Recon Raccoons : Predict the Customer Spending for Q4 - 2025

- Sahil Sangani
- Pitupoom Soontornthanon
- Matupoom Soontornthanon


```
# import all necessary libraries
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso, Ridge

# import all datasets
fraud_claim_tran = pd.read_csv("fraud_claim_tran_20250325.csv")
fraud_claim_case = pd.read_csv('fraud_claim_case_20250325.csv')
account_dim = pd.read_csv("account_dim_20250325.csv")
rams_batch_cur = pd.read_csv('rams_batch_cur_20250325.csv')
statement_fact = pd.read_csv('statement_fact_20250325.csv')
syf_id =  pd.read_csv('syf_id_20250325.csv')
transaction_fact =  pd.read_csv('transaction_fact_20250325.csv')
wrld_stor_tran_fact = pd.read_csv('wrld_stor_tran_fact_20250325.csv')
```

Introduction

Our primary research goal is to **develop a predictive model that accurately forecasts customer spending for Q4 2025 (October to December 2025)**. Our secondary research goal is to ensure that **the model provides reliable interpretative insights into the relationships between the variables in the dataset**. To achieve this, we will utilize account information, total spending amounts, employee codes, and account card types as key input variables.

This model has diverse applications. One significant advantage is its ability to predict a client's expected spending in Q4 2025, which can help determine appropriate credit line extensions. Additionally, these insights will enable businesses to strategically target high-spending customers with tailored advertisements and promotions, further encouraging increased spending.

Data Cleaning & Preprocessing

In this step, we perform data cleaning and preprocessing. Since we have data available for Q2, Q3, and Q4 of 2024 (but not Q1), we combine the total transaction amounts for each account

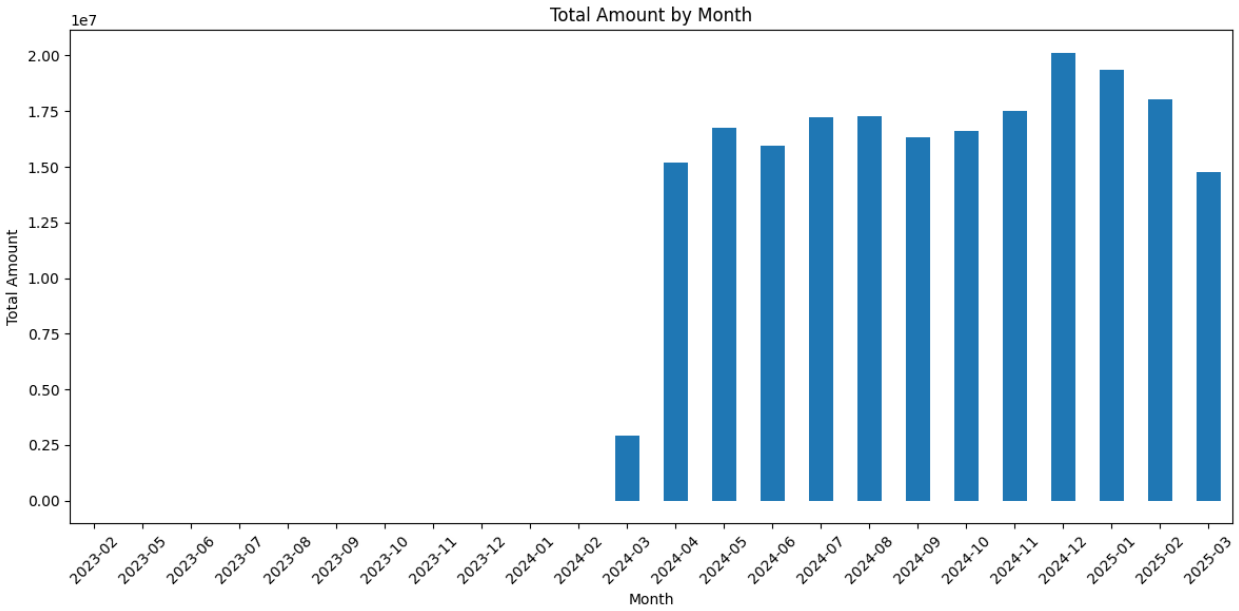
number, separated by these quarters. Later, we observe that using the total spending amounts for Q2 and Q3 separately as explanatory variables leads to multicollinearity due to their strong linear relationship. Therefore, we combine these two variables into one and use it as an explanatory variable.

```
# Combine transaction_fact and wrld_stor_tran_fact
combined_transactions = pd.concat([transaction_fact,
wrld_stor_tran_fact], ignore_index=True)
combined_transactions =
combined_transactions.merge(account_dim[['current_account_nbr',
'client_id', 'employee_code', 'account_card_type']],
on='current_account_nbr', how='left')
combined_transactions = combined_transactions[['current_account_nbr',
'client_id', 'transaction_date', 'transaction_amt', 'employee_code',
'account_card_type']]
combined_transactions.head(3)
```

```
  current_account_nbr      client_id transaction_date  transaction_amt
\
0    X7jfKh6xrPAB8Tx6       JTX290DC       2024-06-05            15.78

1    yntD77AZDylS48Q4  SVRDCMSNAS887       2024-06-19            14.85

2    LIJPI0sK28Pa7fX2  SVRDCMSNAS887       2024-06-26           136.16


  employee_code account_card_type
0           NaN         DUAL CARD
1             H         DUAL CARD
2             H         DUAL CARD
```

```
# Convert transaction_date to datetime
combined_transactions['transaction_date'] =
pd.to_datetime(combined_transactions['transaction_date'])

# Create month column
combined_transactions['month'] =
combined_transactions['transaction_date'].dt.to_period('M')

# Group by month and sum transaction amounts
monthly_totals = combined_transactions.groupby('month')
['transaction_amt'].sum()

# Create the histogram
plt.figure(figsize=(12, 6))
monthly_totals.plot(kind='bar')
plt.title('Total Amount by Month')
plt.xlabel('Month')
plt.ylabel('Total Amount')
plt.xticks(rotation=45)
```

```
plt.tight_layout()
plt.show()
```



