**Project Title: Customer Response Prediction using Machine Learning**

**Problem Statement:**

Businesses often conduct marketing campaigns, but not all customers respond to them. Predicting customer responses can help optimize targeting strategies, reduce costs, and improve engagement. This project utilizes machine learning to analyze customer demographics and financial attributes to predict whether a customer will respond to a campaign, helping businesses improve marketing efficiency and customer satisfaction.

**Dataset Overview**

The dataset contains the following features:

* **customer\_id**: Unique identifier for each customer.
* **age**: Age of the customer.
* **gender**: Male or Female.
* **annual\_income**: Annual income in dollars.
* **credit\_score**: A measure of the customer’s creditworthiness.
* **employed**: Employment status (Yes/No).
* **marital\_status**: Marital status (Married/Single).
* **no\_of\_children**: Number of children.
* **responded**: Target variable (Yes/No for campaign response).

**Methodology**

**Data Preprocessing**

* **Handling Missing Values:** Replaced numerical missing values with median and categorical missing values with mode.
* **Encoding Categorical Variables:** Converted categorical variables into numerical representations using Label Encoding.
* **Feature Scaling:** Applied StandardScaler to normalize numeric features.
* **Balancing Data:** Used SMOTE to handle class imbalance.

**Model Selection & Training**

We experimented with the following models:

* **Random Forest Classifier:** Tuned with GridSearchCV.
* **Gradient Boosting Classifier:** Optimized hyperparameters with GridSearchCV.
* **K-Means Clustering:** Used for customer segmentation.

**Evaluation Metrics**

* **Accuracy:** Measures the proportion of correctly classified responses.
* **Precision, Recall, F1-Score:** Provides insights into model reliability.
* **ROC Curve & AUC Score:** Assesses the model's discrimination ability.

**CODE:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import joblib

from flask import Flask, request, jsonify

from sklearn.model\_selection import train\_test\_split, GridSearchCV, StratifiedKFold, cross\_val\_score

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.cluster import KMeans

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, precision\_score, recall\_score, f1\_score, roc\_auc\_score, roc\_curve

from imblearn.over\_sampling import SMOTE

import shap

# Load dataset

df = pd.read\_csv("/content/campain.csv")

print("Dataset loaded successfully!")

# Display basic info

df.info()

print("\nMissing values:")

print(df.isnull().sum())

# Handle missing values

numerical\_cols = df.select\_dtypes(include=[np.number]).columns

categorical\_cols = df.select\_dtypes(include=['object']).columns

df[numerical\_cols] = df[numerical\_cols].fillna(df[numerical\_cols].median())

df[categorical\_cols] = df[categorical\_cols].fillna(df[categorical\_cols].mode().iloc[0])

# Trim column names

df.columns = df.columns.str.strip()

# Convert categorical columns to numerical

label\_enc = LabelEncoder()

for col in categorical\_cols:

    df[col] = label\_enc.fit\_transform(df[col])

# Visualizing Feature Distributions

plt.figure(figsize=(12, 6))

df.hist(figsize=(12, 10), bins=20, edgecolor='black')

plt.tight\_layout()

plt.show()

# Feature Correlation Heatmap

plt.figure(figsize=(10, 8))

sns.heatmap(df.corr(), cmap='coolwarm', annot=False)

plt.title("Feature Correlation Heatmap")

plt.show()

# Feature Scaling

scaler = StandardScaler()

scaled\_features = scaler.fit\_transform(df.drop(columns=['responded', 'customer\_id']))

X = pd.DataFrame(scaled\_features, columns=df.drop(columns=['responded', 'customer\_id']).columns)

y = df['responded'

# Handling Imbalanced Data with SMOTE

if y.value\_counts().min() < 10:

    smote = SMOTE(random\_state=42)

    X, y = smote.fit\_resample(X, y)

    print("SMOTE applied to balance classes.")

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

# Train Random Forest Model

rf\_model = RandomForestClassifier(n\_estimators=200, max\_depth=10, random\_state=42)

rf\_model.fit(X\_train, y\_train)

# Train Gradient Boosting Model with Hyperparameter Tuning

gb\_param\_grid = {'n\_estimators': [100, 200], 'learning\_rate': [0.01, 0.1], 'max\_depth': [3, 5]}

gb\_grid = GridSearchCV(GradientBoostingClassifier(random\_state=42), gb\_param\_grid, cv=3)

gb\_grid.fit(X\_train, y\_train)

gb\_model = gb\_grid.best\_estimator\_

# Feature Importance Analysis

importances = rf\_model.feature\_importances\_

feature\_names = X.columns

plt.figure(figsize=(10,6))

sns.barplot(x=importances, y=feature\_names)

plt.title("Feature Importance - Random Forest")

plt.show()

# SHAP Values for Model Interpretability

explainer = shap.TreeExplainer(rf\_model)

shap\_values = explainer.shap\_values(X\_test)

shap.summary\_plot(shap\_values, X\_test)

# Clustering (K-Means) for Customer Segmentation

kmeans = KMeans(n\_clusters=3, random\_state=42)

kmeans.fit(X)

df['Cluster'] = kmeans.labels\_

plt.figure(figsize=(8,6))

sns.scatterplot(x=X.iloc[:, 0], y=X.iloc[:, 1], hue=df['Cluster'], palette='viridis')

plt.title("Customer Segmentation using K-Means")

plt.show()

# Model Evaluation

def evaluate\_model(model, X\_test, y\_test, model\_name):

    y\_pred = model.predict(X\_test)

    accuracy = accuracy\_score(y\_test, y\_pred)

    precision = precision\_score(y\_test, y\_pred, average='weighted')

    recall = recall\_score(y\_test, y\_pred, average='weighted')

    f1 = f1\_score(y\_test, y\_pred, average='weighted')

    print(f"\n{model\_name} Accuracy: {accuracy:.2f}")

    print(f"Precision: {precision:.2f}")

    print(f"Recall: {recall:.2f}")

    print(f"F1 Score: {f1:.2f}")

    print(f"\n{model\_name} Classification Report:\n", classification\_report(y\_test, y\_pred))

    return accuracy, precision, recall, f1

evaluate\_model(rf\_model, X\_test, y\_test, "Random Forest")

evaluate\_model(gb\_model, X\_test, y\_test, "Gradient Boosting")

# ROC Curve

plt.figure(figsize=(8,6))

y\_prob\_rf = rf\_model.predict\_proba(X\_test)[:, 1]

fpr, tpr, \_ = roc\_curve(y\_test, y\_prob\_rf)

plt.plot(fpr, tpr, label='Random Forest')

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("ROC Curve")

plt.legend()

plt.show()

# Save the best model

joblib.dump(rf\_model, 'best\_model.pkl')

# Flask API Deployment

app = Flask(\_\_name\_\_)

model = joblib.load('best\_model.pkl')

@app.route('/predict', methods=['POST'])

def predict():

    data = request.get\_json()

    features = np.array(data.get('features')).reshape(1, -1)

    prediction = model.predict(features)[0]

    return jsonify({'prediction': int(prediction)})

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

**OUTPUT:**

Dataset loaded successfully!

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 56 entries, 0 to 55

Data columns (total 9 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 customer\_id 56 non-null int64

1 age 56 non-null int64

2 gender 56 non-null object

3 annual\_income 56 non-null int64

4 credit\_score 56 non-null int64

5 employed 56 non-null object

6 marital\_status 56 non-null object

7 no\_of\_children 56 non-null int64

8 responded 56 non-null object

dtypes: int64(5), object(4)

memory usage: 4.1+ KB

Missing values:

