0.672783
4	0.674638
...	...
20635	0.130105
20636	0.128043
20637	0.159383
20638	0.143713
20639	0.153403

Name: median\_house\_value, Length: 20640, dtype: float64

In [7]: LR = [100,1000,10000]

```
for i in LR:
    #Defines linear regression model and its structure
    model = Sequential()
    model.add(Dense(1, input_shape=(13,)))

    #Compiles model
    model.compile(Adam(learning_rate=1), 'mean_squared_error')

    #Fits model
    history = model.fit(X_train, y_train, epochs = 500, validation_split = 0.1, verbose = 0)
    history_dict=history.history

    #Plots model's training cost/loss and model's validation split cost/loss
    loss_values = history_dict['loss']
    val_loss_values=history_dict['val_loss']
    plt.figure()
    plt.plot(loss_values,'bo',label='training loss')
    plt.plot(val_loss_values,'r',label='val training loss')
```

13is ως αποτέλεσμα του one hot vector encoding όπου χώρισε το χαρακτηριστικό ocean proximity στις 5 κατηγορίες που έχει.

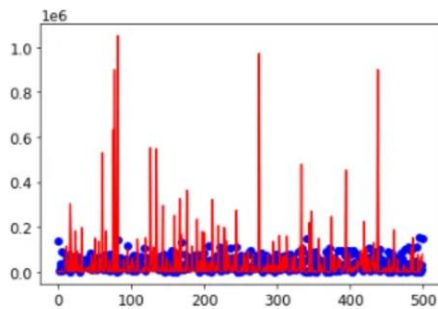
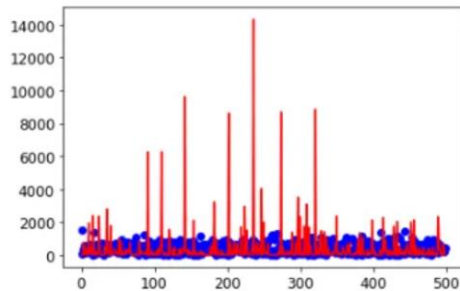
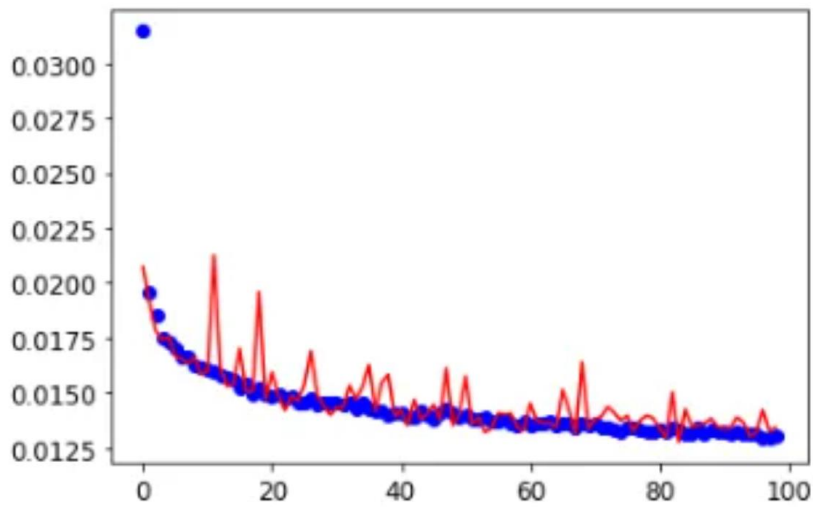
Στη συνέχεια επαναληπτικά εφαρμόζουμε τον αλγόριθμο δημιουργώντας μοντέλα με την μέθοδο της βιβλιοθήκης keras όπου το 10% των συνολικών προτύπων για εκπαίδευση θα επιλεγούν με δειγματοληψία για να ελεγχθεί το σφάλμα. Όπου θα αναπαρασταθούν με μπλε γραμμή για το train και το training test error με κόκκινη. (χωρίς να βγει ολοκληρωμένο αποτέλεσμα λόγω πτώσης συστήματος κι αδυναμία κάρτας γραφικών δεν μπορούσαν να τρέξουν πολύ ώρα τα τεστ).