According to the above histogram, we have only data from Q2-Q4 of 2024 available (not Q1).

```
# Create new columns to indicate the quarter and year
combined_transactions['quarter'] =
pd.to_datetime(combined_transactions['transaction_date']).dt.quarter
combined_transactions['year'] =
pd.to_datetime(combined_transactions['transaction_date']).dt.year
combined_transactions
```

```
        current_account_nbr      client_id transaction_date
transaction_amt  \
0          X7jfKh6xrPAB8Tx6       JTX290DC       2024-06-05
15.78
1          yntD77AZDylS48Q4  SVRDCMSNAS887       2024-06-19
14.85
2          LIJPI0sK28Pa7fX2  SVRDCMSNAS887       2024-06-26
136.16
3          CMAr5Apxwdzpvoze    ARPPYALC768       2024-08-15
8.74
4          eJSfTCGPvJulGzd3    ARPPYALC768       2024-08-17
26.65
...                     ...            ...              ...
...
1547185    uqlll7MEoZkj3vur  SVRDCMSNAS887       2024-11-12
9.57
1547186    ZGjYc8aETjtytUYR  SVRDCMSNAS887       2025-03-10
16.11
```

```
1547187    4JpGONAzyfmoxi7A  SVRDCMSNAS887       2025-03-20
20.37
1547188    tUUUlJJZ2ETedT7g  SVRDCMSNAS887       2024-11-21
180.00
1547189    u7AfBCIabLPkr5y7  SVRDCMSNAS887       2024-12-08
2.50

        employee_code account_card_type    month  quarter  year
0                 NaN         DUAL CARD  2024-06        2  2024
1                   H         DUAL CARD  2024-06        2  2024
2                   H         DUAL CARD  2024-06        2  2024
3                 NaN              PLCC  2024-08        3  2024
4                 NaN              PLCC  2024-08        3  2024
...               ...               ...      ...      ...   ...
1547185             H         DUAL CARD  2024-11        4  2024
1547186           NaN         DUAL CARD  2025-03        1  2025
1547187             H         DUAL CARD  2025-03        1  2025
1547188             H         DUAL CARD  2024-11        4  2024
1547189             H         DUAL CARD  2024-12        4  2024

[1547190 rows x 9 columns]
```

```
# Convert missing value to 'N' (not a high spending customer)
np.unique(combined_transactions['employee_code'].fillna('N').values)
combined_transactions['employee_code'] =
combined_transactions['employee_code'].fillna('N')

# Extract only observations we are interested in (Q2, Q3, Q4 of 2024)
combined_transactions =
combined_transactions[(combined_transactions['year'] == 2024) &
(combined_transactions['quarter'] != 1)]
combined_transactions
```

```
        current_account_nbr      client_id transaction_date
transaction_amt  \
0          X7jfKh6xrPAB8Tx6       JTX290DC       2024-06-05
15.78
1          yntD77AZDylS48Q4  SVRDCMSNAS887       2024-06-19
14.85
2          LIJPI0sK28Pa7fX2  SVRDCMSNAS887       2024-06-26
136.16
3          CMAr5Apxwdzpvoze    ARPPYALC768       2024-08-15
8.74
4          eJSfTCGPvJulGzd3    ARPPYALC768       2024-08-17
26.65
...                     ...            ...              ...
...
1547180    9gJsz8tSaVOwokPd  SVRDCMSNAS887       2024-10-12
25.67
1547183    WBBqIOirecnTtAoI  SVRDCMSNAS887       2024-10-18
```

```
17.48
1547185    uqlll7MEoZkj3vur  SVRDCMSNAS887       2024-11-12
9.57
1547188    tUUUlJJZ2ETedT7g  SVRDCMSNAS887       2024-11-21
180.00
1547189    u7AfBCIabLPkr5y7  SVRDCMSNAS887       2024-12-08
2.50
```

```
        employee_code account_card_type    month  quarter  year
0                   N         DUAL CARD  2024-06        2  2024
1                   H         DUAL CARD  2024-06        2  2024
2                   H         DUAL CARD  2024-06        2  2024
3                   N              PLCC  2024-08        3  2024
4                   N              PLCC  2024-08        3  2024
...               ...               ...      ...      ...   ...
1547180             H         DUAL CARD  2024-10        4  2024
1547183             H         DUAL CARD  2024-10        4  2024
1547185             H         DUAL CARD  2024-11        4  2024
1547188             H         DUAL CARD  2024-11        4  2024
1547189             H         DUAL CARD  2024-12        4  2024

[1154636 rows x 9 columns]
```

```
# Separate the data into quarters
Q2 = combined_transactions[combined_transactions['quarter'] == 2]
[['current_account_nbr', 'transaction_amt', 'employee_code',
'account_card_type']]
Q3 = combined_transactions[combined_transactions['quarter'] == 3]
[['current_account_nbr', 'transaction_amt', 'employee_code',
'account_card_type']]
Q4 = combined_transactions[combined_transactions['quarter'] == 4]
[['current_account_nbr', 'transaction_amt', 'employee_code',
'account_card_type']]
Q2.head(1)
```

```
  current_account_nbr  transaction_amt employee_code account_card_type
0    X7jfKh6xrPAB8Tx6            15.78             N         DUAL CARD
```

```
# Group by current_account_nbr and sum transaction amounts for each
quarter

Q2_sum = Q2.groupby('current_account_nbr').agg({'transaction_amt':
'sum'}).reset_index()
Q2_sum = Q2_sum.rename(columns={'transaction_amt':
'Q2_transaction_amt'})

Q3_sum = Q3.groupby('current_account_nbr').agg({'transaction_amt':
'sum'}).reset_index()
Q3_sum = Q3_sum.rename(columns={'transaction_amt':
'Q3_transaction_amt'})
```

```
Q4_sum = Q4.groupby('current_account_nbr').agg({'transaction_amt':
'sum'}).reset_index()
Q4_sum = Q4_sum.rename(columns={'transaction_amt':
'Q4_transaction_amt'})
Q4_sum.head(3)
```

```
  current_account_nbr  Q4_transaction_amt
0    00iP5U82D8XwVQ9G             3449.74
1    00oyr3QppAzjLws4              515.96
2    033o9yHYen3xoz6k            14571.86
```

```
# Create X (Q2 and Q3)
X = Q2_sum.merge(Q3_sum, on='current_account_nbr', how='outer')
X.shape[0] - X[X['Q2_transaction_amt'].isnull() |
X['Q3_transaction_amt'].isnull()].shape[0]

# Fill the missing value with 0 (indicating no transaction in that
quarter)
X['Q3_transaction_amt'] = X['Q3_transaction_amt'].fillna(0)
X
```

```
      current_account_nbr  Q2_transaction_amt  Q3_transaction_amt
0       00iP5U82D8XwVQ9G             1331.59             2262.72
1       00oyr3QppAzjLws4              625.52              390.04
2       033o9yHYen3xoz6k             8855.70            10198.43
3       034bM166vNmgLiIA               19.98             2615.57
4       03n28YA8ljfM9tor              141.11              682.16
...                  ...                 ...                 ...
11065   zxwuHFEBf4ERmY9F             8869.41             1104.80
11066   zyZhjzJwhpOgSvmc             1209.14             3314.56
11067   zyikbceuTT3GcAH6                 NaN                0.00
11068   zzBy2qNM78aRV580              546.99             4396.62
11069   zzEuUBBmvGiVnabb             7594.29             5658.56

[11070 rows x 3 columns]
```

```
# Create y (Q4)
y = Q4_sum
y
```

```
      current_account_nbr  Q4_transaction_amt
0       00iP5U82D8XwVQ9G             3449.74
1       00oyr3QppAzjLws4              515.96
2       033o9yHYen3xoz6k            14571.86
3       034bM166vNmgLiIA             1207.40
4       03cqv0gFjEIiQG0x             1232.85
...                  ...                 ...
10907   zxwuHFEBf4ERmY9F             1981.68
10908   zyZhjzJwhpOgSvmc             4194.89
10909   zz3nbtZXS41NZk0h              239.82
```

```
10910    zzBy2qNM78aRV580              520.00
10911    zzEuUBBmvGiVnabb            12242.41

[10912 rows x 2 columns]
```

```
# Merge with account_dim to get employee_code and account_card_type
X = X.merge(account_dim[['current_account_nbr', 'employee_code',
'account_card_type']], on='current_account_nbr', how='left')
X['employee_code'] = X['employee_code'].fillna('N')
X
```

```
      current_account_nbr  Q2_transaction_amt  Q3_transaction_amt  \
0        00iP5U82D8XwVQ9G             1331.59             2262.72
1        00oyr3QppAzjLws4              625.52              390.04
2        033o9yHYen3xoz6k             8855.70            10198.43
3        034bM166vNmgLiIA               19.98             2615.57
4        03n28YA8ljfM9tor              141.11              682.16
...                   ...                 ...                 ...
11065    zxwuHFEBf4ERmY9F             8869.41             1104.80
11066    zyZhjzJwhpOgSvmc             1209.14             3314.56
11067    zyikbceuTT3GcAH6                 NaN                0.00
11068    zzBy2qNM78aRV580               546.99             4396.62
11069    zzEuUBBmvGiVnabb              7594.29             5658.56

      employee_code account_card_type
0                 N              PLCC
1                 N              PLCC
2                 H         DUAL CARD
3                 N         DUAL CARD
4                 N              PLCC
...             ...               ...
11065             N              PLCC
11066             N         DUAL CARD
11067             N         DUAL CARD
11068             N         DUAL CARD
11069             H         DUAL CARD

[11070 rows x 5 columns]
```

```
X['Q2_transaction_amt'].isna().sum()
```

