In [24]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [19]:
```
train_df = pd.read_csv('Training Data.csv')
test_df = pd.read_csv('Test Data.csv')
```

In [20]:
```
train_df.head(3)
```

Out[20]:

	Id	Income	Age	Experience	Married/Single	House_Ownership	Car_Ownership	Profes
0	1	1303834	23	3	single	rented	no	Mechanical_engi
1	2	7574516	40	10	single	rented	no	Software_Devel
2	3	3991815	66	4	married	rented	no	Technical_w

In [53]:
```
# Null Check Code:

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Efficient Null Check Summary
null_summary = train_df.isnull().sum()
null_summary = null_summary[null_summary > 0].sort_values(ascending=False)

if not null_summary.empty:
    # Display Table
    print("🔎 Null Value Summary:\n")
    display(null_summary.to_frame(name='Missing Values'))

    # Optional % of Missing Values
    null_percent = (train_df.isnull().mean() * 100).round(2)
    null_percent = null_percent[null_percent > 0].sort_values(ascending=False)

    # Visual Null Heatmap
    plt.figure(figsize=(12, 0.4 * len(null_summary)))
    sns.barplot(x=null_percent.values, y=null_percent.index, palette='pastel')
    plt.xlabel('Percentage of Missing Values')
    plt.title('Missing Data Summary')
    plt.xlim(0, 100)
    plt.grid(axis='x', linestyle='--', alpha=0.3)
    plt.tight_layout()
    plt.show()

else:
    print("✅ No missing values found in training data.")
```

✅ No missing values found in training data.

In [21]:
```
## EDA: Exploratory Data Analysis:
```

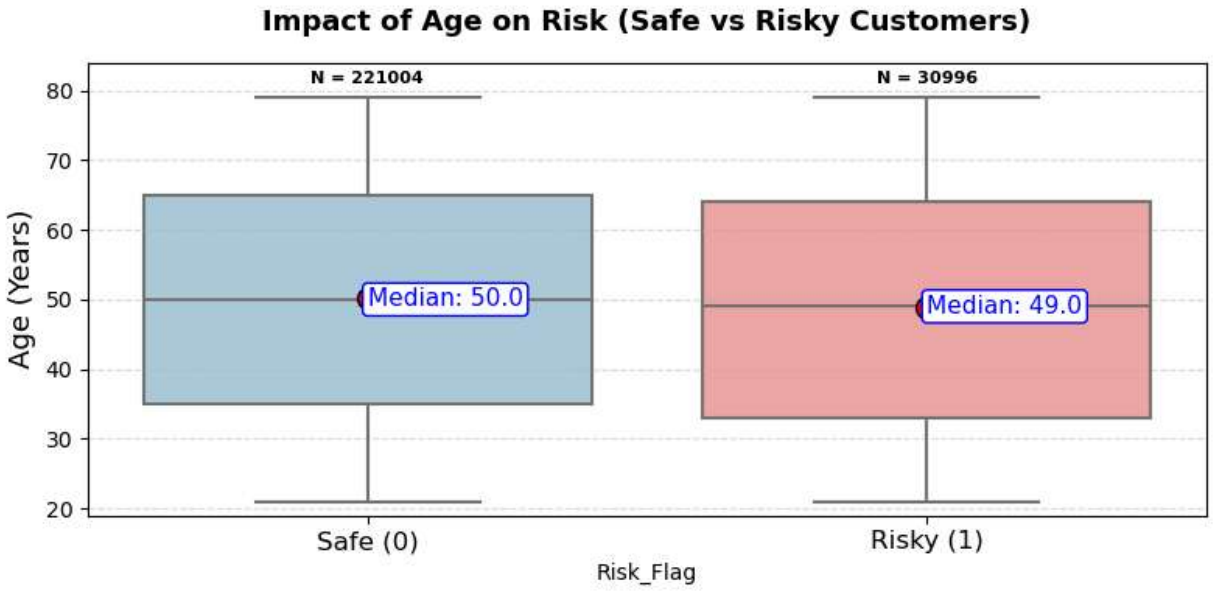
In [22]:
```
print(train_df.columns)
```

```
Index(['Id', 'Income', 'Age', 'Experience', 'Married/Single',
       'House_Ownership', 'Car_Ownership', 'Profession', 'CITY', 'STATE',
       'CURRENT_JOB_YRS', 'CURRENT_HOUSE_YRS', 'Risk_Flag'],
      dtype='object')
```

In [62]:
```
train_df['CITY'].nunique()
```

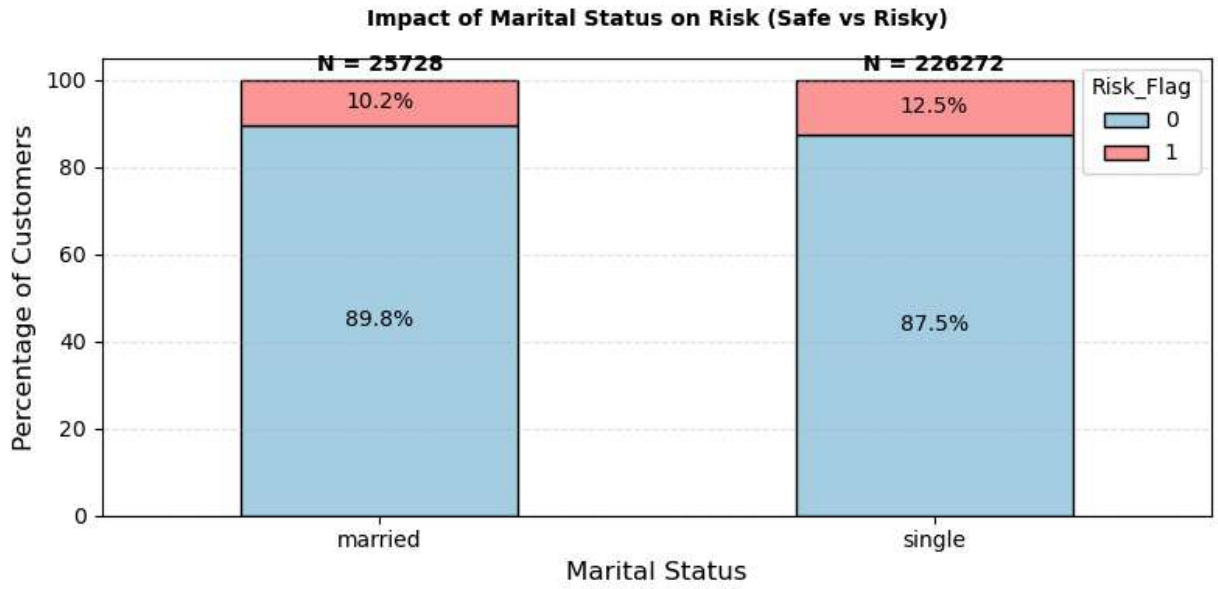
Out[62]:
```
317
```

In [37]:
```
#1. Impact of Age on Risk:

# Age vs Risk_Flag
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

plt.figure(figsize=(8, 4))

# Boxplot (cleaner, no swarm clutter)
sns.boxplot(x='Risk_Flag', y='Age', data=train_df, palette=['#a6cee3', '#fb9a99']
            meanprops={"marker":"o", "markerfacecolor":"red", "markeredgecolor":'

# Custom x-tick labels
plt.xticks([0, 1], ['Safe (0)', 'Risky (1)'], fontsize=12)

# Title & Sub-Title
plt.title('Impact of Age on Risk (Safe vs Risky Customers)', fontsize=13, weight=

# Counts above boxes (outside plot area)
counts = train_df['Risk_Flag'].value_counts().sort_index()
y_pos = train_df['Age'].max() + 2
for i, count in enumerate(counts):
    plt.text(i, y_pos, f'N = {count}', ha='center', fontsize=8, weight='bold', co

# Median values beside boxes
medians = train_df.groupby('Risk_Flag')['Age'].median()
for i, median in enumerate(medians):
    plt.text(i, median, f'Median: {median:.1f}', ha='left', fontsize=11, color='b

# Y-axis label
plt.ylabel('Age (Years)', fontsize=13)

# Grid & Aesthetics
plt.grid(axis='y', linestyle='--', alpha=0.4)
plt.ylim(train_df['Age'].min() - 2, train_df['Age'].max() + 5)

plt.tight_layout()
plt.show()
```



In [46]:
```
train_df['Married/Single'] = train_df['Married/Single'].astype('str')
train_df['Risk_Flag'] = train_df['Risk_Flag'].astype('str')
```

In [52]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

cat_col = 'Married/Single'
target_col = 'Risk_Flag'

# Ensure string type & fill NaNs
train_df[cat_col] = train_df[cat_col].astype(str).fillna('Unknown')

# Compute counts
grouped_counts = train_df.groupby([cat_col, target_col]).size().reset_index(name=

# Pivot table for % calculation
pivot_table = grouped_counts.pivot(index=cat_col, columns=target_col, values='cou

# Percentages table
pivot_table_perc = pivot_table.div(pivot_table.sum(axis=1), axis=0) * 100

# Plot as stacked bar
fig, ax = plt.subplots(figsize=(8, 4))
pivot_table_perc.plot(kind='bar', stacked=True, color=['#a6cee3', '#fb9a99'], edg

# Annotate percentages inside bars (x pos is index-based, not category names)
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    if height > 0:
        ax.text(x + width/2, y + height/2, f'{height:.1f}%', ha='center', va='cen

# Add counts (N=) above bars
total_counts = pivot_table.sum(axis=1)
for i, total in enumerate(total_counts):
    ax.text(i, 102, f'N = {int(total)}', ha='center', fontsize=10, weight='bold')

# Titles & labels
ax.set_title('Impact of Marital Status on Risk (Safe vs Risky)', fontsize=10, wei
ax.set_ylabel('Percentage of Customers', fontsize=12)
ax.set_xlabel('Marital Status', fontsize=12)
ax.set_ylim(0, 105)
ax.set_xticklabels(ax.get_xticklabels(), rotation=0)

# Grid & style
ax.grid(axis='y', linestyle='--', alpha=0.3)
plt.tight_layout()
plt.show()
```

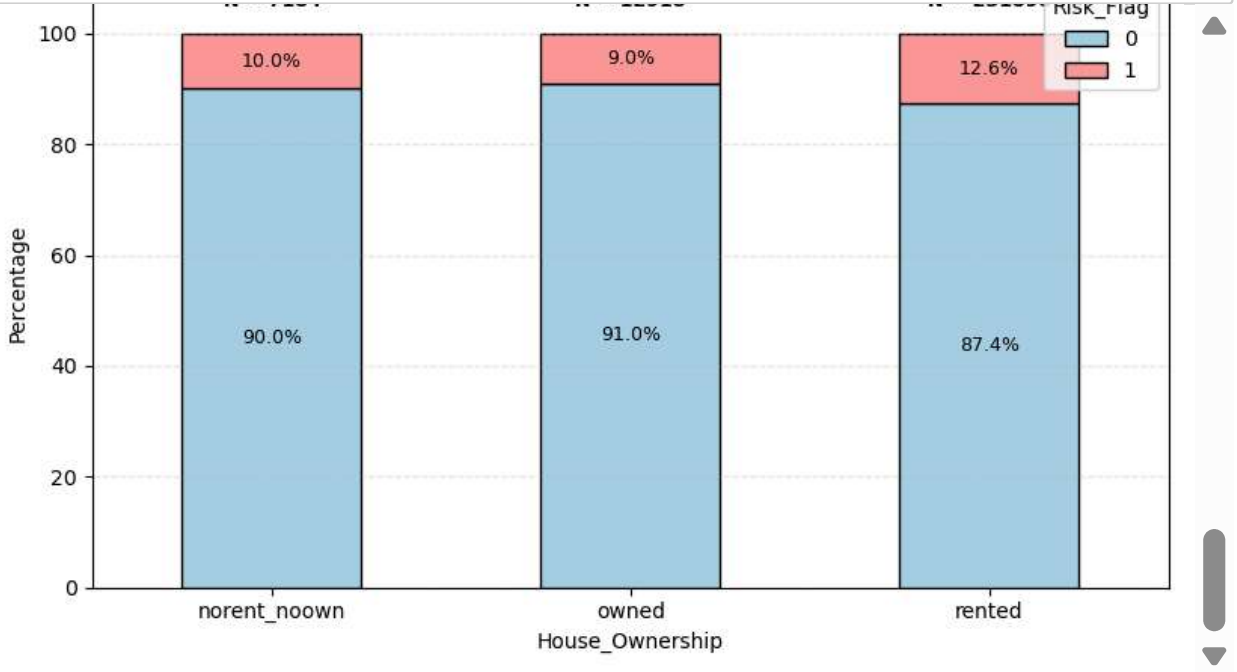


In [54]:
```
# function that predicts eda vs risk  mentioning only columns
```

In [60]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

def eda_vs_risk_flag(df, columns, target_col='Risk_Flag'):
    """
    EDA visualization function for columns (auto-detects categorical/continuous)
    """
    if target_col not in df.columns:
        print(f"❗ Target column '{target_col}' not found in DataFrame. Exiting.
        return

    print(f"📊 Data Shape: {df.shape}")
    print(f"📊 Columns Available: {list(df.columns)}")
    print(f"📊 Risk_Flag Distribution:\n{df[target_col].value_counts(dropna=Fals

    for col in columns:
        if col not in df.columns:
            print(f"⚠️ Column '{col}' not found. Skipping...")
            continue

        print(f"\n🔹 Analyzing '{col}' vs '{target_col}'")

        if pd.api.types.is_numeric_dtype(df[col]):
            print(f"  ➔ '{col}' detected as Continuous")
            plt.figure(figsize=(8, 5))
            sns.boxplot(x=target_col, y=col, data=df, palette='pastel', showmeans
                        meanprops={"marker": "o", "markerfacecolor": "red", "mark
            plt.title(f'{col} vs {target_col}', fontsize=14)
            plt.xlabel(target_col)
            plt.ylabel(col)
            plt.grid(axis='y', linestyle='--', alpha=0.3)
            plt.tight_layout()
            plt.show()

        else:
            print(f"  ➔ '{col}' detected as Categorical")
            df[col] = df[col].astype(str).fillna('Unknown')
            counts = df.groupby([col, target_col]).size().reset_index(name='count

            if counts.empty:
                print(f"⚠️ No data to plot for '{col}'. Skipping...")
                continue

            pivot = counts.pivot(index=col, columns=target_col, values='count').f
            pivot_perc = pivot.div(pivot.sum(axis=1), axis=0) * 100

            ax = pivot_perc.plot(kind='bar', stacked=True, figsize=(5, 2), color=
            for c in ax.containers:
                ax.bar_label(c, fmt='%.1f%%', label_type='center', fontsize=7)

            totals = pivot.sum(axis=1)
            for i, total in enumerate(totals):
                ax.text(i, 105, f'N = {int(total)}', ha='center', fontsize=7, wei

            plt.title(f'{col} vs {target_col}', fontsize=10)
            plt.ylabel('Percentage')
```

```
    plt.xlabel(col)
    plt.ylim(0, 110)
    plt.xticks(rotation=0)
    plt.grid(axis='y', linestyle='--', alpha=0.3)
    plt.tight_layout()
    plt.show()
```

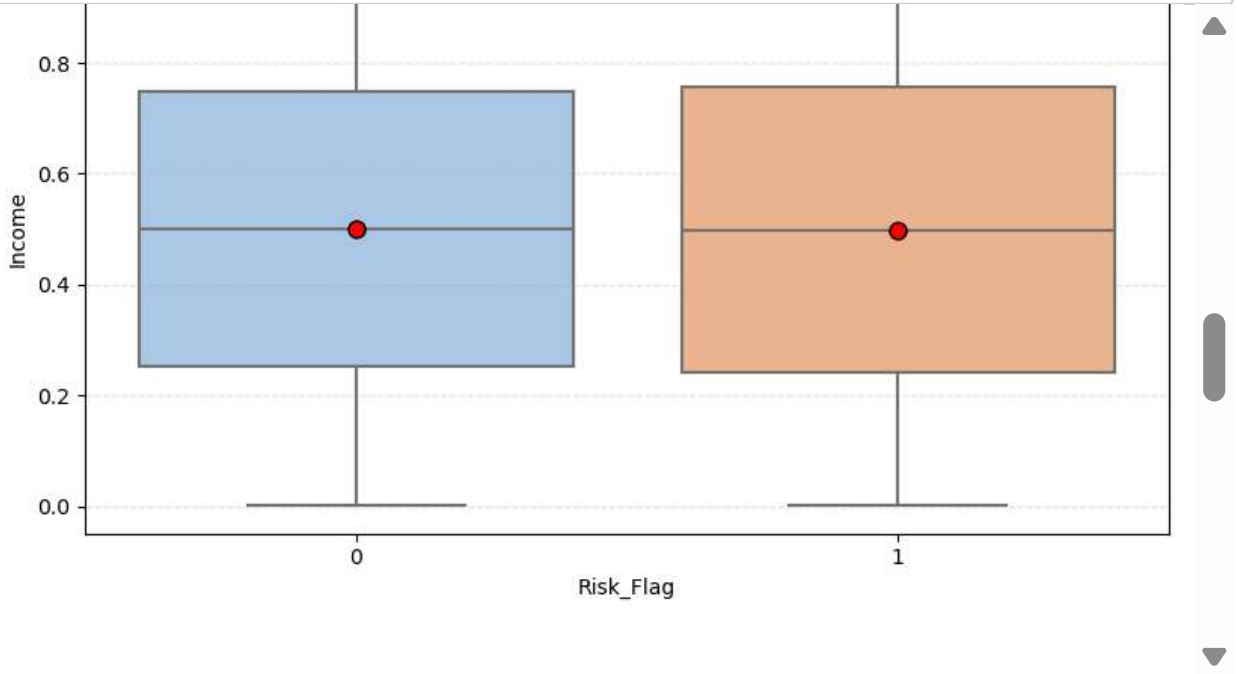
In [57]:
```
eda_vs_risk_flag(train_df, columns=['Age', 'Income', 'Married/Single', 'House_Own
```



In [89]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

def eda_vs_risk_flag(df, columns, target_col='Risk_Flag', top_n=10):
    """
    EDA visualization for columns (auto-detects continuous/categorical) vs Risk_F
    For high-cardinality categorical vars, limits to top_n values.

    Args:
    - df: DataFrame
    - columns: list of columns to analyze vs Risk_Flag
    - target_col: binary target column (default='Risk_Flag')
    - top_n: limit top N categories (only for categorical variables)
    """
    if target_col not in df.columns:
        print(f"❗ Target column '{target_col}' not found in DataFrame. Exiting.
        return

    print(f" Data Shape: {df.shape}")
    print(f" Columns Available: {list(df.columns)}")
    print(f" {target_col} Distribution:\n{df[target_col].value_counts(dropna=Fals

    for col in columns:
        if col not in df.columns:
            print(f"⚠️ Column '{col}' not found. Skipping...")
            continue

        print(f"\n🔹 Analyzing '{col}' vs '{target_col}'")

        if pd.api.types.is_numeric_dtype(df[col]):
            print(f"  ➔ '{col}' detected as Continuous")
            plt.figure(figsize=(8, 5))
            sns.boxplot(x=target_col, y=col, data=df, palette='pastel', showmeans
                        meanprops={"marker": "o", "markerfacecolor": "red", "mark
            plt.title(f'{col} vs {target_col}', fontsize=14)
            plt.xlabel(target_col)
            plt.ylabel(col)
            plt.grid(axis='y', linestyle='--', alpha=0.3)
            plt.tight_layout()
            plt.show()

        else:
            print(f"  ➔ '{col}' detected as Categorical")

            # Reduce to top_n categories if many unique values
            top_values = df[col].value_counts().head(top_n).index
            df[col] = df[col].apply(lambda x: x if x in top_values else 'Other').

            counts = df.groupby([col, target_col]).size().reset_index(name='count

            if counts.empty:
                print(f" No data to plot for '{col}'. Skipping...")
                continue

            pivot = counts.pivot(index=col, columns=target_col, values='count').f
            pivot_perc = pivot.div(pivot.sum(axis=1), axis=0) * 100
```

```
        ax = pivot_perc.plot(kind='bar', stacked=True, figsize=(6, 2), color=
        for c in ax.containers:
            ax.bar_label(c, fmt='%.1f%%', label_type='center', fontsize=8)

        totals = pivot.sum(axis=1)
        for i, total in enumerate(totals):
            ax.text(i, 105, f'N = {int(total)}', ha='center', fontsize=6, wei

        plt.title(f'{col} (Top {top_n}) vs {target_col}', fontsize=12)
        plt.ylabel('Percentage')
        plt.xlabel(col)
        plt.ylim(0, 110)
        plt.xticks(rotation=0)
        plt.grid(axis='y', linestyle='--', alpha=0.3)
        plt.tight_layout()
        plt.show()
```

In [82]:
```
eda_vs_risk_flag(train_df, columns=['Age', 'Income', 'Car_Ownership', 'CITY'], to
```



In [92]:
```
def suggest_best_test(df, feature_col, target_col='Risk_Flag'):
    """
    Suggests the optimal statistical test for Risk_Flag vs feature_col.

    Args:
        df: DataFrame
        feature_col: Feature column to analyze vs Risk_Flag
        target_col: Target column (binary, default 'Risk_Flag')

    Returns:
        Suggested statistical test (string)
    """

    if feature_col not in df.columns:
        print(f" Feature column '{feature_col}' not found.")
        return None

    if target_col not in df.columns:
        print(f" Target column '{target_col}' not found.")
        return None

    feature_unique = df[feature_col].nunique(dropna=False)
    target_unique = df[target_col].nunique(dropna=False)

    print(f" Feature '{feature_col}' unique values: {feature_unique}")
    print(f" Target '{target_col}' unique values: {target_unique}")

    # Check if Risk_Flag is binary categorical
    if target_unique != 2:
        print(" Target column should be binary (0/1). Exiting.")
        return None

    # If feature is numeric
    if pd.api.types.is_numeric_dtype(df[feature_col]):
        print(f" '{feature_col}' detected as Continuous Variable.")

        if feature_unique <= 2:
            suggestion = "T-Test (2 groups mean comparison)"
        elif feature_unique > 2 and feature_unique <= 10:
            suggestion = "ANOVA (3+ groups mean comparison)"
        else:
            suggestion = "T-Test (binary target) or Consider dimensionality reduc

    # If feature is categorical
    else:
        print(f"➔ '{feature_col}' detected as Categorical Variable.")

        if feature_unique <= 2:
            suggestion = "Chi-Square Test of Independence (2x2 table)"
        elif feature_unique > 2 and feature_unique <= 10:
            suggestion = "Chi-Square Test of Independence (Rx2 table)"
        else:
            suggestion = "Chi-Square Test (consider grouping high-cardinality cat

    print(f"✅ Suggested Statistical Test: {suggestion}")
    return suggestion
```

In [93]:
```
# Example with CITY
suggest_best_test(df=train_df, feature_col='CITY')

# Example with Age
suggest_best_test(df=train_df, feature_col='Age')
```

```
📊 Feature 'CITY' unique values: 5
📊 Target 'Risk_Flag' unique values: 2
➔ 'CITY' detected as Categorical Variable.
✅ Suggested Statistical Test: Chi-Square Test of Independence (Rx2 table)
📊 Feature 'Age' unique values: 59
📊 Target 'Risk_Flag' unique values: 2
➔ 'Age' detected as Continuous Variable.
✅ Suggested Statistical Test: T-Test (binary target) or Consider dimensionalit
y reduction
```

Out[93]:
```
'T-Test (binary target) or Consider dimensionality reduction'
```

In [97]:
```
# Example with CITY
suggest_best_test(df=train_df, feature_col='Risk_Flag')

# Example with Age
suggest_best_test(df=train_df, feature_col='Age')
```

```
📊 Feature 'Risk_Flag' unique values: 2
📊 Target 'Risk_Flag' unique values: 2
➔ 'Risk_Flag' detected as Categorical Variable.
✅ Suggested Statistical Test: Chi-Square Test of Independence (2x2 table)
📊 Feature 'Age' unique values: 59
📊 Target 'Risk_Flag' unique values: 2
➔ 'Age' detected as Continuous Variable.
✅ Suggested Statistical Test: T-Test (binary target) or Consider dimensionalit
y reduction
```

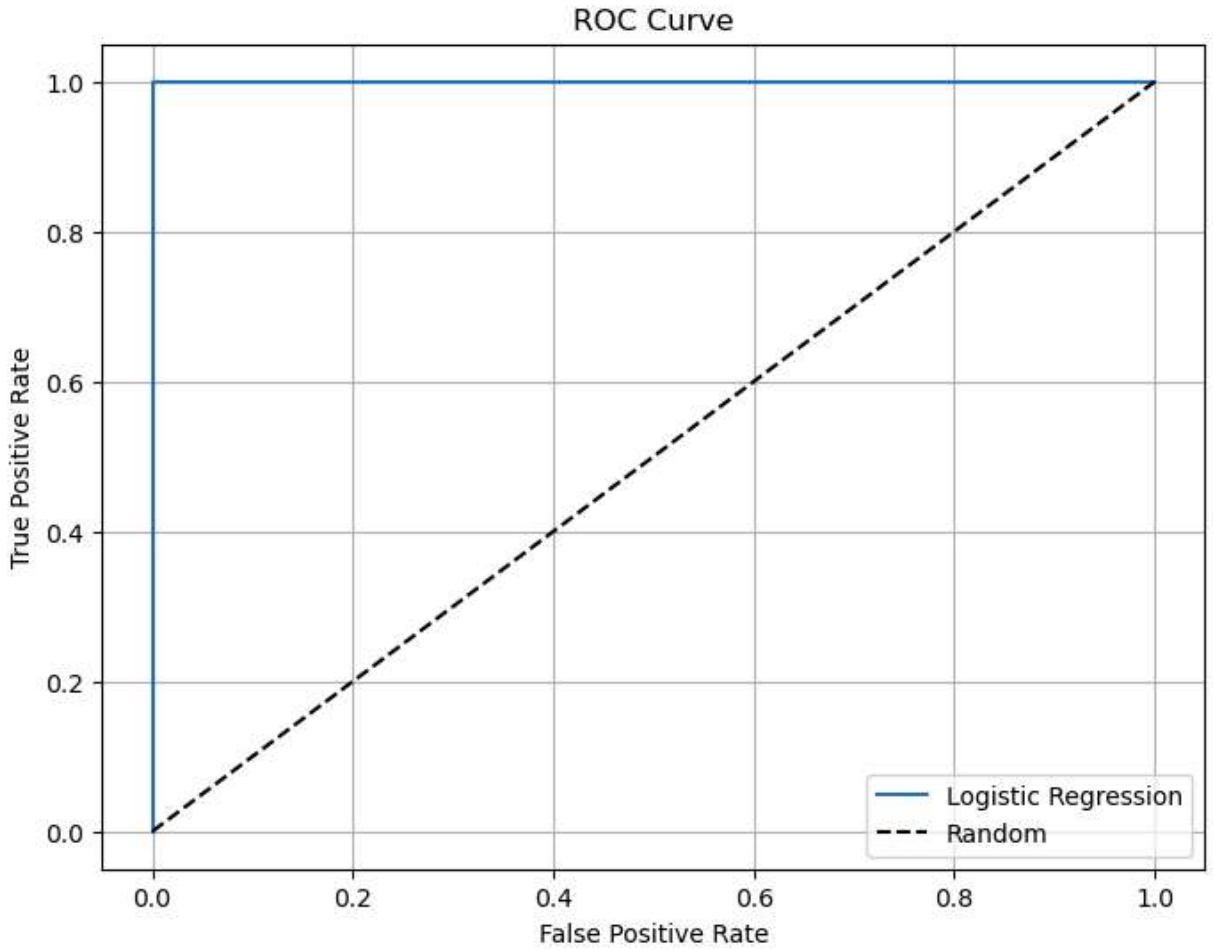
Out[97]:
```
'T-Test (binary target) or Consider dimensionality reduction'
```

In [ ]:
```
# logistic regression of risk model
```

In [4]:
```
# Import required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# For model building
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_sco

# Load dataset
df = pd.read_csv('Training Data.csv')  # Replace with actual filename

# --- Add a synthetic 'risk' column ---
# Define 'high risk' as: Income < 2,000,000 AND Car_Ownership == 'no'
df['risk'] = np.where((df['Income'] < 2000000) & (df['Car_Ownership'] == 'no'), 1

# --- Drop non-useful columns ---
# --- Drop non-useful columns (safely) ---
columns_to_drop = ['ID', 'CITY', 'STATE']
df = df.drop(columns=[col for col in columns_to_drop if col in df.columns])


# --- One-hot encode categorical variables ---
df = pd.get_dummies(df, drop_first=True)

# --- Separate features and target ---
X = df.drop('risk', axis=1)
y = df['risk']

# --- Train-test split ---
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s

# --- Feature scaling ---
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# --- Model Building ---
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train_scaled, y_train)

# --- Evaluation ---
y_pred = logreg.predict(X_test_scaled)
y_prob = logreg.predict_proba(X_test_scaled)[:, 1]

print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("ROC AUC Score:", roc_auc_score(y_test, y_prob))

# --- ROC Curve ---
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label='Logistic Regression')
```

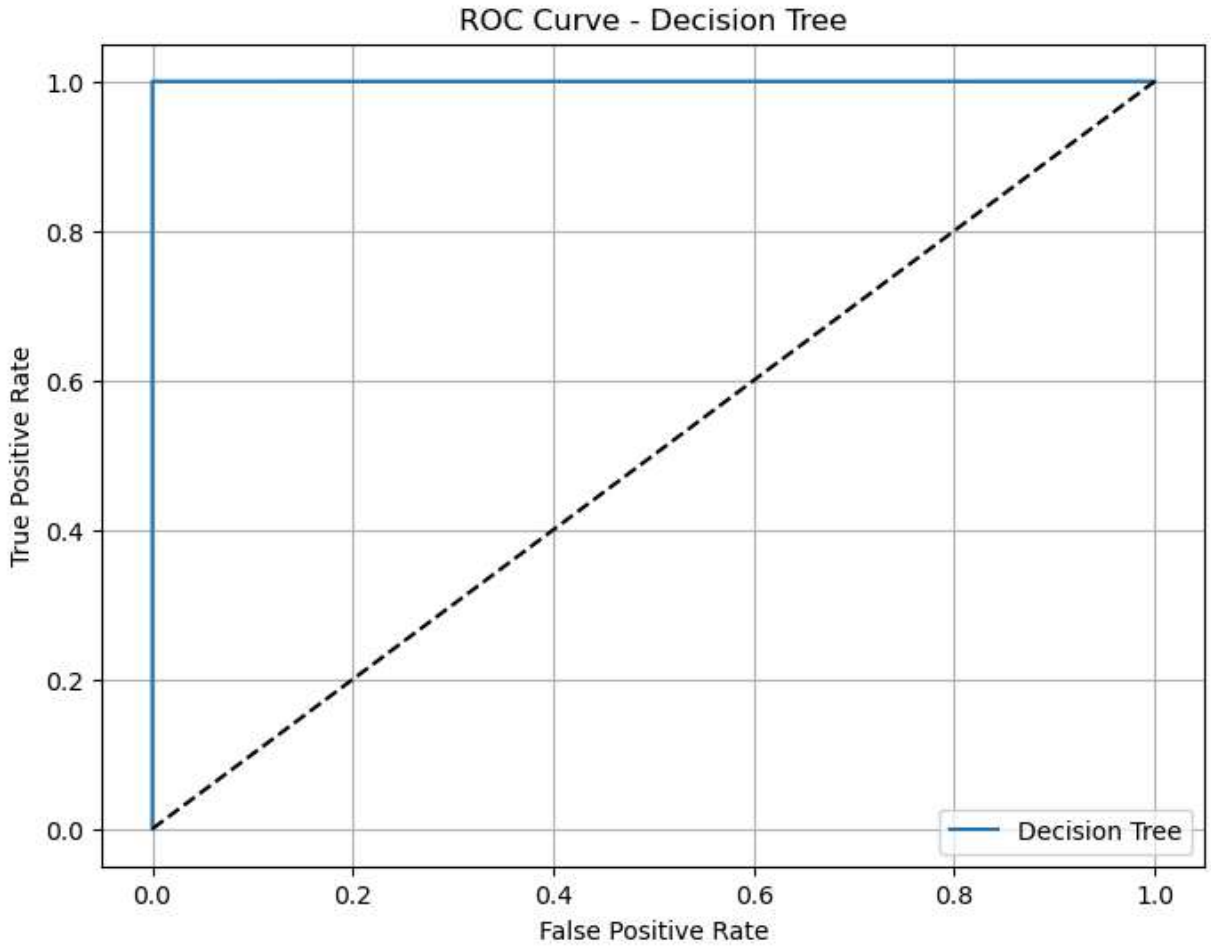
```
plt.plot([0, 1], [0, 1], 'k--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid(True)
plt.show()
```

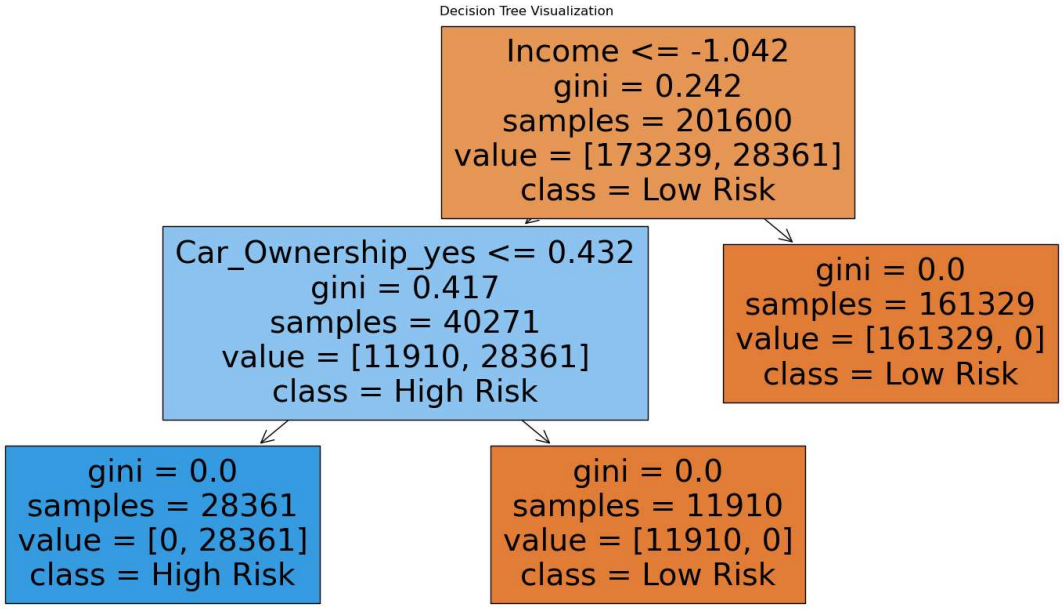
```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     43371
           1       1.00      1.00      1.00      7029

    accuracy                           1.00     50400
   macro avg       1.00      1.00      1.00     50400
weighted avg       1.00      1.00      1.00     50400

Confusion Matrix:
 [[43338    33]
 [   16  7013]]
Accuracy Score: 0.9990277777777777
ROC AUC Score: 0.9999934854223482
```



In [7]:
```
# Import Decision Tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree

#  Model Initialization & Training
dtree = DecisionTreeClassifier(max_depth=6,random_state=42)  # You can tune max_d
dtree.fit(X_train_scaled, y_train)

#  Predictions
y_pred = dtree.predict(X_test_scaled)
y_prob = dtree.predict_proba(X_test_scaled)[:, 1]

# Evaluation
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("ROC AUC Score:", roc_auc_score(y_test, y_prob))

#  Plot ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label='Decision Tree')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Decision Tree')
plt.legend()
plt.grid(True)
plt.show()

#  Optional: Visualize the Tree
#  Optional: Visualize the Tree
plt.figure(figsize=(20,10))
plot_tree(dtree, feature_names=list(X.columns), class_names=['Low Risk', 'High Ri
plt.title("Decision Tree Visualization")
plt.show()
```

```
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     43371
           1       1.00      1.00      1.00      7029

    accuracy                           1.00     50400
   macro avg       1.00      1.00      1.00     50400
weighted avg       1.00      1.00      1.00     50400

Confusion Matrix:
 [[43371     0]
 [    1  7028]]
Accuracy Score: 0.9999801587301588
ROC AUC Score: 0.9999288661260493
```


ROC Curve - Decision Tree
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Decision Tree


Decision Tree Visualization
Income <= -1.042
gini = 0.242
samples = 201600
value = [173239, 28361]
class = Low Risk
Car_Ownership_yes <= 0.432
gini = 0.417
samples = 40271
value = [11910, 28361]
class = High Risk
gini = 0.0
samples = 161329
value = [161329, 0]
class = Low Risk
gini = 0.0
samples = 28361
value = [0, 28361]
class = High Risk
gini = 0.0
samples = 11910
value = [11910, 0]
class = Low Risk

In [ ]:
```
#pipeline
#hyperparameter tunning
#accuracy
#test data prediction and accuracy and matrix of it
```

In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score

# Load data
data = pd.read_csv('Training Data.csv')

# Separate features and target
X = data.drop(columns=['Risk_Flag'])
y = data['Risk_Flag']

# Identify numeric and categorical columns
numeric_features = X.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X.select_dtypes(include=['object']).columns

# Numeric pipeline
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

# Categorical pipeline
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Combine transformers with ColumnTransformer
preprocessor = ColumnTransformer(transformers=[
    ('num', numeric_transformer, numeric_features),
    ('cat', categorical_transformer, categorical_features)
])

# Full pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('clf', DecisionTreeClassifier(random_state=42))
])

# Parameter grid for GridSearchCV
param_grid = {
    'clf__max_depth': [5, 10, 15, None],
    'clf__min_samples_split': [2, 5, 10],
    'clf__min_samples_leaf': [1, 2, 4],
    'clf__criterion': ['gini', 'entropy'],
    'clf__max_features': [None, 'sqrt', 'log2']
}

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s

# GridSearchCV
```

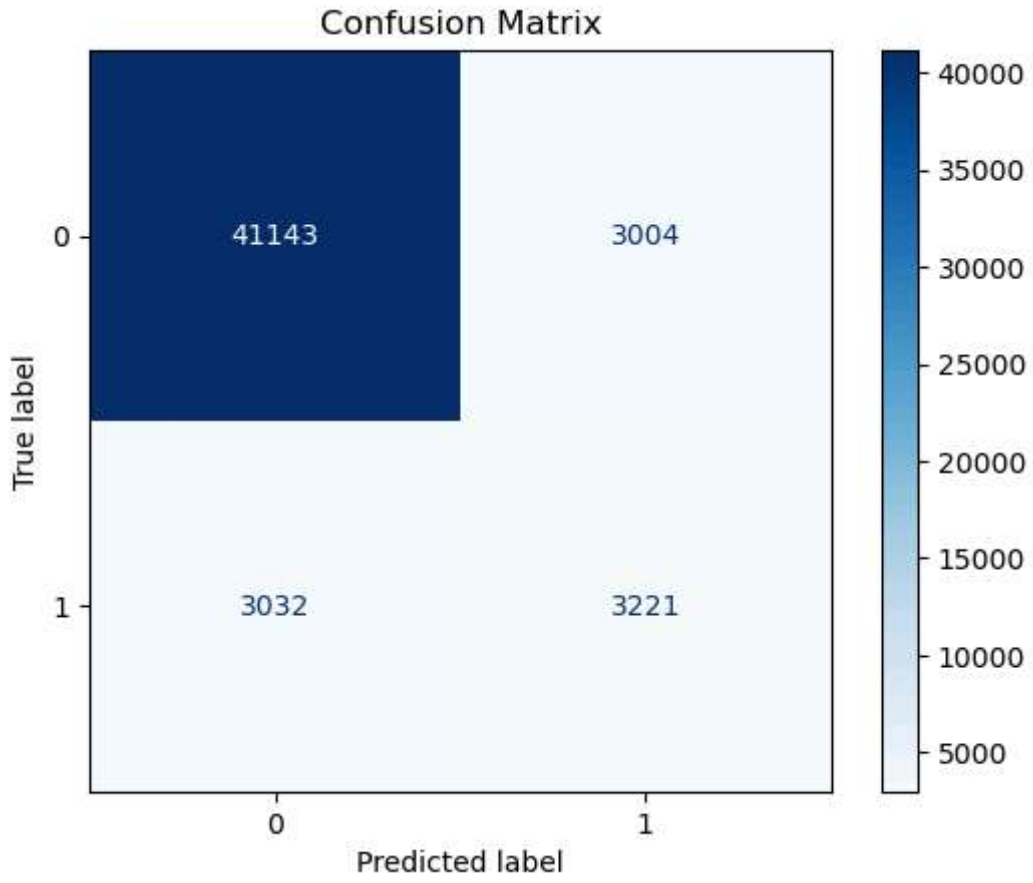
```
grid_search = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    cv=5,
    scoring='f1',
    n_jobs=-1,
    verbose=1
)

# Fit and evaluate
grid_search.fit(X_train, y_train)
y_pred = grid_search.predict(X_test)

print("Best Parameters:", grid_search.best_params_)
print("Test Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
Best Parameters: {'clf__criterion': 'entropy', 'clf__max_depth': None, 'clf__ma
x_features': 'log2', 'clf__min_samples_leaf': 1, 'clf__min_samples_split': 2}
Test Accuracy: 0.8802380952380953
Classification Report:
               precision    recall  f1-score   support

           0       0.93      0.93      0.93     44147
           1       0.52      0.52      0.52      6253

    accuracy                           0.88     50400
   macro avg       0.72      0.72      0.72     50400
weighted avg       0.88      0.88      0.88     50400
```

In [2]:
```
# Predict on the test data using the best found model
y_pred = grid_search.predict(X_test)  # or random_search.predict(X_test) if you u

# Calculate accuracy
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", accuracy)
```

```
Test Accuracy: 0.8802380952380953
```

In [3]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Assuming y_test and y_pred are already defined
# y_pred = grid_search.predict(X_test)

# Create confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Display confusion matrix with labels
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=grid_search.cla
disp.plot(cmap=plt.cm.Blues)

plt.title("Confusion Matrix")
plt.show()
```

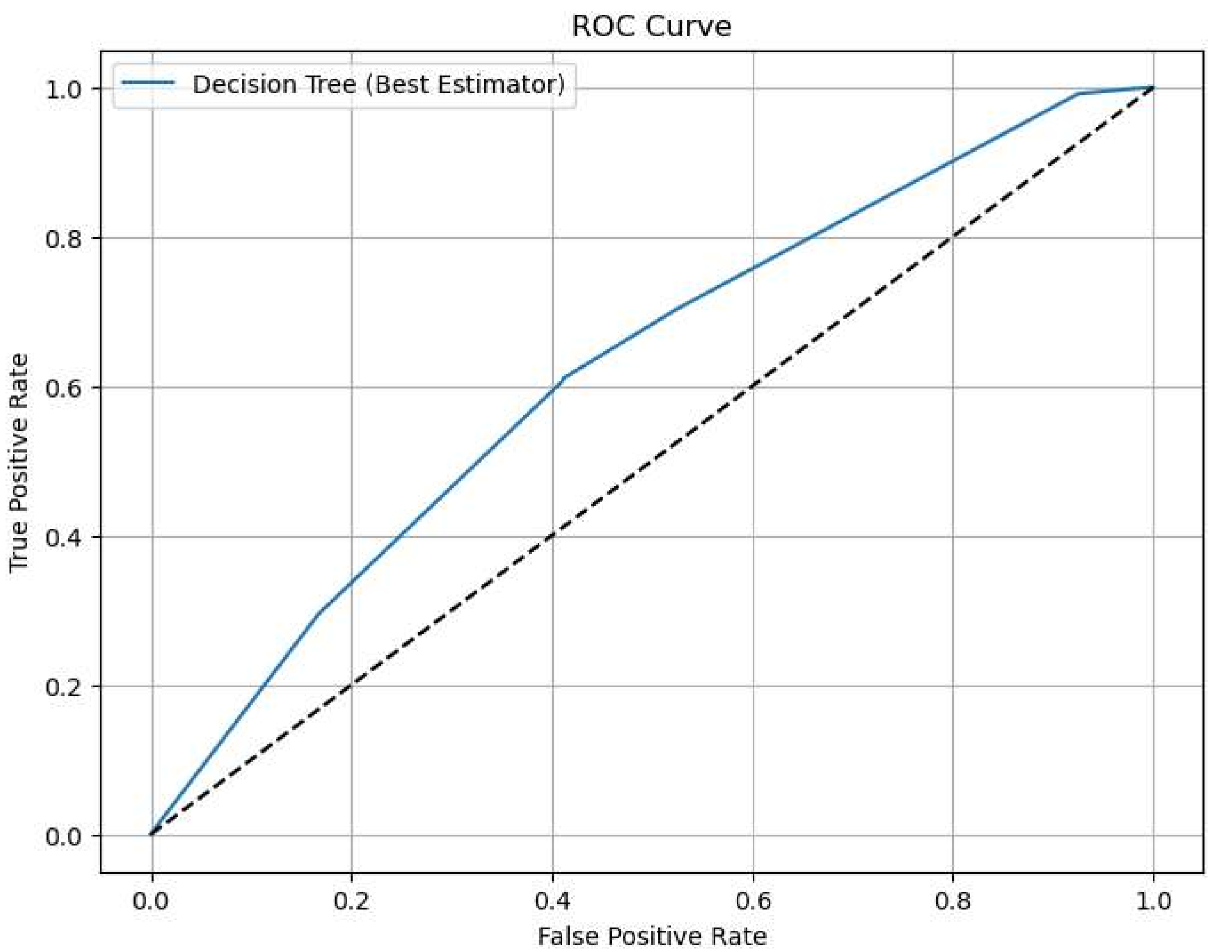


In [8]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matr

# Load dataset
data = pd.read_csv('Training Data.csv')

# Drop non-useful columns if they exist
columns_to_drop = ['ID', 'CITY', 'STATE']
data = data.drop(columns=[col for col in columns_to_drop if col in data.columns])

# Separate features and target
X = data.drop(columns='Risk_Flag')
y = data['Risk_Flag']

# Identify numeric and categorical columns
numeric_features = X.select_dtypes(include=['int64', 'float64']).columns.tolist()
categorical_features = X.select_dtypes(include=['object']).columns.tolist()

# Preprocessing for numeric features
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

# Preprocessing for categorical features
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Combine preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Create pipeline with Decision Tree classifier
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', DecisionTreeClassifier(random_state=42))
])

# Define parameter grid for GridSearchCV
param_grid = {
    'classifier__max_depth': [3, 5, 10, None],
    'classifier__min_samples_split': [2, 5, 10]
}
```

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s

# Grid Search
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy', n_jobs
grid_search.fit(X_train, y_train)

# Best parameters and score
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validated Accuracy:", grid_search.best_score_)

# Evaluate on test data
y_pred = grid_search.predict(X_test)
y_prob = grid_search.predict_proba(X_test)[:, 1]

print("Test Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("ROC AUC Score:", roc_auc_score(y_test, y_prob))

# Plot ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label='Decision Tree (Best Estimator)')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid(True)
plt.show()
```

```
Best Parameters: {'classifier__max_depth': 3, 'classifier__min_samples_split':
2}
Best Cross-Validated Accuracy: 0.8772420634920636
Test Accuracy: 0.8759325396825397
Confusion Matrix:
 [[44147     0]
 [ 6253     0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.88      1.00      0.93     44147
           1       0.00      0.00      0.00      6253

    accuracy                           0.88     50400
   macro avg       0.44      0.50      0.47     50400
weighted avg       0.77      0.88      0.82     50400

ROC AUC Score: 0.6230286794886533
```

```
C:\Users\ASUS\Downloads\conda ass\Lib\site-packages\sklearn\metrics\_classifica
tion.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` param
eter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\ASUS\Downloads\conda ass\Lib\site-packages\sklearn\metrics\_classifica
tion.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` param
eter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\ASUS\Downloads\conda ass\Lib\site-packages\sklearn\metrics\_classifica
tion.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` param
eter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```



In [ ]: