BUSN 41204 -- Spring 2025 -- Problem Set 1

Name: Praphulla Chandra

Date: 04/01/2025

```
pip install shap flaml
```

```
Requirement already satisfied: shap in /usr/local/lib/python3.11/dist-packages (0.47.
Requirement already satisfied: flaml in /usr/local/lib/python3.11/dist-packages (2.3.
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (fro
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.1
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/di
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>=1.2.0 in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (f
```

Note: If you want to re-run the AutoML code, you may need to downgrade NumPy and restart the session. After restarting, re-import NumPy. This step is only necessary if you intend to re-run the AutoML code.

```
# Downgrade NumPy and then restart the session by clicking the Runtime tab
!pip uninstall -y numpy
!pip install numpy==1.26.3
```

```
Found existing installation: numpy 2.0.2
Uninstalling numpy-2.0.2:
  Successfully uninstalled numpy-2.0.2
Collecting numpy==1.26.3
  Downloading numpy-1.26.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.2/61.2 kB 4.0 MB/s eta 0:00:00
Downloading numpy-1.26.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.3/18.3 MB 80.8 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-1.26.3
```

```
from flaml import AutoML
```

```
import ipywidgets as widgets
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import shap

# from flaml import AutoML
from sklearn.cluster import KMeans
from sklearn.datasets import load_wine
from sklearn.decomposition import PCA
from sklearn.ensemble import (
    GradientBoostingClassifier,
    GradientBoostingRegressor,
    RandomForestClassifier,
    RandomForestRegressor,
)
from sklearn.linear_model import LinearRegression
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix,
    mean_absolute_error,
    mean_squared_error,
    r2_score,
)
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.tree import DecisionTreeClassifier
```

Assignment: Understanding House Price Prediction Using Regression Models

Objective: The goal of this assignment is to explore the relationship between house prices and various features using exploratory data analysis (EDA) and regression models. By the end of the assignment, students will be able to visualize data, apply regression techniques, and compare model performances. Before we get started, we import all the packages used in this problem set.

Part 1: Dataset Overview and Preprocessing

1.**Dataset Overview**

The dataset contains house sale prices and various features such as:

date: The date the house was sold.

price: Target variable representing house sale prices.

bedrooms: Number of bedrooms.

bathrooms: Number of bathrooms.

sqft_living: Square footage of the living space.

sqft_lot: Square footage of the lot.

floors: Number of floors.

waterfront: Indicator if the house is located on a waterfront.

view: View rating of the house.

condition: Condition rating of the house.

sqft_above: Square footage of the living space above ground.

sqft_basement: Square footage of the basement.

yr_built: Year the house was built.

yr_renovated: Year of renovation (0 if never renovated).

street: Street address of the property.

city: City where the house is located.

statezip: State and zip code.

country: Country of the property.

The dataset includes both numerical and categorical features.

Check the first few rows of the dataset:

```
df = pd.read_csv(
    "https://raw.githubusercontent.com/chansen776/MBA-ML-Course-Materials/refs/heads/main
)
print(df.head())
```

```
                  date      price  bedrooms  bathrooms  sqft_living  sqft_lot  \
0  2014-05-02 00:00:00   313000.0       3.0       1.50         1340      7912
1  2014-05-02 00:00:00  2384000.0       5.0       2.50         3650      9050
2  2014-05-02 00:00:00   342000.0       3.0       2.00         1930     11947
3  2014-05-02 00:00:00   420000.0       3.0       2.25         2000      8030
4  2014-05-02 00:00:00   550000.0       4.0       2.50         1940     10500
```

```
   floors  waterfront  view  condition  sqft_above  sqft_basement  yr_built  \
0     1.5           0     0          3        1340              0      1955
1     2.0           0     4          5        3370            280      1921
2     1.0           0     0          4        1930              0      1966
3     1.0           0     0          4        1000           1000      1963
4     1.0           0     0          4        1140            800      1976

   yr_renovated                    street       city  statezip country
0          2005      18810 Densmore Ave N  Shoreline  WA 98133     USA
1             0           709 W Blaine St    Seattle  WA 98119     USA
2             0  26206-26214 143rd Ave SE       Kent  WA 98042     USA
3             0           857 170th Pl NE   Bellevue  WA 98008     USA
4          1992         9105 170th Ave NE    Redmond  WA 98052     USA
```

2.**Feature Engineering & Encoding Categorical Data**

Convert categorical features (city, statezip, street) into numerical representations.

Apply one-hot encoding where necessary:

```
df = pd.get_dummies(df, columns=["city", "statezip"], drop_first=True)
```

3.**Feature Scaling**

Important: When standardizing data, ensure that the scaling is performed only on the training data to prevent data leakage.

The mean and standard deviation should be calculated from the training data and then applied to both the training and test sets.

```
# Define predictor and target variable
X = df[["sqft_living"]]
y = df["price"]

# 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_state=42
)

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit scaler only on training data and transform both training and test sets
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

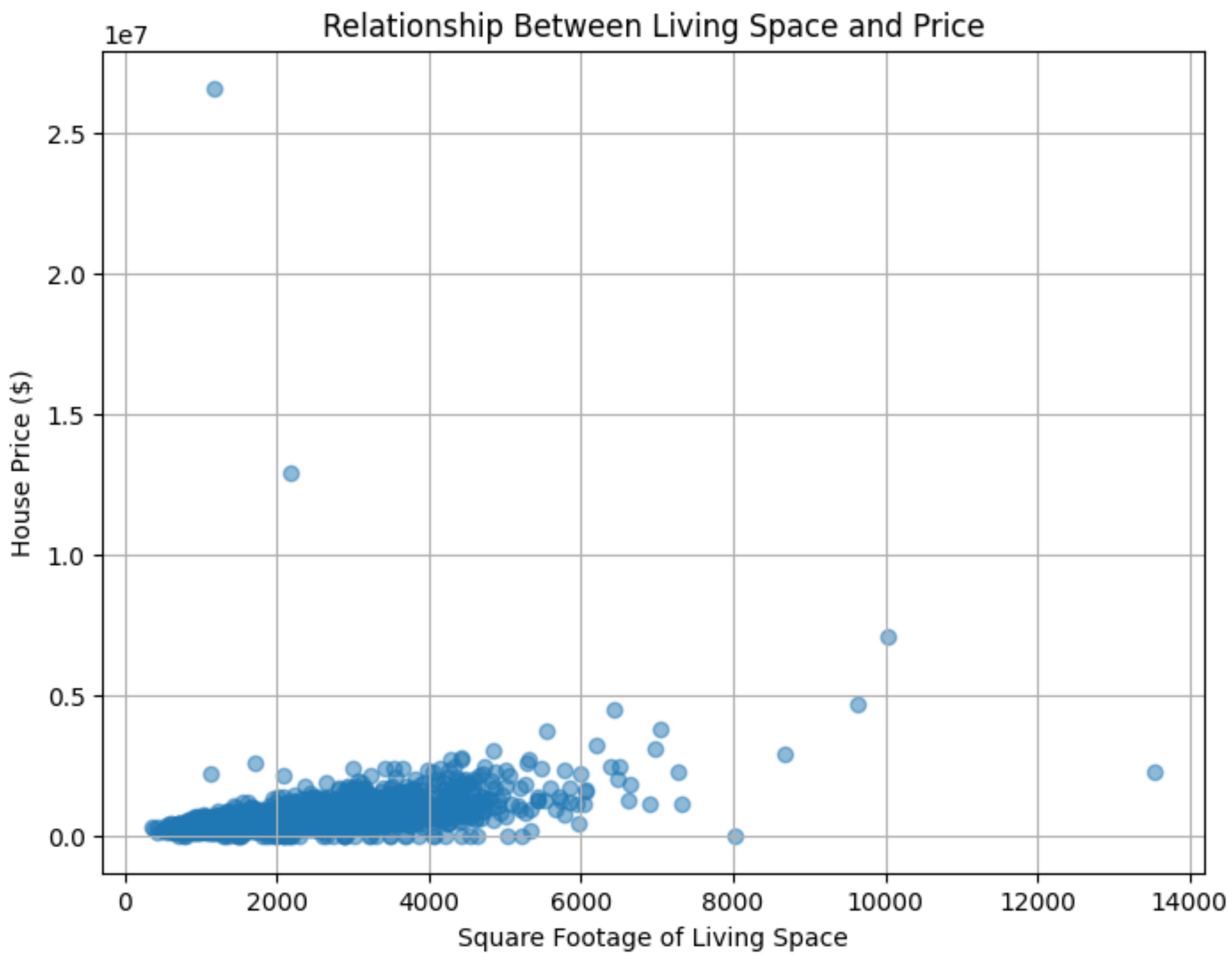
Part 2: Exploratory Data Analysis (EDA)

Visualization

1.Scatter Plot of sqft_living vs. price

