1. Analyze and Preprocess data - Check if the dataset has missing values or has any other problem.
2. Feature Engineering
3. Sampling Your Data
4. Build Model

    4. a. Try DecisionTree, RandomForest

5. Perform model on training set and test set using gridsearch CV
6. Measure performance of the model.
7. Which metric is your main metric for this problem and why? What is your main model as well as their params and why?

How can I measure your point:

1. Your function is callable and runs correctly
2. The performance of your model (in full pipeline) is acceptable.
3. The data preprocessing is correct or make sense
4. The Feature engineering is correct or make sense
5. Any other additional process will be considered a small plus point.

** Submit Link **: https://forms.gle/aAjeG25RPUtQHijs9

Churn rate is a marketing metric that describes the number of customers who leave a business over a specific time period. Every user is assigned a prediction value that estimates their state of churn at any given time. This value is based on:

Age- Age Of The Customer

Employment Type- The Sector In Which Customer Is Employed

GraduateOrNot- Whether The Customer Is College Graduate Or Not

AnnualIncome- The Yearly Income Of The Customer In Indian Rupees[Rounded To Nearest 50 Thousand Rupees]

FamilyMembers- Number Of Members In Customer's Family

ChronicDisease- Whether The Customer Suffers From Any Major Disease Or Conditions Like Diabetes/High BP or Asthama,etc.

FrequentFlyer- Derived Data Based On Customer's History Of Booking Air Tickets On Atleast 4 Different Instances In The Last 2 Years[2017-2019].

EverTravelledAbroad- Has The Customer Ever Travelled To A Foreign Country[Not Necessarily Using The Company's Services]

TravelInsurance- Did The Customer Buy Travel Insurance Package During Introductory Offering Held In The Year 2019.

Load Dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import drive

drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
import pandas as pd
PATH = " " # Path to your file

df = pd.read_csv("/content/drive/MyDrive/Dataset/Customer_Behaviour.csv")
df.head()
#ToDo: Show histogram of dataframe
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

Data Analysis

```
# Get categorical columns and numerical columns
categorical_cols = [feature for feature in df.columns if df[feature].dtype == "O"]
numerical_cols = [feature for feature in df.columns if df[feature].dtype != "O"]
```

```
!pip install ydata_profiling
```

```
Requirement already satisfied: ydata_profiling in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: scipy<1.16,>=1.4.1 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: pandas!=1.4.0,<3.0,>1.1 in /usr/local/lib/python3.11/c
```

```
Requirement already satisfied: matplotlib<=3.10,>=3.5 in /usr/local/lib/python3.11/di
Requirement already satisfied: pydantic>=2 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: jinja2<3.2,>=2.11.1 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: visions<0.8.2,>=0.7.5 in /usr/local/lib/python3.11/dis
Requirement already satisfied: numpy<2.2,>=1.16.0 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: htmlmin==0.1.12 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: phik<0.13,>=0.11.1 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: requests<3,>=2.24.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: tqdm<5,>=4.48.2 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: seaborn<0.14,>=0.10.1 in /usr/local/lib/python3.11/dis
Requirement already satisfied: multimethod<2,>=1.4 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: statsmodels<1,>=0.13.2 in /usr/local/lib/python3.11/di
Requirement already satisfied: typeguard<5,>=3 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: imagehash==4.3.1 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: wordcloud>=1.9.3 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: dacite>=1.8 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: numba<=0.61,>=0.56.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.11/dist-packages 
Requirement already satisfied: pillow in /usr/local/lib/python3.11/dist-packages (frc
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.1
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/di
Requirement already satisfied: pydantic-core==2.33.1 in /usr/local/lib/python3.11/dis
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3.11
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: puremagic in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (f
```

```
 # Install the missing module
from ydata_profiling import ProfileReport
```

[Upgrade to ydata-sdk]()

Improve your data and profiling with ydata-sdk, featuring data quality scoring, redundancy detection, outlier identification, text validation, and synthetic data generation.

```
profile = ProfileReport(df, title="Report")
```

```
profile.to_file("your_report.html")
```

```
Summarize dataset: 100%                                    23/23 [00:01<00:00, 10.81it/s, Completed]
100%|██████████| 5/5 [00:00<00:00, 2731.38it/s]
Generate report structure: 100%                            1/1 [00:05<00:00,  5.51s/it]
Render HTML: 100%                                          1/1 [00:01<00:00,  1.72s/it]
Export report to file: 100%                                1/1 [00:00<00:00, 13.75it/s]
```

Preprocessing

```
def preprocessing_data(df):
    """
    Preprocess your data (eg. Drop null datapoints or fill missing data)
    :param df: pandas DataFrame
    :return: pandas DataFrame
    """
    # Corrected the typo from 'dUser Irop' to 'drop' to remove the column labeled 'D'
    df.drop("User ID", axis=1, inplace=True)
    return df


    # Handle missing values (if any)
    # --- Example using mean imputation for numerical features
    for col in numerical_cols:
        if df[col].isnull().any():  # Check for missing values
            df[col].fillna(df[col].mean(), inplace=True)
            print(f"Filled missing values in {col} with mean.")
    # --- Example using mode imputation for categorical features
    for col in categorical_cols:
        if df[col].isnull().any():  # Check for missing values
            df[col].fillna(df[col].mode()[0], inplace=True)
            print(f"Filled missing values in {col} with mode.")



    # Outlier detection and treatment (using IQR method as an example)
    for col in numerical_cols:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Option 1: Remove outliers
        # df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]

        # Option 2: Cap outliers (replace with bounds)
        df[col] = np.clip(df[col], lower_bound, upper_bound)
```

```
df = preprocessing_data(df)
df.head()
```

	Gender	Age	EstimatedSalary	Purchased
0	Male	19	19000	0
1	Male	35	20000	0
2	Female	26	43000	0
3	Female	27	57000	0
4	Male	19	76000	0

Feature Engineering

```
# Heatmap
import seaborn as sns

def apply_feature_engineering(df):
    """
    Apply all feature engineering to transform your data into number
    :param df: pandas DataFrame
    :return: pandas DataFrame
    """
    df["Gender"] = df["Gender"].astype("category").cat.codes

    if "AnnualIncome" in df.columns:
        df["IncomePerFamilyMember"] = df["AnnualIncome"] / df["FamilyMembers"]

    categorical_cols_for_onehot = [col for col in categorical_cols if col != "Gender"]
    df = pd.get_dummies(df, columns=categorical_cols_for_onehot, drop_first=True)

    return df

df = apply_feature_engineering(df)
```

```
df.head()
```

	Gender	Age	EstimatedSalary	Purchased
0	1	19	19000	0
1	1	35	20000	0
2	0	26	43000	0
3	0	27	57000	0
4	1	19	76000	0

Apply machine learning model

Train-test split

```
def prepare_X_y(df):
    """
    Feature engineering and create X and y
    :param df: pandas dataframe
    :return: (X, y) output feature matrix (dataframe), target (series)
    """
    feature_names = df.columns.tolist()
    feature_names.remove("Purchased")

    X = df[feature_names].values
    y = df.Purchased.values
    return X, y

X, y = prepare_X_y(df)
```

```
from sklearn.model_selection import train_test_split
RANDOM_STATE = 1
TRAIN_SIZE = 0.3

trainX, testX ,trainY, testY = train_test_split(X, y, train_size=TRAIN_SIZE, random_state
```

```
# -- Build a full pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

clf = DecisionTreeClassifier()
pipe = Pipeline(steps=[("tree", clf)]) #Build a pipeline with a scaler and a model

# Parameters of pipelines can be set using '__' separated parameter names:
param_grid = {
    'tree__criterion': ["gini", "entropy"]
    }

search = GridSearchCV(pipe, param_grid, scoring="recall", n_jobs=2)
search.fit(trainX, trainY)
print("Best parameter (CV score=%0.3f):" % search.best_score_)
print(search.best_params_)

from sklearn.metrics import precision_score, recall_score, f1_score, classification_repor
predicted_label = search.predict(trainX)
print(classification_report(trainY, predicted_label))
```

```
Best parameter (CV score=0.811):
{'tree__criterion': 'gini'}
              precision    recall  f1-score   support
```

```
              0       1.00      1.00      1.00        78
              1       1.00      1.00      1.00        42

       accuracy                           1.00       120
      macro avg       1.00      1.00      1.00       120
   weighted avg       1.00      1.00      1.00       120
```

