CANCER DATA

Benign and Malignant Cancer Data

Progetto per “Ingegneria della Conoscenza” AA 2024/2025

LINK REPOSITORY GITHUB:
https://github.com/AlessadroLombardi/IconProject2425

GRUPPO DI LAVORO:

ALESSANDRO LOMBARDI ,	735970 ,	a.lombardi59@studenti.uniba.it
FILIPPO NARDULLI ,	736153 ,	f.nardulli5@studenti.uniba.it

INDICE

#0 INTRODUZIONE

Questo progetto nasce con l'idea di predire la diagnosi (maligna o benigna) di un tumore sfruttando un DataSet disponibile online.
A tal proposito:

- È stata modellata un'**Ontologia** di riferimento che offre una rappresentazione formale e concettualizzata della realtà presa in esame, affinché possa venire interrogata tramite **Query**.
- Sono state utilizzate tecniche di **Apprendimento Supervisionato** e **Non Supervisionato**.

PROBLEMA IN QUESTIONE:
Questo DataSet presenta i dati di pazienti con un tumore. Questo può essere di tipo benigno o maligno.
Per ogni paziente sono state salvate, oltre ad un ID univoco e la diagnosi del cancro, le caratteristiche visive di esso, nonché i valori medi di quest'ultimi.
Un tale DataSet può essere utilizzato per addestrare o utilizzare modelli e algoritmi per effettuare delle diagnosi.

MATERIALE UTILIZZATO:

LINGUAGGIO DI PROGRAMMAZIONE

- Python: https://www.python.org/downloads/

PROVIDER DEL DATASET

- Kaggle: https://www.kaggle.com/datasets/

APP PER REALIZZAZIONE DELL'ONTOLOGIA

- Protégé: https://protege.stanford.edu/

LIBRERIE PYTHON

- Scikit-learn: https://scikit-learn.org/
- Pandas: https://pandas.pydata.org/
- Seaborn: https://seaborn.pydata.org/
- OwlReady2: https://owlready2.readthedocs.io/en/v0.37/index.html

#1 DATASET

Proponiamo un Set di Dati relativo allo screening delle diagnosi di tumore, reperito dal seguente indirizzo https://www.kaggle.com/datasets/erdemtaha/cancer-data?resource=download.

#1.1 DESCRIZIONE DATASET (DOMINIO)

FEATURE: 32 **CAMPIONE**: 570 istanze

FEATURE	FEATURE ROLE	DOMAIN TYPE	DESCRIPTION
id	Feature	Integer (0-800000)	Identificativo unico per riconoscere i pazienti.
Raggio Medio	Feature	Continuous (0-3000)	Media del raggio delle cellule tumorali.
Trama Media	Feature	Continuous (0-3000)	Media della variazione dell'intensità della texture nelle cellule.
Perimetro Medio	Feature	Continuous (0-3000)	Media della misura del perimetro delle cellule.
Area Media	Feature	Continuous (0-3000)	Media dell'area delle cellule tumorali.
Levigatezza Media	Feature	Continuous (0-3000)	Media della variazione locale dei contorni delle cellule.
Compattezza Media	Feature	Continuous (0-3000)	Media del rapporto tra perimetro² e area delle cellule.
Concavità Media	Feature	Continuous (0-3000)	Media del grado di concavità nei contorni delle cellule.
Punti Concavi Medi	Feature	Continuous (0-3000)	Media del numero di punti concavi nei contorni delle cellule.
Simmetria Media	Feature	Continuous (0-3000)	Media della simmetria della forma delle cellule.
Dimensione Frattale Media	Feature	Continuous (0-3000)	Media della complessità dei contorni delle cellule.
Raggio Errore Standard	Feature	Continuous (0-3000)	Variazione del raggio delle cellule (errore standard).
Trama Errore Standard	Feature	Continuous (0-3000)	Variazione della texture delle cellule (errore standard).
Perimetro Errore	Feature	Continuous (0-3000)	Variazione del perimetro delle cellule (errore standard).

Standard			
Area Errore Standard	Feature	Continuous (0-3000)	Variazione dell’area delle cellule (errore standard).
Levigatezza Errore Standard	Feature	Continuous (0-3000)	Variazione della levigatezza delle cellule (errore standard).
Compattezza Errore Standard	Feature	Continuous (0-3000)	Variazione della compattezza delle cellule (errore standard).
Concavità Errore Standard	Feature	Continuous (0-3000)	Variazione della concavità delle cellule (errore standard).
Punti Concavi Errore Standard	Feature	Continuous (0-3000)	Variazione dei punti concavi delle cellule (errore standard).
Simmetria Errore Standard	Feature	Continuous (0-3000)	Variazione della simmetria delle cellule (errore standard).
Dimensione Frattale Errore Standard	Feature	Continuous (0-3000)	Variazione della dimensione frattale delle cellule (errore standard).
Raggio Peggiore	Feature	Continuous (0-3000)	Peggior valore osservato del raggio delle cellule.
Trama Peggiore	Feature	Continuous (0-3000)	Peggior valore osservato della trama delle cellule.
Perimetro Peggiore	Feature	Continuous (0-3000)	Peggior valore osservato del perimetro delle cellule.
Area Peggiore	Feature	Continuous (0-3000)	Peggior valore osservato dell’area delle cellule.
Levigatezza Peggiore	Feature	Continuous (0-3000)	Peggior valore osservato della levigatezza delle cellule.
Compattezza Peggiore	Feature	Continuous (0-3000)	Peggior valore osservato della compattezza delle cellule.
Concavità Peggiore	Feature	Continuous (0-3000)	Peggior valore osservato della concavità delle cellule.
Punti Concavi Peggiori	Feature	Continuous (0-3000)	Peggior valore osservato dei punti concavi delle cellule.
Simmetria Peggiore	Feature	Continuous (0-3000)	Peggior valore osservato della simmetria delle cellule.

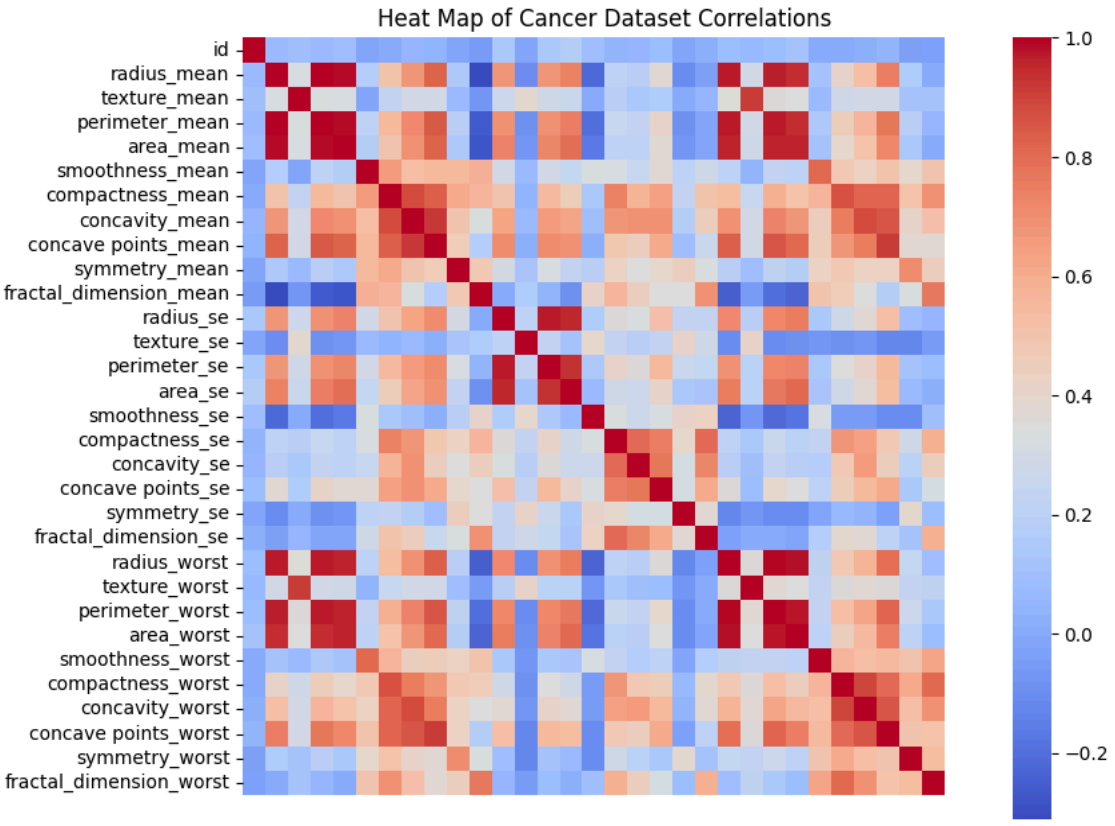
Dimensione Frattale Peggiore	Feature	Continuous (0-3000)	Peggior valore osservato della dimensione frattale delle cellule.
Diagnosi	**Target**	Binary (0 / 1)	Tipo di diagnosi del tumore. (0 = Benigno / 1 = Maligno)

#1.2 OSSERVAZIONE GRAFICA DEI DATI

Abbiamo svolto delle osservazioni grafiche che ci permettessero di valutare la correlazione dei dati e soprattutto per capire in che modo la diagnosi fosse influenzata da essi.
Qui di seguito riportiamo una heatMap che abbiamo realizzato in linguaggio Python:

MATRICE DI CORRELAZIONE
Una **matrice di correlazione** è un insieme di valori numerici che esprimono relazioni tra le feature del DataSet.

La rappresentazione di questa matrice di correlazione viene effettuata tramite **HeatMap**, utilizzabile tramite la libreria **Seaborn** in Python.



#2 ONTOLOGIA

Un'**Ontologia** è una rappresentazione strutturata della conoscenza in un certo dominio.
In parole semplici, è un modo per organizzare e definire i concetti e le relazioni tra di essi in modo chiaro e formale.

Durante l'analisi del dominio si identificano i **concetti** principali, le **relazioni** e le **proprietà** che caratterizzano il dominio.
Successivamente questi concetti, relazioni e proprietà possono essere formalizzate mediante un linguaggio di rappresentazione formale come ad esempio **OWL - Ontology Web Language**.

#2.1 ANALISI DOMINIO

Abbiamo deciso di dividere gli attributi del DataSet in:

- ***"ciò che deve essere rappresentato"*:**
  Gli attributi fondamentali e cruciali che devono essere rappresentati per catturare le informazioni essenziali dal DataSet.
  - Nel nostro contesto si parla di Pazienti, Diagnosi e Cancro.
- ***"ciò che caratterizza ciò che deve essere rappresentato"*:**
  Le proprietà delle entità rappresentate.
  - Nel caso del Cancro, ad esempio, possono essere area, perimetro, etc.

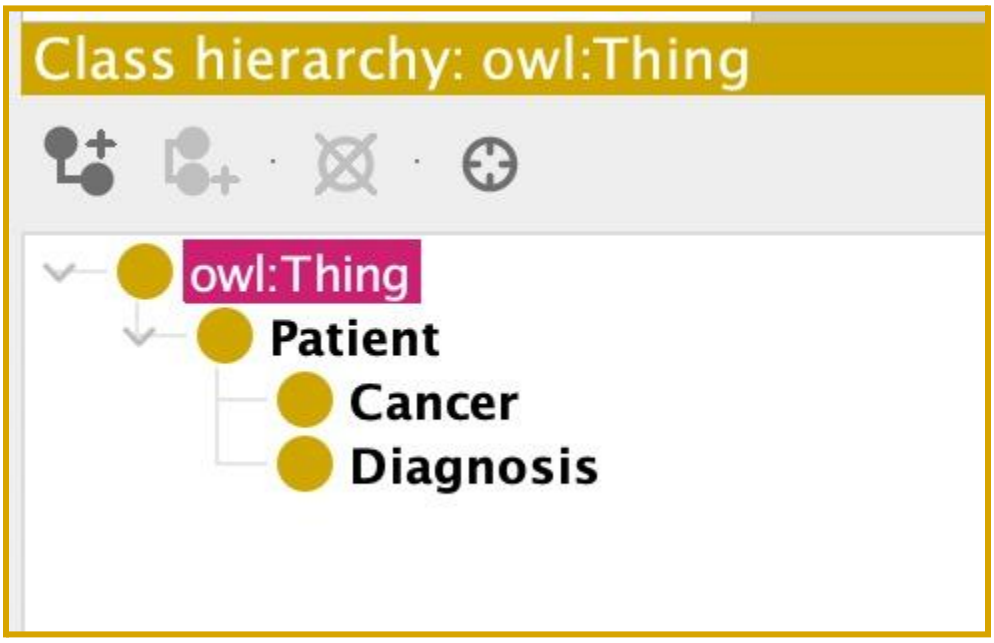
#2.2 SOFTWARE PER LA REALIZZAZIONE DELL'ONTOLOGIA

L'ontologia è stata creata mediante il software **Protegè**, software open-source per creare e gestire ontologie e che permette di definire concetti, classi e relazioni in ontologie basate su standard come **OWL**.

#2.2.1 CLASSI

Abbiamo deciso di rappresentare come **Classi/Entity** dell'ontologia:

- **Patient**:
  Rappresenta l'insieme dei pazienti sottoposti al test.
  A questa classe abbiamo associato un'unica proprietà:
  - id: identificativo unico per il riconoscimento
- **Diagnosis**:
  Sottoclasse di Patient.
  Rappresenta la diagnosi del paziente riguardante il tumore che ha come proprietà:
  - Diagnosis_value: può avere come valori benigno (B) o maligno (M) per verificare lo stato del tumore.
- **Cancer**:
  Sottoclasse di Patient.
  Rappresenta l'insieme dei tumori diagnosticati ai vari pazienti a cui vengono associati i valori visivi:
  - radius_mean
  - texture_mean
  - perimeter_mean
  - ...

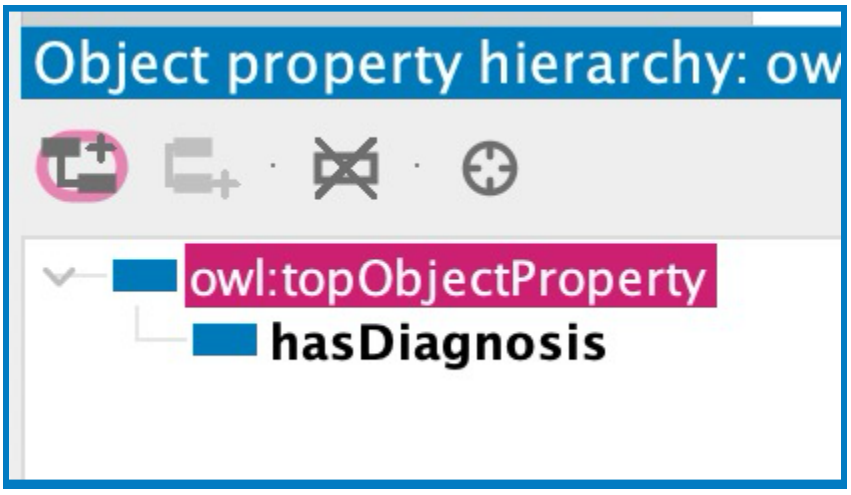


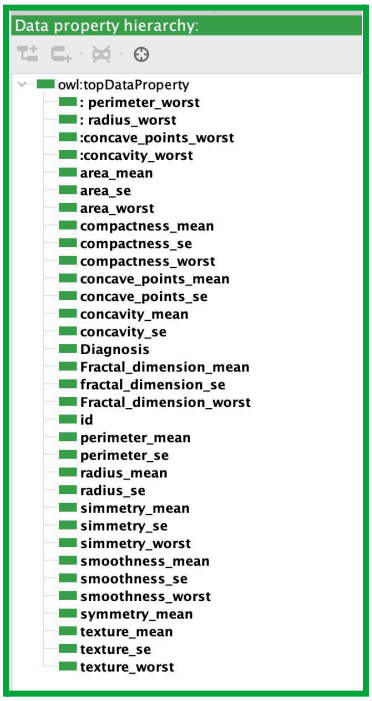
#2.2.2 OBJECT PROPERTY

Una **object property** permette di mettere in relazione due individui, siano essi di classi distinte o della stessa classe.

Tra le classi abbiamo definito una relazione che rappresenta come queste interagiscono tra di loro. La relazione in questione:

- ***hasDiagnosis(Patient,Cancer) -> Diagnosis***





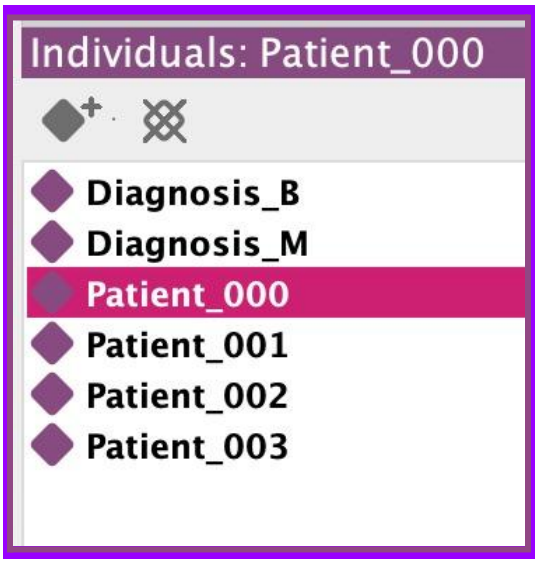
#2.2.3 DATA PROPERTY

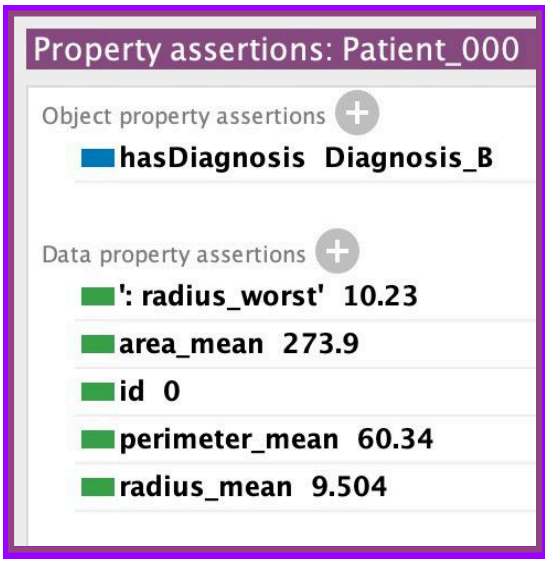
Una **data property** definisce attributi o informazioni specifiche associate a un'istanza, mette in relazione un individuo con un valore primitivo.

- Le data property dei pazienti sono:
  l'id per identificarlo e i valori delle sue sottoclassi.
- Le data property di Cancer sono:
  i vari valori visivi (area, perimeter, radius, …)
- Le data property di Diagnosis è:
  diagnosis_value.

#2.2.4 INDIVIDUALS

Per alcune entità abbiamo inserito delle **istanze** (**individuals**), per individuare pazienti a cui attribuire lo stato della diagnosi, ed effettuare prove di [query DL](#).



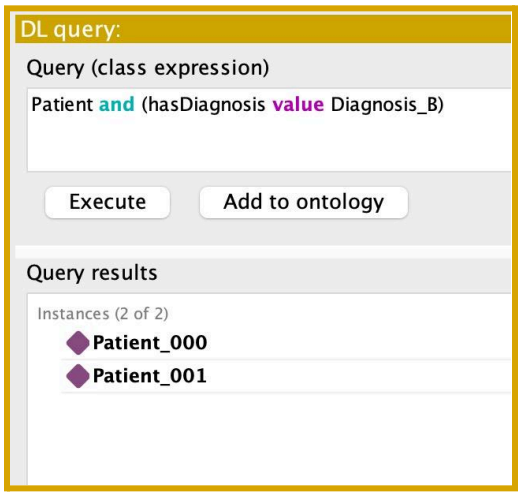


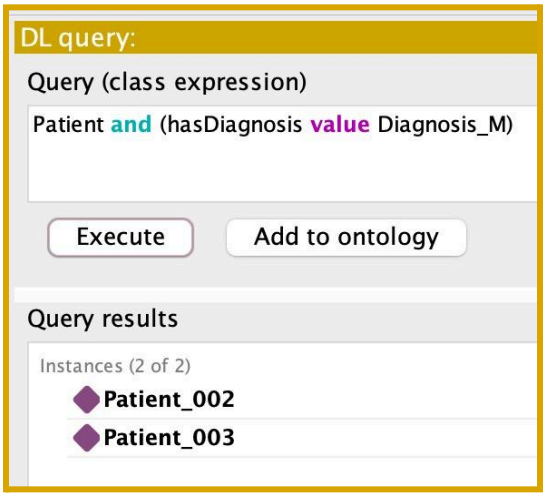
#3 QUERY

#3.1 DL Query

Interroghiamo l'ontologia tramite **DL - Descriptions Logics**, che è un linguaggio formale utilizzato principalmente per la modellazione concettuale e l'ontologia nelle discipline dell'intelligenza artificiale e della rappresentazione della conoscenza.

QUERY REALIZZATE:





#3.1 OwlReady2

È stato poi creato il file “ontologyQuery.py” con il quale è possibile consultare l’ontologia direttamente in Python grazie alla libreria **OwlReady2** per la manipolazione di ontologie e il ragionamento.

Con il seguente codice è quindi possibile estrarre dall’ontologia la lista di classi, object property e data property.

```
print("ONTOLOGIA\n")
onto = get_ontology("2.Ontologia/Ontologia.owl").load()


# Stampa di tutte le classi dell'ontologia
print("############################################################")
print("LISTA DELLE CLASSI DELL'ONTOLOGIA\n")
classes = list(onto.classes())
for cls in classes:
    print(f"• CLASSE: {cls.name}")
print()

# Stampa di tutte le object properties dell'ontologia
print("############################################################")
print("LISTA DELLE OBJECT PROPERTIES DELL'ONTOLOGIA\n")
object_properties = list(onto.object_properties())
for prop in object_properties:
    print(f"• OBJECT PROPERTY: {prop.name}")
print()

# Stampa di tutte le data properties dell'ontologia
print("############################################################")
print("LISTA DELLE DATA PROPERTIES DELL'ONTOLOGIA\n")
data_properties = list(onto.data_properties())
for prop in data_properties:
    print(f"• PROPERTY: {prop.name}")
print()
```

```
ONTOLOGIA

############################################
LISTA DELLE CLASSI DELL'ONTOLOGIA

• CLASSE: Cancer
• CLASSE: Diagnosis
• CLASSE: Patient

############################################
LISTA DELLE OBJECT PROPERTIES DELL'ONTOLOGIA

• OBJECT PROPERTY: hasDiagnosis

############################################
LISTA DELLE DATA PROPERTIES DELL'ONTOLOGIA

• PROPERTY: area_mean
• PROPERTY: area_se
• PROPERTY: area_worst
• PROPERTY: compactenss_mean
• PROPERTY: compactenss_se
• PROPERTY: compactenss_worst
• PROPERTY: concave_points_mean
• PROPERTY: concave_points_se
• PROPERTY: concave_points_worst
• PROPERTY: concavity_mean
• PROPERTY: concavity_se
• PROPERTY: concavity_worst
• PROPERTY: diagnosis
• PROPERTY: fractal_dimension_mean
• PROPERTY: fractal_dimension_se
• PROPERTY: fractal_dimension_worst
• PROPERTY: id
• PROPERTY: perimeter_mean
• PROPERTY: perimeter_se
• PROPERTY: perimeter_worst
• PROPERTY: radius_mean
• PROPERTY: radius_se
• PROPERTY: radius_worst
• PROPERTY: simmetry_mean
• PROPERTY: simmetry_se
• PROPERTY: simmetry_worst
• PROPERTY: smoothness_mean
• PROPERTY: smoothness_se
• PROPERTY: smoothness_worst
• PROPERTY: texture_mean
• PROPERTY: texture_se
• PROPERTY: texture_worst
```

Attraverso il seguente codice è stato invece possibile interrogare l’ontologia tramite query.

```
print("\n##############################################QUERIES###############")

# Query per ottenere pazienti con diagnosi "B" (Benigni)
diagnosis_B = onto.search_one(type=onto.Diagnosis, diagnosis="B")
query_result = list(onto.search(type=onto.Patient, hasDiagnosis=diagnosis_B))

print("\nPatients with Benigni diagnosis:")
for patient in query_result:
    print(f"Patient: {patient.name}")


# Query per ottenere pazienti con diagnosi "M" (Maligni)
diagnosis_M = onto.search_one(type=onto.Diagnosis, diagnosis="M")
query_result = list(onto.search(type=onto.Patient, hasDiagnosis=diagnosis_M))

print("\nPatients with Maligni diagnosis:")
for patient in query_result:
    print(f"Patient: {patient.name}")
```

```
############################################

PAZIENTI CON DIAGNOSI BENIGNA:
• PAZIENTI: Patient_000, Patient_001

PAZIENTI CON DIAGNOSI MALIGNA:
• PAZIENTI: Patient_002, Patient_003
```

#4 APPRENDIMENTO SUPERVISIONATO

L'**apprendimento supervisionato** è un tipo di apprendimento automatico in cui un modello viene addestrato utilizzando un insieme di dati già etichettati.
Al modello vengono forniti esempi con input e i relativi output corretti, e il suo compito è imparare a riconoscere le relazioni tra loro (**generalizzare**), in modo da poter fare previsioni su nuovi dati in futuro.
A seconda del tipo di problema, l'apprendimento supervisionato può essere usato per la **classificazione**, quando il dominio dei valori di output è **discreto e finito**, oppure per la **regressione**, quando il dominio dei valori di output è **continuo e infinito**.
Durante il processo di addestramento, il modello analizza i dati e cerca di individuare schemi e relazioni.
Una volta addestrato, viene testato su nuovi dati per verificare se ha davvero imparato a generalizzare o se ha semplicemente memorizzato gli esempi.

#4.1 DECISIONI DI PROGETTO

In questo progetto, l'obiettivo affidato all'apprendimento supervisionato è risolvere un **problema di classificazione**.
Abbiamo utilizzato la colonna "*Diagnosis*" del DataSet come **variabile target** da predire basandosi sulle altre caratteristiche dei pazienti.

La prima cosa che abbiamo fatto è stata decidere quali modelli di apprendimento supervisionato utilizzare.
Data l'elevata sensibilità del tema, quale lo stato di un tumore, era essenziale adottare algoritmi che garantissero alte prestazioni e precisione nelle previsioni.
Abbiamo scelto di realizzare i modelli consigliati dagli autori nonché contributori del DataSet, ovvero **SVM** e **K-NN**.
Tuttavia abbiamo deciso di provare questi dati anche con altri due modelli quali **Random Forest** e **Neural Network**.

RECAP MODELLI REALIZZATI:
1. **K NN - K Nearest Neighbors**
2. **Random Forest**
3. **SVM - Support Vector Machine**
4. **Neural Network**

#4.2 METRICHE

Andiamo adesso ad analizzare le metriche con cui sono stati valutati i modelli ed i risultati.

#4.2.1 DI VALIDAZIONE

La **Cross-Validation** è un'altra tecnica utilizzata per valutare le prestazioni dei modelli in modo accurato e affidabile.

Abbiamo scelto di dividere i dati in **5 fold** uguali, in modo da addestrare e testare il modello 5 volte, ognuna delle quali usa una fold diversa come set di test e l'unione delle rimanenti come set di addestramento.
La valutazione verrà eseguita sulla Cross-Validation: alcune metriche valuteranno la migliore delle sue iterazione, mentre altre valuteranno le prestazioni complessive su tutte le iterazioni.

#4.2.2 DI VALUTAZIONE

Per ogni algoritmo implementato sono stati prodotti 4 tipologie di grafici differenti:

- **ROC Curve (Receiver Operating Characteristic Curve)**:
  Grafico in cui l'asse delle ordinate rappresenta il **True Positive Rate**.
  L'asse delle ascisse rappresenta il **False Positive Rate**.
- **Precision-Recall Curve**:
  Questa curva rappresenta il trade-off tra la **Precision** e il **Recall** di un modello di classificazione binaria.
- **Bar Chart di Varianza e Deviazione Standard**:
  Questo tipo di grafico a barre rappresenta la **Varianza** e la **Deviazione Standard** dei punteggi di **cross-validation**.
  Aiutano a comprendere quanto i risultati siano consistenti o variabili su diverse iterazioni della cross-validation.
- **Matrice di Confusione**:
  La matrice di confusione è una tabella che mostra il numero di previsioni corrette ed errate fatte da un modello di classificazione in termini di veri positivi **TP**, falsi positivi **FP**, veri negativi **TN** e falsi negativi **FN**.
- **Stampa finale di alcune metriche**:
  Classification Report e dei risultati stampati nel terminale:
  esplorazione del DataSet, cv_scores_mean, cv_scores_variance, accuracy score, cv_score_devstandard, AUC e f1_score.

SIAMO CONSAPEVOLI CHE:
Alcune di queste metriche si basano sul modello finale, allenato con i migliori parametri ottenuti dalla Cross-Validation, e **non tengono conto dell'interezza dei risultati** ottenuti durante il processo **di Cross-Validation**.

TUTTAVIA:
Dopo diverse esecuzioni abbiamo verificato la **stabilità dei risultati**, motivo per cui abbiamo deciso di includerle.

PROPRIO PER QUESTO:
Per garantire una valutazione più affidabile, **abbiamo comunque inserito metriche che evidenziano la variabilità dei risultati della Cross-Validation e la stabilità del modello durante l'ottimizzazione dei parametri**, come ad esempio la Varianza e la Deviazione Standard.

#4.3 SELEZIONE DELLE FEATURE

FEATURES SELEZIONATE:
Abbiamo ritenuto che nessuna delle features del nostro DataSet fosse irrilevante, in quanto erano tutte impattanti sulla diagnosi.

FEATURES SCARTATE:
/

#4.4 PREPROCESSING DEI DATI

L'addestramento di ogni modello inizia con l'esecuzione del Preprocessing dei dati.

#4.4.1 DUMMIFICATION & LABEL ENCODING

La **Dummification** è un passo di Preprocessing che serve a trasformare feature categoriche in feature numeriche. Nello specifico ogni valore di dominio della feature, quindi ogni categoria, diventa una nuova feature binaria il cui valore indica l'assenza o la presenza di quella categoria.

Nel nostro caso non abbiamo esattamente svolto un lavoro di Dummification ma piuttosto un lavoro di LABEL ENCODING in quanto abbiamo trasformato una variabile da categoriale binaria {'M', 'B'} a numerica binaria {1, 0}.

```
# Preprocessing
dataset['diagnosis'] = dataset['diagnosis'].map({'M': 1, 'B': 0})
```

#4.4.2 SMOTE

La funzione **`SMOTE()`**, implementabile grazie all'uso della libreria **imblearn**, ha lo scopo di ridimensionare la classe di esempi quando si lavora con DataSet sbilanciati, dove una classe è rappresentata in modo significativamente minore rispetto all'altra classe.

Questo passo non è stato necessario in quanto il nostro DataSet è stato PREBILANCIATO dagli autori.

#4.4.3 STANDARDIZZAZIONE

Usiamo la funzione **`StandardScaler()`** della libreria **sklearn**.
Questo viene fatto per evitare che variabili con scale diverse influenzino in modo sproporzionato modelli di machine learning che sono sensibili alle differenze nelle scale dei dati.

Questo passo non è stato necessario per il modello di apprendimento supervisionato RANDOM FOREST (**non** dipende dalle distanze delle osservazioni)**.**

Questo passo è risultato FONDAMENTALE per modelli di Machine Learning quali K-NN, NEURAL NETWORK, SVM, K-MEANS (dipendono dalle distanze delle osservazioni)**.**

```
# Standardizzazione
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

#4.5 DIVISIONE DEI DATI

SUDDIVISIONE DATASET: **80% TrainSet** – **20% TestSet**.

Questa suddivisione è sembrata un compromesso ragionevole tra la necessità di avere abbastanza dati per l'addestramento e la necessità di avere abbastanza dati per valutare accuratamente le prestazioni del modello.

- **Con una proporzione PIÙ ALTA per il TestSet**:
  Avremmo avuto meno dati disponibili per l'addestramento del modello, ciò avrebbe potuto comportare una **minore capacità del modello di apprendere** le complessità dei dati e avrebbe potuto portare a una generalizzazione meno accurata.
- **Con una proporzione PIÙ ALTA per il TrainSet**:
  Saremmo potuti incorrere in un rischio di **overfitting**, acquisendo anche il rumore nei dati e risultando meno in grado di generalizzare su nuovi dati.

```
# Divido il dataset in set di addestramento e test (80% training, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42, shuffle=True, stratify=y)
```

#4.6 K-NEAREST NEIGHBORS (K-NN)

L'algoritmo su cui si basa è il **Nearest Neighbor (NN)**.
Questo è un algoritmo di classificazione che assegna al dato in input classe pari alla classe dell'esempio che gli è più simile.
La similarità è calcolata in base ad una metrica come ad esempio la Distanza Euclidea.

Il **K-NN** è la sua versione migliorata in cui si contano le classi dei K vicini più vicini e si assegna al nostro dato la classe più frequente tra loro.

#4.6.1 DECISIONI DI PROGETTO

Abbiamo scelto di implementare inizialmente il modello K-NN perché consigliato dagli autori del dataset "CancerData" abbiamo riscontrato essere **semplice** da realizzare, **versatile** e **performante** su DataSet di qualsiasi dimensione.
Inoltre, si dimostra particolarmente **robusto** in presenza di rumore e valori anomali e quindi avranno un impatto limitato sull'output complessivo, poiché il risultato finale si basa sui dati circostanti.

#4.6.2 OTTIMIZZAZIONE DEL VALORE 'K'

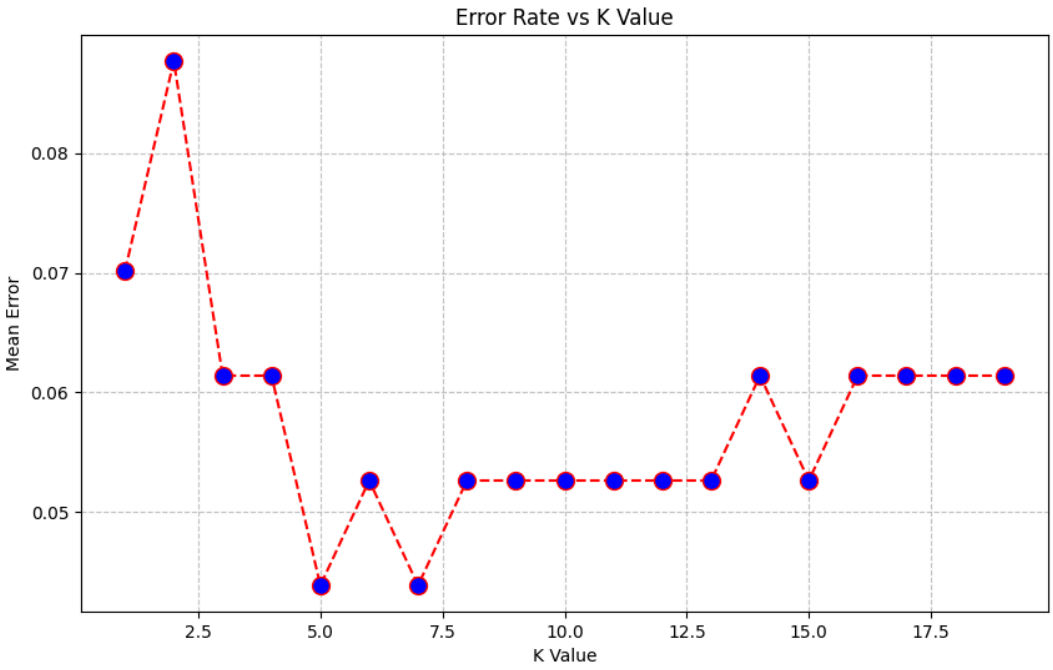
In genere:

- Valori di K troppo piccoli portano a problemi di **overfitting**.
- Valori di K troppo grandi portano a problemi di **underfitting**.

Per trovare il **valore ottimale** di K (numero di vicini) si itera per k da 1 a √N
(N = numero di osservazioni nel DataSet).
Per ogni k si:

- Crea un classificatore K-NN con k vicini.
- Addestra il modello sui dati di training (X_train, y_train).
- Effettua previsioni su X_test.
- Calcola l'errore medio di classificazione e lo salva in error.

```
# Calcolo del numero di vicini ottimale per il KNN sulla base del minor mean error
error = []
for i in range(1, 20):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))
```



Consultando il grafico ottenuto constatiamo che esistono 2 valori ottimali per il parametro k.
Abbiamo deciso di scegliere tra questi il valore minore in modo tale da ottenere un modello più sensibile e reattivo. Il valore in questione è **k = 5**.

#4.6.3 ADDESTRAMENTO

Creiamo il nostro classificatore con **k** vicini.
Usiamo il metodo **`fit()`** in **scikit-learn** per addestrare un modello sui dati forniti.

A differenza di altri algoritmi di machine learning il K-NN non apprende un modello matematico basato sui dati di training ma memorizza semplicemente i dati di training.

```
# Determinazione K ottimale
optimal_k = error.index(min(error)) + 1
print(f"\nK ottimale trovato: {optimal_k}")

# Training con K ottimale
neigh = KNeighborsClassifier(n_neighbors=optimal_k)
neigh.fit(X_train_scaled, y_train)
```

#4.6.4 PREDIZIONE

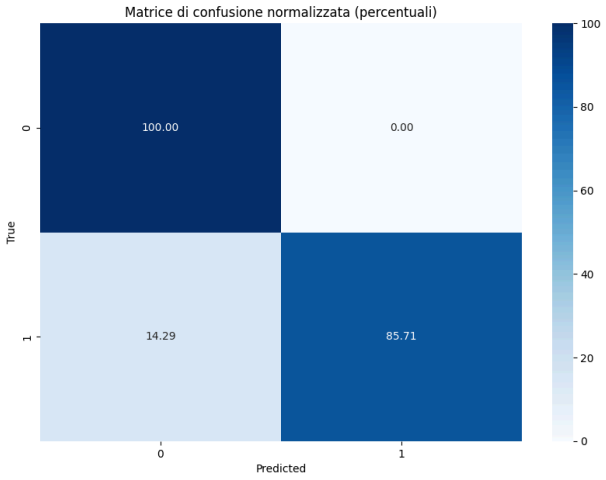
Usiamo il metodo **`predict()`** in **scikit-learn** per fare **previsioni** su ogni dato del TestSet basandosi sulle informazioni apprese dal modello addestrato in precedenza.

Andiamo poi a calcolare l'**accuratezza** facendo il rapporto tra il numero di previsioni corrette e il numero totale di previsioni effettuate.
Usiamo quindi il metodo **`accuracy_score()`** in **scikit-learn** che restituisce la percentuale di previsioni corrette sul TestSet.

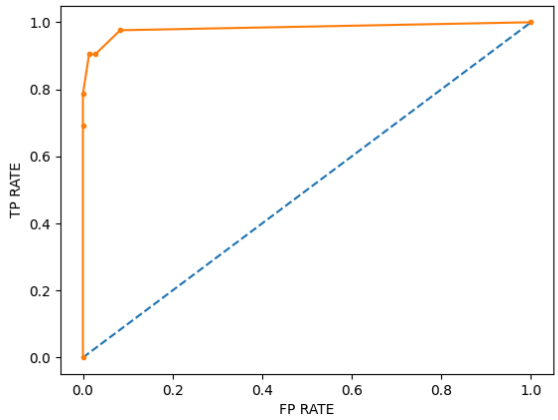
```
# Effettua previsioni sul test set
prediction = neigh.predict(X_test)
accuracy = accuracy_score(y_test, prediction)
print(f"\nAccuracy score: {accuracy:.2f}")
```

#4.6.5 VALUTAZIONE FINALE

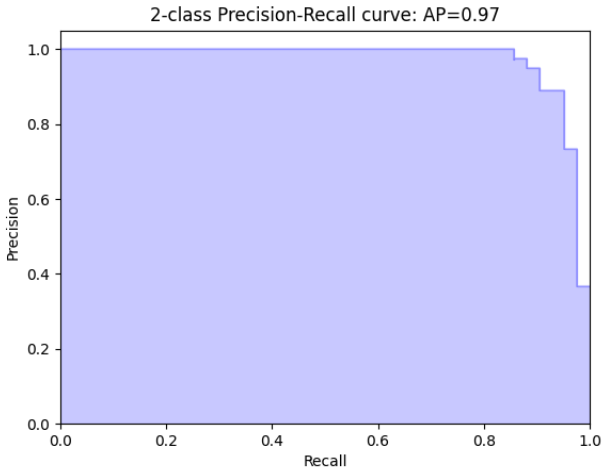
MATRICE DI CONFUSIONE
Il grafico mostra un'ottima capacità del modello nel riconoscere le classi positive (Cancro Malevolo).
Il modello potrebbe avere qualche errore nel riconoscere i falsi positivi (Cancro Benevolo).
Poiché diamo maggiore importanza ai TP riteniamo il modello efficiente.



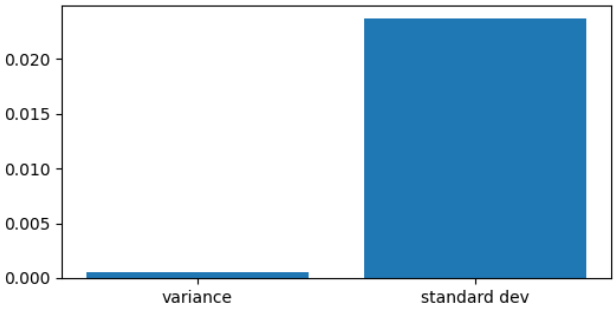
ROC CURVE
Il grafico ROC mostra un'eccellente capacità di riconoscere i positivi senza falsi allarmi iniziali. Da FP R = 0.1, il TP R raggiunge 1 e si stabilizza, segnalando una quasi perfetta separazione tra classi.
L'AUC di 0.979 conferma l'alta efficacia del modello nel distinguere tra classi positive e negative.



PRECISION-RECALL CURVE
Il grafico un'ottima capacità di classificare correttamente i positivi iniziali.
Da lì, la precisione cala gradualmente fino a 0.6 e scendendo più bruscamente a 0.4 sul finale segnalando un aumento dei falsi positivi nelle ultime predizioni.
L'AP di 0.97 conferma l'elevata qualità complessiva del modello



BAR CHART DI VARIANZA E DEVIAZIONE STANDARD
Il grafico indica una bassa dispersione nei risultati del modello. Ciò suggerisce che il modello ha performance stabili e costanti attraverso le diverse esecuzioni, con fluttuazioni minime nelle sue predizioni.

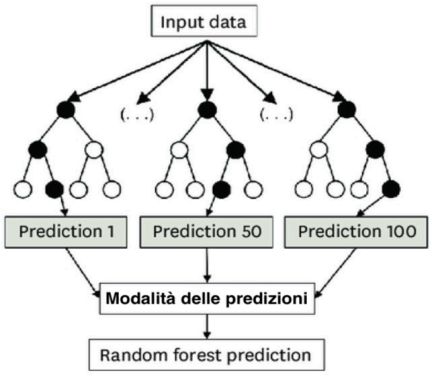


```
0   id                        569 non-null    int64
1   diagnosis                 569 non-null    int64
2   radius_mean               569 non-null    float64
3   texture_mean              569 non-null    float64
4   perimeter_mean            569 non-null    float64
5   area_mean                 569 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
```

```
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
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97        72
           1       0.97      0.90      0.94        42

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114


Confusion matrix:
 [[71  1]
 [ 4 38]]

Cross-validation results:
Mean accuracy: 0.9626
Standard deviation: 0.0237

cv_score variance:0.0005603188020770436

AUC: 0.982

F1 Score: 0.9383
```

#4.7 RANDOM FOREST



Algoritmo che crea una "foresta" di **alberi decisionali**, ciascuno dei quali prende decisioni basate su un campione casuale dei dati e delle caratteristiche.
Il risultato finale è dato dalla:

- **Media** nel caso di **Regressione**.
- **Modalità** nel caso di **Classificazione**.

delle previsioni fatte da tutti gli alberi.

#4.7.1 DECISIONI DI PROGETTO

I vantaggi del Random Forest includono una buona capacità di **generalizzazione**, resistenza all'**overfitting**, la capacità di **gestire sia feature numeriche che categoriali** e può **gestire facilmente classi sbilanciate**.
Tende ad avere una buona **precisione** e **stabilità** nei problemi di classificazione.
Inoltre, la sua capacità di addestrare gli alberi in modo parallelizzato consente una **rapida esecuzione** anche con risorse computazionali limitate.

#4.7.2 OTTIMIZZAZIONE PARAMETRI DEL CLASSIFICATORE

Per costruire il classificatore Random Forest usiamo il metodo `RandomForestClassifier()` presente nella libreria **scikit-learn** che prende 3 parametri:

- `max_depth`:
  La profondità massima dell'albero decisionale.
  - Valore Alto: alberi più **complessi**, ma con rischio di **overfitting**.
  - Valore Basso: alberi più **semplici**, ma con rischio di **underfitting**.
- `random_state`:
  Controlla la generazione dei numeri casuali all'interno del modello. Fornendo un valore specifico ci si assicura che l'addestramento e la previsione del modello siano riproducibili.
- `n_estimators`:
  Numero di alberi decisionali.
  - Valore Alto: modello più **complesso e performante**, ma con rischio di **overfitting**.
  - Valore Basso: modello più **veloce**, ma con **performance inferiori**.

STEP #1
Per ottimizzare i parametri del modello Random Forest andiamo per prima cosa a definire un intervallo di valori da testare per ciascun parametro.
STEP #2

Utilizziamo cicli annidati per testare tutte le possibili combinazioni dei valori definiti per i parametri.
Per ogni combinazione di parametri:

- Viene creato un modello Random Forest con i parametri specificati.
- Viene eseguita una cross-validation a 5 fold tramite **`cross_val_score`**.
  La cross-validation suddivide il DataSet in 5 parti (fold), addestra il modello su 4 parti e testa la sua performance sulla parte rimanente.
  Questo processo viene ripetuto 5 volte, con ogni parte utilizzata una volta per il test. La funzione **`cv_scores.mean()`** calcola la media del punteggio di cross-validation.

STEP #3
Trovata la combinazione ottimale viene creato il modello finale.

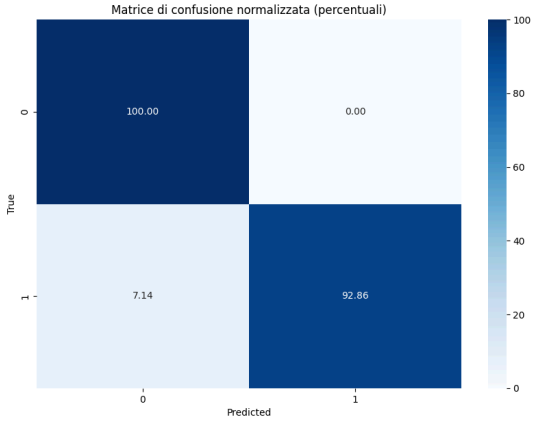
```python
# intervallo di valori da testare per max_depth
max_depth_values = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

# intervallo di valori per random_state
random_state_values = [0, 4, 16, 64, 256, 1024, 4096]

# Memorizza i punteggi medi di cross-validation per ogni combinazione di max_depth e random_state
scores = []
for max_depth in max_depth_values:
    for random_state in random_state_values:
        RFC = RandomForestClassifier(max_depth=max_depth, random_state=random_state)
        cv_scores = cross_val_score(RFC, X, y, cv=5)
        scores.append((max_depth, random_state, cv_scores.mean()))
```

```
Valore migliore max_depth: 8.0
Valore migliore random_state: 16.0
```

#4.7.3 VALUTAZIONE FINALE

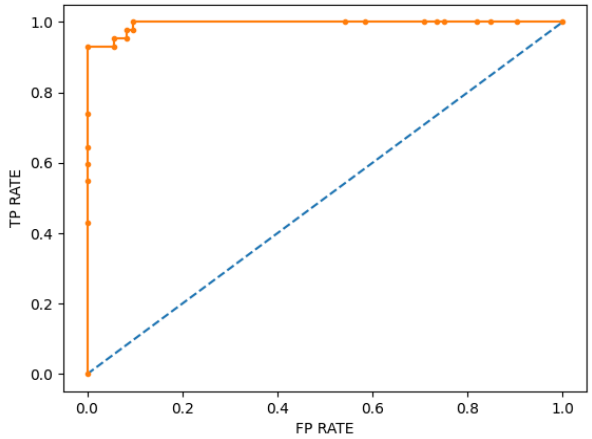
MATRICE DI CONFUSIONE

Il grafico indica un modello altamente efficace nel riconoscere correttamente sia le classi positive che negative.
La combinazione di TP perfetti e una buona percentuale di TN suggerisce un modello bilanciato, con un F1-score elevato che riflette prestazioni complessive molto robuste.



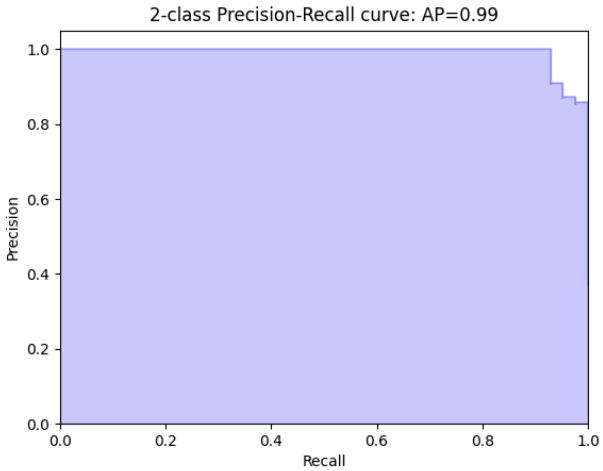
ROC CURVE

Il grafico ROC evidenzia un'eccellente capacità del modello di identificare i positivi senza falsi allarmi iniziali.
A partire da un FP Rate di 0.15, il TP Rate raggiunge 1 e si stabilizza, indicando una separazione quasi perfetta tra le classi.
L'AUC di 0.994 conferma un'accuratezza eccezionale del modello.



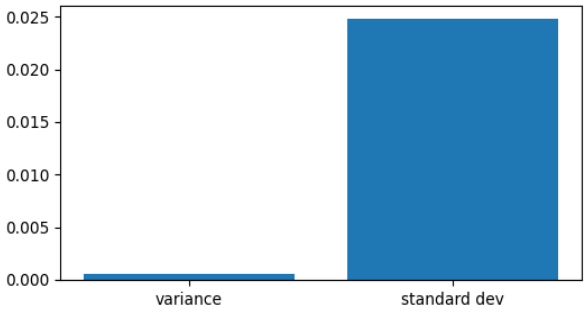
PRECISION-RECALL CURVE

Il grafico mostra un'ottima capacità del modello di classificare correttamente i positivi iniziali.
Da recall = 0.92 fino a 1, la precisione diminuisce gradualmente segnalando un lieve aumento dei falsi positivi.
L'AP di 0.99 conferma un'eccezionale performance complessiva del modello.



BAR CHART DI VARIANZA E DEVIAZIONE STANDARD

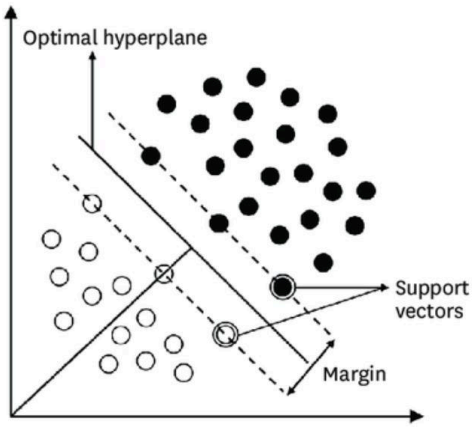
Il grafico mostra che i risultati del modello sono molto stabili, con una bassa dispersione tra le diverse esecuzioni. Questo suggerisce che le prestazioni del modello sono costanti e affidabili.




```
Classification report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        72
           1       1.00      0.93      0.96        42

    accuracy                           0.97       114
   macro avg       0.98      0.96      0.97       114
weighted avg       0.97      0.97      0.97       114


Confusion matrix:
 [[72  0]
 [ 3 39]]




cv_scores mean:0.9648967551622419

cv_score variance:0.000614757853287195

cv_score dev standard:0.0247943109056734


AUC: 0.994

f1 score:  0.9629629629629629
```

#4.8 SUPPORT VECTOR MACHINES (SVM)

L'obiettivo principale di una **SVM** è trovare un **iperpiano** che **separi ottimamente le classi** o **predica i valori target** (nel caso della regressione). Le **SVM massimizzano il margine** tra l'iperpiano e i **support vectors** (i punti più vicini), rendendo il modello **più robusto e resistente all'overfitting**.



#4.8.1. DECISIONI DI PROGETTO

Questo modello è stato scelto, oltre per il fatto di essere stato consigliato dagli autori del DataSet, in quanto **potente** per la **classificazione binaria**, in particolare quando le classi sono ben separate.
È adatto per problemi con un numero moderato di feature e può **gestire relazioni non-lineari** attraverso l'uso di kernel.

Abbiamo scelto di usare il **kernel RBF - Radial Basis Function** per questo modello perché è possibile comprendere strutture complesse e non lineari che potrebbero essere presenti nei dati, migliorando le probabilità di ottenere un modello di classificazione più accurato e generalizzabile.

#4.8.2 OTTIMIZZAZIONE PARAMETRO GAMMA

Il parametro **gamma** è un iperparametro cruciale quando si usa il kernel RBF in un classificatore SVM.

Controlla quanto l'influenza di un singolo esempio di addestramento si estenda.

- Valori più alti di gamma rendono le decisioni più locali.
- Valori più bassi danno una maggiore influenza alle istanze circostanti.

Per trovare il miglior valore gamma abbiamo utilizzato la tecnica della **grid search** insieme alla cross-validation.

STEP #1
Si inizia definendo una griglia di valori di gamma.
STEP #2
Dopo aver creato un modello SVM, si esegue la ricerca a griglia che addestrerà e valuterà il modello su diverse combinazioni di parametri gamma utilizzando la cross-validation con 5 fold.
STEP #3

Con il metodo fit, si otterrà il modello SVM ottimizzato con i migliori parametri gamma individuati dalla grid search.

```python
# Definisco i valori di gamma da testare
param_grid = {'gamma': [1e-4, 1e-3, 0.01, 0.1, 1, 10, 100]}

# Creazione del modello SVC
svm_model = svm.SVC(kernel='rbf', probability=True)

# Ricerca a griglia con cross-validation
grid_search = GridSearchCV(svm_model, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Accesso ai risultati della ricerca a griglia
results = grid_search.cv_results_

# Stampa dei valori di gamma testati e delle prestazioni corrispondenti
for gamma, mean_score in zip(results['param_gamma'], results['mean_test_score']):
    print(f"Gamma: {gamma:<10}Mean Score: {mean_score}")

# Stampa il valore di gamma migliore
print("\nMiglior valore di gamma:", grid_search.best_params_['gamma'])
```

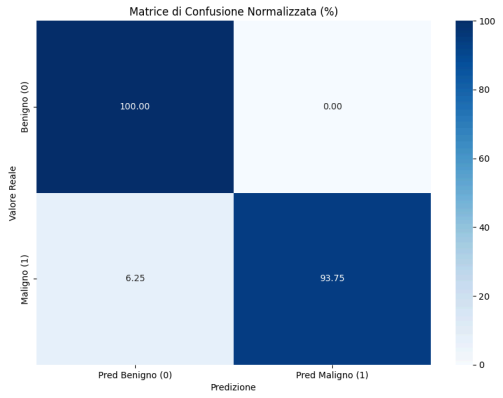
```
Gamma: 0.0001    Mean Score: 0.9378006872852233
Gamma: 0.001     Mean Score: 0.9357388316151202
Gamma: 0.01      Mean Score: 0.6252577319587629
Gamma: 0.1       Mean Score: 0.627319587628866
Gamma: 1.0       Mean Score: 0.627319587628866
Gamma: 10.0      Mean Score: 0.627319587628866
Gamma: 100.0     Mean Score: 0.627319587628866

Miglior valore di gamma: 0.0001
```

Questi risultati ci fanno capire che il modello sarà ottimizzato per il valore gamma **0.0001**.

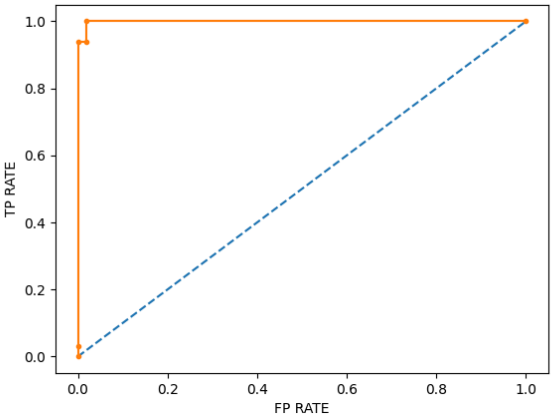
```python
# Addestramento del classificatore SVM con il miglior gamma
clf = svm.SVC(kernel='rbf', gamma=grid_search.best_params_['gamma'], probability=True)
clf.fit(X_train, y_train)
```

#4.8.3 VALUTAZIONE FINALE

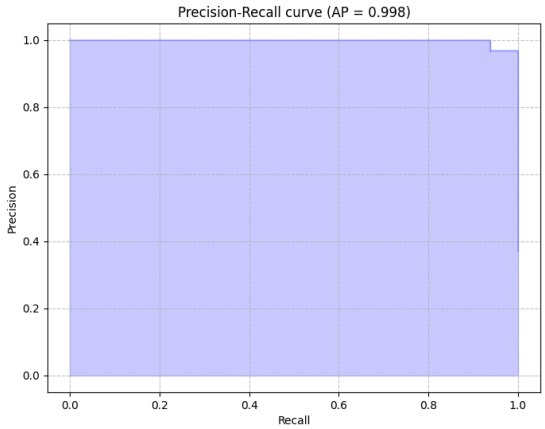
MATRICE DI CONFUSIONE
Il grafico mostra un'ottima capacità del modello nel riconoscere le classi positive (Cancro Malevolo).
Il modello potrebbe avere qualche errore nel riconoscere i falsi positivi (Cancro Benevolo).
Poiché diamo maggiore importanza ai TP riteniamo il modello efficiente.



ROC CURVE
Il grafico ROC segnala una buona capacità iniziale di riconoscere i positivi senza falsi allarmi.
Da FPR = 0.15, il TPR cresce fino a 0.9, per poi raggiungere 1 a partire da FPR = 0.98, indicando una separazione quasi perfetta tra le classi.
L'AUC di 0.965 conferma un'elevata performance complessiva del modello.

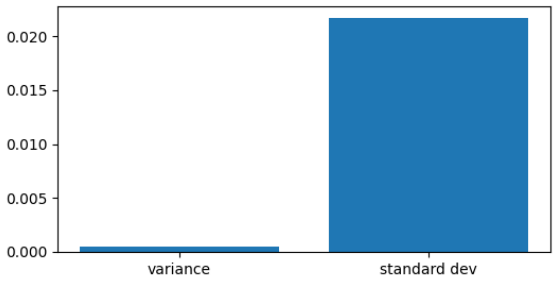


PRECISION-RECALL CURVE
La Precision-Recall Curve mostra un'ottima capacità di classificare correttamente i positivi iniziali.
Da recall = 0.84 fino a 1, la precisione diminuisce gradualmente per poi calare significativamente a 0.4 sul finale suggerendo un aumento dei falsi positivi.
L'AP di 0.97 conferma una performance complessiva molto buona del modello.



BAR CHART DI VARIANZA E DEVIAZIONE STANDARD

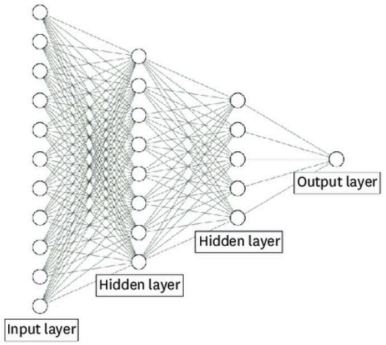
Il grafico indica che i risultati del modello sono molto stabili, con una bassa dispersione tra le diverse esecuzioni. Questo suggerisce che il modello ha prestazioni consistenti e affidabili.




```
Miglior score: 0.9689

Accuracy Score: 0.9767

Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        54
           1       1.00      0.94      0.97        32

    accuracy                           0.98        86
   macro avg       0.98      0.97      0.97        86
weighted avg       0.98      0.98      0.98        86


Confusion matrix:
 [[54  0]
 [ 2 30]]

Risultati Cross-validation:
Media accuracy: 0.9689
Deviazione standard: 0.0217
Varianza: 0.0005
AUC: 0.999

F1 Score: 0.9677
```

#4.9 NEURAL NETWORK

Le Reti Neurali simulano neuroni artificiali collegati tra loro, elaborando stimoli attraverso pesi e soglie.
L'attivazione avviene se il risultato supera la soglia, i pesi quantificano l'importanza degli input.



#4.9.1 DECISIONI DI PROGETTO

Il modello è stato scelto per la sua capacità di affrontare **problemi di apprendimento profondi e complessi** e per la sua capacità di raggiungere **prestazioni di livello superiore** in termini di accuratezza e precisione.

Abbiamo scelto di definire una rete neurale sequenziale con due livelli:

- **1° LIVELLO**:
  64 neuroni nascosti, utilizza la funzione di attivazione **ReLU**;
- **2° LIVELLO**:
  32 neuroni nascosti, utilizza la funzione di attivazione **ReLU**;
- **3° LIVELLO**:
  1 singolo neurone con una funzione di attivazione **sigmoidale,** che produce un valore compreso tra 0 e 1 per la classificazione binaria.

#4.9.2 CREAZIONE MODELLO E ADDESTRAMENTO

STEP #1
Definiamo la funzione `create_model()` che costruisce e restituisce un modello di rete neurale creato con **Keras**:

1. Usiamo il modello sequenziale, che permette di impilare i livelli uno dopo l'altro.
2. Aggiungiamo 2 **fully connected layer** rispettivamente di 64 e di 30 unità nascoste che usano la **ReLU - Rectified Linear Unit** come funzione di attivazione dei neuroni.
3. Aggiungiamo un **fully connected layer** di una sola unità che usa la **sigmoid** come funzione di attivazione dei neuroni per comprimere l'output in un intervallo tra 0 e 1.
4. La funzione termina con la compilazione del modello, utilizzando la funzione `binary_crossentropy`, che è appropriata per problemi di classificazione binaria e l'ottimizzatore `adam` che viene utilizzato per addestrare la rete, ed è noto per la sua efficacia in una varietà di scenari.

```python
# Costruisco il modello della rete neurale
def create_model():
    model = keras.Sequential([
        keras.layers.Input(shape=(X_train_scaled.shape[1],)),
        keras.layers.Dense(64, activation='relu'),
        keras.layers.Dense(32, activation='relu'),
        keras.layers.Dense(1, activation='sigmoid')
    ])

    # Compilazione del modello
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    return model
```

STEP #2

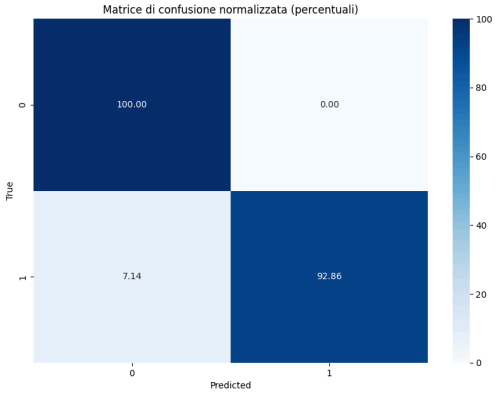
Si crea quindi un'istanza del modello e inizia l'addestramento sui dati.
I parametri in input sono:

- **`epochs`**:
  Specifica quante volte l'intero TrainingSet verrà utilizzato per aggiornare i pesi del modello.
  - Abbiamo scelto di addestrare il modello per 30 epochs, dato che maggiori epoche potrebbero consentire al modello di adattarsi meglio ai dati di addestramento, ma potrebbero anche portare all'**overfitting**.
- **`batch_size`**:
  Definisce la dimensione del batch, ovvero quante istanze di addestramento vengono utilizzate prima di aggiornare i pesi del modello.
  - Abbiamo scelto di utilizzare un batch di dimensione 64, perché rappresenta un **compromesso** tra l'accuratezza del modello e l'efficienza di addestramento. Potrebbe fornire un'adeguata quantità di dati per apprendere modelli complessi senza richiedere troppo tempo computazionale.

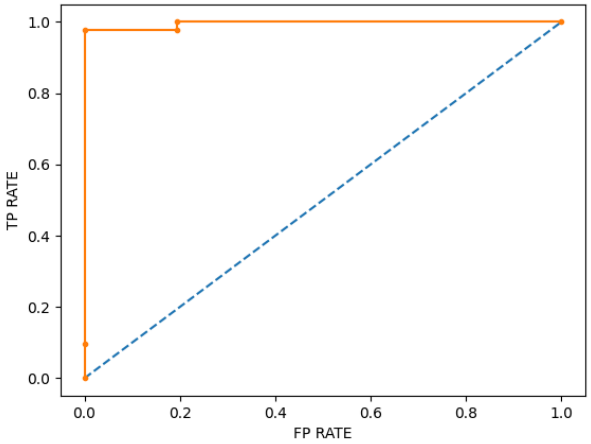
```python
# Creazione istanza del modello e addestramento
model1 = create_model()
model1.fit(X_train_scaled, y_train, epochs=30, batch_size=64)
```

#4.9.3 VALUTAZIONE FINALE

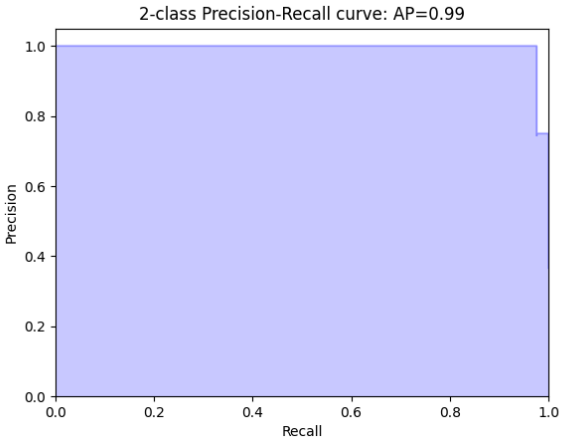
MATRICE DI CONFUSIONE
Il grafico indica un'elevata capacità del modello nel riconoscere sia i positivi che i negativi.
L'F1-score risulta molto alto, evidenziando ancora di più l'efficacia del modello.



ROC CURVE
Il grafico ROC mostra una più che elevata capacità di riconoscere i positivi.
L'AUC conferma la quasi perfezione del modello.

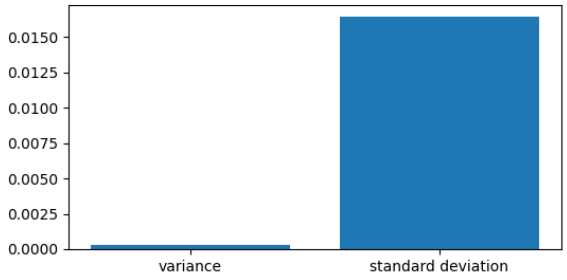


PRECISION-RECALL CURVE
Anche in questo grafico come nel precedente è più che evidente l'elevata capacità del modello.
L'AP di 0.99 va solamente a confermare ciò che è già visibile dal grafico.



BAR CHART DI VARIANZA E DEVIAZIONE STANDARD

Il Bar Chart di Varianza e Deviazione Standard mostra una varianza di circa 0.001 e una deviazione standard di 0.015, indicando che che le prestazioni del modello sono costanti e affidabili.




```
Classification report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        72
           1       1.00      0.95      0.98        42

    accuracy                           0.98       114
   macro avg       0.99      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114


Confusion matrix:
 [[72  0]
 [ 2 40]]
```

```
cv_scores mean:0.9626373648643494

cv_score variance:0.00046371205678497063

cv_score standard deviation:0.02153397447720626
```

```
AUC: 0.998

f1 score:  0.975609756097561
```

#4.10 CONCLUSIONI

Nella seguente tabella sono elencate le metriche usate per confrontare i modelli.

	ACCURATEZZA	VARIANZA	DEV. STANDARD	F1-SCORE	AVERAGE-PRECISION	AUC
K-NN	0.95	0.0005	0.0237	0.9383	0.96	0.982
RANDOM FOREST	0.97	0.0006	0.0247	0.9629	0.98	0.994
SVM	0.97	0.0005	0.0217	0.9677	0.98	0.999
NEURAL NETWORK	0.99	0.0002	0.0161	0.9879	0.99	0.998

CONCLUSIONI DALLA TABELLA

- **Neural Network è il miglior modello**
  - Ha la **maggiore accuratezza (0.99)** e il **miglior F1-score (0.9879)**, il che indica un'eccellente capacità di classificazione.
  - Ha anche la **varianza più bassa (0.0002)** e la **deviazione standard più piccola (0.0161)**, quindi è più stabile rispetto agli altri modelli.
- **SVM e Random Forest sono molto validi**
  - Entrambi hanno un'**accuratezza di 0.97**, con SVM leggermente migliore nell'**F1-score (0.9677 vs 0.9629)**.
  - SVM ha anche il miglior **AUC (0.999)**, indicando un'eccellente separabilità tra le classi.
- **K-NN è il meno performante**
  - Ha la **minor accuratezza (0.95)** e il **minore F1-score (0.9383)**.
  - Maggiore **varianza e deviazione standard**, quindi è meno stabile rispetto agli altri modelli.

CONCLUSIONE GENERALE

Considerando l'importanza di una diagnosi accurata nel contesto medico, abbiamo quindi deciso di adottare il modello di **Neural Network** per la classificazione dei tumori, garantendo così un'analisi più precisa e affidabile per supportare il processo diagnostico.

#5 APPRENDIMENTO NON SUPERVISIONATO: CLUSTERING

Questa tecnica ci consente di identificare naturali raggruppamenti nei dati, anche senza etichette di classe, e ciò potrebbe rivelare relazioni sottostanti tra le risposte al questionario e fornire una prospettiva diversa sulla distribuzione dei dati.

Il clustering può essere di due tipi:

- **Hard Clustering**:
  Prevede l'assegnazione statica di ciascun esempio a una classe di appartenenza.
- **Soft Clustering**:
  Utilizza distribuzioni di probabilità per assegnare le classi associate a ciascun esempio.

#5.1 DECISIONI PROGETTUALI

Questa tecnica è stata scelta per raggruppare i pazienti in categorie basate sulle caratteristiche presenti nel file `‘CancerData.csv'`.
L'obiettivo è individuare dei **cluster**, ovvero gruppi di esempi simili a **centroidi** calcolati in modo automatico.
L'utilizzo di questa tecnica nel progetto mira a creare una nuova colonna da aggiungere alle features relative ai dati di ciascun lavoratore, in un nuovo file chiamato `‘CancerData-clusters.csv'`.
Questo potrebbe aprire nuove prospettive sulla varietà delle risposte e potenzialmente portare a nuove intuizioni sul diabete e contribuire a una diagnosi più accurata e comprensiva.

Nel nostro caso, abbiamo scelto di utilizzare una tecnica di **Hard Clustering**, in particolare il **k-means**, implementato tramite la libreria **Scikit-learn**.

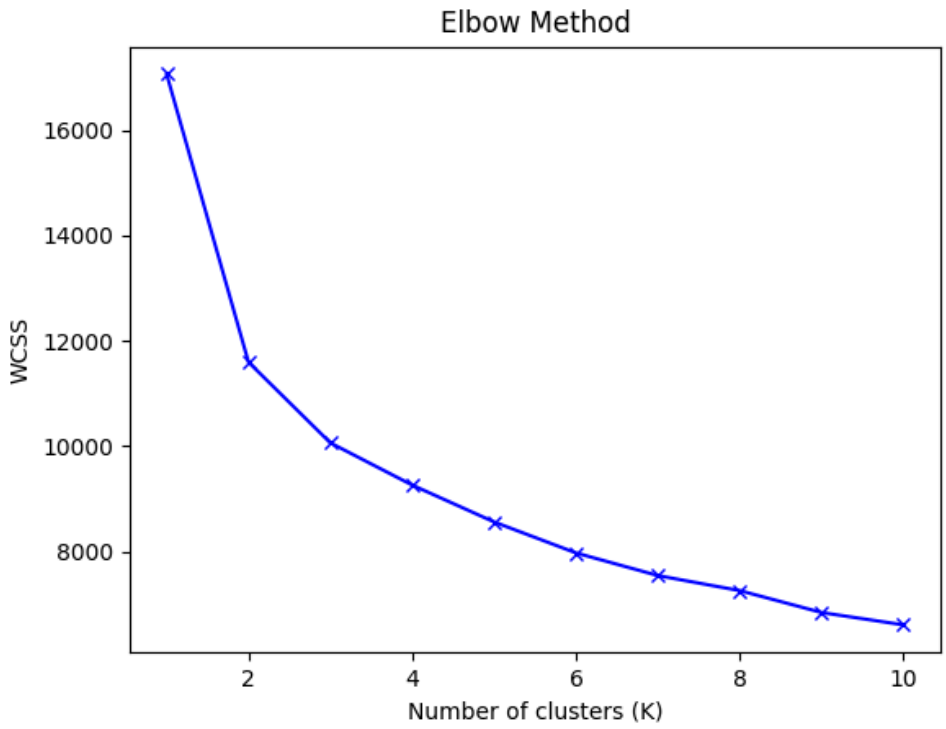
#5.2 K-MEANS

Il **K-Means** è un algoritmo di clustering che suddivide un insieme di dati in **K cluster**, con K valore predefinito, in modo che gli oggetti all'interno dello stesso cluster siano simili tra loro e diversi da quelli di altri cluster e la distanza tra i punti e il centroide del cluster sia la minima possibile.

STEP #1
Abbiamo dovuto, per prima cosa, determinare il numero ottimale di cluster.
A tal scopo, abbiamo utilizzato il **Metodo del Gomito** (Elbow Method):

- Sull'asse **Y** rappresentiamo il **WCSS - Within-Cluster Sum of Squares**, che misura la compattezza dei cluster.
- Sull'asse **X** rappresentiamo il numero di cluster **K**.
    - Il punto ottimale viene scelto osservando dove il WCSS **smette di diminuire significativamente**, formando un angolo simile a un "gomito".

```
wcss = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=40)
    kmeans.fit(dataset)
    wcss.append(kmeans.inertia_)
```



STEP #2
Dall'analisi precedente, il valore di **K ottimale è risultato essere 2**.
Addestriamo quindi il nostro modello sui 2 cluster.

```
kmeans_final = KMeans(n_clusters=2, n_init=10, random_state=42)
kmeans_final.fit(X_scaled)
```

I parametri scelti per la configurazione del modello indicano:

- `n_clusters`:
  Il numero dei cluster in cui dividere i dati del DataSet.
  Nel nostro caso avrà valore pari a 2.
- `n_init`:
  Specifica il numero di volte che l'algoritmo sarà eseguito con diverse inizializzazioni casuali dei centroidi.
    - Un valore maggiore aumenta le probabilità di ottenere una soluzione di migliore qualità a scapito di un maggior tempo di calcolo.

  Abbiamo scelto un valore non troppo alto, ma sufficiente per massimizzare la qualità dell'output, quindi 10.
- `random_state`:
  Questo parametro controlla la riproducibilità dei dati.
    - Fissandolo ad un valore alto, il modello fornirà gli stessi risultati quando viene addestrato con gli stessi dati.

  Abbiamo fissato il suo valore a 42.

#5.3 VALUTAZIONE FINALE DEL MODELLO

Sono state valutate le prestazioni del modello utilizzando due metriche:

- **WCSS (Within-Cluster Sum of Squares)**:
  Calcola la somma dei quadrati delle distanze di ogni punto rispetto al proprio centroide.
  - Un valore più basso di WCSS indica una maggiore coesione all'interno dei cluster.
- **Silhouette Score**:
  Valuta quanto un punto sia vicino al proprio cluster rispetto ai cluster vicini.
  - **Valori vicini a 1**: Cluster ben separati.
  - **Valori vicini a 0**: Cluster che si sovrappongono.
  - **Valori negativi**: Assegnazioni errate dei punti ai cluster.

	WITHIN-CLUSTER SUM OF SQUARES	SILHOUETTE SCORE
K-MEANS	11595.53	0.3434

CONCLUSIONE GENERALE

Con uno Silhouette Score pari a 0.3434 è possibile affermare che il clustering è **accettabile**, ma non perfettamente separato.

Il risultato ottimale sarebbe stato ottenere un valore superiore a 0.5, che avrebbe indicato una separazione più netta. Da ciò abbiamo dedotto che potrebbero esserci sovrapposizioni tra i cluster e quindi deduciamo che il dataset **non è naturalmente divisibile in due gruppi ben distinti.** Questa sovrapposizione può derivare da dati rumorosi e ciò potrebbe aver creato difficoltà a creare un confine di decisione chiaro tra i gruppi.

#6 CONCLUSIONE

In questo progetto, abbiamo voluto esplorare il tema sensibile del cancro da diverse prospettive, utilizzando le tecniche di intelligenza artificiale per analizzare e classificare i dati relativi a questa condizione.
Abbiamo raggiunto l'obiettivo prefissato, di sviluppare un utile strumento ausiliario e di supporto per diagnosticare la possibilità di avere un tumore benigno o maligno attraverso l'uso dell'AI.

#6.1 POSSIBILI SVILUPPI

Questo progetto potrebbe avere una serie di sviluppi futuri per migliorare la sua efficacia e l'applicazione pratica delle scoperte.
Alcune possibili direzioni per lo sviluppo futuro possono essere:

- **L'espansione del DataSet**:
  Banalmente raccogliere dati aggiuntivi e aumentare le dimensioni del DataSet potrebbe migliorare ulteriormente le prestazioni dei modelli.
  Un DataSet più grande può fornire maggiore variabilità nei dati e consentire una migliore generalizzazione.
- **L'integrazione con l'interpretazione medica**:
  Ovvero collegare le scoperte dei modelli di apprendimento supervisionato con specialisti in ambito medico che potrebbero aiutarci ad indirizzare meglio il nostro lavoro.