```
plt.figure(figsize=(8, 6))
plt.scatter(df["sqft_living"], df["price"], alpha=0.5)
plt.xlabel("Square Footage of Living Space")
plt.ylabel("House Price ($)")
plt.title("Relationship Between Living Space and Price")
plt.grid(True)
plt.show()

print("Correlation of sqft_living on price: ", df["sqft_living"].corr(df["price"]))
```



```
Correlation of sqft_living on price:  0.43041002543262824
```

2.**Task:**Create a scatter plot of sqft_basement vs. price.

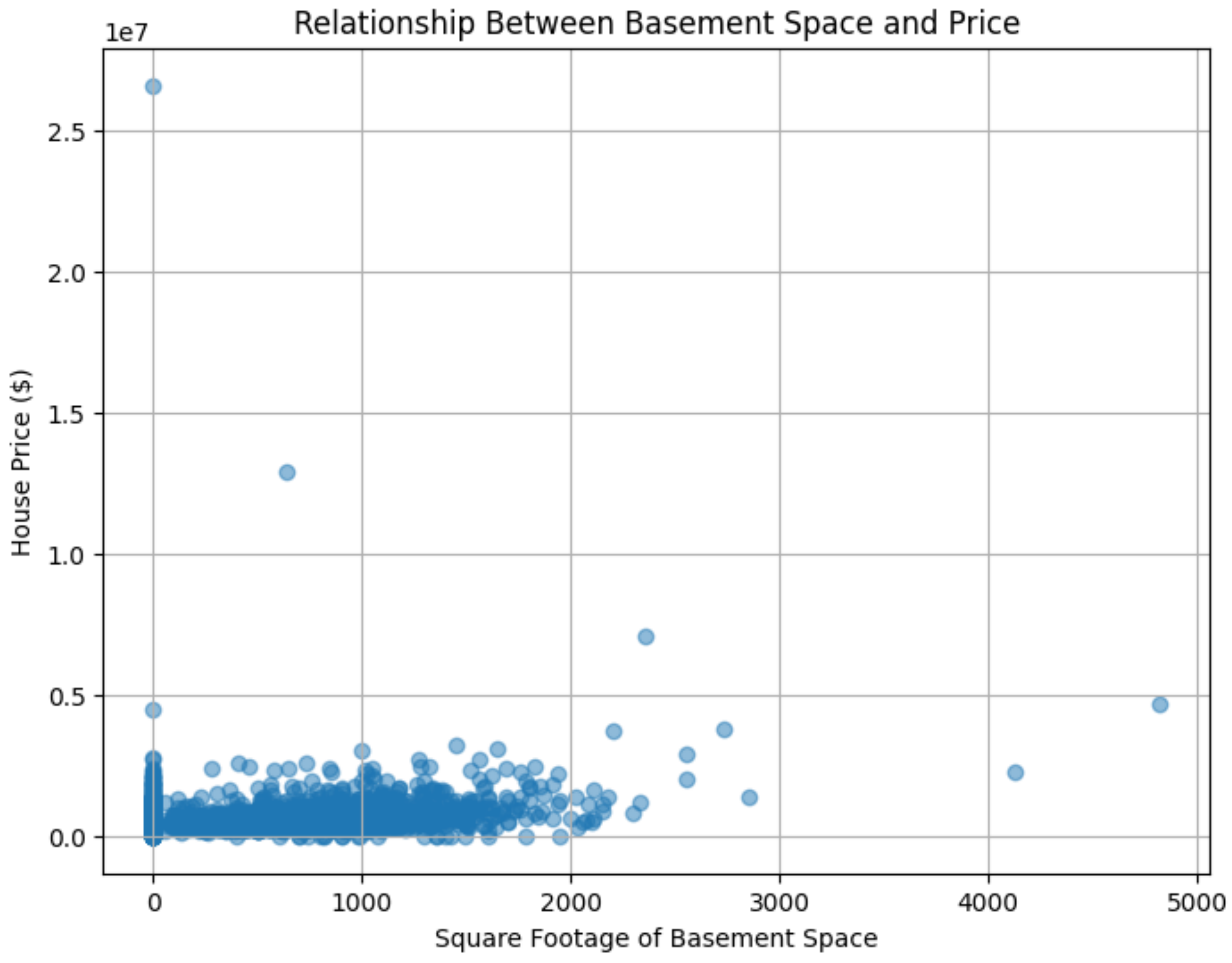
You have code for the scatter plot of sqft_living vs. price above.

The variable name you need is sqft_basement.

You can use ChatGPT or any other LLM to ask how to modify the provided code.

```
# Insert code and plot here
plt.figure(figsize=(8, 6))
plt.scatter(df["sqft_basement"], df["price"], alpha=0.5)
plt.xlabel("Square Footage of Basement Space")
plt.ylabel("House Price ($)")
plt.title("Relationship Between Basement Space and Price")
plt.grid(True)
plt.show()

print("Correlation of sqft_basement on price: ", df["sqft_basement"].corr(df["price"]))
```



```
Correlation of sqft_basement on price:  0.21042657177482943
```

3.**Question:** Which feature do you think has a stronger correlation with price: sqft_living or sqft_basement? Calculate the correlation coefficient for both and explain your reasoning.

Your Answer:

The sqft_living has a stronger correlation with price. Simply by looking at the two charts, we can see that the plot of price against living space shows less vertical variance at each living space. But if we also look at the correlations, price's correlation with living space (at 0.43) is much better than its correlation with basement space (0.21), suggesting more of its variance is explained by living space than by basement space.

Part 3: Building Regression Models

1.Linear Regression (Example Given)

2.Train a **Linear Regression model** using sqft_living as the predictor.

3.Evaluate the model using R-squared (R²), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE)

```
X = df[["sqft_living"]]
y = df["price"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("R²:", r2_score(y_test, y_pred))
print("MAE:", mean_absolute_error(y_test, y_pred))
print("RMSE:", mean_squared_error(y_test, y_pred))
```

```
R²: 0.029065410341410414
MAE: 225375.25345857345
RMSE: 990204087727.1417
```

Random Forest and Gradient Boosting Regression

Below is the code for Random Forest and Gradient Boosting Regression. Do not worry about understanding every part of the code initially.

```
# Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)
```

```
y_pred_rf = rf_model.predict(X_test_scaled)

print("Random Forest Results:")
print("R²:", r2_score(y_test, y_pred_rf))
print("MAE:", mean_absolute_error(y_test, y_pred_rf))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred_rf)))

# Gradient Boosting Regressor
gb_model = GradientBoostingRegressor(
    n_estimators=100, learning_rate=0.1, random_state=42
)
gb_model.fit(X_train_scaled, y_train)
y_pred_gb = gb_model.predict(X_test_scaled)

print("Gradient Boosting Results:")
print("R²:", r2_score(y_test, y_pred_gb))
print("MAE:", mean_absolute_error(y_test, y_pred_gb))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred_gb)))
```

```
Random Forest Results:
R²: 0.005948601777127971
MAE: 233146.1318696576
RMSE: 1006866.2673542678
Gradient Boosting Results:
R²: 0.006307078747391892
MAE: 224314.3204614628
RMSE: 1006684.7018355782
```

Your Task:

1. Explain what the provided code is doing. You can use ChatGPT or any LLM to help you understand it.

[Insert explanation here]

1. The code is fitting 3 separate models (linear regression, random forest, and gradient boosting regressor) to predict house prices based on sqft_living. It is then validating these models by predicting prices for the test data (that it was not trained on) to evaluate how good these models are. Specifically we are using R², Mean Absolute Error (MAE), and Root Mean Square Error (RMSE) to evaluate how well the predictive models work.

Compare the Results:

2. Which model performed better based on the evaluation metrics (R², MAE, RMSE)?
3. Consider the presence of outliers in the scatter plot from the beginning of the assignment.
4. Which evaluation metric (RMSE, R², or MAE) do you think is better for this example?

5. Explain your reasoning.

Think about how each metric handles outliers.

You can use ChatGPT to help think about the trade-offs.

Hint: We have used this example throughout most of Notes 1. If you set the seed to the same value used in those notebooks, you can compare your results.

[Insert your reasoning here]

2. Based on R², the linear model fit the best, since we want to maximize R². Based on MAE or on RMSE, the Gradient Boosting performed the best, since we want to minimize MAE and RMSE.
3. Mean Absolute Error is least affected by outliers, since it does not remove the effect of direction of the prediction. The error where the predicted datapoint is higher than reality can be offset by another datapoint where the predicted datapoint is lower than reality.
4. I would lean towards the MAE and RMSE metrics here, since they both suggest that the Gradient Boosted model has the best predictor.
5. Each of these metrics measure something slightly different in how well these models predict the underlying behavior. The R² is somewhat more sensitive to outliers (while the MAE is least sensitive). Since both the MAE and the RMSE suggest that the Boosted Gradient model best fits the data, we use a voting approach to suggest that the Gradient Boosted model appears to have the best predictor.

Bonus Question:

1.Experiment by adding more features (e.g., bedrooms, bathrooms, sqft_basement).

2.Compare which model performs best with multiple predictors.

```
#Model using additional fields (bedrooms, bathrooms, year built)
X = df[["sqft_living", "bedrooms", "bathrooms", "yr_built"]]
y = df["price"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

#Linear Model Regressor
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
print("Linear Model Results:")
print("R²:", r2_score(y_test, y_pred))
print("MAE:", mean_absolute_error(y_test, y_pred))
print("RMSE:", mean_squared_error(y_test, y_pred))

# Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)
y_pred_rf = rf_model.predict(X_test_scaled)

print("Random Forest Results:")
print("R²:", r2_score(y_test, y_pred_rf))
print("MAE:", mean_absolute_error(y_test, y_pred_rf))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred_rf)))

# Gradient Boosting Regressor
gb_model = GradientBoostingRegressor(
    n_estimators=100, learning_rate=0.1, random_state=42
)
gb_model.fit(X_train_scaled, y_train)
y_pred_gb = gb_model.predict(X_test_scaled)

print("Gradient Boosting Results:")
print("R²:", r2_score(y_test, y_pred_gb))
print("MAE:", mean_absolute_error(y_test, y_pred_gb))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred_gb)))
```

```
Linear Model Results:
R²: 0.03059903758118343
MAE: 216572.14898502768
RMSE: 988640023599.6014
Random Forest Results:
R²: 0.005948601777127971
MAE: 233146.1318696576
RMSE: 1006866.2673542678
Gradient Boosting Results:
R²: 0.006307078747391892
MAE: 224314.3204614628
RMSE: 1006684.7018355782
```

Additional continuous variables were selected that were judged to be orthogonal to square footage and may impact the price.

The linear model improved in R², but declined in both MAE and RMSE. The other two models did not significantly change, suggesting these additional parameters did not significantly impact predictions with random forests or gradient boosting approaches.

The best model (using MAE and RMSE, as before) remains the Boosted Gradient model.

Multi-class Classification

In many predictive modeling tasks, the goal is to classify observations into distinct categories based on their characteristics. When there are only two possible categories (e.g., "Yes" or "No," "Up" or "Down"), the problem is called binary classification. However, when there are **more than two possible categories**, we use **multi-class classification**.

An automobile company is expanding into new markets with its existing products. Market research shows that customer behavior in these markets mirrors the existing one.

Previously, the sales team classified customers into four segments (A, B, C, D) using rule-based methods based on demographics, spending behavior, and past purchases. To scale this approach, the company now requires a multi-class machine learning model to automate customer segmentation for new market entrants.

The A, B, C, D segmentation represents customer tiers in the automobile market:

- A (Loyal Customers) – High-spending, repeat buyers, likely to prefer premium or luxury cars.
- B (Regular Customers) – Moderate spenders, consistent buyers, often opting for mid-range vehicles.
- C (Occasional Customers) – Low-spending, infrequent buyers, likely considering budget-friendly or used cars.
- D (Potential Customers) – Least engaged, may be first-time buyers or exploring options with low commitment.

Your task is to develop a predictive solution that accurately assigns new customers to the appropriate segment, enabling data-driven decision-making and scalable outreach.

Dataset_overview:

Dataset Overview The dataset contains information about automobile customers. Each record represents an individual customer with various demographic and behavioral attributes. The key variables in the dataset include:

- **ID**: Unique identifier for each customer. Gender: Customer's gender.
- **Ever_Married**: Indicates if the customer has ever been married.
- **Age**: Customer's age.
- **Graduated**: Indicates whether the customer is a graduate.
- **Profession**: Customer's occupation

- **Profession**: Customer's occupation.
- **Work_Experience**: Number of years the customer has worked.
- **Spending_Score**: Indicates spending behavior, classified as Low, Average, or High based on purchasing patterns.
- **Family_Size**: Number of family members in the household.
- **Hidden_Customer_Category**: An internally assigned, anonymized classification of customers based on undisclosed criteria, such as spending behavior, or engagement patterns.

Discussion Question: Binary vs. Multi-Class Classification prior to predictive tasks.

If you were primarily interested in predicting who your potential customers are—perhaps so you can target them with incentives to encourage engagement—would you prefer a binary classification model (Potential Customer: 0/1) or a multi-class classification model that categorizes users into the four segments (A, B, C, D)? You could use ChatGPT to help you think these tradeoffs.

Answer:

I would prefer a binary classification model. We are trying to identify whether a person is a customer or not. This is outside the sccope of the 4-segment classification. A model might classify an unlikely customer into one of these 4 buckets without much confidence in the answer, since this person may not be interested at all.

We must first filter down the list of users to potential customers using a binary classification, and then we could perhaps look at what segment they belong to to identify what type of incentives (and to what extent) we should offer to these customers.

Task 1: Data Preprocessing and Exploration

1.1 Load and Explore the Dataset

- Load the **train and test datasets**.
- Display summary statistics using `.describe()` and `.info()`.

```
# change to your file directory
automobile_customers = pd.read_csv(
    "https://raw.githubusercontent.com/chansen776/MBA-ML-Course-Materials/refs/heads/main
)
```

```
automobile_customers.info()
automobile_customers.describe()
```

```
automobile_customers.isnull().sum()
automobile_customers.dropna(inplace=True)
```

Question: What are the two code lines above doing? (You could use your favorite LLM to help).

Answer:

The first line is looking for null values, and counting the number of null values in the dataframe. This serves as a quick check to ensure there are no invalid data in the table.

The second line serves to deal with any lines that contain any N/A values. If a row has N/A in any of the columns, it will drop the entire row of data within the dataframe (inplace) so that the automobile_customers dataframe is updated in place.

1.2 Visualizing Categorical Variables Using Bar Plots

- Use **bar plots** to visualize the distribution of categorical variables.
- Focus on key categorical features such as **Gender, Ever_Married, Graduated, Profession, Spending_Score, Hidden_Customer_Category, and Segmentation**.
- Bar plots help understand the frequency of each category and identify potential imbalances.

```
# Generate bar plots for each categorical column
categorical_cols = [
    "Gender",
    "Ever_Married",
    "Graduated",
    "Profession",
    "Spending_Score",
    "Hidden_Customer_Category",
    "Segmentation",
]

for col in categorical_cols:
    plt.figure(figsize=(6, 4))
    sns.countplot(
        x=automobile_customers[col],
        order=automobile_customers[col].value_counts().index,
    )
    plt.title(f"Bar Plot of {col}")
    plt.xticks(rotation=45)
    plt.show()
```

1.3 Discover Correlation with Other Variables

Investigate whether certain **demographic or behavioral attributes** are strong predictors of segmentation.

```
# Boxplots for Numerical Features vs. Segmentation
# Boxplots
numerical_cols = ["Age", "Work_Experience", "Family_Size"]

for col in numerical_cols:
```

```
for col in numerical_cols:
    plt.figure(figsize=(8, 5))
    sns.boxplot(x="Segmentation", y=col, data=automobile_customers, palette="coolwarm")
    plt.title(f"{col} Distribution Across Segmentation (Boxplot)")
    plt.xlabel("Segmentation")
    plt.ylabel(col)
    plt.show()
```

```
# Countplots for Categorical Features vs. Segmentation
# Countplots
categorical_cols = [
    "Gender",
    "Ever_Married",
    "Graduated",
    "Profession",
    "Spending_Score",
    "Hidden_Customer_Category",
]

for col in categorical_cols:
    plt.figure(figsize=(8, 5))
    sns.countplot(
        x=col, hue="Segmentation", data=automobile_customers, palette="coolwarm"
    )
    plt.title(f"Distribution of {col} Across Segmentation (Count Plot)")
    plt.xticks(rotation=45)
    plt.legend(title="Segmentation")
    plt.show()
```

Question: Analyze and Interpret Data Insights

1. Summarize your key insights from the dataset, focusing on patterns or trends that could influence modeling decisions for customer segmentation.
2. Here, we examine the full dataset before modeling. Is there a potential issue with analyzing the entire dataset before constructing models based on these insights? What risks might arise from this approach?

Answer:

1. There are patterns in the other data columns that can be useful in predicting the segmentation of the customer. For example, Work Experience is a great metric for distinguishing between Segment A and D. Or that customers who have never been married are not likely to belong in Segment A.

   But overall, very few of these parameters - on their own - serve as very good indicators of which segment a particular customer might fall within. While there are a few indicators like the ones discussed above, they must be used in conjunction with all the data to drive a better predictive segmentation model.

2. There are some scenarios where we have so much data that we simply cannot extract

insight by merely looking at the dataset, or we would be spending a lot of time doing so. Data may be clustered where interaction terms are highly predictive, for example. Analyzing the dataset in this way might mean that we miss this insight.

But more crucially, there may be cases where multiple factors are correlated with each other. So while they may all be good predictors of the outcome, including all of them could lead to over-representation and ignoring the confounding effects between these factors.

We may therefore prejudice ourselves and either think there are predictive terms where there really aren't or that there are no predictive ability here while there is.

1.4 Convert Categorical Variables and Encode Data

Encode categorical variables using **LabelEncoder** for all categorical columns, except the target column `Segmentation`. **LabelEncoder** will be applied to **all categorical variables**, including both binary and multi-class categories.

```
# List of categorical columns (excluding the target 'Segmentation')
categorical_cols = [
    "Gender",
    "Ever_Married",
    "Graduated",
    "Profession",
    "Spending_Score",
    "Hidden_Customer_Category",
]

# Initialize LabelEncoder
encoder = LabelEncoder()

# Loop through each categorical column and apply LabelEncoder
for col in categorical_cols:
    automobile_customers[col] = encoder.fit_transform(
        automobile_customers[col].astype(str)
    )
```

Task 2: Build & Compare Classification Models

For Task 2, you'll follow these steps:

- 2.1 Split the dataset into train and test sets.
- 2.2 Train 3 multi-class classification models covered in Note2:
    - Logistic Regression

    - Decision Tree Classifier
    - Random Forest Classifier
- 2.3 Evaluate the models using performance metrics such as accuracy, precision, recall, F1-score, and confusion matrix.

2.1 Split the Dataset into Train and Test Sets

- Use **`train_test_split`** to split the data into **80% training** and **20% testing** sets.
- Ensure the split is **randomized** to help the model generalize (set_seed= `42`).

```
# Separate features (X) and target (y)
X = automobile_customers.drop(columns=["Segmentation", "ID"])
y = automobile_customers["Segmentation"]

# Perform the train-test split (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print(f"Training set shape: {X_train.shape}")
print(f"Testing set shape: {X_test.shape}")
```

```
Training set shape: (6352, 9)
Testing set shape: (1589, 9)
```

Question: What did you notify about the `train_test_split` method? Why we use `stratify` parameter?

Answer: It takes the x- (the columns that we will be using to predict the segmentation) and the y- (the known segmentation itself) inputs and splits them into two buckets (the train and the test) with an 80/20 proportion, as given by the test_size of 0.2.

The random_state sets the seed for the random generator so this segmentation is repeatable. The stratify parameter ensures that equal proportions of each of the 4 segments (since y is passed as the argument value) is equally represented in the train and the test data. So if segment B has a 60% representation in the overall data, y_train and y_test would each have a 60% representation of segment B. This ensures that no bias is introduced in the training and test data when we carry out the split.

2.2 Train 3 Multi-Class Classification Models

1. Train the following models using the prepared dataset:
    - **Decision Tree Classifier**
    - **Random Forest Classifier**
    - **Gradient Boosting Classifier**
2. Since tree-based models do not require feature scaling, use the raw numerical features for training.
3. **Perform Grid Search Cross-Validation (GridSearchCV) with 5-Fold Cross-Validation (cv=5)** to optimize hyperparameters
4. Fit each model to the training dataset and generate predictions on the test dataset.
5. Evaluate the models using accuracy, precision, recall, and F1-score, and analyze the results.

```
# Define hyperparameter grid
param_grid_dt = {
    "max_depth": [10, 20, 30],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
}

# Perform Grid Search
dt_grid = GridSearchCV(
    DecisionTreeClassifier(random_state=42),
    param_grid_dt,
    cv=5,
    scoring="accuracy",
    n_jobs=-1,
    verbose=2,
)
dt_grid.fit(X_train, y_train)

# Get best model and evaluate
best_dt = dt_grid.best_estimator_
y_pred_dt = best_dt.predict(X_test)

print("\nDecision Tree Best Parameters:", dt_grid.best_params_)
print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))
print(
    "Decision Tree Classification Report:\n", classification_report(y_test, y_pred_dt)
)
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=5; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=4, min_samples_split=2; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=4, min_samples_split=2; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=2, min_samples_split=2; total time=   0.1s
```

```
[CV] END max_depth=10, min_samples_leaf=2, min_samples_split=2; total time=   0.1s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=5; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=4, min_samples_split=2; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=4, min_samples_split=5; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=4, min_samples_split=5; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=4, min_samples_split=5; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=4, min_samples_split=5; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=4, min_samples_split=5; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=5; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=4, min_samples_split=10; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=4, min_samples_split=10; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=2, min_samples_split=2; total time=   0.1s
[CV] END max_depth=10, min_samples_leaf=4, min_samples_split=10; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=4, min_samples_split=10; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=4, min_samples_split=10; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=5; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=5; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=5; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=5; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=5; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=5; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=2, min_samples_split=2; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=2, min_samples_split=2; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=2, min_samples_split=2; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=2, min_samples_split=2; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=2, min_samples_split=2; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=2, min_samples_split=5; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=2, min_samples_split=10; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=2, min_samples_split=2; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=2, min_samples_split=5; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=2, min_samples_split=5; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=2, min_samples_split=5; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=10; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=2, min_samples_split=5; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=2, min_samples_split=10; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=1, min_samples_split=2; total time=   0.0s
[CV] END max_depth=10, min_samples_leaf=2, min_samples_split=5; total time=   0.0s
[CV] END max_depth=20, min_samples_leaf=2, min_samples_split=10; total time=   0.0s
```

Question: Briefly explain what `GridSearchCV` is doing. Why we do cross-validation in the training dataset?

Answer:

It is running the Decision Tree Classifier model based on a grid of hyper parameters. These are parameters required by the Decision Tree Classifier, such as max_depth, min_samples_split, and min_samples_leaf.

- max_depth is the maximum levels that the decision tree can have. We evaluate values of 10, 20, and 30.
- min_samples_split is the smallest number of samples required to split a node in the decision tree. This serves as a crucial dial to balance between overfitting and model complexity. We evaluate values of 2, 5, and 10.
- min_samples_leaf is the smallest number of samples requires to be a leaf node. This also helps control the balance between overfitting and model complexity. We evaluate values of 1, 2, and 4.

In addition to running this, it performs a 5-fold cross validation, meaning it subsets the training data again into a train and test dataset 5 times (so that each datum shows up in the test data atleast once). This is used to validate the model so we can get out-of-sample predictions to evaluate these models.

We can then find the best predictor out of all the different models we fit (so now we have the model utilizing the best hyperparameters).

Question: Based on the classification report for decision tree, what do terms like precision, recall, F1-score mean? Briefly interpret the decision tree classifier results on the test dataset. (You could use ChatGPT to help interpretation.)

Answer:

Precision: This measures the proportion of correct positives (true positives) out of all predicted positives. It is the (True Positives) / (True Positives + False Positives).

Recall: This is the proportion of correct positives (true positives) out of all actual positives, even the ones the model predicted as negative. It is the (True Positives) / (True Positives + False Negatives).

F1-Score: This is a average of precision and recall, using the formula (2 * Precision * Recall) / (Precision + Recall). This effectively combines the two measures above and a gives a single score to combine both false positives and false negative error rates.

Support: This simply measures how many instances of each class exist in the dataset, and so

Support: This simply measures how many instances of each class exist in the dataset, and so gives a measure of how much data exists to *support* the confidence in the other measures given above.

Training code for random forests with grid search cross-validation (would take around 5 minutes to run)

```
# Random Forests
# Define hyperparameter grid
param_grid_rf = {
    "n_estimators": [500],
    "max_depth": [10, 20, 30],
    "min_samples_split": [2, 5, 10],
}

# Perform Grid Search
rf_grid = GridSearchCV(
    RandomForestClassifier(random_state=42),
    param_grid_rf,
    cv=5,
    scoring="accuracy",
    n_jobs=-1,
    verbose=2,
)
rf_grid.fit(X_train, y_train)

# Get best model and evaluate
best_rf = rf_grid.best_estimator_
y_pred_rf = best_rf.predict(X_test)

print("\nRandom Forest Best Parameters:", rf_grid.best_params_)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print(
    "Random Forest Classification Report:\n", classification_report(y_test, y_pred_rf)
)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV] END max_depth=10, min_samples_split=10, n_estimators=500; total time=   5.9s
[CV] END max_depth=10, min_samples_split=10, n_estimators=500; total time=   6.0s
[CV] END max_depth=10, min_samples_split=10, n_estimators=500; total time=   6.3s
[CV] END max_depth=10, min_samples_split=10, n_estimators=500; total time=   6.3s
[CV] END max_depth=10, min_samples_split=10, n_estimators=500; total time=   6.4s
[CV] END max_depth=10, min_samples_split=5, n_estimators=500; total time=   7.0s
[CV] END max_depth=10, min_samples_split=5, n_estimators=500; total time=   7.0s
[CV] END max_depth=10, min_samples_split=2, n_estimators=500; total time=   7.1s
[CV] END max_depth=10, min_samples_split=5, n_estimators=500; total time=   7.1s
[CV] END max_depth=10, min_samples_split=5, n_estimators=500; total time=   7.2s
[CV] END max_depth=10, min_samples_split=5, n_estimators=500; total time=   7.2s
[CV] END max_depth=10, min_samples_split=2, n_estimators=500; total time=   7.3s
[CV] END max_depth=10, min_samples_split=2, n_estimators=500; total time=   7.4s
```

```
[CV] END max_depth=10, min_samples_split=2, n_estimators=500; total time=   7.4s
[CV] END max_depth=10, min_samples_split=2, n_estimators=500; total time=   7.5s
[CV] END max_depth=20, min_samples_split=10, n_estimators=500; total time=   9.6s
[CV] END max_depth=20, min_samples_split=10, n_estimators=500; total time=   9.7s
[CV] END max_depth=20, min_samples_split=10, n_estimators=500; total time=  10.0s
[CV] END max_depth=20, min_samples_split=10, n_estimators=500; total time=  10.1s
[CV] END max_depth=20, min_samples_split=5, n_estimators=500; total time=  10.2s
[CV] END max_depth=20, min_samples_split=10, n_estimators=500; total time=  10.2s
[CV] END max_depth=20, min_samples_split=5, n_estimators=500; total time=  10.7s
[CV] END max_depth=20, min_samples_split=5, n_estimators=500; total time=  10.8s
[CV] END max_depth=20, min_samples_split=5, n_estimators=500; total time=  10.8s
[CV] END max_depth=20, min_samples_split=5, n_estimators=500; total time=  10.9s
[CV] END max_depth=20, min_samples_split=2, n_estimators=500; total time=  11.8s
[CV] END max_depth=20, min_samples_split=2, n_estimators=500; total time=  11.8s
[CV] END max_depth=30, min_samples_split=10, n_estimators=500; total time=   4.6s
[CV] END max_depth=20, min_samples_split=2, n_estimators=500; total time=  11.9s
[CV] END max_depth=20, min_samples_split=2, n_estimators=500; total time=  12.0s
[CV] END max_depth=20, min_samples_split=2, n_estimators=500; total time=  12.1s
[CV] END max_depth=30, min_samples_split=2, n_estimators=500; total time=  11.9s
[CV] END max_depth=30, min_samples_split=5, n_estimators=500; total time=   5.2s
[CV] END max_depth=30, min_samples_split=2, n_estimators=500; total time=  12.1s
[CV] END max_depth=30, min_samples_split=5, n_estimators=500; total time=   5.9s
[CV] END max_depth=30, min_samples_split=5, n_estimators=500; total time=   5.8s
[CV] END max_depth=30, min_samples_split=10, n_estimators=500; total time=   5.1s
[CV] END max_depth=30, min_samples_split=10, n_estimators=500; total time=   5.3s
[CV] END max_depth=30, min_samples_split=10, n_estimators=500; total time=   5.2s
[CV] END max_depth=30, min_samples_split=5, n_estimators=500; total time=   5.7s
[CV] END max_depth=30, min_samples_split=5, n_estimators=500; total time=   5.6s
[CV] END max_depth=30, min_samples_split=2, n_estimators=500; total time=   6.6s
[CV] END max_depth=30, min_samples_split=10, n_estimators=500; total time=   5.5s
[CV] END max_depth=30, min_samples_split=2, n_estimators=500; total time=   6.8s
[CV] END max_depth=30, min_samples_split=2, n_estimators=500; total time=   6.5s

Random Forest Best Parameters: {'max_depth': 30, 'min_samples_split': 10, 'n_estimato
Random Forest Accuracy: 0.775959723096287
Random Forest Classification Report:
               precision    recall  f1-score   support

           A       0.75      0.76      0.76       193
           B       0.81      0.93      0.87       721
           C       0.73      0.66      0.69       438
           D       0.73      0.54      0.62       237

    accuracy                           0.78      1589
```

Considering time cost, training code for gradient boosting is **without grid search cross-validation**.

```
# Pre-Selected GBM Hyperparameters, considering time cost
best_gb = GradientBoostingClassifier(
    n_estimators=200, max_depth=7, learning_rate=0.1, random_state=42
)
```

```
# Train the model
best_gb.fit(X_train, y_train)

# Make predictions
y_pred_gb = best_gb.predict(X_test)

# Evaluate performance
print("\nGradient Boosting Accuracy:", accuracy_score(y_test, y_pred_gb))
print(
    "Gradient Boosting Classification Report:\n",
    classification_report(y_test, y_pred_gb),
)
```

```
Gradient Boosting Accuracy: 0.775330396475771
Gradient Boosting Classification Report:
               precision    recall  f1-score   support

           A       0.79      0.79      0.79       193
           B       0.85      0.92      0.88       721
           C       0.70      0.66      0.68       438
           D       0.64      0.54      0.59       237

    accuracy                           0.78      1589
   macro avg       0.74      0.73      0.73      1589
weighted avg       0.77      0.78      0.77      1589
```

Confusion matrix for three tree models

```
for model, name in zip(
    [best_dt, best_rf, best_gb], ["Decision Tree", "Random Forest", "Gradient Boosting"]
):
    cm = confusion_matrix(y_test, model.predict(X_test))
    plt.figure(figsize=(6, 6))
    sns.heatmap(
        cm,
        annot=True,
        fmt="d",
        cmap="Blues",
        xticklabels=model.classes_,
        yticklabels=model.classes_,
    )
    plt.title(f"Confusion Matrix for {name}")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()
```

2.3 Model Explainability with SHAP & AutoML using FLAML

SHAP Value Analysis

```
# Sample a small subset of X_test
small_X_test = X_test.sample(n=100, random_state=42)

# Explain the Random Forest Model
explainer_rf = shap.TreeExplainer(best_rf)

shap_values_rf = explainer_rf.shap_values(small_X_test)
shap_values_mean = np.mean(np.abs(shap_values_rf), axis=2)

print("SHAP Summary Plot (Mean Across All Classes)")
print("SHAP Summary Plot (Mean Across All Classes)")
shap.summary_plot(
    shap_values_mean, small_X_test, plot_type="bar", feature_names=small_X_test.columns
)
```

Question: Briefly explain the most influential predictors for customer segmentation, based on SHAP value.

Answer:

The most influential predictor is the work_experience of the customer. As shown in the early exploratory graphs, work experience serves as an excellent distinguisher between segments A and D and serves decently to separate out segment B pretty decently purely on a visual basis.

As shown by the SHAP value, this has the greatest indicator on the customer segmentation.

FLAML AutoML for Multi-Class Classification

Train models using **FLAML AutoML**, an efficient and lightweight AutoML framework.

```
# Initialize FLAML AutoML
automl = AutoML()

# Train AutoML with time limit
automl.fit(X_train, y_train, task="classification", time_budget=300)

# Get predictions from the best model
y_pred_flaml = automl.predict(X_test)

# Evaluate model performance
print("FLAML AutoML Accuracy:", accuracy_score(y_test, y_pred_flaml))
print(
    "FLAML AutoML Classification Report:\n", classification_report(y_test, y_pred_flaml)
)

# Print the best model selected by FLAML
print("\nFLAML AutoML Best Model:", automl.model.estimator)
```

```
[flaml.automl.logger: 03-06 14:50:44] {1728} INFO - task = classification
[flaml.automl.logger: 03-06 14:50:44] {1739} INFO - Evaluation method: cv
[flaml.automl.logger: 03-06 14:50:44] {1838} INFO - Minimizing error metric: log_loss
[flaml.automl.logger: 03-06 14:50:44] {1955} INFO - List of ML learners in AutoML Run
[flaml.automl.logger: 03-06 14:50:44] {2258} INFO - iteration 0, current learner lgbm
[flaml.automl.logger: 03-06 14:50:44] {2393} INFO - Estimated sufficient time budget=
[flaml.automl.logger: 03-06 14:50:44] {2442} INFO -  at 0.1s,   estimator lgbm's best
[flaml.automl.logger: 03-06 14:50:44] {2258} INFO - iteration 1, current learner lgbm
[flaml.automl.logger: 03-06 14:50:44] {2442} INFO -  at 0.2s,   estimator lgbm's best
[flaml.automl.logger: 03-06 14:50:44] {2258} INFO - iteration 2, current learner lgbm
[flaml.automl.logger: 03-06 14:50:44] {2442} INFO -  at 0.3s,   estimator lgbm's best
[flaml.automl.logger: 03-06 14:50:44] {2258} INFO - iteration 3, current learner sgd
[flaml.automl.logger: 03-06 14:50:45] {2442} INFO -  at 1.1s,   estimator sgd's best
[flaml.automl.logger: 03-06 14:50:45] {2258} INFO - iteration 4, current learner lgbm
[flaml.automl.logger: 03-06 14:50:45] {2258} INFO - iteration 4, current learner lgbm
[flaml.automl.logger: 03-06 14:50:45] {2442} INFO -  at 1.3s,   estimator lgbm's best
[flaml.automl.logger: 03-06 14:50:45] {2258} INFO - iteration 5, current learner xgbo
[flaml.automl.logger: 03-06 14:50:45] {2442} INFO -  at 1.5s,   estimator xgboost's b
[flaml.automl.logger: 03-06 14:50:45] {2258} INFO - iteration 6, current learner lgbm
[flaml.automl.logger: 03-06 14:50:46] {2442} INFO -  at 1.7s,   estimator lgbm's best
[flaml.automl.logger: 03-06 14:50:46] {2258} INFO - iteration 7, current learner lgbm
[flaml.automl.logger: 03-06 14:50:46] {2442} INFO -  at 1.8s,   estimator lgbm's best
[flaml.automl.logger: 03-06 14:50:46] {2258} INFO - iteration 8, current learner lgbm
[flaml.automl.logger: 03-06 14:50:46] {2442} INFO -  at 1.9s,   estimator lgbm's best
[flaml.automl.logger: 03-06 14:50:46] {2258} INFO - iteration 9, current learner lgbm
[flaml.automl.logger: 03-06 14:50:46] {2442} INFO -  at 2.3s,   estimator lgbm's best
[flaml.automl.logger: 03-06 14:50:46] {2258} INFO - iteration 10, current learner xgb
[flaml.automl.logger: 03-06 14:50:46] {2442} INFO -  at 2.5s,   estimator xgboost's b
[flaml.automl.logger: 03-06 14:50:46] {2258} INFO - iteration 11, current learner ext
[flaml.automl.logger: 03-06 14:50:47] {2442} INFO -  at 2.7s,   estimator extra_tree'
[flaml.automl.logger: 03-06 14:50:47] {2258} INFO - iteration 12, current learner rf
[flaml.automl.logger: 03-06 14:50:47] {2442} INFO -  at 3.0s,   estimator rf's best e
[flaml.automl.logger: 03-06 14:50:47] {2258} INFO - iteration 13, current learner rf
[flaml.automl.logger: 03-06 14:50:47] {2442} INFO -  at 3.2s,   estimator rf's best e
```

```
[flaml.automl.logger: 03-06 14:50:47] {2258} INFO - iteration 14, current learner ext
[flaml.automl.logger: 03-06 14:50:47] {2442} INFO -  at 3.4s,   estimator extra_tree'
[flaml.automl.logger: 03-06 14:50:47] {2258} INFO - iteration 15, current learner xgb
[flaml.automl.logger: 03-06 14:50:48] {2442} INFO -  at 3.8s,   estimator xgboost's b
[flaml.automl.logger: 03-06 14:50:48] {2258} INFO - iteration 16, current learner ext
[flaml.automl.logger: 03-06 14:50:48] {2442} INFO -  at 4.0s,   estimator extra_tree'
[flaml.automl.logger: 03-06 14:50:48] {2258} INFO - iteration 17, current learner lgb
[flaml.automl.logger: 03-06 14:50:48] {2442} INFO -  at 4.3s,   estimator lgbm's best
[flaml.automl.logger: 03-06 14:50:48] {2258} INFO - iteration 18, current learner sgd
[flaml.automl.logger: 03-06 14:50:49] {2442} INFO -  at 5.7s,   estimator sgd's best 
[flaml.automl.logger: 03-06 14:50:49] {2258} INFO - iteration 19, current learner lgb
[flaml.automl.logger: 03-06 14:50:52] {2442} INFO -  at 8.1s,   estimator lgbm's best
[flaml.automl.logger: 03-06 14:50:52] {2258} INFO - iteration 20, current learner xgb
[flaml.automl.logger: 03-06 14:50:52] {2442} INFO -  at 8.2s,   estimator xgboost's b
[flaml.automl.logger: 03-06 14:50:52] {2258} INFO - iteration 21, current learner xgb
[flaml.automl.logger: 03-06 14:50:52] {2442} INFO -  at 8.3s,   estimator xgboost's b
[flaml.automl.logger: 03-06 14:50:52] {2258} INFO - iteration 22, current learner xgb
[flaml.automl.logger: 03-06 14:50:52] {2442} INFO -  at 8.4s,   estimator xgboost's b
[flaml.automl.logger: 03-06 14:50:52] {2258} INFO - iteration 23, current learner rf 
[flaml.automl.logger: 03-06 14:50:52] {2442} INFO -  at 8.7s,   estimator rf's best e
[flaml.automl.logger: 03-06 14:50:52] {2258} INFO - iteration 24, current learner rf 
[flaml.automl.logger: 03-06 14:50:53] {2442} INFO -  at 9.0s,   estimator rf's best e
[flaml.automl.logger: 03-06 14:50:53] {2258} INFO - iteration 25, current learner xgb
[flaml.automl.logger: 03-06 14:50:53] {2442} INFO -  at 9.2s,   estimator xgboost's b
```

Question: Did FLAML AutoML provide better performance compared to manually tuned models? Briefly explain.

Answer: If we use the accuracy (the number of correct predictions) / (total number of predictions) as the singular metric, we see that the machine learning model has an accuracy of 0.80, which is better than 0.77-0.78 accuracy values of the manually tuned models.

Even in the individual class f1-scores, the machine learning model outperforms the manually trained models. This is done since both the above metrics could be skewed by an unbalanced support between the classes (a huge preponderance of one class being predicted well that overshadows errors in the minority class predictions). Here, all 4 classes appear to have a f1-score, with the machine learning at par or better for each class.

Therefore, we can conclude that the AutoML provided better performance compared to manually tuned models.

Short-Answer Questions

- Question 1 : You are part of the marketing team at a subscription-based service (e.g., streaming platform). The goal is to predict whether a user will subscribe to a premium plan based on their usage patterns and demographics. You're using a machine learning model to predict subscription likelihood (Class 1: Subscribe, Class 0: No Subscribe).

How would you prioritize precision or recall for a subscription service when targeting users for a premium plan?

Here, we would like to get an idea if a user will sign-up for the premium plan, so that we can perhaps offer them a free trial to entice them to sign-up. The cost of a false-positive (labelling them as a potential customer while they are not) is just the cost of a trial and is therefore quite low. The cost of a false-negative (labelling them as a non-customer and not reaching out to them) is potentially losing the revenue from their subscription.

As such, our cost of false-negatives is high, and so we want to minimize that, making Recall the more important metric.

- Question 2: You are working for an e-commerce company and have built a classification model to segment customers into high, medium, and low-value groups based on their likelihood of making future purchases (Class 1: High-Value, Class 2: Medium-Value, Class 3: Low-Value).

Is it likely that data imbalance issues will occur, and what precautions should be taken when interpreting the results?

Yes, data imbalance issues will occur since there are going to be many more low-value customers than high-value customers.

We should be careful with accuracy (or weighted average statistics) since they can be skewed by predicting the majority class well. We should look deeper into the individual class statistics of precision, recall, and f1-scores.

We should also verify if the predictions make sense by comparing it to expert opinion (or a gut-check) and evaluating whether we care more about false-positives or false-negatives in the predictions.

We can adjust the training data as well by over- or under- sampling to adjust this imbalance.

- Question 3: Dynamic Pricing Model for Airline Industry You work at an airline, and the company uses a model to predict whether a customer will purchase a flight ticket at a given price (Class 1: Purchase, Class 0: No Purchase). The price changes dynamically based on factors like demand, booking time, and competitor pricing.

How would you adjust your model's decision threshold to optimize for maximum profit in this dynamic pricing scenario? What considerations should you take into account when setting this threshold?

The objective would be to maximize revenues from the sales, and as such we would need to evaluate whether we are underselling the available seats, or over-selling them (by charging too low a price).

We can modify the threshold depending on whether the demand is high (allowing us to increase prices and be more discerning of taking on a customer), and how close we are to the flight (what is the opportunity cost of selling the ticket now vs waiting for a better customer who would be willing to pay more).

- Question 4: You work for an online payment company and are tasked with building a model to identify fraudulent transactions (Class 1: Fraud, Class 0: Legitimate). The dataset contains transaction amounts, user behavior, and historical fraud data.

How would you use model explainability tools like SHAP or partial dependence plots to communicate the rationale behind the model's decisions to stakeholders who may not be familiar with machine learning?

Fraud detection is a sensitive subject for payments, and consistent inaccurate predictions can cause customer dissatisfaction with the payment company, and therefore there is a high cost associated with both false positives (customer dissatisfaction) and false negatives (fraudulent purchases).

While machine learning tools may be significantly better than simple classification trees, they suffer from being a black-box approach with limited insight into the factors that they are using to determine whether a transaction is fraudulent or not.

SHAP serves to help prod this black box so we can atleast identify what are they key factors it is considering when classifying a transaction. These can then be investigated and checked with experts to evaluate whether the model is working on a reasonable basis. This also helps assuage the experts' fears that the model is going off of totally irrelevant data or is otherwise hallucinating.

Question 5: Supervised learning is widely used in business applications to make data-driven decisions.

Task:

1. Identify a real-world business case where supervised learning can be applied. Define the outcome variable (target variable) for this problem.
2. Explain how you would evaluate the model's performance: Which evaluation metrics (e.g., Accuracy, Precision, Recall, F1-score, RMSE, AUC-ROC) would you choose?

3. Justify why those metrics are the most appropriate for your chosen business case.

💡 Hint: Consider cases like customer churn prediction, fraud detection, loan default prediction, or sales forecasting. Think about the cost of errors and which metric best captures the business objective.

Answer

We can use supervised learning for identifying species of weeds based on images. This allows us to build robots that can identify a specific plant as a crop or a weed, and therefore target potent herbicides selectively on the weeds.

False-positives (identifying a crop as a weed) results in a crop getting sprayed by powerful herbicies which may kill that specific plant, while false-negatives (ignoring a weed) results in that weed being given the chance to continue growing. It would depend on the type of weed, but generally a farmer would be more willing to accept the possible loss of a single plant over a weed spreading and risking his field, and so the cost of a false-negative is generally higher. We should therefore prioritize Recall as the more important metric.