Programma seconda 3

May 10, 2024

Indice:

1 Puliamo il dataset

[5]:
```
import sys
import pandas as pd
import numpy as np
import sklearn
import matplotlib

print('Python: {}'.format(sys.version))
print('Pandas: {}'.format(pd.__version__))
print('Numpy: {}'.format(np.__version__))
print('Sklearn: {}'.format(sklearn.__version__))
print('Matplotlib: {}'.format(matplotlib.__version__))
```

```
Python: 3.11.9 (tags/v3.11.9:de54cf5, Apr  2 2024, 10:12:12) [MSC v.1938 64 bit
(AMD64)]
Pandas: 2.2.1
Numpy: 1.26.4
Sklearn: 1.4.1.post1
Matplotlib: 3.8.4
```

[12]:
```
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
import seaborn as sns
```

[11]:
```
cleveland = pd.read_csv(r'C:\Users\david\Documents\heart.csv')
```

[13]:
```
print( 'Shape of DataFrame: {}'.format(cleveland.shape))
print (cleveland.loc[1])
```

```
Shape of DataFrame: (303, 14)
age          37.0
sex           1.0
cp            2.0
trestbps    130.0
chol        250.0
fbs           0.0
restecg       1.0
thalach     187.0
exang         0.0
oldpeak       3.5
slope         0.0
ca            0.0
thal          2.0
target        1.0
Name: 1, dtype: float64
```

[14]:
```
# print the last twenty or so data points
cleveland.loc[280:]
```

[14]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	\
280	42	1	0	136	315	0	1	125	1	1.8	
281	52	1	0	128	204	1	1	156	1	1.0	
282	59	1	2	126	218	1	1	134	0	2.2	
283	40	1	0	152	223	0	1	181	0	0.0	
284	61	1	0	140	207	0	0	138	1	1.9	
285	46	1	0	140	311	0	1	120	1	1.8	
286	59	1	3	134	204	0	1	162	0	0.8	
287	57	1	1	154	232	0	0	164	0	0.0	
288	57	1	0	110	335	0	1	143	1	3.0	
289	55	0	0	128	205	0	2	130	1	2.0	
290	61	1	0	148	203	0	1	161	0	0.0	
291	58	1	0	114	318	0	2	140	0	4.4	
292	58	0	0	170	225	1	0	146	1	2.8	
293	67	1	2	152	212	0	0	150	0	0.8	
294	44	1	0	120	169	0	1	144	1	2.8	
295	63	1	0	140	187	0	0	144	1	4.0	
296	63	0	0	124	197	0	1	136	1	0.0	
297	59	1	0	164	176	1	0	90	0	1.0	
298	57	0	0	140	241	0	1	123	1	0.2	
299	45	1	3	110	264	0	1	132	0	1.2	
300	68	1	0	144	193	1	1	141	0	3.4	
301	57	1	0	130	131	0	1	115	1	1.2	
302	57	0	1	130	236	0	0	174	0	0.0	

	slope	ca	thal	target
280	1	0	1	0
281	1	0	0	0
282	1	1	1	0
283	2	0	3	0
284	2	1	3	0
285	1	2	3	0
286	2	2	2	0
287	2	1	2	0
288	1	1	3	0
289	1	1	3	0
290	2	1	3	0
291	0	3	1	0
292	1	2	1	0
293	1	0	3	0
294	0	0	1	0
295	2	2	3	0
296	1	0	2	0
297	1	2	1	0

```
298      1   0     3       0
299      1   0     3       0
300      1   2     3       0
301      1   1     3       0
302      1   1     2       0
```

[15]:
```
# remove missing data (indicated with a "?")
data = cleveland[~cleveland.isin(['?'])]
data.loc[280:]
```

[15]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
280	42	1	0	136	315	0	1	125	1	1.8	1	0	1	0
281	52	1	0	128	204	1	1	156	1	1.0	1	0	0	0
282	59	1	2	126	218	1	1	134	0	2.2	1	1	1	0
283	40	1	0	152	223	0	1	181	0	0.0	2	0	3	0
284	61	1	0	140	207	0	0	138	1	1.9	2	1	3	0
285	46	1	0	140	311	0	1	120	1	1.8	1	2	3	0
286	59	1	3	134	204	0	1	162	0	0.8	2	2	2	0
287	57	1	1	154	232	0	0	164	0	0.0	2	1	2	0
288	57	1	0	110	335	0	1	143	1	3.0	1	1	3	0
289	55	0	0	128	205	0	2	130	1	2.0	1	1	3	0
290	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
291	58	1	0	114	318	0	2	140	0	4.4	1	3	1	1
292	58	0	0	170	225	1	0	146	1	2.8	1	2	1	1
293	67	1	2	152	212	0	0	150	0	0.8	2	0	3	1
294	44	1	0	120	169	0	1	144	1	2.8	0	0	1	0
295	63	1	0	140	187	0	0	144	1	4.0	2	2	3	1
296	63	0	0	124	197	0	1	136	1	0.0	1	0	2	1
297	59	1	0	164	176	1	0	90	0	1.0	1	2	1	1
298	57	0	0	140	241	0	1	123	1	0.2	2	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	2	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	2	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	2	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	2	1	3	0

```
291      0   3     1        0
292      1   2     1        0
293      1   0     3        0
294      0   0     1        0
295      2   2     3        0
296      1   0     2        0
297      1   2     1        0
298      1   0     3        0
299      1   0     3        0
300      1   2     3        0
301      1   1     3        0
302      1   1     2        0
```

[16]:
```
# drop rows with NaN values from DataFrame
data = data.dropna(axis=0)
data.loc[280:]
```

[16]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	\
280	42	1	0	136	315	0	1	125	1	1.8	
281	52	1	0	128	204	1	1	156	1	1.0	
282	59	1	2	126	218	1	1	134	0	2.2	
283	40	1	0	152	223	0	1	181	0	0.0	
284	61	1	0	140	207	0	0	138	1	1.9	
285	46	1	0	140	311	0	1	120	1	1.8	
286	59	1	3	134	204	0	1	162	0	0.8	
287	57	1	1	154	232	0	0	164	0	0.0	
288	57	1	0	110	335	0	1	143	1	3.0	
289	55	0	0	128	205	0	2	130	1	2.0	
290	61	1	0	148	203	0	1	161	0	0.0	
291	58	1	0	114	318	0	2	140	0	4.4	
292	58	0	0	170	225	1	0	146	1	2.8	
293	67	1	2	152	212	0	0	150	0	0.8	
294	44	1	0	120	169	0	1	144	1	2.8	
295	63	1	0	140	187	0	0	144	1	4.0	
296	63	0	0	124	197	0	1	136	1	0.0	
297	59	1	0	164	176	1	0	90	0	1.0	
298	57	0	0	140	241	0	1	123	1	0.2	
299	45	1	3	110	264	0	1	132	0	1.2	
300	68	1	0	144	193	1	1	141	0	3.4	
301	57	1	0	130	131	0	1	115	1	1.2	
302	57	0	1	130	236	0	0	174	0	0.0	

	slope	ca	thal	target
280	1	0	1	0
281	1	0	0	0
282	1	1	1	0
283	2	0	3	0

```
284      2   1     3       0
285      1   2     3       0
286      2   2     2       0
287      2   1     2       0
288      1   1     3       0
289      1   1     3       0
290      2   1     3       0
291      0   3     1       0
292      1   2     1       0
293      1   0     3       0
294      0   0     1       0
295      2   2     3       0
296      1   0     2       0
297      1   2     1       0
298      1   0     3       0
299      1   0     3       0
300      1   2     3       0
301      1   1     3       0
302      1   1     2       0
```

[17]:
```
# print the shape and data type of the dataframe
print(data.shape)
print(data.dtypes)
```

```
(303, 14)
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

[18]:
```
# transform data to numeric to enable further analysis
data = data.apply(pd.to_numeric)
data.dtypes
```

[18]:
```
age           int64
sex           int64
cp            int64
```

```
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

[19]:
```
# print data characteristics
data.describe()
```

[19]:

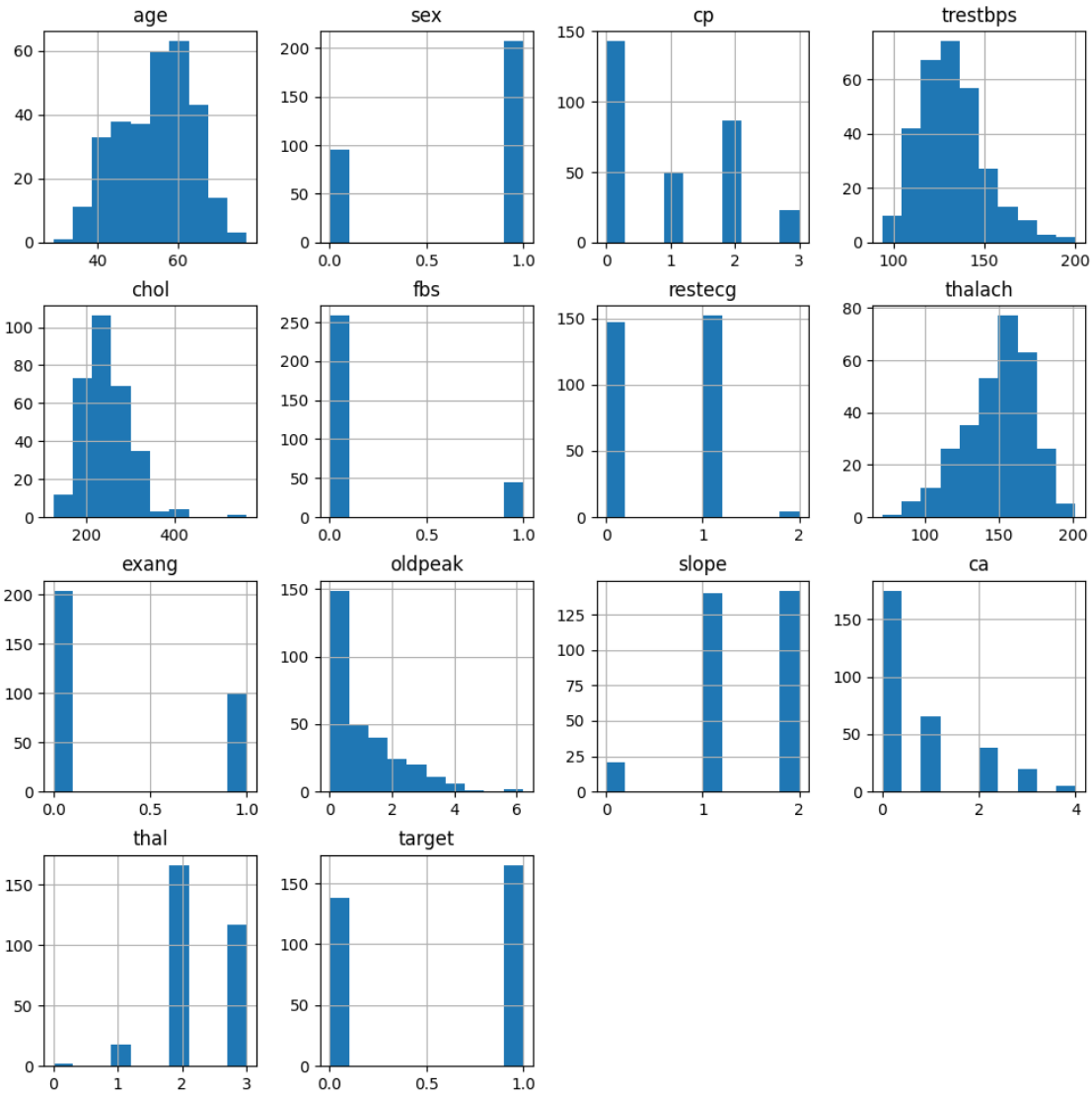
	age	sex	cp	trestbps	chol	fbs	\
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	

	restecg	thalach	exang	oldpeak	slope	ca	\
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	
mean	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	
std	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	
min	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	
50%	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	
75%	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	
max	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	

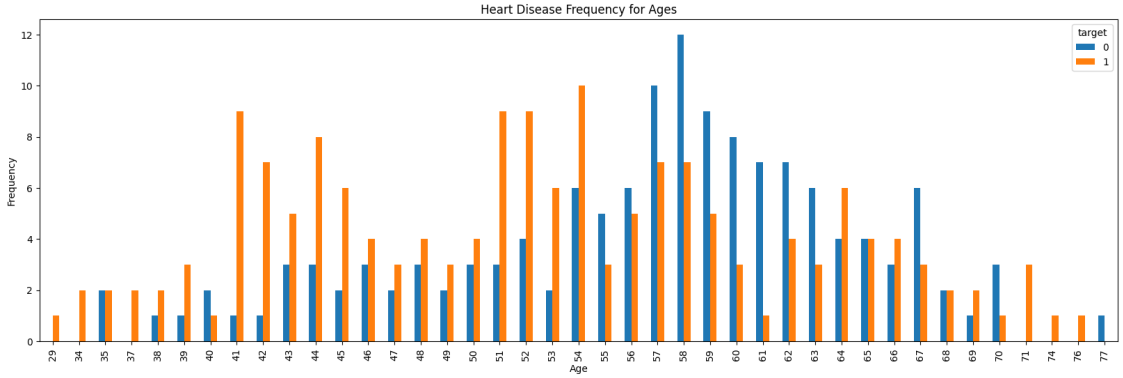
	thal	target
count	303.000000	303.000000
mean	2.313531	0.544554
std	0.612277	0.498835
min	0.000000	0.000000
25%	2.000000	0.000000
50%	2.000000	1.000000
75%	3.000000	1.000000
max	3.000000	1.000000

2 Visualiziamo i grafici

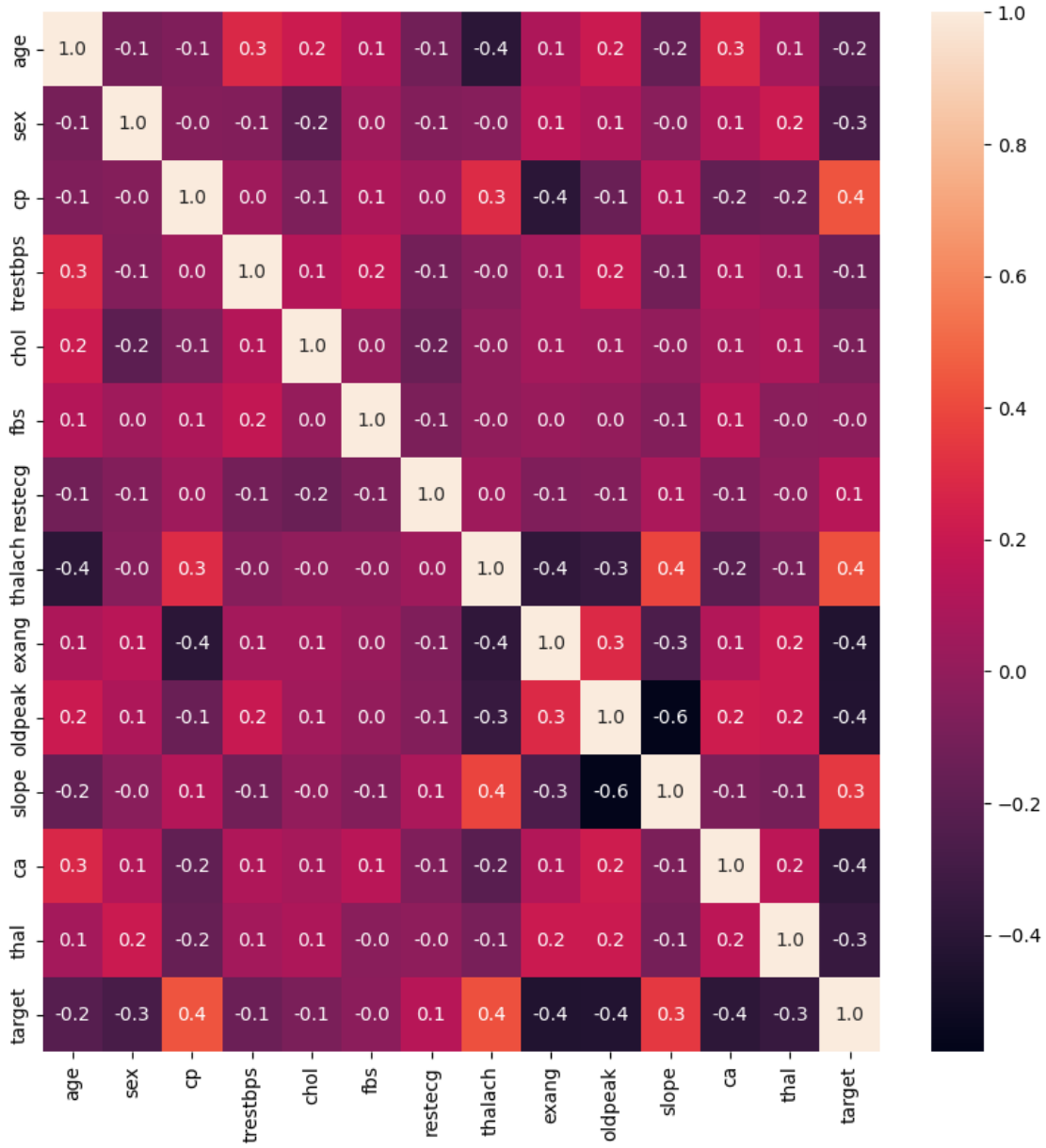
[20]:
```
# histograms for each variable
data.hist(figsize = (12, 12))
plt.show()
```



[21]:
```
pd.crosstab(data.age,data.target).plot(kind="bar",figsize=(20,6))
plt.title('Heart Disease Frequency for Ages')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



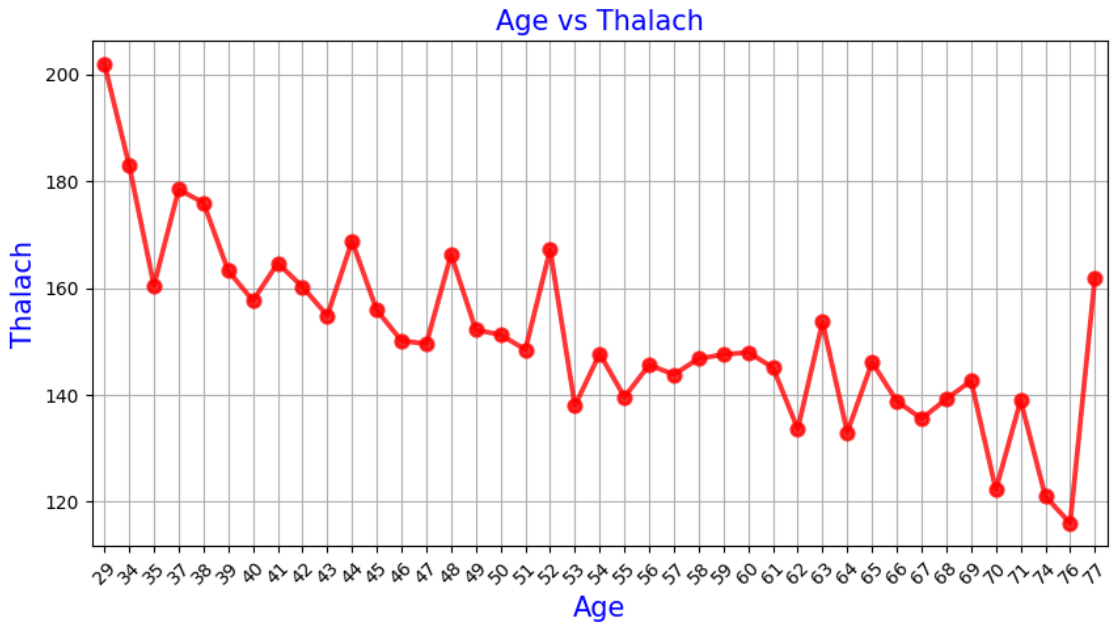
[22]:
```
plt.figure(figsize=(10,10))
sns.heatmap(data.corr(),annot=True,fmt='.1f')
plt.show()
```



3 Age vs Thalach

[23]:
```
age_unique=sorted(data.age.unique())
age_thalach_values=data.groupby('age')['thalach'].count().values
mean_thalach=[]
for i,age in enumerate(age_unique):
    mean_thalach.append(sum(data[data['age']==age].thalach)/
 ↪age_thalach_values[i])
```

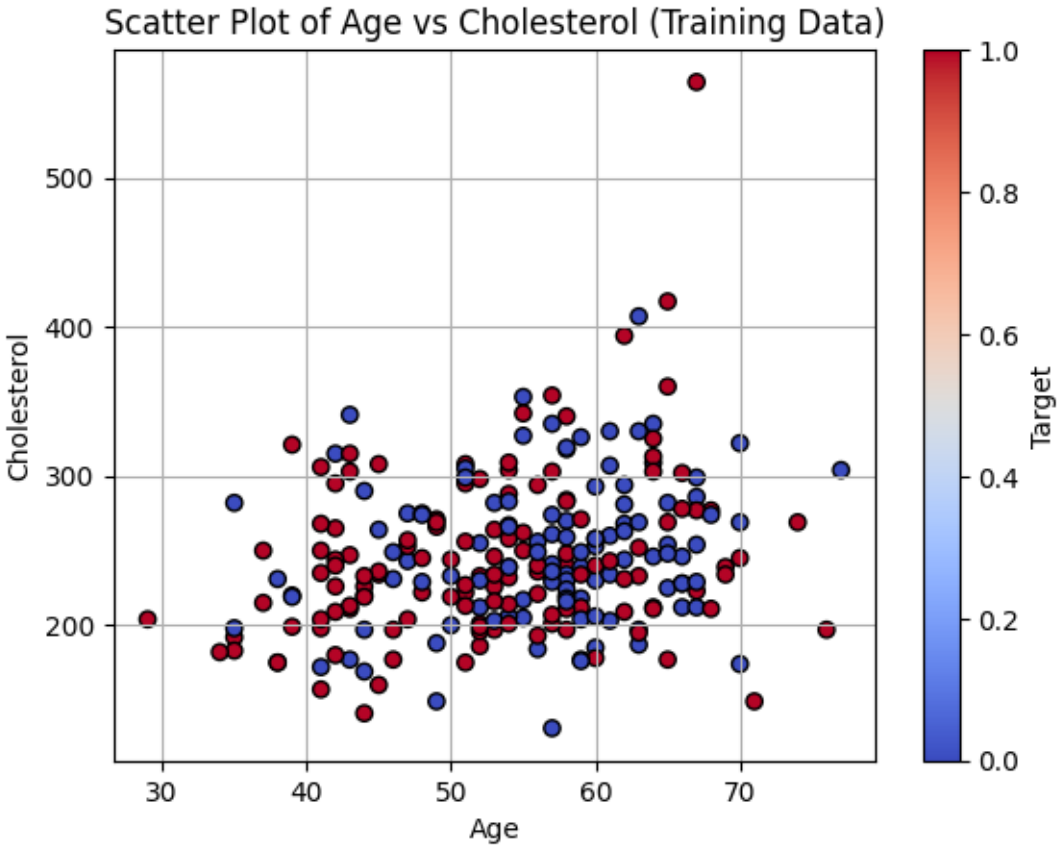
```
plt.figure(figsize=(10,5))
sns.pointplot(x=age_unique,y=mean_thalach,color='red',alpha=0.8)
plt.xlabel('Age',fontsize = 15,color='blue')
plt.xticks(rotation=45)
plt.ylabel('Thalach',fontsize = 15,color='blue')
plt.title('Age vs Thalach',fontsize = 15,color='blue')
plt.grid()
plt.show()
```



4 Test e train dataset

[26]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Carica il dataset
heart_data = pd.read_csv(r'C:\Users\david\Documents\heart.csv')

# Dividi il dataset in features (X) e target variable (y)
X = heart_data.drop('target', axis=1)  # Features
y = heart_data['target']               # Target variable

# Suddividi il dataset in set di addestramento e di test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

```
# Grafico per visualizzare i dati di addestramento
plt.scatter(X_train['age'], X_train['chol'], c=y_train, cmap='coolwarm',␣
 ↪marker='o', edgecolors='k')
plt.xlabel('Age')
plt.ylabel('Cholesterol')
plt.title('Scatter Plot of Age vs Cholesterol (Training Data)')
plt.colorbar(label='Target')
plt.grid(True)
plt.show()
```



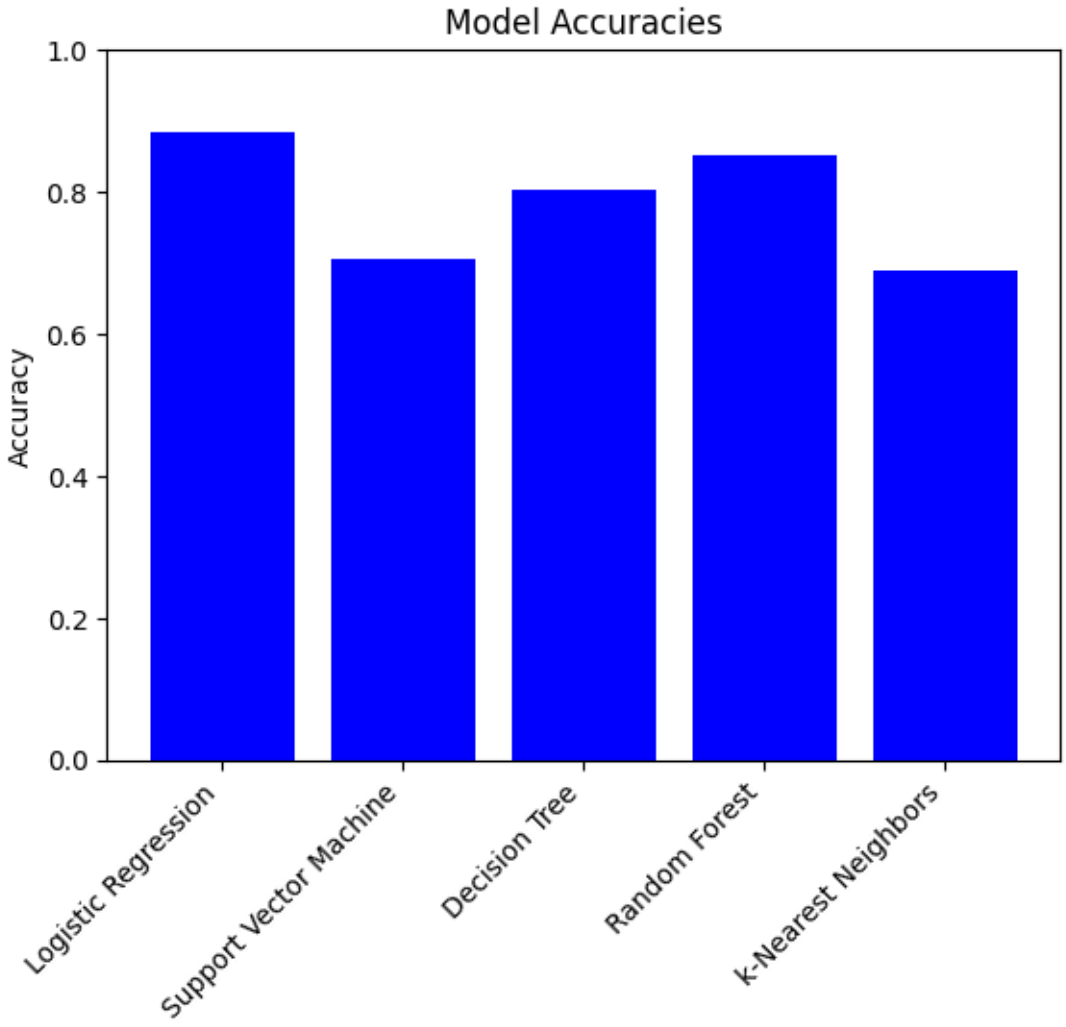
5 Accuratezza dei vari modelli

[5]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Carica il dataset
heart_data = pd.read_csv(r'C:\Users\david\Documents\heart.csv')

# Dividi il dataset in features (X) e target variable (y)
X = heart_data.drop('target', axis=1)  # Features
y = heart_data['target']               # Target variable

# Suddividi il dataset in set di addestramento e di test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Inizializza i modelli
models = {
    'Logistic Regression': LogisticRegression(),
    'Support Vector Machine': SVC(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'k-Nearest Neighbors': KNeighborsClassifier()
}

# Addestra e valuta i modelli
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    results[name] = accuracy

# Plot
plt.bar(results.keys(), results.values(), color='blue')
plt.ylabel('Accuracy')
plt.title('Model Accuracies')
plt.ylim(0, 1)
plt.xticks(rotation=45, ha='right')  # Ruota le etichette sull'asse x per una␣
 ↪migliore leggibilità
plt.show()
```

```
C:\Users\david\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n
2kfra8p0\LocalCache\local-packages\Python311\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```



6 Logistic Regression model

[10]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
# Carica il dataset
heart_data = pd.read_csv(r'C:\Users\david\Documents\heart.csv')

# Dividi il dataset in features (X) e target variable (y)
X = heart_data.drop('target', axis=1)
y = heart_data['target']

# Suddividi il dataset in set di addestramento e di test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Addestra il modello di regressione logistica
model = LogisticRegression().fit(X_train, y_train)

# Valuta l'accuratezza del modello sui dati di test
accuracy = accuracy_score(y_test, model.predict(X_test))
print(f'Accuracy del modello di regressione logistica: {accuracy:.2f}')
```

```
Accuracy del modello di regressione logistica: 0.89

C:\Users\david\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n
2kfra8p0\LocalCache\local-packages\Python311\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

7 Support Vector Machine model

[8]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Carica il dataset
heart_data = pd.read_csv(r'C:\Users\david\Documents\heart.csv')

# Dividi il dataset in features (X) e target variable (y)
X = heart_data.drop('target', axis=1)
y = heart_data['target']
```

```
# Suddividi il dataset in set di addestramento e di test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Definisci i parametri della griglia da testare
param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [0.1, 0.01, 0.001]}

# Cerca i migliori parametri con la cross-validation
grid_search = GridSearchCV(SVC(kernel='rbf'), param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Valuta l'accuratezza del modello migliore
accuracy = accuracy_score(y_test, grid_search.best_estimator_.predict(X_test))
print(f'Accuracy del modello Support Vector Machine: {accuracy:.2f}')
```

```
Accuracy del modello Support Vector Machine: 0.67
```

8 Decision Tree model

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Carica il dataset
heart_data = pd.read_csv(r'C:\Users\david\Documents\heart.csv')

# Dividi il dataset in features (X) e target variable (y)
X = heart_data.drop('target', axis=1)  # Features
y = heart_data['target']               # Target variable

# Suddividi il dataset in set di addestramento e di test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Inizializza e addestra il modello di albero decisionale
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)

# Valuta l'accuratezza del modello sui dati di test
y_pred = decision_tree.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy of Decision Tree model: {accuracy:.2f}')
```

```
Accuracy of Decision Tree model: 0.84
```

9 Random Forest model

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Carica il dataset
heart_data = pd.read_csv(r'C:\Users\david\Documents\heart.csv')

# Dividi il dataset in features (X) e target variable (y)
X = heart_data.drop('target', axis=1)  # Features
y = heart_data['target']               # Target variable

# Suddividi il dataset in set di addestramento e di test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Definisci i parametri della griglia da testare
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Inizializza il classificatore Random Forest
random_forest = RandomForestClassifier(random_state=42)

# Utilizza la ricerca dei parametri tramite cross-validation per trovare i␣
 ↪migliori parametri
grid_search = GridSearchCV(random_forest, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Ottieni il miglior modello dalla ricerca dei parametri
best_random_forest = grid_search.best_estimator_

# Valuta l'accuratezza del modello sui dati di test
y_pred = best_random_forest.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy of Random Forest model: {accuracy:.2f}')
```

```
Accuracy of Random Forest model: 0.85
```

10 k-Nearest Neighbors model

[3]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Carica il dataset
heart_data = pd.read_csv(r'C:\Users\david\Documents\heart.csv')

# Dividi il dataset in features (X) e target variable (y)
X = heart_data.drop('target', axis=1)  # Features
y = heart_data['target']               # Target variable

# Suddividi il dataset in set di addestramento e di test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Definisci i parametri della griglia da testare
param_grid = {'n_neighbors': range(1, 21)}  # Testa k da 1 a 20

# Inizializza il classificatore k-Nearest Neighbors
knn = KNeighborsClassifier()

# Utilizza la ricerca dei parametri tramite cross-validation per trovare il␣
 ↪miglior valore di k
grid_search = GridSearchCV(knn, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Ottieni il miglior modello dalla ricerca dei parametri
best_knn = grid_search.best_estimator_

# Valuta l'accuratezza del modello sui dati di test
y_pred = best_knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy of k-Nearest Neighbors model: {accuracy:.2f}')
```

```
Accuracy of k-Nearest Neighbors model: 0.66
```