Epoch 99: early stopping

The R2 score on the Train set is: 0.777

The R2 score on the Test set is: 0.776



```
# Runs and plots the performance of a model with the same parameters from before (and a learning rate of 10000  
# but now with an activation function (Relu)
```

```
model = Sequential()  
model.add(Dense(1, input_shape=(13,), activation = 'relu'))  
model.compile(Adam(learning_rate=10000), 'mean_squared_error')  
history = model.fit(X_train, y_train, epochs = 500, validation_split = 0.1, verbose = 0)
```

```
history_dict=history.history  
loss_values = history_dict['loss']  
val_loss_values=history_dict['val_loss']  
plt.plot(loss_values,'bo',label='training loss')  
plt.plot(val_loss_values,'r',label='training loss val')
```

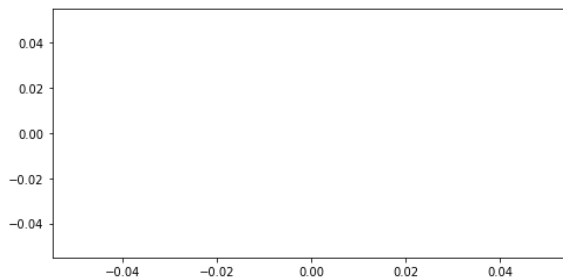
(Κάποιες φορές δεν τρέχουν τα γραφήματα)

Τα αποτελέσματα του πρώτου νευρωνικού δικτύου(με ένα κρυφό επίπεδο) βγαίνουν αρνητικά διότι έχουμε βάλει λίγα πρότυπα για εκπαίδευση, καθώς με ένα οποιοδήποτε μεγαλύτερο

νούμερο αδυναμεί το σύστημα να βγάλει γραφική αναπαράσταση σε ένα ουσιώδες χρονικό διάστημα.

Για το 9<sup>ο</sup> ερώτημα παράγουμε ένα νευρωνικό δίκτυο πολλών επιπέδων για την διαδικασία μάθησης. Με σταθερό ρυθμό μάθησης

```
plt.plot(val_loss_values, 'r', label='training loss val')  
Out[8]: [<matplotlib.lines.Line2D at 0x7fa14140dcd0>]
```



```
In [9]: #Runs model (the one with the activation function, although this doesn't really matter as they perform the sa  
# with its current weights on the training and testing data  
y_train_pred = model.predict(X_train)  
y_test_pred = model.predict(X_test)  
  
# Calculates and prints r2 score of training and testing data  
print("The R2 score on the Train set is:\t{:0.3f}".format(r2_score(y_train, y_train_pred)))  
print("The R2 score on the Test set is:\t{:0.3f}".format(r2_score(y_test, y_test_pred)))
```

```
The R2 score on the Train set is:      -2.764  
The R2 score on the Test set is:     -2.766
```

```
In [ ]: # Defines "deep" model and its structure  
model = Sequential()  
model.add(Dense(13, input_shape=(13,), activation='relu'))  
model.add(Dense(13, activation='relu'))  
model.add(Dense(13, activation='relu'))  
model.add(Dense(13, activation='relu'))  
model.add(Dense(13, activation='relu'))  
model.add(Dense(1,))  
model.compile(Adam(learning_rate=0.003), 'mean_squared_error')  
  
# Runs model for 2000 iterations and assigns this to 'history'  
history = model.fit(X_train, y_train, epochs = 6000, validation_split = 0.2, verbose = 0)  
  
# Plots 'history'  
history_dict=history.history  
loss_values = history_dict['loss']  
val_loss_values=history_dict['val_loss']  
plt.plot(loss_values, 'bo', label='training loss')  
plt.plot(val_loss_values, 'r', label='training loss val')
```

