```
from google.colab import files
upload = files.upload()
```

```
Choose Files age_predict...cleaned.csv
• age_predictions_cleaned.csv(text/csv) - 205351 bytes, last modified: 2/6/2025 - 100% done
Saving age_predictions_cleaned.csv to age_predictions_cleaned.csv
```

```
from google.colab import files
upload = files.upload()
```

```
Choose Files dataPrep.py
• dataPrep.py(n/a) - 472 bytes, last modified: 2/6/2025 - 100% done
Saving dataPrep.py to dataPrep.py
```

```
import pandas as pd
df = pd.read_csv('age_predictions_cleaned.csv')
df
```

	PAQ605	BMXBMI	LBXGLU	LBXGLT	LBXIN	age_group
0	2.00000	1.393867	110.000000	150.000000	14.910000	0
1	2.00000	1.099498	89.000000	80.000000	3.850000	0
2	2.00000	0.629968	89.000000	68.000000	6.140000	0
3	2.00000	0.292901	104.000000	84.000000	16.150000	0
4	1.00000	1.426249	103.000000	81.000000	10.920000	0
...	...	...	...	...	...	...
3519	1.02742	0.616286	102.219362	164.917739	20.862093	1
3520	2.00000	0.233323	95.785948	80.607026	4.404918	1
3521	2.00000	0.855922	94.000000	137.370937	13.323040	1
3522	2.00000	1.182898	117.464956	150.116814	26.663496	1
3523	2.00000	0.130003	88.165628	107.000000	3.637186	1

3524 rows × 6 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
from dataPrep import dataPreperation
df2 = dataPreperation(df)
df2
```

	PAQ605	BMXBMI	LBXGLU	LBXGLT	LBXIN	age_group
0	2.0	1.393867	110.000000	150.000000	14.910000	0
1	2.0	1.099498	89.000000	80.000000	3.850000	0
2	2.0	0.629968	89.000000	68.000000	6.140000	0
3	2.0	0.292901	104.000000	84.000000	16.150000	0
5	2.0	0.565206	110.000000	100.000000	6.080000	0
...	...	...	...	...	...	...
3517	2.0	0.225154	109.963071	105.950761	5.224519	1
3518	2.0	0.557655	95.370052	80.629948	4.039809	1
3520	2.0	0.233323	95.785948	80.607026	4.404918	1
3521	2.0	0.855922	94.000000	137.370937	13.323040	1
3523	2.0	0.130003	88.165628	107.000000	3.637186	1

2580 rows × 6 columns

Next steps: Generate code with df2 | View recommended plots | New interactive sheet

```
df3 = dataPreperation(df2)
df3
```

	PAQ605	BMXBMI	LBXGLU	LBXGLT	LBXIN	age_group
0	2.0	1.393867	110.000000	150.000000	14.910000	0
1	2.0	1.099498	89.000000	80.000000	3.850000	0
2	2.0	0.629968	89.000000	68.000000	6.140000	0
3	2.0	0.292901	104.000000	84.000000	16.150000	0
5	2.0	0.565206	110.000000	100.000000	6.080000	0
...	...	...	...	...	...	...
3517	2.0	0.225154	109.963071	105.950761	5.224519	1
3518	2.0	0.557655	95.370052	80.629948	4.039809	1
3520	2.0	0.233323	95.785948	80.607026	4.404918	1
3521	2.0	0.855922	94.000000	137.370937	13.323040	1
3523	2.0	0.130003	88.165628	107.000000	3.637186	1

2517 rows × 6 columns

Next steps: Generate code with df3 View recommended plots New interactive sheet

```
df4 = dataPreperation(df3)
df4
```

	PAQ605	BMXBMI	LBXGLU	LBXGLT	LBXIN	age_group
0	2.0	1.393867	110.000000	150.000000	14.910000	0
1	2.0	1.099498	89.000000	80.000000	3.850000	0
2	2.0	0.629968	89.000000	68.000000	6.140000	0
3	2.0	0.292901	104.000000	84.000000	16.150000	0
5	2.0	0.565206	110.000000	100.000000	6.080000	0
...	...	...	...	...	...	...
3517	2.0	0.225154	109.963071	105.950761	5.224519	1
3518	2.0	0.557655	95.370052	80.629948	4.039809	1
3520	2.0	0.233323	95.785948	80.607026	4.404918	1
3521	2.0	0.855922	94.000000	137.370937	13.323040	1
3523	2.0	0.130003	88.165628	107.000000	3.637186	1

2501 rows × 6 columns

Next steps: Generate code with df4 View recommended plots New interactive sheet

```
df5 = dataPreperation(df4)
df5
```

	PAQ605	BMXBMI	LBXGLU	LBXGLT	LBXIN	age_group
0	2.0	1.393867	110.000000	150.000000	14.910000	0
1	2.0	1.099498	89.000000	80.000000	3.850000	0
2	2.0	0.629968	89.000000	68.000000	6.140000	0
3	2.0	0.292901	104.000000	84.000000	16.150000	0
5	2.0	0.565206	110.000000	100.000000	6.080000	0
...	...	...	...	...	...	...
3517	2.0	0.225154	109.963071	105.950761	5.224519	1
3518	2.0	0.557655	95.370052	80.629948	4.039809	1
3520	2.0	0.233323	95.785948	80.607026	4.404918	1
3521	2.0	0.855922	94.000000	137.370937	13.323040	1
3523	2.0	0.130003	88.165628	107.000000	3.637186	1

2493 rows × 6 columns

Next steps: Generate code with df5 View recommended plots New interactive sheet

```
df6 = dataPreperation(df5)
df6
```

	PAQ605	BMXBMI	LBXGLU	LBXGLT	LBXIN	age_group
0	2.0	1.393867	110.000000	150.000000	14.910000	0
1	2.0	1.099498	89.000000	80.000000	3.850000	0
2	2.0	0.629968	89.000000	68.000000	6.140000	0
3	2.0	0.292901	104.000000	84.000000	16.150000	0
5	2.0	0.565206	110.000000	100.000000	6.080000	0
...	...	...	...	...	...	...
3517	2.0	0.225154	109.963071	105.950761	5.224519	1
3518	2.0	0.557655	95.370052	80.629948	4.039809	1
3520	2.0	0.233323	95.785948	80.607026	4.404918	1
3521	2.0	0.855922	94.000000	137.370937	13.323040	1
3523	2.0	0.130003	88.165628	107.000000	3.637186	1

2491 rows × 6 columns

Next steps: Generate code with df6 | View recommended plots | New interactive sheet

```
df7 = dataPreperation(df6)
df7
```

	PAQ605	BMXBMI	LBXGLU	LBXGLT	LBXIN	age_group
0	2.0	1.393867	110.000000	150.000000	14.910000	0
1	2.0	1.099498	89.000000	80.000000	3.850000	0
2	2.0	0.629968	89.000000	68.000000	6.140000	0
3	2.0	0.292901	104.000000	84.000000	16.150000	0
5	2.0	0.565206	110.000000	100.000000	6.080000	0
...	...	...	...	...	...	...
3517	2.0	0.225154	109.963071	105.950761	5.224519	1
3518	2.0	0.557655	95.370052	80.629948	4.039809	1
3520	2.0	0.233323	95.785948	80.607026	4.404918	1
3521	2.0	0.855922	94.000000	137.370937	13.323040	1
3523	2.0	0.130003	88.165628	107.000000	3.637186	1

2491 rows × 6 columns

Next steps: Generate code with df7 | View recommended plots | New interactive sheet

```
X = df7.drop('age_group', axis=1)
y = df7['age_group']
```

```
y.value_counts()
```

age_group	count
1	1292
0	1199

dtype: int64

```
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, random_state=30)
```

```
def runModel(model, X_train, X_test, y_train, y_test):
  model.fit(X_train, y_train)

  y_pred = model.predict(X_test)
```

```
  y_train_pred = model.predict(X_train)

  acc_test = accuracy_score(y_test, y_pred)
  acc_train = accuracy_score(y_train, y_train_pred)

  y_pred_proba = model.predict_proba(X_test)

  cm = confusion_matrix(y_test, y_pred)
  # cr = classification_report(y_test, y_pred)
  print(f'Train Accuracy: {acc_train}')
  print(f'Test Accuracy: {acc_test}')
  print(cm)
  return acc_test, acc_train, cm, y_pred_proba, y_pred


from sklearn.linear_model import LogisticRegression as LR
from sklearn.tree import DecisionTreeClassifier as DT
from xgboost import XGBClassifier as XGB
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.ensemble import RandomForestClassifier as RF
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


# model = LogisticRegression(max_iter= 10000)

models_arr = [LR(max_iter=10000), DT(), XGB(), KNN(), RF()]
for model in models_arr:
  print(model.__class__.__name__)
  runModel(model, X_train, X_test, y_train, y_test)
  acc_test, acc_train, cm, y_pred_proba, y_pred = runModel(model, X_train, X_test, y_train, y_test)
```

```
LogisticRegression
Train Accuracy: 0.6666666666666666
Test Accuracy: 0.6724598930481284
[[228 113]
 [132 275]]
Train Accuracy: 0.6666666666666666
Test Accuracy: 0.6724598930481284
[[228 113]
 [132 275]]
DecisionTreeClassifier
Train Accuracy: 1.0
Test Accuracy: 0.7633689839572193
[[243  98]
 [ 79 328]]
Train Accuracy: 1.0
Test Accuracy: 0.7780748663101604
[[254  87]
 [ 79 328]]
XGBClassifier
Train Accuracy: 0.9994262765347103
Test Accuracy: 0.8475935828877005
[[272  69]
 [ 45 362]]
Train Accuracy: 0.9994262765347103
Test Accuracy: 0.8475935828877005
[[272  69]
 [ 45 362]]
KNeighborsClassifier
Train Accuracy: 0.8559954102122777
Test Accuracy: 0.7874331550802139
[[222 119]
 [ 40 367]]
Train Accuracy: 0.8559954102122777
Test Accuracy: 0.7874331550802139
[[222 119]
 [ 40 367]]
RandomForestClassifier
Train Accuracy: 1.0
Test Accuracy: 0.8422459893048129
[[271  70]
 [ 48 359]]
Train Accuracy: 1.0
Test Accuracy: 0.8368983957219251
[[269  72]
 [ 50 357]]
```

```
df7
```

	PAQ605	BMXBMI	LBXGLU	LBXGLT	LBXIN	age_group
0	2.0	1.393867	110.000000	150.000000	14.910000	0
1	2.0	1.099498	89.000000	80.000000	3.850000	0
2	2.0	0.629968	89.000000	68.000000	6.140000	0
3	2.0	0.292901	104.000000	84.000000	16.150000	0
5	2.0	0.565206	110.000000	100.000000	6.080000	0
...	...	...	...	...	...	...
3517	2.0	0.225154	109.963071	105.950761	5.224519	1
3518	2.0	0.557655	95.370052	80.629948	4.039809	1
3520	2.0	0.233323	95.785948	80.607026	4.404918	1
3521	2.0	0.855922	94.000000	137.370937	13.323040	1
3523	2.0	0.130003	88.165628	107.000000	3.637186	1

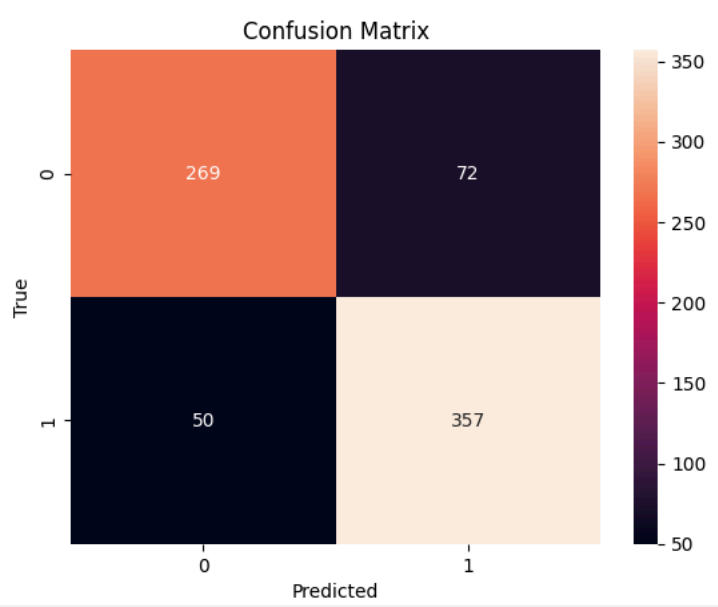
2491 rows × 6 columns

Next steps: Generate code with df7 View recommended plots New interactive sheet

```
# prompt: visualize cm

import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'cm' is your confusion matrix from the previous code
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```




```
from sklearn.preprocessing import MinMaxScaler
minmaxscaller = MinMaxScaler()
df7_scaled = minmaxscaller.fit_transform(df7)
df7_scaled = pd.DataFrame(df7_scaled, columns=df7.columns)
df7_scaled
```

	PAQ605	BMXBMI	LBXGLU	LBXGLT	LBXIN	age_group
0	0.0	0.741379	0.808654	0.731544	0.725086	0.0
1	0.0	0.584643	0.323462	0.261745	0.182131	0.0
2	0.0	0.334643	0.323462	0.181208	0.294551	0.0
3	0.0	0.155172	0.670028	0.288591	0.785960	0.0
4	0.0	0.300161	0.808654	0.395973	0.291605	0.0
...	...	...	...	...	...	...
2486	0.0	0.119101	0.807801	0.435911	0.249608	1.0
2487	0.0	0.296140	0.470638	0.265973	0.191449	1.0
2488	0.0	0.123451	0.480247	0.265819	0.209373	1.0
2489	0.0	0.454952	0.438984	0.646785	0.647179	1.0
2490	0.0	0.068438	0.304184	0.442953	0.171683	1.0

2491 rows × 6 columns

Next steps: Generate code with df7_scaled | View recommended plots | New interactive sheet

```
X_scaled = df7_scaled.drop('age_group', axis=1)
y_scaled = df7_scaled['age_group']


X_train_scaled, X_test_scaled, y_train_scaled, y_test_scaled = train_test_split(X_scaled, y_scaled, train_size=0.7, random_state=30)


X_train_scaled.shape, X_test_scaled.shape, y_train_scaled.shape, y_test_scaled.shape
```

```
((1743, 5), (748, 5), (1743,), (748,))
```

```
for model in models_arr:
  print(model.__class__.__name__)
  runModel(model, X_train_scaled, X_test_scaled, y_train_scaled, y_test_scaled)
```

```
LogisticRegression
Train Accuracy: 0.663798049340218
Test Accuracy: 0.6697860962566845
[[227 114]
 [133 274]]
DecisionTreeClassifier
Train Accuracy: 1.0
Test Accuracy: 0.7740641711229946
[[251  90]
 [ 79 328]]
XGBClassifier
Train Accuracy: 0.9994262765347103
Test Accuracy: 0.8475935828877005
[[272  69]
 [ 45 362]]
KNeighborsClassifier
Train Accuracy: 0.8387837062535858
Test Accuracy: 0.7606951871657754
[[219 122]
 [ 57 350]]
RandomForestClassifier
Train Accuracy: 1.0
Test Accuracy: 0.8422459893048129
[[274  67]
 [ 51 356]]
```

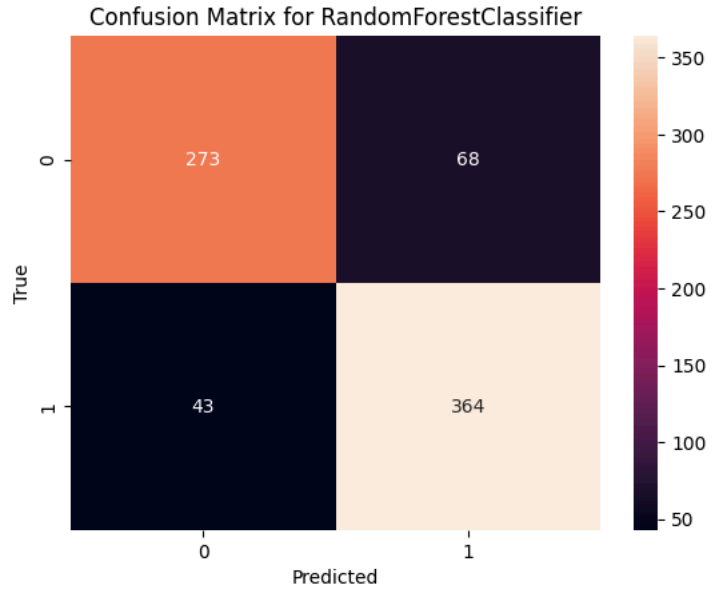
```
# prompt: visualise the randomforestclassifier cm and write the train and test accuracy

# Assuming 'cm' is your confusion matrix from the previous code
# You need to call runModel for RandomForestClassifier and capture its output
# Example:
model = RF()
acc_test, acc_train, cm, y_pred_proba, y_pred = runModel(model, X_train, X_test, y_train, y_test)

sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for RandomForestClassifier')
plt.show()

print(f'Train Accuracy for RandomForestClassifier: {acc_train}')
print(f'Test Accuracy for RandomForestClassifier: {acc_test}')
```

```
Train Accuracy: 1.0
Test Accuracy: 0.8516042780748663
[[273  68]
 [ 43 364]]
```




```
Train Accuracy for RandomForestClassifier: 1.0
Test Accuracy for RandomForestClassifier: 0.8516042780748663
```

```
y_train_scaled.value_counts()
```

age_group	count
1.0	885
0.0	858

dtype: int64

```
from imblearn.over_sampling import SMOTE
smote = SMOTE()
X_train_smote, y_train_smote = smote.fit_resample(X_train_scaled, y_train_scaled)
```

```
y_train_smote.value_counts()
```

age_group	count
1.0	885
0.0	885

dtype: int64

```
for model in models_arr:
  print(model.__class__.__name__)
  runModel(model, X_train_smote, X_test_scaled, y_train_smote, y_test_scaled)
```

```
LogisticRegression
Train Accuracy: 0.6717514124293785
Test Accuracy: 0.6778074866310161
[[234 107]
 [134 273]]
DecisionTreeClassifier
Train Accuracy: 1.0
Test Accuracy: 0.7754010695187166
[[261  80]
 [ 88 319]]
XGBClassifier
Train Accuracy: 0.9994350282485875
Test Accuracy: 0.8703208556149733
[[291  50]
 [ 47 360]]
KNeighborsClassifier
Train Accuracy: 0.8429378531073446
Test Accuracy: 0.7633689839572193
[[221 120]
 [ 57 350]]
```

```
RandomForestClassifier
Train Accuracy: 1.0
Test Accuracy: 0.8475935828877005
[[274  67]
 [ 47 360]]
```

```
df_smote = pd.concat([X_train_smote, y_train_smote], axis=1)
df_smote
```

	PAQ605	BMXBMI	LBXGLU	LBXGLT	LBXIN	age_group
0	0.0	0.178105	0.381524	0.379107	0.352052	1.0
1	0.0	0.017241	0.346566	0.966443	0.704958	0.0
2	0.0	0.233961	0.882803	0.769783	0.679980	1.0
3	0.0	0.550161	0.369670	0.563758	0.165439	0.0
4	0.0	0.326023	0.415879	0.228188	0.358861	0.0
...	...	...	...	...	...	...
1765	0.0	0.189968	0.518592	0.620371	0.286924	0.0
1766	0.0	0.080197	0.512144	0.219235	0.684890	0.0
1767	0.0	0.528054	0.481933	0.282868	0.152098	0.0
1768	0.0	0.320582	0.317136	0.286192	0.334978	0.0
1769	0.0	0.675744	0.260740	0.232985	0.226349	0.0

1770 rows × 6 columns

Next steps: Generate code with df_smote | View recommended plots | New interactive sheet

```
df8 = dataPreperation(df_smote)
df8
```

	PAQ605	BMXBMI	LBXGLU	LBXGLT	LBXIN	age_group
0	0.0	0.178105	0.381524	0.379107	0.352052	1.0
1	0.0	0.017241	0.346566	0.966443	0.704958	0.0
2	0.0	0.233961	0.882803	0.769783	0.679980	1.0
3	0.0	0.550161	0.369670	0.563758	0.165439	0.0
4	0.0	0.326023	0.415879	0.228188	0.358861	0.0
...	...	...	...	...	...	...
1765	0.0	0.189968	0.518592	0.620371	0.286924	0.0
1766	0.0	0.080197	0.512144	0.219235	0.684890	0.0
1767	0.0	0.528054	0.481933	0.282868	0.152098	0.0
1768	0.0	0.320582	0.317136	0.286192	0.334978	0.0
1769	0.0	0.675744	0.260740	0.232985	0.226349	0.0

1770 rows × 6 columns

Next steps: Generate code with df_smote | View recommended plots | New interactive sheet

```
df9 = dataPreperation(df_smote)
df9
```

	PAQ605	BMXBMI	LBXGLU	LBXGLT	LBXIN	age_group
0	0.0	0.178105	0.381524	0.379107	0.352052	1.0
1	0.0	0.017241	0.346566	0.966443	0.704958	0.0
2	0.0	0.233961	0.882803	0.769783	0.679980	1.0
3	0.0	0.550161	0.369670	0.563758	0.165439	0.0
4	0.0	0.326023	0.415879	0.228188	0.358861	0.0
...	...	...	...	...	...	...
1765	0.0	0.189968	0.518592	0.620371	0.286924	0.0
1766	0.0	0.080197	0.512144	0.219235	0.684890	0.0
1767	0.0	0.528054	0.481933	0.282868	0.152098	0.0
1768	0.0	0.320582	0.317136	0.286192	0.334978	0.0
1769	0.0	0.675744	0.260740	0.232985	0.226349	0.0

1770 rows × 6 columns

Next steps: Generate code with df_smote | View recommended plots | New interactive sheet

```
X = df9.drop('age_group', axis=1)
y = df9['age_group']


X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=30)


models_output = []
for model in models_arr:
  print(model.__class__.__name__)
  acc_test, acc_train, cm, y_pred_proba, y_pred=  runModel(model, X_train, X_test, y_train, y_test)
  models_output.append([model.__class__.__name__, acc_test, acc_train, cm, y_pred_proba,y_pred])
```

```
LogisticRegression
Train Accuracy: 0.6771589991928975
Test Accuracy: 0.6534839924670434
[[173  93]
 [ 91 174]]
DecisionTreeClassifier
Train Accuracy: 1.0
Test Accuracy: 0.743879472693032
[[188  78]
 [ 58 207]]
XGBClassifier
Train Accuracy: 1.0
Test Accuracy: 0.7928436911487758
[[196  70]
 [ 40 225]]
KNeighborsClassifier
Train Accuracy: 0.8297013720742534
Test Accuracy: 0.7137476459510358
[[160 106]
 [ 46 219]]
RandomForestClassifier
Train Accuracy: 1.0
Test Accuracy: 0.815442561205273
[[206  60]
 [ 38 227]]
```

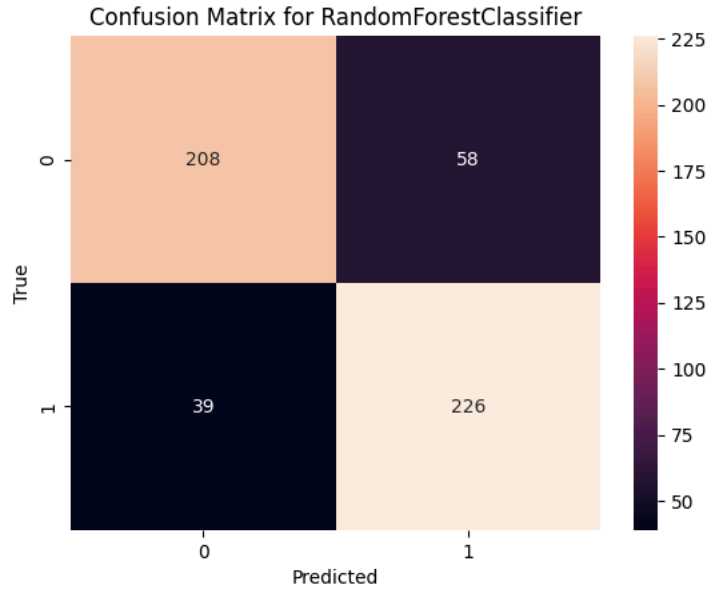
```
# prompt: visualise the randomforestclassifier cm and write the train and test accuracy

# Assuming 'cm' is your confusion matrix from the previous code
# You need to call runModel for RandomForestClassifier and capture its output
# Example:
model = RF()
acc_test, acc_train, cm, y_pred_proba, y_pred = runModel(model, X_train, X_test, y_train, y_test)

sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for RandomForestClassifier')
plt.show()

print(f'Train Accuracy for RandomForestClassifier: {acc_train}')
print(f'Test Accuracy for RandomForestClassifier: {acc_test}')
```

```
Train Accuracy: 1.0
Test Accuracy: 0.8173258003766478
[[208  58]
 [ 39 226]]
```




```
Train Accuracy for RandomForestClassifier: 1.0
Test Accuracy for RandomForestClassifier: 0.8173258003766478
```

```
models_output[0][4]
```

```
array([[0.32894274, 0.67105726],
       [0.68715284, 0.31284716],
       [0.79072109, 0.20927891],
       ...,
       [0.6118815 , 0.3881185 ],
       [0.69828459, 0.30171541],
       [0.66271712, 0.33728288]])
```

```
models_output[0][5]
```

```
array([1., 0., 0., 0., 1., 1., 1., 0., 0., 1., 0., 1., 1., 0., 1., 0., 0.,
       0., 0., 0., 0., 1., 1., 1., 0., 1., 1., 1., 1., 1., 0., 0., 1.,
       0., 0., 1., 1., 0., 1., 1., 0., 1., 1., 0., 0., 0., 1., 0., 1., 0.,
       0., 1., 0., 1., 0., 1., 0., 0., 1., 0., 0., 0., 1., 0., 1., 1., 0.,
       0., 1., 0., 0., 1., 1., 1., 0., 0., 0., 1., 0., 1., 0., 1., 1., 0.,
       0., 1., 1., 1., 0., 0., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 1., 0., 1.,
       0., 0., 1., 0., 0., 0., 1., 1., 0., 0., 1., 1., 1., 1., 0., 1., 0.,
       0., 0., 0., 1., 1., 1., 0., 0., 1., 0., 0., 1., 1., 0., 0., 1., 0.,
       0., 1., 1., 1., 1., 1., 0., 0., 1., 0., 0., 0., 1., 0., 0., 1., 0.,
       0., 1., 0., 0., 1., 1., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 1.,
       1., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1.,
       1., 1., 1., 0., 1., 1., 1., 0., 0., 0., 0., 1., 0., 1., 1., 0., 1.,
       1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 1., 0., 1., 1., 0., 1.,
       1., 1., 1., 1., 1., 1., 0., 1., 1., 0., 0., 1., 1., 1., 1., 1., 0.,
       1., 0., 1., 0., 0., 0., 0., 1., 1., 1., 1., 0., 0., 0., 1., 0., 1.,
       1., 1., 0., 1., 0., 0., 0., 1., 0., 0., 0., 1., 0., 1., 1., 1., 1.,
       1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 0., 1., 1., 1., 0.,
       1., 1., 0., 0., 1., 1., 1., 1., 1., 0., 0., 0., 1., 1., 1., 0., 0.,
       1., 0., 1., 0., 1., 0., 1., 1., 0., 0., 1., 0., 1., 1., 1., 0., 0.,
       1., 0., 1., 1., 1., 0., 0., 1., 0., 1., 1., 0., 0., 1., 0., 0., 0.,
       1., 1., 1., 1., 0., 0., 1., 0., 0., 1., 0., 0., 1., 1., 1., 1., 1.,
       1., 1., 0., 1., 0., 1., 0., 1., 1., 1., 0., 0., 0., 0., 1., 1., 1.,
       1., 0., 0., 1., 1., 1., 1., 0., 0., 1., 0., 1., 1., 0., 1., 0., 1.,
       1., 1., 0., 0., 1., 1., 0., 0., 0., 0., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 1., 1., 0.,
       0., 0., 1., 0., 1., 0., 1., 1., 1., 1., 1., 0., 0., 0., 1., 0., 0.,
       0., 1., 1., 0., 0., 1., 1., 0., 1., 0., 1., 0., 1., 1., 0., 0., 1.,
       0., 1., 1., 1., 1., 1., 0., 1., 0., 1., 1., 1., 1., 0., 0., 0., 1.,
       1., 1., 0., 0., 0., 1., 1., 0., 0., 0., 1., 0., 0., 1., 0., 1., 0.,
       1., 0., 0., 0.])
```

```
df_y_pred = pd.DataFrame(models_output[2][5], columns=['Prediction'])
df_y_pred
```

	Prediction
0	1
1	0
2	0
3	0
4	1
...	...
526	0
527	1
528	0
529	0
530	0

Next steps: Generate code with df_y_pred | View recommended plots | New interactive sheet

```
df_pred_proba = pd.DataFrame(models_output[2][4], columns=['0', '1'])
df_pred_proba
```

	0	1
0	0.188831	0.811169
1	0.891312	0.108688
2	0.999858	0.000142
3	0.912850	0.087150
4	0.365254	0.634746
...	...	...
526	0.987729	0.012271
527	0.171840	0.828160
528	0.960344	0.039656
529	0.997162	0.002838
530	0.992900	0.007100

531 rows × 2 columns

Next steps: Generate code with df_pred_proba | View recommended plots | New interactive sheet

```
df_both = pd.concat([df_y_pred, df_pred_proba], axis=1)
df_both
```

	Prediction	0	1
0	1	0.188831	0.811169
1	0	0.891312	0.108688
2	0	0.999858	0.000142
3	0	0.912850	0.087150
4	1	0.365254	0.634746
...	...	...	...
526	0	0.987729	0.012271
527	1	0.171840	0.828160
528	0	0.960344	0.039656
529	0	0.997162	0.002838
530	0	0.992900	0.007100

531 rows × 3 columns