# Introduction to Business Data Analytics

**Homework #4 Part 2**

Seth Abayo, Dola Qiu, Lanston Chen, Yizhou Sun, Yihua Wang

(put your names above (incl. any nicknames))

Note: This is a team homework assignment. Discussing this homework with your classmates outside your MSBA team is a **violation** of the Honor Code. If you **borrow code** from somewhere else, please add a comment in your code to **make it clear** what the source of the code is (e.g., a URL would sufficient). If you borrow code and you don't provide the source, it is a violation of the Honor Code.

Total grade: ________ out of ___145___ points

**(145 points) Use numeric prediction techniques to build a predictive model for the HW4.xlsx dataset. This dataset is provided on Canvas and contains data about whether or not different consumers made a purchase in response to a test mailing of a certain catalog and, in case of a purchase, how much money each consumer spent. The data file has a brief description of all the attributes in a separate worksheet. We would like to build predictive models to predict how much will the customers spend; Spending is the target variable (numeric value: amount spent).**
**Use Python for this exercise.**
**<u>Whenever applicable use random state 42 (10 points).</u>**

(a) **(50 points) After exploring the data, build numeric prediction models that predict Spending. Use linear regression, k-NN, and regression tree techniques. Briefly discuss the models you have built. Use cross-validation with 10 folds to estimate the generalization performance. Present the results for each of the three techniques and discuss which one yields the best performance.**

[part a is worth 50 points in total:
15 points for exploring the data (i.e., descriptive statistics including min max mean and stdv, visualizations, target variable distribution)
10 points for correctly building linear regression model - provide screenshots and explain what you are doing and the corresponding results
10 points for correctly building k-NN model - provide screenshots and explain what you are doing and the corresponding results
10 points for correctly building regression tree model - provide screenshots and explain what you are doing and the corresponding results
5 points for discussing which of the three models yields the best performance]

**Import libraries**

```
[ ] # Imports

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

    # Sklearn imports
    from sklearn.linear_model import LinearRegression, Lasso, Ridge
    from sklearn.neighbors import KNeighborsRegressor
    from sklearn.tree import DecisionTreeRegressor
    from sklearn.model_selection import cross_val_score, GridSearchCV
    from sklearn.pipeline import Pipeline
    from sklearn.preprocessing import StandardScaler
```

**Import dataset from file loaded in drive**

mount the drive

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

read data

```
[ ] data = pd.read_excel('/content/drive/MyDrive/Emory/HW4.xlsx')

    # check dimension
    print(data.shape)

    data.head()
```

(2000, 25)

| | sequence_number | US | source_a | source_c | source_b | source_d | source_e | source_m | source_o | source_h | ... | source_x | source_w | Freq | last_update_days_ago | 1st_update_days_ago | Web order | Gender=male | Address_is_res | Purchase | Spending |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 2 | 3662 | 3662 | 1 | 0 | 1 | 1 | 127.87 |
| **1** | 2 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 2900 | 2900 | 1 | 1 | 0 | 0 | 0.00 |
| **2** | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 2 | 3883 | 3914 | 0 | 0 | 0 | 1 | 127.48 |
| **3** | 4 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 1 | 829 | 829 | 0 | 1 | 0 | 0 | 0.00 |
| **4** | 5 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 1 | 869 | 869 | 0 | 0 | 0 | 0 | 0.00 |

5 rows × 25 columns

## Drop duplicate and null values from the data

### Check and drop duplicates in any

```
[ ] data.drop_duplicates()
    print(data.shape)
```

(2000, 25)

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 25 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   sequence_number       2000 non-null   int64
 1   US                    2000 non-null   int64
 2   source_a              2000 non-null   int64
 3   source_c              2000 non-null   int64
 4   source_b              2000 non-null   int64
 5   source_d              2000 non-null   int64
 6   source_e              2000 non-null   int64
 7   source_m              2000 non-null   int64
 8   source_o              2000 non-null   int64
 9   source_h              2000 non-null   int64
 10  source_r              2000 non-null   int64
 11  source_s              2000 non-null   int64
 12  source_t              2000 non-null   int64
 13  source_u              2000 non-null   int64
 14  source_p              2000 non-null   int64
 15  source_x              2000 non-null   int64
 16  source_w              2000 non-null   int64
 17  Freq                  2000 non-null   int64
 18  last_update_days_ago  2000 non-null   int64
 19  1st_update_days_ago   2000 non-null   int64
 20  Web order             2000 non-null   int64
 21  Gender=male           2000 non-null   int64
 22  Address_is_res        2000 non-null   int64
 23  Purchase              2000 non-null   int64
 24  Spending              2000 non-null   float64
```

By checking the data info, we can see that there does not include any null or duplicate values inside the dataset

## Data Exploration

- Check statistical summary of the data

```
[ ] data.describe().iloc[1:]
```

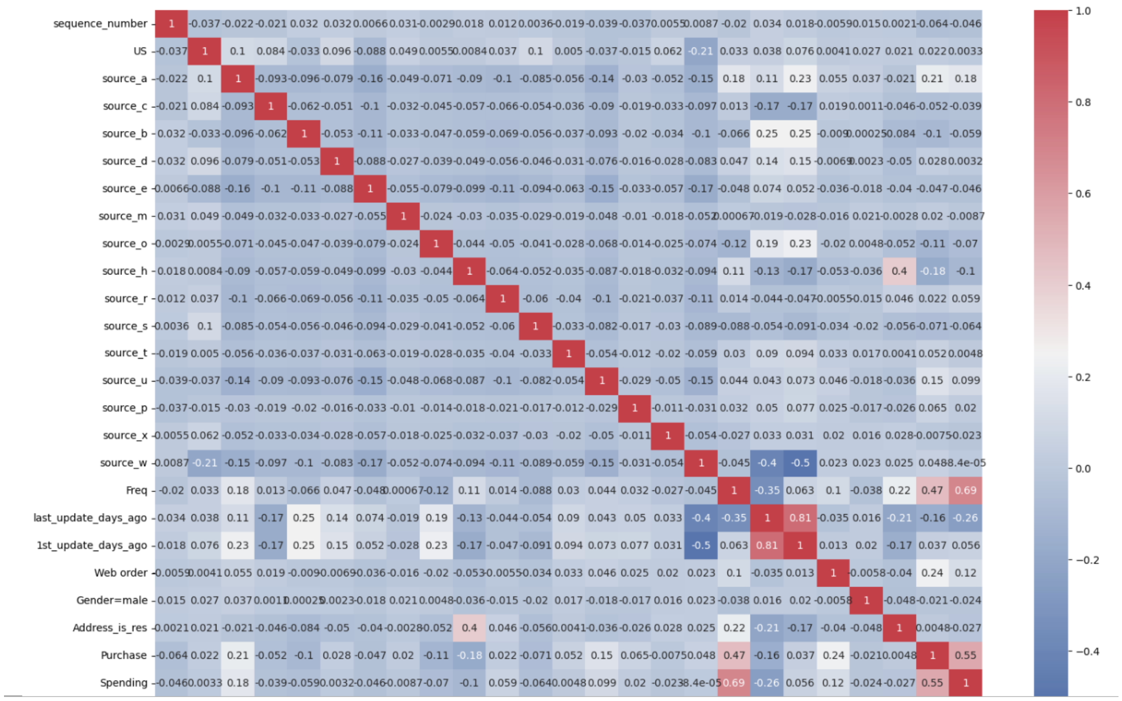
| | sequence_number | US | source_a | source_c | source_b | source_d | source_e | source_m | source_o | source_h | ... | source_x | source_w | Freq | last_update_days_ago | 1st_update_days_ago | Web order | Gender=male | Address_is_res |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **mean** | 1000.500000 | 0.824500 | 0.126500 | 0.056000 | 0.060000 | 0.041500 | 0.151000 | 0.01650 | 0.033500 | 0.052500 | ... | 0.018000 | 0.137500 | 1.417000 | 2155.101000 | 2435.601500 | 0.426000 | 0.524500 | 0.22100 |
| **std** | 577.494589 | 0.380489 | 0.332495 | 0.229979 | 0.237546 | 0.199493 | 0.358138 | 0.12742 | 0.179983 | 0.223089 | ... | 0.132984 | 0.344461 | 1.405738 | 1141.302846 | 1077.872233 | 0.494617 | 0.499524 | 0.41502 |
| **min** | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.00000 | 0.000000 | 0.000000 | ... | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 1.000000 | 0.000000 | 0.000000 | 0.00000 |
| **25%** | 500.750000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.00000 | 0.000000 | 0.000000 | ... | 0.000000 | 0.000000 | 1.000000 | 1133.000000 | 1671.250000 | 0.000000 | 0.000000 | 0.00000 |
| **50%** | 1000.500000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.00000 | 0.000000 | 0.000000 | ... | 0.000000 | 0.000000 | 1.000000 | 2280.000000 | 2721.000000 | 0.000000 | 1.000000 | 0.00000 |
| **75%** | 1500.250000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.00000 | 0.000000 | 0.000000 | ... | 0.000000 | 0.000000 | 2.000000 | 3139.250000 | 3353.000000 | 1.000000 | 1.000000 | 0.00000 |
| **max** | 2000.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.00000 | 1.000000 | 1.000000 | ... | 1.000000 | 1.000000 | 15.000000 | 4188.000000 | 4188.000000 | 1.000000 | 1.000000 | 1.00000 |

7 rows × 25 columns

## Create the correlated coefficient table to observe the relationship

```
[ ] f, ax = plt.subplots(figsize=(18, 12))
    corr = data.corr()
    sns.heatmap(corr, annot=True, annot_kws={"size": 10}, mask=np.zeros_like(corr, dtype=np.bool),
                cmap=sns.diverging_palette(240, 10, as_cmap=True), ax=ax)


    plt.show()
```



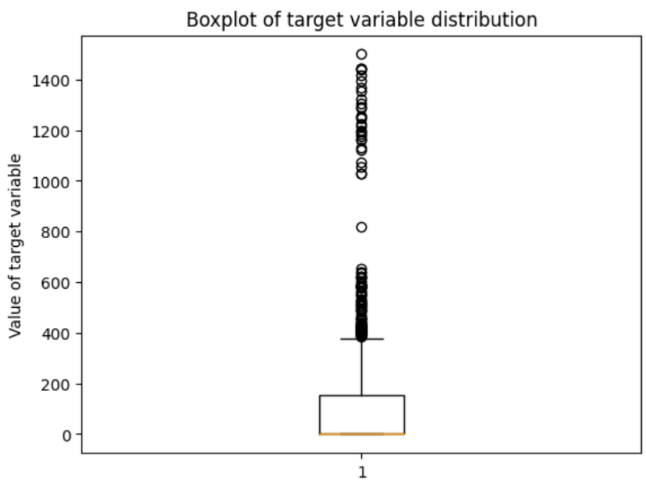

From overbearing the preceding table we created, we would get the following insight to help us to do data analysis:

1. Address_is_res and source_h have a very high positive correlation, suggesting customers with residential addresses are highly likely to be sourced from source_h.
2. Freq and Spending have a very high positive correlation, indicating that customers with a higher number of transactions in the last year are likely to have spent more in test mailing.
3. last_update_days_ago and 1st_update_days_ago are also highly positively correlated, suggesting that records that were created a long time ago tend to also have their last update a long time ago.
4. 1st_update_days_ago and source_w, and last_update_days_ago and source_w have a high negative correlation, meaning source_w is likely newer customers or those with more recent updates.
5. last_update_days_ago and Freq have a negative correlation, suggesting that more frequently transacting customers have more recent updates.
6. Address_is_res and Freq have a positive correlation, implying that the customers with residential addresses are likely to have a higher number of transactions.

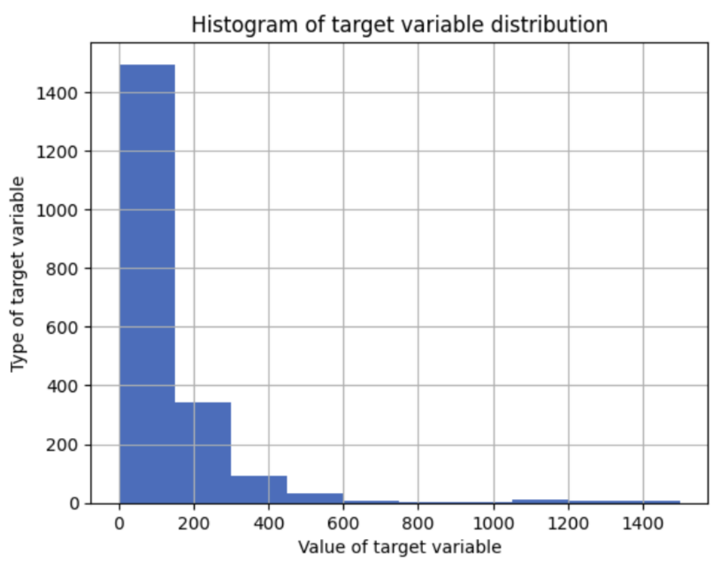
7. Address_is_res and last_update_days_ago have a negative correlation, indicating that customers with residential addresses tend to have more recent updates.
8. US and source_w have a high negative correlation, suggesting that source_w likely contains non-US addresses.
9. source_a and Purchase, and source_a and Spending are positively correlated, indicating that individuals sourced from source_a are more likely to make purchases and spend more in test mailings.

## Generate the plot for target variable distribution

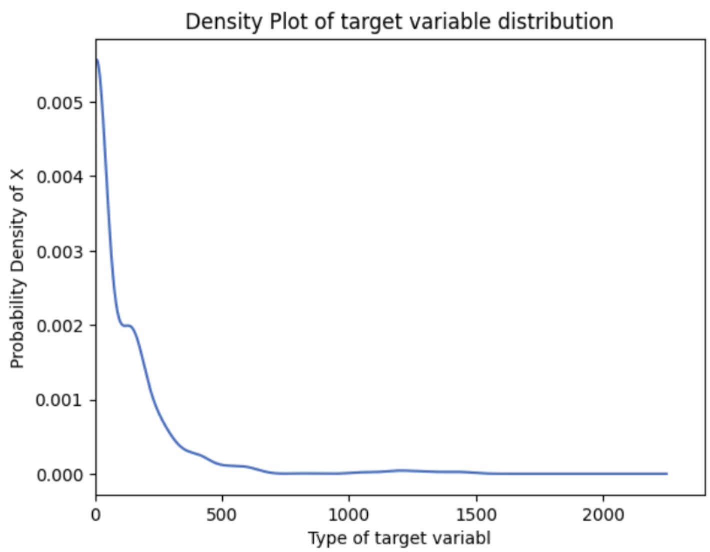
```
plt.boxplot(target)
plt.ylabel("Value of target variable")
plt.title('Boxplot of target variable distribution')
plt.show()
```



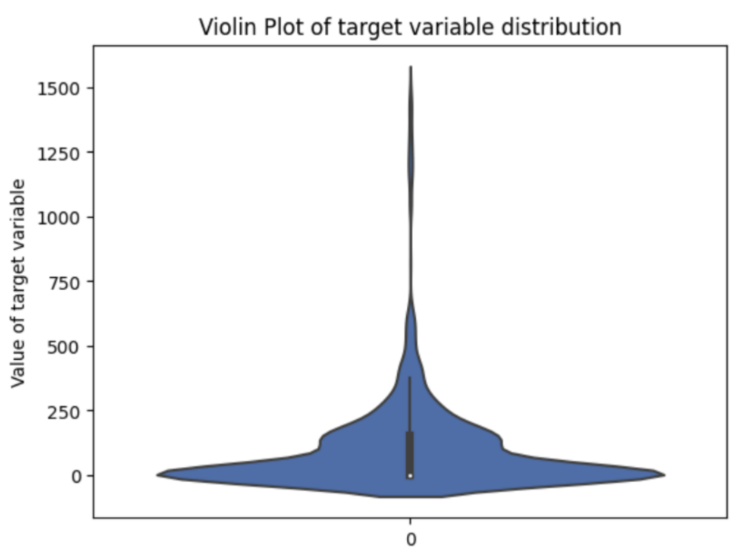

```
target.hist()
plt.ylabel("Type of target variable")
plt.xlabel("Value of target variable")
plt.title('Histogram of target variable distribution')
plt.show()
```



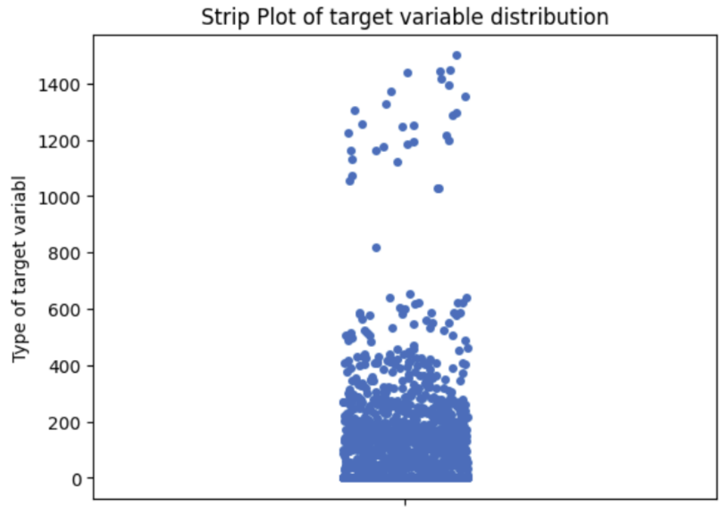
```python
target.plot(kind='density')
plt.xlabel("Type of target variabl")
plt.xlim(0,None)
plt.ylabel("Probability Density of X")
plt.title('Density Plot of target variable distribution')
plt.show()
```




```python
sns.violinplot(target)
plt.ylabel("Value of target variable")
plt.title('Violin Plot of target variable distribution')
plt.show()
```



```
[ ] sns.stripplot(target)
    plt.ylabel("Type of target variabl")
    plt.title('Strip Plot of target variable distribution')
    plt.show()
```




To better understand the distribution for the target variable which is spending, we created various plot including boxplot, histogram, density plot, violin plot, and strip plot. From observing these plots, it is obvious that the distribution for Spending is highly right-skewed. Hence, for the measuring factors MSE and RMSE which would be sensitive to outliers, we believe Mean Absolute Error would be a better measurement for giving linear penalty to the errors to make it less affected by the extreme values.

**Split data by dropping the target variable from the dataset**

# Split data

```
[ ] X = data.drop(columns=['Spending'], axis=1)
    y = data['Spending']
```

**Build numeric prediction for target variable**

```python
from os import ST_NODEV
# Define models

# Linear Regression
lr = LinearRegression()

# kNN Classifier
knn_reg = Pipeline([
                ('sc', StandardScaler()),
                ('knn', KNeighborsRegressor(n_neighbors=5))
            ])

# Decision tree
tree_reg = DecisionTreeRegressor(random_state=42)
```

We built these three models separately to gain results for the target variable. In order to standardize the scale of different variables, we utilized the StandardScaler in the Pipeline for kNN regression. By setting a random_state, we ensured that the results are reproducible. When running the code multiple times with random_state = 42, we will get the same tree structure every time, provided the input data and parameters remain unchanged for decision tree.

**Model Evaluation**

```python
result_a = pd.DataFrame({
        'Model': ['Linear Regression','k-NN Regression','Regression Tree'],
        'MAE': [np.round(-np.mean(lr_scores),2),np.round(-np.mean(knn_scores),2),np.round(-np.mean(tree_scores),2)],
        'σ (sigma)': ['+/-'+str(round(lr_std,2)), '+/-'+str(round(knn_std,2)), '+/-'+str(round(tree_std,2))]
})

result_a
```

| | Model | MAE | σ (sigma) |
|---|---|---|---|
| 0 | Linear Regression | 70.50 | +/-6.72 |
| 1 | k-NN Regression | 59.16 | +/-9.18 |
| 2 | Regression Tree | 66.74 | +/-10.69 |

By what we mentioned before, we input the calculation method from MAE to evaluate the performance of our models. From the result, we should judge the performance of the model from both their MAE and σ. The k-NN Regression model yields the lowest MAE, making it the best performer in terms of average prediction accuracy. However, it has a slightly higher σ than Linear Regression, indicating a bit more variability across folds. Despite the variability, the significant reduction in MAE (over 10 units lower than Linear Regression and 7 units lower than the Regression Tree) makes the k-NN model more compelling. Thus, considering the balance between performance (MAE) and consistency (σ), the k-NN Regression is the best choice based on the provided results.

**(b) (50 points) Engage in feature engineering (i.e., create new features based on existing features) to optimize the performance of linear regression, k-NN, and regression tree techniques. Present the results for each of the three techniques (choose the best performing model for each technique in case you try multiple models) and discuss which of the three yields the best performance. Use cross-validation with 10 folds to estimate the generalization performance. Discuss whether and why the generalization performance was improved or not.**

[part a is worth 50 points in total:
10 points for correctly building the new linear regression model and improving the performance as much as possible - provide screenshots and explain what you are doing and the corresponding results
10 points for correctly building the new k-NN model and improving the performance as much as possible - provide screenshots and explain what you are doing and the corresponding results
10 points for correctly building the new regression tree model and improving the performance as much as possible - provide screenshots and explain what you are doing and the corresponding results
20 points for discussing if the generalization performance was improved or not for each of the techniques (linear regression, kNN, and regression tree) and justifying why it was improved or alternatively why it was not improved]

### Feature Engineering

```
# creating features based on exisitng ones ( Do the transformation on numeric columns)

def transform_col(df, cols, degree):
    """
    This functiom takes in list of colums and degree then it perform some transformations based on the passed degree.
    I limited the degree to 3 for simplicity but can be extended to any degree. Also it is worth noting that I am only transforming numeric features

    :param df: our dataframe we want to apply transformation to
    :param cols: list of columns from the dataframe to be used
    :param degree: degree of transformation

    :return dataframe
    """

    if degree not in [2,3]:
        raise ValueError("Degree must be 2 or 3")

    transformed_data = df.copy()
    for col in cols:
        if col not in df.columns:
            raise ValueError(f"{col} not found in given dataset")

        new_col_name = f"{col}_{degree}"
        transformed_data[new_col_name] = df[col] ** degree

    return transformed_data

#Create some polynomial features to capture nonlinear information
transformed_df = transform_col(df = data, cols = ['last_update_days_ago','1st_update_days_ago','Freq'], degree=2)
transformed_df = transform_col(df = transformed_df, cols = ['last_update_days_ago','1st_update_days_ago','Freq'], degree=3)

#Create some Time-Based features
transformed_df['aveg_number_of_days'] = transformed_df['last_update_days_ago'] - transformed_df['1st_update_days_ago']

#Create some Aggregated Features
source_cols = [col for col in data.columns if 'source_' in col]
transformed_df['Total_Sources'] = data[source_cols].sum(axis=1)
```

*#create polynomial features \ Time based features \ Aggregate the features*
*#['Avg_Spending_Per_Freq']=transformed_df.groupby('Freq')['Spending'].transform('mean')*
*I want to add this one but it may presents some data leakage as we are calculating on entire dataset*

### Split the dataset

```
X_feature = transformed_df.drop(columns=['Spending'], axis=1)
y_feature = transformed_df['Spending']
```

### Build the model

```python
# Define models for feaure engineering

# Linear Regression
lr_feature = LinearRegression()

# kNN Classifier
knn_reg_feature = Pipeline([
        ('sc', StandardScaler()),
        ('knn', KNeighborsRegressor(n_neighbors=5))
      ])

# Decision tree
tree_reg_feature = DecisionTreeRegressor(random_state=42)
```

*# set the random state = 42 to avoid random results*

### Use 10-folds Cross-Validation to estimate the generalization performance

```python
def evaluate_model(model, X, y, cv=10, scoring='neg_mean_absolute_error'):
    scores = cross_val_score(model, X, y, cv=cv, scoring=scoring)
    std = scores.std()
    return scores, std


# Evaluate models
lr_scores_feat, lr_std_feat = evaluate_model(lr_feature, X_feature, y_feature)
knn_scores_feat, knn_std_feat = evaluate_model(knn_reg_feature, X_feature, y_feature)
tree_scores_feat, tree_std_feat = evaluate_model(tree_reg_feature, X_feature, y_feature)



print("Feature Engineering Model Results")

result_feature = pd.DataFrame({
    'Model': ['Linear Regression','k-NN Regression','Regression Tree'],
    'MAE': [np.round(-np.mean(lr_scores_feat),2),np.round(-np.mean(knn_scores_feat),2),np.round(-np.mean(tree_scores_feat),2)],
    'σ (sigma)': ['+/-'+str(round(lr_std_feat,2)), '+/-'+str(round(knn_std_feat,2)), '+/-'+str(round(tree_std_feat,2))]
})

result_feature
```

### The performance

Feature Engineering Model Results

| | Model | MAE | σ (sigma) |
|---|---|---|---|
| 0 | Linear Regression | 57.00 | +/-7.3 |
| 1 | k-NN Regression | 58.85 | +/-8.76 |
| 2 | Regression Tree | 62.82 | +/-10.66 |

### Explaining for the generation performance

Based on the results you've provided, it appears that feature engineering has led to improvements in the generalization performance of the models for Linear Regression, k-NN Regression, and Regression Tree. Let's discuss why these improvements may have occurred:

1. Linear Regression:

Before Feature Engineering (MAE: 70.50): Linear regression had a relatively high MAE, which suggests that the model was not fitting the data well.

After Feature Engineering (MAE: 57.00): After feature engineering, the MAE decreased to 57.00, indicating that the model's ability to predict the target variable (Spending) improved.

Possible Justification: The polynomial features created during feature engineering may have introduced nonlinear relationships between the input features and the target variable. This allows linear regression to capture more complex patterns in the data, leading to better predictions.

2. k-NN Regression:

Before Feature Engineering (MAE: 59.16): k-NN regression had a moderate MAE, indicating reasonable performance but with room for improvement.

After Feature Engineering (MAE: 58.85): The MAE slightly decreased to 58.85, suggesting a modest improvement.

Possible Justification: Feature engineering might have helped in identifying interactions and higher-order relationships in the data, which k-NN can capture. However, the improvement is not substantial, possibly because k-NN is already a flexible model that adapts to data patterns.

3. Regression Tree:

Before Feature Engineering (MAE: 66.74): The regression tree had the highest MAE, indicating poor generalization.

After Feature Engineering (MAE: 62.82): The MAE decreased to 62.82, showing an improvement but still higher than the other models.

Possible Justification: Feature engineering could have made the data more suitable for the regression tree model. By creating new features and interactions, it becomes easier for the tree to partition the data effectively. However, the improvement might be limited due to the inherent limitations of regression trees in capturing complex relationships.

In summary, feature engineering seems to have generally improved the generalization performance of the models, particularly in the case of Linear Regression and k-NN Regression. The introduction of polynomial features and interactions likely allowed the models to better capture the underlying patterns in the data.

**(c) (35 points) Engage in parameter tuning to optimize the performance of linear regression, k-NN, and regression tree techniques. Use cross-validations with 10 folds to estimate the generalization performance. Present the results for each of the three techniques and discuss which one yields the best performance.**

[part a is worth 35 points in total:
10 points for correctly optimizing at least two parameters for linear regression model and improving the performance as much as possible - provide screenshots and explain what you are doing and the corresponding results
10 points for correctly optimizing at least two parameters for linear k-NN model and improving the performance as much as possible - provide screenshots and explain what you are doing and the corresponding results
10 points for correctly optimizing at least two parameters for linear regression tree model and improving the performance as much as possible - provide screenshots and explain what you are doing and the corresponding results
5 points for discussing which of the three models yields the best performance]

**Linear Regression (L1 and L2)**

```
# For Linear Regression: L1 (Lasso) and L2 (Ridge) regularization
lr_params = {
    'alpha': [0.001, 0.01, 0.1, 1, 10],  # Regularization strength
}

############################################################ LASSO ############################################################
lr_grid_l1 = GridSearchCV(Lasso(random_state=42), lr_params, cv=10, scoring='neg_mean_absolute_error')

############################################################ RIDGE ############################################################
lr_grid_l2 = GridSearchCV(Ridge(random_state=42), lr_params, cv=10, scoring='neg_mean_absolute_error')

lr_grid_l1.fit(X_feature, y_feature)
lr_grid_l2.fit(X_feature, y_feature)
```

For Lasso (L1 regularization), the best alpha value is 1, resulting in an MAE of 55.21.
For Ridge (L2 regularization), the best alpha value is 10, resulting in an MAE of 56.54.
Both Lasso and Ridge perform reasonably well after parameter tuning, with Lasso having a slightly lower MAE. L1 regularization (Lasso) tends to perform feature selection, which might explain its slight advantage in this case.

**k-NN**

```
########################################### For k-NN: number of neighbors and distance metric ###########################################
knn_params = {
    'kneighborsregressor__n_neighbors': [1, 3, 5, 7, 9],  # Number of neighbors
    'kneighborsregressor__metric': ['euclidean', 'manhattan']  # Distance metric
}
knn_pipe = Pipeline([('scaler', StandardScaler()), ('kneighborsregressor', KNeighborsRegressor())])
knn_grid = GridSearchCV(knn_pipe, knn_params, cv=10, scoring='neg_mean_absolute_error')

knn_grid.fit(X_feature, y_feature)
```

The best combination of parameters is using the Manhattan distance metric and 9 neighbors, resulting in an MAE of 57.05.k-NN has a higher MAE compared to Linear Regression and Regression Tree, indicating that it might not be the best model for this dataset even after parameter tuning.

**Regression Tree**

```python
########################### For Regression Tree: tree depth and minimum samples split #######################################
tree_params = {
    'max_depth': [3, 5, 7, 9],  # Depth of the tree
    'min_samples_split': [2, 5, 10]  # Minimum number of samples required to split an internal node
}
tree_grid = GridSearchCV(DecisionTreeRegressor(random_state=42), tree_params, cv=10, scoring='neg_mean_absolute_error')

tree_grid.fit(X_feature, y_feature)

# Compile results into a DataFrame
results_feature = pd.DataFrame({
    'Model': ['Linear Regression (L1)', 'Linear Regression (L2)', 'k-NN', 'Regression Tree'],
    'Best Parameters': [lr_grid_l1.best_params_, lr_grid_l2.best_params_, knn_grid.best_params_, tree_grid.best_params_],
    'MAE': [np.round(-lr_grid_l1.best_score_,2), np.round(-lr_grid_l2.best_score_,2), np.round(-knn_grid.best_score_,2), np.round(-tree_grid.best_score_,2)]
})

results_feature
```

The best parameters are a maximum tree depth of 5 and a minimum sample split of 10, resulting in an MAE of 51.43.The Regression Tree model has the lowest MAE after parameter tuning, suggesting that it performs the best among the three models.

## The result

| | Model | Best Parameters | MAE |
|---|---|---|---|
| 0 | Linear Regression (L1) | {'alpha': 1} | 55.21 |
| 1 | Linear Regression (L2) | {'alpha': 10} | 56.54 |
| 2 | k-NN | {'kneighborsregressor__metric': 'manhattan', '... | 57.05 |
| 3 | Regression Tree | {'max_depth': 5, 'min_samples_split': 10} | 51.43 |

Conclusion: Based on the results, the Regression Tree model yields the best performance with the lowest MAE of 51.43 after parameter tuning. It appears to capture the underlying patterns in the data more effectively than Linear Regression and k-NN. Therefore, for this dataset, the Regression Tree model is recommended for predictive modeling.