customer\_ltv

# 1.0 DATA PREPARATION[¶](#X2f56ca402d3ac7c467e58123c54ebef74847c61)

In [46]:

import pandas as pd  
import numpy as np  
import plotnine as pn  
from xgboost import XGBClassifier, XGBRegressor  
from sklearn.model\_selection import GridSearchCV  
import matplotlib.pyplot as plt

In [47]:

pn.options.dpi = 300

In [48]:

cdnow\_raw\_df = pd.read\_csv('../data/cdnow.csv', index\_col=0)

In [49]:

cdnow\_raw\_df.head(3)

Out[49]:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | customer\_id | date | quantity | price |
| 0 | 1 | 1997-01-01 | 1 | 11.77 |
| 1 | 2 | 1997-01-12 | 1 | 12.00 |
| 2 | 2 | 1997-01-12 | 5 | 77.00 |

In [50]:

cdnow\_raw\_df.info()

<class 'pandas.core.frame.DataFrame'>  
Index: 69659 entries, 0 to 69658  
Data columns (total 4 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 customer\_id 69659 non-null int64   
 1 date 69659 non-null object   
 2 quantity 69659 non-null int64   
 3 price 69659 non-null float64  
dtypes: float64(1), int64(2), object(1)  
memory usage: 2.7+ MB

In [51]:

# Print the index range  
customer\_range = cdnow\_raw\_df.customer\_id.min(), cdnow\_raw\_df.customer\_id.max()  
print(f"Customer range: {customer\_range}")

Customer range: (1, 23570)

In [52]:

# convert to datetime  
cdnow\_df = (  
 cdnow\_raw\_df  
 .assign(date=lambda x: pd.to\_datetime(x['date']))  
)

In [53]:

cdnow\_df['date'].dtype

Out[53]:

dtype('<M8[ns]')

# 2.0 COHORT ANALYSIS[¶](#Xf40790c8dabba173139f9a0a0288484323f07a1)

In [54]:

# Extract just the date part without time  
min\_date = cdnow\_df['date'].min().date()  
max\_date = cdnow\_df['date'].max().date()  
  
# Print the range of dates  
print(f"Data ranges from {min\_date} to {max\_date}")

Data ranges from 1997-01-01 to 1998-06-30

## Get Range of Initial Purchases[¶](#Get-Range-of-Initial-Purchases)

In [55]:

cdnow\_first\_purchase\_tbl = (  
 cdnow\_df  
 .sort\_values(['customer\_id','date'])  
 .groupby('customer\_id')  
 .first()  
 )

In [56]:

cdnow\_first\_purchase\_tbl.head()

Out[56]:

date

quantity

price

customer\_id

1

1997-01-01

1

11.77

2

1997-01-12

1

12.00

3

1997-01-02

2

20.76

4

1997-01-01

2

29.33

5

1997-01-01

2

29.33

In [57]:

# Extract just the date part without time  
min\_date = cdnow\_first\_purchase\_tbl['date'].min().date()  
max\_date = cdnow\_first\_purchase\_tbl['date'].max().date()  
  
# Print the range of dates  
print(f"Dates range of initial purchases from {min\_date} to {max\_date}")

Dates range of initial purchases from 1997-01-01 to 1997-03-25

Despite containing a year and a half of data, all customers were acquired between 1/1997 and 3/1997.

## Visualize: All purchases within cohort[¶](#Visualize:-All-purchases-within-cohort)

In [58]:

cdnow\_df.reset\_index() \  
 .set\_index('date') \  
 [['price']] \  
 .resample(rule='MS') \  
 .sum() \  
 .plot()

Out[58]:

<Axes: xlabel='date'>

![No description has been provided for this image](data:image/png;base64;base64,)

## Visualize: Individual Customer Purchases[¶](#Visualize:-Individual-Customer-Purchases)

In [59]:

# Select a subset of customer IDs  
ids = cdnow\_df['customer\_id'].unique()  
ids\_selected = ids[:12]

In [60]:

# ggplot API  
# Filter the DataFrame for the selected customer IDs  
cdnow\_cust\_id\_subset\_df = (  
 cdnow\_df[cdnow\_df['customer\_id'].isin(ids\_selected)]  
 .groupby(['customer\_id', 'date'])  
 .sum()  
 .reset\_index()  
)  
  
# Create the plot  
plot = (  
 pn.ggplot(  
 cdnow\_cust\_id\_subset\_df,  
 pn.aes('date', 'price', group='customer\_id')  
 )  
 + pn.geom\_line()  
 + pn.geom\_point()  
 + pn.facet\_wrap('~customer\_id')  
 + pn.scale\_x\_date(  
 date\_breaks='1 year',  
 date\_labels='%Y'  
 )  
)  
  
# Display the plot  
print(plot)

/tmp/ipykernel\_798878/2476615928.py:26: FutureWarning: Using print(plot) to draw and show the plot figure is deprecated and will be removed in a future version. Use plot.show().  
/home/blackitalian/miniconda3/envs/cdnow/lib/python3.11/site-packages/plotnine/geoms/geom\_path.py:113: PlotnineWarning: geom\_path: Each group consist of only one observation. Do you need to adjust the group aesthetic?  
/home/blackitalian/miniconda3/envs/cdnow/lib/python3.11/site-packages/plotnine/geoms/geom\_path.py:113: PlotnineWarning: geom\_path: Each group consist of only one observation. Do you need to adjust the group aesthetic?  
/home/blackitalian/miniconda3/envs/cdnow/lib/python3.11/site-packages/plotnine/geoms/geom\_path.py:113: PlotnineWarning: geom\_path: Each group consist of only one observation. Do you need to adjust the group aesthetic?  
/home/blackitalian/miniconda3/envs/cdnow/lib/python3.11/site-packages/plotnine/geoms/geom\_path.py:113: PlotnineWarning: geom\_path: Each group consist of only one observation. Do you need to adjust the group aesthetic?

![No description has been provided for this image](data:image/png;base64;base64,)

In [61]:

# Number of unique customers in the subset  
unique\_customers = cdnow\_cust\_id\_subset\_df['customer\_id'].nunique()  
  
# Define the number of rows and columns for the subplots  
num\_cols = 3  
num\_rows = (unique\_customers + num\_cols - 1) // num\_cols  
  
fig, axes = plt.subplots(num\_rows, num\_cols, figsize=(15, 10), sharex=True, sharey=True)  
  
# Flatten the axes array for easy iteration  
axes = axes.flatten()  
  
for ax, (customer\_id, group) in zip(axes, cdnow\_cust\_id\_subset\_df.groupby('customer\_id')):  
 ax.plot(group['date'], group['price'], marker='o', linestyle='-')  
 ax.set\_title(f'Customer ID: {customer\_id}')  
 ax.set\_xlabel('Date')  
 ax.set\_ylabel('Price')  
 ax.xaxis.set\_major\_formatter(plt.matplotlib.dates.DateFormatter('%Y'))  
 ax.grid(True)   
  
# Remove any empty subplots  
for i in range(unique\_customers, len(axes)):  
 fig.delaxes(axes[i])  
  
plt.tight\_layout()  
plt.show()

![No description has been provided for this image](data:image/png;base64;base64,)

# 3.0 FEATURE ENGINEERING[¶](#Xabde457e9e5c0e6efce4acbd156c6723c95be82)

Frame problem:

* What will the customers spend in the next 90-days? (Regression)
* What is the probability of a customer to make a purchase in the next 90-days? (Classification)

## 3.1 TIME SPLITTING[¶](#X97230852b867ed49dd6783cd03729e49c50314e)

In [62]:

n\_days = 90  
max\_date = cdnow\_df['date'].max()  
  
# Calculate the cutoff date (90 days before end of data)  
cutoff = max\_date - pd.to\_timedelta(n\_days, unit='d')

In [63]:

temporal\_in\_df = cdnow\_df[cdnow\_df['date'] <= cutoff]  
temporal\_out\_df = cdnow\_df[cdnow\_df['date'] > cutoff]

In [64]:

# Extract the minimum and maximum dates  
min\_date = temporal\_in\_df['date'].min().date()  
max\_date = temporal\_in\_df['date'].max().date()  
  
# Print the range of dates  
print(f"Dates range of temporal\_in\_df from {min\_date} to {max\_date}")

Dates range of temporal\_in\_df from 1997-01-01 to 1998-04-01

In [65]:

# Extract the minimum and maximum dates  
min\_date = temporal\_out\_df['date'].min().date()  
max\_date = temporal\_out\_df['date'].max().date()  
  
# Print the range of dates  
print(f"Dates range of temporal\_out\_df from {min\_date} to {max\_date}")

Dates range of temporal\_out\_df from 1998-04-02 to 1998-06-30

## 3.2 FEATURE ENGINEERING (RFM)[¶](#X5dc05e5c121074fde6aa472738ece1985020eb4)

### Make Targets From Data[¶](#Make-Targets-From-Data)

In [66]:

# flag for all data in the last 90 days  
targets\_df = (  
 temporal\_out\_df  
 .drop(['quantity','date'], axis=1) # must drop date to .sum()  
 .groupby('customer\_id')  
 .sum()  
 .rename({'price': 'spend\_90\_total'}, axis=1)  
 .assign(spend\_90\_flag=1) # Add a new column 'spend\_90\_flag' with value 1  
)

In [67]:

targets\_df.head(3)

Out[67]:

spend\_90\_total

spend\_90\_flag

customer\_id

3

16.99

1

9

41.98

1

25

73.43

1

In [68]:

# From almost 70k down to 3301  
targets\_df.info()

<class 'pandas.core.frame.DataFrame'>  
Index: 3301 entries, 3 to 23561  
Data columns (total 2 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 spend\_90\_total 3301 non-null float64  
 1 spend\_90\_flag 3301 non-null int64   
dtypes: float64(1), int64(1)  
memory usage: 77.4 KB

### Make Recency (Date) Features From 'in' Data[¶](#X48022c0d2534e8767219a53521a1414627837c6)

In [69]:

max\_date = temporal\_in\_df['date'].max()

In [70]:

# show the date diff from the last date  
recency\_features\_df = (  
 temporal\_in\_df  
 #.drop(['quantity','price'], axis=1) <- not necessary  
 [['customer\_id','date']]  
 .groupby('customer\_id')  
 .apply(lambda x: (x['date'].max() - max\_date).days, include\_groups=False)  
 .to\_frame(name='recency') # <- changes customer\_id column to become index  
 .set\_axis(['recency'], axis=1)  
)

In [71]:

recency\_features\_df.head(3)

Out[71]:

recency

customer\_id

1

-455

2

-444

3

-127

In [72]:

recency\_features\_df.info()

<class 'pandas.core.frame.DataFrame'>  
Index: 23570 entries, 1 to 23570  
Data columns (total 1 columns):  
 # Column Non-Null Count Dtype  
--- ------ -------------- -----  
 0 recency 23570 non-null int64  
dtypes: int64(1)  
memory usage: 368.3 KB

### Make Frequency (Count) Features From 'in' Data[¶](#Xe23f9508631ff9390e4e42c41fb9b7275ad228b)

In [73]:

# how frequent customers are making purchases  
frequency\_features\_df = (  
 temporal\_in\_df[['customer\_id','date']]  
 .groupby('customer\_id')  
 .count()  
 .set\_axis(['frequency'], axis=1)  
)

In [74]:

frequency\_features\_df.head()

Out[74]:

frequency

customer\_id

1

1

2

2

3

5

4

4

5

11

In [75]:

frequency\_features\_df.info()

<class 'pandas.core.frame.DataFrame'>  
Index: 23570 entries, 1 to 23570  
Data columns (total 1 columns):  
 # Column Non-Null Count Dtype  
--- ------ -------------- -----  
 0 frequency 23570 non-null int64  
dtypes: int64(1)  
memory usage: 368.3 KB

### Make Price (Monetary) Features From 'in' Data[¶](#Xebf69b8b7acf20a421cd1eaca911f3a01a442e0)

In [76]:

# Add total and average spend  
price\_features\_df = (  
 temporal\_in\_df  
 .groupby('customer\_id')  
 .aggregate({'price':['sum','mean']})  
 .set\_axis(['price\_sum','price\_mean'], axis=1)  
)

In [77]:

price\_features\_df.head(3)

Out[77]:

price\_sum

price\_mean

customer\_id

1

11.77

11.770

2

89.00

44.500

3

139.47

27.894

In [78]:

price\_features\_df.info()

<class 'pandas.core.frame.DataFrame'>  
Index: 23570 entries, 1 to 23570  
Data columns (total 2 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 price\_sum 23570 non-null float64  
 1 price\_mean 23570 non-null float64  
dtypes: float64(2)  
memory usage: 552.4 KB

### 3.3 Combine Features[¶](#X8c1f9260f912cb2bc24731a95603881407296d2)

In [79]:

features\_df = (  
 pd.concat(  
 [recency\_features\_df,frequency\_features\_df,price\_features\_df],axis=1  
 )  
 .merge(  
 targets\_df,  
 left\_index=True,  
 right\_index=True,  
 how='left'  
 )  
 .fillna(0) # Customers that did not make a purchase show as NaN otherwise  
)

In [80]:

features\_df.head(3)

Out[80]:

recency

frequency

price\_sum

price\_mean

spend\_90\_total

spend\_90\_flag

customer\_id

1

-455

1

11.77

11.770

0.00

0.0

2

-444

2

89.00

44.500

0.00

0.0

3

-127

5

139.47

27.894

16.99

1.0

In [81]:

features\_df.info()

<class 'pandas.core.frame.DataFrame'>  
Index: 23570 entries, 1 to 23570  
Data columns (total 6 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 recency 23570 non-null int64   
 1 frequency 23570 non-null int64   
 2 price\_sum 23570 non-null float64  
 3 price\_mean 23570 non-null float64  
 4 spend\_90\_total 23570 non-null float64  
 5 spend\_90\_flag 23570 non-null float64  
dtypes: float64(4), int64(2)  
memory usage: 1.3 MB

In [82]:

# Print the index range  
index\_range = features\_df.index.min(), features\_df.index.max()  
print(f"Index range: {index\_range}")

Index range: (1, 23570)

In [83]:

features\_df.spend\_90\_flag.value\_counts()

Out[83]:

spend\_90\_flag  
0.0 20269  
1.0 3301  
Name: count, dtype: int64

The 90 flag class is highly imbalanced.

In [84]:

features\_df.to\_pickle('../artifacts/features\_df.pkl')

# 4.0 MACHINE LEARNING[¶](#X20e206bc87c025ecb5913820e9bb8f709eea7cc)

## Imports[¶](#Imports)

## 4.1 Next 90-Day Spend Prediction (Regression)[¶](#X88d31bd2507ce46929b270b88af921fd19d9f60)

In [85]:

# import cudf  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import GridSearchCV, train\_test\_split, cross\_val\_predict, learning\_curve  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score, classification\_report, roc\_auc\_score  
from xgboost import XGBRegressor, XGBClassifier, plot\_importance  
from catboost import CatBoostRegressor, CatBoostClassifier  
import joblib

In [86]:

# !nvidia-smi

In [87]:

# Data prep  
features\_df = pd.read\_pickle('../artifacts/features\_df.pkl')  
# features\_df = cudf.DataFrame.from\_pandas(pandas\_df)  
y\_reg = features\_df['spend\_90\_total']  
X = features\_df[['recency', 'frequency', 'price\_sum', 'price\_mean']]  
# params  
cv = 10  
n\_jobs = -1  
random\_seed=42

In [88]:

# Define the XGBRegressor with hyperparameters  
xgb\_reg\_spec = XGBRegressor(  
 objective='reg:squarederror',  
 random\_state=random\_seed  
 # tree\_method='hist', # gpu\_hist depreciated  
 # device='cuda', # now use this  
 # gpu\_id=0  
)  
  
xgb\_reg\_param\_grid = {  
 'learning\_rate': [0.01, 0.1, 0.2, 0.3],  
 'n\_estimators': [100, 200, 300],  
 'max\_depth': [3, 4, 5],  
 'subsample': [0.7, 0.8, 0.9, 1.0],  
 'colsample\_bytree': [0.7, 0.8, 0.9, 1.0],  
 'min\_child\_weight': [1, 3, 5],  
 'alpha': [0, 0.1, 0.5, 1], # L1 regularization term on weights  
 'lambda': [1, 1.5, 2, 3] # L2 regularization term on weights  
}  
  
# Grid search for XGBRegressor  
xgb\_reg\_grid\_search = GridSearchCV(  
 estimator=xgb\_reg\_spec,  
 param\_grid=xgb\_reg\_param\_grid,  
 scoring='neg\_mean\_absolute\_error',  
 refit=True,  
 cv=cv,  
 n\_jobs=n\_jobs  
)

In [89]:

# Define the CatBoostRegressor with hyperparameters  
cat\_reg\_spec = CatBoostRegressor(  
 loss\_function='MAE',  
 # task\_type='GPU',  
 random\_seed=random\_seed,  
 verbose=0   
)  
  
cat\_reg\_param\_grid = {  
 'learning\_rate': [0.01, 0.1, 0.2, 0.3],  
 'depth': [3, 4, 5, 6],  
 'iterations': [100, 200, 300],  
 'l2\_leaf\_reg': [1, 3, 5, 7]  
}  
  
# Grid search for CatBoostRegressor  
cat\_reg\_grid\_search = GridSearchCV(  
 estimator=cat\_reg\_spec,  
 param\_grid=cat\_reg\_param\_grid,  
 scoring='neg\_mean\_absolute\_error',  
 refit=True,  
 cv=cv,  
 n\_jobs=n\_jobs  
)

In [90]:

# Fit the model using the training data  
xgb\_reg\_grid\_search.fit(X, y\_reg)  
cat\_reg\_grid\_search.fit(X, y\_reg)

Out[90]:

GridSearchCV(cv=10,  
 estimator=<catboost.core.CatBoostRegressor object at 0x7fcef7c75390>,  
 n\_jobs=-1,  
 param\_grid={'depth': [3, 4, 5, 6], 'iterations': [100, 200, 300],  
 'l2\_leaf\_reg': [1, 3, 5, 7],  
 'learning\_rate': [0.01, 0.1, 0.2, 0.3]},  
 scoring='neg\_mean\_absolute\_error')

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**  
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

  GridSearchCV[?Documentation for GridSearchCV](https://scikit-learn.org/1.5/modules/generated/sklearn.model_selection.GridSearchCV.html)iFitted

GridSearchCV(cv=10,  
 estimator=<catboost.core.CatBoostRegressor object at 0x7fcef7c75390>,  
 n\_jobs=-1,  
 param\_grid={'depth': [3, 4, 5, 6], 'iterations': [100, 200, 300],  
 'l2\_leaf\_reg': [1, 3, 5, 7],  
 'learning\_rate': [0.01, 0.1, 0.2, 0.3]},  
 scoring='neg\_mean\_absolute\_error')

best\_estimator\_: CatBoostRegressor

<catboost.core.CatBoostRegressor object at 0x7fcff885fe90>

CatBoostRegressor

<catboost.core.CatBoostRegressor object at 0x7fcff885fe90>

In [91]:

# Get the best models  
best\_xgb\_reg\_model = xgb\_reg\_grid\_search.best\_estimator\_  
best\_cat\_reg\_model = cat\_reg\_grid\_search.best\_estimator\_  
  
# Save the best models to disk  
joblib.dump(best\_xgb\_reg\_model, '../models/best\_xgb\_reg\_model.pkl')  
joblib.dump(best\_cat\_reg\_model, '../models/best\_cat\_reg\_model.pkl')

Out[91]:

['../models/best\_cat\_reg\_model.pkl']

In [92]:

# Generate predictions using cross-validation  
xgb\_reg\_predictions = cross\_val\_predict(best\_xgb\_reg\_model, X, y\_reg, cv=cv)  
cat\_reg\_predictions = cross\_val\_predict(best\_cat\_reg\_model, X, y\_reg, cv=cv)  
  
# Save predictions  
joblib.dump(xgb\_reg\_predictions, '../artifacts/xgb\_reg\_predictions.pkl')  
joblib.dump(cat\_reg\_predictions, '../artifacts/cat\_reg\_predictions.pkl')

Out[92]:

['../artifacts/cat\_reg\_predictions.pkl']

In [93]:

# Debugging: Check types and shapes  
print(f"xgb\_reg\_predictions type: {type(xgb\_reg\_predictions)}, shape: {xgb\_reg\_predictions.shape}")  
print(f"cat\_reg\_predictions type: {type(cat\_reg\_predictions)}, shape: {cat\_reg\_predictions.shape}")  
print(f"y\_reg type: {type(y\_reg)}, shape: {y\_reg.shape}")

xgb\_reg\_predictions type: <class 'numpy.ndarray'>, shape: (23570,)  
cat\_reg\_predictions type: <class 'numpy.ndarray'>, shape: (23570,)  
y\_reg type: <class 'pandas.core.series.Series'>, shape: (23570,)

In [94]:

# Ensure the data types and lengths match  
print(f"Length of xgb\_reg\_predictions: {len(xgb\_reg\_predictions)}")  
print(f"Length of cat\_reg\_predictions: {len(cat\_reg\_predictions)}")  
print(f"Length of y\_reg: {len(y\_reg)}")

Length of xgb\_reg\_predictions: 23570  
Length of cat\_reg\_predictions: 23570  
Length of y\_reg: 23570

In [95]:

assert len(xgb\_reg\_predictions) == len(y\_reg), "Length of XGB predictions and true values do not match."  
assert len(cat\_reg\_predictions) == len(y\_reg), "Length of CatBoost predictions and true values do not match."

### Regression Interpretation[¶](#Regression-Interpretation)

In [110]:

# To load the models back  
best\_xgb\_reg\_model = joblib.load('../models/best\_xgb\_reg\_model.pkl')  
best\_cat\_reg\_model = joblib.load('../models/best\_cat\_reg\_model.pkl')

In [111]:

# Load Predictions  
xgb\_reg\_predictions = joblib.load('../artifacts/xgb\_reg\_predictions.pkl')  
cat\_reg\_predictions = joblib.load('../artifacts/cat\_reg\_predictions.pkl')

In [112]:

# load features and set y\_reg  
features\_df = joblib.load('../artifacts/features\_df.pkl')  
y\_reg = features\_df['spend\_90\_total']

In [113]:

# Evaluation function  
def evaluate\_model(predictions, true\_values):  
 mae = mean\_absolute\_error(true\_values, predictions)  
 mse = mean\_squared\_error(true\_values, predictions)  
 rmse = np.sqrt(mse)  
 r2 = r2\_score(true\_values, predictions)  
 return mae, mse, rmse, r2

In [114]:

# Evaluate models  
xgb\_mae, xgb\_mse, xgb\_rmse, xgb\_r2 = evaluate\_model(xgb\_reg\_predictions, y\_reg)  
cat\_mae, cat\_mse, cat\_rmse, cat\_r2 = evaluate\_model(cat\_reg\_predictions, y\_reg)  
  
print("XGBRegressor:")  
print(f"MAE: {xgb\_mae:.4f}, MSE: {xgb\_mse:.4f}, RMSE: {xgb\_rmse:.4f}, R²: {xgb\_r2:.4f}")  
print("CatBoostRegressor:")  
print(f"MAE: {cat\_mae:.4f}, MSE: {cat\_mse:.4f}, RMSE: {cat\_rmse:.4f}, R²: {cat\_r2:.4f}")

XGBRegressor:  
MAE: 10.5976, MSE: 1024.4278, RMSE: 32.0067, R²: 0.4625  
CatBoostRegressor:  
MAE: 8.0699, MSE: 1288.0488, RMSE: 35.8894, R²: 0.3242

In [115]:

# Residual plots  
plt.figure(figsize=(14, 6))  
  
plt.subplot(1, 2, 1)  
sns.residplot(x=xgb\_reg\_predictions, y=y\_reg - xgb\_reg\_predictions, lowess=True, color='blue', line\_kws={'color': 'red'})  
plt.title('XGBRegressor Residuals')  
plt.xlabel('Predicted')  
plt.ylabel('Residuals')  
  
plt.subplot(1, 2, 2)  
sns.residplot(x=cat\_reg\_predictions, y=y\_reg - cat\_reg\_predictions, lowess=True, color='green', line\_kws={'color': 'red'})  
plt.title('CatBoostRegressor Residuals')  
plt.xlabel('Predicted')  
plt.ylabel('Residuals')  
  
plt.tight\_layout()  
plt.show()

![No description has been provided for this image](data:image/png;base64;base64,)

In [116]:

# Actual vs Predicted plots  
plt.figure(figsize=(14, 6))  
  
plt.subplot(1, 2, 1)  
plt.scatter(y\_reg, xgb\_reg\_predictions, alpha=0.3, color='blue')  
plt.plot([y\_reg.min(), y\_reg.max()], [y\_reg.min(), y\_reg.max()], 'r--')  
plt.title('XGBRegressor: Actual vs Predicted')  
plt.xlabel('Actual')  
plt.ylabel('Predicted')  
  
plt.subplot(1, 2, 2)  
plt.scatter(y\_reg, cat\_reg\_predictions, alpha=0.3, color='green')  
plt.plot([y\_reg.min(), y\_reg.max()], [y\_reg.min(), y\_reg.max()], 'r--')  
plt.title('CatBoostRegressor: Actual vs Predicted')  
plt.xlabel('Actual')  
plt.ylabel('Predicted')  
  
plt.tight\_layout()  
plt.show()

![No description has been provided for this image](data:image/png;base64;base64,)

In [117]:

# Load the XGBoost model  
best\_xgb\_reg\_model = joblib.load('../models/best\_xgb\_reg\_model.pkl')  
  
# Plot feature importance  
plt.figure(figsize=(10, 6))  
plot\_importance(best\_xgb\_reg\_model, importance\_type='weight')  
plt.title('XGBoost Feature Importance')  
plt.show()

<Figure size 1000x600 with 0 Axes>

![No description has been provided for this image](data:image/png;base64;base64,)

In [118]:

# Plot XGBoost feature importance  
plt.figure(figsize=(10, 6))  
plot\_importance(best\_xgb\_reg\_model, importance\_type='weight')  
plt.title('XGBoost Feature Importance')  
plt.show()  
  
# Get CatBoost feature importance  
feature\_importances = best\_cat\_reg\_model.get\_feature\_importance()  
feature\_names = best\_cat\_reg\_model.feature\_names\_  
  
# Create a DataFrame for visualization  
importance\_df = pd.DataFrame({'Feature': feature\_names, 'Importance': feature\_importances})  
  
# Plot CatBoost feature importance  
plt.figure(figsize=(10, 6))  
importance\_df.sort\_values(by='Importance', ascending=False).plot(kind='bar', x='Feature', y='Importance', legend=False, ax=plt.gca())  
plt.title('CatBoost Feature Importance')  
plt.show()

<Figure size 1000x600 with 0 Axes>

![No description has been provided for this image](data:image/png;base64;base64,)

![No description has been provided for this image](data:image/png;base64;base64,)

XGBRegressor:

* MAE: 10.5976
* MSE: 1024.4278
* RMSE: 32.0067
* R²: 0.4625

CatBoostRegressor:

* MAE: 8.0699
* MSE: 1288.0488
* RMSE: 35.8894
* R²: 0.3242

Comparison:

1. Mean Absolute Error (MAE):
   * CatBoostRegressor has a lower MAE (8.0699) compared to XGBRegressor (10.5976), indicating that CatBoost has, on average, smaller absolute errors.
2. Mean Squared Error (MSE):
   * XGBRegressor has a lower MSE (1024.4278) compared to CatBoostRegressor (1288.0488), indicating that XGB has smaller squared errors overall.
3. Root Mean Squared Error (RMSE): \*XGBRegressor has a lower RMSE (32.0067) compared to CatBoostRegressor (35.8894), indicating that XGB has fewer large errors compared to CatBoost.
4. R² (R-squared):
   * XGBRegressor has a higher R² (0.4625) compared to CatBoostRegressor (0.3242), indicating that XGB explains more of the variance in the target variable.

Conclusion:

MAE suggests that CatBoost has better average prediction accuracy. MSE and RMSE suggest that XGBoost handles large errors better than CatBoost. R² indicates that XGBoost explains more variance in the data than CatBoost. Given these metrics, XGBRegressor seems to be the better-performing model overall, especially considering its ability to handle large errors and explain variance in the target variable.

## 4.2 XGBoost Regression Parameter Tuning[¶](#Xccf60d254ab2e58d7eef9502e347a843f0b7a7c)

Bayesian Optimization

In [119]:

from bayes\_opt import BayesianOptimization  
from sklearn.model\_selection import cross\_val\_score, cross\_val\_predict  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score  
from xgboost import XGBRegressor  
import joblib  
import numpy as np  
import pandas as pd

In [120]:

# Data prep  
features\_df = pd.read\_pickle('../artifacts/features\_df.pkl')  
# features\_df = cudf.DataFrame.from\_pandas(pandas\_df)  
y\_reg = features\_df['spend\_90\_total']  
X = features\_df[['recency', 'frequency', 'price\_sum', 'price\_mean']]  
# params  
cv = 10  
n\_jobs = -1  
random\_seed=42

In [121]:

def xgb\_evaluate(max\_depth, gamma, subsample, colsample\_bytree, learning\_rate, alpha, lambda\_, min\_child\_weight, scale\_pos\_weight):  
 params = {  
 'max\_depth': int(max\_depth),  
 'gamma': gamma,  
 'subsample': subsample,  
 'colsample\_bytree': colsample\_bytree,  
 'learning\_rate': learning\_rate,  
 'n\_estimators': 100,  
 'min\_child\_weight': min\_child\_weight,  
 'alpha': alpha,  
 'lambda': lambda\_,  
 'scale\_pos\_weight': scale\_pos\_weight,  
 'objective': 'reg:squarederror',  
 'eval\_metric': 'rmse'  
 }  
 xgb\_model = XGBRegressor(\*\*params)  
 cv\_result = cross\_val\_score(xgb\_model, X, y\_reg, cv=10, scoring='neg\_mean\_absolute\_error')  
 return np.mean(cv\_result)  
  
# Set the parameter bounds  
param\_bounds = {  
 'max\_depth': (3, 10),  
 'gamma': (0, 0.5),  
 'subsample': (0.6, 1.0),  
 'colsample\_bytree': (0.6, 1.0),  
 'learning\_rate': (0.01, 0.3),  
 'alpha': (0, 1),  
 'lambda\_': (1, 3),  
 'min\_child\_weight': (1, 10),  
 'scale\_pos\_weight': (1, 3)  
}  
  
# Initialize Bayesian Optimization  
optimizer = BayesianOptimization(f=xgb\_evaluate, pbounds=param\_bounds, random\_state=42)  
  
# Optimize  
optimizer.maximize(init\_points=10, n\_iter=50)  
  
# Get the best parameters  
best\_params\_bayes = optimizer.max['params']  
best\_params\_bayes['max\_depth'] = int(best\_params\_bayes['max\_depth']) # convert max\_depth to int  
print(f"Best parameters found by Bayesian Optimization: {best\_params\_bayes}")  
  
# Fit the model with the best parameters  
best\_xgb\_model\_bayes = XGBRegressor(\*\*best\_params\_bayes, objective='reg:squarederror', n\_estimators=100)  
best\_xgb\_model\_bayes.fit(X, y\_reg)  
  
# Save the best model  
joblib.dump(best\_xgb\_model\_bayes, '../models/best\_xgb\_reg\_model\_bayes\_tuned.pkl')  
  
# Generate predictions using cross-validation  
xgb\_reg\_predictions\_bayes\_tuned = cross\_val\_predict(best\_xgb\_model\_bayes, X, y\_reg, cv=10)  
  
# Save the predictions  
joblib.dump(xgb\_reg\_predictions\_bayes\_tuned, '../artifacts/xgb\_reg\_predictions\_bayes\_tuned.pkl')

| iter | target | alpha | colsam... | gamma | lambda\_ | learni... | max\_depth | min\_ch... | scale\_... | subsample |  
-------------------------------------------------------------------------------------------------------------------------------------  
| 1 | -10.73 | 0.3745 | 0.9803 | 0.366 | 2.197 | 0.05525 | 4.092 | 1.523 | 2.732 | 0.8404 |  
| 2 | -10.72 | 0.7081 | 0.6082 | 0.485 | 2.665 | 0.07158 | 4.273 | 2.651 | 1.608 | 0.8099 |  
| 3 | -10.83 | 0.4319 | 0.7165 | 0.3059 | 1.279 | 0.09472 | 5.565 | 5.105 | 2.57 | 0.6799 |  
| 4 | -10.81 | 0.5142 | 0.837 | 0.02323 | 2.215 | 0.05945 | 3.455 | 9.54 | 2.931 | 0.9234 |  
| 5 | -10.81 | 0.3046 | 0.6391 | 0.3421 | 1.88 | 0.04539 | 6.466 | 1.309 | 2.819 | 0.7035 |  
| 6 | -10.99 | 0.6625 | 0.7247 | 0.26 | 2.093 | 0.06361 | 9.787 | 7.976 | 2.879 | 0.9579 |  
| 7 | -11.0 | 0.5979 | 0.9687 | 0.04425 | 1.392 | 0.02312 | 5.277 | 4.498 | 1.543 | 0.9315 |  
| 8 | -10.97 | 0.3568 | 0.7124 | 0.2713 | 1.282 | 0.2426 | 3.522 | 9.882 | 2.544 | 0.6795 |  
| 9 | -10.81 | 0.005522 | 0.9262 | 0.3534 | 2.458 | 0.2337 | 3.518 | 4.226 | 1.232 | 0.9452 |  
| 10 | -11.12 | 0.6233 | 0.7324 | 0.03178 | 1.622 | 0.1043 | 8.107 | 6.738 | 2.774 | 0.7889 |  
| 11 | -10.78 | 0.0 | 0.8218 | 0.5 | 2.767 | 0.2369 | 3.472 | 2.427 | 2.227 | 0.8117 |  
| 12 | -10.74 | 0.7841 | 0.6 | 0.5 | 3.0 | 0.05267 | 4.727 | 1.347 | 1.849 | 0.6585 |  
| 13 | -12.45 | 1.0 | 0.6 | 0.5 | 1.752 | 0.01 | 3.706 | 1.428 | 1.26 | 1.0 |  
| 14 | -10.73 | 0.2395 | 0.8836 | 0.2901 | 1.174 | 0.0549 | 5.52 | 5.117 | 2.543 | 0.7749 |  
| 15 | -10.73 | 0.3657 | 0.8099 | 0.421 | 2.964 | 0.09841 | 4.604 | 2.194 | 2.477 | 0.6822 |  
| 16 | -10.93 | 0.03846 | 0.6 | 0.5 | 3.0 | 0.2032 | 4.378 | 3.449 | 1.978 | 0.6 |  
| 17 | -10.74 | 0.3139 | 0.8651 | 0.3884 | 2.783 | 0.07701 | 5.082 | 1.068 | 2.934 | 0.6328 |  
| 18 | -12.21 | 0.9192 | 0.6 | 0.4696 | 3.0 | 0.01001 | 5.813 | 2.011 | 2.099 | 0.6565 |  
| 19 | -10.82 | 0.1998 | 0.7721 | 0.4657 | 3.0 | 0.1557 | 4.247 | 1.35 | 2.45 | 0.6 |  
| 20 | -12.34 | 0.8621 | 0.9771 | 0.2251 | 3.0 | 0.01 | 3.871 | 2.321 | 2.399 | 0.6982 |  
| 21 | -10.88 | 0.08432 | 0.8621 | 0.3272 | 2.661 | 0.2872 | 3.336 | 2.248 | 2.1 | 0.9048 |  
| 22 | -10.86 | 0.7346 | 0.7895 | 0.1038 | 1.657 | 0.2047 | 4.756 | 4.234 | 1.708 | 0.6994 |  
| 23 | -10.7 | 0.2276 | 0.8222 | 0.4637 | 2.554 | 0.09663 | 4.721 | 1.629 | 2.635 | 0.7586 |  
| 24 | -10.95 | 0.0 | 0.6 | 0.5 | 2.629 | 0.2312 | 4.485 | 2.092 | 2.015 | 0.7844 |  
| 25 | -11.46 | 0.7717 | 0.9446 | 0.1846 | 1.843 | 0.2704 | 8.605 | 9.328 | 2.451 | 0.8682 |  
| 26 | -11.18 | 0.4623 | 0.7567 | 0.02664 | 1.141 | 0.2106 | 7.342 | 1.419 | 1.832 | 0.9639 |  
| 27 | -11.0 | 0.484 | 0.6262 | 0.2096 | 2.099 | 0.09615 | 7.377 | 8.702 | 1.262 | 0.7813 |  
| 28 | -11.02 | 0.4746 | 0.695 | 0.3495 | 2.299 | 0.269 | 4.851 | 1.012 | 2.636 | 0.7144 |  
| 29 | -11.39 | 0.3941 | 0.6084 | 0.379 | 2.524 | 0.2769 | 6.995 | 3.374 | 1.288 | 0.8222 |  
| 30 | -10.86 | 0.1186 | 0.6 | 0.5 | 2.57 | 0.2223 | 3.917 | 2.897 | 1.568 | 0.892 |  
| 31 | -11.49 | 0.441 | 0.9933 | 0.4269 | 2.956 | 0.01532 | 8.161 | 9.2 | 2.404 | 0.9861 |  
| 32 | -10.86 | 0.0 | 0.9126 | 0.4999 | 3.0 | 0.2137 | 4.726 | 1.815 | 3.0 | 0.6 |  
| 33 | -10.75 | 0.6721 | 0.6 | 0.4973 | 2.394 | 0.106 | 4.489 | 3.298 | 1.392 | 0.8051 |  
| 34 | -12.24 | 0.0 | 0.6 | 0.5 | 2.368 | 0.01 | 4.266 | 1.946 | 3.0 | 1.0 |  
| 35 | -10.75 | 0.3788 | 0.8954 | 0.4447 | 2.989 | 0.1267 | 4.83 | 1.551 | 2.427 | 0.6 |  
| 36 | -10.83 | 0.7869 | 0.62 | 0.1698 | 2.055 | 0.03522 | 4.025 | 4.597 | 1.784 | 0.7702 |  
| 37 | -11.8 | 0.2184 | 0.6175 | 0.1902 | 1.368 | 0.255 | 8.68 | 8.065 | 1.549 | 0.7471 |  
| 38 | -12.1 | 0.4572 | 1.0 | 0.5 | 2.454 | 0.01 | 4.456 | 1.402 | 2.267 | 0.6 |  
| 39 | -10.9 | 0.7798 | 0.7604 | 0.4895 | 2.523 | 0.02707 | 4.464 | 2.961 | 1.767 | 0.8899 |  
| 40 | -10.87 | 0.4393 | 0.724 | 0.4797 | 1.855 | 0.1527 | 4.308 | 5.681 | 2.427 | 0.6871 |  
| 41 | -10.8 | 0.3944 | 0.8018 | 0.09051 | 2.311 | 0.1001 | 4.275 | 8.712 | 2.222 | 0.6337 |  
| 42 | -11.18 | 0.09894 | 0.8263 | 0.3418 | 1.528 | 0.155 | 7.285 | 9.416 | 1.875 | 0.6929 |  
| 43 | -11.57 | 0.581 | 0.8486 | 0.118 | 2.919 | 0.01403 | 7.144 | 6.802 | 2.512 | 0.797 |  
| 44 | -10.91 | 0.2842 | 0.7714 | 0.3593 | 2.811 | 0.2523 | 4.613 | 1.979 | 2.702 | 0.8078 |  
| 45 | -10.93 | 0.7553 | 0.7566 | 0.01666 | 1.771 | 0.2539 | 4.938 | 4.215 | 1.823 | 0.6405 |  
| 46 | -10.85 | 0.4273 | 0.6 | 0.5 | 2.686 | 0.1677 | 4.348 | 2.994 | 1.457 | 0.6891 |  
| 47 | -10.81 | 0.3621 | 0.8132 | 0.3928 | 2.855 | 0.2067 | 4.582 | 2.133 | 2.441 | 0.7954 |  
| 48 | -11.03 | 0.7379 | 0.6937 | 0.3464 | 2.744 | 0.1116 | 7.461 | 6.163 | 1.703 | 0.6783 |  
| 49 | -10.8 | 0.1108 | 0.8605 | 0.05836 | 1.356 | 0.05611 | 6.258 | 9.247 | 2.836 | 0.9 |  
| 50 | -11.01 | 0.1035 | 0.8377 | 0.1983 | 2.269 | 0.1952 | 6.934 | 8.857 | 2.225 | 0.9974 |  
| 51 | -10.96 | 0.4355 | 0.6801 | 0.0997 | 1.526 | 0.2809 | 3.782 | 9.313 | 1.431 | 0.8644 |  
| 52 | -10.77 | 0.5788 | 0.8813 | 0.2415 | 2.988 | 0.08442 | 5.854 | 4.112 | 1.996 | 0.7841 |  
| 53 | -10.95 | 0.4951 | 0.7517 | 0.4928 | 2.699 | 0.2029 | 5.683 | 7.146 | 2.17 | 0.8633 |  
| 54 | -10.89 | 0.4701 | 0.6899 | 0.2176 | 2.863 | 0.1127 | 5.028 | 1.556 | 2.998 | 0.7492 |  
| 55 | -10.88 | 0.04077 | 0.7042 | 0.4621 | 2.89 | 0.2062 | 4.892 | 1.397 | 2.67 | 0.7296 |  
| 56 | -11.06 | 0.1418 | 0.7409 | 0.1918 | 1.809 | 0.06869 | 9.444 | 2.554 | 2.37 | 0.6113 |  
| 57 | -10.67 | 0.4926 | 1.0 | 0.1998 | 1.984 | 0.1426 | 3.854 | 1.303 | 2.912 | 0.8897 |  
| 58 | -11.44 | 0.8964 | 0.7372 | 0.05708 | 1.657 | 0.2723 | 6.179 | 1.972 | 1.18 | 0.731 |  
| 59 | -10.75 | 0.4027 | 1.0 | 0.5 | 3.0 | 0.2046 | 4.653 | 1.349 | 2.95 | 0.6378 |  
| 60 | -10.73 | 0.02271 | 0.8846 | 0.2957 | 2.845 | 0.04439 | 4.953 | 2.222 | 2.402 | 0.7613 |  
=====================================================================================================================================  
Best parameters found by Bayesian Optimization: {'alpha': 0.49259245903377535, 'colsample\_bytree': 1.0, 'gamma': 0.1998203246565923, 'lambda\_': 1.9844260653862364, 'learning\_rate': 0.14258711885517153, 'max\_depth': 3, 'min\_child\_weight': 1.303195393263668, 'scale\_pos\_weight': 2.911633258171247, 'subsample': 0.8896653182904212}

/home/blackitalian/miniconda3/envs/cdnow/lib/python3.11/site-packages/xgboost/core.py:160: UserWarning: [22:56:23] WARNING: /home/conda/feedstock\_root/build\_artifacts/xgboost-split\_1717021965948/work/src/learner.cc:742:   
Parameters: { "lambda\_" } are not used.  
  
/home/blackitalian/miniconda3/envs/cdnow/lib/python3.11/site-packages/xgboost/core.py:160: UserWarning: [22:56:23] WARNING: /home/conda/feedstock\_root/build\_artifacts/xgboost-split\_1717021965948/work/src/learner.cc:742:   
Parameters: { "lambda\_" } are not used.  
  
/home/blackitalian/miniconda3/envs/cdnow/lib/python3.11/site-packages/xgboost/core.py:160: UserWarning: [22:56:24] WARNING: /home/conda/feedstock\_root/build\_artifacts/xgboost-split\_1717021965948/work/src/learner.cc:742:   
Parameters: { "lambda\_" } are not used.  
  
/home/blackitalian/miniconda3/envs/cdnow/lib/python3.11/site-packages/xgboost/core.py:160: UserWarning: [22:56:25] WARNING: /home/conda/feedstock\_root/build\_artifacts/xgboost-split\_1717021965948/work/src/learner.cc:742:   
Parameters: { "lambda\_" } are not used.  
  
/home/blackitalian/miniconda3/envs/cdnow/lib/python3.11/site-packages/xgboost/core.py:160: UserWarning: [22:56:25] WARNING: /home/conda/feedstock\_root/build\_artifacts/xgboost-split\_1717021965948/work/src/learner.cc:742:   
Parameters: { "lambda\_" } are not used.  
  
/home/blackitalian/miniconda3/envs/cdnow/lib/python3.11/site-packages/xgboost/core.py:160: UserWarning: [22:56:26] WARNING: /home/conda/feedstock\_root/build\_artifacts/xgboost-split\_1717021965948/work/src/learner.cc:742:   
Parameters: { "lambda\_" } are not used.  
  
/home/blackitalian/miniconda3/envs/cdnow/lib/python3.11/site-packages/xgboost/core.py:160: UserWarning: [22:56:26] WARNING: /home/conda/feedstock\_root/build\_artifacts/xgboost-split\_1717021965948/work/src/learner.cc:742:   
Parameters: { "lambda\_" } are not used.  
  
/home/blackitalian/miniconda3/envs/cdnow/lib/python3.11/site-packages/xgboost/core.py:160: UserWarning: [22:56:26] WARNING: /home/conda/feedstock\_root/build\_artifacts/xgboost-split\_1717021965948/work/src/learner.cc:742:   
Parameters: { "lambda\_" } are not used.  
  
/home/blackitalian/miniconda3/envs/cdnow/lib/python3.11/site-packages/xgboost/core.py:160: UserWarning: [22:56:26] WARNING: /home/conda/feedstock\_root/build\_artifacts/xgboost-split\_1717021965948/work/src/learner.cc:742:   
Parameters: { "lambda\_" } are not used.  
  
