In [12]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
```

In [13]:
```
df = pd.read_csv(r"C:\Users\KUMALIKA\OneDrive\Desktop\Woksen_Academics\Term_1\Managerial Communication\loan_data.csv",encoding="ISO-8859-1")
```

In [14]:
```
df
```

Out[14]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.paid
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	0	0	0	0
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	0	0	0	0
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6	1	0	0	0
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2	1	0	0	0
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9573	0	all_other	0.1461	344.76	12.180755	10.39	672	10474.000000	215372	82.1	2	0	0	1
9574	0	all_other	0.1253	257.70	11.141862	0.21	722	4380.000000	184	1.1	5	0	0	1
9575	0	debt_consolidation	0.1071	97.81	10.596635	13.09	687	3450.041667	10036	82.9	8	0	0	1
9576	0	home_improvement	0.1600	351.58	10.819778	19.18	692	1800.000000	0	3.2	5	0	0	1
9577	0	debt_consolidation	0.1392	853.43	11.264464	16.28	732	4740.000000	37879	57.0	6	0	0	1

9578 rows × 14 columns

In [15]:
```
le = LabelEncoder()
df['purpose'] = le.fit_transform(df['purpose'])
```

In [16]:
```
X = df.drop('not.fully.paid', axis=1)
y = df['not.fully.paid']
```

In [17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [18]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [20]:
```
log_reg = LogisticRegression()
log_reg.fit(X_train_scaled, y_train)
y_pred_log = log_reg.predict(X_test_scaled)
```

In [21]:
```
y_pred = log_reg.predict(X_test_scaled)
print("=== Model Evaluation ===")
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
=== Model Evaluation ===
Accuracy: 0.8392484342379958
              precision    recall  f1-score   support

           0       0.84      1.00      0.91      1611
           1       0.38      0.02      0.03       305

    accuracy                           0.84      1916
   macro avg       0.61      0.51      0.47      1916
weighted avg       0.77      0.84      0.77      1916
```

In [22]:
```
print("\n=== User Loan Application ===")
user_input = {
    'credit.policy': 1,
    'purpose': 'credit_card',
    'int.rate': 0.12,
    'installment': 200.0,
    'log.annual.inc': 10.5,
    'dti': 15.0,
    'fico': 700,
    'days.with.cr.line': 4000,
    'revol.bal': 5000,
    'revol.util': 40.0,
    'inq.last.6mths': 1,
    'delinq.2yrs': 0,
    'pub.rec': 0
}
```

```
=== User Loan Application ===
```

In [23]:
```
user_input['purpose'] = le.transform([user_input['purpose']])[0]
```

In [24]:
```
user_df = pd.DataFrame([user_input])
user_scaled = scaler.transform(user_df)
```

In [25]:
```
prediction = log_reg.predict(user_scaled)[0]
prediction_proba = log_reg.predict_proba(user_scaled)[0][1]
```

In [26]:
```
print("\n=== Prediction Result ===")
if prediction == 1:
    print(" The applicant is likely NOT to fully pay the loan.")
else:
    print(" The applicant is likely to fully pay the loan.")
print(f"Probability of NOT fully paying: {prediction_proba:.2f}")
```

```
=== Prediction Result ===
 The applicant is likely to fully pay the loan.
Probability of NOT fully paying: 0.13
```

In [27]:
```
dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)
y_pred_tree = dtree.predict(X_test)
```

In [28]:
```
print("=== Logistic Regression ===")
print("Accuracy:", accuracy_score(y_test, y_pred_log))
print(classification_report(y_test, y_pred_log))

print("\n=== Decision Tree ===")
print("Accuracy:", accuracy_score(y_test, y_pred_tree))
print(classification_report(y_test, y_pred_tree))
```

```
=== Logistic Regression ===
Accuracy: 0.8392484342379958
              precision    recall  f1-score   support

           0       0.84      1.00      0.91      1611
           1       0.38      0.02      0.03       305

    accuracy                           0.84      1916
   macro avg       0.61      0.51      0.47      1916
weighted avg       0.77      0.84      0.77      1916


=== Decision Tree ===
Accuracy: 0.7494780793319415
              precision    recall  f1-score   support

           0       0.86      0.84      0.85      1611
           1       0.24      0.26      0.25       305

    accuracy                           0.75      1916
   macro avg       0.55      0.55      0.55      1916
weighted avg       0.76      0.75      0.75      1916
```

In [29]:
```
import pandas as pd
import numpy as np
import gradio as gr
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
```

In [21]:
```
df = pd.read_csv("loan_data.csv", encoding="ISO-8859-1")
```

In [22]:
```
df
```

Out[22]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.paid
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	0	0	0	0
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	0	0	0	0
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6	1	0	0	0
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2	1	0	0	0
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9573	0	all_other	0.1461	344.76	12.180755	10.39	672	10474.000000	215372	82.1	2	0	0	1
9574	0	all_other	0.1253	257.70	11.141862	0.21	722	4380.000000	184	1.1	5	0	0	1
9575	0	debt_consolidation	0.1071	97.81	10.596635	13.09	687	3450.041667	10036	82.9	8	0	0	1
9576	0	home_improvement	0.1600	351.58	10.819778	19.18	692	1800.000000	0	3.2	5	0	0	1
9577	0	debt_consolidation	0.1392	853.43	11.264464	16.28	732	4740.000000	37879	57.0	6	0	0	1

9578 rows × 14 columns

In [23]:
```
le = LabelEncoder()
df["purpose"] = le.fit_transform(df["purpose"])
```

In [24]:
```
X = df.drop("not.fully.paid", axis=1)
y = df["not.fully.paid"]
```

In [25]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [26]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

In [27]:
```
log_reg = LogisticRegression()
log_reg.fit(X_train_scaled, y_train)
```

Out[27]:

```
LogisticRegression()
```

In [28]:
```
def predict_loan(credit_policy, purpose, int_rate, installment, log_annual_inc, dti, fico,
                 days_with_cr_line, revol_bal, revol_util, inq_last_6mths, delinq_2yrs, pub_rec):

    user_input = {
        'credit.policy': 1 if credit_policy == "Yes" else 0,
        'purpose': le.transform([purpose])[0],
        'int.rate': float(int_rate),
        'installment': float(installment),
        'log.annual.inc': float(log_annual_inc),
        'dti': float(dti),
        'fico': int(fico),
        'days.with.cr.line': int(days_with_cr_line),
        'revol.bal': int(revol_bal),
        'revol.util': float(revol_util),
        'inq.last.6mths': int(inq_last_6mths),
        'delinq.2yrs': int(delinq_2yrs),
        'pub.rec': int(pub_rec)
    }

    df_input = pd.DataFrame([user_input])
    scaled = scaler.transform(df_input)
    pred = log_reg.predict(scaled)[0]
    prob = log_reg.predict_proba(scaled)[0][1]

    if pred == 1:
        result = " The applicant is likely NOT to fully pay the loan."
    else:
        result = " The applicant is likely to fully pay the loan."

    return f"{result}\n Probability of NOT fully paying: {prob:.2f}"
```

In [29]:
```
interface = gr.Interface(
    fn=predict_loan,
    inputs=[
        gr.Radio(["Yes", "No"], label="Credit Policy"),
        gr.Radio(list(le.classes_), label="Loan Purpose"),
        gr.Radio(["0.10", "0.12", "0.14", "0.16", "0.18"], label="Interest Rate"),
        gr.Radio(["100", "200", "300", "400", "500"], label="Installment"),
        gr.Radio(["9.0", "10.0", "10.5", "11.0", "12.0"], label="Log Annual Income"),
        gr.Radio(["10", "15", "20", "25", "30"], label="DTI"),
        gr.Radio(["650", "680", "700", "720", "750"], label="FICO"),
        gr.Radio(["2000", "3000", "4000", "5000", "6000"], label="Days with Credit Line"),
        gr.Radio(["1000", "3000", "5000", "7000", "9000"], label="Revolving Balance"),
        gr.Radio(["20", "40", "60", "80", "100"], label="Revol Util %"),
        gr.Radio(["0", "1", "2", "3", "4"], label="Inquiries Last 6 Months"),
        gr.Radio(["0", "1", "2"], label="Delinquencies Last 2 Years"),
        gr.Radio(["0", "1", "2"], label="Public Records")
    ],
    outputs="text",
    title="Loan Repayment Prediction",
    description="Will the applicant fully repay the loan? Select options and click Predict."
)
```

In [30]:
```
interface.launch()
```

```
* Running on local URL:  http://127.0.0.1:7862
* To create a public link, set `share=True` in `launch()`.
```

Out[30]:

In [ ]: