ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

DEPARTMENT OF INFORMATICS

4/3/2022

Προσέγγιση Τιμών Ακινήτων με χρήση Αλγορίθμων Μηχανικής Μάθησης

ΓΕΩΡΓΙΟΣ ΜΠΑΛΤΖΑΚΗΣ
email: george-baltzakis@hotmail.gr

Εισαγωγή

Η παρούσα εργασία έχει ως στόχο την ανάπτυξη αλγορίθμων μηχανικής μάθησης για την προσέγγιση της διάμεσης τιμής ενός ακινήτου σε μια ευρύτερη γεωγραφική περιοχή της πολιτείας της Καλιφόρνια βάσει ενός συνόλου αντικειμενικών χαρακτηριστικών των ακινήτων στην εν λόγω περιοχή. Κάθε τέτοια περιοχή αποτελεί στην πραγματικότητα την μικρότερη γεωγραφική οντότητα που καταγράφηκε στην σχετική απογραφή του 1990 με πληθυσμιακό εύρος μεταξύ των 600 και 3000 κατοίκων. Το σχετικό σύνολο των δεδομένων είναι αποθηκευμένο στο αρχείο **"housing.csv"**.

Η διαδικασία εκπαίδευσης των εμπλεκόμενων μηχανισμών μηχανικής μάθησης θα πρέπει να βασιστεί σε ένα σύνολο αντικειμενικών γνωρισμάτων των ακινήτων που υπάρχουν σε κάθε γεωγραφική περιοχή το οποίο περιλαμβάνει τα παρακάτω χαρακτηριστικά:

i. το γεωγραφικό μήκος (longitude) του κέντρου της περιοχής.
ii. το γεωγραφικό πλάτος (latitude) του κέντρου της περιοχής.
iii. την διάμεση ηλικία των ακινήτων (housing_median_age) της περιοχής.
iv. το συνολικό πλήθος δωματίων (total_rooms) των ακινήτων της περιοχής.
v. το συνολικό πλήθος υπνοδωματίων (total_bedrooms) των ακινήτων της περιοχής.
vi. τον πληθυσμό (population) της περιοχής
vii. το πλήθος των νοικοκυριών (households) της περιοχής
viii. το διάμεσο εισόδημα (median_income) των κατοίκων της περιοχής
ix. την εγγύτητα προς τον ωκεανό (ocean_proximity) της περιοχής

ώστε να προσεγγιστεί:

x. η διάμεση τιμή (median_house_value) των ακινήτων της περιοχής.

Η υλοποίηση των μηχανισμών μάθησης, η προ επεξεργασία και οπτικοποίηση των δεδομένων, πραγματοποιήθηκε με τη βοήθεια της γλώσσας Προγραμματισμού **Python** και μερικών βιβλιοθηκών της. Πιο συγκεκριμένα, χρησιμοποιήθηκαν:

- Python 3.7.11
- Jupyter 1.0.0
- Pandas 1.3.5
- NumPy 1.21.2
- TensorFlow 2.0.0
- Scikit-Learn 1.0.2
- Matplotlib 3.5.1

1. Προ Επεξεργασία Δεδομένων:
    a. Εισαγωγή Δεδομένων (Έλεγχος για διπλότυπες εγγραφές)

```python
import numpy as np
import pandas as pd
from sklearn import preprocessing
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
```

```python
# Import Data
original_data = pd.read_csv('housing.csv')
data = pd.read_csv('housing.csv')

# Drop Duplicates
data.drop_duplicates()

# A First View of Data
print("\n\tData Info\n")
data.info()
print("\n\n\tData Head\n")
data.head()
```

```
    Data Info

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB


    Data Head
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

    b. Έλεγχος για εγγραφές που περιέχουν κενά πεδία:
        i. Αντικατάσταση των πεδίων αυτών με τη διάμεση τιμή του συγκεκριμένου χαρακτηριστικού.

```python
# 4) Search for null values
for col in data.columns:

    # Get indecies of "data DataFrame" rows, if data['col'] has null values
    null_values_index = data[col].index[pd.isnull(data[col])]

    # If null_rows_index isn't empty
    if len(null_values_index) > 0:

        # Get col's median
        median = data[col].median()

        # Replace the NaN values with median value of this column
        for data_values_index in null_values_index:
            data.at[data_values_index, col] = median
```

c. Εφαρμογή *Min Max* Αλγορίθμου Κλιμάκωσης (scaling) στο διάστημα [0,1], για τα αριθμητικά Δεδομένα.

```
# 2) MinMax Scaling to [0, 1]
data_columns_for_scaling = ['housing_median_age', 'total_rooms', 'total_bedrooms',
                            'population', 'households', 'median_income', 'median_house_value']
data[data_columns_for_scaling] = preprocessing.MinMaxScaler().fit_transform(data[data_columns_for_scaling])
```

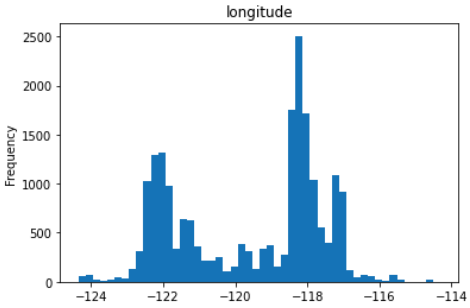
d. Εφαρμογή *One Hot Vector Encoding* για το κατηγορικό χαρακτηριστικό.

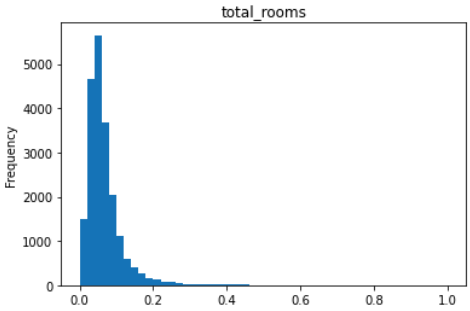
```
# 3) One Hot Vector Encoding
data.groupby('ocean_proximity')['ocean_proximity'].count()

# We save this Series for Visualization
ocean_proximity = data.ocean_proximity

# Create a One Hot Vector Encoding DataFrame
one_hot_vector_encoding = pd.get_dummies(data.ocean_proximity, prefix = 'Ocean_Proximity')

# Join that to the data DataFrame
data = data.join(one_hot_vector_encoding)

# Delete the column: ocean_proximity
data.drop(['ocean_proximity'], axis = 1, inplace = True)
```

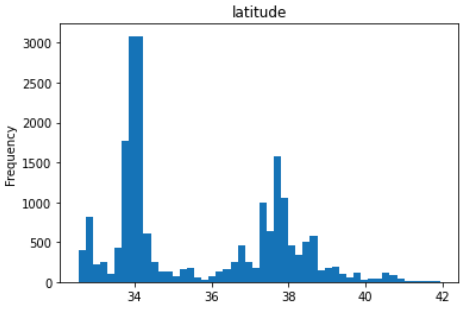
2. Οπτικοποίηση Δεδομένων:
   a. Προβολή Ιστογράμματος για κάθε χαρακτηριστικό.

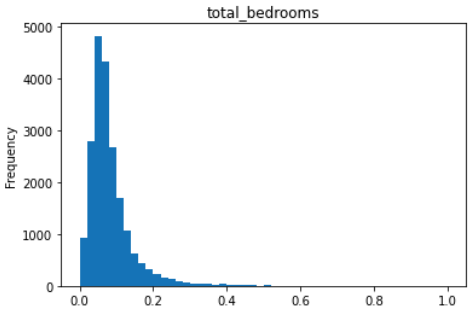
```
#
# -- Data Visualization --
#
```

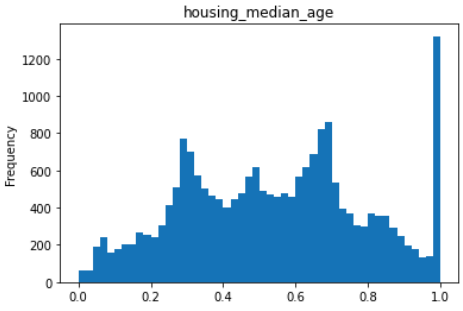
```
# 1) Frequency Histograms of each column
#
# For all numeric columns
for i in data.columns[:-5]:
    plt.figure()
    plt.hist(data[i], bins=50)
    plt.gca().set(title=i, ylabel='Frequency')

# For Categorical feature: ocean_proximity
plt.figure()
plt.title('ocean_proximity')
plt.ylabel('Frequency')
ocean_proximity.value_counts().plot(kind='bar')
plt.show()
```

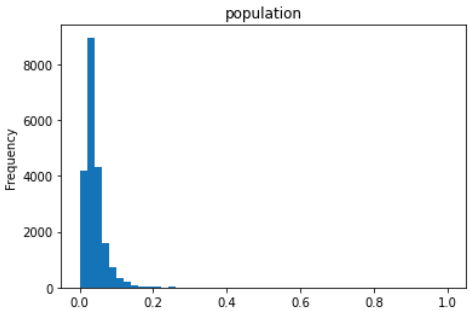

longitude
Frequency
2500
2000
1500
1000
500
0
−124
−122
−120
−118
−116
−114

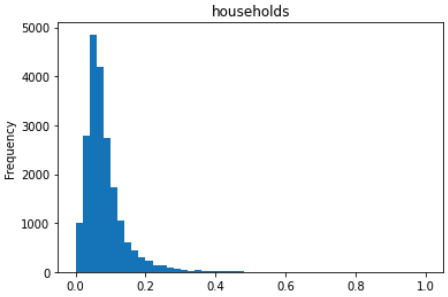

total_rooms
Frequency
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0

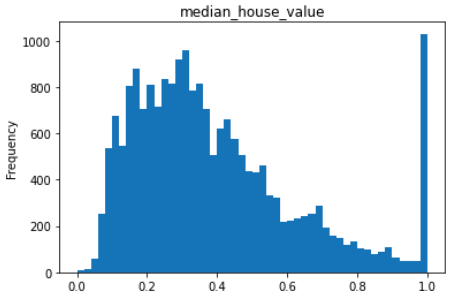

latitude
Frequency
3000
2500
2000
1500
1000
500
0
34
36
38
40
42

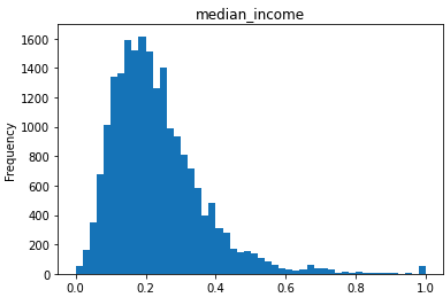

total_bedrooms
Frequency
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0

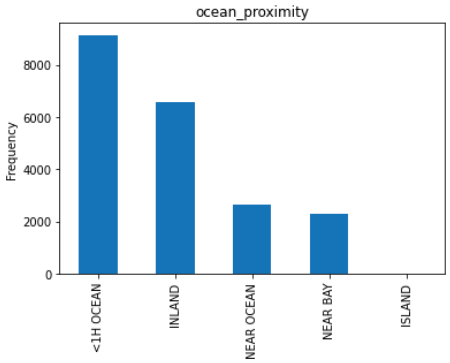

housing_median_age
Frequency
1200
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0


population
Frequency
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0


households
Frequency
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0

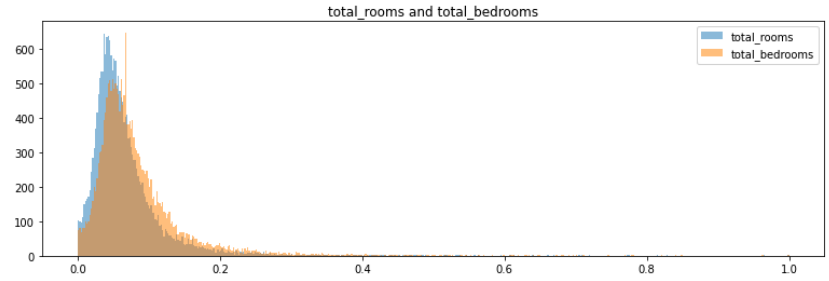

median_house_value
Frequency
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0


median_income
Frequency
1600
1400
1200
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0

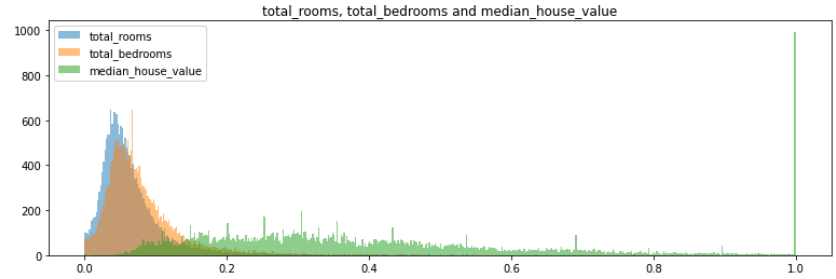

ocean_proximity
Frequency
8000
6000
4000
2000
0
<1H OCEAN
INLAND
NEAR OCEAN
NEAR BAY
ISLAND

b. Προβολή Πολλαπλών Ιστογραμμάτων.

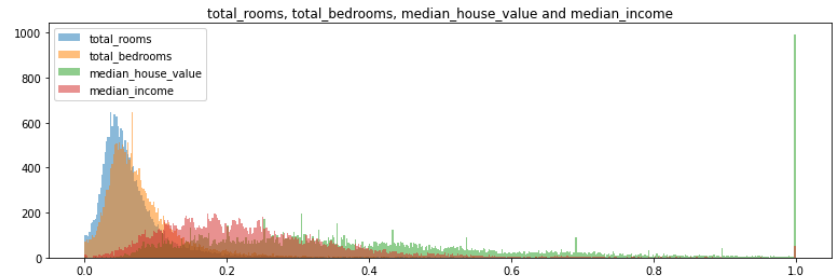
```
# 2) Histogram Overlapping

# First between total_rooms and total_bedrooms columns
plt.figure(figsize=(13, 4))
plt.title('total_rooms and total_bedrooms')
plt.hist(data.total_rooms, bins = 500, alpha = 0.5, label = 'total_rooms')
plt.hist(data.total_bedrooms, bins = 500, alpha = 0.5, label = 'total_bedrooms')
plt.legend(loc='upper right')
plt.show()
```




```
# Second between total_rooms, total_bedrooms and median_house_value
plt.figure(figsize=(13, 4))
plt.title('total_rooms, total_bedrooms and median_house_value')
plt.hist(data.total_rooms, bins = 500, alpha = 0.5, label = 'total_rooms')
plt.hist(data.total_bedrooms, bins = 500, alpha = 0.5, label = 'total_bedrooms')
plt.hist(data.median_house_value, bins = 500, alpha = 0.5, label = 'median_house_value')
plt.legend(loc='upper left')
plt.show()
```




```
# And Last between total_rooms, total_bedrooms, median_house_value and median_income
plt.figure(figsize=(13, 4))
plt.title('total_rooms, total_bedrooms, median_house_value and median_income')
plt.hist(data.total_rooms, bins = 500, alpha = 0.5, label = 'total_rooms')
plt.hist(data.total_bedrooms, bins = 500, alpha = 0.5, label = 'total_bedrooms')
plt.hist(data.median_house_value, bins = 500, alpha = 0.5, label = 'median_house_value')
plt.hist(data.median_income, bins = 500, alpha = 0.5, label = 'median_income')
plt.legend(loc='upper left')
plt.show()
```



3. Παλινδρόμηση Δεδομένων:

    a. Προετοιμασία και διαχωρισμός δεδομένων (για *Least Mean Squares* και *Least Squares*)
        i. K-Fold Cross Validation

```python
# Dataset Preparation
independent_columns = ['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
        'Ocean_Proximity_<1H OCEAN', 'Ocean_Proximity_INLAND', 'Ocean_Proximity_ISLAND',
       'Ocean_Proximity_NEAR BAY', 'Ocean_Proximity_NEAR OCEAN']
dependent_column = 'median_house_value'

X = data[independent_columns].to_numpy(dtype=float)
Y = data[dependent_column].to_numpy(dtype=float)

# 10-Fold Cross-Validation
train_index, test_index = my_KFold(X, 10)

# Define the training and testing sets
X_train, X_test = X[train_index], X[test_index]
Y_train, Y_test = Y[train_index], Y[test_index]
```

```python
# k-Fold Cross-Validation
def my_KFold(vx, k):

    kf = KFold(n_splits = k)
    vtrain_index, vtest_index = next(kf.split(vx))

    return vtrain_index, vtest_index
```

    b. *Least Mean Squares* Αλγόριθμος Γραμμικής Παλινδρόμησης:
        i. Εκπαίδευση (Train) με *Gradient Descent* και συνάρτηση απώλειας *Mean Squared Error*.

```python
# Cost of Mean Squared Error (MSE)
def mean_squared_error(y_true, y_predicted):

    # Calculating the loss or cost
    cost = np.sum((y_true-y_predicted)**2) / len(y_true)
    return cost
```

```python
# Gradient descent with MSE minimization
def gradient_descent(x, y, epochs = 1000, learning_rate = 0.00006,
                     stopping_threshold = 1e-6):

    # Initializing weight, bias, learning rate and iterations
    current_weights = np.ones(len(x[0]))
    current_bias = 0.01
    epochs = epochs
    learning_rate = learning_rate
    n = float(len(x))

    costs = []
    previous_cost = None

    # Estimation of optimal parameters
    for i in range(epochs):

        # Making predictions
        y_predicted = np.dot(x, current_weights) + current_bias

        # Calculation the current cost
        current_cost = mean_squared_error(y, y_predicted)

        # If the change in cost is less than or equal to
        # stopping_threshold we stop the gradient descent
        if previous_cost and abs(previous_cost-current_cost)<=stopping_threshold:
            break

        previous_cost = current_cost

        costs.append(current_cost)

        # Calculating the gradients
        weight_derivative = -(2/n) * np.dot(x.T,(y-y_predicted))
        bias_derivative = -(2/n) * sum(y-y_predicted)

        # Updating weights and bias
        current_weights = current_weights - (learning_rate * weight_derivative)
        current_bias = current_bias - (learning_rate * bias_derivative)

        # Printing the parameters for each Epoch
        print('\n-----------------------------------------------------------------\n')
        print('Epoch:', i+1, '\n\nMSE Cost:', current_cost, '\nMAE Cost:', mean_absolute_error(y,
         y_predicted), '\nWeights:\n', current_weights, '\nBias:', current_bias)

    return current_weights, current_bias, costs
```

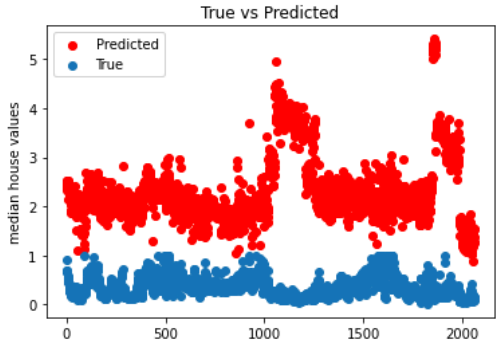
```python
#
# 1) LMS Training
#
# Estimating weights and bias using Gradient Descent with MSE minimization
ep = 500
estimated_weights, estimated_bias, costs = gradient_descent(X_train, Y_train, epochs=ep)

print('\n-----------------------------------------------------------------\n')
print('Final Estimated Weights: ', estimated_weights, '\n\nFinal Estimated Bias: ', estimated_bias)
```

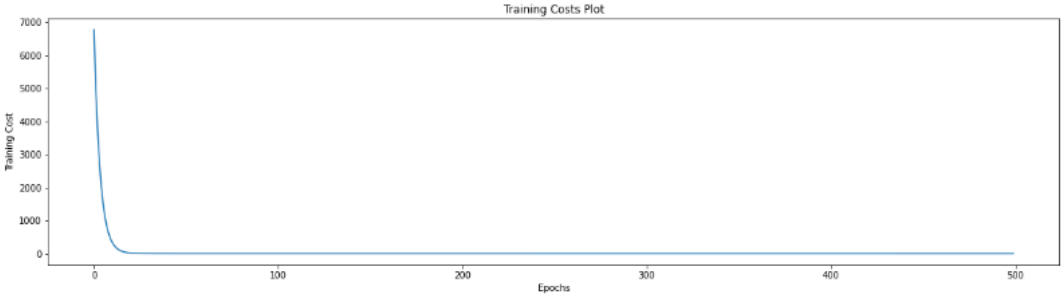
```
------------------------------------------------------------

Epoch: 1

MSE Cost: 6773.161026707132
MAE Cost: 82.29455364968523
Weights:
 [-0.17827258  1.34917855  1.00529871  1.00066246  1.00082664  1.00039892
  1.00081518  1.00230198  1.00481644  1.00310121  1.0000027   1.00053714
  1.00141786]
Bias: 0.019875346437962238

------------------------------------------------------------

Epoch: 2

MSE Cost: 5002.0922248690185
MAE Cost: 70.65578160611246
Weights:
 [0.83403302 1.0485425  1.00074646 1.00009303 1.00011676 1.00005675
 1.00011538 1.00033319 1.00077154 1.00034256 1.00000052 1.00005457
 1.00022746]
Bias: 0.011396652645228773

------------------------------------------------------------

Epoch: 3

MSE Cost: 3694.364849691922
MAE Cost: 60.77321327020683
Weights:
 [-0.0360078   1.30628716  1.00465906  1.00058215  1.00072719  1.00035139
  1.00071739  1.00203424  1.00434109  1.00261927  1.00000253  1.00044823
  1.00127832]
Bias: 0.018689438237653587
```

ii. Δοκιμή (Test), και προβολή αποτελεσμάτων

```
#
# LMS Testing
#
Y_pred = X_test.dot(estimated_weights.T) + estimated_bias

#
print('\n', 'LMS Testing MSE (with', ep, 'training Epochs):', mean_squared_error(Y_test, Y_pred))
print('\n', 'LMS Testing MAE (with', ep, 'training Epochs):', mean_absolute_error(Y_test, Y_pred))

# True vs Predicted Plot
plt.figure()
plt.scatter(list(range(len(Y_pred))), pd.DataFrame(Y_pred), label='Predicted', color='red')
plt.scatter(list(range(len(Y_pred))), pd.DataFrame(Y_test), label='True')
plt.ylabel('median house values')
plt.title('True vs Predicted')
plt.legend(loc='best')
plt.show()

# Cost plot
costs = pd.DataFrame(costs)
plt.figure(figsize=(20, 5))
plt.plot(costs)
plt.autoscale()
plt.title('Training Costs Plot')
plt.ylabel('Training Cost')
plt.xlabel('Epochs')
plt.show()
```

```
LMS Testing MSE (with 500 training Epochs): 4.420479071378372

LMS Testing MAE (with 500 training Epochs): 1.9724113151838392
```





c. *Least Squares* Αλγόριθμος Γραμμικής Παλινδρόμησης:

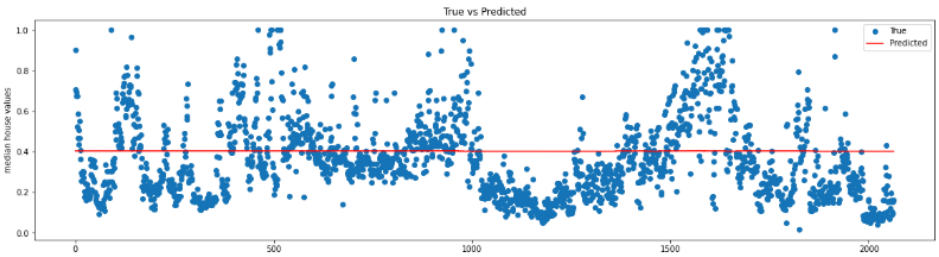
i. Υλοποίηση Αλγορίθμου

```
#
# 2) LS Building the model
#
m = []
X_mean = []
Y_mean = np.mean(Y_train)

for i in range(len(X_train[0])):
    X_mean.append(np.mean(X_train[i]))
    num = 0
    den = 0
    for j in range(len(X_train)):
        num += (X_train[j, i] - X_mean[i])*(Y_train[j] - Y_mean)
        den += (X_train[j, i] - X_mean[i])**2
    m.append(num / den)

c = Y_mean - np.dot(np.array(m), np.array(X_mean))
```

ii. Δοκιμή (Test), και προβολή αποτελεσμάτων

```
#
# LS Testing
#
LS_predicted = np.dot(np.array(m), X_test.T) + c

print('\n', 'LS Testing MSE:', mean_squared_error(Y_test, LS_predicted))
print('\n', 'LS Testing MAE:', mean_absolute_error(Y_test, LS_predicted))

# True vs Predicted Plot
plt.figure(figsize=(20, 5))
plt.scatter(list(range(len(LS_predicted))), pd.DataFrame(Y_test), label='True')
plt.plot(pd.DataFrame(LS_predicted), label='Predicted', color='red')
plt.ylabel('median house values')
plt.title('True vs Predicted')
plt.legend(loc='best')
plt.show()
```

```
LS Testing MSE: 0.042155660171509485

LS Testing MAE: 0.16764192465561106
```



d. Μη-Γραμμική Παλινδρόμηση με πολυστρωματικό Νευρωνικό Δίκτυο:

i. Προετοιμασία Δεδομένων

```
#
# Non-Linear Regression with Neural Networks
#

# Preparation

# Split to train, validation, test sets
X_train_validation = X_train.copy()
Y_train_validation = Y_train.copy()

train_index, validation_index = my_KFold(X_train_validation, 10)

X_train = X_train_validation[train_index]
Y_train = Y_train_validation[train_index]

X_validation = X_train_validation[validation_index]
Y_validation = Y_train_validation[validation_index]
```

ii. Δημιουργία και Εκπαίδευση (Train) Μοντέλου

```
# Create the Model
model = Sequential()
model.add(Dense(128, activation="relu", input_dim=13))
model.add(Dense(32, activation="relu"))
model.add(Dense(8, activation="relu"))
# Since the regression is performed, a Dense layer containing a single neuron with a linear activation function.
# Typically, ReLu-based activation are used but since it is performed regression, it is needed a linear activation.
model.add(Dense(1, activation="linear"))

# Compile model: The model is initialized with the Adam optimizer and then it is compiled.
model.compile(loss='mean_absolute_error', optimizer=Adam(lr=1e-3, decay=1e-3 / 200))

# Patient early stopping
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=200)

# Fit the model
history = model.fit(X_train, Y_train, validation_data=(X_validation, Y_validation), epochs=50, batch_size=100, verbose=2, callbacks=[es])
```

iii. Δοκιμή (Test), προβολή αποτελεσμάτων

```
#
# NN Testing
#

# Calculate predictions
NN_predicted = model.predict(X_test)

print('\n', 'NN Testing MSE:', mean_squared_error(Y_test, NN_predicted))
print('\n', 'NN Testing MAE:', mean_absolute_error(Y_test, NN_predicted))

# True vs Predicted Plot
plt.figure(figsize=(20, 5))
plt.scatter(list(range(len(NN_predicted))), pd.DataFrame(Y_test), label='True')
plt.scatter(list(range(len(NN_predicted))), pd.DataFrame(NN_predicted), label='Predicted',
 color='red')
plt.ylabel('median house values')
plt.title('True vs Predicted')
plt.legend(loc='best')
plt.show()
```

```
NN Testing MSE: 185.6908787764589

NN Testing MAE: 0.14050895666509014
```