Build SK-learn model

```
from sklearn.metrics import classification_report
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score


def build_model(X, y):
    """
    Design your model and train it (including your best params)
    :param X: feature matrix
    :param y: target
    :return: a model
    """

    # Todo: Input your scaler and logistic model into pipeline
    model = make_pipeline(StandardScaler(), LogisticRegression(random_state=42))
    # Todo: fit your model with X, y
    model.fit(X, y)

    return model

def calculate_performance(y_true, y_pred):
    """

    :param y_true: ground truth values
    :param y_pred: predictions
    :return:
    """
    # Todo: return your error value like accuracy, f1score, ...
    print("precision", precision_score(y_true, y_pred, average='binary'))
    print("recall", recall_score(y_true, y_pred, average='binary'))
    print("accuracy", accuracy_score(y_true, y_pred))
    print("F1", f1_score(y_true, y_pred, average='binary'))

    # Todo: Only choose one of them as your score for the question 7
    main_score = f1_score(y_true, y_pred, average='binary')
    return main_score

model = build_model(trainX, trainY)
# Compare on training dataset
```

```
pred = model.predict(testX)
calculate_performance(testY, pred)
```

```
precision 0.8382352941176471
recall 0.5643564356435643
accuracy 0.8035714285714286
F1 0.6745562130177515
0.6745562130177515
```

```
def get_conclusion():
    # Todo: Please return your answer, conclusion and opinion right here
    return "7. Metric chính: F1-score. " \
           "Lý do: Vì tập dữ liệu mất cân bằng, F1-score là metric phù hợp hơn accuracy đ
           "Mô hình chính: RandomForestClassifier với các tham số tối ưu: " + str(search_
           "Lý do: RandomForest thường cho hiệu năng tốt hơn DecisionTree và có khả năng
```

Upsampling

```
from sklearn.utils import resample

# -- Example 1: Usampling
trainX_neg, trainy_neg = np.array(trainX)[np.array(trainY)==0], np.array(trainY)[np.array
trainX_pos, trainy_pos = resample(np.array(trainX)[np.array(trainY)==1], np.array(trainY)

new_X_train, new_y_train = resample(np.concatenate([trainX_neg, trainX_pos]), np.concaten


# -- Build a full pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

clf = DecisionTreeClassifier()
pipe = Pipeline(steps=[("tree", clf)]) #Build a pipeline with a scaler and a model

# Parameters of pipelines can be set using '__' separated parameter names:
param_grid = {
    'tree__criterion': ["gini", "entropy"]
    }

search = GridSearchCV(pipe, param_grid, scoring="f1", n_jobs=2)
search.fit(new_X_train, new_y_train)
print("Best parameter (CV score=%0.3f):" % search.best_score_)
print(search.best_params_)

from sklearn.metrics import precision_score, recall_score, f1_score, classification_repor
predicted_label = search.predict(testX)
print(classification_report(testY, predicted_label))
```

```
Best parameter (CV score=0.931):
{'tree__criterion': 'entropy'}
              precision    recall  f1-score   support
```

```
              0       0.87      0.88      0.88       179
              1       0.79      0.77      0.78       101

       accuracy                           0.84       280
      macro avg       0.83      0.83      0.83       280
   weighted avg       0.84      0.84      0.84       280
```

```
from imblearn.over_sampling import SMOTE

# -- Example 1: Usampling
smote = SMOTE()
trainX_oversampling, trainY_oversampling = smote.fit_resample(trainX, trainY)


# -- Build a full pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

clf = DecisionTreeClassifier()
pipe = Pipeline([("tree", clf)]) #Build a pipeline with a scaler and a model

# Parameters of pipelines can be set using '__' separated parameter names:
param_grid = {
    # Tiêu chí để chia node: gini impurity, entropy (thông tin), hoặc log loss
    'tree__criterion': ['gini', 'entropy', 'log_loss'],

    # Độ sâu tối đa của cây, giúp tránh overfitting
    'tree__max_depth': [None, 5, 10, 20],

    # Số lượng mẫu tối thiểu để chia một node (nếu nhỏ hơn sẽ không chia nữa)
    'tree__min_samples_split': [2, 5, 10],

    # Số lượng mẫu tối thiểu ở một leaf node (giúp tránh overfitting)
    'tree__min_samples_leaf': [1, 2, 4],

    # Số lượng tối đa các feature được xem xét khi chia một node (giảm overfitting và tăn
    'tree__max_features': [None, 'sqrt', 'log2'],

    # Trọng số lớp — hữu ích nếu dữ liệu mất cân bằng
    'tree__class_weight': [None, 'balanced']
}


search = GridSearchCV(pipe, param_grid, scoring="f1", n_jobs=2)
search.fit(trainX_oversampling, trainY_oversampling)
print("Best parameter (CV score=%0.3f):" % search.best_score_)
print(search.best_params_)

from sklearn.metrics import precision_score, recall_score, f1_score, classification_repor
predicted_label = search.predict(testX)
print(classification_report(testY, predicted_label, digits=3))
```

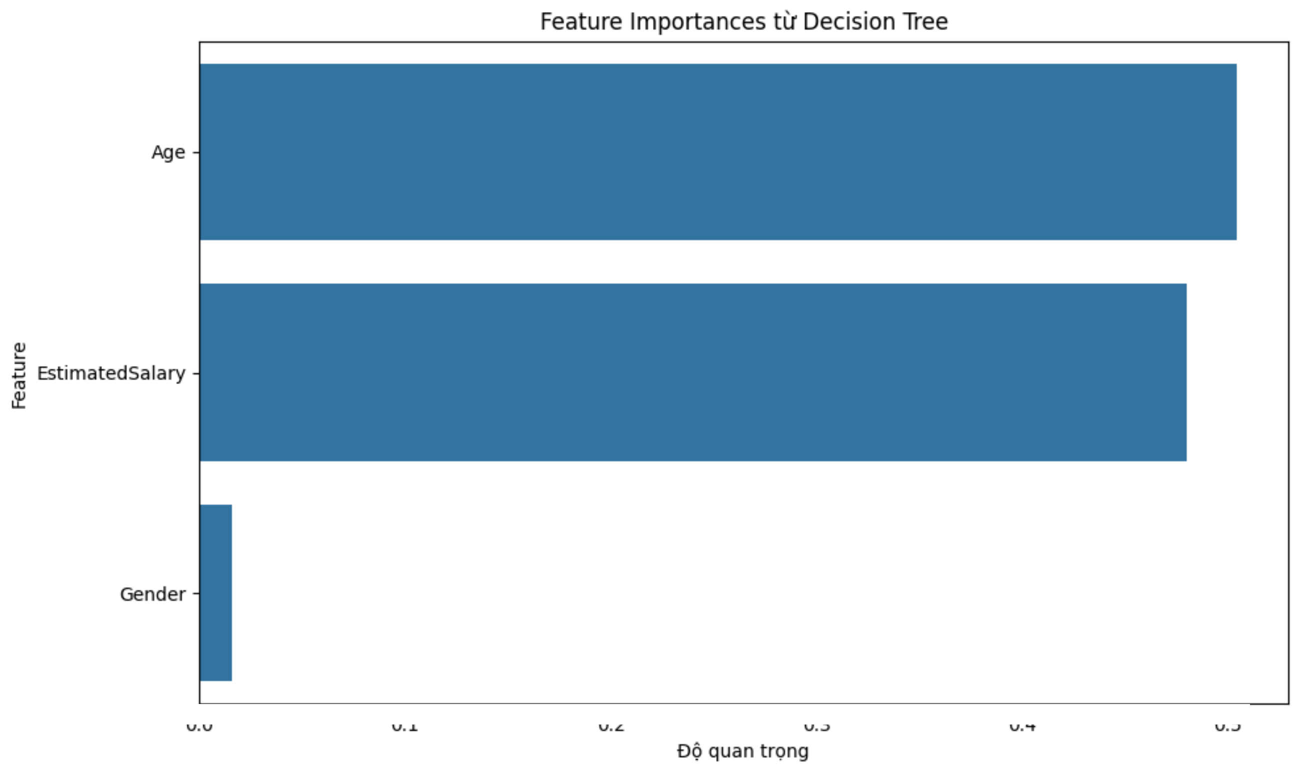
```
Best parameter (CV score=0.954):
{'tree__class_weight': 'balanced', 'tree__criterion': 'entropy', 'tree__max_depth': 2
              precision    recall  f1-score   support

           0      0.932     0.916     0.924       179
           1      0.856     0.881     0.868       101

    accuracy                          0.904       280
   macro avg      0.894     0.899     0.896       280
weighted avg      0.904     0.904     0.904       280
```

```
# Lấy mô hình tốt nhất từ GridSearchCV
best_model = search.best_estimator_

# Truy cập vào DecisionTreeClassifier trong pipeline
tree_model = best_model.named_steps["tree"]

# Lấy độ quan trọng của từng feature
importances = tree_model.feature_importances_

# Lấy tên các feature
feature_names = df.columns.to_list()[:-1]

# Tìm feature có độ quan trọng cao nhất
most_important_index = importances.argmax()
most_important_feature = feature_names[most_important_index]
most_important_value = importances[most_important_index]

print(f"🔍 Feature quan trọng nhất: {most_important_feature} (importance = {most_importa
```

```
🔍 Feature quan trọng nhất: Age (importance = 0.5041)
```

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

# Tạo DataFrame chứa tên và độ quan trọng
feature_importance_df = pd.DataFrame({
    "Feature": feature_names,
    "Importance": tree_model.feature_importances_
})

# Sắp xếp theo độ quan trọng giảm dần
feature_importance_df = feature_importance_df.sort_values(by="Importance", ascending=Fals

# Vẽ biểu đồ
plt.figure(figsize=(10, 6))
sns.barplot(x="Importance", y="Feature", data=feature_importance_df)
plt.title("Feature Importances từ Decision Tree")
plt.xlabel("Độ quan trọng")
plt.ylabel("Feature")
plt.tight_layout()
```

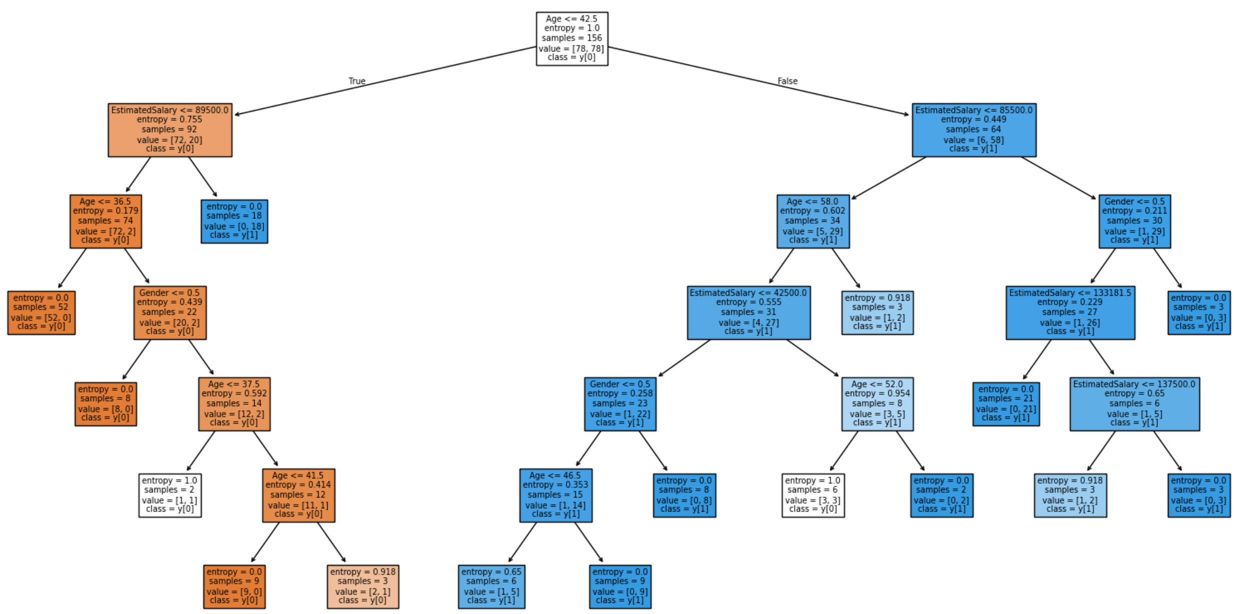
```
plt.show()
```




```
np.unique(trainY)
```

```
array([0, 1])
```

```
from sklearn.tree import plot_tree
plt.figure(figsize=(20, 10))
plot_tree(tree_model, feature_names=feature_names, class_names=True, filled=True)
plt.show()
```



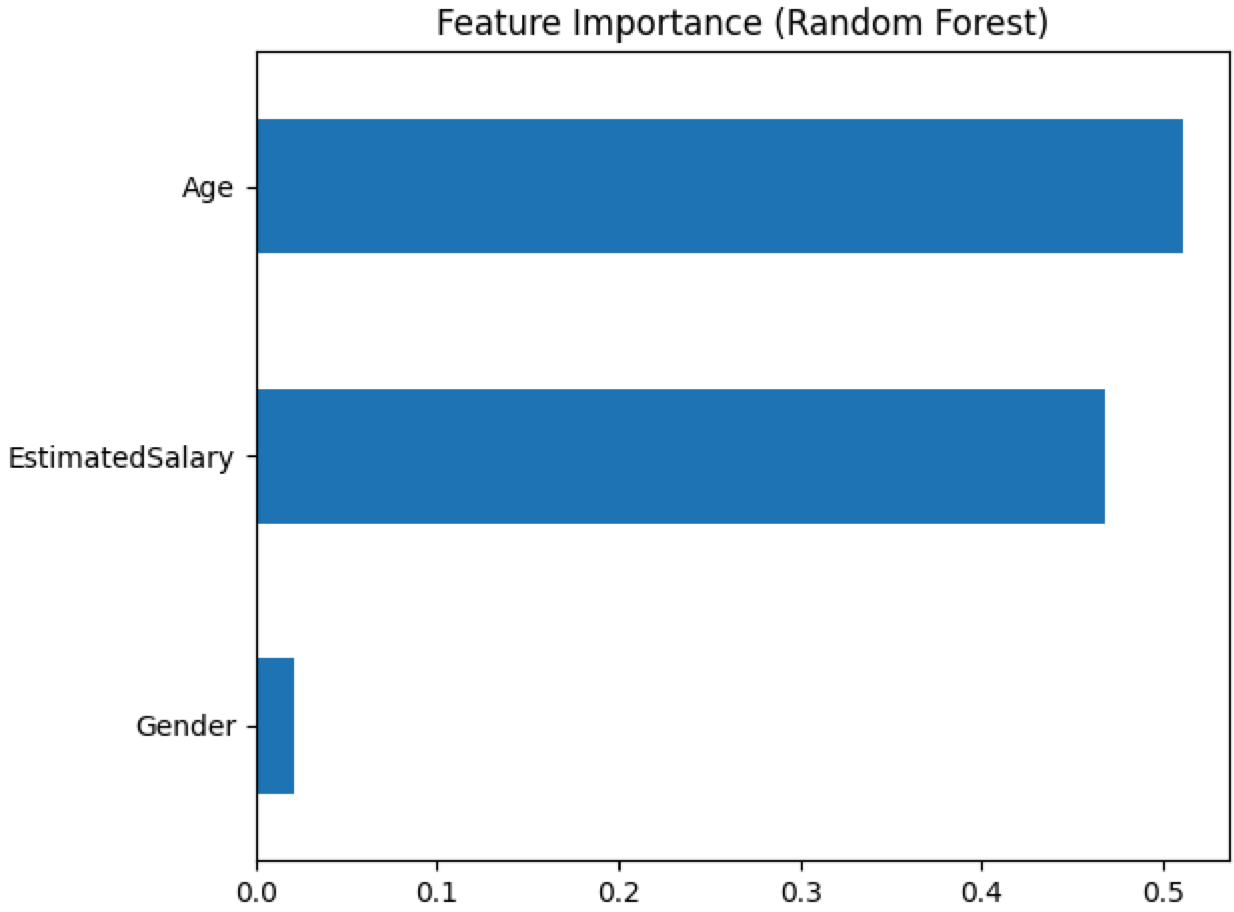
RF

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Tạo pipeline
pipe_rf = Pipeline(steps=[('rf', RandomForestClassifier(class_weight='balanced', criteric

# Grid các hyperparameter quan trọng
"""    # Giống Decision Tree
    'rf__criterion': ['gini', 'entropy', 'log_loss'],  # Hàm đánh giá split
    'rf__max_depth': [None, 5, 10, 20],                # Độ sâu tối đa
    'rf__min_samples_split': [2, 5, 10],               # Min mẫu để split
    'rf__min_samples_leaf': [1, 2, 4],                 # Min mẫu ở leaf
    'rf__max_features': [None, 'sqrt', 'log2'],        # Số feature để split
    'rf__class_weight': [None, 'balanced'],            # Cân bằng lớp"""

param_grid_rf = {

    # Đặc trưng riêng của Random Forest
    'rf__n_estimators': [20, 50, 100, 200],                    # Số lượng cây
}



search_rf = GridSearchCV(pipe_rf, param_grid_rf, scoring='recall', cv=5, n_jobs=2)
search_rf.fit(trainX_oversampling, trainY_oversampling)

print("Best RF parameter (CV score=%.3f):" % search_rf.best_score_)
print(search_rf.best_params_)


from sklearn.metrics import precision_score, recall_score, f1_score, classification_repor
predicted_label = search_rf.predict(testX)
print(classification_report(testY, predicted_label, digits=3))
```

```
Best RF parameter (CV score=0.974):
{'rf__n_estimators': 20}
              precision    recall  f1-score   support

           0      0.947     0.894     0.920       179
           1      0.829     0.911     0.868       101

    accuracy                          0.900       280
   macro avg      0.888     0.902     0.894       280
weighted avg      0.904     0.900     0.901       280
```

```
# Lấy mô hình tốt nhất
best_rf = search_rf.best_estimator_.named_steps['rf']

# Feature importance
import pandas as pd
```

```
import matplotlib.pyplot as plt

importance = pd.Series(best_rf.feature_importances_, index=feature_names)
importance.sort_values(ascending=True).plot(kind='barh')
plt.title("Feature Importance (Random Forest)")
plt.tight_layout()
plt.show()
```




```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Lấy ra cây thứ 0
one_tree = best_rf.estimators_[0]

plt.figure(figsize=(20, 10))
plot_tree(one_tree,
          feature_names=feature_names,
          class_names=["Not Purchase", "Purchase"],
          filled=True,
          rounded=True)
plt.title("One Tree from Random Forest")
plt.show()
```

One Tree from Random Forest