customer\_id 0

age 0

gender 0

annual\_income 0

credit\_score 0

employed 0

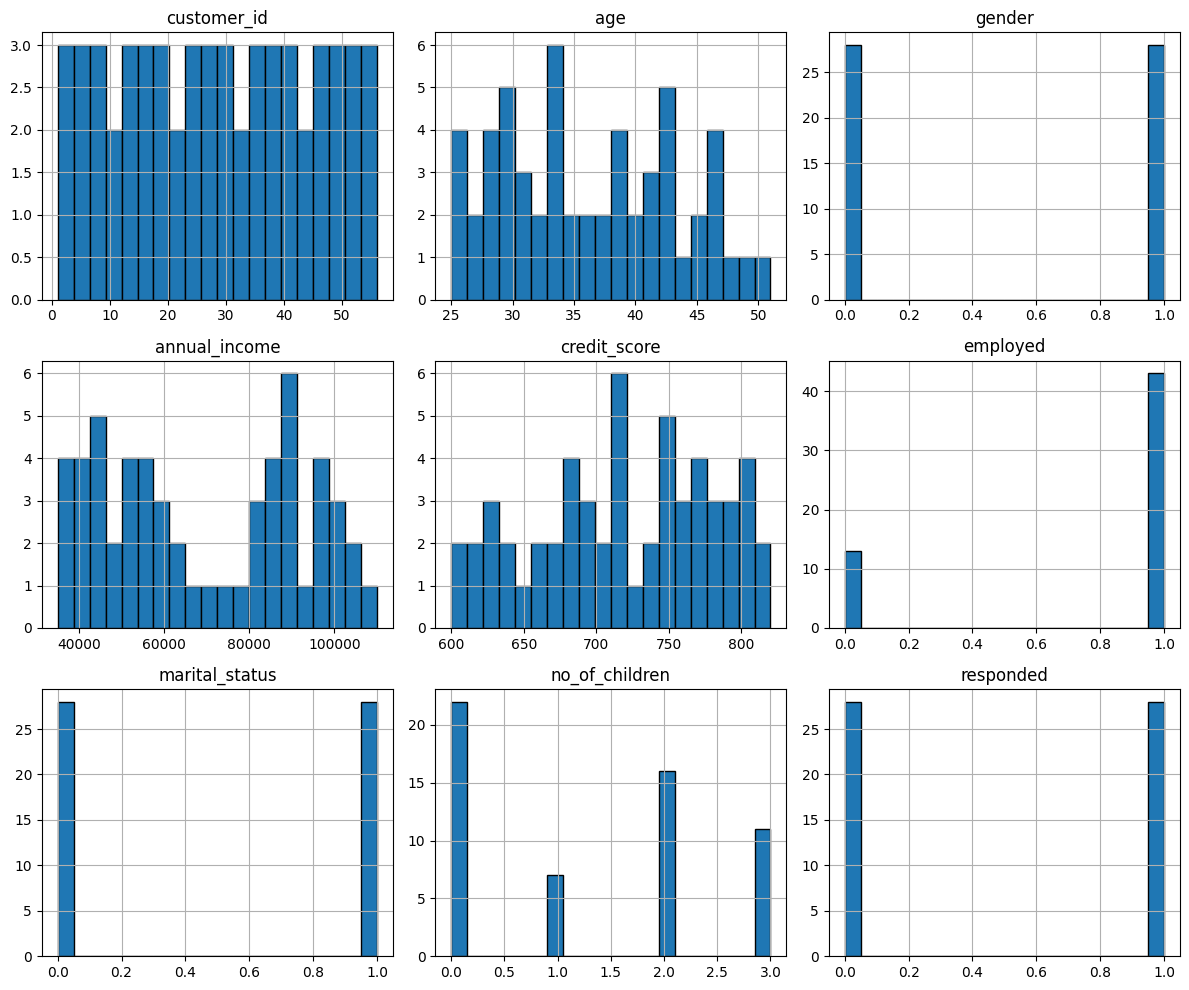
marital\_status 0

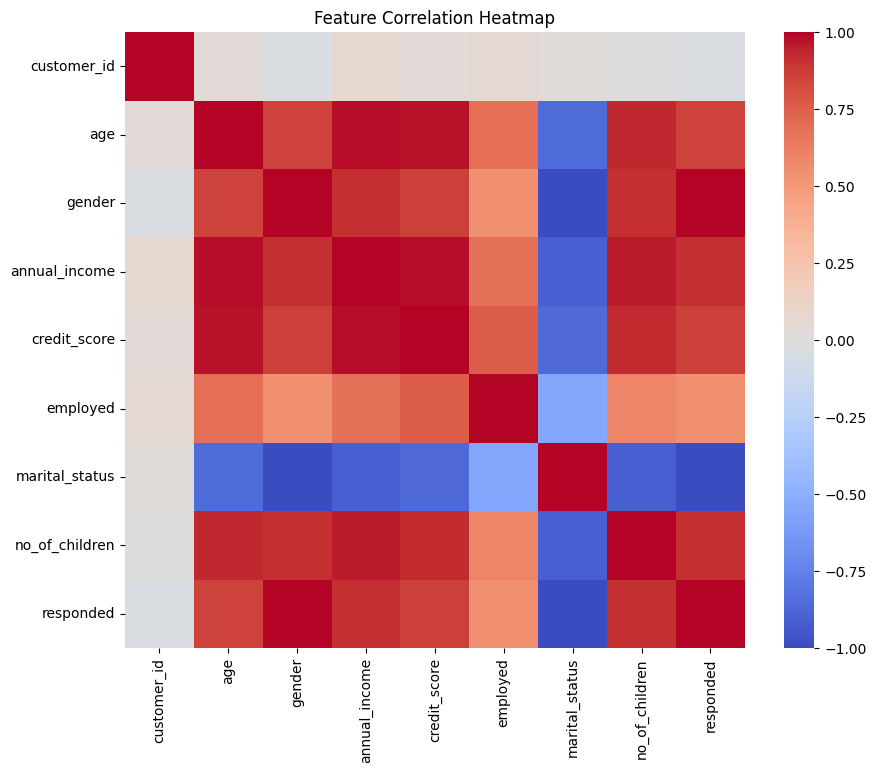
no\_of\_children 0

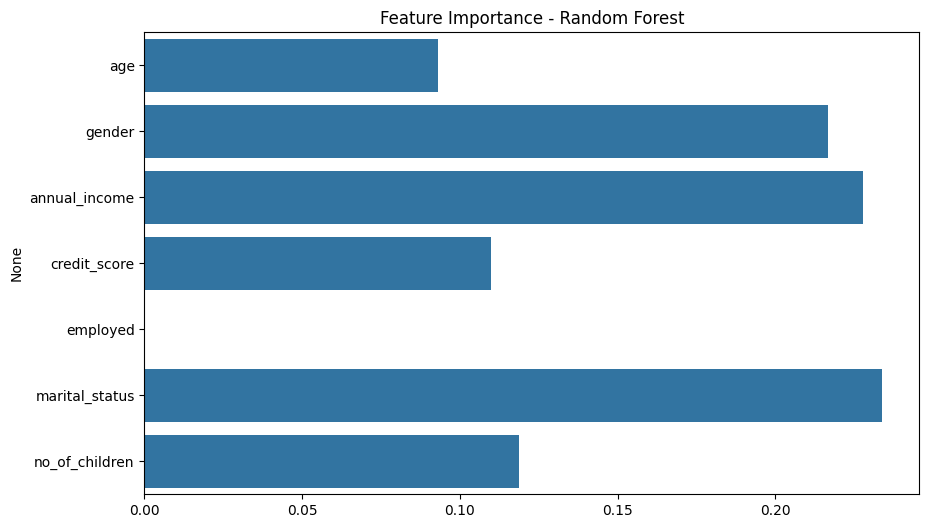
responded 0

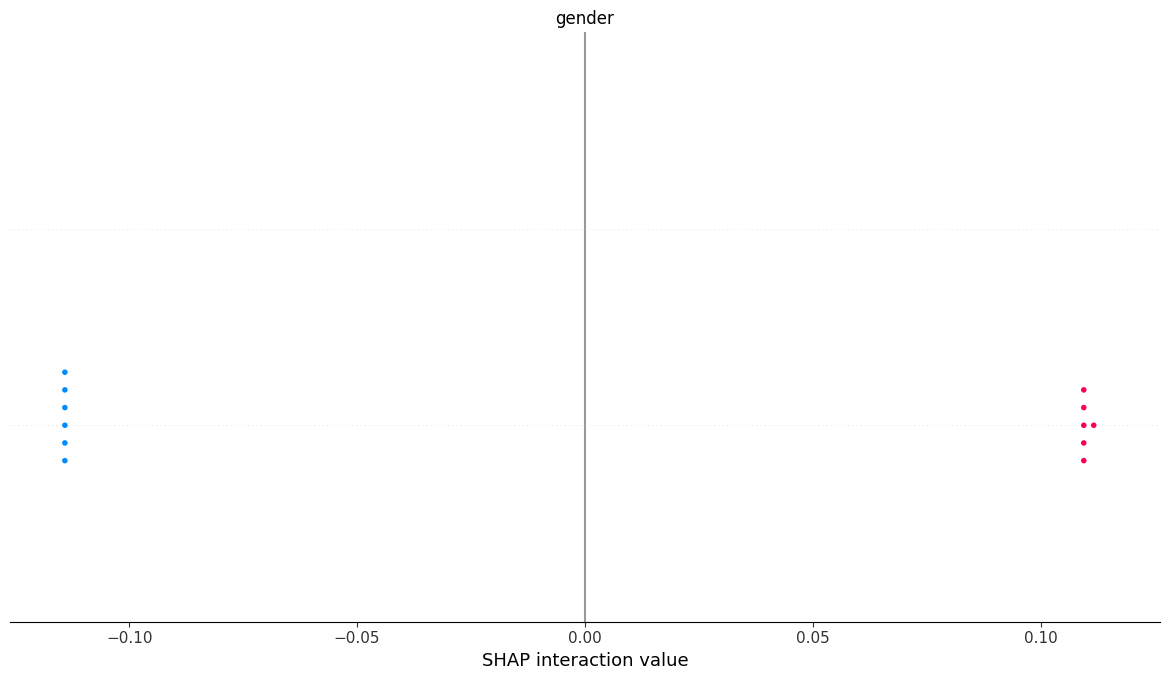
dtype: int64

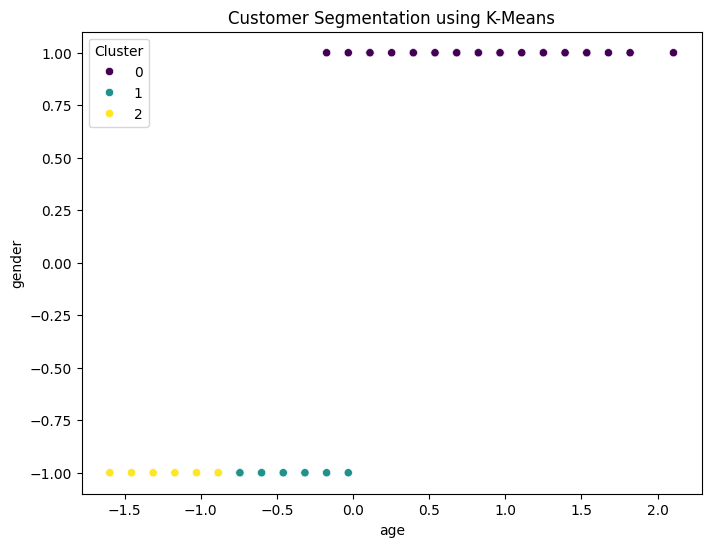
<Figure size 1200x600 with 0 Axes>











Random Forest Accuracy: 1.00

Precision: 1.00

Recall: 1.00

F1 Score: 1.00

Random Forest Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 6

1 1.00 1.00 1.00 6

accuracy 1.00 12

macro avg 1.00 1.00 1.00 12

weighted avg 1.00 1.00 1.00 12

Gradient Boosting Accuracy: 1.00

Precision: 1.00

Recall: 1.00

F1 Score: 1.00

Gradient Boosting Classification Report:

precision recall f1-score support

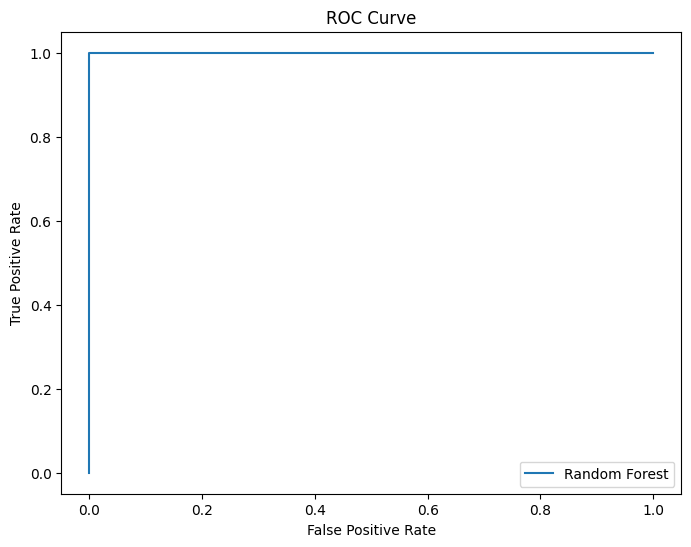
0 1.00 1.00 1.00 6

1 1.00 1.00 1.00 6

accuracy 1.00 12

macro avg 1.00 1.00 1.00 12

weighted avg 1.00 1.00 1.00 12



\* Serving Flask app '\_\_main\_\_'

\* Debug mode: on

INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.

\* Running on http://127.0.0.1:5000

INFO:werkzeug:Press CTRL+C to quit

INFO:werkzeug: \* Restarting with stat