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

```
df=pd.read_csv('Credit_Card_prediction_dataset.csv')
```

```
display(df)
```

	age	income	credit_score	dependents	home_owner	Credit_card_approved
0	54	100000.00	334	0	1	1
1	67	85233.42	593	2	1	1
2	29	16737.15	502	0	0	1
3	42	69332.50	367	3	0	0
4	58	28211.14	430	0	1	1
...	...	...	...	...	...	...
995	68	18279.98	379	3	0	0
996	41	8244.06	653	4	1	1
997	39	16194.69	460	2	1	1
998	52	38739.91	726	2	1	1
999	24	11278.55	702	0	0	1

1000 rows × 6 columns

```
df.head()
```

	age	income	credit_score	dependents	home_owner	Credit_card_approved
0	54	100000.00	334	0	1	1
1	67	85233.42	593	2	1	1
2	29	16737.15	502	0	0	1
3	42	69332.50	367	3	0	0
4	58	28211.14	430	0	1	1

Data Cleaning

```
df.isnull()
```

	age	income	credit_score	dependents	home_owner	Credit_card_approved
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...
995	False	False	False	False	False	False
996	False	False	False	False	False	False
997	False	False	False	False	False	False
998	False	False	False	False	False	False
999	False	False	False	False	False	False

1000 rows × 6 columns

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

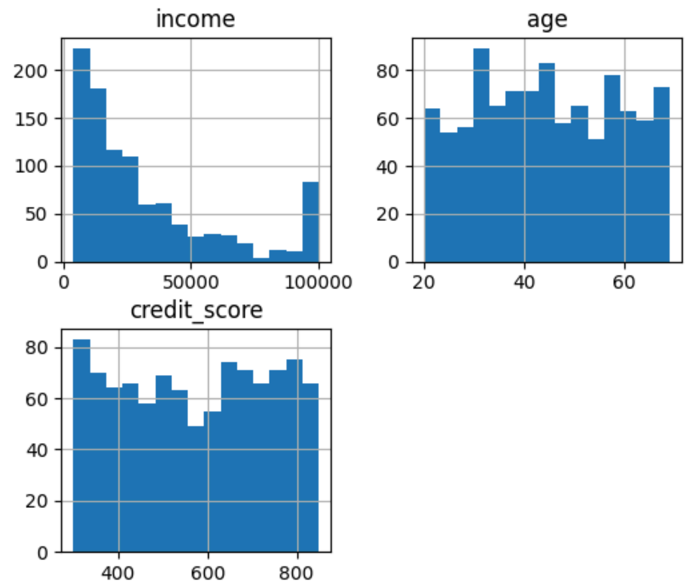
	0
age	0
income	0
credit_score	0
dependents	0
home_owner	0
Credit_card_approved	0

dtype: int64

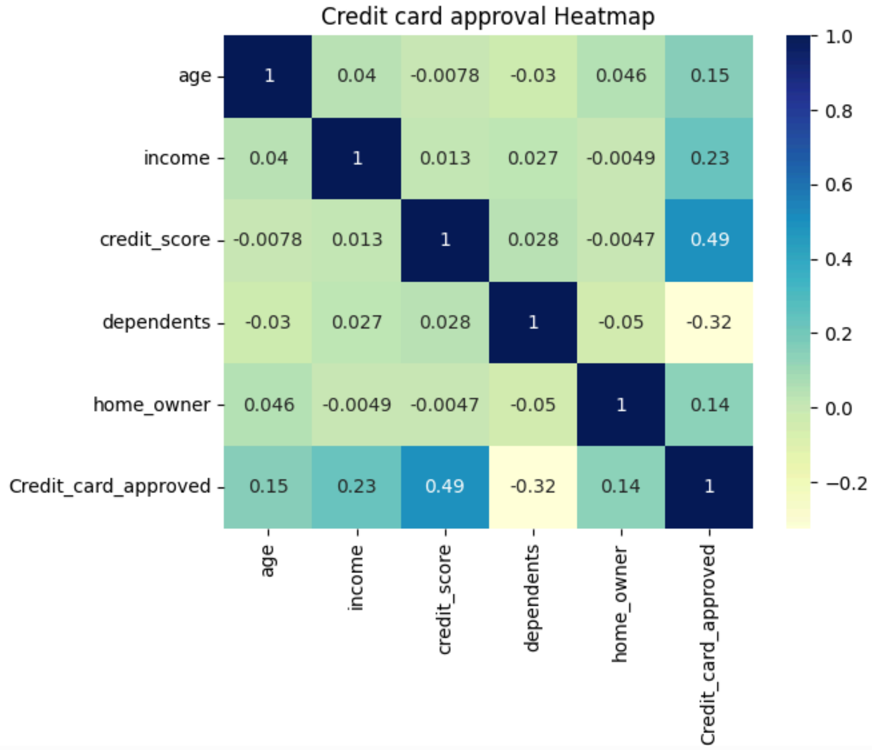
Data Visualization

Using Histogram to understand feature distribution

```
selected_features = ['income', 'age','credit_score' ] # Replacing 'Type_income' with an existing column named 'ID' or any other existing
myfile_selected = myfile[selected_features]
myfile_selected.hist(bins=15,figsize=(6,5))
plt.show()
```

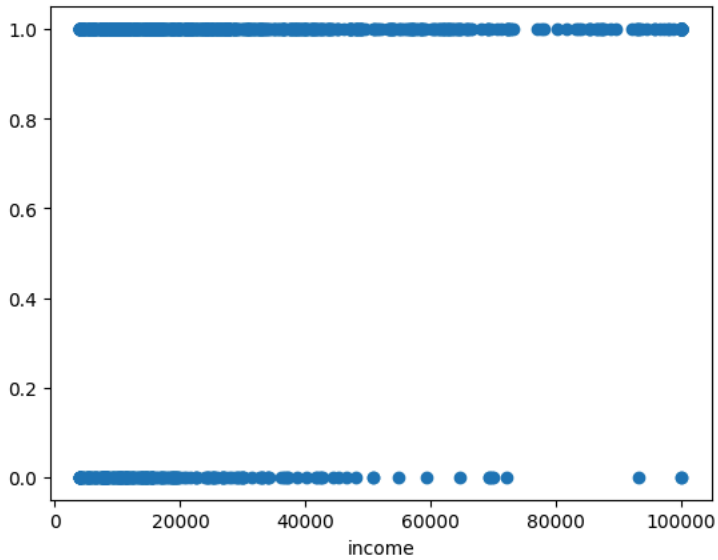



```
import seaborn as sns

# Select only numerical features for correlation analysis
numerical_features = df.select_dtypes(include=['number'])

sns.heatmap(numerical_features.corr(), annot=True, cmap='YlGnBu')
plt.title("Credit card approval Heatmap")
plt.show()
```

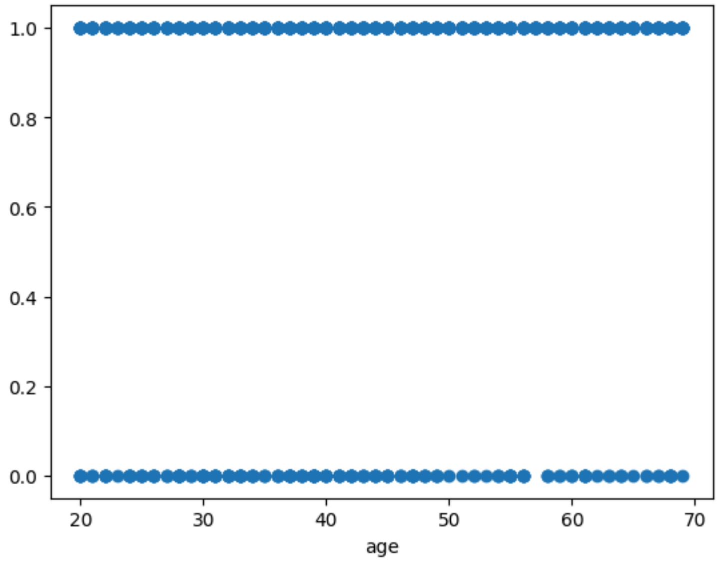


Using scatter plot

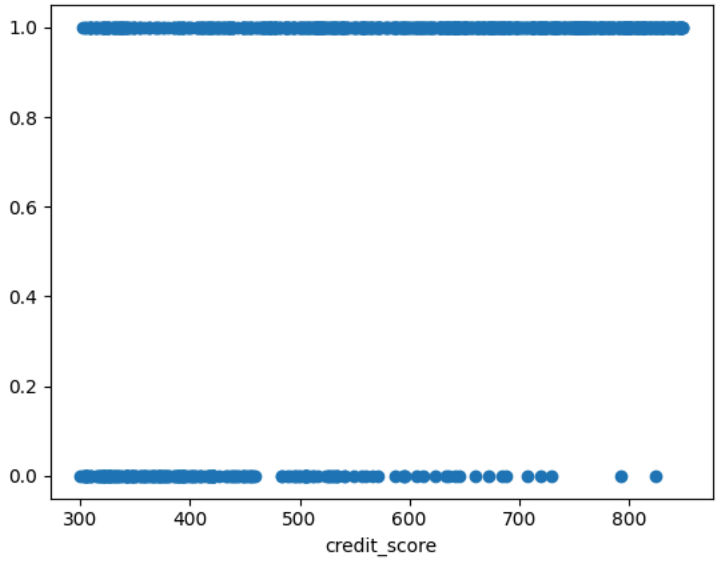
```
plt.scatter(df.income, df.Credit_card_approved)
plt.xlabel('income')
plt.show()
```




```
plt.scatter(df.age,df.Credit_card_approved)
plt.xlabel('age')
plt.show()
```




```
plt.scatter(df.credit_score, df.Credit_card_approved)
plt.xlabel('credit_score')
plt.show()
```




```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 6 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   age                   1000 non-null   int64  
 1   income                1000 non-null   float64
 2   credit_score          1000 non-null   int64  
 3   dependents            1000 non-null   int64  
 4   home_owner            1000 non-null   int64  
 5   Credit_card_approved  1000 non-null   int64  
dtypes: float64(1), int64(5)
memory usage: 47.0 KB
```

Using logistic regression

```
x=df.iloc[:,[0,1,2]]
y=df.iloc[:,5]
```

```
from sklearn.model_selection import train_test_split
```

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3)
```

```
from sklearn.linear_model import LogisticRegression
```

```
reg_model=LogisticRegression()
```

```
reg_model.fit(xtrain,ytrain)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
reg_model.predict(xtest)
```

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

```
y_pred=reg_model.predict(xtest)
```

```
reg_model.predict_proba(xtest)
```

```
array([[2.06834187e-03, 9.97931658e-01],
       [2.45938114e-01, 7.54061886e-01],
       [6.39051341e-01, 3.60948659e-01],
       [2.33428597e-02, 9.76657140e-01],
       [7.83565942e-02, 9.21643406e-01],
       [1.27592266e-02, 9.87240773e-01],
       [1.61954940e-01, 8.38045060e-01],
       [7.37850760e-02, 9.26214924e-01],
       [1.60193021e-02, 9.83980698e-01],
       [4.22878299e-03, 9.95771217e-01],
       [5.03983503e-01, 4.96016497e-01],
       [2.93404232e-01, 7.06595768e-01],
       [2.34393176e-02, 9.76560682e-01],
       [2.93326813e-01, 7.06673187e-01],
       [4.58174776e-01, 5.41825224e-01],
       [1.09781361e-01, 8.90218639e-01],
       [9.22237027e-02, 9.07776297e-01],
       [7.94803745e-01, 2.05196255e-01],
       [9.72180515e-02, 9.02781949e-01],
       [8.18517973e-02, 9.18148203e-01],
       [6.72154159e-03, 9.93278458e-01],
       [6.23303015e-02, 9.37669698e-01],
       [9.41197735e-01, 5.88022649e-02],
       [2.63162945e-01, 7.36837055e-01],
       [4.72766916e-02, 9.52723308e-01],
       [7.16028105e-01, 2.83971895e-01],
       [4.06378054e-01, 5.93621946e-01],
       [7.28562932e-01, 2.71437068e-01],
       [3.57202705e-01, 6.42797295e-01],
       [6.13536769e-03, 9.93864632e-01],
       [6.91509382e-01, 3.08490618e-01],
       [2.51251095e-01, 7.48748905e-01],
       [3.97309926e-01, 6.02690074e-01],
       [2.97745583e-01, 7.02254417e-01],
       [2.22080508e-02, 9.77791949e-01],
       [1.94772886e-01, 8.05227114e-01],
       [4.12903304e-01, 5.87096696e-01],
       [2.06643295e-02, 9.79335671e-01],
       [3.66542397e-01, 6.33457603e-01],
       [5.94727059e-02, 9.40527294e-01],
       [6.18788119e-03, 9.93812119e-01],
       [1.24251222e-03, 9.98757488e-01],
       [4.65794074e-01, 5.34205926e-01],
       [3.25262170e-02, 9.67473783e-01],
       [6.94448783e-02, 9.30555122e-01],
       [5.78966935e-01, 4.21033065e-01],
       [3.91404676e-03, 9.96085953e-01],
       [3.13908221e-03, 9.96860918e-01],
       [4.90845229e-01, 5.09154771e-01],
       [5.96284086e-01, 4.03715914e-01],
       [4.74608811e-01, 5.25391189e-01],
       [7.75108753e-01, 2.24891247e-01],
       [7.67247876e-01, 2.32752124e-01],
       [6.46886917e-02, 9.35311308e-01],
       [6.60118851e-03, 9.93398811e-01],
       [2.06784415e-01, 7.93215585e-01],
```

```
       [2.21467471e-03, 9.97785325e-01],
       [6.91510558e-01, 3.08489442e-01],
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(ytest,y_pred)
print(confusion_matrix)
```

```
[[ 32  21]
 [ 21 226]]
```

```
from sklearn.metrics import classification_report
print(classification_report(ytest,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.60      0.60      0.60        53
           1       0.91      0.91      0.91       247

    accuracy                           0.86       300
   macro avg       0.76      0.76      0.76       300
weighted avg       0.86      0.86      0.86       300
```

```
from sklearn.linear_model import LogisticRegression

# Create and train the model
model = LogisticRegression()
model.fit(xtrain, ytrain)

# Print accuracy
print("Logistic Regression Accuracy:", model.score(xtest, ytest))
```

```
Logistic Regression Accuracy: 0.86
```

Using Navie Bayes

```
from sklearn.naive_bayes import GaussianNB
classifier=GaussianNB()
```

```
classifier.fit(xtrain,ytrain)
```

```
▾ GaussianNB
GaussianNB()
```

```
y_predict=classifier.predict(xtest)
```

```
from sklearn.metrics import confusion_matrix
conf_mat=confusion_matrix(ytest,y_predict)
print(conf_mat)
```

```
[[ 32  21]
 [ 35 212]]
```

```
from sklearn.metrics import classification_report
print(classification_report(ytest,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.48      0.60      0.53        53
           1       0.91      0.86      0.88       247

    accuracy                           0.81       300
   macro avg       0.69      0.73      0.71       300
weighted avg       0.83      0.81      0.82       300
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Step 1: Initialize the model
nb_model = GaussianNB()

# Step 2: Train the model
# Replace X_train and y_train with the actual variable names used in previous cells.
nb_model.fit(xtrain, ytrain)
```

```
# Step 3: Predict on test data
# Replace X_test with xtest which is your testing dataset
y_pred = nb_model.predict(xtest)

# Step 4: Calculate and print accuracy
accuracy = accuracy_score(ytest, y_pred)  # Replace y_test with ytest
print("Naive Bayes Accuracy:", accuracy)
```

```
Naive Bayes Accuracy: 0.8333333333333334
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
classifier=DecisionTreeClassifier() #we are using default parameters criterion=gini ,max_depth is not restricted
classifier=classifier.fit(xtrain,ytrain)
```

```
y_pred=classifier.predict(xtest)
print(y_pred)
```

```
[1 1 0 1 1 1 0 1 1 1 0 1 1 0 0 1 1 0 1 1 1 1 0 1 1 0 0 1 1 1 0 1 0 1 1 0 0
 1 1 1 1 1 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 0 1 1 1 0
 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1
 1 1 0 1 1 1 0 1 1 1 1 1 1 0 0 1 0 0 0 1 0 0 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0
 1 0 1 1 1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0
 1 1 0 0 0 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 1 1 0 1 1 0 1 1 1 0 1 1 1
 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 1 1
 1 1 1 1]
```

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(ytest,y_pred)
print(cm)
```

```
[[ 33  20]
 [ 41 206]]
```

```
from sklearn.metrics import classification_report
print(classification_report(ytest,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.45      0.62      0.52        53
           1       0.91      0.83      0.87       247

    accuracy                           0.80       300
   macro avg       0.68      0.73      0.70       300
weighted avg       0.83      0.80      0.81       300
```

```
from sklearn.tree import export_graphviz
```

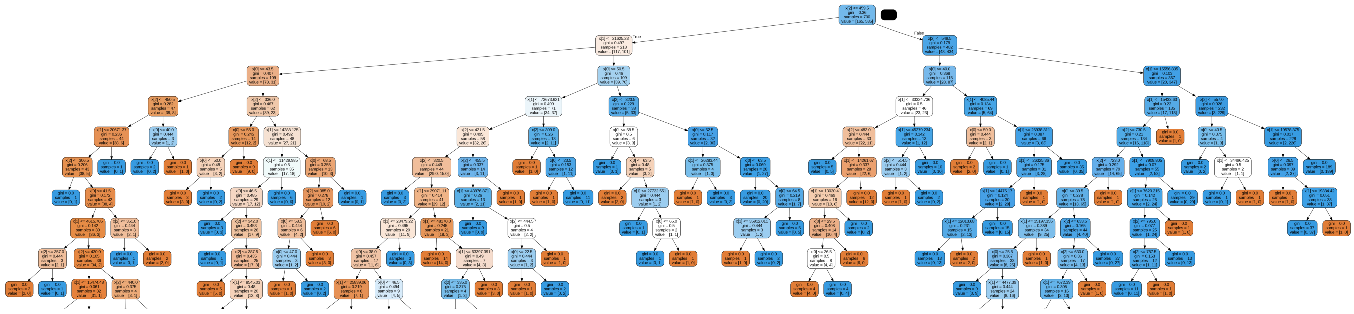
```
!pip install six
```

```
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (1.17.0)
```

```
from six import StringIO
from IPython.display import Image
import pydotplus
```

```
data = StringIO()
```

```
export_graphviz(classifier,out_file=data,filled=True,rounded=True)
class_names=['o','1']
graph=pydotplus.graph_from_dot_data(data.getvalue())
Image(graph.create_png())
```

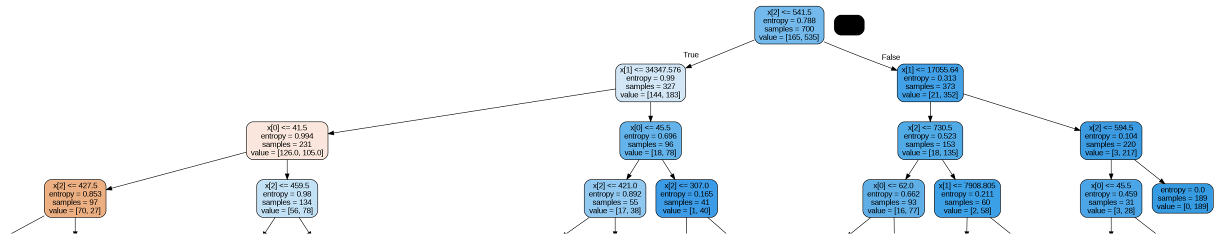


```
from sklearn import metrics
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics

# Assuming xtrain and ytrain from previous cells are the intended training data
x_train = xtrain
y_train = ytrain
x_test = xtest
y_test = ytest

classifier1=DecisionTreeClassifier(criterion='entropy',max_depth=5)
classifier1.fit(x_train,y_train)
y_pred1 =classifier1.predict(x_test)
print(' Accuracy: ',metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy:  0.7966666666666666
```

```
dot_data=StringIO()
export_graphviz(classifier1,out_file=dot_data,filled=True,rounded=True)
graph=pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```




```
import numpy as np

# Take input from the user
print("Enter the following details:")

income= float(input("income (RS): "))
age = float(input("age(yrs): "))
credit_score= float(input("credit_score(cr): "))



# Combine inputs into a 2D array (as expected by the model)
user_input = np.array([[age,income,credit_score]])

# If you used scaling during training, apply the same scaler here
# For example: user_input = scaler.transform(user_input)

# Predict whether approved using the trained classifier
predicted_approved = classifier.predict(user_input)

# Show the result
print("\n✅ Recommended approved for Given Conditions:", predicted_approved[0])
```

```
Enter the following details:
```

```
# Updated features list
features = ['income', 'age','credit_score']
# Recalculate means for only those columns
feature_means = myfile[features].mean()

print("\nEnter the following values (or press Enter to skip):")

user_input = []

for feature in features:
```

```
        value = input(f"{feature}: ")
        if value:
            user_input.append(float(value))
```