Mounting

```
import pandas as pd
data=pd.read_csv('/content/drive/MyDrive/health_data.csv')
```

Basic Information

[ ] ↳ *3 cells hidden*

Preprocessing

Dropping

```
datadrop= data.drop(columns=['Column 1', 'id'])
```

```
datadrop.info()
```

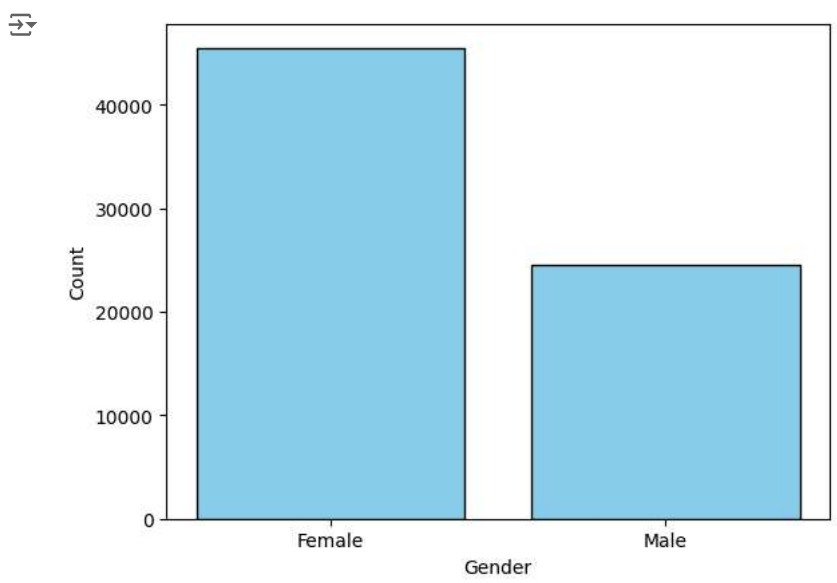
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   age          70000 non-null  int64  
 1   gender       70000 non-null  int64  
 2   height       70000 non-null  int64  
 3   weight       70000 non-null  float64
 4   ap_hi        70000 non-null  int64  
 5   ap_lo        70000 non-null  int64  
 6   cholesterol  70000 non-null  int64  
 7   gluc         70000 non-null  int64  
 8   smoke        70000 non-null  int64  
 9   alco         70000 non-null  int64  
 10  active       70000 non-null  int64  
 11  cardio       70000 non-null  int64  
dtypes: float64(1), int64(11)
memory usage: 6.4 MB
```

```
df=datadrop.drop_duplicates()
```

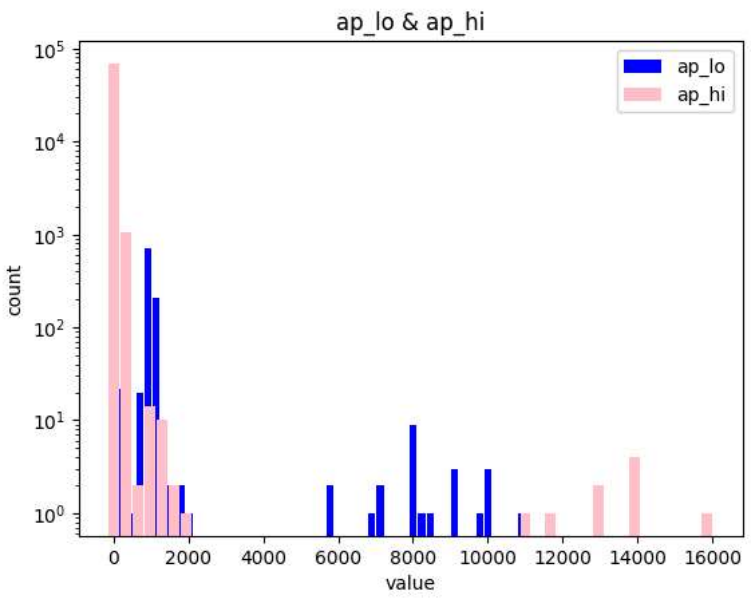
```
(df.duplicated()).sum()
```

```
0
```

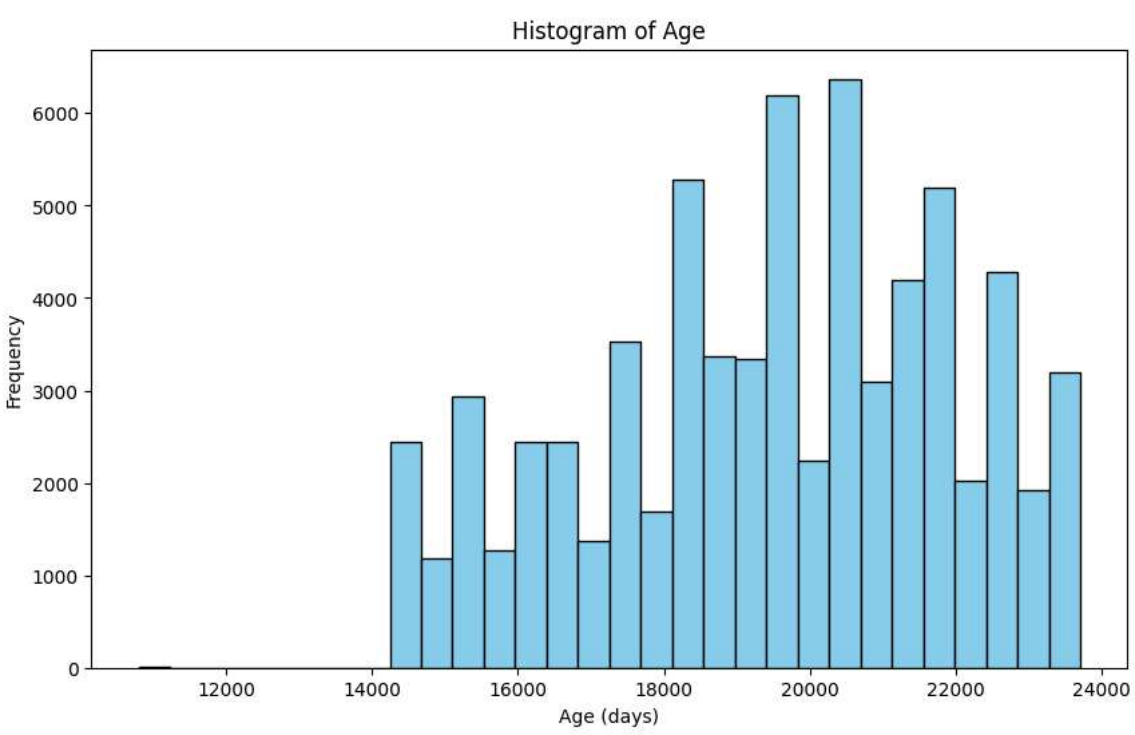
```
import matplotlib.pyplot as plt

# Assuming df['gender'] contains gender information (0 for female, 1 for male)
gender_counts = df['gender'].value_counts()

# Map gender labels
gender_labels = {0: 'Female', 1: 'Male'}
gender_counts.index = gender_counts.index.map(gender_labels)

# Plotting the histogram
plt.bar(gender_counts.index, gender_counts.values, color='skyblue', edgecolor='black')
plt.xlabel('Gender')
plt.ylabel('Count')

plt.show()
```



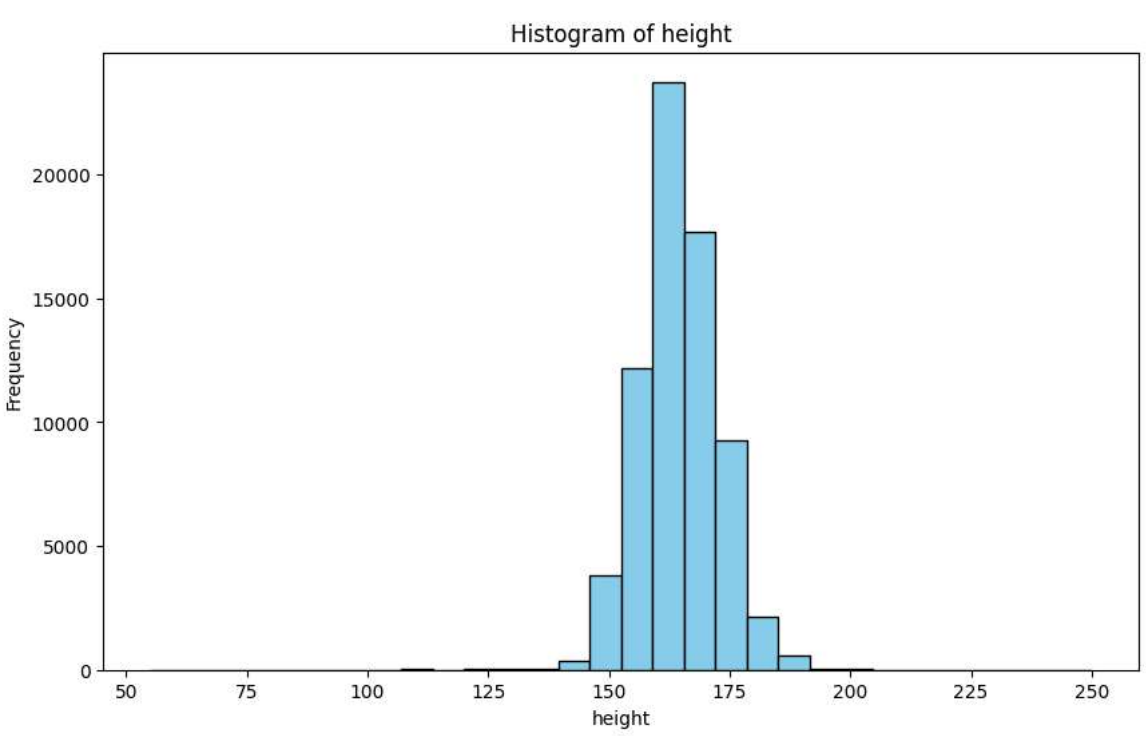

```
import matplotlib.pyplot as plt
plt.hist(data['ap_lo'],bins=50,rwidth=0.85,color="blue",log=True,label="ap_lo")
plt.hist(data['ap_hi'],bins=50,rwidth=0.85,color="pink",log=True,label="ap_hi")
plt.legend()
plt.title("ap_lo & ap_hi")
plt.xlabel("value")
plt.ylabel("count")
plt.show()
```



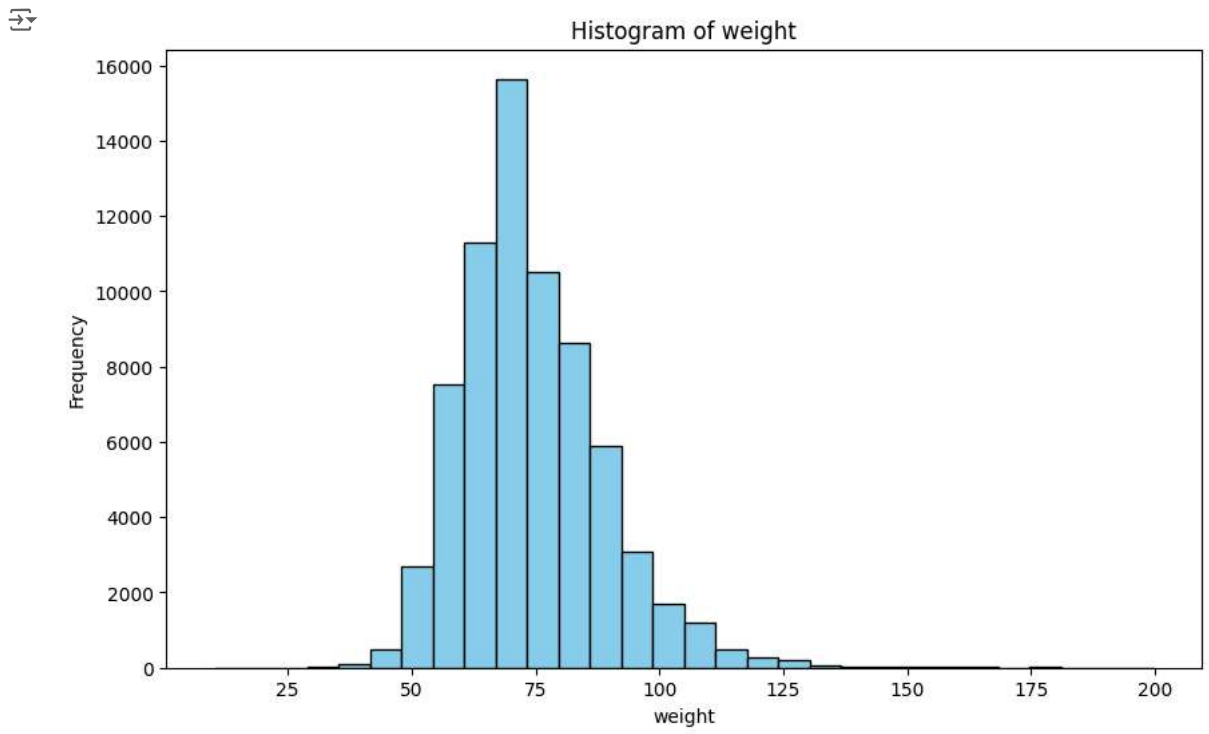

```
import pandas as pd
import matplotlib.pyplot as plt


# Plot histogram of the 'age' attribute
plt.figure(figsize=(10, 6))
plt.hist(data['age'], bins=30, color='skyblue', edgecolor='black')

# Add labels and title
plt.xlabel('Age (days)')
plt.ylabel('Frequency')
plt.title('Histogram of Age')

# Show plot
plt.show()
```



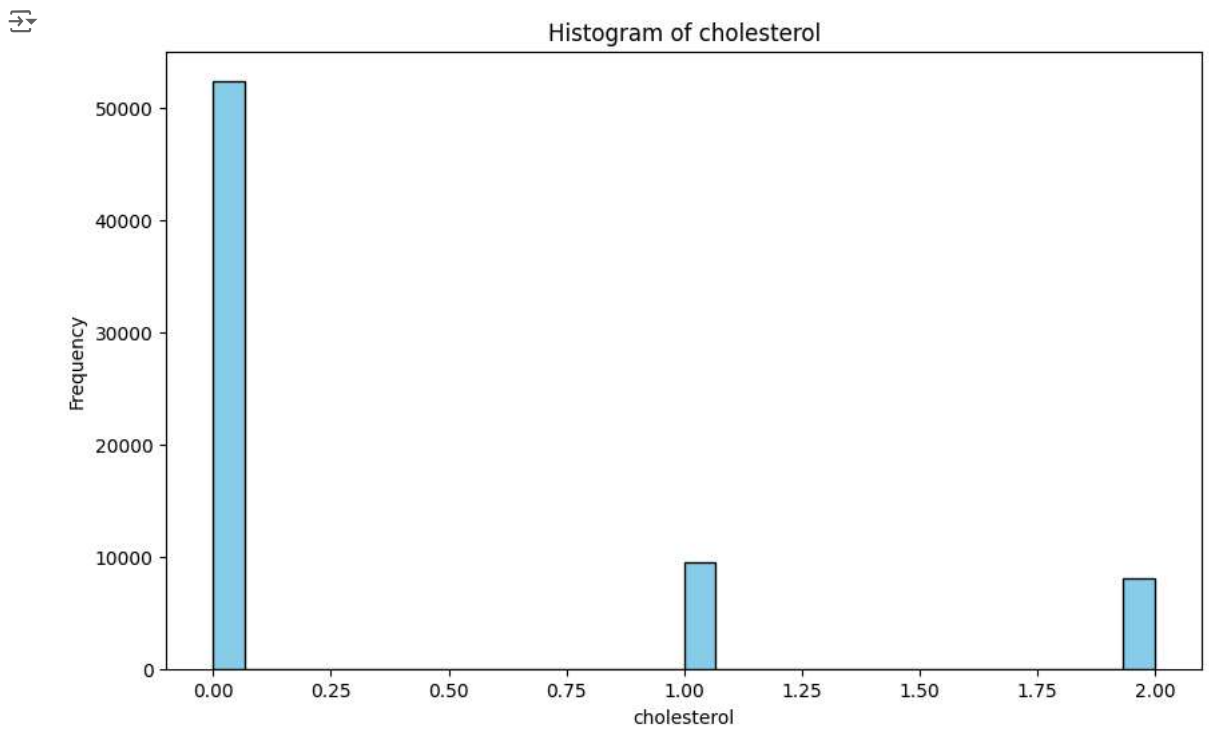

```
import pandas as pd
import matplotlib.pyplot as plt


# Plot histogram of the 'age' attribute
plt.figure(figsize=(10, 6))
plt.hist(data['height'], bins=30, color='skyblue', edgecolor='black')

# Add labels and title
plt.xlabel('height')
plt.ylabel('Frequency')
plt.title('Histogram of height')

# Show plot
plt.show()
```



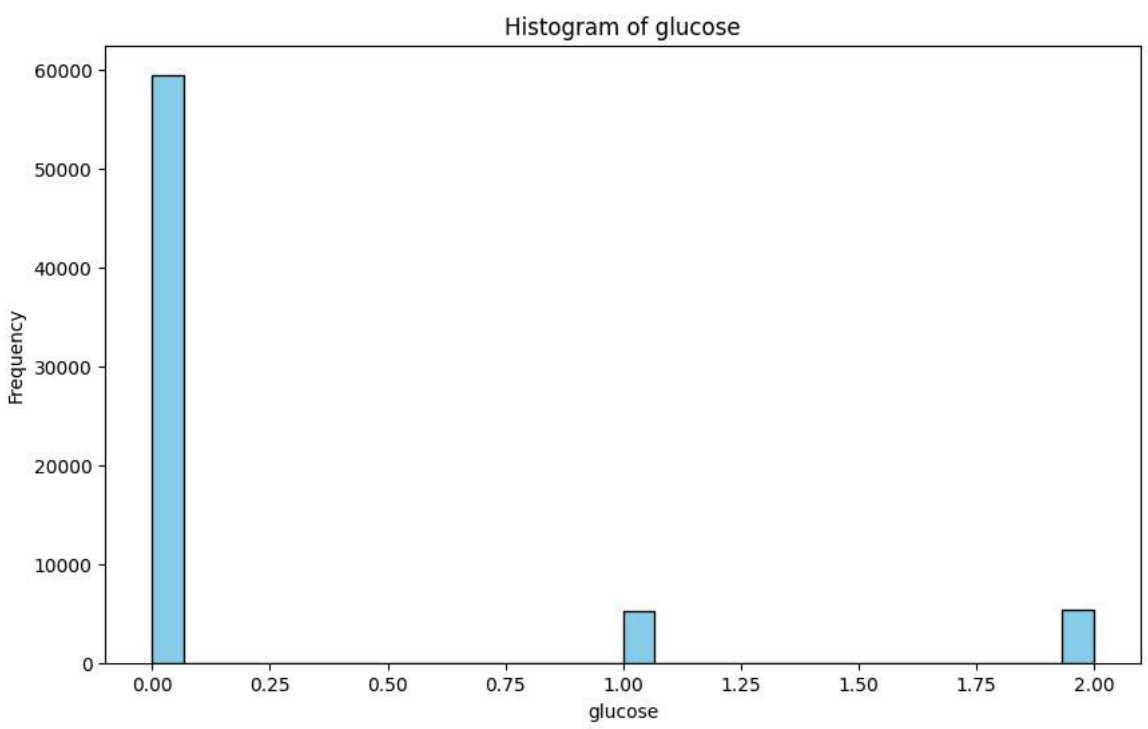
```
import pandas as pd
import matplotlib.pyplot as plt


# Plot histogram of the 'age' attribute
plt.figure(figsize=(10, 6))
plt.hist(data['weight'], bins=30, color='skyblue', edgecolor='black')

# Add labels and title
plt.xlabel('weight')
plt.ylabel('Frequency')
plt.title('Histogram of weight')

# Show plot
plt.show()
```



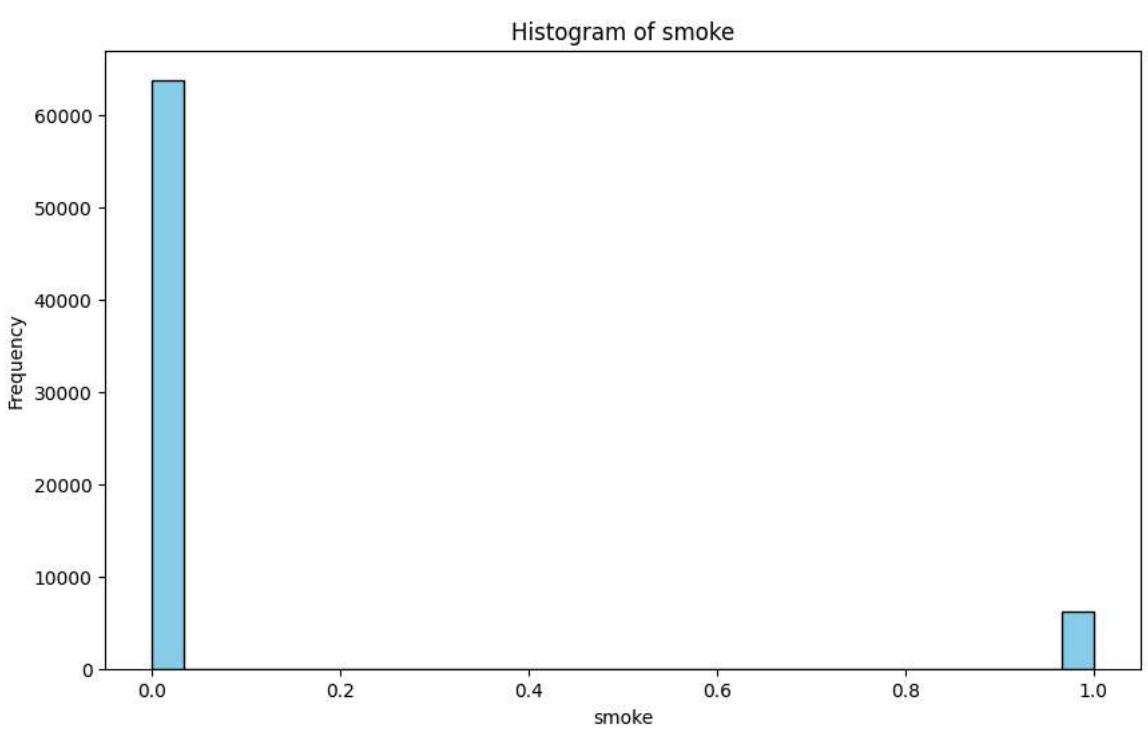

```
import pandas as pd
import matplotlib.pyplot as plt


# Plot histogram of the 'age' attribute
plt.figure(figsize=(10, 6))
plt.hist(data['cholesterol'], bins=30, color='skyblue', edgecolor='black')

# Add labels and title
plt.xlabel('cholesterol')
plt.ylabel('Frequency')
plt.title('Histogram of cholesterol')

# Show plot
plt.show()
```



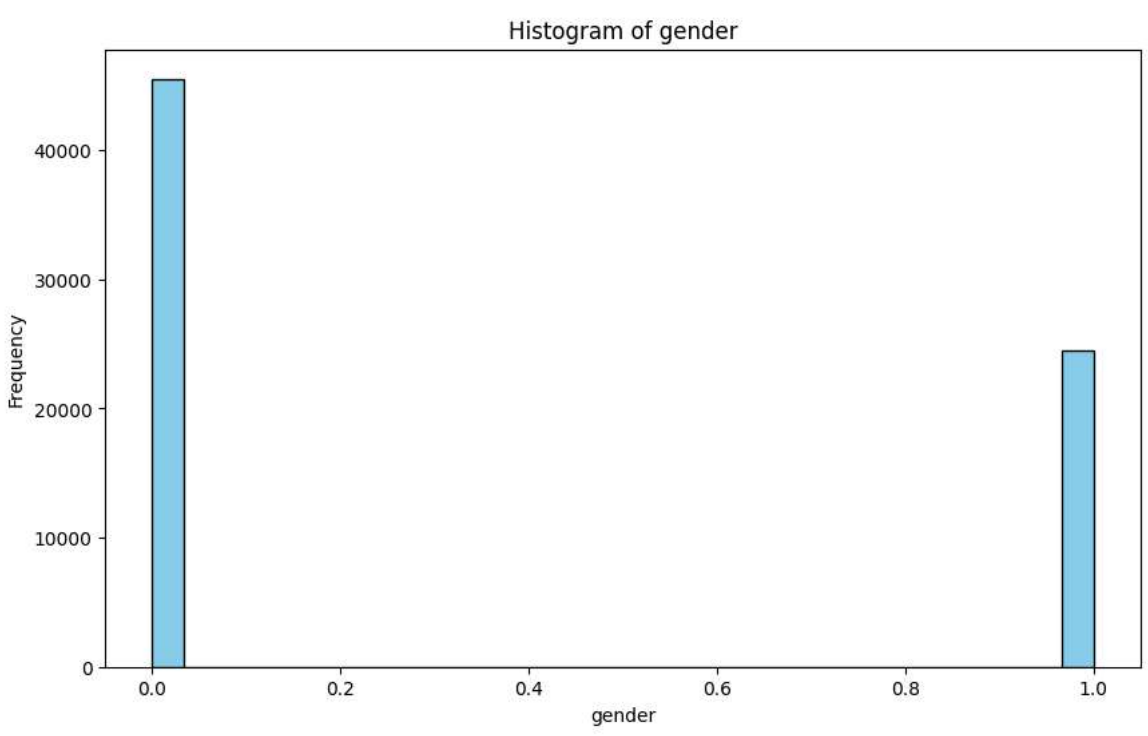

```
import pandas as pd
import matplotlib.pyplot as plt


# Plot histogram of the 'age' attribute
plt.figure(figsize=(10, 6))
plt.hist(data['gluc'], bins=30, color='skyblue', edgecolor='black')

# Add labels and title
plt.xlabel('glucose')
plt.ylabel('Frequency')
plt.title('Histogram of glucose')

# Show plot
plt.show()
```



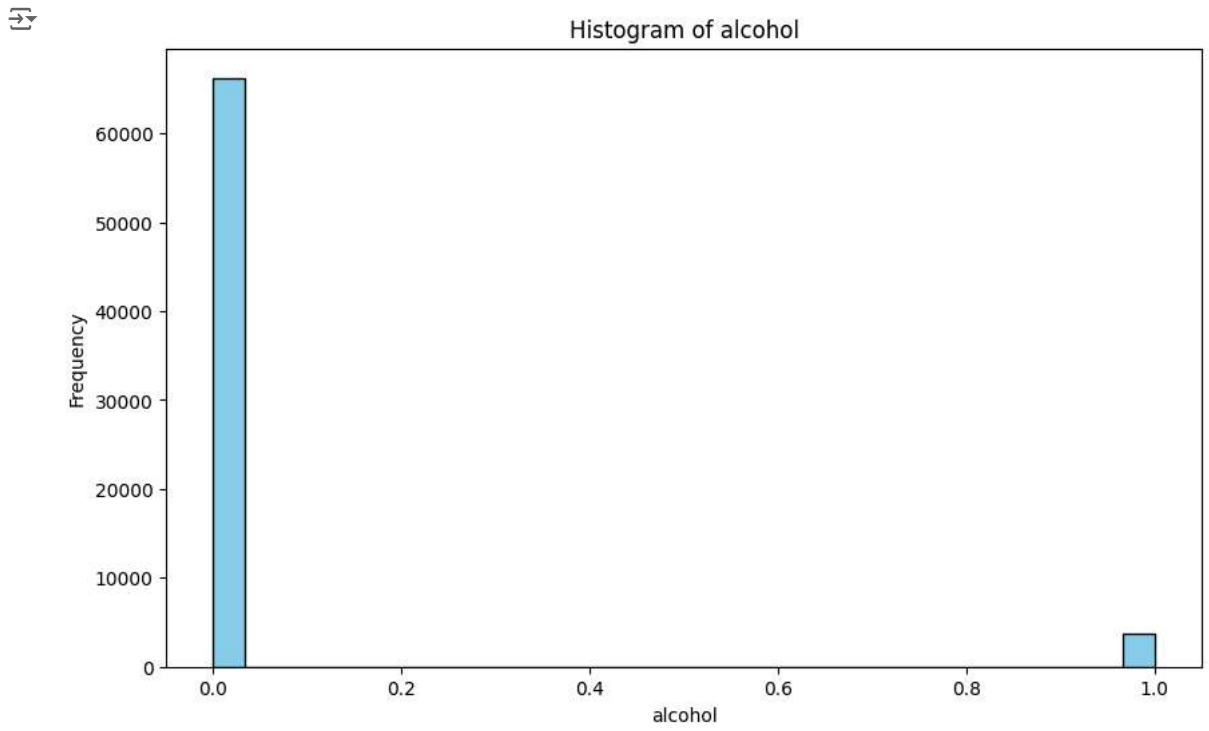

```
import pandas as pd
import matplotlib.pyplot as plt


# Plot histogram of the 'age' attribute
plt.figure(figsize=(10, 6))
plt.hist(data['smoke'], bins=30, color='skyblue', edgecolor='black')

# Add labels and title
plt.xlabel('smoke')
plt.ylabel('Frequency')
plt.title('Histogram of smoke')

# Show plot
plt.show()
```



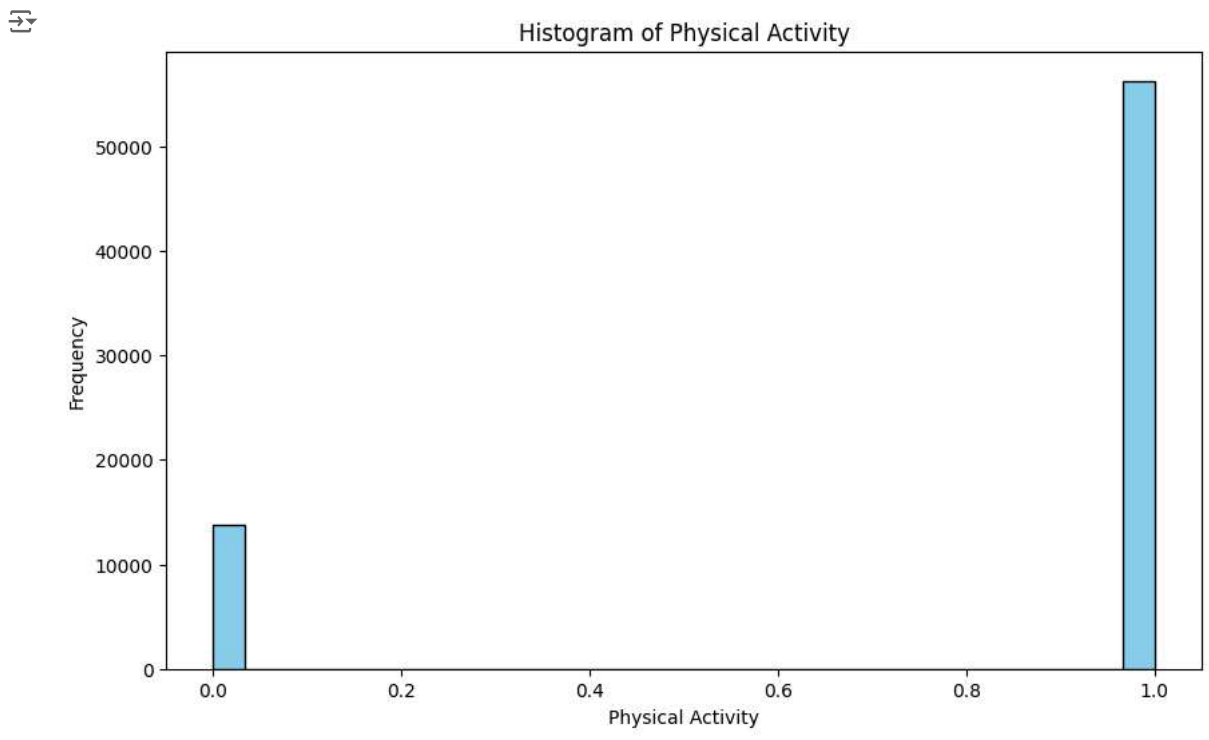

```
import pandas as pd
import matplotlib.pyplot as plt


# Plot histogram of the 'age' attribute
plt.figure(figsize=(10, 6))
plt.hist(data['gender'], bins=30, color='skyblue', edgecolor='black')

# Add labels and title
plt.xlabel('gender')
plt.ylabel('Frequency')
plt.title('Histogram of gender')

# Show plot
plt.show()
```



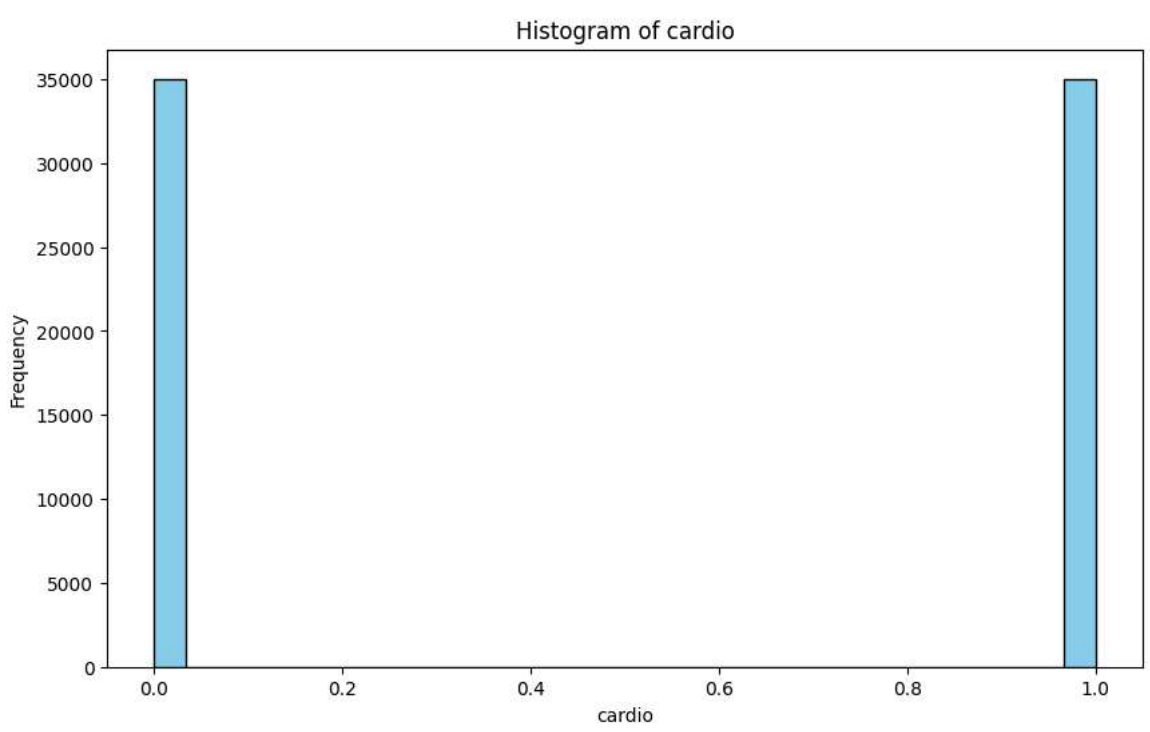
```
import pandas as pd
import matplotlib.pyplot as plt


# Plot histogram of the 'age' attribute
plt.figure(figsize=(10, 6))
plt.hist(data['alco'], bins=30, color='skyblue', edgecolor='black')

# Add labels and title
plt.xlabel('alcohol')
plt.ylabel('Frequency')
plt.title('Histogram of alcohol')

# Show plot
plt.show()
```




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


# Plot histogram of the 'age' attribute
plt.figure(figsize=(10, 6))
plt.hist(data['active'], bins=30, color='skyblue', edgecolor='black')

# Add labels and title
plt.xlabel('Physical Activity')
plt.ylabel('Frequency')
plt.title('Histogram of Physical Activity')

# Show plot
plt.show()
```




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


# Plot histogram of the 'age' attribute
plt.figure(figsize=(10, 6))
plt.hist(data['cardio'], bins=30, color='skyblue', edgecolor='black')

# Add labels and title
plt.xlabel('cardio')
plt.ylabel('Frequency')
plt.title('Histogram of cardio')

# Show plot
plt.show()
```




```
datadrop.cardio.value_counts()
```

```
cardio
0    35021
1    34979
Name: count, dtype: int64
```

```
#  df = datadrop[datadrop['ap_hi'] <= 300]
#  df = datadrop.reset_index(drop=True)  # Reset index to avoid index mismatch
#  df = datadrop[(datadrop['ap_lo'] <= datadrop['ap_hi']) & (datadrop['ap_lo'] > 10)]
```

```
datadrop.describe().T
```

	count	mean	std	min	25%	50%	75%	max
age	70000.0	19468.865814	2467.251667	10798.0	17664.0	19703.0	21327.0	23713.0
gender	70000.0	0.349571	0.476838	0.0	0.0	0.0	1.0	1.0
height	70000.0	164.359229	8.210126	55.0	159.0	165.0	170.0	250.0
weight	70000.0	74.205690	14.395757	10.0	65.0	72.0	82.0	200.0
ap_hi	70000.0	128.817286	154.011419	-150.0	120.0	120.0	140.0	16020.0
ap_lo	70000.0	96.630414	188.472530	-70.0	80.0	80.0	90.0	11000.0
cholesterol	70000.0	0.366871	0.680250	0.0	0.0	0.0	1.0	2.0
gluc	70000.0	0.226457	0.572270	0.0	0.0	0.0	0.0	2.0
smoke	70000.0	0.088129	0.283484	0.0	0.0	0.0	0.0	1.0
alco	70000.0	0.053771	0.225568	0.0	0.0	0.0	0.0	1.0
active	70000.0	0.803729	0.397179	0.0	1.0	1.0	1.0	1.0
cardio	70000.0	0.499700	0.500003	0.0	0.0	0.0	1.0	1.0

Train Test split

```
X=df.drop("cardio", axis=1)
y=df["cardio"]
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,train_size=0.7, test_size=0.3)
X_train.shape
```

```
(48983, 11)
```

```
print(type(y_test))
print(type(y_train))
print(y_test[:5])
print(y_train[:5])
```

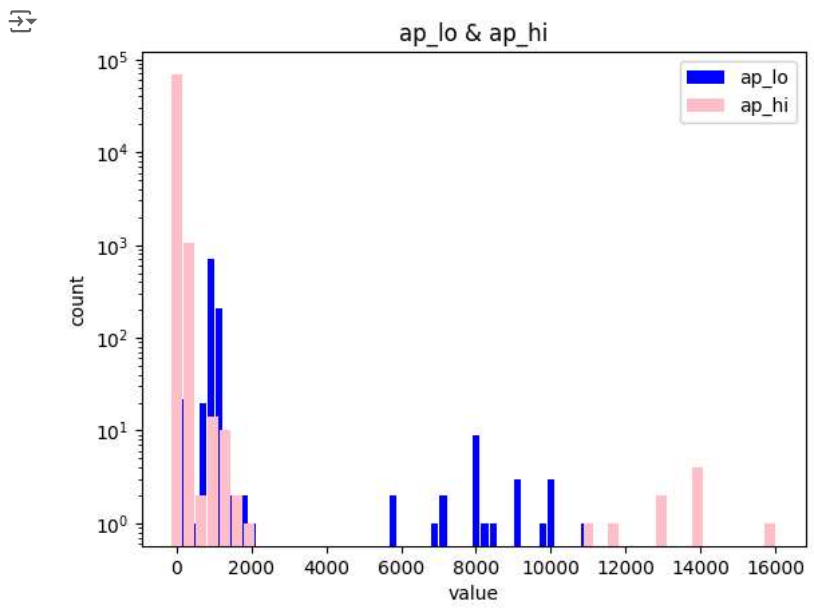
```
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
31874    0
14192    1
54984    0
33991    1
25935    0
Name: cardio, dtype: int64
61046    1
9745     1
32       0
1830     0
510      0
Name: cardio, dtype: int64
```

Scaling

```
#from sklearn.preprocessing import MinMaxScaler

# Scale input features
#scaler_X = MinMaxScaler()
#X_train_scaled = scaler_X.fit_transform(X_train)
#X_test_scaled = scaler_X.transform(X_test)

# Scale target variable
#scaler_y = MinMaxScaler()
#y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1))
#y_test_scaled = scaler_y.transform(y_test.values.reshape(-1, 1))
```

```
import numpy as np
from sklearn.preprocessing import StandardScaler

# Initialize StandardScaler
scaler = StandardScaler()

# Fit and transform the training data
X_train_scaled = scaler.fit_transform(X_train)

# Transform the testing data using the same scaler
X_test_scaled = scaler.transform(X_test)
```

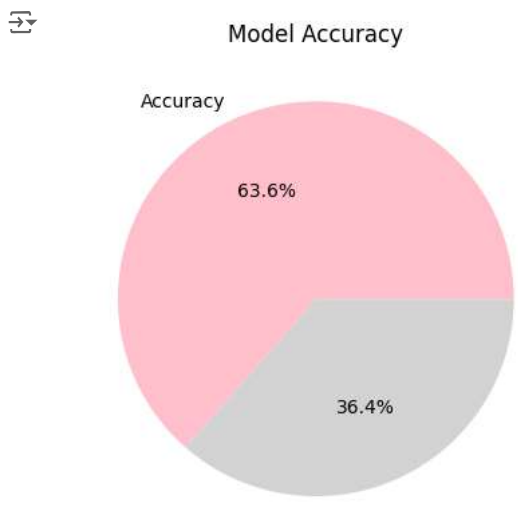
```
import matplotlib.pyplot as plt
plt.hist(df['ap_lo'],bins=50,rwidth=0.85,color="blue",log=True,label="ap_lo")
plt.hist(df['ap_hi'],bins=50,rwidth=0.85,color="pink",log=True,label="ap_hi")
plt.legend()
plt.title("ap_lo & ap_hi")
plt.xlabel("value")
plt.ylabel("count")
plt.show()
```



Classifiers

Decision Tree

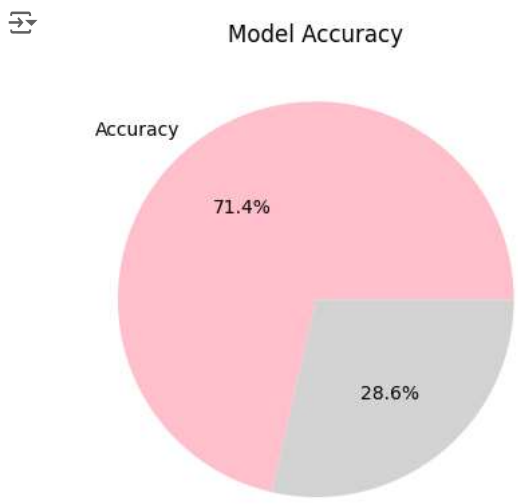
```
import time

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score

Dtree= DecisionTreeClassifier()
start=time.time()
Dtree.fit(X_train_scaled,y_train)
predict_Dtree=Dtree.predict(X_test_scaled)
end=time.time()
# Calculate accuracy
accuracy = accuracy_score(y_test, predict_Dtree)
# Plot accuracy
plt.pie([accuracy, 1 - accuracy], labels=['Accuracy', ''], colors=['pink', 'lightgray'], autopct='%1.1f%%')
plt.title("Model Accuracy")
plt.show()

print("Accuracy : ",accuracy_score(predict_Dtree, y_test))

print("F1-score: ",f1_score(predict_Dtree, y_test))

print("Precision Score : ",precision_score(predict_Dtree, y_test))

print("Recall Score : ",recall_score(predict_Dtree, y_test))

calculate=end-start
print("Time : ", calculate)
```




```
Accuracy :  0.6355928166531701
F1-score:  0.6364759551416082
Precision Score :  0.6375059495478343
Recall Score :  0.635449283613246
Time :  0.5257558822631836
```

Random Forest

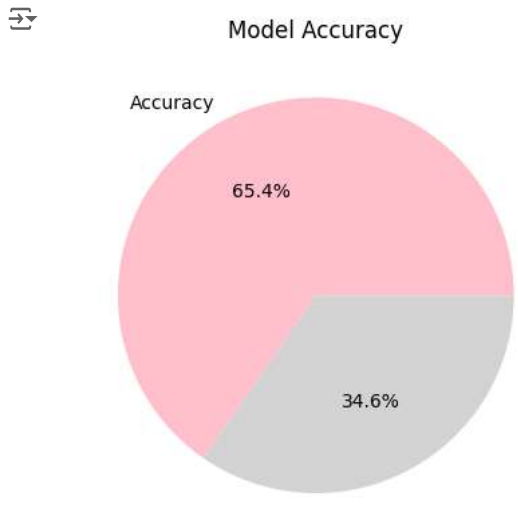
```
import time

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score

Rf= RandomForestClassifier()
start=time.time()
Rf.fit(X_train_scaled,y_train)
predict_Rf=Rf.predict(X_test_scaled)
end=time.time()
accuracy = accuracy_score(predict_Rf, y_test)
# Plot accuracy
plt.pie([accuracy, 1 - accuracy], labels=['Accuracy', ''], colors=['pink', 'lightgray'], autopct='%1.1f%%')
plt.title("Model Accuracy")
plt.show()


print("Accuracy : ",accuracy_score(predict_Rf, y_test))

print("F1-score: ",f1_score(predict_Rf, y_test))

print("Precision Score : ",precision_score(predict_Rf, y_test))

print("Recall Score : ",recall_score(predict_Rf, y_test))

calculate=end-start
print("Time : ", calculate)
```




```
Accuracy :  0.7140951745820036
F1-score:  0.7086690612561887
Precision Score :  0.694907187053784
Recall Score :  0.7229870258492621
Time :  14.988072395324707
```

Ridge Classifier

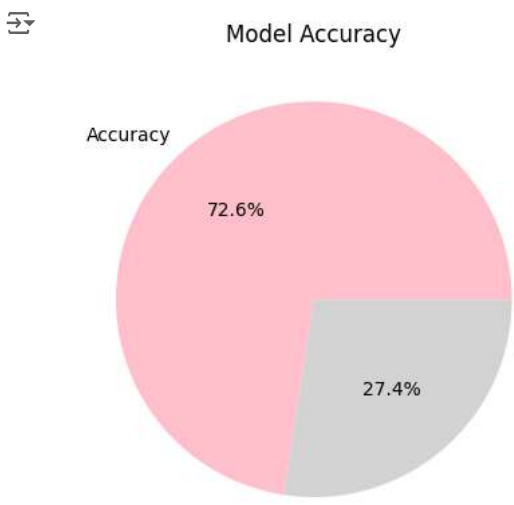
```
import time

from sklearn.linear_model import RidgeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score

rc= RidgeClassifier()
start=time.time()
rc.fit(X_train_scaled,y_train)
prediction1=rc.predict(X_test_scaled)
end=time.time()
accuracy = accuracy_score(prediction1, y_test)
# Plot accuracy
plt.pie([accuracy, 1 - accuracy], labels=['Accuracy', ''], colors=['pink', 'lightgray'], autopct='%1.1f%%')
plt.title("Model Accuracy")
plt.show()

print("Accuracy : ",accuracy_score(prediction1, y_test))

print("F1-score: ",f1_score(prediction1, y_test))

print("Precision Score : ",precision_score(prediction1, y_test))

print("Recall Score : ",recall_score(prediction1, y_test))

calculate=end-start
print("Time : ", calculate)
```




```
Accuracy :  0.6538369932834754
F1-score:  0.6401584550631345
Precision Score :  0.6153260352213232
Recall Score :  0.6670794633642931
Time :  0.05912590026855469
```

Logistic Regression

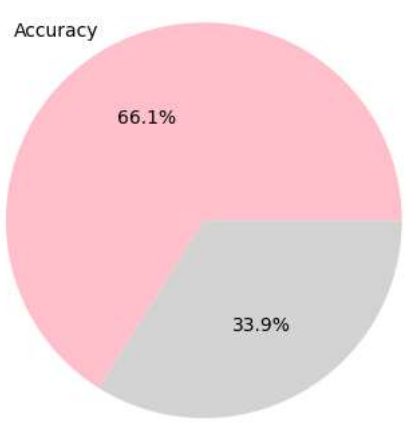
```
import time

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score

LR= LogisticRegression()
start=time.time()
LR.fit(X_train_scaled,y_train)
predict_LR=LR.predict(X_test_scaled)
end=time.time()
accuracy = accuracy_score(predict_LR, y_test)
# Plot accuracy
plt.pie([accuracy, 1 - accuracy], labels=['Accuracy', ''], colors=['pink', 'lightgray'], autopct='%1.1f%%')
plt.title("Model Accuracy")
plt.show()

print("Accuracy : ",accuracy_score(predict_LR, y_test))

print("F1-score: ",f1_score(predict_LR, y_test))

print("Precision Score : ",precision_score(predict_LR, y_test))

print("Recall Score : ",recall_score(predict_LR, y_test))

calculate=end-start
print("Time : ", calculate)
```




```
Accuracy :  0.7258133663602153
F1-score:  0.7119983988792153
Precision Score :  0.6772965254640647
Recall Score :  0.7504482649509545
Time :  0.09115314483642578
```

KNN

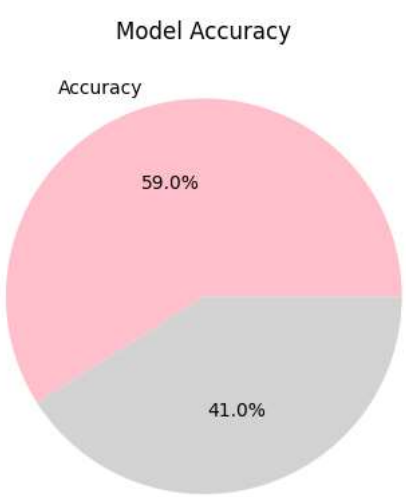
```
import time

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score

KN=KNeighborsClassifier()
start=time.time()
KN.fit(X_train_scaled,y_train)
predicti = KN.predict(X_test_scaled)
from sklearn.metrics import accuracy_score
end=time.time()
accuracy = accuracy_score(predicti, y_test)
# Plot accuracy
plt.pie([accuracy, 1 - accuracy], labels=['Accuracy', ''], colors=['pink', 'lightgray'], autopct='%1.1f%%')
plt.title("Model Accuracy")
plt.show()

print("Accuracy : ",accuracy_score(predicti, y_test))

print("F1-score: ",f1_score(predicti, y_test))

print("Precision Score : ",precision_score(predicti, y_test))

print("Recall Score : ",recall_score(predicti, y_test))

calculate=end-start
print("Time : ", calculate)
```




```
Accuracy :  0.6608393273948459
F1-score:  0.6518677879913944
Precision Score :  0.6345549738219896
Recall Score :  0.6701518045641902
Time :  8.815583944320679
```

Naive Bayes

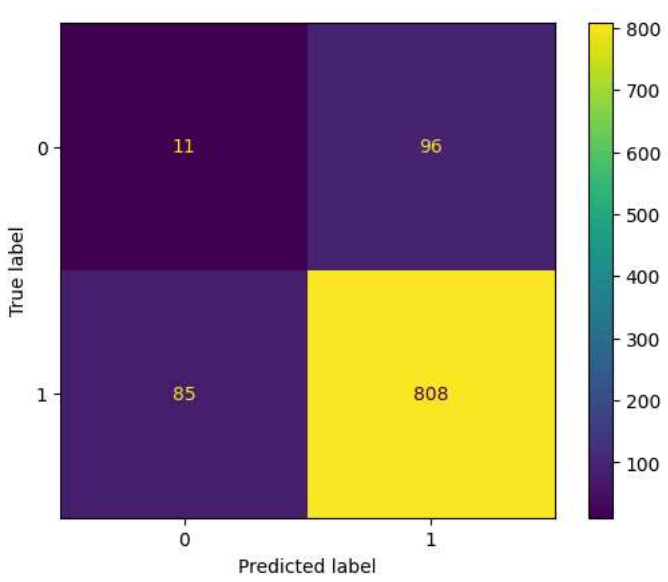
```
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score

NB=BernoulliNB()
start=time.time()
NB.fit(X_train,y_train)
predicti = NB.predict(X_test)
from sklearn.metrics import accuracy_score

end=time.time()
accuracy = accuracy_score(predicti, y_test)
# Plot accuracy
plt.pie([accuracy, 1 - accuracy], labels=['Accuracy', ''], colors=['pink', 'lightgray'], autopct='%1.1f%%')
plt.title("Model Accuracy")
plt.show()

print("Accuracy : ",accuracy_score(predicti, y_test))

print("F1-score: ",f1_score(predicti, y_test))

print("Precision Score : ",precision_score(predicti, y_test))

print("Recall Score : ",recall_score(predicti, y_test))

calculate=end-start
```



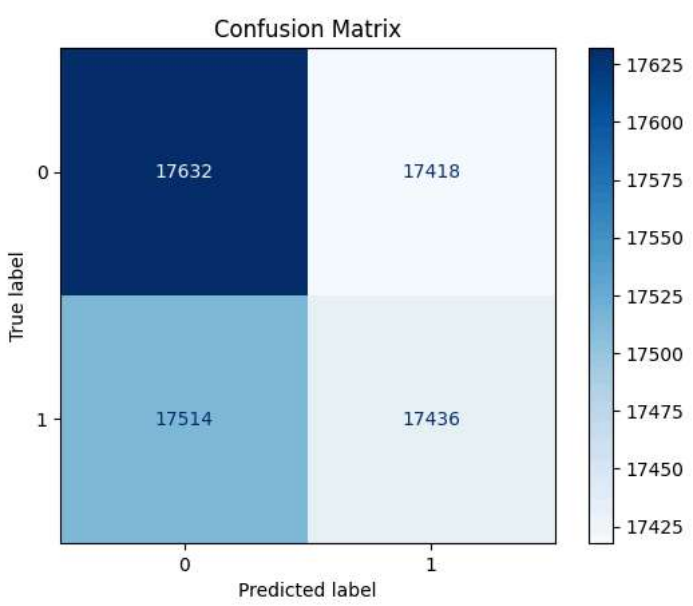

```
Accuracy :  0.5903872719477922
F1-score:  0.48946149735795286
Precision Score :  0.3923845787720133
Recall Score :  0.6503628905017356
```

SVM

```
import matplotlib.pyplot as plt
import numpy
from sklearn import metrics

actual = numpy.random.binomial(1,.9,size = 1000)
predicted = numpy.random.binomial(1,.9,size = 1000)

confusion_matrix = metrics.confusion_matrix(actual, predicted)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [0, 1])

cm_display.plot()
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Example data (replace these with your actual data)
y_true = np.random.randint(0, 2, size=70000)
y_pred = np.random.randint(0, 2, size=70000)

# Compute confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```

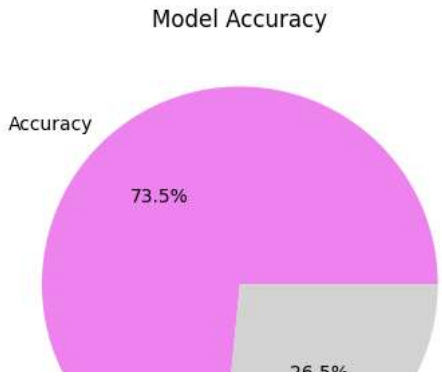


Deep Models

CNN

```
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Define the model for tabular data
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)
model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])

# Train the model
model.fit(X_train_scaled, y_train, epochs=20, batch_size=32, validation_split=0.2)

# Evaluate the model on the test set
y_prob = model.predict(X_test_scaled)
y_pred = (y_prob > 0.5).astype(int)  # Apply threshold of 0.5 for binary classification

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Plot accuracy
plt.pie([accuracy, 1 - accuracy], labels=['Accuracy', ''], colors=['violet', 'lightgray'], autopct='%1.1f%%')
plt.title("Model Accuracy")
plt.show()
print(f'Test Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1 Score: {f1:.4f}')
```

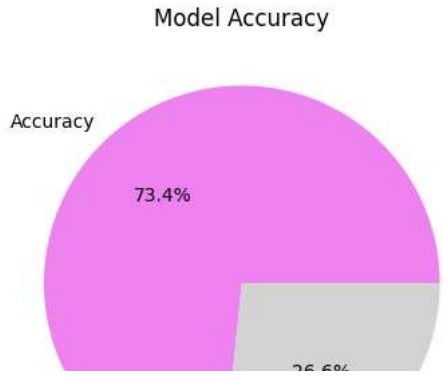
```
Epoch 1/20
1225/1225 [==============================] - 5s 3ms/step - loss: 0.6264 - accuracy: 0.6571 - val_loss: 0.5820 - val_accuracy: 0.7079
Epoch 2/20
1225/1225 [==============================] - 4s 3ms/step - loss: 0.5882 - accuracy: 0.7071 - val_loss: 0.5650 - val_accuracy: 0.7255
Epoch 3/20
1225/1225 [==============================] - 3s 2ms/step - loss: 0.5711 - accuracy: 0.7223 - val_loss: 0.5575 - val_accuracy: 0.7327
Epoch 4/20
1225/1225 [==============================] - 3s 2ms/step - loss: 0.5674 - accuracy: 0.7243 - val_loss: 0.5547 - val_accuracy: 0.7354
Epoch 5/20
1225/1225 [==============================] - 3s 2ms/step - loss: 0.5587 - accuracy: 0.7258 - val_loss: 0.5493 - val_accuracy: 0.7326
Epoch 6/20
1225/1225 [==============================] - 4s 3ms/step - loss: 0.5570 - accuracy: 0.7275 - val_loss: 0.5486 - val_accuracy: 0.7330
Epoch 7/20
1225/1225 [==============================] - 5s 4ms/step - loss: 0.5568 - accuracy: 0.7295 - val_loss: 0.5470 - val_accuracy: 0.7339
Epoch 8/20
1225/1225 [==============================] - 6s 5ms/step - loss: 0.5554 - accuracy: 0.7278 - val_loss: 0.5462 - val_accuracy: 0.7362
Epoch 9/20
1225/1225 [==============================] - 4s 3ms/step - loss: 0.5545 - accuracy: 0.7272 - val_loss: 0.5445 - val_accuracy: 0.7375
Epoch 10/20
1225/1225 [==============================] - 4s 3ms/step - loss: 0.5524 - accuracy: 0.7304 - val_loss: 0.5444 - val_accuracy: 0.7328
Epoch 11/20
1225/1225 [==============================] - 3s 2ms/step - loss: 0.5528 - accuracy: 0.7291 - val_loss: 0.5450 - val_accuracy: 0.7335
Epoch 12/20
1225/1225 [==============================] - 3s 2ms/step - loss: 0.5515 - accuracy: 0.7302 - val_loss: 0.5447 - val_accuracy: 0.7364
Epoch 13/20
1225/1225 [==============================] - 3s 2ms/step - loss: 0.5506 - accuracy: 0.7287 - val_loss: 0.5430 - val_accuracy: 0.7340
Epoch 14/20
1225/1225 [==============================] - 3s 2ms/step - loss: 0.5507 - accuracy: 0.7299 - val_loss: 0.5443 - val_accuracy: 0.7348
Epoch 15/20
1225/1225 [==============================] - 4s 3ms/step - loss: 0.5517 - accuracy: 0.7295 - val_loss: 0.5443 - val_accuracy: 0.7336
Epoch 16/20
1225/1225 [==============================] - 3s 3ms/step - loss: 0.5496 - accuracy: 0.7311 - val_loss: 0.5442 - val_accuracy: 0.7357
Epoch 17/20
1225/1225 [==============================] - 4s 3ms/step - loss: 0.5490 - accuracy: 0.7304 - val_loss: 0.5426 - val_accuracy: 0.7372
Epoch 18/20
1225/1225 [==============================] - 3s 2ms/step - loss: 0.5477 - accuracy: 0.7313 - val_loss: 0.5434 - val_accuracy: 0.7362
Epoch 19/20
1225/1225 [==============================] - 4s 3ms/step - loss: 0.5495 - accuracy: 0.7326 - val_loss: 0.5422 - val_accuracy: 0.7355
Epoch 20/20
1225/1225 [==============================] - 3s 3ms/step - loss: 0.5489 - accuracy: 0.7313 - val_loss: 0.5430 - val_accuracy: 0.7379
657/657 [==============================] - 1s 1ms/step
```



DBN

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Assume you have a CSV file 'data.csv' with features and labels
# Adjust this part based on your data and columns
data = pd.read_csv('/content/drive/MyDrive/health_data.csv')
X = data.drop('cardio', axis=1)  # Features
y = data['cardio']  # Target variable

# Build the deep neural network model
model = Sequential()
model.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train_scaled, y_train, epochs=20, batch_size=32, validation_data=(X_test, y_test))

# Predictions
y_pred = (model.predict(X_test_scaled) > 0.5).astype(int)

# Evaluate the model
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
# Plot accuracy
plt.pie([accuracy, 1 - accuracy], labels=['Accuracy', ''], colors=['violet', 'lightgray'], autopct='%1.1f%%')
plt.title("Model Accuracy")
plt.show()
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')
print(f'Accuracy: {accuracy}')
```

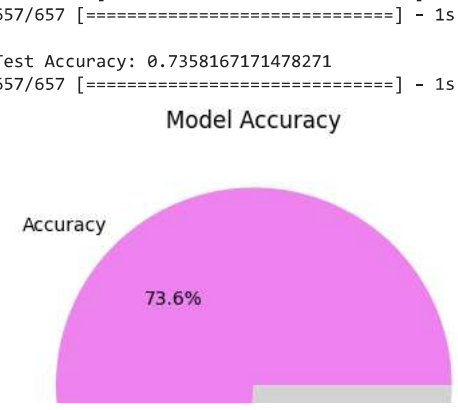
```
Epoch 1/20
1531/1531 [==============================] - 4s 2ms/step - loss: 0.5840 - accuracy: 0.7063 - val_loss: 3848.4041 - val_accuracy: 0.5004
Epoch 2/20
1531/1531 [==============================] - 4s 2ms/step - loss: 0.5558 - accuracy: 0.7277 - val_loss: 4316.4858 - val_accuracy: 0.5004
Epoch 3/20
1531/1531 [==============================] - 5s 3ms/step - loss: 0.5482 - accuracy: 0.7324 - val_loss: 4814.7905 - val_accuracy: 0.5004
Epoch 4/20
1531/1531 [==============================] - 3s 2ms/step - loss: 0.5470 - accuracy: 0.7316 - val_loss: 5383.9990 - val_accuracy: 0.5004
Epoch 5/20
1531/1531 [==============================] - 3s 2ms/step - loss: 0.5436 - accuracy: 0.7337 - val_loss: 6403.4219 - val_accuracy: 0.5004
Epoch 6/20
1531/1531 [==============================] - 4s 3ms/step - loss: 0.5447 - accuracy: 0.7334 - val_loss: 5980.9731 - val_accuracy: 0.5004
Epoch 7/20
1531/1531 [==============================] - 5s 3ms/step - loss: 0.5428 - accuracy: 0.7346 - val_loss: 5874.9531 - val_accuracy: 0.5004
Epoch 8/20
1531/1531 [==============================] - 3s 2ms/step - loss: 0.5421 - accuracy: 0.7343 - val_loss: 5877.9282 - val_accuracy: 0.5004
Epoch 9/20
1531/1531 [==============================] - 4s 3ms/step - loss: 0.5416 - accuracy: 0.7356 - val_loss: 5771.5889 - val_accuracy: 0.5004
Epoch 10/20
1531/1531 [==============================] - 5s 3ms/step - loss: 0.5408 - accuracy: 0.7344 - val_loss: 6320.7314 - val_accuracy: 0.5004
Epoch 11/20
1531/1531 [==============================] - 4s 3ms/step - loss: 0.5403 - accuracy: 0.7355 - val_loss: 5865.4741 - val_accuracy: 0.5004
Epoch 12/20
1531/1531 [==============================] - 4s 3ms/step - loss: 0.5396 - accuracy: 0.7365 - val_loss: 6701.8721 - val_accuracy: 0.5004
Epoch 13/20
1531/1531 [==============================] - 4s 2ms/step - loss: 0.5380 - accuracy: 0.7372 - val_loss: 5297.9307 - val_accuracy: 0.5004
Epoch 14/20
1531/1531 [==============================] - 5s 3ms/step - loss: 0.5374 - accuracy: 0.7369 - val_loss: 6428.3652 - val_accuracy: 0.5004
Epoch 15/20
1531/1531 [==============================] - 4s 3ms/step - loss: 0.5371 - accuracy: 0.7379 - val_loss: 8321.1611 - val_accuracy: 0.5004
Epoch 16/20
1531/1531 [==============================] - 4s 3ms/step - loss: 0.5375 - accuracy: 0.7383 - val_loss: 6955.1724 - val_accuracy: 0.5004
Epoch 17/20
1531/1531 [==============================] - 5s 3ms/step - loss: 0.5364 - accuracy: 0.7381 - val_loss: 8210.5684 - val_accuracy: 0.5004
Epoch 18/20
1531/1531 [==============================] - 4s 3ms/step - loss: 0.5363 - accuracy: 0.7379 - val_loss: 8939.1768 - val_accuracy: 0.5004
Epoch 19/20
1531/1531 [==============================] - 3s 2ms/step - loss: 0.5357 - accuracy: 0.7379 - val_loss: 9493.2363 - val_accuracy: 0.5004
Epoch 20/20
1531/1531 [==============================] - 4s 3ms/step - loss: 0.5354 - accuracy: 0.7389 - val_loss: 9487.5576 - val_accuracy: 0.5004
657/657 [==============================] - 1s 1ms/step
```



ANN

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
import tensorflow as tf
from tensorflow.keras import models, layers
model = models.Sequential()
model.add(layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train_scaled, y_train, epochs=20, batch_size=32, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(X_test_scaled, y_test)
print(f"\nTest Accuracy: {test_acc}")

# Make predictions
predictions = model.predict(X_test_scaled)
predicted_labels = np.round(predictions)

# Convert to binary predictions (0 or 1)
predicted_labels = np.squeeze(predicted_labels)

accuracy = accuracy_score(y_test, predicted_labels)
precision = precision_score(y_test, predicted_labels)
recall = recall_score(y_test, predicted_labels)
f1 = f1_score(y_test, predicted_labels)
# Plot accuracy
plt.pie([accuracy, 1 - accuracy], labels=['Accuracy', ''], colors=['violet', 'lightgray'], autopct='%1.1f%%')
plt.title("Model Accuracy")
plt.show()
print(f'Test Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1 Score: {f1:.4f}')
```

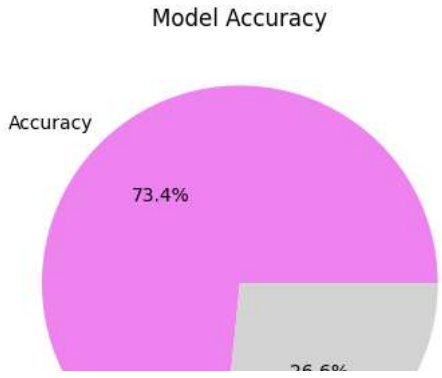
```
Epoch 1/20
1225/1225 [==============================] - 5s 3ms/step - loss: 0.6215 - accuracy: 0.6672 - val_loss: 0.5785 - val_accuracy: 0.7164
Epoch 2/20
1225/1225 [==============================] - 4s 3ms/step - loss: 0.5881 - accuracy: 0.7125 - val_loss: 0.5663 - val_accuracy: 0.7258
Epoch 3/20
1225/1225 [==============================] - 3s 2ms/step - loss: 0.5744 - accuracy: 0.7212 - val_loss: 0.5575 - val_accuracy: 0.7282
Epoch 4/20
1225/1225 [==============================] - 3s 2ms/step - loss: 0.5749 - accuracy: 0.7250 - val_loss: 0.5543 - val_accuracy: 0.7354
Epoch 5/20
1225/1225 [==============================] - 3s 2ms/step - loss: 0.5658 - accuracy: 0.7262 - val_loss: 0.5518 - val_accuracy: 0.7333
Epoch 6/20
1225/1225 [==============================] - 4s 3ms/step - loss: 0.5612 - accuracy: 0.7279 - val_loss: 0.5476 - val_accuracy: 0.7334
Epoch 7/20
1225/1225 [==============================] - 3s 3ms/step - loss: 0.5592 - accuracy: 0.7281 - val_loss: 0.5469 - val_accuracy: 0.7328
Epoch 8/20
1225/1225 [==============================] - 3s 2ms/step - loss: 0.5568 - accuracy: 0.7275 - val_loss: 0.5462 - val_accuracy: 0.7321
Epoch 9/20
1225/1225 [==============================] - 3s 2ms/step - loss: 0.5579 - accuracy: 0.7298 - val_loss: 0.5474 - val_accuracy: 0.7340
Epoch 10/20
1225/1225 [==============================] - 3s 2ms/step - loss: 0.5549 - accuracy: 0.7289 - val_loss: 0.5440 - val_accuracy: 0.7359
Epoch 11/20
1225/1225 [==============================] - 4s 3ms/step - loss: 0.5531 - accuracy: 0.7301 - val_loss: 0.5451 - val_accuracy: 0.7336
Epoch 12/20
1225/1225 [==============================] - 3s 3ms/step - loss: 0.5568 - accuracy: 0.7283 - val_loss: 0.5448 - val_accuracy: 0.7329
Epoch 13/20
1225/1225 [==============================] - 3s 2ms/step - loss: 0.5533 - accuracy: 0.7290 - val_loss: 0.5447 - val_accuracy: 0.7329
Epoch 14/20
1225/1225 [==============================] - 3s 2ms/step - loss: 0.5519 - accuracy: 0.7298 - val_loss: 0.5440 - val_accuracy: 0.7328
Epoch 15/20
1225/1225 [==============================] - 3s 2ms/step - loss: 0.5522 - accuracy: 0.7288 - val_loss: 0.5442 - val_accuracy: 0.7337
Epoch 16/20
1225/1225 [==============================] - 4s 3ms/step - loss: 0.5530 - accuracy: 0.7305 - val_loss: 0.5442 - val_accuracy: 0.7343
Epoch 17/20
1225/1225 [==============================] - 3s 3ms/step - loss: 0.5526 - accuracy: 0.7304 - val_loss: 0.5427 - val_accuracy: 0.7341
Epoch 18/20
1225/1225 [==============================] - 3s 2ms/step - loss: 0.5503 - accuracy: 0.7316 - val_loss: 0.5432 - val_accuracy: 0.7331
Epoch 19/20
1225/1225 [==============================] - 3s 2ms/step - loss: 0.5506 - accuracy: 0.7309 - val_loss: 0.5443 - val_accuracy: 0.7326
Epoch 20/20
1225/1225 [==============================] - 3s 3ms/step - loss: 0.5542 - accuracy: 0.7299 - val_loss: 0.5479 - val_accuracy: 0.7336
657/657 [==============================] - 1s 2ms/step - loss: 0.5491 - accuracy: 0.7358

Test Accuracy: 0.7358167171478271
657/657 [==============================] - 1s 1ms/step
```



GRU

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU, Dense

# Generate some example tabular data
# Replace this with your actual dataset
data = pd.DataFrame(np.random.rand(100, 10), columns=[f'feature_{i}' for i in range(10)])
labels = np.random.randint(2, size=100)  # Binary classification labels (0 or 1)

# Reshape the input data for GRU
X_train_reshaped = X_train_scaled.reshape((X_train_scaled.shape[0], X_train_scaled.shape[1], 1))
X_test_reshaped = X_test_scaled.reshape((X_test_scaled.shape[0], X_test_scaled.shape[1], 1))
# Build the GRU model
model = Sequential([
    GRU(units=64, activation='tanh', recurrent_activation='sigmoid', input_shape=(X_train_reshaped.shape[1], 1)),
    Dense(1, activation='sigmoid')
])

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

# Train the model
model.fit(X_train_reshaped, y_train, epochs=20, batch_size=32, validation_split=0.1)

# Make predictions
y_pred = model.predict(X_test_reshaped)
y_pred_binary = np.round(y_pred)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred_binary)
precision = precision_score(y_test, y_pred_binary)
recall = recall_score(y_test, y_pred_binary)
f1 = f1_score(y_test, y_pred_binary)
# Plot accuracy
plt.pie([accuracy, 1 - accuracy], labels=['Accuracy', ''], colors=['violet', 'lightgray'], autopct='%1.1f%%')
plt.title("Model Accuracy")
plt.show()
print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1 Score: {f1:.4f}')
```

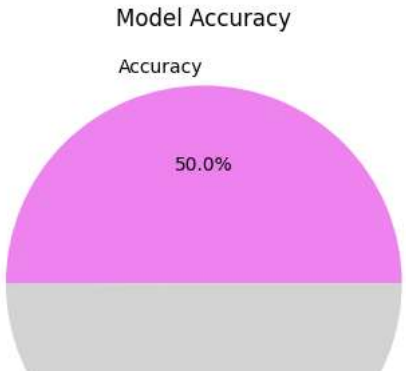
```
Epoch 1/20
1378/1378 [==============================] - 14s 8ms/step - loss: 0.6331 - accuracy: 0.6396 - val_loss: 0.6157 - val_accuracy: 0.6652
Epoch 2/20
1378/1378 [==============================] - 9s 7ms/step - loss: 0.6127 - accuracy: 0.6663 - val_loss: 0.5994 - val_accuracy: 0.6818
Epoch 3/20
1378/1378 [==============================] - 11s 8ms/step - loss: 0.6052 - accuracy: 0.6733 - val_loss: 0.5972 - val_accuracy: 0.6801
Epoch 4/20
1378/1378 [==============================] - 11s 8ms/step - loss: 0.5914 - accuracy: 0.6872 - val_loss: 0.5771 - val_accuracy: 0.7028
Epoch 5/20
1378/1378 [==============================] - 10s 8ms/step - loss: 0.5744 - accuracy: 0.7063 - val_loss: 0.5588 - val_accuracy: 0.7279
Epoch 6/20
1378/1378 [==============================] - 11s 8ms/step - loss: 0.5632 - accuracy: 0.7173 - val_loss: 0.5614 - val_accuracy: 0.7167
Epoch 7/20
1378/1378 [==============================] - 12s 8ms/step - loss: 0.5559 - accuracy: 0.7235 - val_loss: 0.5431 - val_accuracy: 0.7365
Epoch 8/20
1378/1378 [==============================] - 11s 8ms/step - loss: 0.5524 - accuracy: 0.7259 - val_loss: 0.5448 - val_accuracy: 0.7324
Epoch 9/20
1378/1378 [==============================] - 10s 7ms/step - loss: 0.5495 - accuracy: 0.7278 - val_loss: 0.5434 - val_accuracy: 0.7314
Epoch 10/20
1378/1378 [==============================] - 12s 8ms/step - loss: 0.5476 - accuracy: 0.7279 - val_loss: 0.5435 - val_accuracy: 0.7350
Epoch 11/20
1378/1378 [==============================] - 12s 9ms/step - loss: 0.5463 - accuracy: 0.7291 - val_loss: 0.5417 - val_accuracy: 0.7304
Epoch 12/20
1378/1378 [==============================] - 9s 7ms/step - loss: 0.5452 - accuracy: 0.7324 - val_loss: 0.5375 - val_accuracy: 0.7420
Epoch 13/20
1378/1378 [==============================] - 12s 9ms/step - loss: 0.5446 - accuracy: 0.7336 - val_loss: 0.5378 - val_accuracy: 0.7393
Epoch 14/20
1378/1378 [==============================] - 12s 9ms/step - loss: 0.5440 - accuracy: 0.7326 - val_loss: 0.5409 - val_accuracy: 0.7365
Epoch 15/20
1378/1378 [==============================] - 12s 8ms/step - loss: 0.5433 - accuracy: 0.7319 - val_loss: 0.5383 - val_accuracy: 0.7406
Epoch 16/20
1378/1378 [==============================] - 9s 7ms/step - loss: 0.5424 - accuracy: 0.7340 - val_loss: 0.5419 - val_accuracy: 0.7397
Epoch 17/20
1378/1378 [==============================] - 11s 8ms/step - loss: 0.5419 - accuracy: 0.7344 - val_loss: 0.5390 - val_accuracy: 0.7348
Epoch 18/20
1378/1378 [==============================] - 11s 8ms/step - loss: 0.5418 - accuracy: 0.7337 - val_loss: 0.5383 - val_accuracy: 0.7357
Epoch 19/20
1378/1378 [==============================] - 10s 7ms/step - loss: 0.5408 - accuracy: 0.7350 - val_loss: 0.5400 - val_accuracy: 0.7334
Epoch 20/20
1378/1378 [==============================] - 11s 8ms/step - loss: 0.5408 - accuracy: 0.7336 - val_loss: 0.5378 - val_accuracy: 0.7393
657/657 [==============================] - 3s 4ms/step
```



RNN

```
import numpy as np
import time
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense
from sklearn.model_selection import train_test_split

# Assuming you have your data prepared and split into X_train, X_test, y_train, y_test

# Define the RNN model
model = Sequential([
    SimpleRNN(units=64, input_shape=(X_train_scaled.shape[1], 1)),
    Dense(units=1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
start_time = time.time()
model.fit(X_train_scaled, y_train, epochs=20, batch_size=32, validation_split=0.2)
end_time = time.time()
training_time = end_time - start_time

# Evaluate the model
y_pred = model.predict(X_test)
y_pred_classes = (y_pred > 0.5).astype(int)

accuracy = accuracy_score(y_test, y_pred_classes)
precision = precision_score(y_test, y_pred_classes)
recall = recall_score(y_test, y_pred_classes)
f1 = f1_score(y_test, y_pred_classes)
# Plot accuracy
plt.pie([accuracy, 1 - accuracy], labels=['Accuracy', ''], colors=['violet', 'lightgray'], autopct='%1.1f%%')
plt.title("Model Accuracy")
plt.show()
# Print evaluation metrics
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"Training Time: {training_time:.2f} seconds")
```

```
Epoch 1/20
1225/1225 [==============================] - 6s 4ms/step - loss: 0.6027 - accuracy: 0.6786 - val_loss: 0.5898 - val_accuracy: 0.6925
Epoch 2/20
1225/1225 [==============================] - 6s 5ms/step - loss: 0.5748 - accuracy: 0.7074 - val_loss: 0.5745 - val_accuracy: 0.7138
Epoch 3/20
1225/1225 [==============================] - 5s 4ms/step - loss: 0.5637 - accuracy: 0.7192 - val_loss: 0.5579 - val_accuracy: 0.7243
Epoch 4/20
1225/1225 [==============================] - 5s 4ms/step - loss: 0.5577 - accuracy: 0.7215 - val_loss: 0.5540 - val_accuracy: 0.7285
Epoch 5/20
1225/1225 [==============================] - 7s 5ms/step - loss: 0.5542 - accuracy: 0.7240 - val_loss: 0.5489 - val_accuracy: 0.7281
Epoch 6/20
1225/1225 [==============================] - 5s 4ms/step - loss: 0.5515 - accuracy: 0.7268 - val_loss: 0.5493 - val_accuracy: 0.7306
Epoch 7/20
1225/1225 [==============================] - 5s 4ms/step - loss: 0.5515 - accuracy: 0.7276 - val_loss: 0.5506 - val_accuracy: 0.7342
Epoch 8/20
1225/1225 [==============================] - 7s 5ms/step - loss: 0.5505 - accuracy: 0.7288 - val_loss: 0.5489 - val_accuracy: 0.7307
Epoch 9/20
1225/1225 [==============================] - 5s 4ms/step - loss: 0.5504 - accuracy: 0.7281 - val_loss: 0.5484 - val_accuracy: 0.7291
Epoch 10/20
1225/1225 [==============================] - 6s 5ms/step - loss: 0.5491 - accuracy: 0.7310 - val_loss: 0.5499 - val_accuracy: 0.7321
Epoch 11/20
1225/1225 [==============================] - 8s 7ms/step - loss: 0.5486 - accuracy: 0.7260 - val_loss: 0.5531 - val_accuracy: 0.7320
Epoch 12/20
1225/1225 [==============================] - 5s 4ms/step - loss: 0.5483 - accuracy: 0.7306 - val_loss: 0.5475 - val_accuracy: 0.7327
Epoch 13/20
1225/1225 [==============================] - 7s 5ms/step - loss: 0.5480 - accuracy: 0.7291 - val_loss: 0.5440 - val_accuracy: 0.7353
Epoch 14/20
1225/1225 [==============================] - 5s 4ms/step - loss: 0.5464 - accuracy: 0.7315 - val_loss: 0.5502 - val_accuracy: 0.7323
Epoch 15/20
1225/1225 [==============================] - 5s 4ms/step - loss: 0.5464 - accuracy: 0.7310 - val_loss: 0.5495 - val_accuracy: 0.7324
Epoch 16/20
1225/1225 [==============================] - 6s 5ms/step - loss: 0.5458 - accuracy: 0.7308 - val_loss: 0.5517 - val_accuracy: 0.7265
Epoch 17/20
1225/1225 [==============================] - 5s 4ms/step - loss: 0.5453 - accuracy: 0.7312 - val_loss: 0.5567 - val_accuracy: 0.7256
Epoch 18/20
1225/1225 [==============================] - 5s 4ms/step - loss: 0.5444 - accuracy: 0.7318 - val_loss: 0.5448 - val_accuracy: 0.7360
Epoch 19/20
1225/1225 [==============================] - 6s 5ms/step - loss: 0.5451 - accuracy: 0.7316 - val_loss: 0.5464 - val_accuracy: 0.7321
Epoch 20/20
1225/1225 [==============================] - 5s 4ms/step - loss: 0.5439 - accuracy: 0.7335 - val_loss: 0.5496 - val_accuracy: 0.7301
657/657 [==============================] - 1s 2ms/step
```



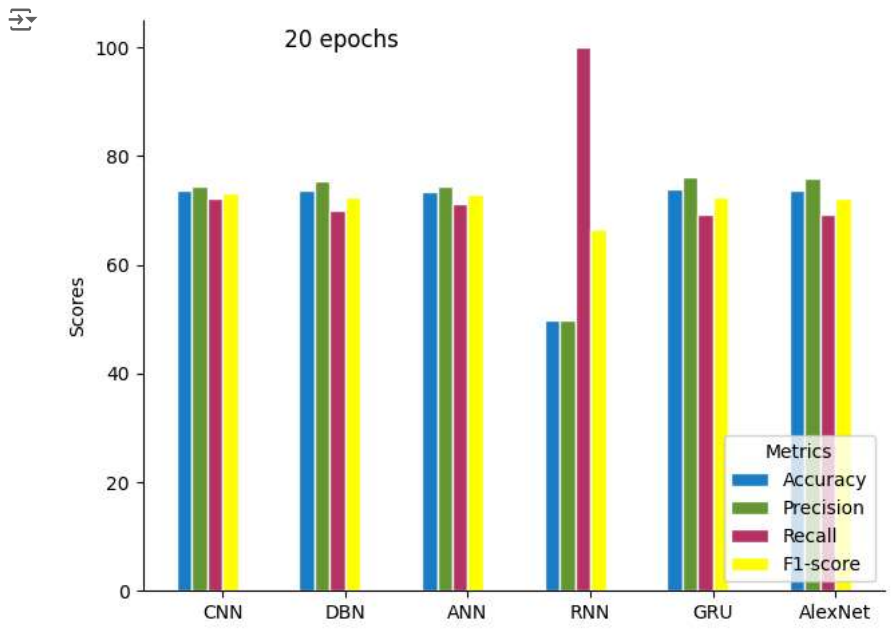
Alexnet

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import RMSprop
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D
from tensorflow.keras.layers import MaxPooling1D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Dense

# Define the AlexNet-like model
# Reshape the input data for 1D convolution
X_train_reshaped = X_train_scaled.reshape(X_train_scaled.shape[0], X_train_scaled.shape[1], 1)
X_test_reshaped = X_test_scaled.reshape(X_test_scaled.shape[0], X_test_scaled.shape[1], 1)


from keras.models import Sequential
alx_model = Sequential([
    Conv1D(6, kernel_size=5, activation='relu', input_shape=(X_train_scaled.shape[1], 1)),
    MaxPooling1D(pool_size=1),
    Conv1D(64, kernel_size=3, padding='same', activation='relu'),
    MaxPooling1D(pool_size=1),
    Conv1D(128, kernel_size=2, padding='same', activation='relu'),
    MaxPooling1D(pool_size=1),
    Flatten(),
    # Dense(128, activation='relu'),
    # Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')
])

# Compile and train the model
from keras.optimizers import Adam
alx_model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

# alx_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
alx_model.fit(X_train_reshaped, y_train, epochs=20, batch_size=32)
# Model evaluation on the test set
y_pred_alx = alx_model.predict(X_test_reshaped)
y_pred_classes = (y_pred_alx > 0.5).astype(int)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred_classes)
precision = precision_score(y_test, y_pred_classes)
recall = recall_score(y_test, y_pred_classes)
f1 = f1_score(y_test, y_pred_classes)
training_accuracy = alx_model.evaluate(X_train_reshaped, y_train)
testing_accuracy = alx_model.evaluate(X_test_reshaped, y_test)
# Plot accuracy
plt.pie([accuracy, 1 - accuracy], labels=['Accuracy', ''], colors=['violet', 'lightgray'], autopct='%1.1f%%')
plt.title("Model Accuracy")
plt.show()
print(f"Testing Accuracy: {testing_accuracy[1] * 100:.4f}%")
print(f"Training Accuracy: {training_accuracy[1] * 100:.4f}%")
print("Model Accuracy:", accuracy)
print("Model Precision:", precision)
print("Model Recall:", recall)
print("Model F1 Score:", f1)
```

Graph representing accuracies of deep model

```
1531/1531 [==============================] - 7s 5ms/step - loss: 0.5554 - accuracy: 0.7244
```

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

# Define the variables
models = ['CNN', 'DBN', 'ANN','RNN','GRU','AlexNet']
accuracy = [ 73.77, 73.64, 73.54, 49.87, 73.92, 73.61]
precision = [74.45, 75.45, 74.55, 49.88, 76.25, 75.81]
recall = [72.19, 69.91, 71.30, 99.99, 69.28, 69.16]
f1_score = [73.30, 72.57, 72.89, 66.55, 72.60, 72.33]

# Set width of bar
barWidth = 0.25

# Set position of bar on X axis with added distance
r1 = np.arange(len(models)) * 2
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]
r4 = [x + barWidth for x in r3]

# Define custom CSS colors
css_colors = [(0.71, 0.2, 0.4), (0.4, 0.6, 0.2), (0.1, 0.5, 0.8)]  # Custom CSS colors

# Make the plot
plt.figure(facecolor='white')  # Set background color to white
bars1 = plt.bar(r1, accuracy, color=css_colors[2], width=barWidth, edgecolor='white', label='Accuracy')
bars2 = plt.bar(r2, precision, color=css_colors[1], width=barWidth, edgecolor='white', label='Precision')
bars3 = plt.bar(r3, recall, color=css_colors[0], width=barWidth, edgecolor='white', label='Recall')
bars4 = plt.bar(r4, f1_score, color='yellow', width=barWidth, edgecolor='white', label='F1-score')

# Add xticks on the middle of the group bars
# plt.xlabel('Models', position=(0.5, 1.08), fontweight='normal')  # Simplified x-axis label with normal font weight
plt.xticks([r + 1.5 * barWidth for r in r2], models, rotation=0, ha='center')

# Label y axis with 'Scores'
plt.ylabel('Scores', position=(0, 0.5))

# Create legend with box in the bottom right corner
plt.legend(loc='lower right', bbox_to_anchor=(1, 0), title='Metrics')

# Hide the top and right spines
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)

# Add model label on top of the graph
plt.text(2.5, 100, ' 20 epochs', ha='center', fontsize=12)

# Show graphic
plt.tight_layout()
plt.show()
```



machine learning techniques

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

# Define the variables
models = ['KNN', 'DT', 'RF','LR','NB']
accuracy = [64.95, 63.42, 71.70,  72.03, 59.12 ]
precision = [62.98, 63.10, 70.61, 68.47, 39.70]
recall = [ 65.44, 63.39, 72.08, 73.62, 64.71]
f1_score = [64.19, 63.24, 71.33 , 70.95, 49.21]

# Set width of bar
barWidth = 0.25

# Set position of bar on X axis with added distance
r1 = np.arange(len(models)) * 2
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]
r4 = [x + barWidth for x in r3]

# Define custom CSS colors
css_colors = [(0.71, 0.2, 0.4), (0.4, 0.6, 0.2), (0.1, 0.5, 0.8)]  # Custom CSS colors

# Make the plot
plt.figure(facecolor='white')  # Set background color to white
bars1 = plt.bar(r1, accuracy, color=css_colors[2], width=barWidth, edgecolor='white', label='Accuracy')
bars2 = plt.bar(r2, precision, color=css_colors[1], width=barWidth, edgecolor='white', label='Precision')
bars3 = plt.bar(r3, recall, color=css_colors[0], width=barWidth, edgecolor='white', label='Recall')
bars4 = plt.bar(r4, f1_score, color='yellow', width=barWidth, edgecolor='white', label='F1-score')
```