/home/blackitalian/miniconda3/envs/cdnow/lib/python3.11/site-packages/xgboost/core.py:160: UserWarning: [22:56:26] WARNING: /home/conda/feedstock\_root/build\_artifacts/xgboost-split\_1717021965948/work/src/learner.cc:742:   
Parameters: { "lambda\_" } are not used.  
  
/home/blackitalian/miniconda3/envs/cdnow/lib/python3.11/site-packages/xgboost/core.py:160: UserWarning: [22:56:26] WARNING: /home/conda/feedstock\_root/build\_artifacts/xgboost-split\_1717021965948/work/src/learner.cc:742:   
Parameters: { "lambda\_" } are not used.

Out[121]:

['../artifacts/xgb\_reg\_predictions\_bayes\_tuned.pkl']

In [122]:

# Evaluation function  
def evaluate\_model(predictions, true\_values):  
 mae = mean\_absolute\_error(true\_values, predictions)  
 mse = mean\_squared\_error(true\_values, predictions)  
 rmse = np.sqrt(mse)  
 r2 = r2\_score(true\_values, predictions)  
 return mae, mse, rmse, r2  
  
# Evaluate the tuned model  
xgb\_mae\_bayes\_tuned, xgb\_mse\_bayes\_tuned, xgb\_rmse\_bayes\_tuned, xgb\_r2\_bayes\_tuned = evaluate\_model(xgb\_reg\_predictions\_bayes\_tuned, y\_reg)  
  
print("Bayesian Tuned XGBRegressor:")  
print(f"MAE: {xgb\_mae\_bayes\_tuned:.4f}, MSE: {xgb\_mse\_bayes\_tuned:.4f}, RMSE: {xgb\_rmse\_bayes\_tuned:.4f}, R²: {xgb\_r2\_bayes\_tuned:.4f}")

Bayesian Tuned XGBRegressor:  
MAE: 10.6487, MSE: 1057.9243, RMSE: 32.5257, R²: 0.4449

### RandomizedSearchCV for thorough hyperparameter tuning[¶](#X2b4fac548ff9e576f9d991cc55d85031a9cbcc6)

In [123]:

from sklearn.model\_selection import RandomizedSearchCV  
# Define the parameter grid for RandomizedSearchCV  
param\_grid = {  
 'learning\_rate': [0.01, 0.05, 0.1, 0.2, 0.3],  
 'n\_estimators': [100, 200, 300, 500],  
 'max\_depth': [3, 4, 5, 6, 8],  
 'min\_child\_weight': [1, 3, 5, 7],  
 'gamma': [0, 0.1, 0.2, 0.3, 0.4],  
 'subsample': [0.6, 0.7, 0.8, 0.9, 1.0],  
 'colsample\_bytree': [0.6, 0.7, 0.8, 0.9, 1.0],  
 'alpha': [0, 0.1, 0.5, 1],  
 'lambda': [1, 1.5, 2, 3],  
 'scale\_pos\_weight': [1, 1.5, 2, 3]  
}  
  
# Initialize the XGBRegressor  
xgb\_model = XGBRegressor(objective='reg:squarederror', random\_state=42)  
  
# Initialize RandomizedSearchCV  
random\_search = RandomizedSearchCV(estimator=xgb\_model, param\_distributions=param\_grid, n\_iter=100,   
 scoring='neg\_mean\_absolute\_error', cv=10, verbose=1, random\_state=42, n\_jobs=-1)  
  
# Fit the model  
random\_search.fit(X, y\_reg)  
  
# Get the best model  
best\_xgb\_model\_random = random\_search.best\_estimator\_  
  
# Save the best model  
joblib.dump(best\_xgb\_model\_random, '../models/best\_xgb\_reg\_model\_random\_tuned.pkl')  
  
# Generate predictions using cross-validation  
xgb\_reg\_predictions\_random\_tuned = cross\_val\_predict(best\_xgb\_model\_random, X, y\_reg, cv=10)  
  
# Save the predictions  
joblib.dump(xgb\_reg\_predictions\_random\_tuned, '../artifacts/xgb\_reg\_predictions\_random\_tuned.pkl')

Fitting 10 folds for each of 100 candidates, totalling 1000 fits

Out[123]:

['../artifacts/xgb\_reg\_predictions\_random\_tuned.pkl']

In [124]:

# Evaluate the tuned model  
xgb\_mae\_random\_tuned, xgb\_mse\_random\_tuned, xgb\_rmse\_random\_tuned, xgb\_r2\_random\_tuned = evaluate\_model(xgb\_reg\_predictions\_random\_tuned, y\_reg)  
  
print("Randomized Search Tuned XGBRegressor:")  
print(f"MAE: {xgb\_mae\_random\_tuned:.4f}, MSE: {xgb\_mse\_random\_tuned:.4f}, RMSE: {xgb\_rmse\_random\_tuned:.4f}, R²: {xgb\_r2\_random\_tuned:.4f}")

Randomized Search Tuned XGBRegressor:  
MAE: 10.6532, MSE: 1078.7137, RMSE: 32.8438, R²: 0.4340

Neither of the two models out perform the initial XGBoost model.

## 4.3 Ensemble[¶](#X209124b07a3486a842eee3ad160c807be05dac6)

In [125]:

import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split, GridSearchCV, KFold  
from sklearn.ensemble import StackingRegressor  
from sklearn.linear\_model import Ridge  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score  
from xgboost import XGBRegressor  
from catboost import CatBoostRegressor  
import joblib

In [126]:

# Data prep  
features\_df = pd.read\_pickle('../artifacts/features\_df.pkl')  
# features\_df = cudf.DataFrame.from\_pandas(pandas\_df)  
y\_reg = features\_df['spend\_90\_total']  
X = features\_df[['recency', 'frequency', 'price\_sum', 'price\_mean']]

In [127]:

# Split data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_reg, test\_size=0.2, random\_state=42)  
  
# Initialize base models with hyperparameter tuning  
xgb\_model = XGBRegressor(objective='reg:squarederror', random\_state=42)  
catboost\_model = CatBoostRegressor(loss\_function='MAE', random\_seed=42, verbose=0)  
  
# Define hyperparameters for grid search  
xgb\_param\_grid = {  
 'learning\_rate': [0.01, 0.1, 0.2],  
 'max\_depth': [3, 5, 7],  
 'n\_estimators': [100, 200, 300]  
}  
  
catboost\_param\_grid = {  
 'learning\_rate': [0.01, 0.1, 0.2],  
 'depth': [3, 5, 7],  
 'iterations': [100, 200, 300]  
}  
  
# Perform grid search for each model  
xgb\_grid\_search = GridSearchCV(estimator=xgb\_model, param\_grid=xgb\_param\_grid, scoring='neg\_mean\_absolute\_error', cv=5, n\_jobs=-1)  
catboost\_grid\_search = GridSearchCV(estimator=catboost\_model, param\_grid=catboost\_param\_grid, scoring='neg\_mean\_absolute\_error', cv=5, n\_jobs=-1)  
  
# Fit grid search  
xgb\_grid\_search.fit(X\_train, y\_train)  
catboost\_grid\_search.fit(X\_train, y\_train)  
  
# Get best models  
best\_xgb\_model = xgb\_grid\_search.best\_estimator\_  
best\_catboost\_model = catboost\_grid\_search.best\_estimator\_  
  
# Initialize stacking model with a ridge regression meta-model  
stacking\_model = StackingRegressor(  
 estimators=[  
 ('xgb', best\_xgb\_model),  
 ('catboost', best\_catboost\_model)  
 ],  
 final\_estimator=Ridge(alpha=1.0),  
 cv=5  
)  
  
# Fit the stacking model  
stacking\_model.fit(X\_train, y\_train)  
  
# Save the stacking model  
joblib.dump(stacking\_model, '../models/stacking\_model\_tuned.pkl')  
  
# Predict on the test set  
y\_pred\_test = stacking\_model.predict(X\_test)

In [128]:

# Evaluate model  
def evaluate\_model(y\_true, y\_pred):  
 mae = mean\_absolute\_error(y\_true, y\_pred)  
 mse = mean\_squared\_error(y\_true, y\_pred)  
 rmse = np.sqrt(mse)  
 r2 = r2\_score(y\_true, y\_pred)  
 return mae, mse, rmse, r2  
  
test\_mae, test\_mse, test\_rmse, test\_r2 = evaluate\_model(y\_test, y\_pred\_test)  
  
print("Tuned Stacking Model Test Metrics:")  
print(f"MAE: {test\_mae:.4f}, MSE: {test\_mse:.4f}, RMSE: {test\_rmse:.4f}, R²: {test\_r2:.4f}")

Tuned Stacking Model Test Metrics:  
MAE: 10.9942, MSE: 1088.4156, RMSE: 32.9911, R²: 0.5656

In [129]:

# Predict and evaluate on the training set (optional, to check for overfitting)  
y\_pred\_train = stacking\_model.predict(X\_train)  
train\_mae, train\_mse, train\_rmse, train\_r2 = evaluate\_model(y\_train, y\_pred\_train)  
  
print("Tuned Stacking Model Train Metrics:")  
print(f"MAE: {train\_mae:.4f}, MSE: {train\_mse:.4f}, RMSE: {train\_rmse:.4f}, R²: {train\_r2:.4f}")

Tuned Stacking Model Train Metrics:  
MAE: 10.4301, MSE: 780.3653, RMSE: 27.9350, R²: 0.5556

#### Ensemble 2[¶](#Ensemble-2)

In [130]:

best\_xgb\_reg\_model = joblib.load('../models/best\_xgb\_reg\_model.pkl')  
stacking\_model = joblib.load('../models/stacking\_model\_tuned.pkl')

In [131]:

# Combining predictions from both models  
xgb\_predictions = best\_xgb\_reg\_model.predict(X\_test)  
stacking\_predictions = stacking\_model.predict(X\_test)  
  
# Weighted average of predictions  
combined\_predictions = (xgb\_predictions + stacking\_predictions) / 2  
  
# Evaluate combined model  
combined\_mae, combined\_mse, combined\_rmse, combined\_r2 = evaluate\_model(y\_test, combined\_predictions)  
  
print("Combined Model Test Metrics:")  
print(f"MAE: {combined\_mae:.4f}, MSE: {combined\_mse:.4f}, RMSE: {combined\_rmse:.4f}, R²: {combined\_r2:.4f}")

Combined Model Test Metrics:  
MAE: 10.4917, MSE: 808.8655, RMSE: 28.4406, R²: 0.6772

## 4.4 Final Regression Model[¶](#X6058aea79802044a2d812aaf598b268500f3b50)

In [132]:

import pandas as pd  
import numpy as np  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score  
import joblib  
  
# Data preparation  
features\_df = pd.read\_pickle('../artifacts/features\_df.pkl')  
y\_reg = features\_df['spend\_90\_total']  
X = features\_df[['recency', 'frequency', 'price\_sum', 'price\_mean']]  
  
# Load the best models  
best\_xgb\_reg\_model = joblib.load('../models/best\_xgb\_reg\_model.pkl')  
stacking\_model = joblib.load('../models/stacking\_model.pkl')  
  
# Combine predictions from both models  
xgb\_predictions = best\_xgb\_reg\_model.predict(X)  
stacking\_predictions = stacking\_model.predict(X)  
combined\_predictions = (xgb\_predictions + stacking\_predictions) / 2  
  
# Save combined predictions to a separate file  
combined\_reg\_predictions\_df = pd.DataFrame({'combined\_predictions': combined\_predictions})  
combined\_reg\_predictions\_df.to\_pickle('../artifacts/combined\_reg\_predictions\_final.pkl')  
  
# Evaluate combined model  
def evaluate\_model(y\_true, y\_pred):  
 mae = mean\_absolute\_error(y\_true, y\_pred)  
 mse = mean\_squared\_error(y\_true, y\_pred)  
 rmse = np.sqrt(mse)  
 r2 = r2\_score(y\_true, y\_pred)  
 return mae, mse, rmse, r2  
  
combined\_mae, combined\_mse, combined\_rmse, combined\_r2 = evaluate\_model(y\_reg, combined\_predictions)  
  
print("Combined Model Metrics on All Data:")  
print(f"MAE: {combined\_mae:.4f}, MSE: {combined\_mse:.4f}, RMSE: {combined\_rmse:.4f}, R²: {combined\_r2:.4f}")

Combined Model Metrics on All Data:  
MAE: 9.9434, MSE: 695.0559, RMSE: 26.3639, R²: 0.6353

#### Feature Importance[¶](#Feature-Importance)

In [133]:

import pandas as pd  
import matplotlib.pyplot as plt  
from xgboost import plot\_importance  
import joblib

In [134]:

# Load the best XGBoost model  
best\_xgb\_reg\_model = joblib.load('../models/best\_xgb\_reg\_model.pkl')  
  
# Plot feature importance for the XGBoost model  
plt.figure(figsize=(10, 6))  
plot\_importance(best\_xgb\_reg\_model, importance\_type='weight')  
plt.title('XGBoost Feature Importance')  
plt.show()

<Figure size 1000x600 with 0 Axes>

![No description has been provided for this image](data:image/png;base64;base64,)

In [135]:

# Load the stacking model  
stacking\_model = joblib.load('../models/stacking\_model.pkl')  
  
# Assuming the stacking model uses XGBoost and CatBoost as base models  
xgb\_model = stacking\_model.named\_estimators\_['xgb']  
catboost\_model = stacking\_model.named\_estimators\_['catboost']  
  
# Plot feature importance for the XGBoost base model in the stacking model  
plt.figure(figsize=(10, 6))  
plot\_importance(xgb\_model, importance\_type='weight')  
plt.title('XGBoost Base Model Feature Importance in Stacking')  
plt.show()  
  
# Plot feature importance for the CatBoost base model  
# CatBoost doesn't have a direct plot\_importance function, so we extract and plot manually  
feature\_importances = catboost\_model.get\_feature\_importance()  
features = ['recency', 'frequency', 'price\_sum', 'price\_mean'] # Replace with actual feature names  
importance\_df = pd.DataFrame({'Feature': features, 'Importance': feature\_importances})  
  
plt.figure(figsize=(10, 6))  
importance\_df.sort\_values(by='Importance', ascending=False).plot(kind='bar', x='Feature', y='Importance', legend=False)  
plt.title('CatBoost Base Model Feature Importance in Stacking')  
plt.show()

<Figure size 1000x600 with 0 Axes>

![No description has been provided for this image](data:image/png;base64;base64,)

<Figure size 1000x600 with 0 Axes>

![No description has been provided for this image](data:image/png;base64;base64,)

#### Final Regression Assessment[¶](#Final-Regression-Assessment)

Given the metrics, the Combined Model on all data demonstrates superior performance in terms of prediction accuracy (MAE, MSE, RMSE) compared to both the previous ensemble models on the test set and the original XGBoost model. Despite having a slightly lower R² than the previous best combined model on the test set, its overall performance is robust and indicates that it is the best performing model.

## 4.5 Next 90-Day Spend Probability (Binary Classification)[¶](#Xabfb3d34b9e70439cde9be2eeddfbd397b53ddc)

In [182]:

import pandas as pd  
from sklearn.model\_selection import train\_test\_split, GridSearchCV  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, roc\_curve  
import matplotlib.pyplot as plt  
import joblib  
from xgboost import XGBClassifier  
from catboost import CatBoostClassifier

In [183]:

features\_df = pd.read\_pickle('../artifacts/features\_df.pkl')  
X = features\_df[['recency','frequency','price\_sum','price\_mean']]  
y\_class = features\_df['spend\_90\_flag']

In [184]:

# Split data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_class, test\_size=0.2, random\_state=42)

In [185]:

# Define the XGBClassifier with hyperparameters  
xgb\_class\_spec = XGBClassifier(  
 objective='binary:logistic',  
 random\_state=42  
)  
  
xgb\_class\_param\_grid = {  
 'learning\_rate': [0.01, 0.1, 0.2, 0.3],  
 'n\_estimators': [100, 200, 300],  
 'max\_depth': [3, 4, 5],  
 'subsample': [0.7, 0.8, 0.9, 1.0],  
 'colsample\_bytree': [0.7, 0.8, 0.9, 1.0],  
 'min\_child\_weight': [1, 3, 5],  
 'alpha': [0, 0.1, 0.5, 1],  
 'lambda': [1, 1.5, 2, 3]  
}  
  
# Grid search for XGBClassifier  
xgb\_class\_grid\_search = GridSearchCV(  
 estimator=xgb\_class\_spec,  
 param\_grid=xgb\_class\_param\_grid,  
 scoring='roc\_auc',  
 refit=True,  
 cv=5,  
 n\_jobs=-1  
)  
  
# Fit the XGBClassifier grid search  
xgb\_class\_grid\_search.fit(X\_train, y\_train)  
best\_xgb\_class\_model = xgb\_class\_grid\_search.best\_estimator\_  
  
# Save the best XGBClassifier model  
joblib.dump(best\_xgb\_class\_model, '../models/best\_xgb\_class\_model\_final.pkl')

Out[185]:

['../models/best\_xgb\_class\_model\_final.pkl']

In [186]:

# Define the CatBoostClassifier with hyperparameters  
cat\_class\_spec = CatBoostClassifier(  
 loss\_function='Logloss',  
 random\_seed=42,  
 verbose=0  
)  
  
cat\_class\_param\_grid = {  
 'learning\_rate': [0.01, 0.1, 0.2, 0.3],  
 'depth': [3, 4, 5, 6],  
 'iterations': [100, 200, 300],  
 'l2\_leaf\_reg': [1, 3, 5, 7]  
}  
  
# Grid search for CatBoostClassifier  
cat\_class\_grid\_search = GridSearchCV(  
 estimator=cat\_class\_spec,  
 param\_grid=cat\_class\_param\_grid,  
 scoring='roc\_auc',  
 refit=True,  
 cv=5,  
 n\_jobs=6  
)  
  
# Fit the CatBoostClassifier grid search  
cat\_class\_grid\_search.fit(X\_train, y\_train)  
best\_cat\_class\_model = cat\_class\_grid\_search.best\_estimator\_  
  
# Save the best CatBoostClassifier model  
joblib.dump(best\_cat\_class\_model, '../models/best\_cat\_class\_model\_final.pkl')

Out[186]:

['../models/best\_cat\_class\_model\_final.pkl']

In [187]:

xgb\_y\_pred = best\_xgb\_class\_model.predict(X\_test)  
xgb\_y\_pred\_proba = best\_xgb\_class\_model.predict\_proba(X\_test)[:, 1]  
xgb\_accuracy = accuracy\_score(y\_test, xgb\_y\_pred)  
xgb\_precision = precision\_score(y\_test, xgb\_y\_pred)  
xgb\_recall = recall\_score(y\_test, xgb\_y\_pred)  
xgb\_f1 = f1\_score(y\_test, xgb\_y\_pred)  
xgb\_roc\_auc = roc\_auc\_score(y\_test, xgb\_y\_pred\_proba)  
  
# Predict and evaluate CatBoost Classifier  
catboost\_y\_pred = best\_cat\_class\_model.predict(X\_test)  
catboost\_y\_pred\_proba = best\_cat\_class\_model.predict\_proba(X\_test)[:, 1]  
catboost\_accuracy = accuracy\_score(y\_test, catboost\_y\_pred)  
catboost\_precision = precision\_score(y\_test, catboost\_y\_pred)  
catboost\_recall = recall\_score(y\_test, catboost\_y\_pred)  
catboost\_f1 = f1\_score(y\_test, catboost\_y\_pred)  
catboost\_roc\_auc = roc\_auc\_score(y\_test, catboost\_y\_pred\_proba)  
  
# Print evaluation metrics  
print("XGBoost Classifier Metrics:")  
print(f"Accuracy: {xgb\_accuracy:.4f}")  
print(f"Precision: {xgb\_precision:.4f}")  
print(f"Recall: {xgb\_recall:.4f}")  
print(f"F1 Score: {xgb\_f1:.4f}")  
print(f"ROC-AUC: {xgb\_roc\_auc:.4f}")  
  
print("\nCatBoost Classifier Metrics:")  
print(f"Accuracy: {catboost\_accuracy:.4f}")  
print(f"Precision: {catboost\_precision:.4f}")  
print(f"Recall: {catboost\_recall:.4f}")  
print(f"F1 Score: {catboost\_f1:.4f}")  
print(f"ROC-AUC: {catboost\_roc\_auc:.4f}")

XGBoost Classifier Metrics:  
Accuracy: 0.8793  
Precision: 0.6758  
Recall: 0.2946  
F1 Score: 0.4104  
ROC-AUC: 0.8457  
  
CatBoost Classifier Metrics:  
Accuracy: 0.8808  
Precision: 0.6741  
Recall: 0.3170  
F1 Score: 0.4312  
ROC-AUC: 0.8467

In [188]:

# Plot ROC Curve  
fpr\_xgb, tpr\_xgb, \_ = roc\_curve(y\_test, xgb\_y\_pred\_proba)  
fpr\_catboost, tpr\_catboost, \_ = roc\_curve(y\_test, catboost\_y\_pred\_proba)  
  
plt.figure(figsize=(10, 6))  
plt.plot(fpr\_xgb, tpr\_xgb, label=f'XGBoost (AUC = {xgb\_roc\_auc:.4f})')  
plt.plot(fpr\_catboost, tpr\_catboost, label=f'CatBoost (AUC = {catboost\_roc\_auc:.4f})')  
plt.plot([0, 1], [0, 1], 'k--')  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.title('ROC Curve')  
plt.legend(loc='best')  
plt.show()

![No description has been provided for this image](data:image/png;base64;base64,)

## 4.6 Catboost Classification Parameter Tuning[¶](#X01b338277b50cebf484672045d764973fe9aadb)

In [189]:

import pandas as pd  
from sklearn.model\_selection import train\_test\_split, GridSearchCV  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, roc\_curve  
import matplotlib.pyplot as plt  
import joblib  
from catboost import CatBoostClassifier

In [190]:

# Load data  
features\_df = pd.read\_pickle('../artifacts/features\_df.pkl')  
X = features\_df[['recency', 'frequency', 'price\_sum', 'price\_mean']]  
y\_class = features\_df['spend\_90\_flag']  
  
# Split data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_class, test\_size=0.2, random\_state=42)

In [191]:

# Define the CatBoostClassifier with initial parameters  
cat\_class\_spec = CatBoostClassifier(  
 loss\_function='Logloss',  
 random\_seed=42,  
 verbose=0  
)  
  
# Define the hyperparameter grid for fine-tuning  
cat\_class\_param\_grid = {  
 'learning\_rate': [0.01, 0.05, 0.1, 0.2],  
 'depth': [4, 6, 8, 10],  
 'iterations': [100, 200, 300, 500],  
 'l2\_leaf\_reg': [1, 3, 5, 7, 9]  
}  
  
# Define the GridSearchCV object  
cat\_class\_grid\_search = GridSearchCV(  
 estimator=cat\_class\_spec,  
 param\_grid=cat\_class\_param\_grid,  
 scoring='roc\_auc',  
 refit=True,  
 cv=5,  
 n\_jobs=-1  
)  
  
# Fit the GridSearchCV object to find the best parameters  
cat\_class\_grid\_search.fit(X\_train, y\_train)  
tuned\_cat\_class\_model = cat\_class\_grid\_search.best\_estimator\_  
  
# Save the best CatBoostClassifier model  
joblib.dump(tuned\_cat\_class\_model, '../models/cat\_class\_model\_tuning.pkl')

/home/blackitalian/miniconda3/envs/cdnow/lib/python3.11/site-packages/joblib/externals/loky/process\_executor.py:752: UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak.

Out[191]:

['../models/cat\_class\_model\_tuning.pkl']

In [192]:

# Predict and evaluate CatBoost Classifier with best parameters  
catboost\_y\_pred = tuned\_cat\_class\_model.predict(X\_test)  
catboost\_y\_pred\_proba = tuned\_cat\_class\_model.predict\_proba(X\_test)[:, 1]  
catboost\_accuracy = accuracy\_score(y\_test, catboost\_y\_pred)  
catboost\_precision = precision\_score(y\_test, catboost\_y\_pred)  
catboost\_recall = recall\_score(y\_test, catboost\_y\_pred)  
catboost\_f1 = f1\_score(y\_test, catboost\_y\_pred)  
catboost\_roc\_auc = roc\_auc\_score(y\_test, catboost\_y\_pred\_proba)  
  
# Print evaluation metrics  
print("CatBoost Classifier Metrics (Tuned):")  
print(f"Accuracy: {catboost\_accuracy:.4f}")  
print(f"Precision: {catboost\_precision:.4f}")  
print(f"Recall: {catboost\_recall:.4f}")  
print(f"F1 Score: {catboost\_f1:.4f}")  
print(f"ROC-AUC: {catboost\_roc\_auc:.4f}")

CatBoost Classifier Metrics (Tuned):  
Accuracy: 0.8806  
Precision: 0.6687  
Recall: 0.3214  
F1 Score: 0.4342  
ROC-AUC: 0.8472

In [193]:

# Plot ROC Curve  
fpr\_catboost, tpr\_catboost, \_ = roc\_curve(y\_test, catboost\_y\_pred\_proba)  
  
plt.figure(figsize=(10, 6))  
plt.plot(fpr\_catboost, tpr\_catboost, label=f'CatBoost (AUC = {catboost\_roc\_auc:.4f})')  
plt.plot([0, 1], [0, 1], 'k--')  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.title('ROC Curve')  
plt.legend(loc='best')  
plt.show()

![No description has been provided for this image](data:image/png;base64;base64,)

Overall Performance: The tuned CatBoost classifier demonstrates slight improvements in recall, F1 score, and ROC-AUC compared to the previous models. These improvements suggest that fine-tuning the CatBoost classifier has resulted in a better balance between identifying true positive cases and maintaining a low false positive rate

In [194]:

# Generate probabilities using the predict\_proba method  
combined\_class\_y\_pred\_final = best\_cat\_class\_model.predict\_proba(X)[:, 1]  
# Save the probabilities as a NumPy array to a pickle file  
joblib.dump(combined\_class\_y\_pred\_final, '../artifacts/tuned\_class\_y\_pred\_final.pkl')  
  
# Convert to DataFrame if needed  
combined\_class\_y\_pred\_final\_df = pd.DataFrame(combined\_class\_y\_pred\_final, columns=['pred\_prob'])  
  
# Save the probabilities to a pickle file  
combined\_class\_y\_pred\_final\_df.to\_pickle('../artifacts/tuned\_class\_y\_pred\_df\_final.pkl')

## 4.7 Classification Ensemble[¶](#X153a3f7d3596cf1acf79877a8a34168de9fcebb)

In [150]:

import pandas as pd  
from sklearn.model\_selection import train\_test\_split, GridSearchCV  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, roc\_curve  
import matplotlib.pyplot as plt  
import joblib  
from xgboost import XGBClassifier  
from catboost import CatBoostClassifier

In [151]:

# Load data  
features\_df = pd.read\_pickle('../artifacts/features\_df.pkl')  
X = features\_df[['recency', 'frequency', 'price\_sum', 'price\_mean']]  
y\_class = features\_df['spend\_90\_flag']  
  
# Split data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_class, test\_size=0.2, random\_state=42)

In [152]:

# Define the XGBClassifier with initial parameters  
xgb\_class\_spec = XGBClassifier(  
 objective='binary:logistic',  
 random\_state=42  
)  
  
xgb\_class\_param\_grid = {  
 'learning\_rate': [0.01, 0.1, 0.2, 0.3],  
 'n\_estimators': [100, 200, 300],  
 'max\_depth': [3, 4, 5],  
 'subsample': [0.7, 0.8, 0.9, 1.0],  
 'colsample\_bytree': [0.7, 0.8, 0.9, 1.0],  
 'min\_child\_weight': [1, 3, 5],  
 'alpha': [0, 0.1, 0.5, 1],  
 'lambda': [1, 1.5, 2, 3]  
}  
  
# Grid search for XGBClassifier  
xgb\_class\_grid\_search = GridSearchCV(  
 estimator=xgb\_class\_spec,  
 param\_grid=xgb\_class\_param\_grid,  
 scoring='roc\_auc',  
 refit=True,  
 cv=5,  
 n\_jobs=-1  
)  
  
# Fit the XGBClassifier grid search  
xgb\_class\_grid\_search.fit(X\_train, y\_train)  
best\_xgb\_class\_model = xgb\_class\_grid\_search.best\_estimator\_  
  
# Save the best XGBClassifier model  
joblib.dump(best\_xgb\_class\_model, '../models/ens\_xgb\_class\_model\_final.pkl')  
  
# Define the CatBoostClassifier with initial parameters  
cat\_class\_spec = CatBoostClassifier(  
 loss\_function='Logloss',  
 random\_seed=42,  
 verbose=0  
)  
  
cat\_class\_param\_grid = {  
 'learning\_rate': [0.01, 0.1, 0.2, 0.3],  
 'depth': [4, 6, 8, 10],  
 'iterations': [100, 200, 300, 500],  
 'l2\_leaf\_reg': [1, 3, 5, 7, 9]  
}  
  
# Grid search for CatBoostClassifier  
cat\_class\_grid\_search = GridSearchCV(  
 estimator=cat\_class\_spec,  
 param\_grid=cat\_class\_param\_grid,  
 scoring='roc\_auc',  
 refit=True,  
 cv=5,  
 n\_jobs=-1  
)  
  
# Fit the CatBoostClassifier grid search  
cat\_class\_grid\_search.fit(X\_train, y\_train)  
best\_cat\_class\_model = cat\_class\_grid\_search.best\_estimator\_  
  
# Save the best CatBoostClassifier model  
joblib.dump(best\_cat\_class\_model, '../models/ens\_cat\_class\_model\_final.pkl')

/home/blackitalian/miniconda3/envs/cdnow/lib/python3.11/site-packages/joblib/externals/loky/process\_executor.py:752: UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak.  
/home/blackitalian/miniconda3/envs/cdnow/lib/python3.11/site-packages/joblib/externals/loky/process\_executor.py:752: UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak.

Out[152]:

['../models/best\_cat\_class\_model\_final.pkl']

In [157]:

# Predict and evaluate XGBoost Classifier with best parameters  
xgb\_y\_pred\_proba = best\_xgb\_class\_model.predict\_proba(X\_test)[:, 1]  
  
# Predict and evaluate CatBoost Classifier with best parameters  
catboost\_y\_pred\_proba = best\_cat\_class\_model.predict\_proba(X\_test)[:, 1]  
  
# Combine predictions using simple average  
combined\_pred\_proba = (xgb\_y\_pred\_proba + catboost\_y\_pred\_proba) / 2  
combined\_y\_pred = (combined\_pred\_proba >= 0.5).astype(int)  
# Save the probabilities as a NumPy array to a pickle file  
joblib.dump(combined\_y\_pred, '../artifacts/combined\_class\_predictions.pkl')  
  
# Convert to DataFrame if necessary  
combined\_class\_y\_pred\_df = pd.DataFrame({'combined\_y\_pred': combined\_y\_pred})  
  
# Save combined predictions to a pickle file  
joblib.dump(combined\_y\_pred, '../artifacts/combined\_class\_predictions\_df.pkl')

Out[157]:

['../artifacts/combined\_class\_y\_pred\_final.pkl']

In [158]:

# Evaluate combined model  
combined\_accuracy = accuracy\_score(y\_test, combined\_y\_pred)  
combined\_precision = precision\_score(y\_test, combined\_y\_pred)  
combined\_recall = recall\_score(y\_test, combined\_y\_pred)  
combined\_f1 = f1\_score(y\_test, combined\_y\_pred)  
combined\_roc\_auc = roc\_auc\_score(y\_test, combined\_pred\_proba)  
  
print("Combined Model Metrics:")  
print(f"Accuracy: {combined\_accuracy:.4f}")  
print(f"Precision: {combined\_precision:.4f}")  
print(f"Recall: {combined\_recall:.4f}")  
print(f"F1 Score: {combined\_f1:.4f}")  
print(f"ROC-AUC: {combined\_roc\_auc:.4f}")

Combined Model Metrics:  
Accuracy: 0.8806  
Precision: 0.6730  
Recall: 0.3155  
F1 Score: 0.4296  
ROC-AUC: 0.8472

In [159]:

# Plot ROC Curve for combined model  
fpr, tpr, \_ = roc\_curve(y\_test, combined\_pred\_proba)  
  
plt.figure(figsize=(10, 6))  
plt.plot(fpr, tpr, label=f'Combined (AUC = {combined\_roc\_auc:.4f})')  
plt.plot([0, 1], [0, 1], 'k--')  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.title('ROC Curve')  
plt.legend(loc='best')  
plt.show()

![No description has been provided for this image](data:image/png;base64;base64,)

Recall and F1 are higher on the tuned CatBoost than the ensemble and is the driving decision for model selection. Tuned CatBoost model is the final model selected.

# 5.0 Productionize[¶](#Xf03a069f16d70320f6274973a0f9e317f2aab32)

## 5.1 Regression[¶](#X6dd1ab3862a5ea638e23338223140660331ddcb)

In [160]:

import pandas as pd  
import numpy as np  
import joblib  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score  
  
def load\_models(xgb\_model\_path, stacking\_model\_path):  
 """  
 Load the pre-trained models from disk.  
   
 Parameters:  
 xgb\_model\_path (str): Path to the saved XGBoost model.  
 stacking\_model\_path (str): Path to the saved stacking model.  
   
 Returns:  
 tuple: Loaded XGBoost model and stacking model.  
 """  
 xgb\_model = joblib.load(xgb\_model\_path)  
 stacking\_model = joblib.load(stacking\_model\_path)  
 return xgb\_model, stacking\_model  
  
def predict\_and\_evaluate(xgb\_model, stacking\_model, X, y=None):  
 """  
 Use the loaded models to predict and evaluate on the given data.  
   
 Parameters:  
 xgb\_model: The loaded XGBoost model.  
 stacking\_model: The loaded stacking model.  
 X (pd.DataFrame): Features for prediction.  
 y (pd.Series, optional): True target values for evaluation.  
   
 Returns:  
 pd.Series: Combined predictions from both models.  
 dict: Evaluation metrics if true target values are provided.  
 """  
 xgb\_predictions = xgb\_model.predict(X)  
 stacking\_predictions = stacking\_model.predict(X)  
 combined\_predictions = (xgb\_predictions + stacking\_predictions) / 2  
   
 if y is not None:  
 mae = mean\_absolute\_error(y, combined\_predictions)  
 mse = mean\_squared\_error(y, combined\_predictions)  
 rmse = np.sqrt(mse)  
 r2 = r2\_score(y, combined\_predictions)  
 metrics = {  
 "MAE": mae,  
 "MSE": mse,  
 "RMSE": rmse,  
 "R²": r2  
 }  
 return pd.Series(combined\_predictions), metrics  
 else:  
 return pd.Series(combined\_predictions), None

#### Example: Deploy as a Web Service using Flask[¶](#X2e76f41fafc3bfcab99156a0274f930241af1be)

In [162]:

from flask import Flask, request, jsonify  
import pandas as pd  
import joblib  
import numpy as np  
  
app = Flask(\_\_name\_\_)  
  
def load\_models(xgb\_model\_path, stacking\_model\_path):  
 """  
 Load the pre-trained models from disk.  
   
 Parameters:  
 xgb\_model\_path (str): Path to the saved XGBoost model.  
 stacking\_model\_path (str): Path to the saved stacking model.  
   
 Returns:  
 tuple: Loaded XGBoost model and stacking model.  
 """  
 xgb\_model = joblib.load(xgb\_model\_path)  
 stacking\_model = joblib.load(stacking\_model\_path)  
 return xgb\_model, stacking\_model  
  
def predict(xgb\_model, stacking\_model, X):  
 """  
 Use the loaded models to predict on the given data.  
   
 Parameters:  
 xgb\_model: The loaded XGBoost model.  
 stacking\_model: The loaded stacking model.  
 X (pd.DataFrame): Features for prediction.  
   
 Returns:  
 pd.Series: Combined predictions from both models.  
 """  
 xgb\_predictions = xgb\_model.predict(X)  
 stacking\_predictions = stacking\_model.predict(X)  
 combined\_predictions = (xgb\_predictions + stacking\_predictions) / 2  
 return pd.Series(combined\_predictions)  
  
# Load the models  
xgb\_model\_path = '../models/best\_xgb\_reg\_model\_final.pkl'  
stacking\_model\_path = '../models/stacking\_model\_final.pkl'  
xgb\_model, stacking\_model = load\_models(xgb\_model\_path, stacking\_model\_path)  
  
@app.route('/predict', methods=['POST'])  
def predict\_endpoint():  
 data = request.get\_json(force=True)  
 df = pd.DataFrame(data)  
 X = df[['recency', 'frequency', 'price\_sum', 'price\_mean']]  
   
 predictions = predict(xgb\_model, stacking\_model, X)  
   
 return jsonify(predictions=predictions.tolist())  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=True)

---------------------------------------------------------------------------  
ModuleNotFoundError Traceback (most recent call last)  
Cell In[162], line 1  
----> 1 from flask import Flask, request, jsonify  
 2 import pandas as pd  
 3 import joblib  
  
ModuleNotFoundError: No module named 'flask'

## 5.2 Classification[¶](#X349fd46b8dfbe5cd94519283b1e0d53e3627cf2)

In [163]:

import joblib  
import pandas as pd  
  
def load\_model(model\_path):  
 """  
 Load the pre-trained CatBoost model from disk.  
   
 Parameters:  
 model\_path (str): Path to the saved model.  
   
 Returns:  
 CatBoostClassifier: Loaded CatBoost model.  
 """  
 model = joblib.load(model\_path)  
 return model  
  
def predict(model, X):  
 """  
 Make predictions using the loaded CatBoost model.  
   
 Parameters:  
 model (CatBoostClassifier): Loaded CatBoost model.  
 X (pd.DataFrame): Data for which predictions are to be made.  
   
 Returns:  
 np.ndarray: Predicted classes.  
 np.ndarray: Predicted probabilities.  
 """  
 y\_pred = model.predict(X)  
 y\_pred\_proba = model.predict\_proba(X)[:, 1]  
 return y\_pred, y\_pred\_proba  
  
# Load the model  
model\_path = '../models/best\_cat\_class\_model\_final.pkl'  
catboost\_model = load\_model(model\_path)  
  
# Example usage with new data  
# Load new data (ensure the new data has the same feature columns as the training data)  
new\_data = pd.read\_csv('path\_to\_new\_data.csv')  
X\_new = new\_data[['recency', 'frequency', 'price\_sum', 'price\_mean']]  
  
# Make predictions  
y\_pred, y\_pred\_proba = predict(catboost\_model, X\_new)

---------------------------------------------------------------------------  
FileNotFoundError Traceback (most recent call last)  
Cell In[163], line 39  
 35 catboost\_model = load\_model(model\_path)  
 37 # Example usage with new data  
 38 # Load new data (ensure the new data has the same feature columns as the training data)  
---> 39 new\_data = pd.read\_csv('path\_to\_new\_data.csv')  
 40 X\_new = new\_data[['recency', 'frequency', 'price\_sum', 'price\_mean']]  
 42 # Make predictions  
  
File ~/miniconda3/envs/cdnow/lib/python3.11/site-packages/pandas/io/parsers/readers.py:1026, in read\_csv(filepath\_or\_buffer, sep, delimiter, header, names, index\_col, usecols, dtype, engine, converters, true\_values, false\_values, skipinitialspace, skiprows, skipfooter, nrows, na\_values, keep\_default\_na, na\_filter, verbose, skip\_blank\_lines, parse\_dates, infer\_datetime\_format, keep\_date\_col, date\_parser, date\_format, dayfirst, cache\_dates, iterator, chunksize, compression, thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapechar, comment, encoding, encoding\_errors, dialect, on\_bad\_lines, delim\_whitespace, low\_memory, memory\_map, float\_precision, storage\_options, dtype\_backend)  
 1013 kwds\_defaults = \_refine\_defaults\_read(  
 1014 dialect,  
 1015 delimiter,  
 (...)  
 1022 dtype\_backend=dtype\_backend,  
 1023 )  
 1024 kwds.update(kwds\_defaults)  
-> 1026 return \_read(filepath\_or\_buffer, kwds)  
  
File ~/miniconda3/envs/cdnow/lib/python3.11/site-packages/pandas/io/parsers/readers.py:620, in \_read(filepath\_or\_buffer, kwds)  
 617 \_validate\_names(kwds.get("names", None))  
 619 # Create the parser.  
--> 620 parser = TextFileReader(filepath\_or\_buffer, \*\*kwds)  
 622 if chunksize or iterator:  
 623 return parser  
  
File ~/miniconda3/envs/cdnow/lib/python3.11/site-packages/pandas/io/parsers/readers.py:1620, in TextFileReader.\_\_init\_\_(self, f, engine, \*\*kwds)  
 1617 self.options["has\_index\_names"] = kwds["has\_index\_names"]  
 1619 self.handles: IOHandles | None = None  
-> 1620 self.\_engine = self.\_make\_engine(f, self.engine)  
  
File ~/miniconda3/envs/cdnow/lib/python3.11/site-packages/pandas/io/parsers/readers.py:1880, in TextFileReader.\_make\_engine(self, f, engine)  
 1878 if "b" not in mode:  
 1879 mode += "b"  
-> 1880 self.handles = get\_handle(  
 1881 f,  
 1882 mode,  
 1883 encoding=self.options.get("encoding", None),  
 1884 compression=self.options.get("compression", None),  
 1885 memory\_map=self.options.get("memory\_map", False),  
 1886 is\_text=is\_text,  
 1887 errors=self.options.get("encoding\_errors", "strict"),  
 1888 storage\_options=self.options.get("storage\_options", None),  
 1889 )  
 1890 assert self.handles is not None  
 1891 f = self.handles.handle  
  
File ~/miniconda3/envs/cdnow/lib/python3.11/site-packages/pandas/io/common.py:873, in get\_handle(path\_or\_buf, mode, encoding, compression, memory\_map, is\_text, errors, storage\_options)  
 868 elif isinstance(handle, str):  
 869 # Check whether the filename is to be opened in binary mode.  
 870 # Binary mode does not support 'encoding' and 'newline'.  
 871 if ioargs.encoding and "b" not in ioargs.mode:  
 872 # Encoding  
--> 873 handle = open(  
 874 handle,  
 875 ioargs.mode,  
 876 encoding=ioargs.encoding,  
 877 errors=errors,  
 878 newline="",  
 879 )  
 880 else:  
 881 # Binary mode  
 882 handle = open(handle, ioargs.mode)  
  
FileNotFoundError: [Errno 2] No such file or directory: 'path\_to\_new\_data.csv'

#### Example: Deploy as a Web Service using Flask[¶](#X2e76f41fafc3bfcab99156a0274f930241af1be)

In [ ]:

from flask import Flask, request, jsonify  
import joblib  
import pandas as pd  
  
app = Flask(\_\_name\_\_)  
  
# Load the model  
model\_path = '../models/best\_cat\_class\_model\_final.pkl'  
catboost\_model = joblib.load(model\_path)  
  
@app.route('/predict', methods=['POST'])  
def predict():  
 data = request.get\_json(force=True)  
 df = pd.DataFrame(data)  
 X = df[['recency', 'frequency', 'price\_sum', 'price\_mean']]  
   
 predictions, probabilities = catboost\_model.predict(X), catboost\_model.predict\_proba(X)[:, 1]  
   
 return jsonify(predictions=predictions.tolist(), probabilities=probabilities.tolist())  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=True)

## 5.3 Combine Features[¶](#X47b61205aa29241d39f12695dc68df85eb242fa)

In [203]:

import numpy as np  
import pandas as pd  
import joblib

In [204]:

# Load (unpickle) the predictions  
reg\_predictions = joblib.load('../artifacts/xgb\_reg\_predictions.pkl')  
class\_predictions= joblib.load('../artifacts/tuned\_class\_y\_pred\_final.pkl')

In [205]:

# Load data  
features\_df = pd.read\_pickle('../artifacts/features\_df.pkl')

In [206]:

features\_df.shape

Out[206]:

(23570, 6)

In [207]:

reg\_predictions.shape

Out[207]:

(23570,)

In [208]:

class\_predictions.shape

Out[208]:

(23570,)

In [209]:

# Ensure reg\_predictions is a NumPy array  
if isinstance(reg\_predictions , pd.DataFrame) or isinstance(reg\_predictions , pd.Series):  
 reg\_predictions = reg\_predictions .values  
  
# Check the shape of the array  
print(f"reg\_predictions shape before flattening: {reg\_predictions .shape}")  
  
# Flatten the array if it has a single column  
if reg\_predictions .ndim == 2 and reg\_predictions .shape[1] == 1:  
 reg\_predictions = reg\_predictions .flatten()  
  
# Verify the shape after flattening  
print(f"reg\_predictions shape after flattening: {reg\_predictions .shape}")  
  
# Convert the flattened array to a DataFrame  
reg\_predictions\_df = pd.DataFrame(reg\_predictions , columns=['pred\_spend'])  
print(reg\_predictions\_df.head())

reg\_predictions shape before flattening: (23570,)  
reg\_predictions shape after flattening: (23570,)  
 pred\_spend  
0 0.952339  
1 3.389285  
2 10.587686  
3 9.366228  
4 36.670902

In [210]:

# Ensure class\_predictions is a NumPy array  
if isinstance(class\_predictions, pd.DataFrame) or isinstance(class\_predictions, pd.Series):  
 class\_predictions = class\_predictions.values  
  
# Check the shape of the array  
print(f"class\_predictions shape before flattening: {class\_predictions.shape}")  
  
# Flatten the array if it has a single column  
if class\_predictions.ndim == 2 and class\_predictions.shape[1] == 1:  
 class\_predictions = class\_predictions.flatten()  
  
# Verify the shape after flattening  
print(f"class\_predictions shape after flattening: {class\_predictions.shape}")  
  
# Convert the flattened array to a DataFrame  
class\_predictions\_df = pd.DataFrame(class\_predictions, columns=['pred\_prob'])  
print(class\_predictions\_df.head())

class\_predictions shape before flattening: (23570,)  
class\_predictions shape after flattening: (23570,)  
 pred\_prob  
0 0.034614  
1 0.053704  
2 0.291609  
3 0.266856  
4 0.553257

In [211]:

# Ensure that the DataFrames have the same number of rows  
# assert len(class\_predictions\_df) == len(reg\_predictions\_df) == len(features\_df), "Mismatch in number of rows"  
  
# Combine the predicted probabilities, regression predictions, and original features  
predictions\_df = pd.concat(  
 [  
 class\_predictions\_df.reset\_index(drop=True),  
 reg\_predictions\_df.reset\_index(drop=True),  
 features\_df.reset\_index(drop=True)  
 ], axis=1  
)

In [212]:

predictions\_df

Out[212]:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | pred\_prob | pred\_spend | recency | frequency | price\_sum | price\_mean | spend\_90\_total | spend\_90\_flag |
| 0 | 0.034614 | 0.952339 | -455 | 1 | 11.77 | 11.770000 | 0.00 | 0.0 |
| 1 | 0.053704 | 3.389285 | -444 | 2 | 89.00 | 44.500000 | 0.00 | 0.0 |
| 2 | 0.291609 | 10.587686 | -127 | 5 | 139.47 | 27.894000 | 16.99 | 1.0 |
| 3 | 0.266856 | 9.366228 | -110 | 4 | 100.50 | 25.125000 | 0.00 | 0.0 |
| 4 | 0.553257 | 36.670902 | -88 | 11 | 385.61 | 35.055455 | 0.00 | 0.0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 23565 | 0.050053 | 1.341710 | -372 | 1 | 36.00 | 36.000000 | 0.00 | 0.0 |
| 23566 | 0.050946 | 0.998963 | -372 | 1 | 20.97 | 20.970000 | 0.00 | 0.0 |
| 23567 | 0.123834 | 5.649462 | -344 | 3 | 121.70 | 40.566667 | 0.00 | 0.0 |
| 23568 | 0.048181 | 1.038381 | -372 | 1 | 25.74 | 25.740000 | 0.00 | 0.0 |
| 23569 | 0.079256 | 3.781356 | -371 | 2 | 94.08 | 47.040000 | 0.00 | 0.0 |

23570 rows × 8 columns

In [213]:

predictions\_df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 23570 entries, 0 to 23569  
Data columns (total 8 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 pred\_prob 23570 non-null float64  
 1 pred\_spend 23570 non-null float32  
 2 recency 23570 non-null int64   
 3 frequency 23570 non-null int64   
 4 price\_sum 23570 non-null float64  
 5 price\_mean 23570 non-null float64  
 6 spend\_90\_total 23570 non-null float64  
 7 spend\_90\_flag 23570 non-null float64  
dtypes: float32(1), float64(5), int64(2)  
memory usage: 1.3 MB

In [214]:

predictions\_df.to\_pickle('../artifacts/predictions\_df.pkl')

# 6.0 BUSINESS VALUE[¶](#Xa042951b6837de20ba0cecc827701d507354c0e)

In [2]:

import pandas as pd  
from pandas.plotting import table  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.metrics import confusion\_matrix, roc\_curve, auc, precision\_recall\_curve, classification\_report

In [3]:

predictions\_df = pd.read\_pickle('../artifacts/predictions\_df.pkl')

In [4]:

predictions\_df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 23570 entries, 0 to 23569  
Data columns (total 8 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 pred\_prob 23570 non-null float64  
 1 pred\_spend 23570 non-null float32  
 2 recency 23570 non-null int64   
 3 frequency 23570 non-null int64   
 4 price\_sum 23570 non-null float64  
 5 price\_mean 23570 non-null float64  
 6 spend\_90\_total 23570 non-null float64  
 7 spend\_90\_flag 23570 non-null float64  
dtypes: float32(1), float64(5), int64(2)  
memory usage: 1.3 MB

In [5]:

features\_df = pd.read\_pickle('../artifacts/features\_df.pkl')

### RFM Analysis[¶](#RFM-Analysis)

In [6]:

# Recency segmentation  
recent\_customers = predictions\_df[predictions\_df['recency'] < -30]  
average\_customers = predictions\_df[(predictions\_df['recency'] >= -30) & (predictions\_df['recency'] < 90)]  
old\_customers = predictions\_df[predictions\_df['recency'] >= -90]  
  
# Frequency segmentation  
frequent\_customers = predictions\_df[predictions\_df['frequency'] > predictions\_df['frequency'].median()]  
infrequent\_customers = predictions\_df[predictions\_df['frequency'] <= predictions\_df['frequency'].median()]  
  
# Monetary segmentation  
high\_value\_customers = predictions\_df[predictions\_df['price\_sum'] > predictions\_df['price\_sum'].median()]  
low\_value\_customers = predictions\_df[predictions\_df['price\_sum'] <= predictions\_df['price\_sum'].median()]

In [7]:

# Recency Segmentation  
fig, ax = plt.subplots(1, 3, figsize=(18, 6), sharey=True)  
fig.suptitle('Recency Segmentation')  
  
sns.histplot(recent\_customers['recency'], bins=30, kde=True, color='green', ax=ax[0])  
ax[0].set\_title('Recent Customers')  
ax[0].set\_xlabel('Recency')  
  
sns.histplot(average\_customers['recency'], bins=30, kde=True, color='blue', ax=ax[1])  
ax[1].set\_title('Average Customers')  
ax[1].set\_xlabel('Recency')  
  
sns.histplot(old\_customers['recency'], bins=30, kde=True, color='red', ax=ax[2])  
ax[2].set\_title('Old Customers')  
ax[2].set\_xlabel('Recency')  
  
plt.tight\_layout(rect=[0, 0, 1, 0.96])  
plt.show()

![No description has been provided for this image](data:image/png;base64;base64,)

In [8]:

# Frequency Segmentation  
fig, ax = plt.subplots(1, 2, figsize=(14, 6), sharey=True)  
fig.suptitle('Frequency Segmentation')  
  
sns.histplot(frequent\_customers['frequency'], bins=30, kde=True, color='orange', ax=ax[0])  
ax[0].set\_title('Frequent Customers')  
ax[0].set\_xlabel('Frequency')  
  
sns.histplot(infrequent\_customers['frequency'], bins=30, kde=True, color='purple', ax=ax[1])  
ax[1].set\_title('Infrequent Customers')  
ax[1].set\_xlabel('Frequency')  
  
plt.tight\_layout(rect=[0, 0, 1, 0.96])  
plt.show()

![No description has been provided for this image](data:image/png;base64;base64,)

In [9]:

# Monetary Segmentation  
fig, ax = plt.subplots(1, 2, figsize=(14, 6), sharey=True)  
fig.suptitle('Monetary Segmentation')  
  
sns.histplot(high\_value\_customers['price\_sum'], bins=30, kde=True, color='gold', ax=ax[0])  
ax[0].set\_title('High-Value Customers')  
ax[0].set\_xlabel('Total Spend')  
  
sns.histplot(low\_value\_customers['price\_sum'], bins=30, kde=True, color='brown', ax=ax[1])  
ax[1].set\_title('Low-Value Customers')  
ax[1].set\_xlabel('Total Spend')  
  
plt.tight\_layout(rect=[0, 0, 1, 0.96])  
plt.show()

![No description has been provided for this image](data:image/png;base64;base64,)

In [10]:

# Combined Segmentation  
high\_value\_frequent\_recent\_customers = predictions\_df[  
 (predictions\_df['price\_sum'] > predictions\_df['price\_sum'].median()) &  
 (predictions\_df['frequency'] > predictions\_df['frequency'].median()) &  
 (predictions\_df['recency'] < 30)  
]  
  
print(f"Number of high-value, frequent, and recent purchases: {high\_value\_frequent\_recent\_customers.shape[0]}")

Number of high-value, frequent, and recent purchases: 9357

#### Visualize Recency Segmentation[¶](#Visualize-Recency-Segmentation)

In [11]:

# Set up the matplotlib figure  
plt.figure(figsize=(14, 6))  
  
# Plot Recency histogram  
plt.subplot(1, 3, 1)  
sns.histplot(predictions\_df['recency'], bins=30, kde=True, color='blue')  
plt.title('Recency Distribution')  
plt.xlabel('Days Since Last Purchase')  
plt.ylabel('Frequency')  
  
# Plot Recency scatter plot  
plt.subplot(1, 3, 2)  
sns.scatterplot(data=predictions\_df, x='recency', y='price\_sum', hue='spend\_90\_flag', palette='coolwarm')  
plt.title('Recency vs Spend')  
plt.xlabel('Days Since Last Purchase')  
plt.ylabel('Total Spend')  
  
# Plot Recency box plot  
plt.subplot(1, 3, 3)  
sns.boxplot(x='spend\_90\_flag', y='recency', data=predictions\_df, palette='coolwarm')  
plt.title('Recency by Spend Flag')  
plt.xlabel('Spend 90 Flag')  
plt.ylabel('Days Since Last Purchase')  
  
plt.tight\_layout()  
plt.savefig('../../images/recency\_segmentation.png')  
plt.show()

/tmp/ipykernel\_148799/1490204064.py:20: FutureWarning:   
  
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.  
  
 sns.boxplot(x='spend\_90\_flag', y='recency', data=predictions\_df, palette='coolwarm')

![No description has been provided for this image](data:image/png;base64;base64,)

#### Visualize Frequency Segmentation[¶](#Visualize-Frequency-Segmentation)

In [12]:

# Set up the matplotlib figure  
plt.figure(figsize=(14, 6))  
  
# Plot Frequency histogram  
plt.subplot(1, 3, 1)  
sns.histplot(predictions\_df['frequency'], bins=30, kde=True, color='green')  
plt.title('Frequency Distribution')  
plt.xlabel('Number of Purchases')  
plt.ylabel('Frequency')  
  
# Plot Frequency scatter plot  
plt.subplot(1, 3, 2)  
sns.scatterplot(data=predictions\_df, x='frequency', y='price\_sum', hue='spend\_90\_flag', palette='coolwarm')  
plt.title('Frequency vs Spend')  
plt.xlabel('Number of Purchases')  
plt.ylabel('Total Spend')  
  
# Plot Frequency box plot  
plt.subplot(1, 3, 3)  
sns.boxplot(x='spend\_90\_flag', y='frequency', data=predictions\_df, palette='coolwarm')  
plt.title('Frequency by Spend Flag')  
plt.xlabel('Spend 90 Flag')  
plt.ylabel('Number of Purchases')  
  
plt.tight\_layout()  
plt.savefig('../../images/frequency\_segmentation.png')  
plt.show()

/tmp/ipykernel\_148799/868795375.py:20: FutureWarning:   
  
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.  
  
 sns.boxplot(x='spend\_90\_flag', y='frequency', data=predictions\_df, palette='coolwarm')

![No description has been provided for this image](data:image/png;base64;base64,)

#### Visualize Monetary Segmentation[¶](#Visualize-Monetary-Segmentation)

In [13]:

# Set up the matplotlib figure  
plt.figure(figsize=(14, 6))  
  
# Plot Monetary histogram  
plt.subplot(1, 3, 1)  
sns.histplot(predictions\_df['price\_sum'], bins=30, kde=True, color='red')  
plt.title('Monetary (Price Sum) Distribution')  
plt.xlabel('Total Spend')  
plt.ylabel('Frequency')  
  
# Plot Monetary scatter plot  
plt.subplot(1, 3, 2)  
sns.scatterplot(data=predictions\_df, x='price\_sum', y='frequency', hue='spend\_90\_flag', palette='coolwarm')  
plt.title('Spend vs Frequency')  
plt.xlabel('Total Spend')  
plt.ylabel('Number of Purchases')  
  
# Plot Monetary box plot  
plt.subplot(1, 3, 3)  
sns.boxplot(x='spend\_90\_flag', y='price\_sum', data=predictions\_df, palette='coolwarm')  
plt.title('Spend by Spend Flag')  
plt.xlabel('Spend 90 Flag')  
plt.ylabel('Total Spend')  
  
plt.tight\_layout()  
plt.savefig('../../images/monetary\_segmentation.png')  
plt.show()

/tmp/ipykernel\_148799/3587048487.py:20: FutureWarning:   
  
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.  
  
 sns.boxplot(x='spend\_90\_flag', y='price\_sum', data=predictions\_df, palette='coolwarm')

![No description has been provided for this image](data:image/png;base64;base64,)

### Customer Segmentation[¶](#Customer-Segmentation)

In [14]:

# High Probability & High Spend  
high\_prob\_high\_spend = predictions\_df[  
 (predictions\_df['pred\_prob'] > 0.7) &   
 (predictions\_df['pred\_spend'] > predictions\_df['pred\_spend'].median())  
]  
  
# Low Probability & Low Spend  
low\_prob\_low\_spend = predictions\_df[  
 (predictions\_df['pred\_prob'] <= 0.7) &   
 (predictions\_df['pred\_spend'] <= predictions\_df['pred\_spend'].median())  
]

In [15]:

# Define threshold values for high and low segmentation  
prob\_threshold = 0.7  
spend\_threshold = predictions\_df['pred\_spend'].median()  
  
# Define segments  
predictions\_df['Segment'] = 'Low Probability & Low Spend'  
predictions\_df.loc[(predictions\_df['pred\_prob'] > prob\_threshold) & (predictions\_df['pred\_spend'] > spend\_threshold), 'Segment'] = 'High Probability & High Spend'  
predictions\_df.loc[(predictions\_df['pred\_prob'] > prob\_threshold) & (predictions\_df['pred\_spend'] <= spend\_threshold), 'Segment'] = 'High Probability & Low Spend'  
predictions\_df.loc[(predictions\_df['pred\_prob'] <= prob\_threshold) & (predictions\_df['pred\_spend'] > spend\_threshold), 'Segment'] = 'Low Probability & High Spend'  
  
# Define color mapping for each segment  
color\_mapping = {  
 'Low Probability & Low Spend': 'blue',  
 'Low Probability & High Spend': 'grey',  
 'High Probability & High Spend': 'red',  
 'High Probability & Low Spend': 'green'  
}  
  
# Set up the matplotlib figure  
plt.figure(figsize=(14, 6))  
  
# Scatter plot for customer segmentation with transparency  
sns.scatterplot(data=predictions\_df, x='pred\_prob', y='pred\_spend', hue='Segment', palette=color\_mapping, s=100, alpha=0.7)  
plt.title('Customer Segmentation based on Predicted Probability and Predicted Spend')  
plt.xlabel('Predicted Probability')  
plt.ylabel('Predicted Spend')  
plt.legend(title='Segment')  
plt.grid(True)  
plt.savefig('../../images/customer\_segmentation\_prob\_of\_spend.png')  
plt.show()

