

```
In [2]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import (accuracy_score, classification_report, confusion_matrix,
mean_squared_error, r2_score)

print("Importing Completed")
```

Importing Completed

```
In [3]: data=pd.read_csv("loan_data.csv")
data.head()
```

```
Out[3]:
```

	person_age	person_gender	person_education	person_income	person_emp_exp	person_home_ownership	loan_amnt	loan_in
0	22.0	female	Master	71948.0	0	RENT	35000.0	PERSC
1	21.0	female	High School	12282.0	0	OWN	1000.0	EDUCAT
2	25.0	female	High School	12438.0	3	MORTGAGE	5500.0	MED
3	23.0	female	Bachelor	79753.0	0	RENT	35000.0	MED
4	24.0	male	Master	66135.0	1	RENT	35000.0	MED



```
In [4]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45000 entries, 0 to 44999
Data columns (total 14 columns):
 #   Column                                Non-Null Count  Dtype
---  -
 0   person_age                           45000 non-null  float64
 1   person_gender                         45000 non-null  object
 2   person_education                     45000 non-null  object
 3   person_income                        45000 non-null  float64
 4   person_emp_exp                       45000 non-null  int64
 5   person_home_ownership               45000 non-null  object
 6   loan_amnt                           45000 non-null  float64
 7   loan_intent                         45000 non-null  object
 8   loan_int_rate                       45000 non-null  float64
 9   loan_percent_income                 45000 non-null  float64
10   cb_person_cred_hist_length          45000 non-null  float64
11   credit_score                        45000 non-null  int64
12   previous_loan_defaults_on_file      45000 non-null  object
13   loan_status                         45000 non-null  int64
dtypes: float64(6), int64(3), object(5)
memory usage: 4.8+ MB
```

```
In [5]: data=data.dropna() # in case if there is any null value
```

```
In [6]: num_col=data.select_dtypes(include=['int64','float64']).columns
        cat_col=data.select_dtypes(include=['object']).columns

        print("Numerical- ",list(num_col))
        print("Categorical- ",list(cat_col))
```

```
Numerical-  ['person_age', 'person_income', 'person_emp_exp', 'loan_amnt', 'loan_int_rate', 'loan_percent_income', 'c
b_person_cred_hist_length', 'credit_score', 'loan_status']
Categorical-  ['person_gender', 'person_education', 'person_home_ownership', 'loan_intent', 'previous_loan_defaults_o
n_file']
```

Encoding the col of categories

```
In [7]: data_encoded= data.copy()
        for col in cat_col:
            data_encoded[col]=data_encoded[col].astype('category').cat.codes
```

Splitting feature and label

```
In [18]: # output (target)

target_class='Loan_Status' if 'Loan_Status' in data_encoded.columns else data_encoded.columns[-1]
target_regr='ApplicationIncome' if 'ApplicationIncome' in data_encoded.columns else num_col[-1]

x_class=data_encoded.drop(columns=[target_class]) # input of classification
y_class=data_encoded[target_class] # output of classification

x_regr=data_encoded.drop(columns=[target_regr]) # input of regression
y_regr=data_encoded[target_regr] # output of regression
```

```
In [19]: # data splitting of classification

xc_train,xc_test,yc_train,yc_test=train_test_split(x_class,y_class,test_size=0.2,random_state=42)

xr_train,xr_test,yr_train,yr_test=train_test_split(x_regr,y_regr,test_size=0.2,random_state=42)
```

Decision Tree

i) Classification

```
In [27]: dt_clf=DecisionTreeClassifier(random_state=42)

dt_clf.fit(xc_train,yc_train)

yc_pred_dt=dt_clf.predict(xc_test)
```

ii) Regression

```
In [28]: dt_reg=DecisionTreeRegressor(random_state=42)

dt_reg.fit(xr_train,yr_train)

yr_pred_dt=dt_reg.predict(xr_test)
```

Random Forest

i) Classification

```
In [32]: rf_clf=RandomForestClassifier(random_state=42)

rf_clf.fit(xc_train,yc_train)

yc_pred_rf=rf_clf.predict(xc_test)
```

ii) Regressor

```
In [33]: rf_reg=RandomForestRegressor(random_state=42)

rf_reg.fit(xr_train,yr_train)

yr_pred_rf=rf_reg.predict(xr_test)
```

Evaluations-

```
In [35]: print("====Classification Results====")
print("\n DT Accuracy\n",accuracy_score(yc_test,yc_pred_dt))
print("\n RF Accuracy\n",accuracy_score(yc_test,yc_pred_rf))

print("\n DT Classification Report\n",classification_report(yc_test,yc_pred_dt))
print("\n RF Classification Report\n",classification_report(yc_test,yc_pred_rf))
```

====Classification Results====

DT Accuracy

0.8981111111111111

RF Accuracy

0.9286666666666666

DT Classification Report

	precision	recall	f1-score	support
0	0.93	0.94	0.93	6990
1	0.77	0.77	0.77	2010
accuracy			0.90	9000
macro avg	0.85	0.85	0.85	9000
weighted avg	0.90	0.90	0.90	9000

RF Classification Report

	precision	recall	f1-score	support
0	0.94	0.97	0.95	6990
1	0.89	0.78	0.83	2010
accuracy			0.93	9000
macro avg	0.91	0.87	0.89	9000
weighted avg	0.93	0.93	0.93	9000

```
In [37]: print("====Regressor Results====")
print("\n DT MSE- ",mean_squared_error(yr_test,yr_pred_dt))
print("\n RF MSE- ",mean_squared_error(yr_test,yr_pred_rf))

print("\n DT R2- ",r2_score(yr_test,yr_pred_dt))
print("\n RF R2- ",r2_score(yr_test,yr_pred_rf))
```

====Regressor Results====

DT MSE- 0.10255555555555555

RF MSE- 0.052970666666666666

DT R2- 0.40875024021523265

RF R2- 0.6946153353404649

Visualization of Evaluation Metrics

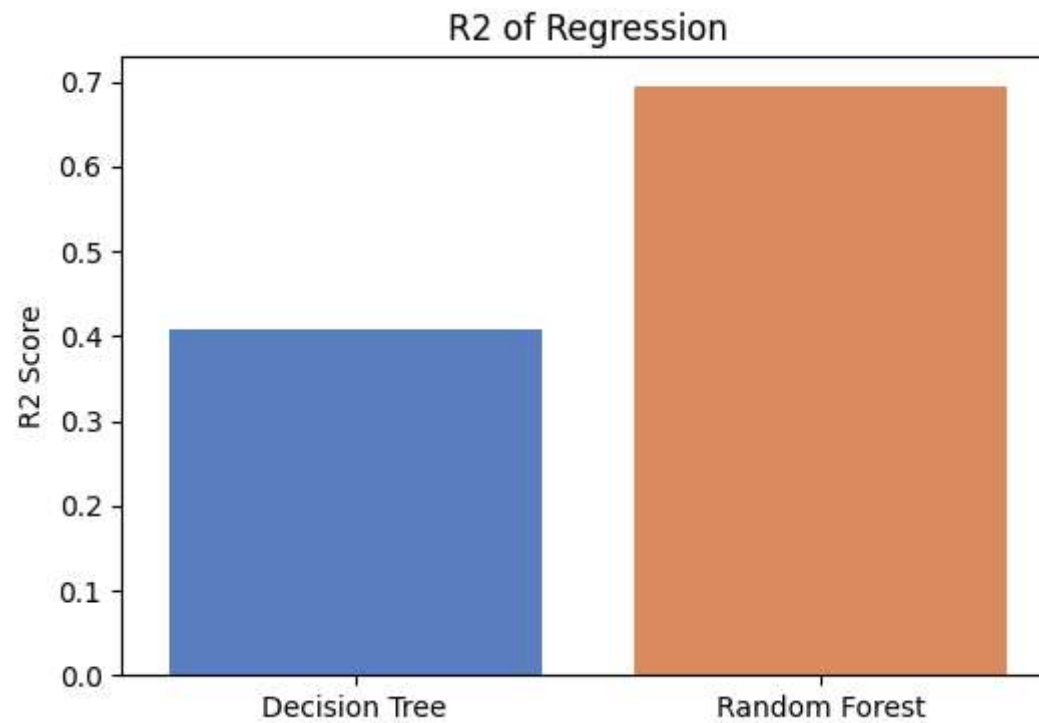
```
In [41]: plt.figure(figsize=(6,4))
sns.barplot(x=['Decision Tree', 'Random Forest'],
            y=[r2_score(yr_test,yr_pred_dt), r2_score(yr_test,yr_pred_rf)],
            palette='muted')
plt.title("R2 of Regression")
plt.ylabel("R2 Score")
plt.show()

plt.figure(figsize=(6,4))
sns.barplot(x=['Decision Tree', 'Random Forest'],
            y=[accuracy_score(yc_test,yc_pred_dt), accuracy_score(yc_test,yc_pred_rf)],
            palette='muted')
plt.title("Accuracy of Regression")
plt.ylabel("Accuracy")
plt.show()
```