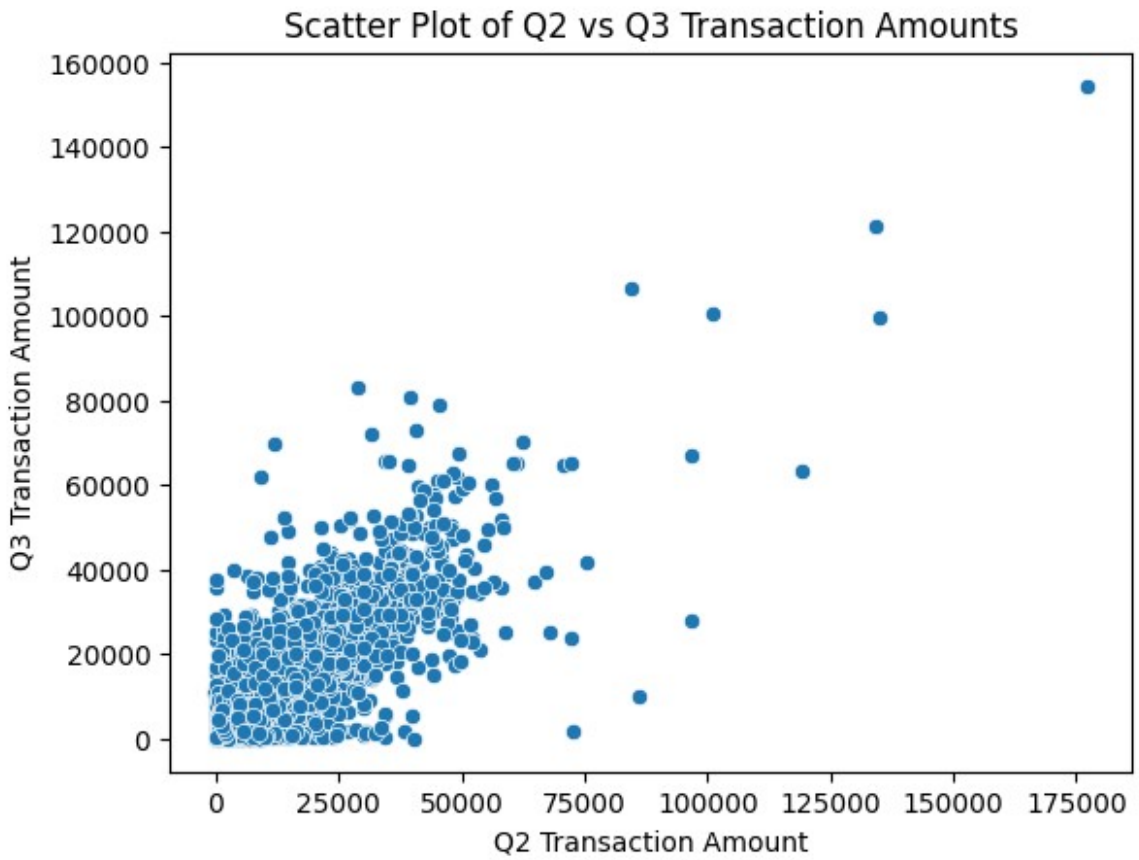
```
np.int64(1691)
```

```
# Drop rows with missing values
X_and_y = X.merge(y, on='current_account_nbr', how='inner')
X_and_y = X_and_y.dropna()
X_and_y = X_and_y.drop(columns=['current_account_nbr'], axis=1)
X_and_y
```

```
      Q2_transaction_amt  Q3_transaction_amt employee_code
account_card_type  \
```

```
0                 1331.59              2262.72              N
PLCC
1                  625.52               390.04              N
PLCC
2                 8855.70             10198.43              H
DUAL CARD
3                   19.98              2615.57              N
DUAL CARD
4                  141.11               682.16              N
PLCC
...                   ...                  ...            ...
...
9534              5379.70              1628.11              N
PLCC
9535              8869.41              1104.80              N
PLCC
9536              1209.14              3314.56              N
DUAL CARD
9537               546.99              4396.62              N
DUAL CARD
9538              7594.29              5658.56              H
DUAL CARD

      Q4_transaction_amt
0                3449.74
1                 515.96
2               14571.86
3                1207.40
4                   2.10
...                   ...
9534             1031.23
9535             1981.68
9536             4194.89
9537              520.00
9538            12242.41

[8101 rows x 5 columns]
```

```
sns.scatterplot(data=X_and_y, x='Q2_transaction_amt',
y='Q3_transaction_amt')
plt.xlabel('Q2 Transaction Amount')
plt.ylabel('Q3 Transaction Amount')
plt.title('Scatter Plot of Q2 vs Q3 Transaction Amounts')
plt.show()
```




```
X_and_y.corr(numeric_only=True)
```

```
                    Q2_transaction_amt  Q3_transaction_amt
Q4_transaction_amt
Q2_transaction_amt            1.000000            0.862460
0.824462
Q3_transaction_amt            0.862460            1.000000
0.893326
Q4_transaction_amt            0.824462            0.893326
1.000000
```

