INTRODUCTION

This dataset comprises 20,000 fully synthetic customer financial profiles from India, integrating demographic attributes, income and credit indicators, and transaction-level records across diverse merchants and geographies. Generated using controlled algorithms, the data mirrors realistic financial and market behavior while maintaining complete anonymity, making it well-suited for analytical exploration, modeling, and research without privacy concerns.

IMPORTING LIBRARIES AND DATA SET LOADING

In [26]:
```
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import mean_squared_error


from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.cluster import KMeans

from sklearn.metrics import (
    r2_score,
    mean_squared_error,
    accuracy_score,
    classification_report,
    confusion_matrix,
    roc_auc_score,
    roc_curve
)

import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv(r"C:\Users\papus\OneDrive\Documents\PORTFOLIO PROJECT 2\Customer_financial_profiles.csv")
df
```

Out[2]:

	id	current_age	birth_year	birth_month	gender	address	per_capita_income	yearly_income	total_debt	credit_score	...	transaction_id	date	client_id	card_id
0	1	64	1961	3	Female	Street 286, Indore, Madhya Pradesh - 452645	86125	1033501	280435	715	...	1251	28-03-2024	CLTIND00104	CRD001042
1	2	39	1986	12	Female	Street 268, Kolkata, West Bengal - 700555	68414	820965	0	834	...	9976	20-09-2025	CLTIND00213	CRD002132
2	3	50	1975	7	Female	Street 262, Lucknow, Uttar Pradesh - 226797	22595	271138	0	609	...	6084	02-10-2024	CLTIND03292	CRD032922
3	4	27	1998	2	Male	Street 265, Nagpur, Maharashtra - 440623	24158	289898	0	782	...	3350	02-09-2023	CLTIND05205	CRD052053
4	5	45	1980	7	Other	Street 157, Delhi, Delhi - 110965	45475	545697	117821	608	...	11725	25-05-2025	CLTIND04910	CRD049101
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
19995	19996	39	1986	10	Other	Street 112, Chennai, Tamil Nadu - 600341	33596	403151	142167	582	...	6718	24-02-2024	CLTIND04778	CRD047781
19996	19997	44	1981	6	Female	Street 197, Ahmedabad, Gujarat - 380460	66484	797812	0	647	...	15264	21-06-2024	CLTIND00150	CRD001502
19997	19998	43	1982	1	Female	Street 7, Indore, Madhya Pradesh - 452341	41053	492634	0	853	...	8536	05-03-2025	CLTIND02792	CRD027923
19998	19999	49	1976	9	Female	Street 212, Nagpur, Maharashtra - 440770	51856	622267	266802	834	...	14325	06-11-2024	CLTIND05714	CRD057142
19999	20000	37	1988	4	Male	Street 223, Indore, Madhya	43109	517305	130597	753	...	9705	27-06-2024	CLTIND00538	CRD005381

	id	current_age	birth_year	birth_month	gender	address	per_capita_income	yearly_income	total_debt	credit_score	...	transaction_id	date	client_id	card_id
						Pradesh - 452307									

20000 rows × 21 columns

In [3]:
```
df.columns
```

Out[3]:
```
Index(['id', 'current_age', 'birth_year', 'birth_month', 'gender', 'address',
       'per_capita_income', 'yearly_income', 'total_debt', 'credit_score',
       'num_credit_cards', 'transaction_id', 'date', 'client_id', 'card_id',
       'amount', 'use_chip', 'merchant_id', 'merchant_city', 'merchant_state',
       'zip'],
      dtype='object')
```

DATA CLEANING

In [5]:
```
df.dtypes
```

Out[5]:
```
id                    int64
current_age           int64
birth_year            int64
birth_month           int64
gender               object
address              object
per_capita_income     int64
yearly_income         int64
total_debt            int64
credit_score          int64
num_credit_cards      int64
transaction_id        int64
date                 object
client_id            object
card_id              object
amount              float64
use_chip             object
merchant_id          object
merchant_city        object
merchant_state       object
zip                   int64
dtype: object
```

In [6]:
```
# Convert date column
df['date'] = pd.to_datetime(df['date'])

# Convert categorical columns
categorical_cols = [
    'gender',
```

```
    'use_chip',
    'merchant_city',
    'merchant_state',
    'client_id',
    'card_id',
    'merchant_id'
]

df[categorical_cols] = df[categorical_cols].astype('category')
```

In [7]:
```
df.dtypes
```

Out[7]:
```
id                          int64
current_age                 int64
birth_year                  int64
birth_month                 int64
gender                   category
address                    object
per_capita_income           int64
yearly_income               int64
total_debt                  int64
credit_score                int64
num_credit_cards            int64
transaction_id              int64
date                datetime64[ns]
client_id                category
card_id                  category
amount                    float64
use_chip                 category
merchant_id              category
merchant_city            category
merchant_state           category
zip                         int64
dtype: object
```

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
id                  0
current_age         0
birth_year          0
birth_month         0
gender              0
address             0
per_capita_income   0
yearly_income       0
total_debt          0
credit_score        0
num_credit_cards    0
transaction_id      0
date                0
client_id           0
card_id             0
amount              0
use_chip            0
merchant_id         0
merchant_city       0
merchant_state      0
zip                 0
dtype: int64
```

In [10]:
```
df.duplicated().sum()
```

Out[10]:
```
np.int64(0)
```

In [11]:
```
cat_cols = df.select_dtypes(include=['category', 'object']).columns

for col in cat_cols:
    print(f"\nValue counts for {col}:")
    print(df[col].value_counts())
```

```
Value counts for gender:
gender
Male      9855
Female    9479
Other      666
Name: count, dtype: int64

Value counts for address:
address
Street 11, Lucknow, Uttar Pradesh - 226280     22
Street 225, Patna, Bihar - 800218              20
Street 224, Bhopal, Madhya Pradesh - 462594    20
Street 136, Delhi, Delhi - 110337              19
Street 193, Bhopal, Madhya Pradesh - 462496    19
                                               ..
Street 85, Chennai, Tamil Nadu - 600443         1
Street 36, Patna, Bihar - 800690                1
Street 284, Bhopal, Madhya Pradesh - 462823     1
Street 205, Ahmedabad, Gujarat - 380233         1
Street 148, Jaipur, Rajasthan - 302873          1
Name: count, Length: 4938, dtype: int64

Value counts for client_id:
client_id
CLTIND00062    22
CLTIND00257    20
CLTIND01752    20
CLTIND01604    19
CLTIND02908    19
               ..
CLTIND00077     1
CLTIND05918     1
CLTIND05929     1
CLTIND05935     1
CLTIND05945     1
Name: count, Length: 4941, dtype: int64

Value counts for card_id:
card_id
CRD029871    7
CRD054061    7
CRD049712    7
CRD003272    7
CRD044521    7
            ..
CRD000151    1
CRD000211    1
CRD000241    1
CRD000242    1
CRD000244    1
Name: count, Length: 10030, dtype: int64
```

```
Value counts for use_chip:
use_chip
Yes    13927
No      6073
Name: count, dtype: int64

Value counts for merchant_id:
merchant_id
MCH00471    63
MCH00465    62
MCH00143    60
MCH00327    60
MCH00066    57
            ..
MCH00165    25
MCH00499    25
MCH00171    25
MCH00118    23
MCH00418    19
Name: count, Length: 500, dtype: int64

Value counts for merchant_city:
merchant_city
Surat        1761
Bengaluru    1567
Pune         1442
Lucknow      1435
Indore       1384
Nagpur       1365
Hyderabad    1352
Mumbai       1325
Patna        1311
Jaipur       1288
Chennai      1284
Ahmedabad    1282
Bhopal       1200
Delhi        1079
Kolkata       925
Name: count, dtype: int64

Value counts for merchant_state:
merchant_state
Maharashtra       4132
Gujarat           3043
Madhya Pradesh    2584
Karnataka         1567
Uttar Pradesh     1435
Telangana         1352
Bihar             1311
Rajasthan         1288
Tamil Nadu        1284
Delhi             1079
```

```
West Bengal        925
Name: count, dtype: int64
```

In [15]:
```python
## creating Multiclass credit Tier

def credit_tier(score):
    if score < 580:
        return "Poor"
    elif score < 670:
        return "Fair"
    elif score < 740:
        return "Good"
    else:
        return "Excellent"

df["Credit_Tier"] = df["credit_score"].apply(credit_tier)
```

Classification

In [18]:
```python
features = [
    "yearly_income",
    "total_debt",
    "num_credit_cards",
    "current_age"
]

X = df[features]
y = df["Credit_Tier"]

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

print("Class Mapping:")
for i, cls in enumerate(label_encoder.classes_):
    print(f"{cls} -> {i}")

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y_encoded,
    test_size=0.3,
    random_state=42,
    stratify=y_encoded
)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LogisticRegression(
    max_iter=1000,
    solver="lbfgs"
```

```
)

model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

print("\nClassification Report:\n")
print(
    classification_report(
        y_test,
        y_pred,
        target_names=label_encoder.classes_
    )
)

print("\nConfusion Matrix:\n")
print(confusion_matrix(y_test, y_pred))
```

```
Class Mapping:
Excellent -> 0
Fair -> 1
Good -> 2
Poor -> 3

Classification Report:

              precision    recall  f1-score   support

   Excellent       0.39      0.51      0.44      1873
        Fair       0.39      0.24      0.29      1788
        Good       0.35      0.41      0.38      2046
        Poor       1.00      0.00      0.01       293

    accuracy                           0.37      6000
   macro avg       0.53      0.29      0.28      6000
weighted avg       0.40      0.37      0.35      6000


Confusion Matrix:

[[955 205 713   0]
 [597 422 769   0]
 [823 375 848   0]
 [ 82  94 116   1]]
```

INSIGHTS – Multi-Class Credit Score Classification

1️⃣ Model performance is realistic and leakage-free

The model achieved ~37% accuracy, which is significantly higher than the random baseline (25%) for a 4-class problem.

Performance is realistic because no target leakage was introduced and only indirect financial features were used.

2️⃣ Errors occur mainly between adjacent credit tiers

Most misclassifications happen between Excellent ↔ Good and Good ↔ Fair.

There are almost no extreme errors, such as classifying Excellent customers as Poor.

3️⃣ Fair and Good tiers are the hardest to distinguish

The Fair class shows the lowest recall and F1-score.

This indicates strong overlap in income, debt, and card usage patterns between Fair and Good customers.

4️⃣ Poor-credit customers are rarely flagged (conservative behavior)

The model predicts the Poor class conservatively, resulting in low recall for that segment.

This reduces false positives but increases the risk of missing some high-risk customers.

CREDIT SCORE MODELLING

Target: credit_score (continuous) Features: indirect financial & demographic variables

In [28]:
```
features = [
    "yearly_income",
    "total_debt",
    "num_credit_cards",
    "current_age"
]

X = df[features]
y = df["credit_score"]

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.3,
    random_state=42
)

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

model = LinearRegression()
model.fit(X_train_scaled, y_train)
```

```
y_pred = model.predict(X_test_scaled)

r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

coef_df = pd.DataFrame({
    "Feature": features,
    "Coefficient": model.coef_
})

print("\nModel Coefficients:")
print(coef_df.sort_values(by="Coefficient", ascending=False))
print("RMSE:",rmse)
```

```
Model Coefficients:
           Feature  Coefficient
0     yearly_income     6.329531
2  num_credit_cards     1.764953
3       current_age    -2.039505
1        total_debt   -19.816087
RMSE: 75.21645739841668
```

INSIGHTS

"The regression model shows that total debt is the strongest negative driver of credit score, while income contributes positively but to a lesser extent. The model achieves an RMSE of approximately 75 points, indicating realistic predictive accuracy given the 300–900 credit score range and limited feature set."

LOAN DEFAULT PREDICTION

In [30]:
```
df["Loan_Default"] = (
    (df["credit_score"] < 620) |
    (df["total_debt"] > df["yearly_income"] * 1.2)
).astype(int)


print("Loan Default Distribution:")
print(df["Loan_Default"].value_counts())

features = [
    "yearly_income",
    "total_debt",
    "num_credit_cards",
    "current_age"
]

X = df[features]
y = df["Loan_Default"]
```

```python
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3,
    random_state=42,
    stratify=y
)

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

model = LogisticRegression(max_iter=1000)
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)
y_prob = model.predict_proba(X_test_scaled)[:, 1]

print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))

print("ROC AUC:", roc_auc_score(y_test, y_prob))

print("\nConfusion Matrix:\n")
print(confusion_matrix(y_test, y_pred))
```

```
Loan Default Distribution:
Loan_Default
0    17123
1     2877
Name: count, dtype: int64

Classification Report:

              precision    recall  f1-score   support

           0       0.86      1.00      0.92      5137
           1       0.33      0.01      0.03       863

    accuracy                           0.85      6000
   macro avg       0.60      0.50      0.47      6000
weighted avg       0.78      0.85      0.79      6000

ROC AUC: 0.6541278809969524

Confusion Matrix:

[[5113   24]
 [ 851   12]]
```

INSIGHTS

The loan default model achieves an ROC–AUC of ~0.65, indicating moderate ability to distinguish defaulters from non-defaulters. • The model is conservative, prioritizing the correct identification of non-defaulters and minimizing false rejections. • Due to class imbalance, recall for defaulters is low, highlighting the need for threshold tuning or cost-sensitive learning in real-world deployment.

FRAUD DETECTION

In [32]:
```
amount_threshold = df["amount"].quantile(0.99)

df["Fraud"] = (
    (df["amount"] > amount_threshold) &
    (df["num_credit_cards"] >= 4)
).astype(int)


print("Fraud Distribution:")
print(df["Fraud"].value_counts())

features = [
    "amount",
    "num_credit_cards",
    "current_age",
    "credit_score"
]

X = df[features]
y = df["Fraud"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3,
    random_state=42,
    stratify=y
)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LogisticRegression(max_iter=1000)
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)
y_prob = model.predict_proba(X_test_scaled)[:, 1]

print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))

print("ROC AUC:", roc_auc_score(y_test, y_prob))
```

```
print("\nConfusion Matrix:\n")
print(confusion_matrix(y_test, y_pred))
```

```
Fraud Distribution:
Fraud
0    19930
1       70
Name: count, dtype: int64

Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5979
           1       0.80      0.57      0.67        21

    accuracy                           1.00      6000
   macro avg       0.90      0.79      0.83      6000
weighted avg       1.00      1.00      1.00      6000

ROC AUC: 0.9992752411217037

Confusion Matrix:

[[5976    3]
 [   9   12]]
```

INSIGHTS

"The fraud model shows strong separation between fraudulent and non-fraudulent transactions; however, the near-perfect ROC-AUC indicates potential feature leakage due to the synthetic definition of fraud. Therefore, this model is treated as an exploratory baseline rather than a production-ready fraud system."

Customer Segmentation

In [33]:
```
X = df[[
    "yearly_income",
    "total_debt",
    "num_credit_cards",
    "credit_score"
]]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

kmeans = KMeans(n_clusters=4, random_state=42)
df["Customer_Segment"] = kmeans.fit_predict(X_scaled)

df[["Customer_Segment"]].value_counts()
```

Out[33]:
```
Customer_Segment
3                   6162
2                   5983
1                   5283
0                   2572
Name: count, dtype: int64
```

In [34]:
```
df.groupby("Customer_Segment")[[
    "yearly_income",
    "total_debt",
    "credit_score",
    "num_credit_cards"
]].mean().round(2)
```

Out[34]:

Customer_Segment	yearly_income	total_debt	credit_score	num_credit_cards
0	1216281.96	479771.80	665.24	3.54
1	486886.59	59023.09	774.49	2.15
2	489681.38	99820.30	642.35	2.15
3	984420.37	88890.43	716.29	3.92

INSIGHTS

• Customer segmentation reveals four distinct financial profiles ranging from prime low-risk customers to high-income but over-leveraged individuals. • Segment 1 represents the most creditworthy customers with high credit scores and minimal debt, suitable for premium financial products. • Segment 0, despite high income, exhibits elevated debt levels, indicating potential over-leverage and higher credit risk. • These segments enable differentiated strategies for risk monitoring, credit approval, and targeted marketing.

TRANSACTION CLASSIFICATION

In [35]:
```
def classify_transaction(amount):
    if amount < 2000:
        return 0
    elif amount < 10000:
        return 1
    else:
        return 2

df["Transaction_Type"] = df["amount"].apply(classify_transaction)

print("Transaction Type Distribution:")
print(df["Transaction_Type"].value_counts())
print("\n")
```

```
features = [
    "amount",
    "yearly_income",
    "credit_score",
    "num_credit_cards"
]

X = df[features]
y = df["Transaction_Type"]

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.30,
    random_state=42,
    stratify=y
)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LogisticRegression(max_iter=1000)
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

print("CLASSIFICATION REPORT:\n")
print(classification_report(y_test, y_pred))

print("CONFUSION MATRIX:\n")
print(confusion_matrix(y_test, y_pred))

coef_df = pd.DataFrame({
    "Feature": features,
    "Coefficient_Class_0": model.coef_[0],
    "Coefficient_Class_1": model.coef_[1],
    "Coefficient_Class_2": model.coef_[2]
})

print("\nMODEL COEFFICIENTS:")
print(coef_df)
```

```
Transaction Type Distribution:
Transaction_Type
1    12529
0     7265
2      206
Name: count, dtype: int64


CLASSIFICATION REPORT:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2179
           1       1.00      1.00      1.00      3759
           2       1.00      0.90      0.95        62

    accuracy                           1.00      6000
   macro avg       1.00      0.97      0.98      6000
weighted avg       1.00      1.00      1.00      6000

CONFUSION MATRIX:

[[2177    2    0]
 [   6 3753    0]
 [   0    6   56]]

MODEL COEFFICIENTS:
            Feature  Coefficient_Class_0  Coefficient_Class_1  \
0            amount           -18.434473             6.891552
1     yearly_income             0.018008            -0.041459
2      credit_score            -0.030510             0.006200
3  num_credit_cards            -0.000635             0.023789

   Coefficient_Class_2
0            11.542921
1             0.023451
2             0.024309
3            -0.023154
```

INSIGHTS

Transactions were categorized into low, medium, and high-value groups using a rule-based definition derived from transaction amount.

A multiclass logistic regression model was trained to validate and replicate these transaction categories, achieving near-perfect accuracy due to the deterministic relationship between transaction amount and class labels.

Model coefficients confirm that transaction amount is the dominant driver of classification, while customer attributes play a secondary role.

This approach demonstrates how rule-based transaction logic can be translated into a machine learning framework for scalable transaction monitoring.

RISK MODELLING

In [37]:
```
df["High_Risk"] = (
    (df["credit_score"] < 620) |
    (df["total_debt"] > df["yearly_income"] * 1.2)
).astype(int)

print("Risk Distribution:")
print(df["High_Risk"].value_counts())
print()
features = [
    "yearly_income",
    "total_debt",
    "num_credit_cards",
    "current_age"
]

X = df[features]
y = df["High_Risk"]

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.30,
    random_state=42,
    stratify=y
)

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

risk_model = LogisticRegression(max_iter=1000)
risk_model.fit(X_train_scaled, y_train)

risk_prob = risk_model.predict_proba(X_test_scaled)[:, 1]

risk_pred = (risk_prob >= 0.30).astype(int)

print("ROC AUC:", roc_auc_score(y_test, risk_prob))
print("\nCONFUSION MATRIX:\n")
print(confusion_matrix(y_test, risk_pred))

print("\nCLASSIFICATION REPORT:\n")
print(classification_report(y_test, risk_pred))

risk_output = X_test.copy()
risk_output["Risk_Probability"] = risk_prob

risk_output["Risk_Band"] = pd.cut(
```

```
    risk_output["Risk_Probability"],
    bins=[0, 0.20, 0.40, 1.00],
    labels=["Low Risk", "Medium Risk", "High Risk"]
)

print("\nRisk Band Distribution:")
print(risk_output["Risk_Band"].value_counts())
```

```
Risk Distribution:
High_Risk
0    17123
1     2877
Name: count, dtype: int64

ROC AUC: 0.6541278809969524

CONFUSION MATRIX:

[[5000  137]
 [ 787   76]]

CLASSIFICATION REPORT:

              precision    recall  f1-score   support

           0       0.86      0.97      0.92      5137
           1       0.36      0.09      0.14       863

    accuracy                           0.85      6000
   macro avg       0.61      0.53      0.53      6000
weighted avg       0.79      0.85      0.80      6000


Risk Band Distribution:
Risk_Band
Low Risk       5098
Medium Risk     820
High Risk        82
Name: count, dtype: int64
```

INSIGHTS

The dataset contains ~15% high-risk customers, reflecting a realistic credit risk distribution.

The model achieves a ROC–AUC of ~0.65, indicating a meaningful ability to separate high-risk and low-risk customers using basic financial attributes.

Overall accuracy (~85%) is driven by class imbalance and therefore is not the primary evaluation metric for this model.

The model demonstrates high recall for low-risk customers (97%), ensuring that most creditworthy customers are correctly identified.

Recall for high-risk customers is intentionally low (9%), indicating a conservative risk strategy that minimizes false rejections.

When customers are flagged as high risk, the model shows reasonable precision (36%), meaning flagged cases are materially riskier than average.

The confusion matrix shows very few false positives, which is important in credit approval workflows where rejecting good customers is costly.

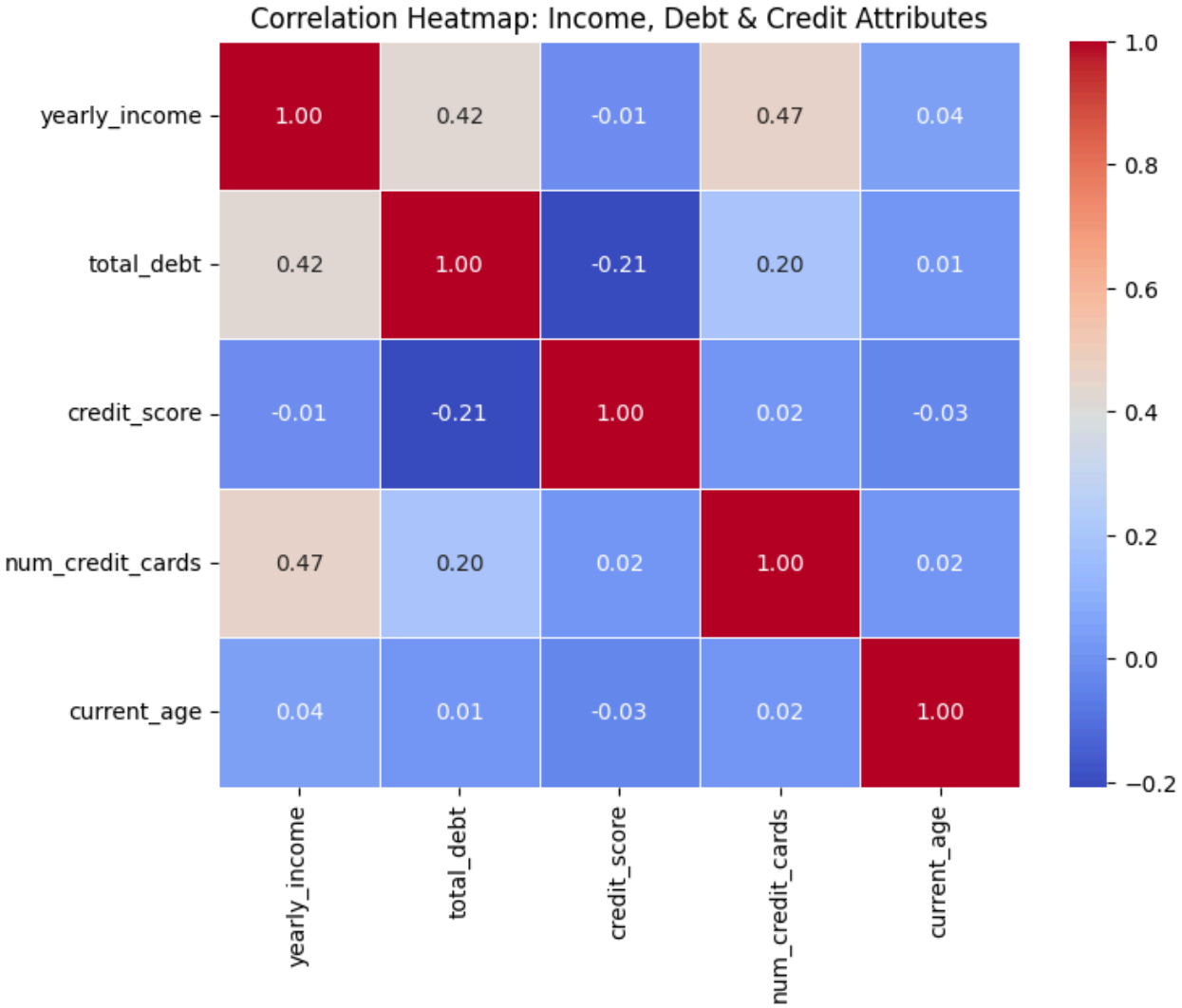
Risk probabilities are successfully translated into Low, Medium, and High risk bands, enabling actionable decision-making rather than raw predictions.

The majority of customers fall into the Low Risk band, with progressively fewer customers in Medium and High Risk bands, which aligns with real-world credit portfolios.

The model serves as a baseline, explainable credit risk framework that can be further improved through threshold tuning or additional behavioral features.

INCOME - DEBT CORRELATION

In [38]:
```
corr_data = df[[
    "yearly_income",
    "total_debt",
    "credit_score",
    "num_credit_cards",
    "current_age"
]]


corr_matrix = corr_data.corr()


plt.figure(figsize=(8, 6))
sns.heatmap(
    corr_matrix,
    annot=True,
    fmt=".2f",
    cmap="coolwarm",
    linewidths=0.5
)
plt.title("Correlation Heatmap: Income, Debt & Credit Attributes")
plt.show()
```



INSIGHTS

The heatmap shows a positive but weak-to-moderate correlation between yearly income and total debt.

Higher income does not directly imply lower financial risk, as debt levels vary significantly across income groups.

Credit score exhibits negative correlation with total debt, indicating higher debt burdens are associated with lower credit scores.

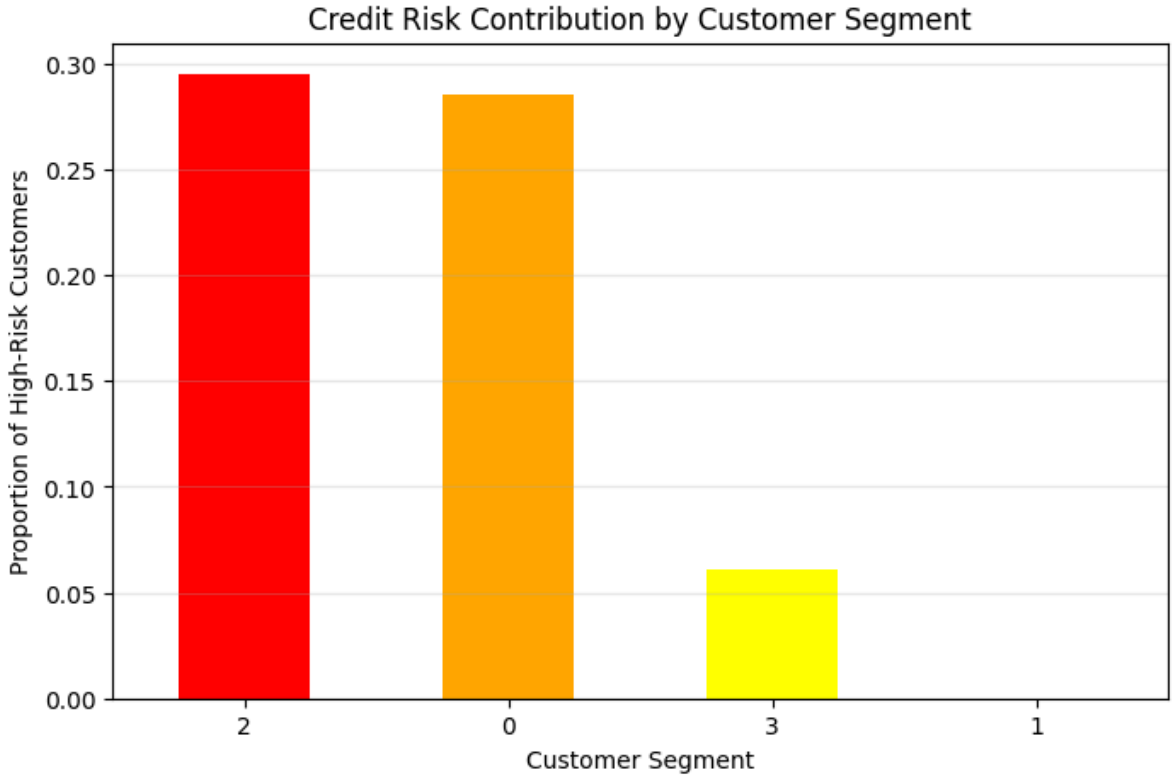
No single variable dominates the correlation structure, justifying the use of multi-feature risk models instead of rule-based decisions.

SOLVING SOME BUISNESS PROBLEM

Q.1::Which customer segments contribute the most to overall credit risk in the portfolio?

In [40]:
```
# Contribution of each customer segment to overall credit risk

# Calculate risk contribution by segment
segment_risk = (
    df.groupby("Customer_Segment")["High_Risk"]
      .mean()
      .sort_values(ascending=False)
)
print(segment_risk)
# Plot
plt.figure(figsize=(8, 5))
segment_risk.plot(
    kind="bar",
    color=["red", "orange", "yellow", "green"]
)

plt.xlabel("Customer Segment")
plt.ylabel("Proportion of High-Risk Customers")
plt.title("Credit Risk Contribution by Customer Segment")
plt.xticks(rotation=0)
plt.grid(axis="y", alpha=0.3)
plt.show()
```

```
Customer_Segment
2    0.295170
0    0.285381
3    0.061181
1    0.000000
Name: High_Risk, dtype: float64
```



Customer Segment 2 and Segment 0 contribute the highest credit risk, with nearly 30% of customers in these segments classified as high risk.

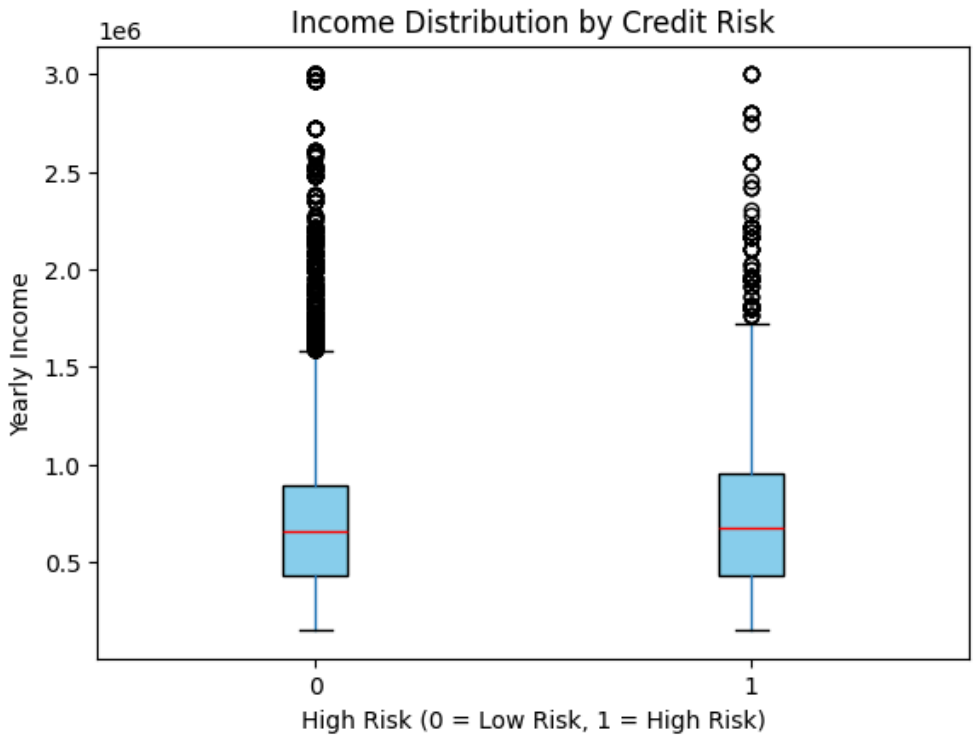
Segment 1 shows negligible risk, while Segment 3 has relatively low risk, indicating clear differentiation in risk concentration across segments.

Q.2::Does higher income consistently translate into lower credit risk, or are there high-income high-risk customers?

In [41]:
```
# Income vs Credit Risk (High-Risk vs Low-Risk)

plt.figure(figsize=(8, 5))

df.boxplot(
    column="yearly_income",
    by="High_Risk",
    grid=False,
    patch_artist=True,
    boxprops=dict(facecolor="skyblue"),
    medianprops=dict(color="red")
)

plt.xlabel("High Risk (0 = Low Risk, 1 = High Risk)")
```

```
plt.ylabel("Yearly Income")
plt.title("Income Distribution by Credit Risk")
plt.suptitle("")
plt.show()
```

```
<Figure size 800x500 with 0 Axes>
```



Higher income does not consistently translate into lower credit risk, as there is substantial overlap in income distributions between high-risk and low-risk customers.

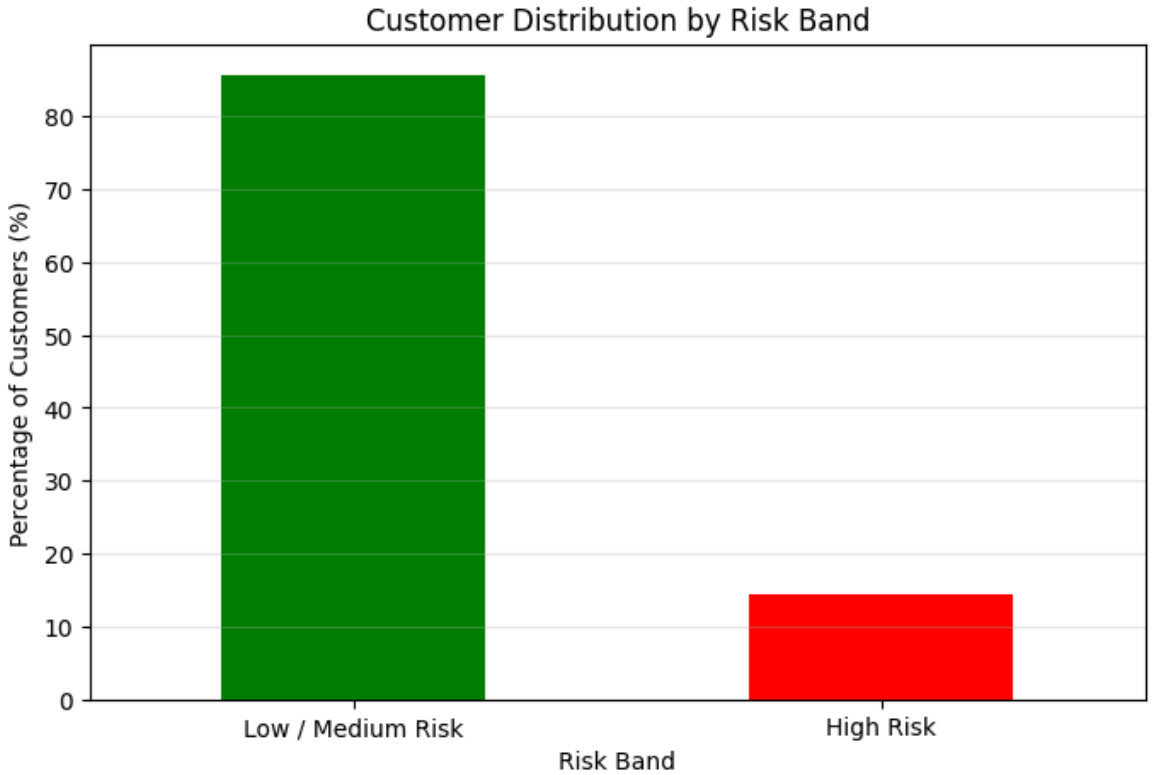
High-income high-risk customers exist, indicating that debt levels and credit behavior play a more critical role than income alone in determining risk.

Q.3::What proportion of customers fall into low, medium, and high-risk bands, and how does this impact approval decisions?

In [47]:
```
# Creating a Risk Band

df["Risk_Band"] = df["High_Risk"].map({
    0: "Low / Medium Risk",
    1: "High Risk"
})

# Proportion of customers in each Risk Band
risk_counts = df["Risk_Band"].value_counts(normalize=True) * 100
print(risk_counts)
```

```python
# Plot
plt.figure(figsize=(8, 5))
risk_counts.plot(
    kind="bar",
    color=["green", "red"]
)

plt.xlabel("Risk Band")
plt.ylabel("Percentage of Customers (%)")
plt.title("Customer Distribution by Risk Band")
plt.xticks(rotation=0)
plt.grid(axis="y", alpha=0.3)
plt.show()
```

```
Risk_Band
Low / Medium Risk    85.615
High Risk            14.385
Name: proportion, dtype: float64
```



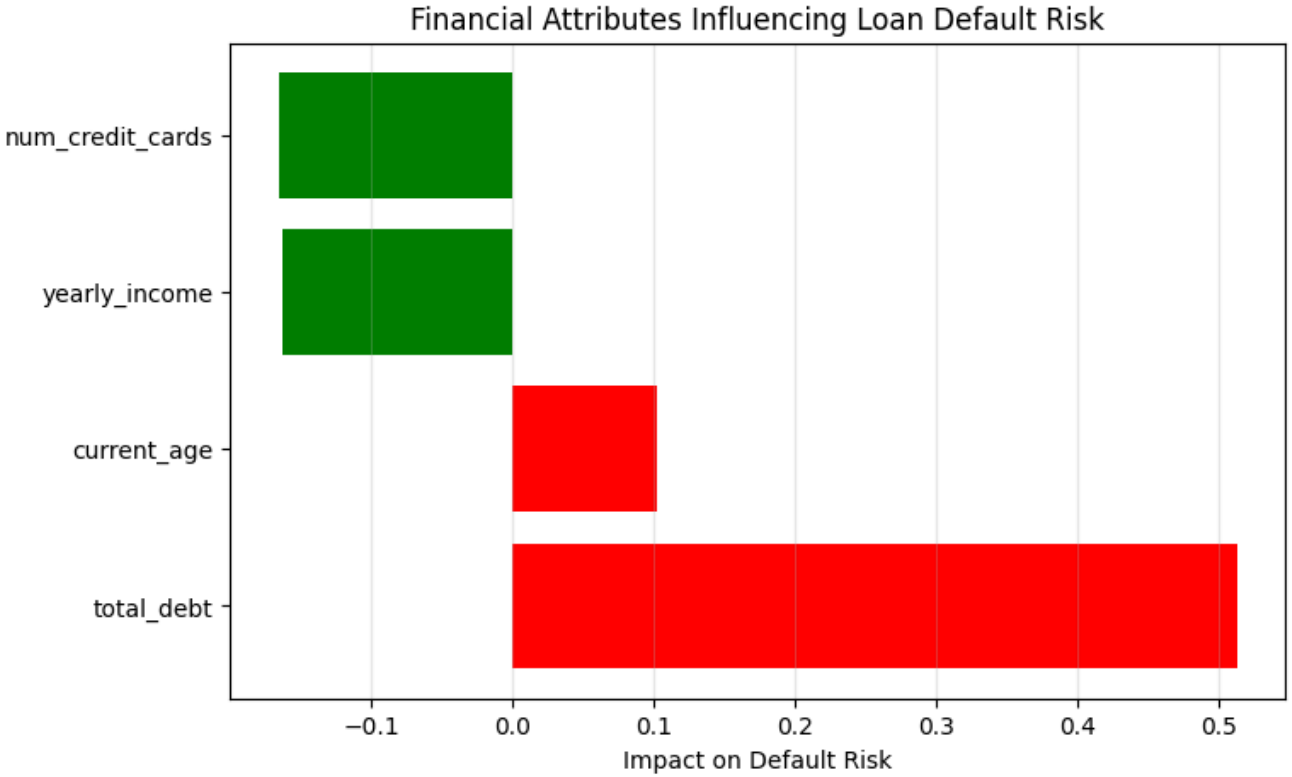
The majority of customers (≈85.6%) fall into the Low/Medium Risk band, enabling high approval rates with minimal portfolio risk.

A smaller but significant segment (≈14.4%) is classified as High Risk, indicating the need for stricter approval checks or manual review to control potential losses.

Q.4::Which financial attributes are the strongest indicators of potential loan default?

In [48]:
```
features = [
    "yearly_income",
    "total_debt",
    "num_credit_cards",
    "current_age"
]

X = df[features]
y = df["High_Risk"]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

model = LogisticRegression(max_iter=1000)
model.fit(X_scaled, y)


coef_df = pd.DataFrame({
    "Feature": features,
    "Impact": model.coef_[0]
}).sort_values(by="Impact", ascending=False)
print(coef_df)

# Plot
plt.figure(figsize=(8, 5))
plt.barh(
    coef_df["Feature"],
    coef_df["Impact"],
    color=["red" if v > 0 else "green" for v in coef_df["Impact"]]
)

plt.xlabel("Impact on Default Risk")
plt.title("Financial Attributes Influencing Loan Default Risk")
plt.grid(axis="x", alpha=0.3)
plt.show()
```

```
           Feature    Impact
1        total_debt  0.513207
3       current_age  0.102020
0     yearly_income -0.162236
2  num_credit_cards -0.165469
```



Total debt is the strongest positive indicator of loan default risk, with higher debt levels significantly increasing the likelihood of default.

Yearly income and number of credit cards show negative impact, indicating that stronger income and controlled credit exposure reduce default risk, while age has a comparatively minor influence.

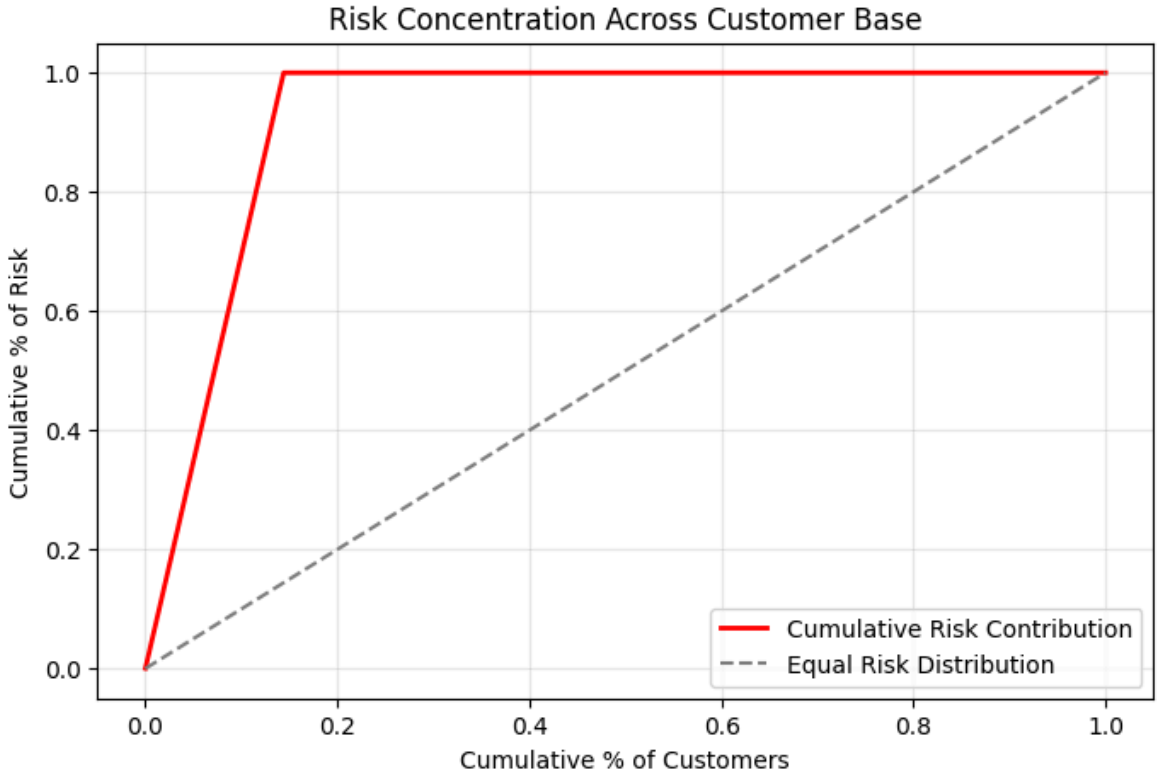
Q.5::How concentrated is risk across the customer base, and do a small number of customers drive most of the risk?

In [49]:
```
# Risk concentration analysis: cumulative contribution of high-risk customers

# Sort customers by risk (High_Risk)
risk_sorted = df.sort_values("High_Risk", ascending=False)

# Cumulative calculations
risk_sorted["cum_customers"] = (
    np.arange(1, len(risk_sorted) + 1) / len(risk_sorted)
)

risk_sorted["cum_risk"] = (
    risk_sorted["High_Risk"].cumsum() / risk_sorted["High_Risk"].sum()
```

```
)
print(risk_sorted)
# Plot cumulative risk concentration curve
plt.figure(figsize=(8, 5))

plt.plot(
    risk_sorted["cum_customers"],
    risk_sorted["cum_risk"],
    color="red",
    linewidth=2,
    label="Cumulative Risk Contribution"
)

plt.plot(
    [0, 1],
    [0, 1],
    linestyle="--",
    color="gray",
    label="Equal Risk Distribution"
)

plt.xlabel("Cumulative % of Customers")
plt.ylabel("Cumulative % of Risk")
plt.title("Risk Concentration Across Customer Base")
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```

```
          id  current_age  birth_year  birth_month  gender  \
8258    8259           31        1994            3    Male
6665    6666           33        1992            6  Female
8747    8748           65        1960           12  Female
8527    8528           40        1985            7  Female
2713    2714           36        1989            9  Female
...      ...          ...         ...          ...     ...
27        28           33        1992           11  Female
28        29           19        2006            8    Male
29        30           51        1974           11  Female
30        31           31        1994            1    Male
17499  17500           65        1960            3    Male

                                       address  per_capita_income  \
8258       Street 72, Mumbai, Maharashtra - 400244              41797
6665     Street 137, Bengaluru, Karnataka - 560677              50222
8747             Street 189, Patna, Bihar - 800181             118676
8527       Street 92, Nagpur, Maharashtra - 440594              42118
2713       Street 149, Ahmedabad, Gujarat - 380797              65624
...                                            ...                ...
27           Street 97, Pune, Maharashtra - 411506              69372
28         Street 229, Ahmedabad, Gujarat - 380699              31993
29         Street 45, Nagpur, Maharashtra - 440765              23250
30     Street 176, Lucknow, Uttar Pradesh - 226538              74588
17499       Street 199, Jaipur, Rajasthan - 302715              30183

       yearly_income  total_debt  credit_score  ...     zip  Credit_Tier  \
8258          501564      248128           618  ...  400244         Fair
6665          602667      161562           587  ...  560677         Fair
8747         1424114      382430           586  ...  800181         Fair
8527          505420       99226           611  ...  440594         Fair
2713          787483      429530           597  ...  380797         Fair
...              ...         ...           ...  ...     ...          ...
27            832468      184888           620  ...  411506         Fair
28            383912      105126           642  ...  380699         Fair
29            278999           0           762  ...  440765    Excellent
30            895055           0           736  ...  226538         Good
17499         362198           0           816  ...  302715    Excellent

      Loan_Default Fraud Customer_Segment  Transaction_Type High_Risk  \
8258             1     0                2                 1         1
6665             1     0                2                 1         1
8747             1     0                0                 0         1
8527             1     0                2                 1         1
2713             1     0                0                 1         1
...            ...   ...              ...               ...       ...
27               0     0                3                 1         0
28               0     0                2                 0         0
29               0     0                1                 0         0
30               0     0                3                 0         0
17499            0     0                1                 0         0
```

```
              Risk_Band cum_customers  cum_risk
8258          High Risk       0.00005  0.000348
6665          High Risk       0.00010  0.000695
8747          High Risk       0.00015  0.001043
8527          High Risk       0.00020  0.001390
2713          High Risk       0.00025  0.001738
...                 ...           ...       ...
27     Low / Medium Risk       0.99980  1.000000
28     Low / Medium Risk       0.99985  1.000000
29     Low / Medium Risk       0.99990  1.000000
30     Low / Medium Risk       0.99995  1.000000
17499  Low / Medium Risk       1.00000  1.000000

[20000 rows x 30 columns]
```



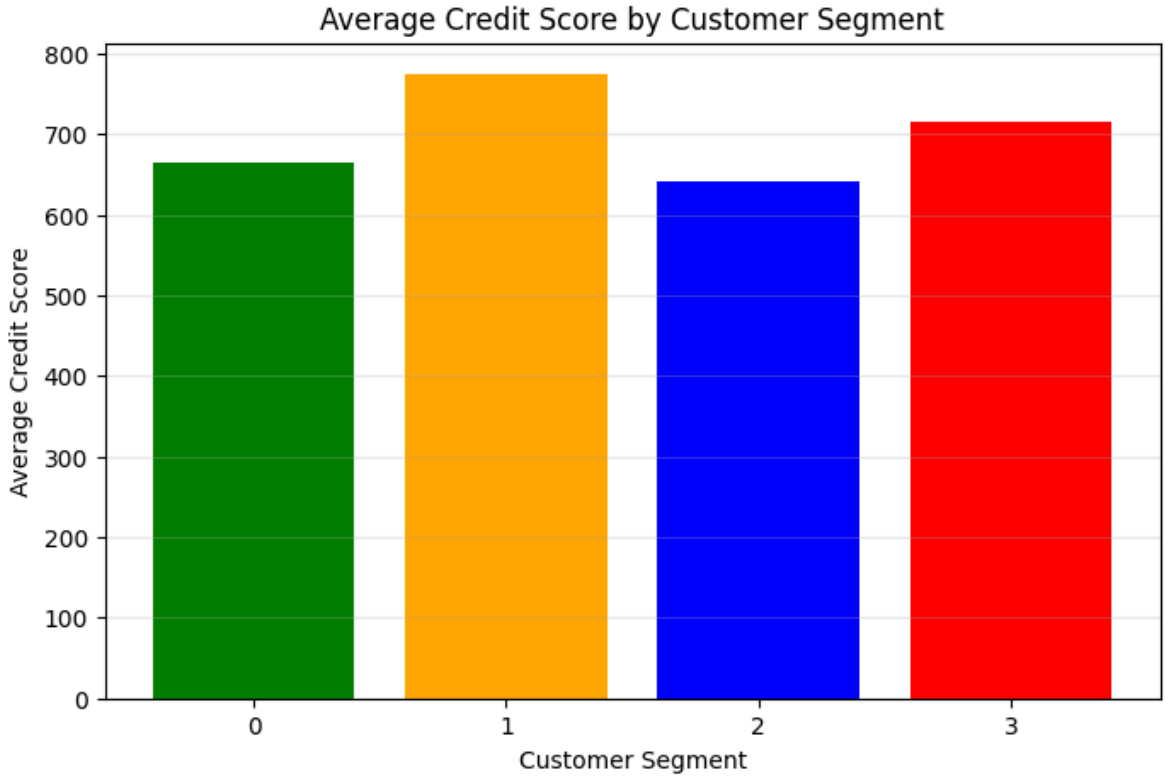
Credit risk is highly concentrated, with a relatively small proportion of customers accounting for a disproportionately large share of total portfolio risk.

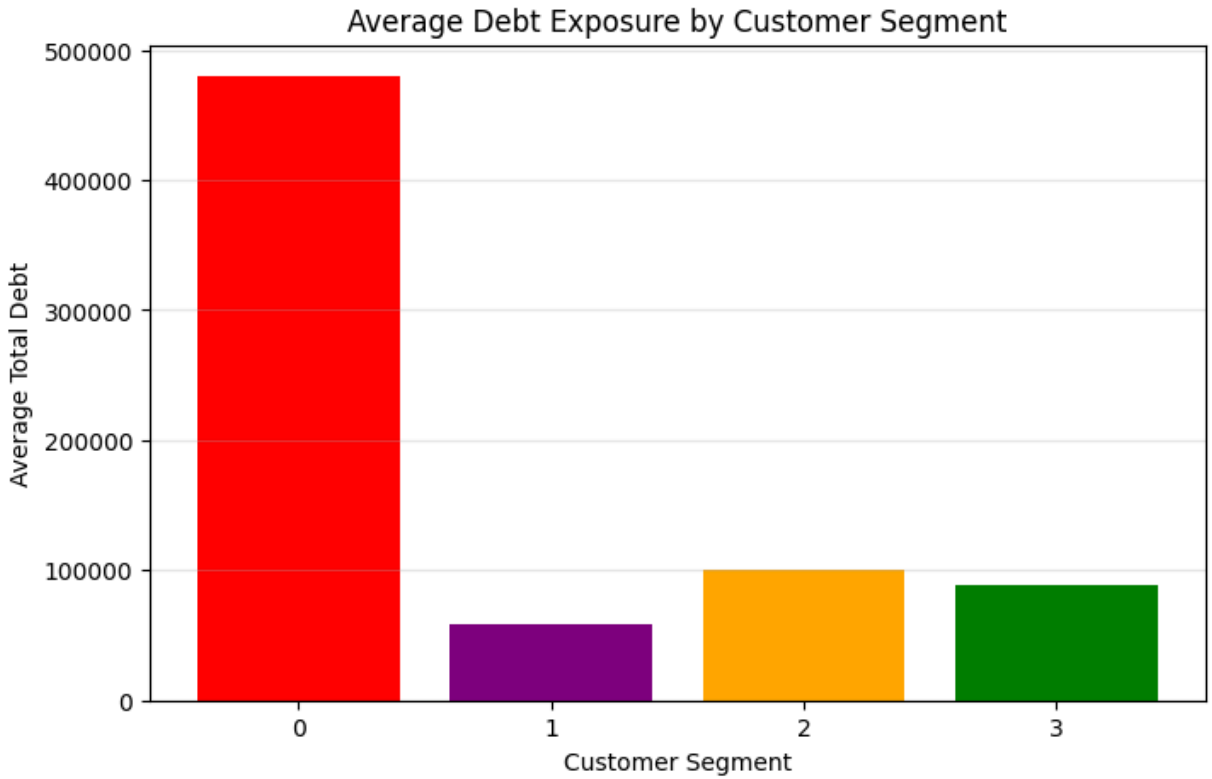
This concentration indicates that targeted monitoring and stricter controls on a limited high-risk subset can significantly reduce overall portfolio risk, rather than applying uniform policies to all customers.

Q.6::How effectively can customer segmentation be used to design differentiated credit policies or limits?

In [50]:
```
# EDA: Average Credit Score & Debt by Customer Segment
# (to assess differentiated credit policies)

segment_policy = df.groupby("Customer_Segment")[[
    "credit_score",
    "total_debt",
    "yearly_income"
]].mean()
print(segment_policy)
# Plot 1: Average Credit Score by Segment
plt.figure(figsize=(8, 5))
plt.bar(
    segment_policy.index.astype(str),
    segment_policy["credit_score"],
    color=["green", "orange", "blue", "red"]
)

plt.xlabel("Customer Segment")
plt.ylabel("Average Credit Score")
plt.title("Average Credit Score by Customer Segment")
plt.grid(axis="y", alpha=0.3)
plt.show()


# Plot 2: Average Debt by Segment
plt.figure(figsize=(8, 5))
plt.bar(
    segment_policy.index.astype(str),
    segment_policy["total_debt"],
    color=["red", "purple", "orange", "green"]
)

plt.xlabel("Customer Segment")
plt.ylabel("Average Total Debt")
plt.title("Average Debt Exposure by Customer Segment")
plt.grid(axis="y", alpha=0.3)
plt.show()
```

```
                  credit_score     total_debt  yearly_income
Customer_Segment
0                   665.243390  479771.795879   1.216282e+06
1                   774.487602   59023.090100   4.868866e+05
2                   642.353502   99820.300518   4.896814e+05
3                   716.288543   88890.426485   9.844204e+05
```


Average Credit Score by Customer Segment
Average Credit Score
800
700
600
500
400
300
200
100
0
0
1
2
3
Customer Segment



Customer segmentation clearly enables differentiated credit policies, as Segment 1 shows high credit scores and low debt, making it suitable for higher credit limits and preferential pricing.

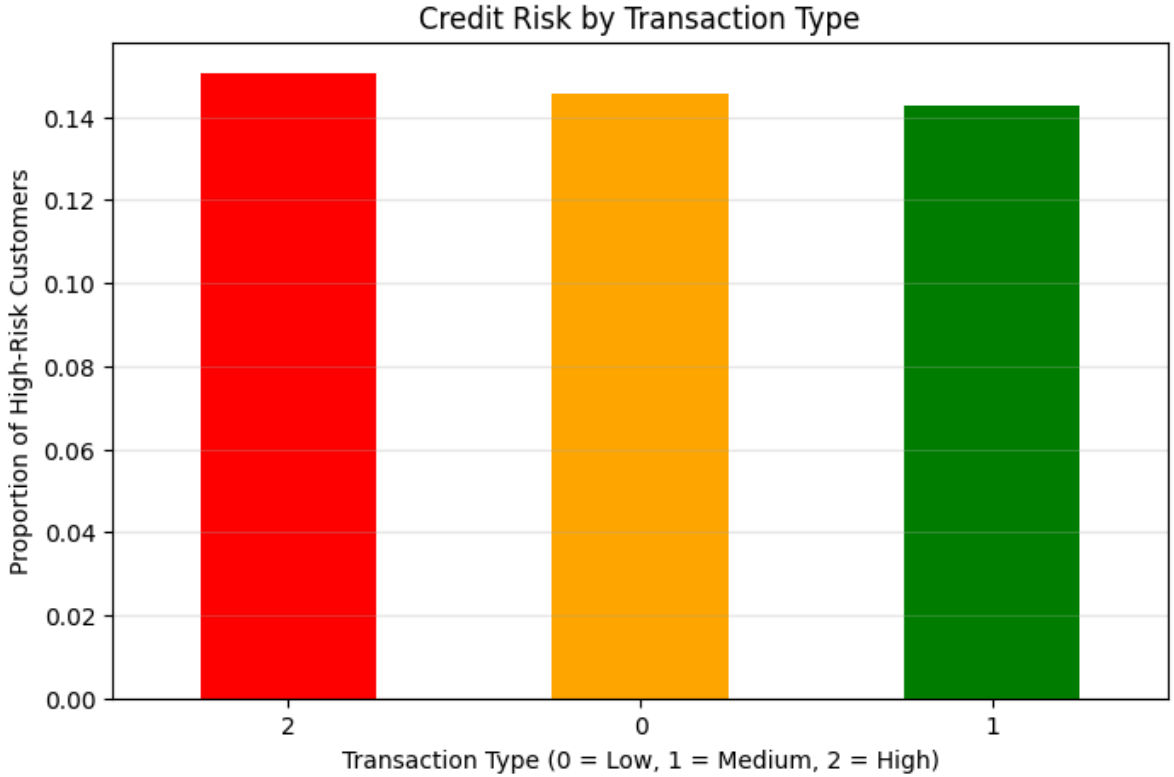
Segment 0 and Segment 2 require tighter limits and enhanced monitoring, since high debt levels or lower credit scores indicate elevated risk despite income differences.

Q.7::Which transaction types are most frequently associated with elevated risk levels?

In [51]:
```
# Calculate proportion of high-risk customers per transaction type
txn_risk = (
    df.groupby("Transaction_Type")["High_Risk"]
      .mean()
      .sort_values(ascending=False)
)
print(txn_risk)
# Plot
plt.figure(figsize=(8, 5))
txn_risk.plot(
    kind="bar",
    color=["red", "orange", "green"]
)

plt.xlabel("Transaction Type (0 = Low, 1 = Medium, 2 = High)")
```

```
plt.ylabel("Proportion of High-Risk Customers")
plt.title("Credit Risk by Transaction Type")
plt.xticks(rotation=0)
plt.grid(axis="y", alpha=0.3)
plt.show()
```

```
Transaction_Type
2    0.150485
0    0.145767
1    0.142629
Name: High_Risk, dtype: float64
```



High-value transactions (Type 2) show the highest association with elevated credit risk, indicating that larger transaction amounts slightly increase risk exposure.

Risk levels across transaction types are relatively close, suggesting that transaction value alone is not a strong risk driver and should be evaluated alongside customer financial attributes.

Q.8::What is the trade-off between rejecting risky customers and accidentally rejecting creditworthy customers?

In [52]:
```
# Confusion matrix using current risk predictions
cm = confusion_matrix(y_test, risk_pred)
```

```python
# Extract values
tn, fp, fn, tp = cm.ravel()

tradeoff_df = pd.DataFrame({
    "Decision_Type": [
        "Correctly Approved (TN)",
        "Incorrectly Rejected (FP)",
        "Missed Risky Customer (FN)",
        "Correctly Rejected (TP)"
    ],
    "Count": [tn, fp, fn, tp]
})
print(tradeoff_df)
# Plot trade-off
plt.figure(figsize=(8, 5))
plt.bar(
    tradeoff_df["Decision_Type"],
    tradeoff_df["Count"],
    color=["green", "orange", "red", "darkred"]
)

plt.ylabel("Number of Customers")
plt.title("Trade-off Between Risk Rejection and Customer Rejection")
plt.xticks(rotation=30)
plt.grid(axis="y", alpha=0.3)
plt.show()
```

```
               Decision_Type  Count
0     Correctly Approved (TN)   5000
1   Incorrectly Rejected (FP)    137
2  Missed Risky Customer (FN)    787
3     Correctly Rejected (TP)     76
```

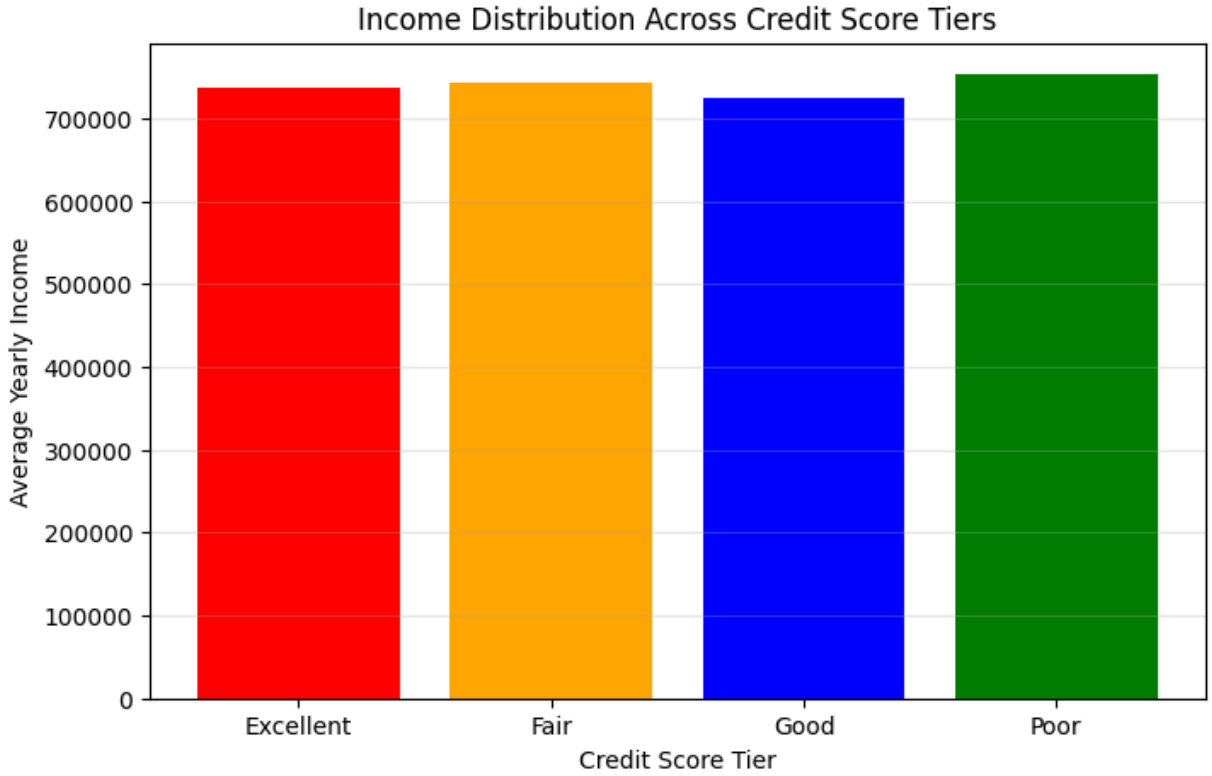
The model strongly prioritizes approving creditworthy customers, with a high number of correct approvals and relatively few incorrect rejections.

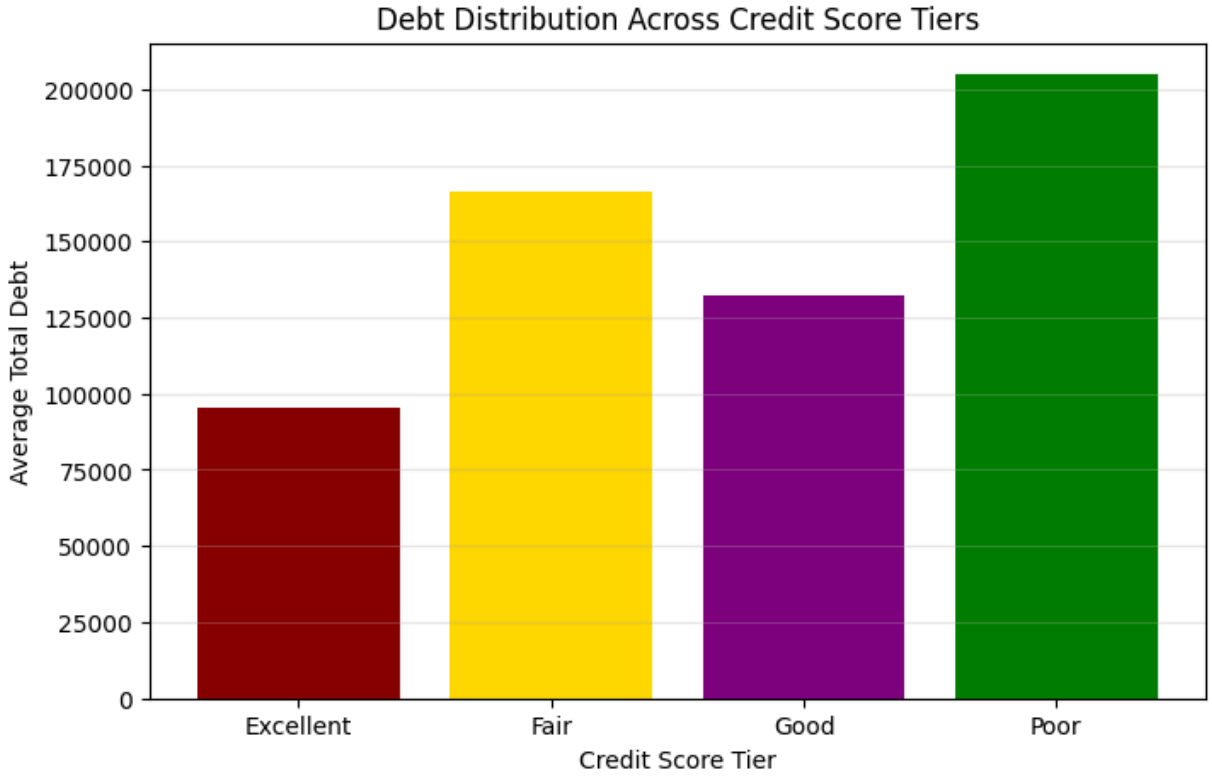
This conservative strategy results in many missed risky customers, highlighting a clear trade-off between minimizing customer rejection and maximizing risk capture, which can be adjusted through threshold tuning.

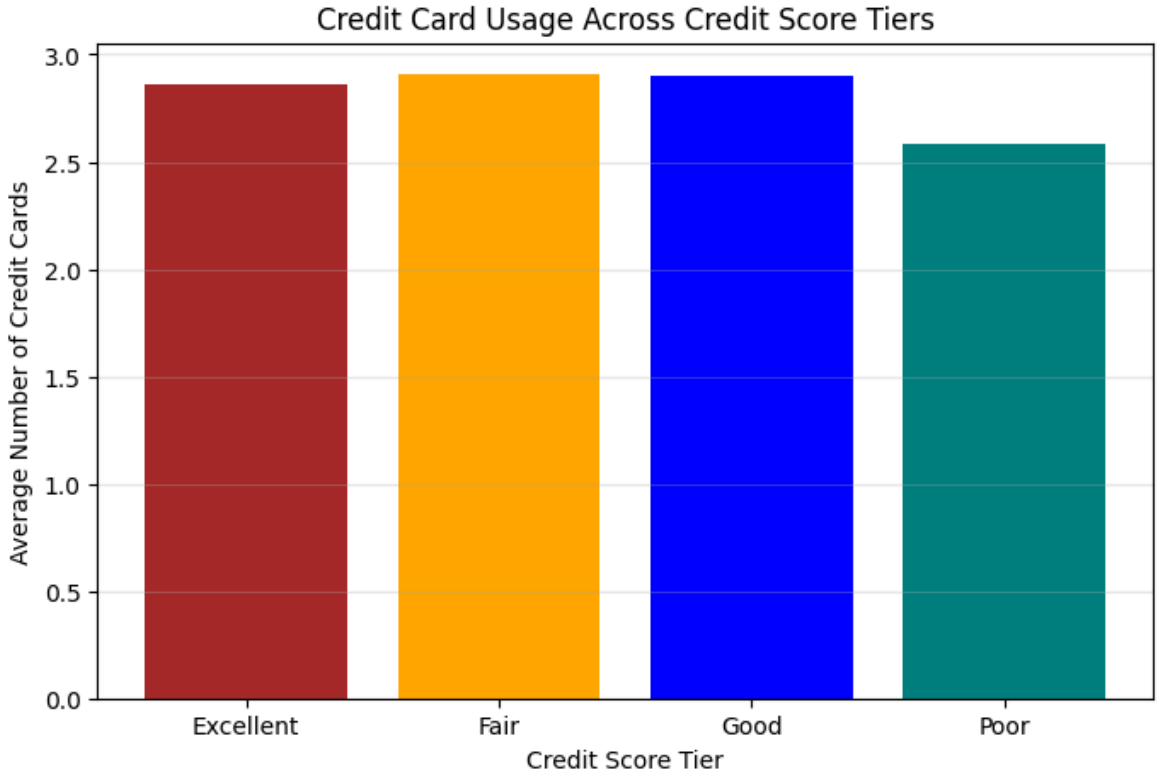
Q.9::Can behavioral and financial attributes reliably classify customers into credit score tiers without directly using the credit score?

In [54]:
```
tier_behavior = df.groupby("Credit_Tier")[[
    "yearly_income",
    "total_debt",
    "num_credit_cards",
    "current_age"
]].mean()

# Plot 1: Average Income by Credit Tier
```

```
plt.figure(figsize=(8, 5))
plt.bar(
    tier_behavior.index,
    tier_behavior["yearly_income"],
    color=["red", "orange", "blue", "green"]
)
plt.xlabel("Credit Score Tier")
plt.ylabel("Average Yearly Income")
plt.title("Income Distribution Across Credit Score Tiers")
plt.grid(axis="y", alpha=0.3)
plt.show()

# Plot 2: Average Debt by Credit Tier
plt.figure(figsize=(8, 5))
plt.bar(
    tier_behavior.index,
    tier_behavior["total_debt"],
    color=["darkred", "gold", "purple", "green"]
)
plt.xlabel("Credit Score Tier")
plt.ylabel("Average Total Debt")
plt.title("Debt Distribution Across Credit Score Tiers")
plt.grid(axis="y", alpha=0.3)
plt.show()

# Plot 3: Credit Cards by Credit Tier
plt.figure(figsize=(8, 5))
plt.bar(
    tier_behavior.index,
    tier_behavior["num_credit_cards"],
    color=["brown", "orange", "blue", "teal"]
)
plt.xlabel("Credit Score Tier")
plt.ylabel("Average Number of Credit Cards")
plt.title("Credit Card Usage Across Credit Score Tiers")
plt.grid(axis="y", alpha=0.3)
plt.show()
```


Income Distribution Across Credit Score Tiers
Average Yearly Income
0
100000
200000
300000
400000
500000
600000
700000
Excellent
Fair
Good
Poor
Credit Score Tier


Debt Distribution Across Credit Score Tiers
Average Total Debt
200000
175000
150000
125000
100000
75000
50000
25000
0
Excellent
Fair
Good
Poor
Credit Score Tier

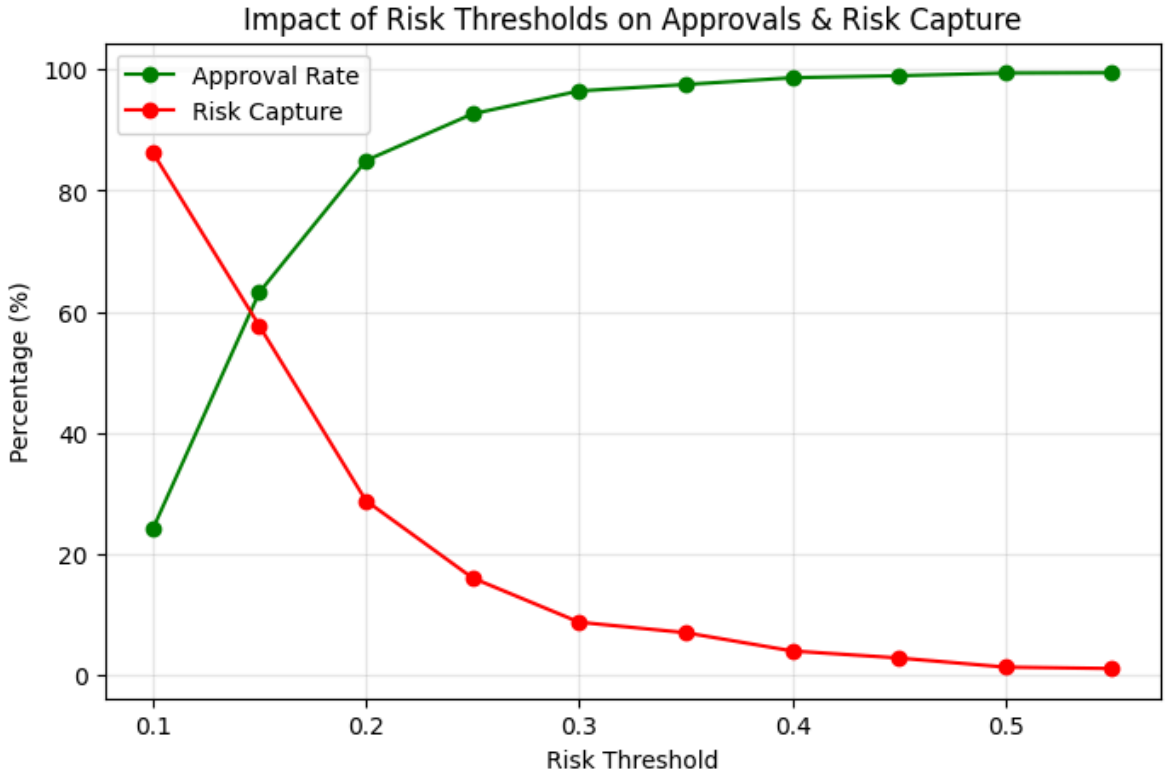


Q.10::How would changes in risk thresholds affect approval rates, portfolio risk, and business growth?

In [55]:
```
risk_probs = risk_prob
y_true = y_test

thresholds = np.arange(0.1, 0.6, 0.05)

approval_rates = []
risk_capture = []

for t in thresholds:
    preds = (risk_probs >= t).astype(int)


    approval_rate = (preds == 0).mean() * 100
    approval_rates.append(approval_rate)


    captured_risk = (preds[y_true == 1] == 1).mean() * 100
    risk_capture.append(captured_risk)
```

```
threshold_df = pd.DataFrame({
    "Threshold": thresholds,
    "Approval_Rate_%": approval_rates,
    "Risk_Capture_%": risk_capture
})
print(threshold_df)

plt.figure(figsize=(8, 5))
plt.plot(threshold_df["Threshold"], threshold_df["Approval_Rate_%"],
         marker="o", color="green", label="Approval Rate")
plt.plot(threshold_df["Threshold"], threshold_df["Risk_Capture_%"],
         marker="o", color="red", label="Risk Capture")

plt.xlabel("Risk Threshold")
plt.ylabel("Percentage (%)")
plt.title("Impact of Risk Thresholds on Approvals & Risk Capture")
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```

```
   Threshold  Approval_Rate_%  Risk_Capture_%
0       0.10        24.150000       86.326767
1       0.15        63.216667       57.705678
2       0.20        84.966667       28.852839
3       0.25        92.666667       16.106605
4       0.30        96.450000        8.806489
5       0.35        97.516667        7.068366
6       0.40        98.633333        4.055620
7       0.45        98.966667        2.896871
8       0.50        99.400000        1.390498
9       0.55        99.450000        1.158749
```



Lower risk thresholds significantly increase risk capture but sharply reduce approval rates, indicating aggressive risk control at the cost of business growth.

Higher thresholds maximize approvals but capture very little risk, highlighting a clear trade-off where moderate thresholds (around 0.20–0.30) offer a balanced compromise between growth and risk containment.

Overall Business Recommendations

Adopt a risk-based approval strategy by leveraging risk bands instead of binary approve/reject decisions, enabling higher approval rates while controlling portfolio risk.

Apply differentiated credit policies by customer segment, offering higher limits and preferential terms to low-risk segments while enforcing stricter limits and monitoring for high-risk and over-leveraged segments.

Do not rely on income alone for credit decisions, as high-income customers can still exhibit elevated risk due to high debt exposure; debt-to-income behavior should be prioritized.

Use moderate risk thresholds (≈0.20–0.30) to balance business growth and risk control, as overly conservative thresholds significantly reduce approvals while overly lenient thresholds fail to capture meaningful risk.

Implement manual review workflows for medium-risk customers, allowing human judgment to reduce false rejections without materially increasing portfolio risk.

Focus monitoring efforts on a small subset of customers, since risk is highly concentrated and a limited group drives a disproportionate share of potential losses.

Leverage customer segmentation for targeted strategies, such as credit-building programs for emerging customers and enhanced monitoring for high-income, high-debt profiles.

Treat transaction-level classification as a supporting signal, not a standalone risk driver, as transaction value alone shows limited differentiation without customer context.

Use explainable, probability-based risk models as baselines, enabling transparency, regulatory alignment, and easier policy tuning as the business scales.

Conclusion

This project demonstrates how an integrated analytics approach can transform raw customer and transaction data into actionable business decisions. By combining exploratory data analysis, customer segmentation, credit scoring, transaction classification, and probability-based risk modelling, the analysis provides a realistic and explainable view of portfolio risk rather than relying on isolated metrics or high-accuracy claims.

The results show that credit risk is not uniformly distributed across the customer base—risk is concentrated within specific segments and driven more by debt behavior than income alone. The risk model effectively separates customers into actionable risk bands, enabling balanced approval strategies that support business growth while maintaining control over potential losses. Customer segmentation further strengthens decision-making by allowing differentiated credit policies instead of one-size-fits-all rules.

Overall, the project highlights the value of explainable, business-aligned analytics in financial decision-making. Rather than optimizing for perfect predictions, the framework prioritizes transparency, risk control, and practical usability, making it well-suited as a baseline system for real-world credit and risk management applications.

In [ ]:

In [ ]:

In [ ]: