Relazione progetto statistica

#statistica #progetto-statistica

Progetto di statistica - Simone Samoggia 970758

Selezione del dataset

Il dataset utilizzato è Obesity Prediction

Pulizia del dataset

Caricamento e Pulizia Iniziale

Inizialmente, ho caricato i dati dal file "obesity_data.csv".
Ho rimosso le righe che presentavano valori mancanti usando `dropna()`, assicurandomi così di non compromettere l'integrità dei dati durante le analisi successive.

Successivamente, ho eseguito una verifica per individuare e rimuovere eventuali duplicati con `drop_duplicates()`, garantendo che ogni riga nel dataset fosse unico.

Rimozione dei Valori Fuori Scala

Ho eseguito un'operazione di filtraggio per eliminare le righe con valori non plausibili nelle colonne numeriche come `Height`, `Weight`, `BMI` e `Age`.
Questo include la rimozione di valori negativi o nulli che avrebbero potuto compromettere l'accuratezza delle analisi.

Controllo e Rimozione dei Valori Estremi

Per migliorare la coerenza dei dati, ho applicato un criterio basato sulla deviazione standard per ogni variabile numerica (`Age`, `Height`, `Weight`, `BMI`). Questo ha implicato la rimozione di valori che differivano significativamente dalla media del dataset, contribuendo così a ridurre l'effetto di eventuali outlier che potrebbero distorcere le analisi statistiche.

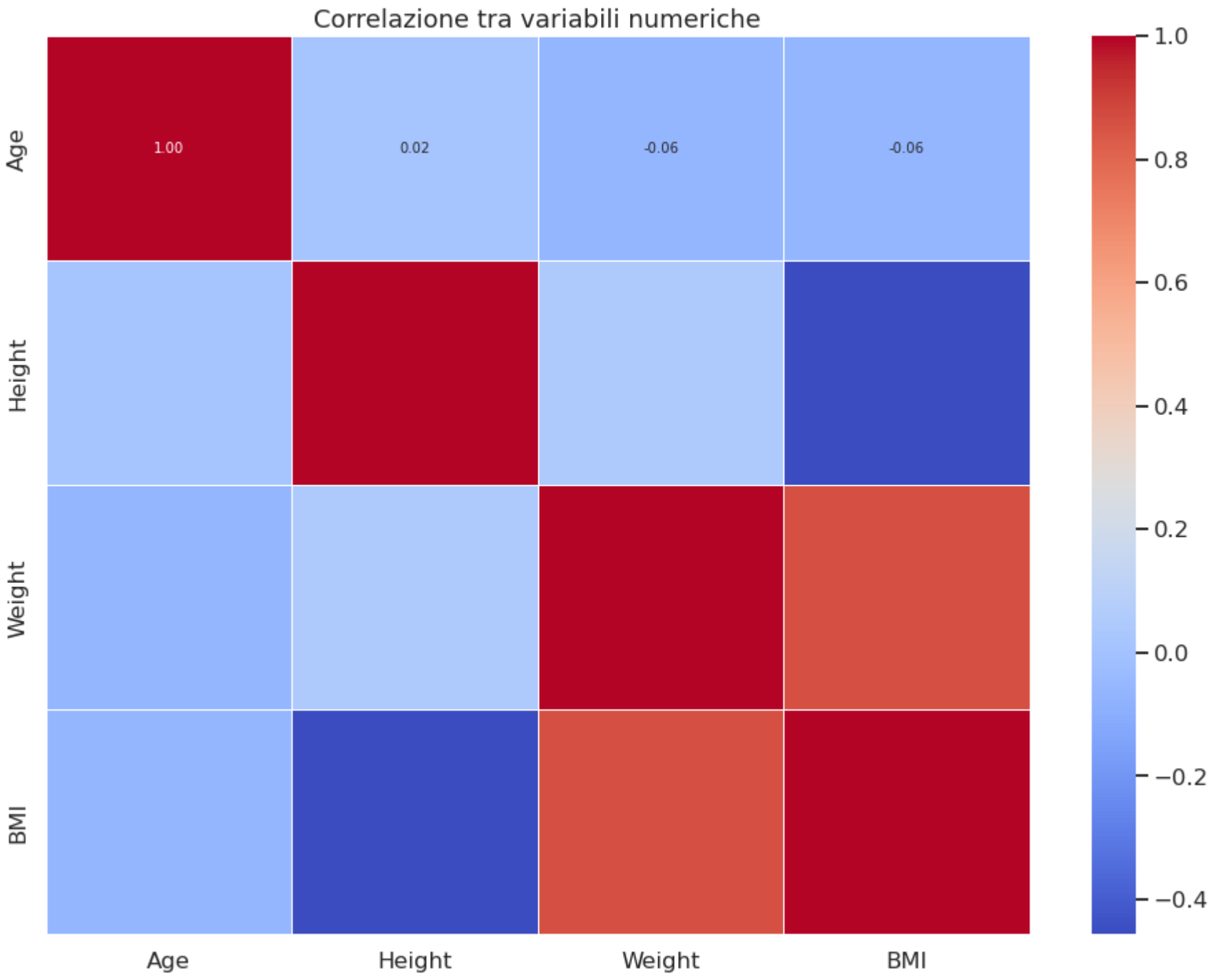
Conversione delle Variabili Categoriali

Ho gestito la variabile `Gender` convertendola da categorica a numerica mediante l'assegnazione di codici numerici ai generi (`0` per maschio e `1` per femmina, ad esempio), facilitando l'analisi dei dati demografici.
Ho anche normalizzato la variabile `ObesityCategory`, che indicava la categoria di obesità di ciascun individuo (`Underweight`, `Normal weight`, `Overweight`, `Obese`), mappandola a una nuova colonna `Cat Numerico` con valori numerici corrispondenti (`0` a `3`). Questo approccio ha reso più semplice l'uso di questa informazione nelle analisi quantitative.

EDA

Matrice di correlazione

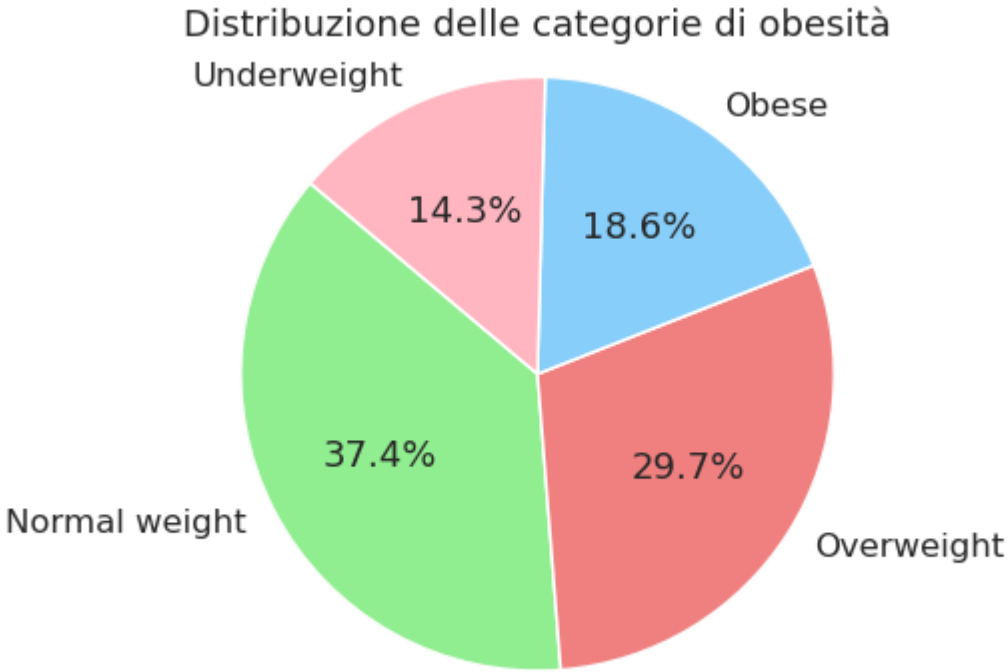
La matrice di correlazione mostra una forte correlazione tra altezza e BMI e tra BMI e peso


Correlazione tra variabili numeriche
Age
Height
Weight
BMI
1.00
0.02
-0.06
-0.06
Age
Height
Weight
BMI
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

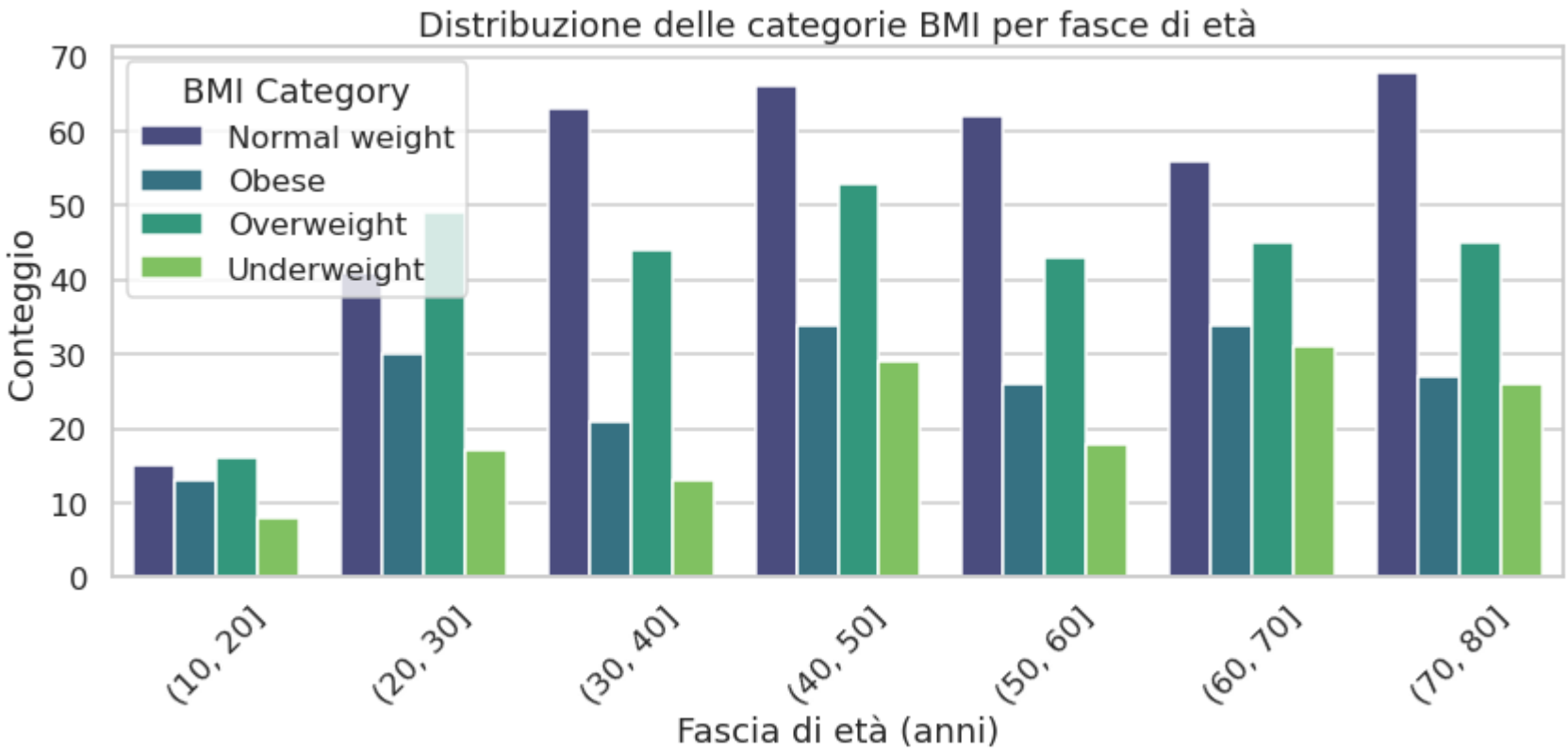
Mentre non mostra altre forti correlazioni tra le altre variabili numeriche.

Studio dei dati

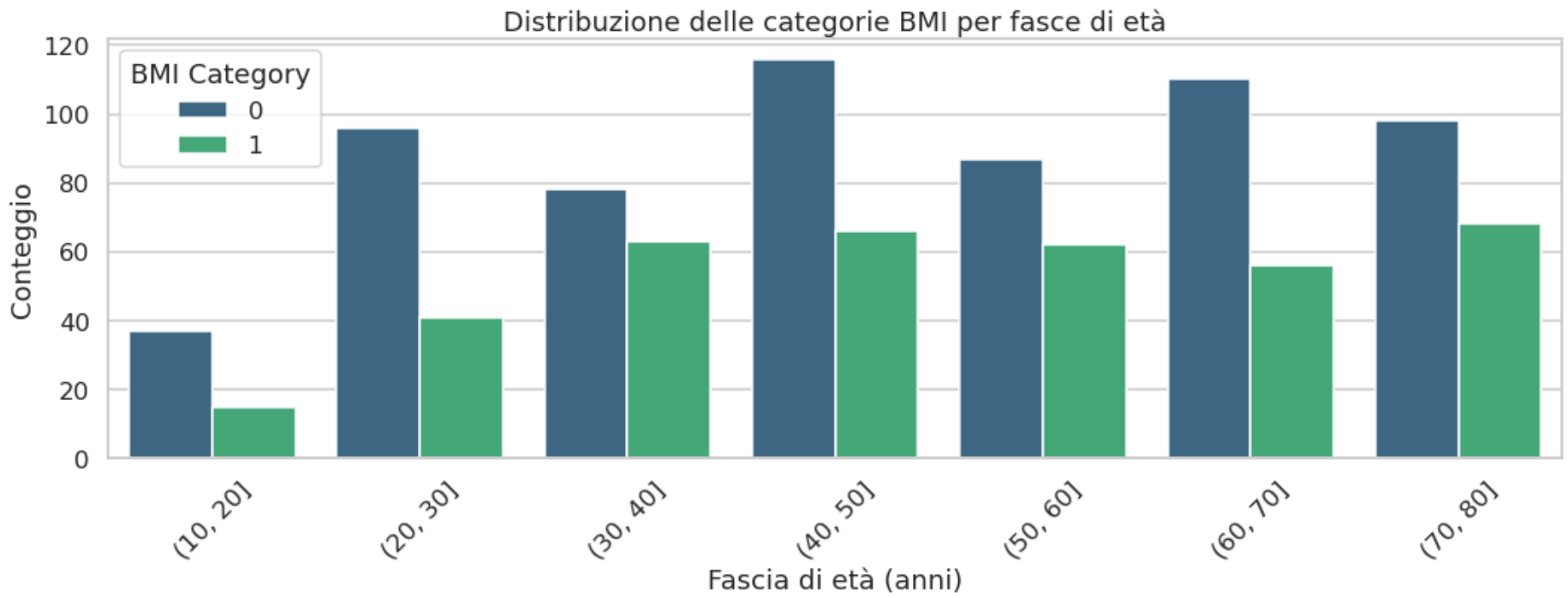
Nel complesso la popolazione è per un 60% non in normopeso



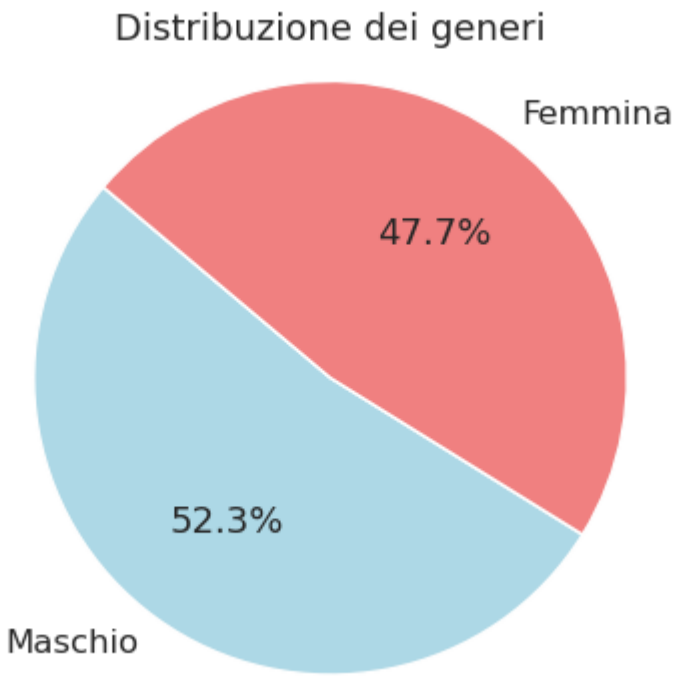
La suddivisione per fasce d'età mostra come nel dataset i disordini alimentari sono abbastanza equamente distribuiti per fasce d'età



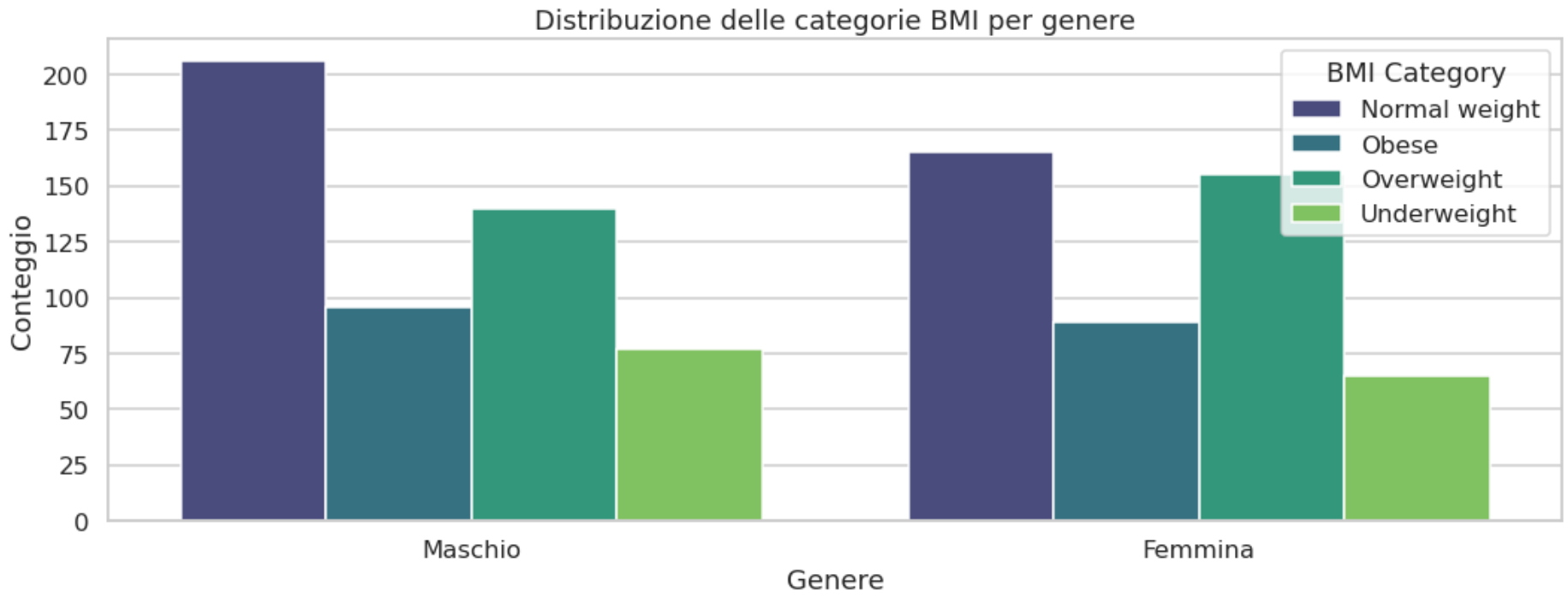
Più in particolare vediamo che le persone che hanno un'età compresa tra i 30 e i 60 anni sono quelle che hanno una maggior incidenza di disordini alimentari.



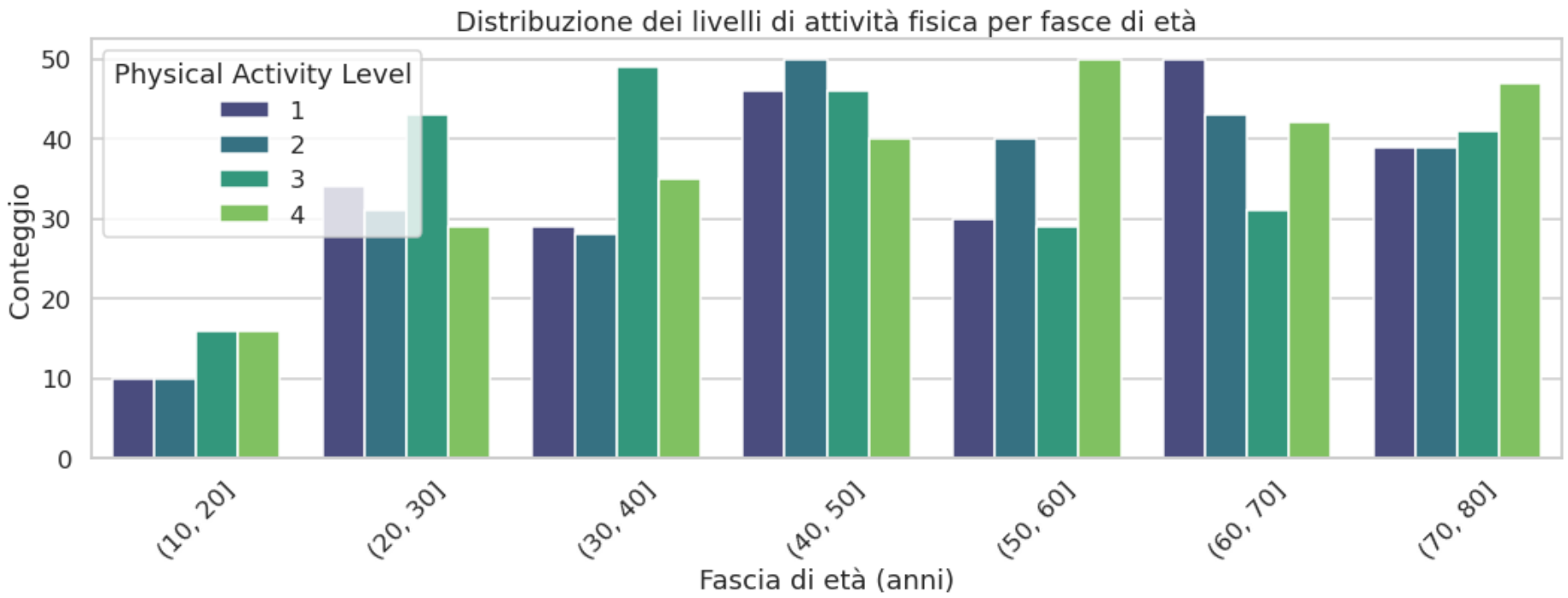
La popolazione è suddivisa abbastanza equamente tra maschi e femmine



Non sembra esserci una grande incidenza del genere sul tipo di fisico.



Non sembra esserci relazione tra i livelli di attività e l'età, anche se possiamo notare come la fascia 40-50 e la fascia 70-80 sono quelle più equamente distribuite

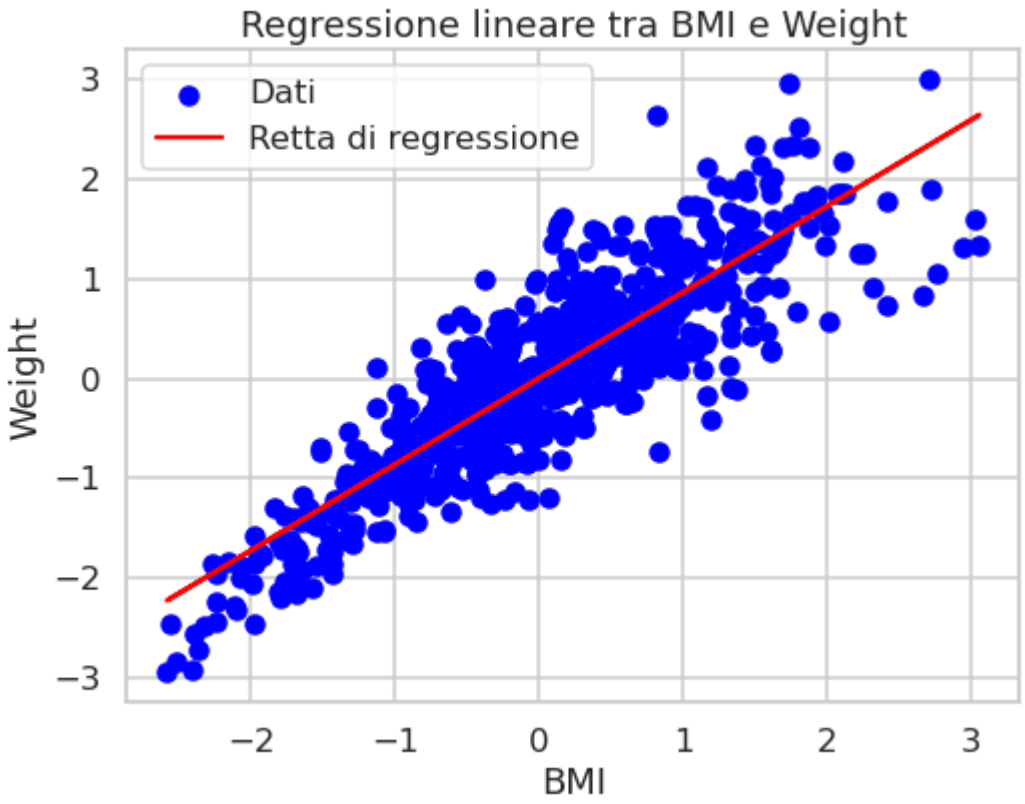


Splitting e training

Dopo aver standardizzato i valori delle colonne numeriche, ho suddiviso il dataframe in due parti con una proporzione 65/35, assegnando il 65% al train set e il restante l'ho ulteriormente suddiviso 50/50 tra test set e validation set.

Regressione lineare

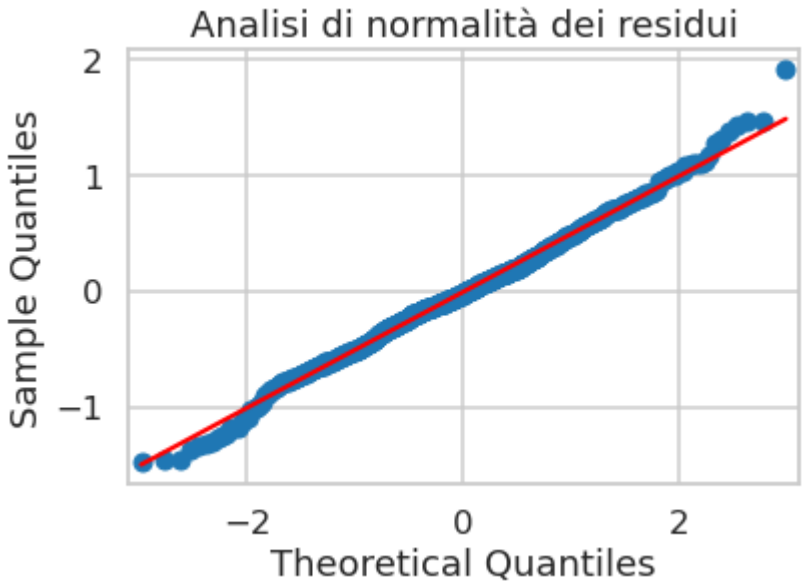
Grazie alla matrice di correlazione, si può notare una forte correlazione tra peso e BMI, la regressione lineare verrà quindi fatta su queste due variabili.

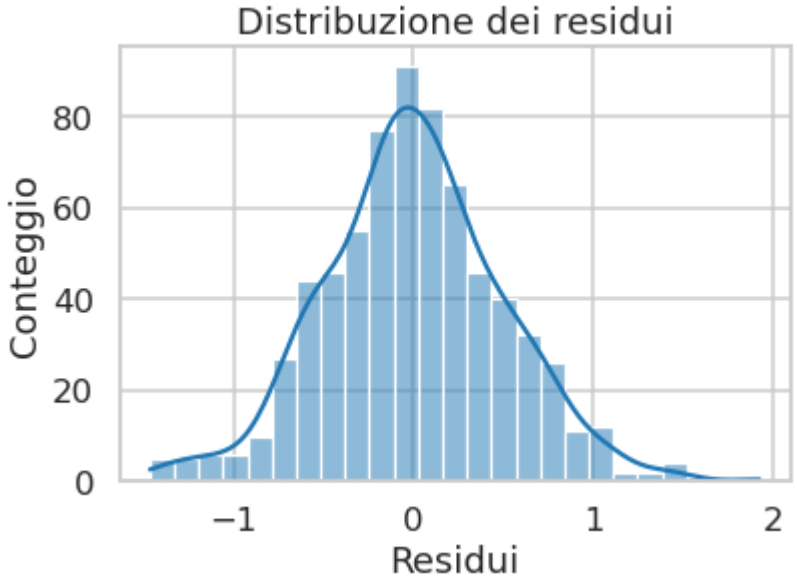



```
Test di Shapiro-Wilk per normalità dei residui: ShapiroResult(statistic=0.9957563845981052, pvalue=0.055435922276715045)
Variabili: BMI e Weight
Coefficiente di regressione: 0.8613
Intercetta: 0.0037
Coefficiente di determinazione (r^2): 0.7517
Mean Squared Error (MSE): 0.2497
```

Normalità dei residui

Il test di Shapiro-Wilk mostra un pvalue intorno a 0.05 e la distribuzione dei residui sembra essere normale.





Modelli e Hyperparameter tuning

Regressione Logistica

Ho eseguito un'analisi utilizzando modelli di machine learning per la classificazione di dati. Inizialmente, ho definito i parametri da testare per la Regressione Logistica, variando i valori di `C` e `max_iter`. Successivamente, ho utilizzato `GridSearchCV` per eseguire una ricerca dei migliori Hyperparameter tramite cross-validation su un training set. Il modello è stato valutato utilizzando MSE.

Dopo aver completato il Grid Search, ho identificato i migliori parametri e l'accuratezza associata al modello ottimale. Ho addestrato il modello finale utilizzando i migliori parametri trovati e ho valutato le sue prestazioni su un set di validazione e un set di test separati, utilizzando le metriche di tasso di errore di classificazione (Misclassification Rate).

Infine, ho riportato i risultati ottenuti nella valutazione della Regressione Logistica con i migliori parametri, evidenziando i tassi di errore di classificazione per i set di validazione e di test, confermando l'efficacia del modello ottimale identificato tramite il processo di ricerca dei migliori Hyperparameter.

SVM (linear, RBF, poly)

Ho svolto un procedimento molto simile per SVM, ma con Grid Search ho assegnato anche alcuni kernel e parametri particolari, come il degree per poly.

Scelta del modello

Dopo aver confrontato i modelli tra loro, scopro che il modello migliore nella mia condizione è *Regressione Logistica*, quindi l'analisi verrà fatta su di esso.

Studio statistico sui risultati della valutazione

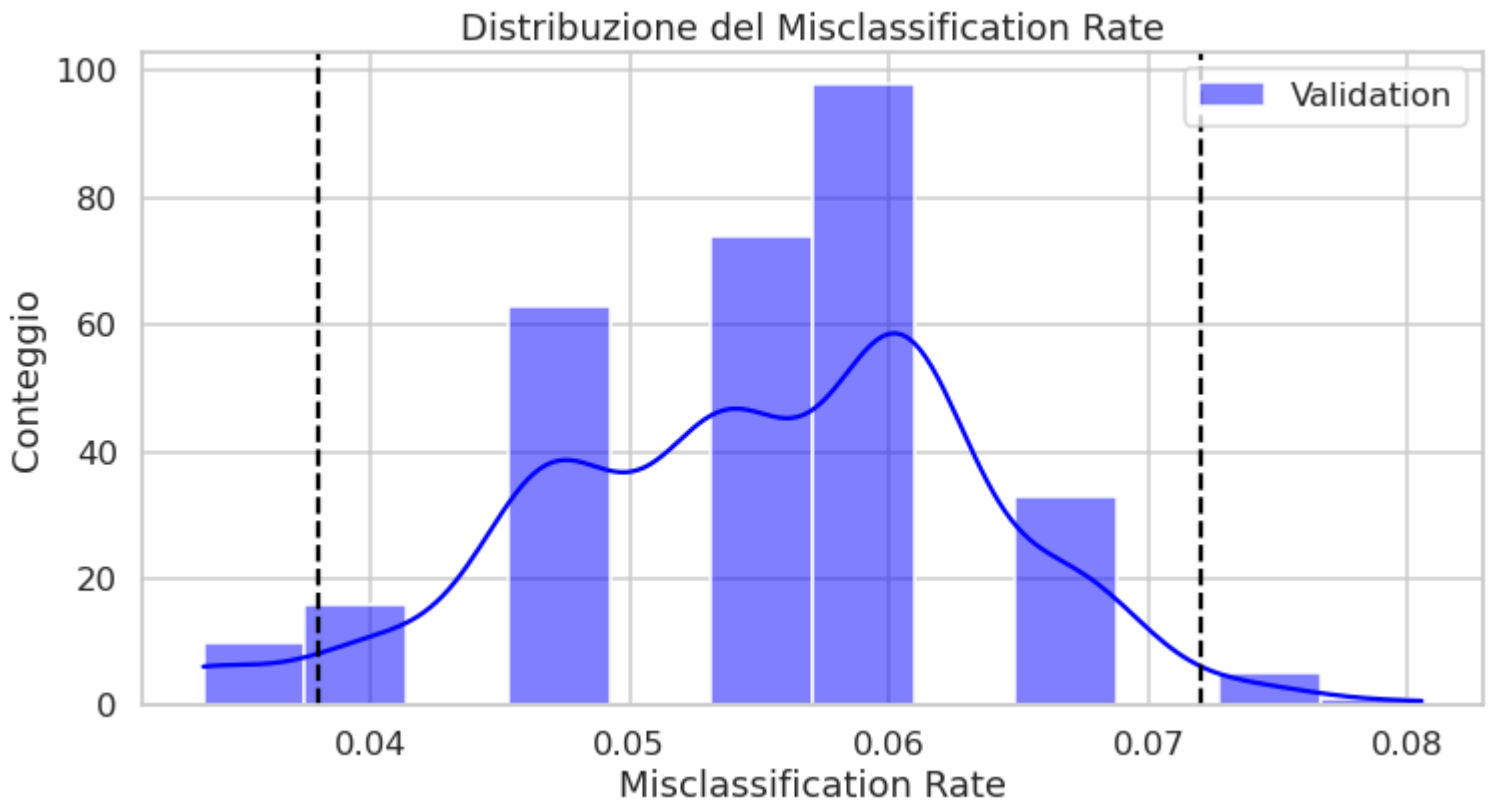
Utilizzando il miglior modello delle fasi precedenti, lo addestro e lo valuto dopo aver aggiunto arbitrariamente del rumore (riproducibile poiché il seed è l'indice dell'iterazione corrente), ma in questo modo il rumore, essendo diverso per ogni iterazione, sarà unico, in questo modo miglioro il test sulla robustezza e l'affidabilità del modello.

Metriche

Dopo aver iterato, prendo il valore dell'errore di ogni iterazione e ne faccio un'analisi con la statistica descrittiva, ovvero ne calcolo **media**, **mediana** e **deviazione standard**.

```
================================================================
Metrica                    Valore
----------------------------------------------------------------
Media                      0.0550
Mediana                    0.0537
Deviazione Standard        0.0086
----------------------------------------------------------------
Intervalli di confidenza al 95%:
MR Validation: (0.037992778284463 46, 0.07198485035088337)
================================================================
```

Grafici MR




```
########################################
#   PROGETTO DI STATISTICA NUMERICA    #
#   DATASET: Obesity Prediction        #
#                                      #
#       Simone Samoggia 970758         #
#                                      #
########################################


import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```python
import scipy
import scipy.stats
import seaborn as sns
import sklearn.metrics as metrics
import statsmodels.api as sm
from scipy.stats import shapiro
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

# Sito del dataset
# https://www.kaggle.com/datasets/mrsimple07/obesity-prediction
# struttura del dataset
# Age,Gender,Height,Weight,BMI,PhysicalActivityLevel,ObesityCategory


## 2. Caricamento e pulizia del dataset
df = pd.read_csv("obesity_data.csv")
df = df.dropna()
df = df.drop_duplicates()


# rimuovi i valori fuori scala
df = df[df["Height"] > 0]
df = df[df["Weight"] > 0]
df = df[df["BMI"] > 0]
df = df[df["PhysicalActivityLevel"] > 0]
```

```
df = df[df["Age"] > 0]

# rimuovi i valori nulli
df = df.dropna()

# rimuovi i valori duplicati
df = df.drop_duplicates()

threshold_param = 3

# Check delle colonne e dei loro tipi
numeric_columns = ["Age", "Height", "Weight", "BMI"]

# rimuovi i valori irrealistici, troppo lontani dalle rispettive medie
for column in df.columns:
    if column in numeric_columns:
        df = df[
            (df[column] > df[column].mean() - threshold_param * df[column].std())
            & (df[column] < df[column].mean() + threshold_param * df[column].std())
        ]
df["Gender"] = (df["Gender"].map({"Male": 0, "Female": 1})).astype("category")

df["Cat Numerico"] = (
    df["ObesityCategory"]
    .map({"Underweight": 0, "Normal weight": 1, "Overweight": 2, "Obese": 3})
    .astype("category")
)

# # stampa csv pulito
```

```
# df.to_csv("obesity_cleaned.csv", index=False)


## 3. EDA
sns.pairplot(df[numeric_columns])
plt.show()

correlation_matrix = df[numeric_columns].corr()

plt.figure(figsize=(16, 12))
sns.heatmap(
    correlation_matrix,
    annot=True,
    cmap="coolwarm",
    fmt=".2f",
    annot_kws={"size": 10},
    linewidths=0.5,
)
plt.title("Correlazione tra variabili numeriche")

sns.set_style("whitegrid")
sns.set_context("talk")

threshold = 0.65
significant_correlations = (
    correlation_matrix.where(
        np.triu(np.ones(correlation_matrix.shape), k=1).astype(bool)
    )
    .stack()
```

```
    .reset_index()
    .rename(columns={0: "correlation"})
    .query("abs(correlation) > @threshold")
)

plt.figure(figsize=(12, 6))


# mostra la distribuzione delle categorie di obesità in relazione all'età
df["Age Group"] = pd.cut(df["Age"], bins=range(10, 90, 10))


# Convertire le fasce di età in numeri interi
age_group_mapping = {
    interval: i for i, interval in enumerate(df["Age Group"].cat.categories)
}

df["Age Group Numeric"] = df["Age Group"].map(age_group_mapping)

sns.countplot(data=df, x="Age Group", hue="ObesityCategory", palette="viridis")
plt.title("Distribuzione delle categorie BMI per fasce di età")
plt.xlabel("Fascia di età (anni)")
plt.ylabel("Conteggio")
plt.xticks(rotation=45)
plt.legend(title="BMI Category", loc="upper left")
plt.tight_layout()


# mostra la distribuzione delle categorie di obesità in relazione al genere in un'altra
```

```
figura
plt.figure(figsize=(15, 6))
sns.countplot(data=df, x="Gender", hue="ObesityCategory", palette="viridis")
plt.title("Distribuzione delle categorie BMI per genere")
plt.xlabel("Genere")
plt.ylabel("Conteggio")
plt.xticks(ticks=[0, 1], labels=["Maschio", "Femmina"])
plt.legend(title="BMI Category", loc="upper right")
plt.tight_layout()

# mostra booleano per la presenza di obesità, overweight e underweight
# se non sei in normopeso, allora sei in una delle altre categorie
df["bool category"] = df["ObesityCategory"].map(
    {"Underweight": 0, "Normal weight": 1, "Overweight": 0, "Obese": 0}
)

plt.figure(figsize=(15, 6))
sns.countplot(data=df, x="Age Group", hue="bool category", palette="viridis")
plt.title("Distribuzione delle categorie BMI per fasce di età")
plt.xlabel("Fascia di età (anni)")
plt.ylabel("Conteggio")
plt.xticks(rotation=45)
plt.legend(title="BMI Category", loc="upper left")
plt.tight_layout()

plt.figure(figsize=(15, 6))
sns.countplot(data=df, x="bool category", palette="viridis")
plt.title("Distribuzione delle categorie BMI")
plt.ylabel("Conteggio")
```

```
plt.xticks(ticks=[0, 1], labels=["Non normopeso", "Normopeso"])
plt.legend(title="BMI Category", loc="upper right")
plt.tight_layout()



# Calcolo delle percentuali per Gender
gender_counts = df["Gender"].value_counts(normalize=True) * 100

# Etichette e colori per il grafico a torta
labels = ["Maschio", "Femmina"]
sizes = gender_counts.values
colors = ["lightblue", "lightcoral"]  # Colori per i generi (0 e 1)

plt.figure(figsize=(12, 6))
plt.pie(sizes, labels=labels, colors=colors, autopct="%1.1f%%", startangle=140)
plt.title("Distribuzione dei generi")
plt.axis("equal")  # Garantisce che il grafico sia disegnato come un cerchio
plt.show()

# Analisi della variabile categorica ObesityCategory
obesity_counts = df["ObesityCategory"].value_counts(normalize=True) * 100

# Etichette e colori per il grafico a torta
labels = obesity_counts.index
sizes = obesity_counts.values
colors = [
    "lightgreen",
    "lightcoral",
    "lightskyblue",
```

```
    "lightpink",
]

# Tracciamento del grafico a torta
plt.figure(figsize=(12, 6))
plt.pie(sizes, labels=labels, colors=colors, autopct="%1.1f%%", startangle=140)
plt.title("Distribuzione delle categorie di obesità")
plt.axis("equal")  # Garantisce che il grafico sia disegnato come un cerchio
plt.show()


# quali sono le fasce di età più attive?
plt.figure(figsize=(15, 6))
sns.countplot(data=df, x="Age Group", hue="PhysicalActivityLevel", palette="viridis")
plt.title("Distribuzione dei livelli di attività fisica per fasce di età")
plt.xlabel("Fascia di età (anni)")
plt.ylabel("Conteggio")
plt.xticks(rotation=45)
plt.legend(title="Physical Activity Level", loc="upper left")
plt.tight_layout()


## 4. Splitting e Training
# Standardizzazione delle colonne numeriche
scaler = StandardScaler()
df[numeric_columns] = scaler.fit_transform(df[numeric_columns])

test_size_size = 0.30
# Split iniziale per creare il training set e il test set
```

```
train_data, remaining_data = train_test_split(
    df, test_size=test_size_size, random_state=42
)

# Split per creare il validation set e il test set 50/50
validation_data, test_data = train_test_split(
    remaining_data, test_size=0.5, random_state=42
)

# Verifica delle dimensioni dei set
print("Dimensioni del training set:", train_data.shape)
print("Dimensioni del validation set:", validation_data.shape)
print("Dimensioni del test set:", test_data.shape)

## 5. Regressione Lineare
# Selezionare le variabili numeriche correlate
x_col = "BMI"
y_col = "Weight"

# Creare il modello di classificazione tra le variabili selezionate
X = train_data[[x_col]]
y = train_data[y_col]
# Esegui la regressione lineare
model = LinearRegression()
model.fit(X, y)

# Stima dei coefficienti
coef = model.coef_[0]
intercept = model.intercept_
```

```
# Calcolo del coefficiente di determinazione r^2
y_pred = model.predict(X)
r_squared = r2_score(y, y_pred)

# Calcolo del Mean Squared Error (MSE)
mse = mean_squared_error(y, y_pred)

# Analisi di normalità dei residui con qqplot
residuals = y - y_pred
sm.qqplot(residuals, line="s")
plt.title("Analisi di normalità dei residui")
plt.show()

# plotta la distribuzione dei residui
sns.histplot(residuals, kde=True)
plt.title("Distribuzione dei residui")
plt.xlabel("Residui")
plt.ylabel("Conteggio")
plt.show()

# esegui shapiro per capire la distribuzione dei residui
shapiro_test = shapiro(residuals)
print(f"Test di Shapiro-Wilk per normalità dei residui: {shapiro_test}")

# Grafico dei punti e della retta di regressione
plt.figure(figsize=(8, 6))
plt.scatter(X, y, color="blue", label="Dati")
plt.plot(X, y_pred, color="red", label="Retta di regressione")
```

```
plt.title(f"Regressione lineare tra {x_col} e {y_col}")
plt.xlabel(x_col)
plt.ylabel(y_col)
plt.legend()
plt.show()

# Stampa dei risultati
print("=====================================================")
print(f"Variabili: {x_col} e {y_col}")
print(f"Coefficiente di regressione: {coef:.4f}")
print(f"Intercetta: {intercept:.4f}")
print(f"Coefficiente di determinazione (r^2): {r_squared:.4f}")
print(f"Mean Squared Error (MSE): {mse:.4f}")
print("=====================================================")


## 6. Classificazione
# Funzione per preparare i dati
def prepare_data(train_data, validation_data, target_col, drop_cols):
    X_train = train_data.drop(columns=drop_cols)
    y_train = train_data[target_col]
    X_val = validation_data.drop(columns=drop_cols)
    y_val = validation_data[target_col]
    return X_train, y_train, X_val, y_val


# Funzione per creare e valutare il modello
def evaluate_model(model, X_train, y_train, X_val, y_val):
    model.fit(X_train, y_train)
```

```
    y_val_pred = model.predict(X_val)
    mr_val = 1 - accuracy_score(y_val, y_val_pred)
    return mr_val


# Funzione per stampare i risultati
def print_results(model_name, mr_val):
    print("=======================================================")
    print(f"MR del modello {model_name}: {mr_val:.4f}")
    print("=======================================================")


# Prepara i dati
target_col = "Cat Numerico"
drop_cols = [
    "Age Group",
    "Age Group Numeric",
    "ObesityCategory",
    "Cat Numerico",
    "Gender",
    "PhysicalActivityLevel",
    "Height",
    "bool category",
]
X_train, y_train, X_val, y_val = prepare_data(
    train_data, validation_data, target_col, drop_cols
)

# 6. Addestramento del Modello
```

```python
# 7. Hyperparameter Tuning

# Definizione dei parametri da testare
parameters = {"C": [0.001, 0.01, 0.1, 1, 10], "max_iter": [100, 500, 1000, 5000]}

# Creazione del modello di regressione logistica
logistic_model = LogisticRegression(random_state=42)

# Inizializzazione del Grid Search
grid_search = GridSearchCV(
    estimator=logistic_model, param_grid=parameters, cv=5, scoring="accuracy", verbose=1
)

# Esecuzione del Grid Search per trovare i migliori parametri
grid_search.fit(X_train, y_train)

# Stampare i migliori parametri trovati
print("Migliori parametri:")
print(grid_search.best_params_)

# Stampare la migliore accuratezza trovata
print("Migliore accuratezza:", grid_search.best_score_)

# Addestramento del modello finale con i migliori parametri trovati
best_logistic_model = grid_search.best_estimator_

# Valutazione del modello su validation e test set
mr_val_log = evaluate_model(best_logistic_model, X_train, y_train, X_val, y_val)
```

```
# Stampare i risultati della Regressione Logistica con i migliori parametri
print_results("Regressione Logistica con Grid Search", mr_val_log)

# Definizione dei parametri per il Grid Search
parameters = {
    "kernel": ["linear", "poly", "rbf"],
    "C": [0.001, 0.01, 0.1, 1],
    "gamma": [0.001, 0.01, 0.1, 1],
}

# Creazione del modello SVM
svm_model = SVC(random_state=42)

# Inizializzazione del Grid Search con cross-validation su validation set
grid_search = GridSearchCV(svm_model, parameters, cv=5, scoring="accuracy", verbose=1)

# Esecuzione del Grid Search per trovare i migliori iperparametri
grid_search.fit(X_train, y_train)

# Miglior modello trovato dal Grid Search
best_svm_model = grid_search.best_estimator_

# Valutazione del modello SVM migliore
mr_val_svm = evaluate_model(best_svm_model, X_train, y_train, X_val, y_val)

# Stampare i risultati del modello SVM
print_results("SVM", mr_val_svm)

# Stampa dei migliori iperparametri
```

```
print("Migliori iperparametri trovati dal Grid Search:")
print(grid_search.best_params_)
print("=====================================================")

## confronta i risultati tra i due modelli
print("Risultati finali:")
print("-----------------------------------------------------")
print(f"MR Regressione Logistica: {mr_val_log:.4f}")
print(f"MR SVM: {mr_val_svm:.4f}")
print("=====================================================")

# scegli il modello migliore
if mr_val_log < mr_val_svm:
    print("Il modello migliore è la Regressione Logistica.")
    best_model = best_logistic_model

else:
    print("Il modello migliore è SVM.")
    best_model = best_svm_model

## 8. Valutazione della Performance
# Una volta definito un modello, bisogna valutarne la performance.

# Preparazione dei dati per il test set
X_test = test_data.drop(columns=drop_cols)
y_test = test_data[target_col]

# Stampa dei risultati finali con matrice di confusione
y_test_pred = best_model.predict(X_test)
```

```python
conf_matrix = metrics.confusion_matrix(y_test, y_test_pred)

# Stampa della matrice di confusione
print("Matrice di confusione:")
print(conf_matrix)

# Calcolo dell'accuratezza
accuracy = accuracy_score(y_test, y_test_pred)
print(f"Accuratezza: {accuracy:.4f}")

# Stampa del report di classificazione
classification_report = metrics.classification_report(y_test, y_test_pred)
print("Report di classificazione:")
print(classification_report)



# 9 Studio statistico sui risultati della valutazione

# Ripetizione delle fasi di addestramento e testing con il modello migliore
n_iterations = 300
MR_finale = np.zeros(n_iterations)

for i in range(n_iterations):
    print(f"Iterazione {i + 1}/{n_iterations}")
    np.random.seed(i)

    # metti del rumore nei dati
    noise = np.random.normal(0, 0.1, X_train.shape)
    X_train_con_errore = X_train + noise
```

```
    # Addestramento e valutazione del modello
    best_model.fit(X_train_con_errore, y_train)
    y_val_pred = best_model.predict(X_val)
    MR_finale[i] = 1 - accuracy_score(y_val, y_val_pred)

from scipy import stats

# Calcolare le statistiche desiderate
MR_mean = np.mean(MR_finale)
MR_median = np.median(MR_finale)
MR_std = np.std(MR_finale)
# Intervalli di confidenza
MR_interval = stats.t.interval(0.95, len(MR_finale) - 1, loc=MR_mean, scale=MR_std)
# Stampa dei risultati in una tabella
print("Risultati delle valutazioni statistiche per MR (Validation Set):")
print("=================================================================")
print("Metrica\t\t\t\t\t\tValore")
print("-----------------------------------------------------------------")
print(f"Media\t\t\t\t\t\t{MR_mean:.4f}")
print(f"Mediana\t\t\t\t\t\t{MR_median:.4f}")
print(f"Deviazione Standard\t\t\t{MR_std:.4f}")
print("-----------------------------------------------------------------")
print("Intervalli di confidenza al 95%:")
print(f"MR Validation: {MR_interval}")
print("=================================================================")

# Plot dei risultati
plt.figure(figsize=(12, 6))
```

```
sns.histplot(MR_finale, color="blue", label="Validation", kde=True)
plt.title("Distribuzione del Misclassification Rate")
plt.xlabel("Misclassification Rate")
plt.ylabel("Conteggio")
plt.legend()
# Aggiungi intervallo di confidenza al plot
plt.axvline(
    x=MR_interval[0],
    color="k",
    linestyle="--",
    label="Intervallo di confidenza (95%)",
)
plt.axvline(x=MR_interval[1], color="k", linestyle="--")
plt.show()
```