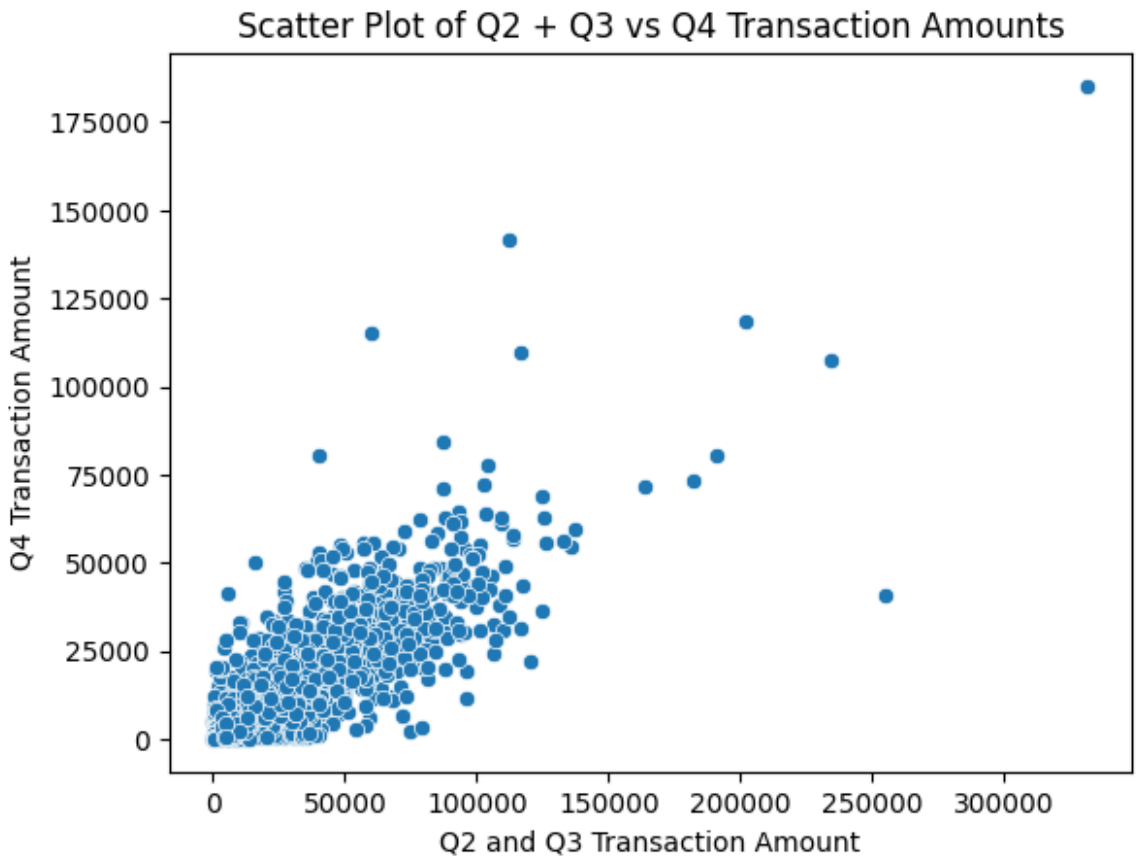
There is a strong linear relationship between Q2 and Q3 transaction amounts (since their correlation coefficient = 0.862460 > 0.7), which might lead to an issue with multicollinearity. We will address this problem by combining Q2 and Q3 transaction amounts.

```
X_and_y['Q2_and_Q3_transaction_amt'] = X_and_y['Q2_transaction_amt'] +
X_and_y['Q3_transaction_amt']
X_and_y = X_and_y.drop(columns=['Q2_transaction_amt',
'Q3_transaction_amt'], axis=1)
X_and_y
```

```
     employee_code account_card_type  Q4_transaction_amt  \
0                N              PLCC             3449.74
```

```
1                  N                PLCC                515.96
2                  H          DUAL CARD              14571.86
3                  N          DUAL CARD               1207.40
4                  N                PLCC                  2.10
...              ...                 ...                   ...
9534               N                PLCC               1031.23
9535               N                PLCC               1981.68
9536               N          DUAL CARD               4194.89
9537               N          DUAL CARD                520.00
9538               H          DUAL CARD              12242.41

      Q2_and_Q3_transaction_amt
0                       3594.31
1                       1015.56
2                      19054.13
3                       2635.55
4                        823.27
...                         ...
9534                    7007.81
9535                    9974.21
9536                    4523.70
9537                    4943.61
9538                   13252.85

[8101 rows x 4 columns]
```

```
sns.scatterplot(data=X_and_y, x='Q2_and_Q3_transaction_amt',
y='Q4_transaction_amt')
plt.xlabel('Q2 and Q3 Transaction Amount')
plt.ylabel('Q4 Transaction Amount')
plt.title('Scatter Plot of Q2 + Q3 vs Q4 Transaction Amounts')
plt.show()
```




```
X_and_y[(X_and_y['Q2_and_Q3_transaction_amt'] > 200000)]
```

	employee_code	account_card_type	Q4_transaction_amt	Q2_and_Q3_transaction_amt
1888	H	DUAL CARD	185236.32	331849.36
2043	H	DUAL CARD	107729.53	234701.69
2432	H	DUAL CARD	118635.79	201879.15
4601	H	DUAL CARD	40595.34	255404.60

```
# Remove outlier
X_and_y_drop = X_and_y[(X_and_y['Q2_and_Q3_transaction_amt'] !=
255404.60) & (X_and_y['Q4_transaction_amt'] != 40595.34)]
X_and_y_drop
```

	employee_code	account_card_type	Q4_transaction_amt
0	N	PLCC	3449.74
1	N	PLCC	515.96
2	H	DUAL CARD	14571.86

```
3                  N         DUAL CARD              1207.40
4                  N              PLCC                 2.10
...              ...               ...                  ...
9534               N              PLCC              1031.23
9535               N              PLCC              1981.68
9536               N         DUAL CARD              4194.89
9537               N         DUAL CARD               520.00
9538               H         DUAL CARD             12242.41

      Q2_and_Q3_transaction_amt
0                       3594.31
1                       1015.56
2                      19054.13
3                       2635.55
4                        823.27
...                         ...
9534                    7007.81
9535                    9974.21
9536                    4523.70
9537                    4943.61
9538                   13252.85

[8100 rows x 4 columns]
```

```
# Perform One-Hot Encoding, creating 0/1 (True/False) indicator
explanatory variables
df_with_ind = pd.get_dummies(X_and_y_drop, drop_first=True, dtype=int)
df_with_ind
```

```
      Q4_transaction_amt  Q2_and_Q3_transaction_amt
employee_code_N  \
0                3449.74                    3594.31                  1

1                 515.96                    1015.56                  1

2               14571.86                   19054.13                  0

3                1207.40                    2635.55                  1

4                   2.10                     823.27                  1

...                  ...                        ...                ...

9534             1031.23                    7007.81                  1

9535             1981.68                    9974.21                  1

9536             4194.89                    4523.70                  1

9537              520.00                    4943.61                  1
```

```
9538             12242.41                    13252.85                  0


      employee_code_Y  account_card_type_PLCC
0                   0                       1
1                   0                       1
2                   0                       0
3                   0                       0
4                   0                       1
...               ...                     ...
9534                0                       1
9535                0                       1
9536                0                       0
9537                0                       0
9538                0                       0

[8100 rows x 5 columns]
```

Fitting LASSO and Ridge Linear Regression Models

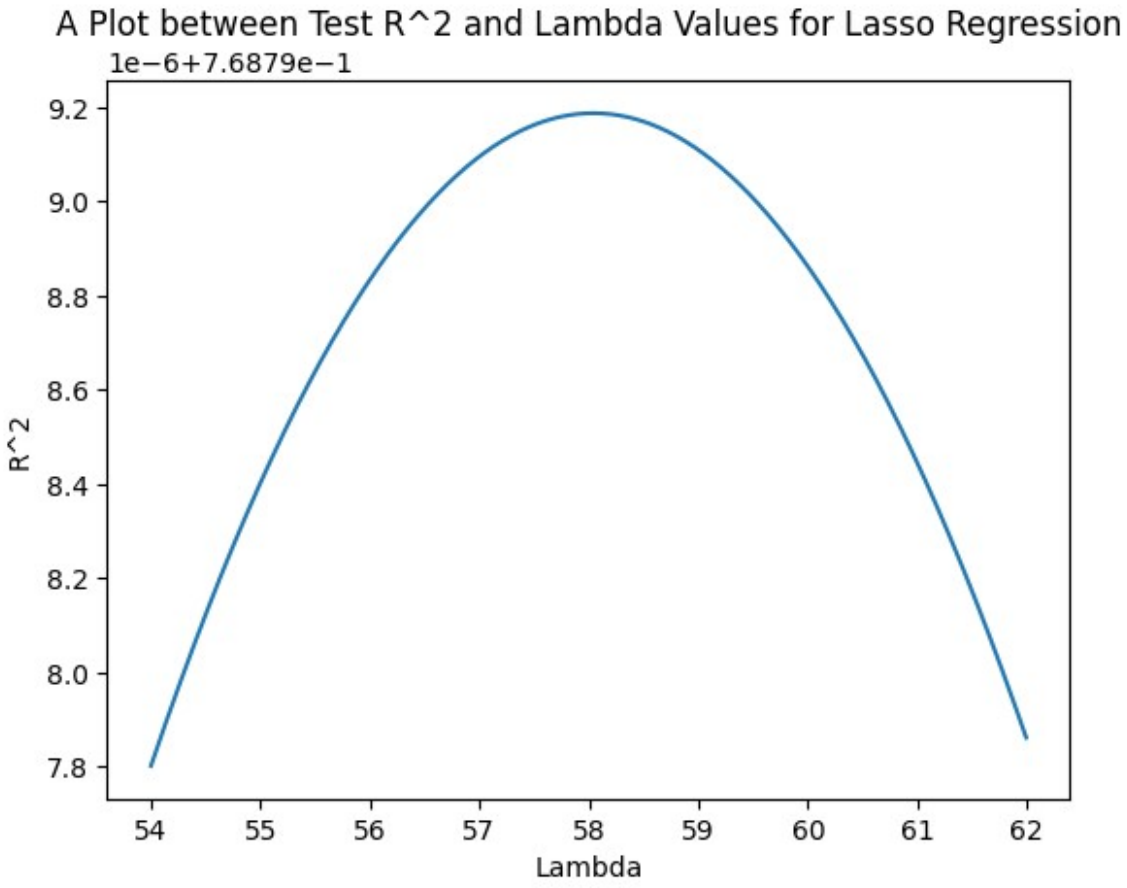
Split the dataset

```
df_train, df_test = train_test_split(df_with_ind, test_size=0.2,
random_state=101)

X_train = df_train.drop(['Q4_transaction_amt'], axis=1)
y_train = df_train['Q4_transaction_amt']
X_test = df_test.drop(['Q4_transaction_amt'], axis=1)
y_test = df_test['Q4_transaction_amt']
```

LASSO Linear Regrssion Model

```
# All lambda values to be tested
lambdas=[]
for lam in np.arange(54, 62, 0.005):
    lambdas.append(lam)
len(lambdas)

# Fit the LASSO regression model for each lambda value
results_lasso = []
for lam in lambdas:
    lasso_mod = Lasso(alpha=lam, max_iter=1000)
    lasso_mod.fit(X_train, y_train)
    results_lasso.append([lam, lasso_mod.score(X_test, y_test)])

df_output_lasso = pd.DataFrame(results_lasso, columns=['lambda',
```

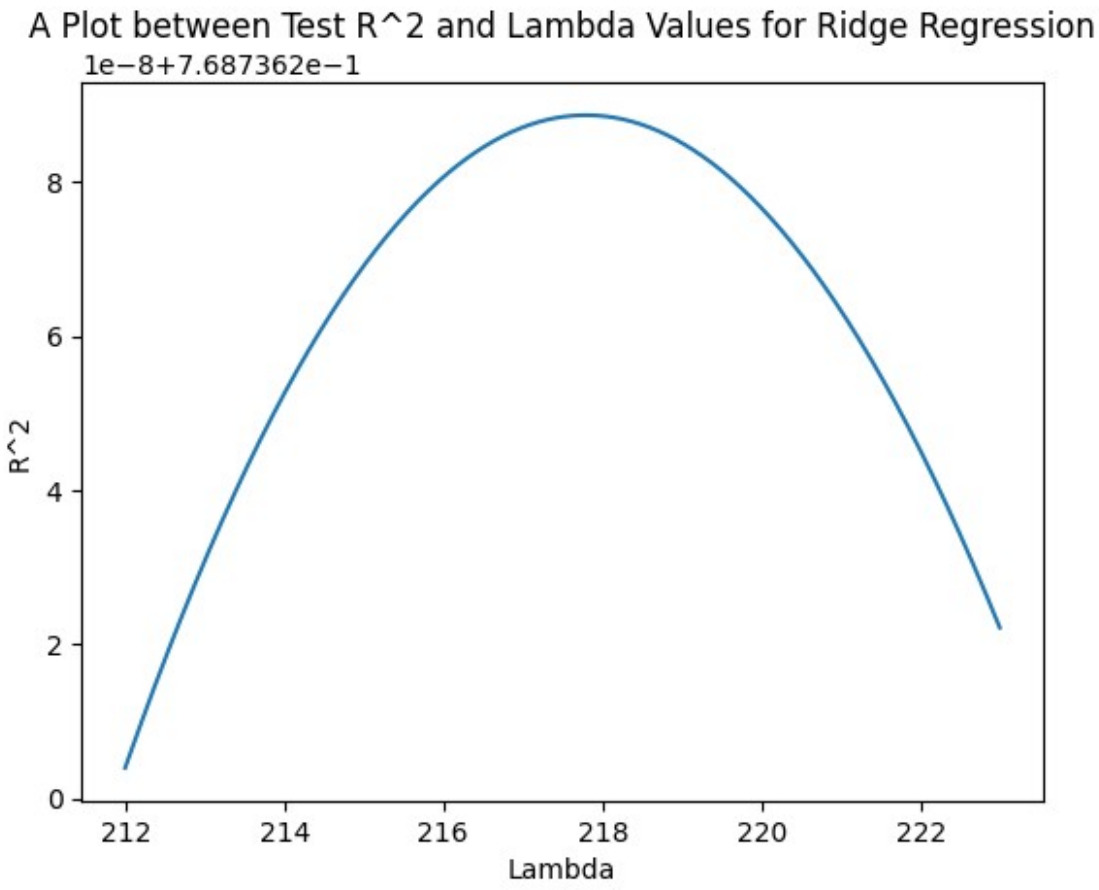
```
'test_r2'])

# Display the plot between test R^2 and lambda values
plt.plot(df_output_lasso['lambda'].values,
df_output_lasso['test_r2'].values)
plt.title('A Plot between Test R^2 and Lambda Values for Lasso
Regression')
plt.xlabel('Lambda')
plt.ylabel('R^2')
plt.show()

# Display the best lambda and test R^2 values
print(f'The best lambda value is
{df_output_lasso[df_output_lasso["test_r2"] ==
df_output_lasso["test_r2"].max()]["lambda"].values[0]}')
print(f'The best test R^2 value is
{df_output_lasso[df_output_lasso["test_r2"] ==
df_output_lasso["test_r2"].max()]["test_r2"].values[0]}')
```




```
The best lambda value is 58.04000000000207
The best test R^2 value is 0.768799187377643
```

Ridge Linear Regrssion Model

```
# All lambda values to be tested
lambdas=[]
for lam in np.arange(212, 223, 0.005):
    lambdas.append(lam)
len(lambdas)

# Fit the Ridge regression model for each lambda value
results_ridge = []
for lam in lambdas:
    ridge_mod = Ridge(alpha=lam, max_iter=1000)
    ridge_mod.fit(X_train, y_train)
    results_ridge.append([lam, ridge_mod.score(X_test, y_test)])

df_output_ridge = pd.DataFrame(results_ridge, columns=['lambda',
'test_r2'])

# Display the plot between test R^2 and lambda values
plt.plot(df_output_ridge['lambda'].values,
df_output_ridge['test_r2'].values)
plt.title('A Plot between Test R^2 and Lambda Values for Ridge
Regression')
plt.xlabel('Lambda')
plt.ylabel('R^2')
plt.show()

# Display the best lambda and test R^2 values
print(f'The best lambda value is
{df_output_ridge[df_output_ridge["test_r2"] ==
df_output_ridge["test_r2"].max()]["lambda"].values[0]}')
print(f'The best test R^2 value is
{df_output_ridge[df_output_ridge["test_r2"] ==
df_output_ridge["test_r2"].max()]["test_r2"].values[0]}')
```




```
The best lambda value is 217.79499999999473
The best test R^2 value is 0.7687362886599971
```

Conclusion

In conclusion, regarding the primary research goal, our best linear regression model is the LASSO Linear Regression model with $\lambda = 58.04$. The equation is expressed as follows:

$$ \hat{\text{Q4\_transaction\_amt}} = 1801.6058130608762 + 0.448517 \cdot \text{Q2\_and\_Q3\_transaction\_amt} \\ - 1740.706655 \cdot \text{employee\_code\_N} - 31.293494 \cdot \text{account\_card\_type\_PLCC} $$

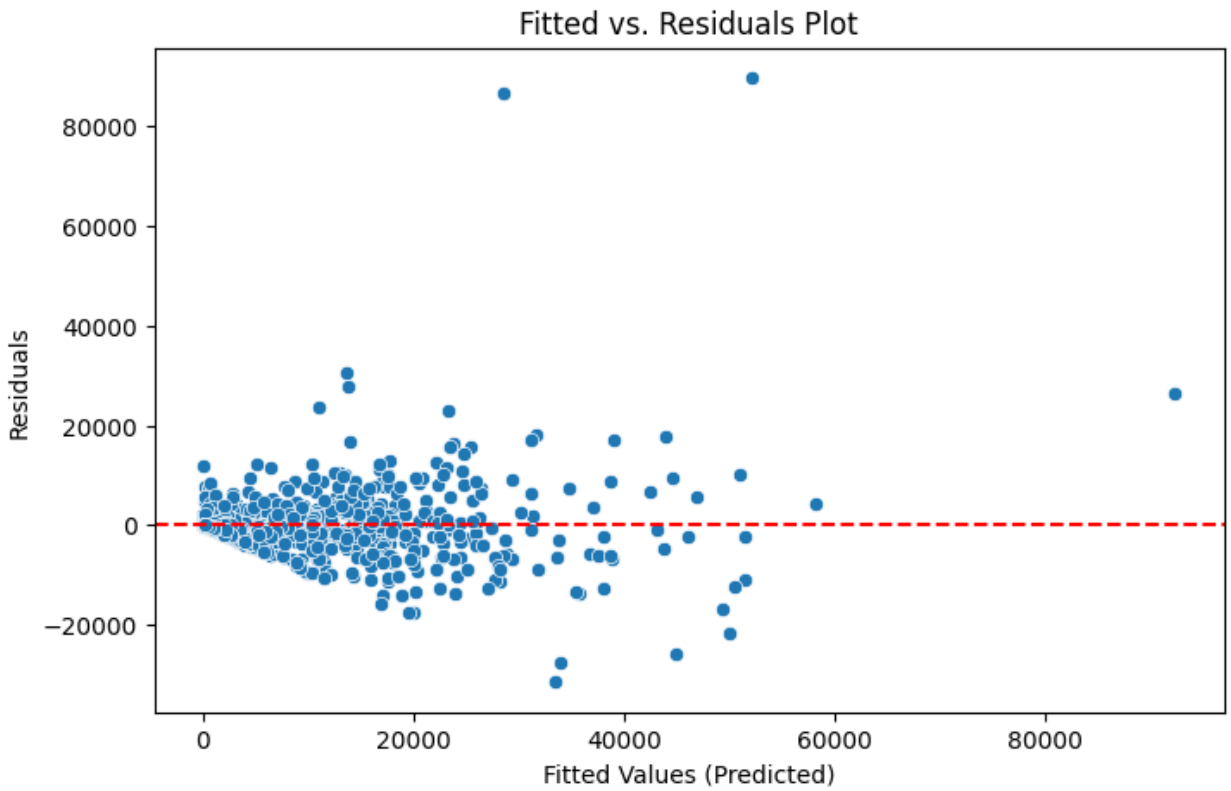
The test R^2 of this model is approximately 0.7688, which is considered good.

Regarding the secondary research goal, according to the fitted vs. residuals plot, as we create small equally sized boxes over the range of fitted values. the number of negative and positive residuals is roughly the same in *all* boxes. Also, our response variable `Q4_transaction_amt` is a numerical variable. Therefore, we can infer that the relationship between the explanatory variable and the response variable is indeed linear. In other words, the model meets the linearity assumption. In addition, there is no issue of multicollinearity. hus, we can conclude that the model provides reliable interpretative insights.

```
lin_mod = Lasso(alpha=58.04, max_iter=1000)
lin_mod.fit(X_train, y_train)
print(f'The test R^2 for the best model (LASSO with lambda = 58.04) is
{lin_mod.score(X_test, y_test)}')
```

```
The test R^2 for the best model (LASSO with lambda = 58.04) is
0.768799187377643
```

```
# Display the fitted vs. residuals plot
y_pred = lin_mod.predict(X_test)
residuals = y_test - y_pred

plt.figure(figsize=(8, 5))
sns.scatterplot(x=y_pred, y=residuals)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel("Fitted Values (Predicted)")
plt.ylabel("Residuals")
plt.title("Fitted vs. Residuals Plot")
plt.show()
```




```
print(f'The intercept of the model = {Lasso(alpha=58.04,
max_iter=1000).fit(X_train, y_train).intercept_}')
```

```
The intercept of the model = 1801.6058130608762
```

```
# Display the coefficients/slopes of the model
df_slopes = pd.DataFrame(Lasso(alpha=58.04).fit(X_train,
y_train).coef_.T, columns=['Lasso_mod'], index=X_train.columns)
df_slopes
```

	Lasso_mod
Q2_and_Q3_transaction_amt	0.448517
employee_code_N	-1740.706655
employee_code_Y	-0.000000
account_card_type_PLCC	-31.293494