In [133…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import keras
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from keras.datasets import mnist
from keras.models import Sequential
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier
from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D,
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
from keras.optimizers import Adam
```

In [134…

```
df = pd.read_csv(r'C:\Users\ayaYM\Downloads\heart-2.csv')
```

In [135…

```
print(df.head())
print(df.info())
print(df.describe())
```

```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   52    1   0       125   212    0        1      168      0      1.0      2
1   53    1   0       140   203    1        0      155      1      3.1      0
2   70    1   0       145   174    0        1      125      1      2.6      0
3   61    1   0       148   203    0        1      161      0      0.0      2
4   62    0   0       138   294    1        1      106      0      1.9      1

   ca  thal  target
0   2     3       0
1   0     3       0
2   0     3       0
3   1     3       0
4   3     2       0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1025 non-null   int64
 1   sex       1025 non-null   int64
 2   cp        1025 non-null   int64
 3   trestbps  1025 non-null   int64
 4   chol      1025 non-null   int64
 5   fbs       1025 non-null   int64
 6   restecg   1025 non-null   int64
 7   thalach   1025 non-null   int64
 8   exang     1025 non-null   int64
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64
 11  ca        1025 non-null   int64
 12  thal      1025 non-null   int64
 13  target    1025 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
None
              age          sex           cp     trestbps        chol  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.00000
mean     54.434146     0.695610     0.942439   131.611707   246.00000
std       9.072290     0.460373     1.029641    17.516718    51.59251
min      29.000000     0.000000     0.000000    94.000000   126.00000
25%      48.000000     0.000000     0.000000   120.000000   211.00000
50%      56.000000     1.000000     1.000000   130.000000   240.00000
75%      61.000000     1.000000     2.000000   140.000000   275.00000
max      77.000000     1.000000     3.000000   200.000000   564.00000

              fbs      restecg      thalach        exang      oldpeak  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.000000
mean      0.149268     0.529756   149.114146     0.336585     1.071512
std       0.356527     0.527878    23.005724     0.472772     1.175053
min       0.000000     0.000000    71.000000     0.000000     0.000000
25%       0.000000     0.000000   132.000000     0.000000     0.000000
50%       0.000000     1.000000   152.000000     0.000000     0.800000
75%       0.000000     1.000000   166.000000     1.000000     1.800000
max       1.000000     2.000000   202.000000     1.000000     6.200000

            slope           ca         thal       target
```

```
count  1025.000000  1025.000000  1025.000000  1025.000000
mean      1.385366     0.754146     2.323902     0.513171
std       0.617755     1.030798     0.620660     0.500070
min       0.000000     0.000000     0.000000     0.000000
25%       1.000000     0.000000     2.000000     0.000000
50%       1.000000     0.000000     2.000000     1.000000
75%       2.000000     1.000000     3.000000     1.000000
max       2.000000     4.000000     3.000000     1.000000
```

In [136…

```
print(df.shape)
```

```
(1025, 14)
```

In [137…

```
df.head(15)
```

Out[137…

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2
5	58	0	0	100	248	0	0	122	0	1.0	1	0	2
6	58	1	0	114	318	0	2	140	0	4.4	0	3	1
7	55	1	0	160	289	0	0	145	1	0.8	1	1	3
8	46	1	0	120	249	0	0	144	0	0.8	2	0	3
9	54	1	0	122	286	0	0	116	1	3.2	1	2	2
10	71	0	0	112	149	0	1	125	0	1.6	1	0	2
11	43	0	0	132	341	1	0	136	1	3.0	1	0	3
12	34	0	1	118	210	0	1	192	0	0.7	2	0	2
13	51	1	0	140	298	0	1	122	1	4.2	1	3	3
14	52	1	0	128	204	1	1	156	1	1.0	1	0	0

In [138…

```
df.describe()
```

Out[138...

	age	sex	cp	trestbps	chol	fbs	re
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.0
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.5
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.5
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.0
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.0
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.0
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.0
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.0

In [139...

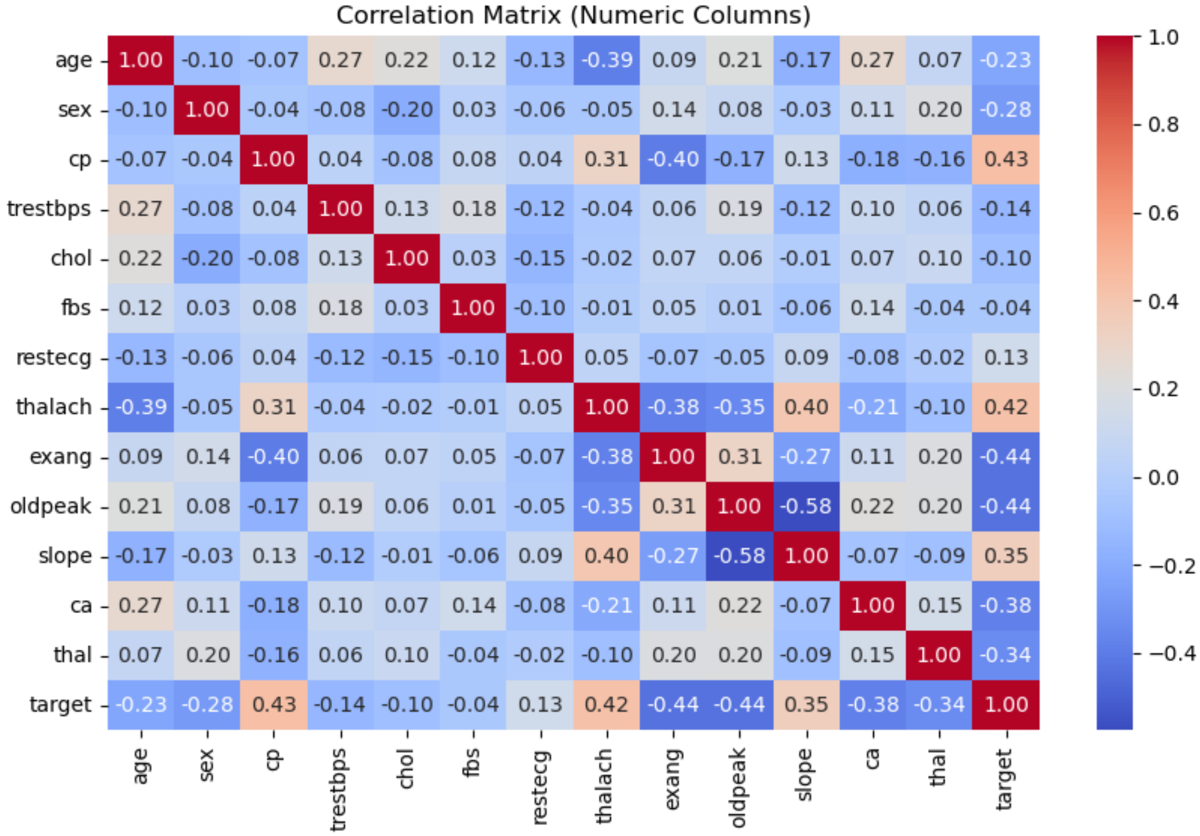
```
df.isnull().sum()
```

Out[139...

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

In [140...

```
plt.figure(figsize=(10, 6))
sns.heatmap(df.select_dtypes(include=['number']).corr(),
            annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix (Numeric Columns)")
plt.show()
```



In [141…

```
X = df.drop('target', axis=1)  # Features
y = df['target']               # Target

X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.3,random_state
```

In [142…

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

In [143…

```
print(df['target'].value_counts())
```

```
target
1    526
0    499
Name: count, dtype: int64
```

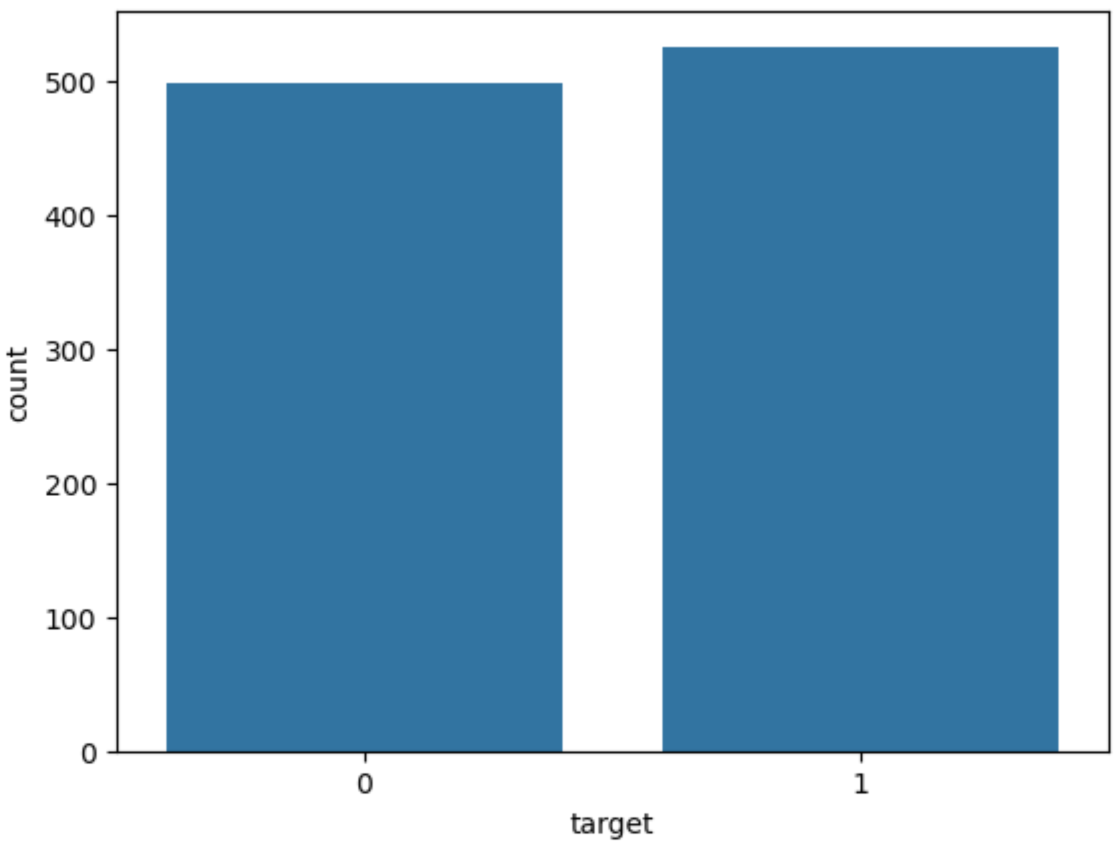
In [144…

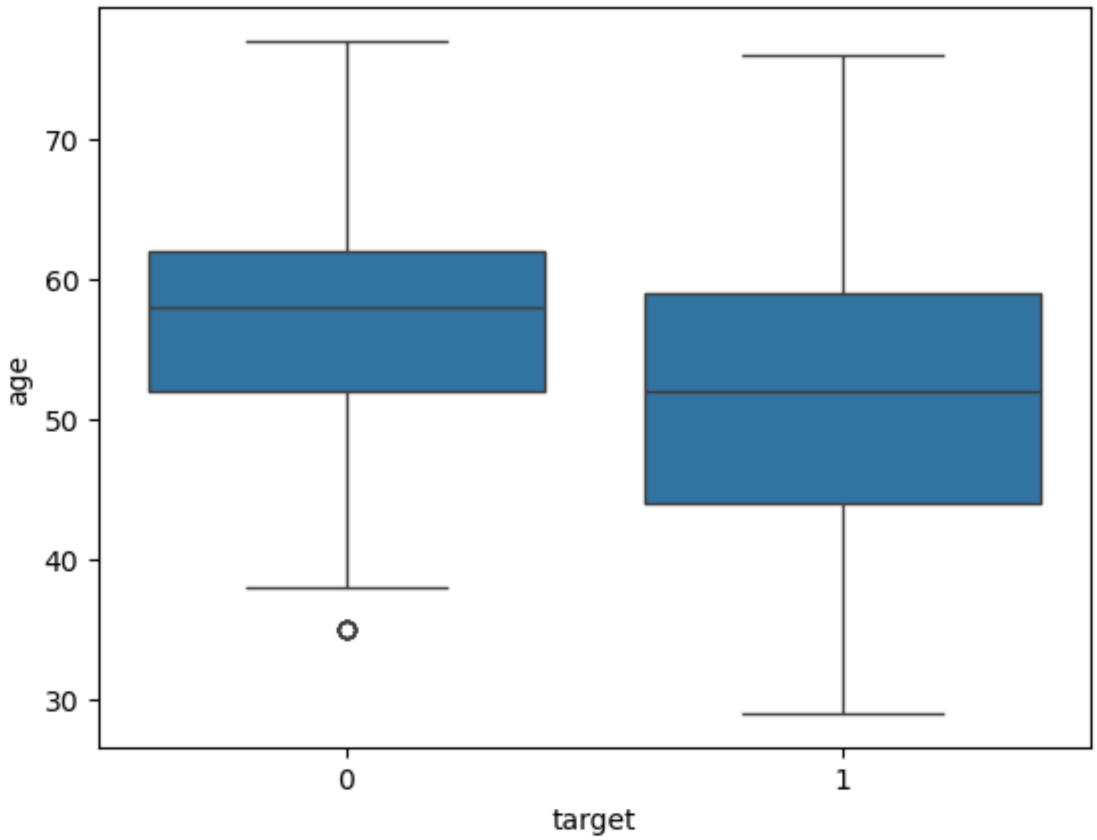
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

In [145…

```python
sns.countplot(x='target', data=df)
plt.show()

# Example boxplot for age vs target
sns.boxplot(x='target', y='age', data=df)
plt.show()
```





SVM models

In [146…

```
svm_models = {
    "SVC (RBF kernel)": SVC(kernel='rbf', random_state=42),
    "Linear SVM": SVC(kernel='linear', random_state=42),
    "Polynomial SVM": SVC(kernel='poly', degree=3, random_state=42)
}

# Train, predict, and evaluate each model
for name, model in svm_models.items():
    print(f"\n--- {name} ---")

    # Train
    model.fit(X_train_scaled, y_train)

    # Predict
    y_pred = model.predict(X_test_scaled)

    # Accuracy
    acc = accuracy_score(y_test, y_pred)
    print(f"Accuracy on test set: {acc:.4f}")
```

```
--- SVC (RBF kernel) ---
Accuracy on test set: 0.9058

--- Linear SVM ---
Accuracy on test set: 0.8474

--- Polynomial SVM ---
Accuracy on test set: 0.9123
```

```
SVC (RBF kernel) → 90.6% accuracy
The RBF (Radial Basis Function) kernel performs very well on this
dataset.
This suggests the decision boundary is nonlinear, and RBF can model
it better than a straight line.

Linear SVM → 84.7% accuracy
The linear kernel has lower performance compared to the nonlinear
kernels.
This suggests that the relationship between features and target in
the heart disease dataset is not purely linear — a linear decision
boundary cannot separate the classes perfectly.

Polynomial SVM → 91.2% accuracy
The polynomial kernel gives the highest accuracy among the three
models.
This indicates that the data has some complex patterns and
interactions that the polynomial kernel can capture effectively.
```

In [147…

```
 # Confusion matrix
 cm = confusion_matrix(y_test, y_pred)
 print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[133  17]
 [ 10 148]]
```

133: True negatives → class 0 correctly predicted as 0

17: False positives → class 0 wrongly predicted as 1

10: False negatives → class 1 wrongly predicted as 0

148: True positives → class 1 correctly predicted as 1

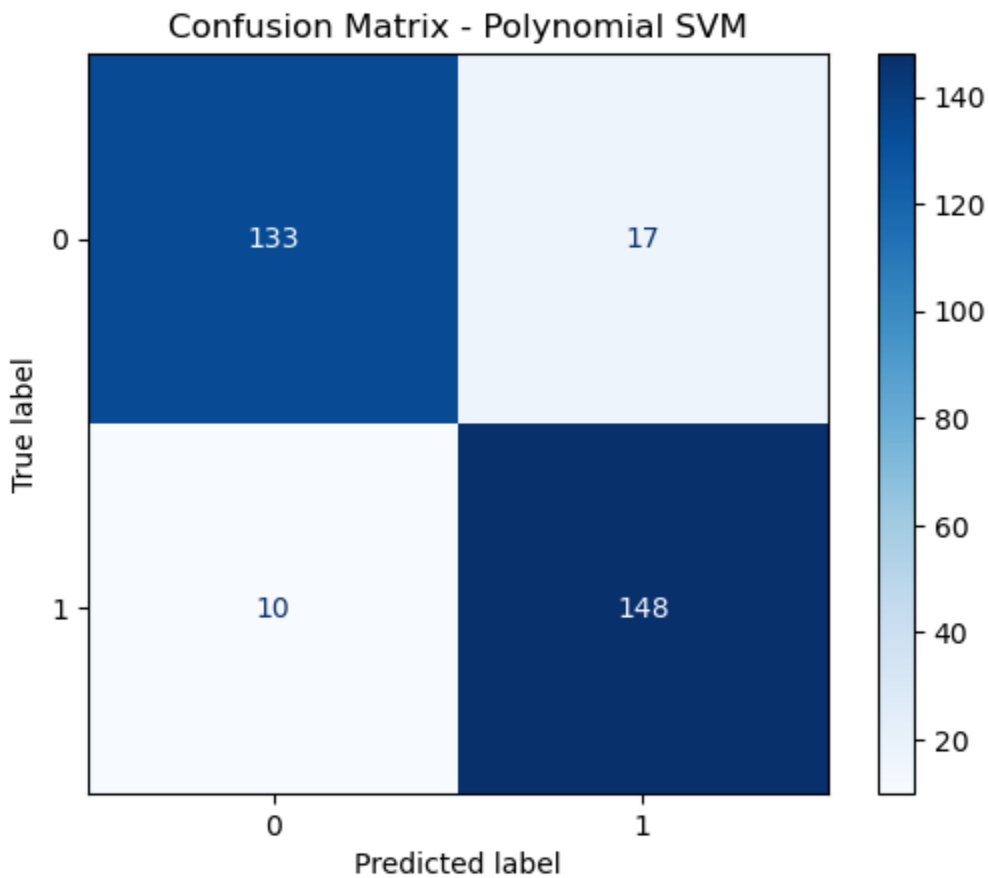
polynomial SVM is performing best, followed closely by the RBF SVM, while the linear SVM is lagging.

In [148…

```
    print("Classification Report:")
    print(classification_report(y_test, y_pred))

    # Plot confusion matrix
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.classes
```

```
    disp.plot(cmap=plt.cm.Blues)
    plt.title(f"Confusion Matrix - {name}")
    plt.show()
```

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

           0       0.93      0.89      0.91       150
           1       0.90      0.94      0.92       158

    accuracy                           0.91       308
   macro avg       0.91      0.91      0.91       308
weighted avg       0.91      0.91      0.91       308
```



Balanced performance on both classes

High overall accuracy (91%)

No obvious sign of extreme class imbalance or favoring one class

Neural Network Implementation Using Keras

GridSearchCV for best ALPHA and LEARNING RATE

In [149…

```
param_grid = {
    'alpha': [0.0001, 0.001, 0.01],
    'learning_rate_init' : [0.001,0.01,0.1]
```

```
}
grid_search = GridSearchCV(MLPClassifier(max_iter=1000, random_state=42), param_gri
grid_search.fit(X_train_scaled, y_train)
alpha_best = grid_search.best_params_['alpha']
learning_rate_best = grid_search.best_params_['learning_rate_init']
print(f"\nBest Alpha from GridSearchCV: {alpha_best}")
print(f"\nBest Learning_rate from GridSearchCV: {learning_rate_best}")
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits

Best Alpha from GridSearchCV: 0.0001

Best Learning_rate from GridSearchCV: 0.001
```

In [150…

```
model = Sequential([
    Dense(64, input_shape=(13,), activation='relu'), #input layer
    Dense(32, activation='relu'),                   #hidden layer
    Dense(2, activation='softmax')                  #output layer
])
```

```
c:\Users\ayaYM\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequ
ential models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [151…

```
model.summary()
```

Model: "sequential_8"

Layer (type)	Output Shape	Param #
dense_24 (Dense)	(None, 64)	896
dense_25 (Dense)	(None, 32)	2,080
dense_26 (Dense)	(None, 2)	66

Total params: 3,042 (11.88 KB)

Trainable params: 3,042 (11.88 KB)

Non-trainable params: 0 (0.00 B)

preparing the model for training :

which optimizer to use

which loss function to minimize

which evaluation metrics to track during training & validation

In [152…

```
model.compile(Adam(learning_rate=learning_rate_best),loss='sparse_categorical_cross
```

Training the model

In [153…

```
history = model.fit(X_train_scaled, y_train, batch_size=10, epochs=20, shuffle=True
```

```
Epoch 1/20
72/72 - 5s - 69ms/step - accuracy: 0.7545 - loss: 0.5214 - val_accuracy: 0.8279 - va
l_loss: 0.3752
Epoch 2/20
72/72 - 0s - 6ms/step - accuracy: 0.8550 - loss: 0.3506 - val_accuracy: 0.8571 - val
_loss: 0.3127
Epoch 3/20
72/72 - 0s - 5ms/step - accuracy: 0.8745 - loss: 0.3003 - val_accuracy: 0.8701 - val
_loss: 0.2902
Epoch 4/20
72/72 - 0s - 5ms/step - accuracy: 0.9010 - loss: 0.2691 - val_accuracy: 0.8701 - val
_loss: 0.2751
Epoch 5/20
72/72 - 0s - 5ms/step - accuracy: 0.9079 - loss: 0.2399 - val_accuracy: 0.8961 - val
_loss: 0.2604
Epoch 6/20
72/72 - 0s - 4ms/step - accuracy: 0.9191 - loss: 0.2151 - val_accuracy: 0.9091 - val
_loss: 0.2436
Epoch 7/20
72/72 - 0s - 5ms/step - accuracy: 0.9372 - loss: 0.1926 - val_accuracy: 0.9123 - val
_loss: 0.2343
Epoch 8/20
72/72 - 0s - 4ms/step - accuracy: 0.9470 - loss: 0.1671 - val_accuracy: 0.9286 - val
_loss: 0.2163
Epoch 9/20
72/72 - 0s - 4ms/step - accuracy: 0.9512 - loss: 0.1515 - val_accuracy: 0.9448 - val
_loss: 0.2114
Epoch 10/20
72/72 - 0s - 5ms/step - accuracy: 0.9637 - loss: 0.1285 - val_accuracy: 0.9513 - val
_loss: 0.1955
Epoch 11/20
72/72 - 0s - 5ms/step - accuracy: 0.9693 - loss: 0.1109 - val_accuracy: 0.9545 - val
_loss: 0.1813
Epoch 12/20
72/72 - 0s - 5ms/step - accuracy: 0.9763 - loss: 0.0998 - val_accuracy: 0.9578 - val
_loss: 0.1680
Epoch 13/20
72/72 - 0s - 5ms/step - accuracy: 0.9833 - loss: 0.0917 - val_accuracy: 0.9643 - val
_loss: 0.1576
Epoch 14/20
72/72 - 0s - 4ms/step - accuracy: 0.9847 - loss: 0.0752 - val_accuracy: 0.9610 - val
_loss: 0.1489
Epoch 15/20
72/72 - 0s - 4ms/step - accuracy: 0.9847 - loss: 0.0689 - val_accuracy: 0.9643 - val
_loss: 0.1391
Epoch 16/20
72/72 - 0s - 5ms/step - accuracy: 0.9902 - loss: 0.0585 - val_accuracy: 0.9643 - val
_loss: 0.1304
Epoch 17/20
72/72 - 0s - 4ms/step - accuracy: 0.9930 - loss: 0.0514 - val_accuracy: 0.9643 - val
_loss: 0.1284
Epoch 18/20
72/72 - 0s - 5ms/step - accuracy: 0.9930 - loss: 0.0445 - val_accuracy: 0.9708 - val
_loss: 0.1164
Epoch 19/20
72/72 - 0s - 4ms/step - accuracy: 0.9902 - loss: 0.0405 - val_accuracy: 0.9773 - val
```

```
_loss: 0.1159
Epoch 20/20
72/72 - 0s - 4ms/step - accuracy: 0.9944 - loss: 0.0360 - val_accuracy: 0.9773 - val
_loss: 0.1129
```

In [154…

```
test_loss, test_accuracy = model.evaluate(X_test_scaled, y_test, verbose=0)
print(f'Test Loss: {test_loss:.4f}')
print(f'Test Accuracy: {test_accuracy:.4f}')
```

```
Test Loss: 0.1129
Test Accuracy: 0.9773
```

In [155…

```
# Get predictions
y_pred_probs = model.predict(X_test_scaled)
y_pred = y_pred_probs.argmax(axis=1)

# Classification report
print(classification_report(y_test, y_pred))

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)
```

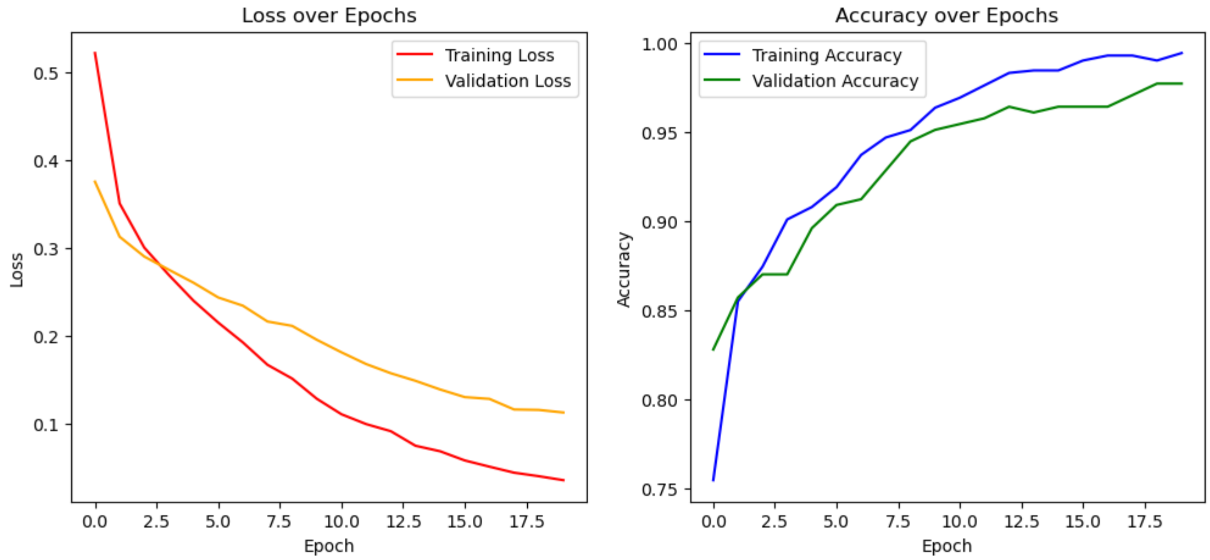
```
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       150
           1       0.98      0.97      0.98       158

    accuracy                           0.98       308
   macro avg       0.98      0.98      0.98       308
weighted avg       0.98      0.98      0.98       308

Confusion Matrix:
[[147   3]
 [  4 154]]
```

The neural network outperforms all SVM models across every metric — accuracy, precision, recall, and F1-score. It makes much fewer mistakes (7 vs 27 errors).

In [156…

```
plt.figure(figsize=(12, 5))

# Loss plot
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss', color='red')
plt.plot(history.history['val_loss'], label='Validation Loss', color='orange')
plt.title('Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Accuracy plot
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy', color='blue')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', color='green
```

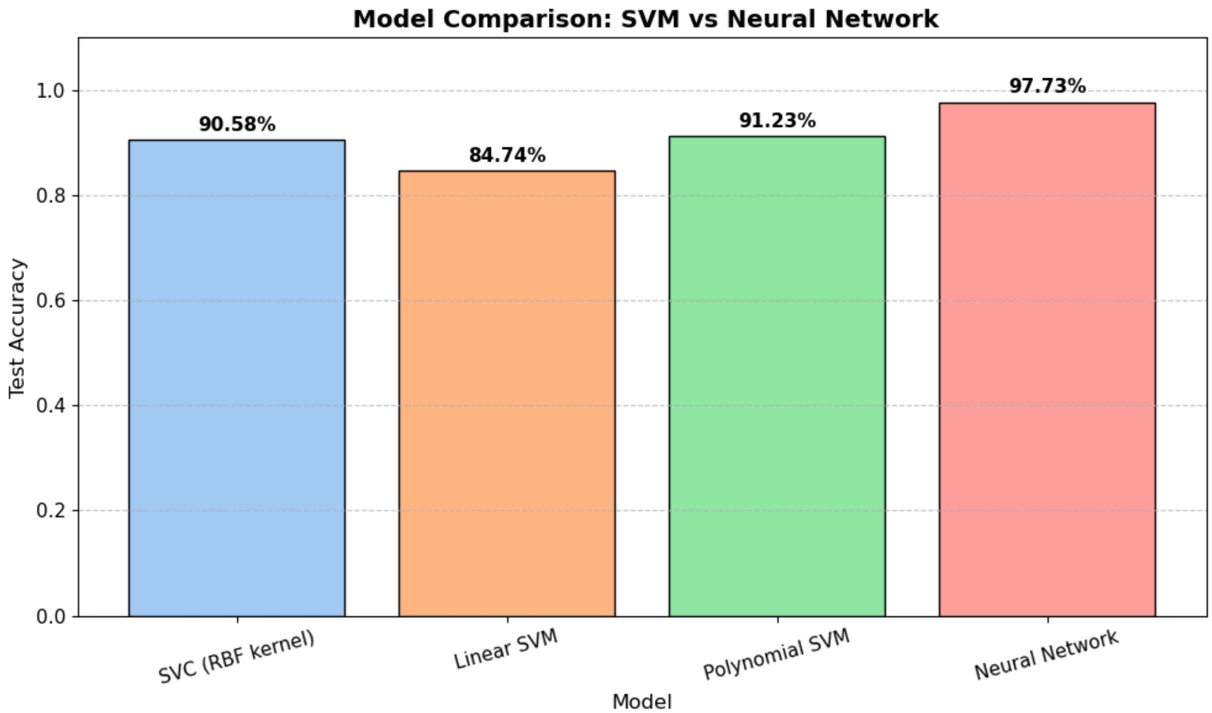
```
plt.title('Accuracy over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



In [157…

```
nn_test_accuracy = test_accuracy  # from model.evaluate

svm_accuracies = {}
for name, svm_model in svm_models.items():
    svm_model.fit(X_train_scaled, y_train)
    y_pred = svm_model.predict(X_test_scaled)
    acc = accuracy_score(y_test, y_pred)
    svm_accuracies[name] = acc

# Step 2: Combine all into one dictionary
all_accuracies = svm_accuracies.copy()
all_accuracies['Neural Network'] = nn_test_accuracy




plt.figure(figsize=(10, 6))
colors = sns.color_palette('pastel')[0:4]
bars = plt.bar(all_accuracies.keys(), all_accuracies.values(), color=colors, edgeco

# Step 4: Add accuracy labels on top (in %)
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2,
             height + 0.01,
             f"{height * 100:.2f}%",
             ha='center',
             va='bottom',
             fontsize=11,
             fontweight='bold')
```

```
plt.ylim(0, 1.1)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xlabel('Model', fontsize=12)
plt.ylabel('Test Accuracy', fontsize=12)
plt.title(' Model Comparison: SVM vs Neural Network', fontsize=14, fontweight='bold
plt.xticks(rotation=15, fontsize=11)
plt.yticks(fontsize=11)
plt.tight_layout()
plt.show()
```



Summary of Key Findings:

Neural Network is the best-performing model with the highest accuracy (97.73%) and a relatively low test loss (0.1129). It seems to generalize well, providing excellent results.

SVM with RBF Kernel is also very strong, achieving 90.58% accuracy, but it's still slightly behind the neural network.

Polynomial SVM is competitive with the RBF kernel, yielding an accuracy of 91.23%. However, it is still a bit less effective than the RBF kernel in this particular test.

Linear SVM performs the worst among the three, with an accuracy of 84.74%. It’s not well-suited for datasets with non-linear relationships, which is likely why its performance is lower.