C:\Users\ADMIN\AppData\Local\Temp\ipykernel_11136\1748549846.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

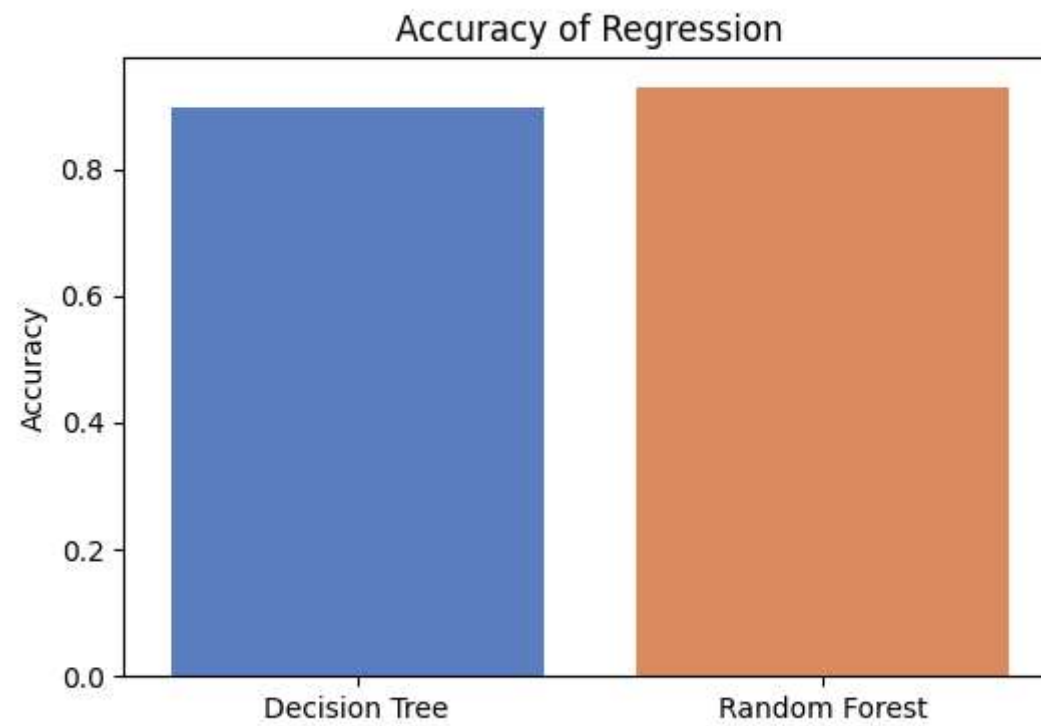
```
sns.barplot(x=['Decision Tree', 'Random Forest'],
```



C:\Users\ADMIN\AppData\Local\Temp\ipykernel_11136\1748549846.py:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x=['Decision Tree', 'Random Forest'],
```



In []: