**Κατηγοριοποίηση όγκων σε καλοήθεις και κακοήθεις**

**Μαλωνάς Κωνσταντίνος**

**Α.Μ:45226**

**Εξ: 13$^{ο}$**

# Περιεχόμενα

# 1. Επιλογή μεθόδου Εξόρυξης Δεδομένων.

Για την εξόρυξη των δεδομένων - γνώσης από το **dataset** που θα μελετήσουμε, θα χρησιμοποιήσουμε την μέθοδο της κατηγοριοποίησης. Η **κατηγοριοποίηση** είναι μία τεχνική της εξόρυξης δεδομένων, κατά την οποία ένα στοιχείο ανατίθεται σε ένα προκαθορισμένο σύνολο κατηγοριών. Γενικότερα, ο στόχος της διαδικασίας αυτής είναι η ανάπτυξη ενός μοντέλου, το οποίο αργότερα θα μπορεί να χρησιμοποιηθεί για την κατηγοριοποίηση μελλοντικών δεδομένων. Τέτοια παραδείγματα είναι ο διαχωρισμός των emails με βάση την επικεφαλίδα τους ή το περιεχόμενό τους, η πρόβλεψη καρκινικών κυττάρων χαρακτηρίζοντας τα ως καλοήθη ή κακοήθη, η κατηγοριοποίηση πελατών μιας τράπεζας ανάλογα με την πιστωτική τους ικανότητα κ.α.

Η κατηγοριοποίηση μπορεί να περιγραφεί ως μία διαδικασία δύο βημάτων:

1. **Εκμάθηση(Learning)**:Στο πρώτο βήμα της διαδικασίας δημιουργείται/προσδιορίζεται το μοντέλο με βάση ένα σύνολο προκατηγοριοποιημένων παραδειγμάτων, που ονομάζεται δεδομένα εκπαίδευσης (**training data**).Τα δεδομένα εκπαίδευσης αναλύονται από ένα αλγόριθμο κατηγοριοποίησης, προκειμένου να σχηματιστεί το μοντέλο. Λόγω του ότι τα δεδομένα εκπαίδευσης ανήκουν σε μία προκαθορισμένη κατηγορία, η οποία είναι γνωστή, η κατηγοριοποίηση αποτελεί μέθοδος εποπτευομένης μάθησης(supervised **learning**). Το μοντέλο (**classifier**), αναπαρίσταται με τη μορφή κανόνων κατηγοριοποίησης(**classification rules**), δέντρων απόφασης(**decision trees**) ή μαθηματικών τύπων.

2. **Κατηγοριοποίηση(Classification)**: Μετά την δημιουργία του μοντέλου, το επόμενο βήμα είναι η αξιολόγησή του. Για να επιτευχθεί αυτό, χρησιμοποιούμε τα δοκιμαστικά δεδομένα(**test data**) για να υπολογίσουν την ακρίβεια του μοντέλου. Το μοντέλο κατηγοριοποιεί τα δοκιμαστικά δεδομένα. Έπειτα, η κατηγορία που σχηματίστηκε με βάση τα δοκιμαστικά δεδομένα συγκρίνεται με την πρόβλεψη που έγινε για τα δεδομένα εκπαίδευσης, τα οποία είναι ανεξάρτητα από αυτά της δοκιμής. Η ακρίβεια του μοντέλου υπολογίζεται από το ποσοστό των δειγμάτων δοκιμής που κατηγοριοποιήθηκαν σωστά σε σχέση με το υπό εκπαίδευση μοντέλο.

Στην περίπτωση που το μοντέλο κριθεί αποδεκτό, τότε μπορεί να χρησιμοποιηθεί για την κατηγοριοποίηση μελλοντικών δειγμάτων δεδομένων, των οποίων η κατηγοριοποίηση είναι άγνωστη.

## 2. Επιλογή σεναρίου.

Σαν σενάριο πάνω στο οποίο θα εφαρμόσουμε τον αλγόριθμο μας για την εξόρυξη δεδομένων είναι αυτό ενός οργανισμού διάγνωσης και αντιμετώπισης **καρκινικών όγκων**. Μέσα από την συλλογή δεδομένων που έχουμε στην διάθεση μας (**dataset**), η οποία αφορά ασθενείς και τα χαρακτηριστικά των όγκων που έχουν παρουσιάσει και τους οποίους κατατάσσει σε **καλοήθεις** ή **κακοήθεις** όγκους , θα εκπαιδεύσουμε το πρόγραμμα μας ώστε να πετυχαίνει με μεγάλο ποσοστό ακρίβειας τον τύπο του όγκου του κάθε ασθενή (καλοήθεις, κακοήθεις). Το συγκεκριμένο σενάριο έχει ιδιαίτερη **αναγκαιότητα** και **χρησιμότητα** καθώς θα αυξάνει την βεβαιότητα για την ορθότητα της εκάστοτε διάγνωσης για κάθε ασθενή, θα μπορεί να συμβάλλει στην μείωση των περιττών εξετάσεων και στην πιο έγκαιρη και στοχευμένη αντιμετώπιση και θεραπεία στις περιπτώσεις ασθενών που απαιτείται.

## 3. Επιλογή δεδομένων.

Επιλέξαμε ένα σύνολο δεδομένων που έχει στοιχεία για τον όγκο που έχει εμφανίσει ο κάθε ασθενής. Τα δεδομένα επιλέχθηκαν από το Link1 και τα μετατρέψαμε σε μορφή **XML** με το εργαλείο που βρήκαμε στο Link2. Τα δεδομένα μας αποτελούνται από **569 εγγραφές**, που κάθε εγγραφή χαρακτηρίζει τον όγκο κάποιου ασθενή. Συγκεκριμένα κάθε εγγραφή περιέχει στοιχεία όπως η διάγνωση, δηλαδή αν ο όγκος είναι καλοήθης ή κακοήθεις, τον μέσο όρο της ακτίνας σαν **radius_mean**, τον μέσο όρο της περιμέτρου σαν **perimeter_mean** κ.α. Κάθε εγγραφή, πριν την επεξεργασία των δεδομένων που θα πραγματοποιηθεί στην συνέχεια έχει 33 χαρακτηριστικά (στήλες). Όλες οι τιμές κάθε εγγραφής είναι τύπου **float**, εκτός από τα πεδία **id** (Ο αριθμός κάθε ασθενή. Θα διαγραφεί κατά την επεξεργασία δεδομένων.) που είναι τύπου **int**, του **diagnosis** που είναι τύπου **string** και του **FIELD33** (Πεδίο που προστέθηκε κατά την αυτόματη μετατροπή του **csv** σε **xml**. Θα διαγραφεί κατά την επεξεργασία δεδομένων.) που είναι τύπου **string**. Οι παραπάνω τύποι τιμών που μπορούμε να τους δούμε με την εντολή **df_1.info()** που βρίσκεται μέσα στην συνάρτηση **process_data** που περιγράφεται παρακάτω ισχύουν μόνο για όταν μετατρέπουμε το **csv** σε **pandas dataframe**. Αφού το ξαναμετατρέψουμε σε **csv** με όνομα **data_2** και τελικά στο **final_data** όλα τα δεδομένα που θα φορτωθούν θα είναι τύπου **str**, οπότε θα γίνουν οι απαραίτητες ενέργειες που περιγράφονται στο **ερώτημα 4** για να μετατραπούν τα δεδομένα στους προαναφερθέντες τύπους (**float**, **int**).

## Τύπος τιμών του dataset.

```
In [21]: df_1.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

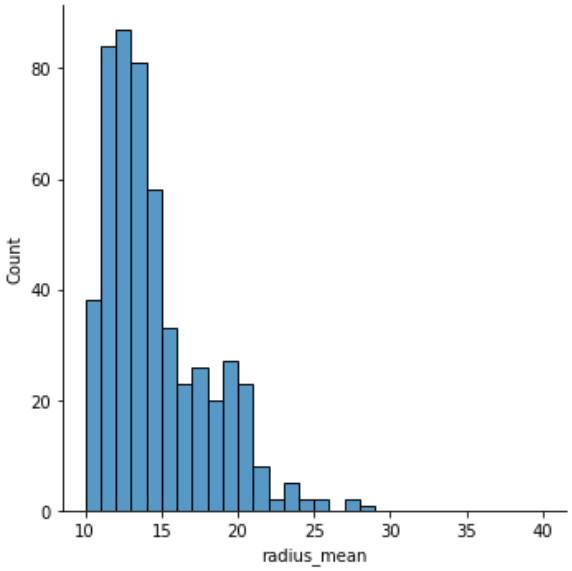
## Μέγιστη και ελάχιστη τιμή κάθε χαρακτηριστικού.

```
Min value of  radius_mean  is 6.981  and max value is  28.11
Min value of  texture_mean  is 9.71  and max value is  39.28
Min value of  perimeter_mean  is 43.79  and max value is  188.5
Min value of  area_mean  is 143.5  and max value is  2501.0
Min value of  smoothness_mean  is 0.05263  and max value is  0.1634
Min value of  compactness_mean  is 0.01938  and max value is  0.3454
Min value of  concavity_mean  is 0.0  and max value is  0.4268
Min value of  concave points_mean  is 0.0  and max value is  0.2012
Min value of  symmetry_mean  is 0.106  and max value is  0.304
Min value of  fractal_dimension_mean  is 0.04996  and max value is  0.09744
Min value of  radius_se  is 0.1115  and max value is  2.873
Min value of  texture_se  is 0.3602  and max value is  4.885
Min value of  perimeter_se  is 0.757  and max value is  21.98
Min value of  area_se  is 6.802000000000005  and max value is  542.2
Min value of  smoothness_se  is 0.001713  and max value is  0.03113
Min value of  compactness_se  is 0.002252  and max value is  0.1354
Min value of  concavity_se  is 0.0  and max value is  0.396
Min value of  concave points_se  is 0.0  and max value is  0.05279
Min value of  symmetry_se  is 0.007882  and max value is  0.07895
Min value of  fractal_dimension_se  is 0.000894800000000002  and max value is  0.02984
Min value of  radius_worst  is 7.93  and max value is  36.04
Min value of  texture_worst  is 12.02  and max value is  49.54
Min value of  perimeter_worst  is 50.41  and max value is  251.2
Min value of  area_worst  is 185.2  and max value is  4254.0
Min value of  smoothness_worst  is 0.07117000000000001  and max value is  0.2226
```

```
Min value of  compactness_worst  is 0.02729  and max value is  1.058
Min value of  concavity_worst  is 0.0  and max value is  1.252
Min value of  concave points_worst  is 0.0  and max value is  0.29100000000000004
Min value of  symmetry_worst  is 0.1565  and max value is  0.6638
Min value of  fractal_dimension_worst  is 0.05504  and max value is  0.2075
```

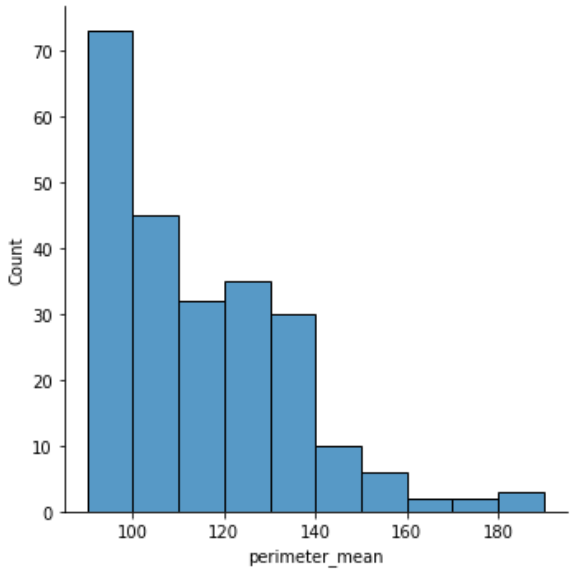
## Περαιτέρω μελέτη των δεδομένων.

**Distribution του radius_mean.**



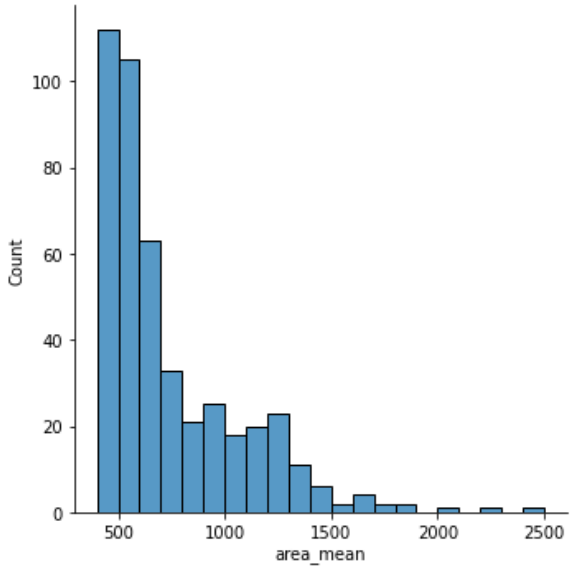
Οι τιμές που εμφανίζονται περισσότερο για το **radius_mean** είναι μεταξύ **12** και **13**.

**Distribution του perimeter_mean.**



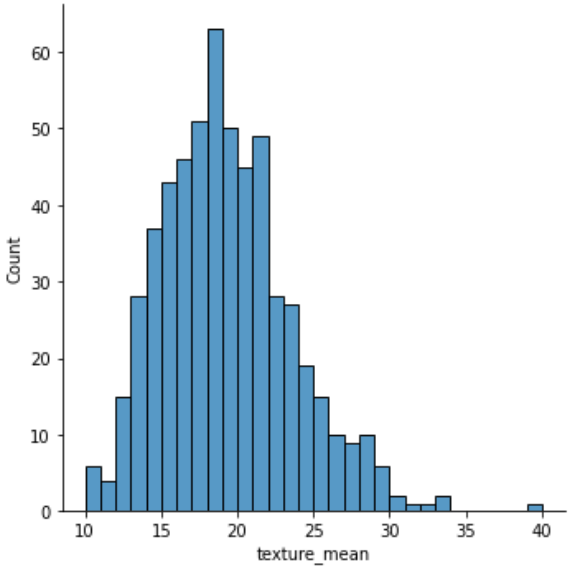

Οι τιμές που εμφανίζονται περισσότερο για το **perimeter_mean** είναι μεταξύ **90** και **100**.

**Distribution του area_mean.**




Οι τιμές που εμφανίζονται περισσότερο για το **area_mean** είναι μεταξύ **400** και **500**.

**Distribution του texture_mean.**




Οι τιμές που εμφανίζονται περισσότερο για το **texture_mean** είναι μεταξύ **18** και **19**.

**Mean.**

Το **mean** είναι ο μέσος όρος των τιμών κάθε ξεχωριστού χαρακτηριστικού.

```
In [62]: df_dataset.mean()
Out[62]:
diagnosis                 0.372583
radius_mean              14.127292
texture_mean             19.289649
perimeter_mean           91.969033
area_mean               654.889104
smoothness_mean           0.096360
compactness_mean          0.104341
concavity_mean            0.088799
concave points_mean       0.048919
symmetry_mean             0.181162
fractal_dimension_mean    0.062798
radius_se                 0.405172
texture_se                1.216853
perimeter_se              2.866059
area_se                  40.337079
smoothness_se             0.007041
compactness_se            0.025478
concavity_se              0.031894
concave points_se         0.011796
symmetry_se               0.020542
fractal_dimension_se      0.003795
radius_worst             16.269190
texture_worst            25.677223
perimeter_worst         107.261213
area_worst              880.583128
smoothness_worst          0.132369
compactness_worst         0.254265
concavity_worst           0.272188
concave points_worst      0.114606
symmetry_worst            0.290076
fractal_dimension_worst   0.083946
dtype: float64
```

**Median.**

Το **median** είναι η τιμή που χωρίζει ένα σύνολο χαρακτηριστικών στα δύο. Μπορεί να εκληφθεί σαν την μεσαία τιμή.

```
In [63]: df_dataset.median()
Out[63]:
diagnosis                  0.000000
radius_mean               13.370000
texture_mean              18.840000
perimeter_mean            86.240000
area_mean                551.100000
smoothness_mean            0.095870
compactness_mean           0.092630
concavity_mean             0.061540
concave points_mean        0.033500
symmetry_mean              0.179200
fractal_dimension_mean     0.061540
radius_se                  0.324200
texture_se                 1.108000
perimeter_se               2.287000
area_se                   24.530000
smoothness_se              0.006380
compactness_se             0.020450
concavity_se               0.025890
concave points_se          0.010930
symmetry_se                0.018730
fractal_dimension_se       0.003187
radius_worst              14.970000
texture_worst             25.410000
perimeter_worst           97.660000
area_worst               686.500000
smoothness_worst           0.131300
compactness_worst          0.211900
concavity_worst            0.226700
concave points_worst       0.099930
symmetry_worst             0.282200
fractal_dimension_worst    0.080040
dtype: float64
```

**Std.**

Το **std** περιγράφει το πόσο απέχουν οι τιμές κάθε χαρακτηριστικού από το **mean**.

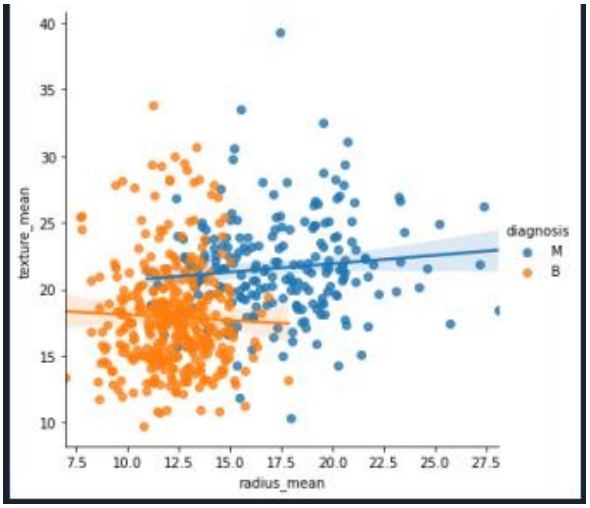
```
In [64]: df_dataset.std()
Out[64]:
diagnosis                 0.483918
radius_mean               3.524049
texture_mean              4.301036
perimeter_mean           24.298981
area_mean               351.914129
smoothness_mean           0.014064
compactness_mean          0.052813
concavity_mean            0.079720
concave points_mean       0.038803
symmetry_mean             0.027414
fractal_dimension_mean    0.007060
radius_se                 0.277313
texture_se                0.551648
perimeter_se              2.021855
area_se                  45.491006
smoothness_se             0.003003
compactness_se            0.017908
concavity_se              0.030186
concave points_se         0.006170
symmetry_se               0.008266
fractal_dimension_se      0.002646
radius_worst              4.833242
texture_worst             6.146258
perimeter_worst          33.602542
area_worst              569.356993
smoothness_worst          0.022832
compactness_worst         0.157336
concavity_worst           0.208624
concave points_worst      0.065732
symmetry_worst            0.061867
fractal_dimension_worst   0.018061
dtype: float64
```

## Διαγράμματα χαρακτηριστικών όγκου και κατάταξης τους σε καλοήθη ή κακοήθη.

**Κώδικας.**

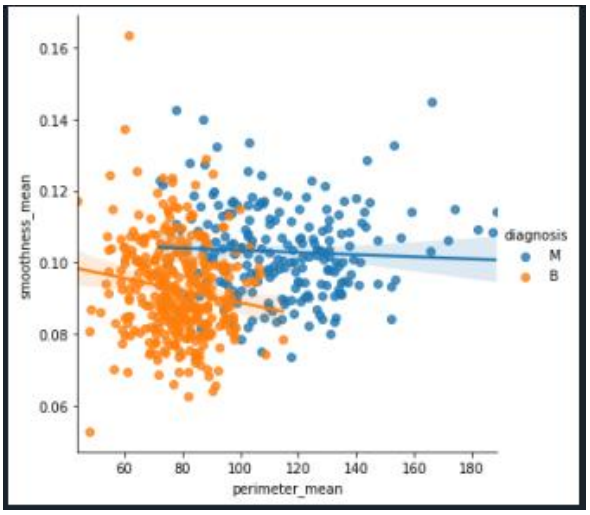
```
#Σχεδιασμός διαγραμμάτων ως προς τα χαρακτηριστικά των όγκων και κατάταξη τους σε καλοήθεις και κακοήθεις.
df = pd.read_csv('C:/Users/user/Desktop/ERGASIES_&_ARXEIA/Διαχείριση_Γνώσης_2/data.csv')
sns.lmplot(x = 'radius_mean', y = 'texture_mean', hue = 'diagnosis', data = df)
sns.lmplot(x = 'perimeter_mean', y = 'smoothness_mean', hue = 'diagnosis', data = df)
sns.lmplot(x = 'area_mean', y = 'compactness_mean', hue = 'diagnosis', data = df)
```

**texture_mean/radius_mean.**



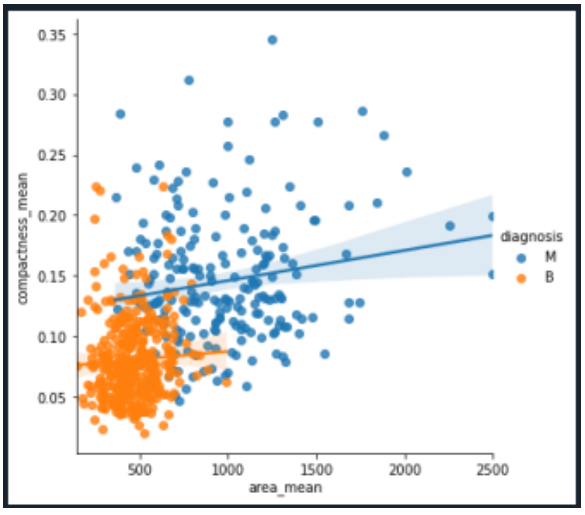
Παρατηρούμε πως όσο μεγαλύτερη η **ακτίνα** τόσο μεγαλύτερη η πιθανότητα ο όγκος του ασθενή να είναι κακοήθης.

**perimeter_mean/ smoothness_mean.**




Όσο μεγαλύτερη η **περίμετρος** η περιπτώσεις κακοηθών όγκων πληθαίνουν.

**compactness_mean/area_mean**




Διαπιστώνουμε πως όσο μεγαλύτερο είναι το **area_mean** και το **compactness_mean** του όγκου υπάρχει μεγαλύτερη πιθανότητα να είναι κακοήθης.

## 4. Προ-επεξεργασία δεδομένων.

Χρειάστηκε να γίνει προσεκτική μελέτη των δεδομένων και των τιμών που θα συμβάλλουν στην πρόβλεψη που θέλουμε να κάνουμε και των δεδομένων που δεν παίζουν κάποιο ρόλο για την κατηγοριοποίηση των αγνώστων εγγραφών για τα οποία θα προβλέψουμε το **target variable** τους (**diagnosis**), ώστε τα τελευταία να αφαιρεθούν. Παρακάτω βλέπουμε τον κώδικα που χρησιμοποιήθηκε για την επεξεργασία των δεδομένων μας.

### Συναρτήσεις για την επεξεργασία και την μελέτη των δεδομένων.

**str_column_to_float.**

```
def str_column_to_float(dataset, column):
    for row in dataset:
        row[column] = float(row[column].strip())
```

**Επεξήγηση κώδικα.**

Κάνουμε όλες τις τιμές του **dataset** τύπου **float**. Μέσα στο **for loop** παίρνουμε κάθε στήλη του **dataset** και μετατρέπουμε την τιμή που βρίσκεται εκεί από **str** σε **float**.

**load_data.**

```
def load_csv(filename):
    dataset = list()
    with open(filename, 'r') as file:
        csv_reader = reader(file)
        for row in csv_reader:
            if not row:
                continue
            dataset.append(row)
    return dataset
```

**Επεξήγηση κώδικα.**

Επιστρέφει μια λίστα με λίστες, που κάθε λίστα ξεχωριστά αντιπροσωπεύει τα χαρακτηριστικά του όγκου ενός ασθενή. Αρχικά δημιουργείται μια κενή λίστα με όνομα **dataset**. Στην συνέχεια διαβάζουμε τα δεδομένα από κάθε σειρά του **csv file** που έχουμε προσδιορίσει με το **path** που έχουμε εκχωρήσει στο **filename** και κάνουμε κάθε φορά **append** την σειρά που διαβάζουμε στην λίστα **dataset**. Τέλος η λίστα **dataset**, η οποία περιέχει τα δεδομένα κάθε σειράς του **csv** που διαβάσαμε προηγουμένως, σε μορφή λίστας επίσης, επιστρέφεται.

**process_data.**

```
def process_data():
    df_1 = pd.read_csv('C:/Users/user/Desktop/ERGASIES_&_ARXEIA/Διαχείριση_Γνώσης_2/data.csv')
    df_1.info()
    df_2 = df_1.drop(['Unnamed: 32', 'id'], index=None, axis = 1)
    df_2['diagnosis'] = df_2['diagnosis'].apply(diagnosis_value)
    df_2.to_csv('C:/Users/user/Desktop/ERGASIES_&_ARXEIA/Διαχείριση_Γνώσης_2/data_2.csv', index=False)

    with open('C:/Users/user/Desktop/ERGASIES_&_ARXEIA/Διαχείριση_Γνώσης_2/data_2.csv') as f:
        with open('C:/Users/user/Desktop/ERGASIES_&_ARXEIA/Διαχείριση_Γνώσης_2/final_data.csv','w') as f1:
            next(f) # skip header line
            for line in f:
                f1.write(line)

    dataset = load_csv('C:/Users/user/Desktop/ERGASIES_&_ARXEIA/Διαχείριση_Γνώσης_2/final_data.csv')
    for i in range(len(dataset[0])):
        str_column_to_float(dataset, i)
    #Κάνουμε την πρώτη στήλη int.
    for i in range(len(dataset)):
        dataset[i][0] = int(dataset[i][0])

    return dataset
```

**Επεξήγηση κώδικα:**

Φορτώνουμε τα δεδομένα μας από το **csv file** και κάνουμε **drop**, τις στήλες **Unnamed: 32** και την **id**. Στην συνέχεια καλούμε την συνάρτηση **diagnosis_value** και κάνουμε το **M -> 1** και το **B -> 0** από την στήλη **diagnosis**. Στην συνέχεια εξάγουμε τα δεδομένα σε μορφή **csv** στον φάκελο **data_2.csv**. Κάνουμε **open** το παραπάνω αρχείο, κάνουμε **skip** μια γραμμή και γράφουμε στο αρχείο **final_data** τα δεδομένα μας. Έπειτα καλούμε την συνάρτηση **load_csv** που επιστρέφει μια λίστα που περιέχει λίστες. Κάθε περιεχόμενη λίστα είναι και μια εγγραφή που αφορά τα δεδομένα για τον όγκο ενός ασθενή μας. Ακολούθως κάνουμε όλες τις τιμές **float** και στην συνέχεια μόνο την πρώτη τιμή από κάθε περιεχόμενη λίστα **int**. Τέλος επιστρέφουμε την λίστα με όνομα **dataset**.

**study_data.**

```
def study_data():
    df_dataset = pd.read_csv('C:/Users/user/Desktop/ERGASIES_&_ARXEIA/Διαχείριση_Γνώσης_2/data_2.csv')

    print(df_dataset.mean())
    print(df_dataset.median())
    print(df_dataset.std())

    sns.displot(df_dataset, x = df_dataset['radius_mean'], bins = list(range(10, 41)))
    sns.displot(df_dataset, x = df_dataset['perimeter_mean'], bins = list(range(90,200,10)))
    sns.displot(df_dataset, x = df_dataset['area_mean'], bins = list(range(400, 2600, 100)))
    sns.displot(df_dataset, x = df_dataset['texture_mean'], bins = list(range(10, 41)))
```

**Επεξήγηση κώδικα.**

Συνάρτηση για περαιτέρω μελέτη των δεδομένων που έχουμε επιλέξει. Δημιουργούμε ένα **pandas dataframe** από το αρχείο **data_2.csv** που είναι ένα **csv file** που η πρώτη γραμμή περιέχει τα ονόματα των στηλών. Βρίσκουμε και τυπώνουμε το **mean**, το **median** και το **std**. Εμφανίζουμε **4 bar plots** με την βιβλιοθήκη **seaborn** στα οποία μπορούμε να δουμε το **distribution** ορισμένων δεδομένων του **dataset** μας. Εμείς βρίσκουμε το **distribution** του **radius_mean**, του **perimeter_mean**, του **area_mean** και του **texture_mean**.

### Dataset πριν την επεξεργασία δεδομένων.

```
In [4]: df_2.head()
Out[4]:
  diagnosis  radius_mean  ...  symmetry_worst  fractal_dimension_worst
0         M        17.99  ...          0.4601                  0.11890
1         M        20.57  ...          0.2750                  0.08902
2         M        19.69  ...          0.3613                  0.08758
3         M        11.42  ...          0.6638                  0.17300
4         M        20.29  ...          0.2364                  0.07678

[5 rows x 31 columns]
```

### Dataset μετά την επεξεργασία δεδομένων.

```
In [6]: df_2.head()
Out[6]:
   diagnosis  radius_mean  ...  symmetry_worst  fractal_dimension_worst
0          1        17.99  ...          0.4601                  0.11890
1          1        20.57  ...          0.2750                  0.08902
2          1        19.69  ...          0.3613                  0.08758
3          1        11.42  ...          0.6638                  0.17300
4          1        20.29  ...          0.2364                  0.07678

[5 rows x 31 columns]
```

## 5. Εξόρυξη δεδομένων.

Για το σενάριο μας επιλέξαμε να χρησιμοποιήσουμε την μέθοδο **knn** (k κοντινότερων γειτόνων).

### Θεωρητικός τρόπος λειτουργίας.

Ο αλγόριθμος των K - κοντινότερων γειτόνων είναι από τους πιο διαδεδομένους αλγόριθμους μηχανικής μάθησης που χρησιμοποιείται για ταξινόμηση προτύπων σε ομάδες αλλά και σε κάποιες περιπτώσεις για την μάθηση συναρτήσεων. Ο γενικός αλγόριθμος έχει ως ακολούθως:

1. Δημιουργία συνόλου εκπαίδευσης $\mathbf{S = \{X_1, X_2, ..., X_N\}}$ , όπου $\mathbf{X_i \in R}$ .

2. Καθορισμός της παραμέτρου **K** . Συνήθως οι τιμές αυτής της παραμέτρου είναι μονοί αριθμοί.

3. Για κάθε νέο πρότυπο $\mathbf{X_i}$ .

   (a) Δημιουργία του συνόλου $\mathbf{S_X}$ με τους **K** κοντινότερους γείτονες από το σύνολο **S** . Για την εύρεση των γειτόνων χρησιμοποιούνται διάφορα κριτήρια απόστασης με το πιο συνηθισμένο την **Ευκλείδεια απόσταση**.

   (b) Εύρεση της κατηγορίας **Y** που πλειοψηφεί στο σύνολο $\mathbf{S_X}$ .

   (c) Ανάθεση του προτύπου στην κατηγορία **Y**. Ο αλγόριθμος αυτός είναι ένας μη παραμετρικός αλγόριθμος μάθησης καθώς δεν απαιτείται η μάθηση κάποιου συνόλου παραμέτρων για την κατηγοριοποίηση. Η μεταβλητή **K** είναι η μόνη που

χρησιμοποιείται και σε πολλές περιπτώσεις η χρήση ενός σχετικά μικρού μονού αριθμού είναι αρκετή.

## Τρόπος λειτουργίας του αλγορίθμου με υλοποίηση Python.

Το πρόγραμμα χωρίζεται σε 3 μέρη. Στην **επεξεργασία και μελέτη των δεδομένων**, στην **αξιολόγηση του μοντέλου** και στις **προβλέψεις** για τα δεδομένα που έχουμε στο dataset μας και στις **προβλέψει**ς πάνω σε **νέα δεδομένα** που προσθέτουμε εμείς είτε χειροκίνητα είτε προσθέτονται αυτόματα με τυχαίες τιμές. Η επεξήγηση της επεξεργασίας και μελέτης δεδομένων περιγράφηκε στο **ερώτημα 4** οπότε **παραλείπεται**.

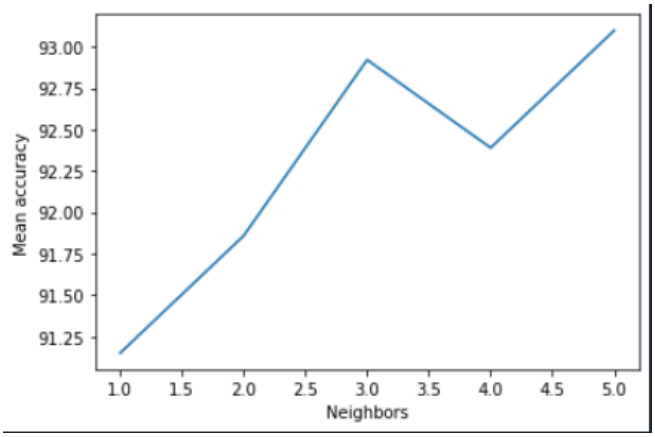
### Συναρτήσεις για την αξιολόγηση του μοντέλου.

**find_best_K.**

```
def find_best_K(K, dataset):

    mean_acc = []
    neighbors = []
    for num in range(K):
          scores = evaluate_algorithm(dataset, k_nearest_neighbors, 5, num + 1)
          print('Scores: %s' % scores)
          print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
          mean_acc.append((sum(scores)/float(len(scores))))
          neighbors.append(num + 1)


    plt.plot(neighbors, mean_acc)
    plt.xlabel('Neighbors')
    plt.ylabel('Mean accuracy')
    plt.show()
```

**Επεξήγηση κώδικα.**

Η συνάρτηση **find_best_K** χρησιμεύει στο να βρούμε τον βέλτιστο αριθμό γειτόνων με τους οποίους πετυχαίνουμε το μέγιστο **accuracy**. Δέχεται σαν παραμέτρους τους αριθμούς των γειτόνων με όνομα **K** και το **dataset**. Καλεί μέσα στο **for loop** την συνάρτηση **evaluate_algorithm** για κάθε τιμή του **K** (από 1 έως τον αριθμό που έχουμε δώσει σαν είσοδο). Ακόμα κάνει **append** στην λίστα **mean_acc** το **mean accuracy** που πετυχαίνεται κάθε φορά με κάθε ξεχωριστή τιμή του **K** και στην λίστα **neighbors** αποθηκεύει την τιμή του **K**. Τέλος σχεδιάζει την γραφική παράσταση του **mean_accuracy** ως προς το **K**. Παρακάτω βλέπουμε το διάγραμμα που τυπώνεται.



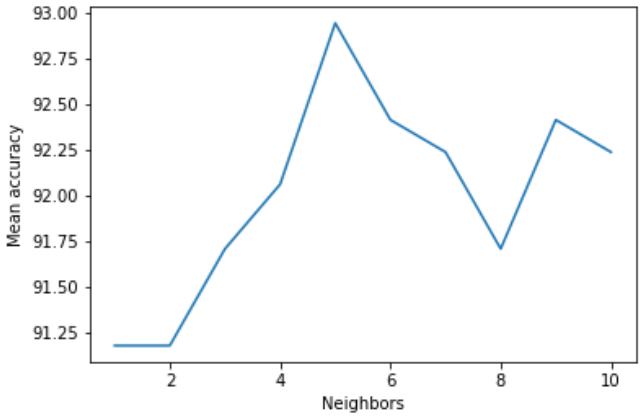

Όπως παρατηρούμε από το διάγραμμα ο βέλτιστος αριθμός γειτόνων είναι ίσος με 5 Κάποιες φορές ο βέλτιστος αριθμός είναι **διαφορετικός**. Αυτό συμβαίνει διότι τα **folds** δημιουργούνται με τυχαίες εγγραφές κάθε φορά.

```
def cross_validation_split(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    for _ in range(n_folds):
        fold = list()
        while len(fold) < fold_size:
            index = randrange(len(dataset_copy))
            fold.append(dataset_copy.pop(index))
        dataset_split.append(fold)
    return dataset_split
```

Όπως βλέπουμε στην μεταβλητή **index** μέσα στο **while loop** χρησιμοποιούμε την συνάρτηση **randrange** με την οποία παίρνουμε κάθε φορά μια τυχαία εγγραφή από το **dataset_copy** για να δημιουργήσουμε κάθε **fold**.

Άμα κάνουμε κάποιες μικρές τροποποιήσεις στην συνάρτηση **cross_validation_split**, ώστε να μην παίρνουμε τυχαίες εγγραφές κάθε φορά από το **dataset_copy** τότε η γραφική μας παράσταση θα βγαίνει κάθε φορά η ίδια.

```
def cross_validation_split_without_randrage(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    index = 0
    for _ in range(n_folds):
        fold = list()
        while len(fold) < fold_size:
            fold.append(dataset_copy[index])
            index += 1
        dataset_split.append(fold)
    return dataset_split
```




**fold_num_and_accuracy.**

Με την κλήση της συνάρτησης **fold_num_and_accuracy**, βρίσκουμε το **mean accuracy** που πετυχαίνει το μοντέλο μας όταν έχει σαν είσοδο καινούργια δεδομένα, για αριθμό **folds** από 2 έως **fold_num** και για 5 γείτονες. Επίσης σχεδιάζεται το **bar diagram** που δείχνει το **mean accuracy** κάθε **fold**. Αυτό που κάνει δηλαδή στην πραγματικότητα αυτή η συνάρτηση είναι να κάνει **k-fold cross validation**. Το **k-fold cross validation** έχει να κάνει με τον προσδιορισμό του βέλτιστου **K** και με την αξιολόγηση του μοντέλου μας ως προς τα αποτελέσματα που μας επιστρέφει όταν κάνει προβλέψεις σε άγνωστα δεδομένα.

Ο αριθμός των **folds** συνήθως προσδιορίζεται από τον αριθμό των δεδομένων που έχουμε στην διάθεση μας. Για παράδειγμα άμα έχουμε 10 εγγραφές στο **dataset** μας το **10-fold cross validation** δεν θα είχε νόημα. Χρησιμοποιούμε την **fold_num_and_accuracy** για να δούμε πως αξιολογείται το μοντέλο μας ανάλογα με τον αριθμό των **folds**. Για να γίνει η επιλογή του αριθμού των **folds** (ώστε να βρούμε στην συνέχεια το βέλτιστο **K** και να κάνουμε καλύτερη αξιολόγηση του μοντέλου μας), πρέπει τόσο το **training set** όσο και το **testing set** κάθε φορά να έχουν το ίδιο **distribution** (πόσο συχνά εμφανίζεται κάθε τιμή) και κάθε **set** να έχει επαρκή **variation** (πόσο διαφορά (**spread out**) έχουν τα **data** μεταξύ τους).

Με βάση τα παραπάνω, η επιλογή του βέλτιστου αριθμού **folds** δεν έχει κάποια συγκεκριμένη “φόρμουλα” διότι είναι σχετικά δύσκολο το να υπολογίσεις κατά πόσο κάθε **fold** αντιπροσωπεύει ολόκληρο το **dataset**. Συνήθως τα **5 folds** αρκούν για μέγεθος **datasets** όπως στην περίπτωση που μελετάμε εμείς (569), για να βρούμε το βέλτιστο **K** και να αξιολογήσουμε το μοντέλο μας.

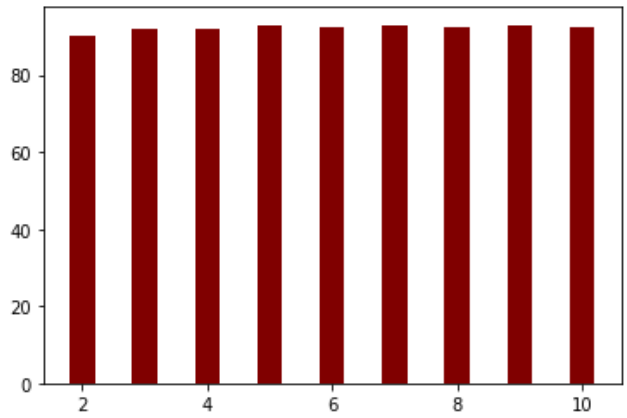
```
def fold_num_and_accuracy(fold_num, dataset):
    mean_acc = []
    folds = []
    for num in range(2, fold_num):
        scores = evaluate_algorithm(dataset, k_nearest_neighbors, num, 5)
        #print('Scores: %s' % scores)
        print('Mean Accuracy: %.3f%%'%(sum(scores)/float(len(scores))), ' for {}'.format(num),' folds.\n')
        mean_acc.append((sum(scores)/float(len(scores))))
        folds.append(num)

    plt.bar(folds, mean_acc, color ='maroon',  width = 0.4)
```

**Επεξήγηση κώδικα.**

Βρίσκουμε το **mean accuracy** ανάλογα με τον αριθμό των **folds** και για **5 γείτονες**. Δημιουργούμε δύο κενές λίστες με όνομα **mean_acc** και **folds**. Στο **for loop** και για αριθμό

**folds** ίσο με **2** έως **fold_num** κάνουμε **evaluate** την απόδοση του αλγορίθμου μας με την χρήση **5** γειτόνων και αριθμό **folds** ίσο με **num** (2 έως fold_num). Έπειτα τυπώνουμε το ποσοστό επί της 100 του **mean accuracy** και το κάνουμε **append** στην λίστα **mean_acc** όπως και το **num** στην λίστα **folds**. Τέλος σχεδιάζεται το **plot bar** που δείχνει το **mean accuracy** με βάση τον αριθμό των **folds**.

```
In [41]: find_best_fold_num(11)
Mean Accuracy: 90.141%  for 2  folds.

Mean Accuracy: 91.887%  for 3  folds.

Mean Accuracy: 91.901%  for 4  folds.

Mean Accuracy: 92.743%  for 5  folds.

Mean Accuracy: 92.553%  for 6  folds.

Mean Accuracy: 92.945%  for 7  folds.

Mean Accuracy: 92.254%  for 8  folds.

Mean Accuracy: 92.769%  for 9  folds.

Mean Accuracy: 92.321%  for 10  folds.
```




**evaluate_algorithm.**

```
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
    folds = cross_validation_split_without_randrage(dataset, n_folds)
    scores = list()
    for fold in folds:
        train_set = list(folds)
        #print(train_set[0],"\n")
        train_set.remove(fold)
        train_set = sum(train_set, []) # Κάνει τα εναπομείναντα folds
        #print(train_set[0],"\n")
        test_set = list()
        for row in fold:
            row_copy = list(row)
            test_set.append(row_copy)
            row_copy[0] = None
        predicted = algorithm(train_set, test_set, *args)
        actual = [row[0] for row in fold]
        accuracy = accuracy_metric(actual, predicted)
        scores.append(accuracy)
    return scores
```

**Επεξήγηση κώδικα.**

Στην **evaluate_algorithm** παίρνουμε σαν παραμέτρους το **dataset** που περιέχει όλες τις λίστες με τα χαρακτηριστικά των όγκων των ασθενών, το **algorithm** που θα χρησιμοποιηθεί για τις προβλέψεις (**knn**), το **n_folds** που είναι ο αριθμός των **folds** και το ***args** που είναι ο αριθμός των γειτόνων. Καλούμε την συνάρτηση **cross_validation_split_without_randrange**

και παίρνουμε πίσω μια λίστα με 5 λίστες (όσα και τα **folds**) που κάθε τέτοια λίστα περιέχει τις λίστες που κάθε μια περιέχει τα χαρακτηριστικά του όγκου του ασθενή. Αρχικοποιούμε την κενή λίστα **scores**. Μέσα στο **for loop** για αριθμό ίσον με αυτόν των **folds**, κάθε φορά, δημιουργούμε την λίστα **train_set** στην οποία εκχωρούμε την λίστα **folds**. Αφαιρούμε από αυτή την λίστα ένα από τα **folds**(αυτό που υποδεικνύεται από το **fold**) και κάνουμε το **train_set** μια ενιαία λίστα που περιέχει τις εγγραφές των ασθενών (Επίσης λίστες με χαρακτηριστικά των όγκων). Δημιουργούμε την κενή λίστα **test_set**. Στο εσωτερικό **for loop** για κάθε γραμμή από το **fold** που έχει αφαιρεθεί προηγουμένως κάνουμε την **target variable** ίση με **None** (row_copy[0] = None). Καλούμε την συνάρτηση **k_nearest_neighbors** (algorithm) και παίρνουμε μια λίστα με 0 και 1 (προβλέψεις για καλοήθεις, κακοήθεις). Στην συνέχεια καλούμε την συνάρτηση **accuracy_metric** από την οποία παίρνουμε το **accuracy%** για το **fold** που επιλέγεται σε κάθε **loop** και το κάνουμε **append** στην λίστα **scores**. Τέλος επιστρέφουμε την **scores**.

**cross_validation_split.**

```
def cross_validation_split(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    for _ in range(n_folds):
        fold = list()
        while len(fold) < fold_size:
            index = randrange(len(dataset_copy))
            fold.append(dataset_copy.pop(index))
        dataset_split.append(fold)
    return dataset_split
```

**Επεξήγηση κώδικα.**

Η συνάρτηση **cross_validation_split** παίρνει σαν είσοδο το **dataset** και τον αριθμό των **folds** (n_folds). Δημιουργεί την κενή λίστα **dataset_split** όπου θα μπουν τα **folds** που θα έχουμε φτιάξει από το **dataset**. Αντιγράφει το **dataset** στην **dataset_copy**. Βρίσκει το μέγεθος κάθε **fold** (fold_size). Στην περίπτωση μας κάθε **fold** θα αποτελείται από 113 εγγραφές αφού χρησιμοποιούμε **n_folds = 5**. Στο **for loop** το οποίο τρέχει για 5 φορές βάζουμε μέσα στην λίστα **fold** 113 τυχαίες εγγραφές, κάθε **fold** εκχωρείται στο **dataset_split**. Τέλος τι **dataset_split** επιστρέφεται.

**cross_validation_split_without_randrage.**

```
def cross_validation_split_without_randrage(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    index = 0
    for _ in range(n_folds):
        fold = list()
        while len(fold) < fold_size:
            fold.append(dataset_copy[index])
            index += 1
        dataset_split.append(fold)
    return dataset_split
```

**Επεξήγηση κώδικα.**

Η συνάρτηση **cross_validation_split_without_randrage** είναι ίδια με την παραπάνω με την μόνη διαφορά πως δεν βάζουμε τυχαίες εγγραφές από το **dataset_copy** κάθε φορά μέσα στο **fold** με την χρήση του **randrange**, αλλά με την σειρά, αρχίζοντας από την εγγραφή 0. Το **k-fold cross validation** χρησιμεύει στο να προσδιορίσουμε το πόσο καλά το μοντέλο μας αποδίδει σε καινούργια δεδομένα. Συνήθως επιλέγεται ένας μονός αριθμός για πλήθος δεδομένων με μερικές εκατοντάδες εγγραφές.

**accuracy_metric.**

```
def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0
```

**Επεξήγηση κώδικα.**

Συγκρίνει την πραγματική τιμή του **target variable** με αυτή που προβλέφθηκε και αυξάνει τον μετρητή **correct** κατά 1 αν είναι ίσες. Τέλος επιστρέφει το ποσοστό επί της 100 των σωστών προβλέψεων.

## Συναρτήσεις για να κάνουμε προβλέψεις.

**k_nearest_neighbors.**

```
def k_nearest_neighbors(train, test, num_neighbors):
    predictions = list()
    for row in test:
        output = predict_classification(train, row, num_neighbors)
        predictions.append(output)

    if test[0][0] == -1:
        for i in range(len(test)):
            print('Predicted: ',predictions[i], '\n' )
    else:
        wrong_pred = 0
        for i in range(len(test)):
            print('Expected %d, Got %d.' % (test[i][0], predictions[i]))
            if test[i][0] != predictions[i]:
                wrong_pred += 1
        print('At', len(test), ' cases we get ', wrong_pred, 'wrong predictions\n')
```

**Επεξήγηση κώδικα.**

Στην συνάρτηση **k_nearest_neighbors** παίρνουμε σαν παραμέτρους το **train** που είναι η λίστα με τις εγγραφές των ασθενών που ξέρουμε το **target variable** τους (0, 1), την λίστα **test** που περιέχει τις εγγραφές των ασθενών στις οποίες δεν γνωρίζουμε σε ποιο **class** ανήκουν (0, 1) και το **num_neighbors** που είναι ο αριθμός των γειτόνων. Αρχικά δημιουργούμε μια κενή λίστα με όνομα **predictions**. Στην συνέχεια στο **for loop** παίρνουμε σε κάθε **loop** μια γραμμή από την λίστα **test**. Καλούμε την συνάρτηση **predict_classification** και παίρνουμε στο **output** 0 ή 1. Το κάνουμε αυτό για όλες τις γραμμές της λίστας **test** και κάθε φορά κάνουμε **append** το **output** στην λίστα **predictions**. Στο **if** ελέγχουμε αν έχουμε δώσει καινούργια δεδομένα για νέους ασθενείς. Σε αυτή την περίπτωση το **target variable** θα είναι ίσο με **-1** και θα μπούμε μέσα στο **if** και θα τυπώσουμε τις προβλέψεις. Αντίθετα

θα μπούμε στο **else** θα τυπώσουμε το σωστό **target variable**, τις **προβλέψεις** και τον αριθμό των **λάθος προβλέψεων**.

**predict_classification.**

```
def predict_classification(train, test_row, num_neighbors):
    neighbors = get_neighbors(train, test_row, num_neighbors)
    output_values = [row[0] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction
```

**Επεξήγηση κώδικα.**

Η συνάρτηση **predict_classification** καλείται από την **k_nearest_neighbors** για κάθε γραμμή από την λίστα **test**. Παίρνει σαν παραμέτρους το **train**, που είναι μια λίστα που περιέχει λίστες με τις εγγραφές των ασθενών για τα χαρακτηριστικά των όγκων που έχει παρουσιάσει ο καθένας, το **test_row**, που είναι μια γραμμή από την λίστα **test** (η λίστα test θυμίζουμε περιέχει λίστες-εγγραφές) και το **num_neighbors** που είναι ο αριθμός των γειτόνων. Έπειτα καλούμε την συνάρτηση **get_neighbors** από την οποία επιστρέφεται μια λίστα που περιέχει τις λίστες-εγγραφές των κοντινότερων γειτόνων, Στην λίστα **output_values** εκχωρούμε το **target_variable** (καλοήθεις(0), κακοήθεις(1)) των γειτόνων. Τέλος στο **prediction** εκχωρούμε την τιμή που εμφανίζεται περισσότερες φορές στην λίστα **output_values** και το επιστρέφουμε.

**get_neighbors.**

```
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors
```

**Επεξήγηση κώδικα.**

Στην συνάρτηση **get_neighbors** παίρνουμε σαν παραμέτρους την λίστα **train** που περιέχει τις εγγραφές που γνωρίζουμε την κλάση τους, το **test row**, που είναι μια γραμμή-εγγραφή από την λίστα **test** στην οποία δεν γνωρίζουμε το **target variable** και θέλουμε να το προβλέψουμε και το **num_neighbors** που είναι ο αριθμός των γειτόνων. Αρχικά δημιουργούμε μια κενή λίστα με όνομα **distances**. Μέσα στο **for loop** παίρνουμε κάθε φορά μια σειρά **train_row** από την λίστα **train**. Στην μεταβλητή **dist** εκχωρούμε την απόσταση των δύο λιστών-εγγραφών που έχει υπολογιστεί από την συνάρτηση **euclidean_distance**. Έπειτα εκχωρούμε κάθε φορά την απόσταση που έχουμε βρει μαζί με το **train_row** από το οποίο το **test_row** απέχει την συγκεκριμένη απόσταση, σε μορφή **tuple**

(train_row, απόσταση), στην λίστα **distances**. Ταξινομούμε σε αύξουσα σειρά την λίστα με τα **tuples** ως προς την απόσταση. Δημιουργούμε την κενή λίστα **neighbors**. Μέσα στο **for loop** και για αριθμό **loops** ίσο με αυτό του **num_neighbors** εκχωρούμε στην λίστα **neighbors** τις λίστες που παίρνουμε από τα **tuples** με τα χαρακτηριστικά των **neighbors** που απέχουν την μικρότερη απόσταση. Τέλος επιστρέφουμε την λίστα **neighbors**.

**add_new_patient_data_and_predict.**

```
def add_new_patient_data_and_predict(df_dataset, dataset, num_of_patients):
    columns = df_dataset.columns
    new_patient = []
    new_patients_list = []

    # Άμα θέλω να δίνω τις τιμές χειροκίνητα αρκεί να ξεσχολιάσω τις γραμμές 322 και 323 και να
    for i in range(num_of_patients):
        #print('Add diagnosis == -1\n')
        new_patient.append(-1)
        for num in range(1, 31):
            #print('Add value between ', min(df_dataset[columns[num]]), ' and ', max(df_dataset[
            #val = float(input('Enter {} '.format(columns[num])))
            val = random.uniform(min(df_dataset[columns[num]]),  max(df_dataset[columns[num]]))
            new_patient.append(val)

        new_patients_list.append(new_patient)
        new_patient = []

    k_nearest_neighbors(dataset, new_patients_list, 5)
```

**Επεξήγηση κώδικα.**

Συνάρτηση η οποία παίρνει σαν παραμέτρους το **df_dataset** που είναι ένα **pandas dataframe**, το **dataset** σε μορφή λιστών για κάθε εγγραφή και το **num_of_patients** που είναι ο αριθμός των νέων ασθενών για τους οποίους θα προσθέσουμε καινούργια δεδομένα. Αρχικά παίρνουμε στην λίστα **columns** τα ονόματα των στηλών από το **df_dataset**. Δημιουργούμε δύο κενές λίστες, την **new_patient** και την **new_patients_list**. Μέσα στο **for loop** και για αριθμό επαναλήψεων ίσο με **num_of_patients** εκχωρούμε στην πρώτη θέση **new_patient[0]** το **-1**. Στο εμφωλευμένο **for loop** και για αριθμό επαναλήψεων ίσο με 30 βάζουμε τυχαίες τιμές στην λίστα **new_patient**, στο εύρος τιμών που κυμαίνεται κάθε χαρακτηριστικό του **dataset**. Άμα δεν θέλουμε να βάζουμε τυχαίες τιμές και να βάζουμε τα καινούργια δεδομένα χειροκίνητα ξεσχολιάζουμε τις δύο εντολές κάτω από το εμφωλευμένο **for loop** και σχολιάζουμε την γραμμή **val = random.uniform(min(df_dataset[columns[num]]),  max(df_dataset[columns[num]]))**. Τέλος καλείται η συνάρτηση **k_nearest_neighbors** και γίνονται οι προβλέψεις για τους καινούργιους ασθενείς.

**euclidean_distance.**

```
def euclidean_distance(row1, row2):
    distance = 0.0
    for i in range(len(row1)-1):
        distance += (row1[i + 1] - row2[i + 1])**2
    return sqrt(distance)
```

**Επεξήγηση κώδικα.**

Στην συνάρτηση **euclidean_distance** σαν παραμέτρους παίρνουμε το **row1** (test_row) και το **row2** (train_row). Μέσα στο **for loop** υπολογίζουμε το τετράγωνο της διαφοράς των

τιμών για κάθε εγγραφή από την δεύτερη τιμή και μετά, γιατί η πρώτη τιμή στο **row2** είναι το **target variable** και στο **row1** είναι **None**. Τέλος επιστρέφουμε την ρίζα της απόστασης.

## Συμπέρασμα.

Λαμβάνοντας υπόψιν τα προηγούμενα πειράματα και τις δοκιμές που γίνανε με τις διάφορες τιμές των παραμέτρων των συναρτήσεων (find_best_K, fold_num_and_accuracy) καταλήγουμε πως ο βέλτιστος αριθμός γειτόνων για να κάνουμε προβλέψεις είναι **K = 5** και ο βέλτιστος αριθμός **folds** για να αξιολογήσουμε το μοντέλο μας και να βρούμε τον αριθμό **K** με τον οποίο θα μεγιστοποιούμε το **accuracy** του μοντέλου, με βάση το πλήθος των δεδομένων μας είναι **folds = 5**. Το επόμενο βήμα είναι να κάνουμε προβλέψεις με τις συναρτήσεις που περιγράφηκαν παραπάνω.

## Προβλέψεις με 5 γείτονες και 5 folds.

```
k_nearest_neighbors(dataset, dataset[0:10], 5)
k_nearest_neighbors(dataset, dataset[0:20], 5)
k_nearest_neighbors(dataset, dataset[0:100], 5)
k_nearest_neighbors(dataset, dataset[0:200], 5)
```

## Αποτελέσματα για 10 cases.

```
In [204]: k_nearest_neighbors(dataset, dataset[0:10], 5)
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 0.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 1.
At 10  cases we get  1 wrong predictions
```

## Αποτελέσματα για 20 cases.

```
In [205]: k_nearest_neighbors(dataset, dataset[0:20], 5)
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 0.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 0.
Expected 1, Got 0.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 0, Got 0.
At 20  cases we get  3 wrong predictions
```

## Αποτελέσματα για 100 cases.

```
Expected 1, Got 1.
Expected 0, Got 0.
Expected 0, Got 0.
Expected 0, Got 0.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 0, Got 0.
Expected 1, Got 1.
Expected 1, Got 0.
Expected 1, Got 1.
Expected 0, Got 0.
Expected 0, Got 0.
Expected 0, Got 0.
Expected 1, Got 0.
Expected 0, Got 1.
Expected 0, Got 0.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 0, Got 0.
Expected 0, Got 0.
Expected 0, Got 0.
Expected 1, Got 0.
At 100  cases we get  11 wrong predictions
```

## Αποτελέσματα για 200 cases.

```
Expected 0, Got 0.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 0, Got 0.
Expected 1, Got 1.
Expected 0, Got 0.
Expected 1, Got 1.
Expected 0, Got 0.
Expected 0, Got 0.
Expected 0, Got 0.
Expected 1, Got 0.
Expected 0, Got 0.
Expected 0, Got 0.
Expected 1, Got 1.
Expected 1, Got 0.
Expected 0, Got 0.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 1.
Expected 1, Got 1.
At 200  cases we get  16 wrong predictions
```

## Προβλέψεις με καινούργιες τιμές που δεν υπάρχουν στο dataset.

```
add_new_patient_data_and_predict(df_dataset, dataset, 10)
add_new_patient_data_and_predict(df_dataset, dataset, 20)
add_new_patient_data_and_predict(df_dataset, dataset, 30)
```

## Αποτελέσματα για 10 τιμές.

```
In [208]: add_new_patient_data_and_predict(df_dataset, dataset, 10)
Predicted:  1

Predicted:  1

Predicted:  1

Predicted:  1

Predicted:  1

Predicted:  1

Predicted:  1

Predicted:  1

Predicted:  1

Predicted:  1
```

### Αποτελέσματα για 20 τιμές.

```
Predicted:  1
Predicted:  1
Predicted:  1
Predicted:  1
Predicted:  1
Predicted:  1
Predicted:  1
Predicted:  1
Predicted:  0
Predicted:  1
Predicted:  1
Predicted:  0
```

### Αποτελέσματα για 30 τιμές.

```
Predicted:  1
Predicted:  1
Predicted:  0
Predicted:  1
Predicted:  1
Predicted:  1
Predicted:  1
Predicted:  1
Predicted:  1
Predicted:  1
Predicted:  1
Predicted:  0
Predicted:  1
Predicted:  1
```

## 6. Διερμηνεία αποτελεσμάτων

Σύμφωνα με τα προηγούμενα πειράματα για να βρούμε το βέλτιστο **K**, τον κατάλληλο αριθμό **folds** για να αξιολογήσουμε καλύτερα το μοντέλο μας αλλά και τις προβλέψεις που κάναμε σε δεδομένα του **dataset** και δικά μας, καταλήξαμε πως στο συγκεκριμένο σενάριο που μελετάμε, μπορούμε να κάνουμε προβλέψεις με υψηλό ποσοστό επιτυχίας (σχεδόν 93%) τόσο για τα δεδομένα του **dataset** όσο και για αυτά που δημιουργούμε εμείς. Άρα στην περίπτωση μας ένας γιατρός που εργάζεται στην θεραπευτική μονάδα του σεναρίου, έχοντας στην διάθεση του τα χαρακτηριστικά του όγκου ενός ασθενή θα μπορεί να τον κατατάσσει σε καλοήθη ή κακοήθη με μεγάλο ποσοστό βεβαιότητας

Με βάση τα παραπάνω χρησιμοποιώντας 5 γείτονες για να κάνουμε προβλέψεις θα έχουμε το μεγαλύτερο ποσοστό επιτυχίας (σωστών προβλέψεων) για το αν ο ασθενής έχει εμφανίσει **κακοήθη** ή **καλοήθη** όγκο. Έτσι μπορούμε να κατατάξουμε με περισσότερη

σιγουριά τους ασθενείς με βάση τα χαρακτηριστικά του όγκου που εξετάζουμε και να έχουμε μια εμπεριστατωμένη γνώμη για την κατάσταση του ασθενή σε πιο σύντομο χρονικό διάστημα. Παρακάτω βλέπουμε κάποια διαγράμματα που κατατάσσουν τον όγκο σε καλοήθη (0) ή κακοήθη (1) με βάση τα χαρακτηριστικά τους.

# 7. Εφαρμογή μοντέλου Kotter για τη Διαχείριση Αλλαγών.

## Ανάπτυξη της αίσθησης της αναγκαιότητας.

Η αναγκαιότητα του παραπάνω συστήματος εξόρυξης γνώσης είναι προφανής, αφού μπορούν να **κατηγοριοποιηθούν εύκολα** οι ασθενείς με βάση τον όγκο που έχουν παρουσιάσει και να προβλεφτεί μια πρώτη **διάγνωση**, με μεγάλο **ποσοστό ακρίβειας** και με **περισσότερη ευκολία** αφού απλά περνάμε τα δεδομένα μας στο σύστημα και αυτό μας επιστρέφει μια **πρόβλεψη-διάγνωση**, που μπορεί να αποβεί σωτήρια για τον ασθενή και για τις ενέργειες που θα ακολουθηθούν από αυτή την φάση και έπειτα. Επίσης θα **λύσει τα χέρια** των εργαζομένων της μονάδας θεραπείας αφού η διαχείριση των δεδομένων θα γίνεται με περισσότερο **ευέλικτο τρόπο**.

## Δημιουργία ηγετικού συνασπισμού – ομάδας καθοδήγησης.

Για την αφομοίωση της νέας γνώσης που επιφέρεται από το σύστημα πρόβλεψης-διάγνωσης ασθενών είναι αναγκαίο να **συσταθεί μια ομάδα εκπαίδευσης** από **προγραμματιστές** και **γιατρούς**. Οι προγραμματιστές που είναι υπεύθυνοι για την ανάπτυξη του συστήματος πρόβλεψης είναι αυτοί που θα **εκπαιδεύσουν το προσωπικό** της μονάδας θεραπείας πάνω στην χρήση του νέου συστήματος διάγνωσης και με την σειρά τους η **ομάδα γιατρών** θα **καθοδηγεί τους εκπαιδευόμενους** γιατρούς-υπαλλήλους ως προς τι δεδομένα δέχεται το νέο σύστημα και στο τι γνώση εξάγεται από αυτό. Αυτή η ομάδα πρέπει να αποτελείται από **διευθυντικά μέλη** και από την πλευρά των προγραμματιστών και από την πλευρά των γιατρών. Μεταξύ τους πρέπει να **υπάρχει καλή επικοινωνία** και **κατανόηση των στόχων** (εκπαίδευση σε πρώτο στάδιο των υπολοίπων) και να **εμπνέουν εμπιστοσύνη** στο προσωπικό.

Η δικιά μας ομάδα θα αποτελείται από τον **διευθυντή** του **ιατρικού προσωπικού**, τους αμέσως κατώτερους στην ιεραρχία **ιατρούς**, τον **διευθυντή** της ομάδας προγραμματιστών και έναν ακόμα κατώτερο σε ιεραρχία προγραμματιστή.

## Ανάπτυξη οράματος στρατηγικής.

Το όραμα μας είναι η **ακριβέστερη** και **συντομότερη διάγνωση** των ασθενών, η **βελτίωση της λειτουργίας** της μονάδας διάγνωσης και θεραπείας που θα επέλθει μέσα από αυτή την πιο καλά δομημένη και σύγχρονη εξόρυξη της γνώσης που θέλουμε και τέλος η **αποδοτικότερη ανάλυση** και **διαχείριση των δεδομένων** που έχουμε για την εξαγωγή μιας διάγνωσης που μπορεί να αποβεί σωτήρια για πολλούς ανθρώπους.

Με αυτόν τον νέο τρόπο εξόρυξης γνώσης τα δεδομένα που έχουμε στον οργανισμό μας θα είναι πιο **εύκολα διαχειρίσιμα** και θα αποφεύγονται χρονοβόρες μέθοδοι άσκοπης καταγραφής τους, επεξεργασίας τους και αποθήκευσης τους, αφού με αυτό το σύστημα

αφού προσκομίσουμε τα δεδομένα μας μπορούμε **αυτόματα** να τα **επεξεργαστούμε**, να τα **εξάγουμε** προς αποθήκευση (**csv**) και να τα περάσουμε στον προβλεπτικό αλγόριθμο για την εξόρυξη γνώσης.

Θα επιτευχθεί μια περισσότερο οργανωμένη και ομαλή λειτουργία του οργανισμού αφού γίνεται πιο εύκολη η δόμηση και διαχείριση των δεδομένων. Το σύστημα εξόρυξης γνώσης είναι κατανοητό τόσο από τους ιατρούς όσο και από τους υπαλλήλους και απαιτεί ελάχιστη εκπαίδευση για να χρησιμοποιηθεί.

## Επικοινωνία του οράματος της αλλαγής.

- Ο νέος τρόπος διαχείρισης δεδομένων και εξαγωγής γνώσης έγινε γνωστός μετά από συνέλευση μεταξύ των διοικητικών υπαλλήλων κάθε τμήματος και στην συνέχεια με την αποστολή ηλεκτρονικών μηνυμάτων.
- Έγινε παρουσίαση του νέου συστήματος από την ομάδα των προγραμματιστών και τον διευθυντή του ιατρικού προσωπικού.
- Συζητήθηκαν οι λόγοι που απαιτείται η μετάβαση σε ένα τέτοιο σύστημα
- Παρουσιάστηκαν τα επιχειρήματα για την αναγκαιότητα της βελτίωσης της μέχρι τώρα λειτουργίας του οργανισμού.
- Λήφθηκαν υπόψιν οι παρατηρήσεις και οι γνώμες των υπαλλήλων.
- Αποφασίστηκε η εγκαθίδρυση του νέου συστήματος.

## Ενδυνάμωση και ευρεία συμμετοχή.

Για την ενδυνάμωση των υπαλλήλων και την ώθηση τους στην χρήση του νέου συστήματος αλλά και στην κατανόηση, του γιατί με αυτόν τον νέο τρόπο εξαγωγής διαγνώσεων οι ίδιοι επωφελούνται κάνοντας πιο εύκολη την εργασία τους αρχικά θα υπάρξει κάποιο **διάστημα εκπαίδευσης** πάνω στο νέο σύστημα. Θα π**αρουσιαστούν** όλα τα **μέρη του συστήματος** και της γνώσης που εξάγουμε από αυτό. Θα συσταθεί **ομάδα εξυπηρέτησης** και **υποστήριξης υπαλλήλων** για να αντιμετωπίζει τυχόν προβλήματα και απορίες που μπορεί να προκύψουν από τους χρήστες τους συστήματος.

## Δημιουργία βραχυπρόθεσμων επιτυχιών.

Μετά από την περίοδο της εκπαίδευσης και από μια μικρή περίοδο αφομοίωσης του καινούργιου συστήματος εξαγωγής γνώσης οι υπάλληλοι είδαν πως γίνετε **πιο εύκολη** η εργασία τους και σε **μικρότερο χρονικό διάστημα**, υπάρχει **καλύτερη οργάνωση** γενικότερα στην μονάδα θεραπείας και οι προβλέψεις-διαγνώσεις που γίνονται έχουν **μεγάλο ποσοστό επιτυχίας** και βοηθάνε στην **καθοδήγηση της μελέτης** και **θεραπείας** των όγκων των ασθενών.

## Παγιοποίηση αποτελεσμάτων και προώθηση επιπρόσθετων αλλαγών.

- Η περιττή εργασία εξαλείφθηκε (επιπλέον καταγραφή και επεξεργασία των δεδομένων).
- Το νέο σύστημα αφομοιώθηκε από όλους τους υπαλλήλους.
- Τα αποτελέσματα που πήραμε ήταν σε μεγάλο ποσοστό σωστά.
- Οι παλιές μέθοδοι καταγραφής δεδομένων, εξόρυξης γνώσης καταργήθηκαν.

## Ενστερνισμός νέας κουλτούρας.

Υπήρχαν υπάλληλοι που ήταν αντίθετοι με τον νέο τρόπο εξόρυξης γνώσης. Τα μέτρα που λήφθηκαν για να ξεπεράσουμε τις διαφωνίες ήταν:

- Δεν γίνεται πλέον διαθέσιμος άλλος τρόπος καταγραφής, επεξεργασίας και εξαγωγής γνώσης από δεδομένα ασθενών (λογισμικό).
- Παρουσιάστηκαν τα οφέλη του νέου συστήματος.
- Η ομάδα εξυπηρέτησης και καθοδήγησης υπαλλήλων βοήθησε όλους τους εργαζόμενους να αφομοιώσουν το σύστημα.
- Τα προβλήματα του συστήματος και οι προτάσεις για βελτίωση του συλλέγονται και θα εφαρμοστούν σε επόμενη έκδοση του συστήματος.