![No description has been provided for this image](data:image/png;base64;base64,)

In [16]:

# Define segments  
predictions\_df['Segment'] = 'Low Probability & Low Spend'  
predictions\_df.loc[(predictions\_df['pred\_prob'] > prob\_threshold) & (predictions\_df['pred\_spend'] > spend\_threshold), 'Segment'] = 'High Probability & High Spend'  
predictions\_df.loc[(predictions\_df['pred\_prob'] > prob\_threshold) & (predictions\_df['pred\_spend'] <= spend\_threshold), 'Segment'] = 'High Probability & Low Spend'  
predictions\_df.loc[(predictions\_df['pred\_prob'] <= prob\_threshold) & (predictions\_df['pred\_spend'] > spend\_threshold), 'Segment'] = 'Low Probability & High Spend'  
  
# Define color mapping for each segment  
color\_mapping = {  
 'Low Probability & Low Spend': 'blue',  
 'Low Probability & High Spend': 'grey',  
 'High Probability & High Spend': 'red',  
 'High Probability & Low Spend': 'green' # Assuming a default color for this segment  
}  
  
# Get unique segments  
segments = predictions\_df['Segment'].unique()  
  
# Separate scatter plots for each segment  
fig, axes = plt.subplots(2, 2, figsize=(14, 12), sharex=True, sharey=True)  
fig.suptitle('Customer Segmentation based on Predicted Probability and Predicted Spend')  
  
for ax, segment in zip(axes.flatten(), segments):  
 sns.scatterplot(  
 data=predictions\_df[predictions\_df['Segment'] == segment],  
 x='pred\_prob',   
 y='pred\_spend',  
 color=color\_mapping.get(segment, 'black'), # Default to black if the segment is not found in the mapping  
 s=100,   
 alpha=0.7,   
 ax=ax  
 )  
 ax.set\_title(segment)  
 ax.set\_xlabel('Predicted Probability')  
 ax.set\_ylabel('Predicted Spend')  
 ax.grid(True)  
  
plt.tight\_layout(rect=[0, 0, 1, 0.96])  
plt.savefig('../../images/customer\_segmentation\_prob\_of\_spend\_2.png')  
plt.show()

![No description has been provided for this image](data:image/png;base64;base64,)

### Round pred\_prob and pred\_spend columns to 2 decimal[¶](#Xc5932ee0ce095f76e5dd06970d1cceb007be75c)

In [30]:

# set DF  
predictions\_round= predictions\_df.copy()  
# Round columns  
predictions\_round['pred\_prob'] = predictions\_round['pred\_prob'].map('{:.2f}'.format)  
predictions\_round['pred\_spend'] = predictions\_round['pred\_spend'].map('{:.2f}'.format)

In [32]:

predictions\_round.head(3)

Out[32]:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | pred\_prob | pred\_spend | recency | frequency | price\_sum | price\_mean | spend\_90\_total | spend\_90\_flag | Segment |
| 0 | 0.03 | 0.95 | -455 | 1 | 11.77 | 11.770 | 0.00 | 0.0 | Low Probability & Low Spend |
| 1 | 0.05 | 3.39 | -444 | 2 | 89.00 | 44.500 | 0.00 | 0.0 | Low Probability & High Spend |
| 2 | 0.29 | 10.59 | -127 | 5 | 139.47 | 27.894 | 16.99 | 1.0 | Low Probability & High Spend |

### What percent of our current customer base made a purchase in the last 90 days?[¶](#X571710a0fe421fa4fb65ec678bf614c51a3a3b0)

In [19]:

features\_df.info()

<class 'pandas.core.frame.DataFrame'>  
Index: 23570 entries, 1 to 23570  
Data columns (total 6 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 recency 23570 non-null int64   
 1 frequency 23570 non-null int64   
 2 price\_sum 23570 non-null float64  
 3 price\_mean 23570 non-null float64  
 4 spend\_90\_total 23570 non-null float64  
 5 spend\_90\_flag 23570 non-null float64  
dtypes: float64(4), int64(2)  
memory usage: 1.3 MB

In [20]:

features\_df.spend\_90\_flag.value\_counts()

Out[20]:

spend\_90\_flag  
0.0 20269  
1.0 3301  
Name: count, dtype: int64

In [21]:

percent = round((3301/23570)\*100,2)  
  
print(f'{percent}% of our current customers made a purchase in the last 90 days')

14.01% of our current customers made a purchase in the last 90 days

### Which customers have the highest spend probability in the next 90 days?[¶](#X6e2b822a0ec94ddb4c42dd5224f8b0e0c7c994f)

In [29]:

# Convert rounded values to strings to ensure rounding persists  
predictions\_round['pred\_prob'] = predictions\_round['pred\_prob'].map('{:.2f}'.format)  
predictions\_round['pred\_spend'] = predictions\_round['pred\_spend'].map('{:.2f}'.format)  
  
# Sort by pred\_prob and get top 10  
highest\_prob = predictions\_round.sort\_values('pred\_prob', ascending=False).head(10)  
  
# Check the sorted DataFrame to ensure values are rounded  
print(highest\_prob)

pred\_prob pred\_spend recency frequency price\_sum price\_mean \  
14047 0.95 1862.58 0 180 7267.15 40.373056   
498 0.95 578.16 -3 100 3427.55 34.275500   
10078 0.95 540.10 0 62 2100.38 33.877097   
7591 0.95 1305.54 -2 165 11478.02 69.563758   
22060 0.95 1108.66 -2 118 3371.80 28.574576   
7982 0.95 974.06 -9 105 5824.14 55.468000   
19596 0.94 267.83 -1 97 2023.00 20.855670   
2483 0.94 84.41 0 59 1438.93 24.388644   
3048 0.94 453.97 -1 97 3484.03 35.917835   
7930 0.93 429.35 -10 54 5486.74 101.606296   
  
 spend\_90\_total spend\_90\_flag Segment   
14047 1709.18 1.0 High Probability & High Spend   
498 951.00 1.0 High Probability & High Spend   
10078 100.43 1.0 High Probability & High Spend   
7591 2512.91 1.0 High Probability & High Spend   
22060 577.10 1.0 High Probability & High Spend   
7982 1148.93 1.0 High Probability & High Spend   
19596 334.57 1.0 High Probability & High Spend   
2483 345.43 1.0 High Probability & High Spend   
3048 778.82 1.0 High Probability & High Spend   
7930 1010.44 1.0 High Probability & High Spend

In [34]:

predictions\_round.sort\_values('pred\_prob', ascending=False).head(10)

Out[34]:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | pred\_prob | pred\_spend | recency | frequency | price\_sum | price\_mean | spend\_90\_total | spend\_90\_flag | Segment |
| 14047 | 0.95 | 1862.58 | 0 | 180 | 7267.15 | 40.373056 | 1709.18 | 1.0 | High Probability & High Spend |
| 498 | 0.95 | 578.16 | -3 | 100 | 3427.55 | 34.275500 | 951.00 | 1.0 | High Probability & High Spend |
| 10078 | 0.95 | 540.10 | 0 | 62 | 2100.38 | 33.877097 | 100.43 | 1.0 | High Probability & High Spend |
| 7591 | 0.95 | 1305.54 | -2 | 165 | 11478.02 | 69.563758 | 2512.91 | 1.0 | High Probability & High Spend |
| 22060 | 0.95 | 1108.66 | -2 | 118 | 3371.80 | 28.574576 | 577.10 | 1.0 | High Probability & High Spend |
| 7982 | 0.95 | 974.06 | -9 | 105 | 5824.14 | 55.468000 | 1148.93 | 1.0 | High Probability & High Spend |
| 19596 | 0.94 | 267.83 | -1 | 97 | 2023.00 | 20.855670 | 334.57 | 1.0 | High Probability & High Spend |
| 2483 | 0.94 | 84.41 | 0 | 59 | 1438.93 | 24.388644 | 345.43 | 1.0 | High Probability & High Spend |
| 3048 | 0.94 | 453.97 | -1 | 97 | 3484.03 | 35.917835 | 778.82 | 1.0 | High Probability & High Spend |
| 7930 | 0.93 | 429.35 | -10 | 54 | 5486.74 | 101.606296 | 1010.44 | 1.0 | High Probability & High Spend |

In [33]:

# Sort by pred\_prob and get top 10  
highest\_prob = predictions\_round.sort\_values('pred\_prob', ascending=False).head(10)  
  
# Set up a figure  
fig, ax = plt.subplots(figsize=(20, 10)) # set size frame  
ax.axis('off') # no axes  
ax.set\_title('Highest Probability of Purchase', fontsize=34)  
  
# Create a table plot  
tbl = table(ax, highest\_prob, loc='center', cellLoc='center', colWidths=[0.1]\*len(highest\_prob.columns))  
  
# Style the table  
tbl.auto\_set\_font\_size(False)  
tbl.set\_fontsize(16)  
tbl.scale(2.2, 2.2)  
  
# Save the table as an image  
image\_path = '../../images/top\_10\_prob.png'  
plt.savefig(image\_path, bbox\_inches='tight', pad\_inches=0.1)  
plt.show()

![No description has been provided for this image](data:image/png;base64;base64,)

### Which customers have recently purchases but are unlikely to buy?[¶](#X5a0a85b6ac5d3bd131c5c65be30b351bdf6224d)

* Incentivize actions to increase probability.
* Provide discounts, encourage referring a friend, nurture by letting them know what's coming.

In [24]:

# customers that purchased in the 90 days but are unlikely to make another purchase  
drop\_out = predictions\_df[  
 (predictions\_df['recency'] >= -90) &   
 (predictions\_df['pred\_prob'] < 0.20)  
].sort\_values('pred\_prob', ascending=False).head(10)  
  
# Display the filtered DataFrame  
drop\_out

Out[24]:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | pred\_prob | pred\_spend | recency | frequency | price\_sum | price\_mean | spend\_90\_total | spend\_90\_flag | Segment |
| 1345 | 0.199453 | 7.109229 | -75 | 2 | 35.73 | 17.865 | 0.00 | 0.0 | Low Probability & High Spend |
| 3648 | 0.199023 | 7.565193 | -83 | 2 | 57.23 | 28.615 | 0.00 | 0.0 | Low Probability & High Spend |
| 23227 | 0.198740 | 7.232706 | -85 | 2 | 55.86 | 27.930 | 0.00 | 0.0 | Low Probability & High Spend |
| 5734 | 0.198520 | 7.797125 | -64 | 2 | 23.96 | 11.980 | 24.87 | 1.0 | Low Probability & High Spend |
| 11173 | 0.198316 | 6.677160 | -73 | 2 | 30.94 | 15.470 | 43.97 | 1.0 | Low Probability & High Spend |
| 21555 | 0.198119 | 6.552014 | -63 | 2 | 23.76 | 11.880 | 0.00 | 0.0 | Low Probability & High Spend |
| 11473 | 0.197447 | 6.677160 | -79 | 2 | 42.21 | 21.105 | 0.00 | 0.0 | Low Probability & High Spend |
| 21007 | 0.197353 | 6.823750 | -66 | 2 | 22.65 | 11.325 | 0.00 | 0.0 | Low Probability & High Spend |
| 1074 | 0.196829 | 7.109229 | -72 | 2 | 29.86 | 14.930 | 33.97 | 1.0 | Low Probability & High Spend |
| 15805 | 0.195369 | 6.201038 | -82 | 2 | 48.46 | 24.230 | 32.49 | 1.0 | Low Probability & High Spend |

### Missed opportunities: spenders that could be unlocked[¶](#X6f6b248b37957780fcb0b4d32e43de4d34c5a48)

* Send bundle offers encourage volume purchases
* Focus on missed opportunities

In [25]:

# Did the model predict any large spending that did not occur?  
# Filtered by did not spend but had a high probability of spend and recency was within the last 90 days.  
missed\_opportunities = predictions\_df[  
 (predictions\_df['spend\_90\_total'] == 0.0) &   
 (predictions\_df['recency'] >= -90)  
].sort\_values('pred\_prob', ascending=False).head(10)  
  
# Display the filtered DataFrame  
missed\_opportunities

Out[25]:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | pred\_prob | pred\_spend | recency | frequency | price\_sum | price\_mean | spend\_90\_total | spend\_90\_flag | Segment |
| 4158 | 0.859090 | 169.061584 | -4 | 36 | 931.68 | 25.880000 | 0.0 | 0.0 | High Probability & High Spend |
| 3887 | 0.815969 | 28.182161 | -14 | 25 | 324.53 | 12.981200 | 0.0 | 0.0 | High Probability & High Spend |
| 5054 | 0.808898 | 175.130753 | -8 | 20 | 797.11 | 39.855500 | 0.0 | 0.0 | High Probability & High Spend |
| 7322 | 0.808325 | 136.193359 | -10 | 22 | 827.28 | 37.603636 | 0.0 | 0.0 | High Probability & High Spend |
| 4517 | 0.806318 | 157.073425 | -25 | 36 | 1053.91 | 29.275278 | 0.0 | 0.0 | High Probability & High Spend |
| 15099 | 0.800327 | 50.185364 | -16 | 21 | 483.12 | 23.005714 | 0.0 | 0.0 | High Probability & High Spend |
| 19946 | 0.797680 | 78.172905 | -8 | 19 | 539.11 | 28.374211 | 0.0 | 0.0 | High Probability & High Spend |
| 21684 | 0.795533 | 283.233673 | -14 | 25 | 2031.06 | 81.242400 | 0.0 | 0.0 | High Probability & High Spend |
| 17522 | 0.792367 | 267.963837 | -5 | 17 | 2280.08 | 134.122353 | 0.0 | 0.0 | High Probability & High Spend |
| 13966 | 0.792274 | 84.294182 | -6 | 17 | 586.78 | 34.516471 | 0.0 | 0.0 | High Probability & High Spend |

In [26]:

# Show predictions that are larger than actual spend  
# Filtered by pred spend is larger than actual, recency is last 90 days, and they actually made a purchase  
under\_spending = predictions\_df[  
 (predictions\_df['pred\_spend'] > predictions\_df['spend\_90\_total']) &   
 (predictions\_df['recency'] >= -90) &   
 (predictions\_df['spend\_90\_flag'] == 1.0)  
].sort\_values('pred\_spend', ascending=False).head(10)  
  
# Display the filtered DataFrame  
under\_spending

Out[26]:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | pred\_prob | pred\_spend | recency | frequency | price\_sum | price\_mean | spend\_90\_total | spend\_90\_flag | Segment |
| 14047 | 0.953252 | 1862.577271 | 0 | 180 | 7267.15 | 40.373056 | 1709.18 | 1.0 | High Probability & High Spend |
| 22060 | 0.945873 | 1108.664795 | -2 | 118 | 3371.80 | 28.574576 | 577.10 | 1.0 | High Probability & High Spend |
| 22491 | 0.809251 | 731.992676 | -6 | 18 | 2271.27 | 126.181667 | 117.45 | 1.0 | High Probability & High Spend |
| 21738 | 0.798294 | 731.774902 | -13 | 19 | 2824.67 | 148.666842 | 495.16 | 1.0 | High Probability & High Spend |
| 2663 | 0.836105 | 562.152405 | -7 | 24 | 3694.52 | 153.938333 | 405.75 | 1.0 | High Probability & High Spend |
| 10078 | 0.946092 | 540.101685 | 0 | 62 | 2100.38 | 33.877097 | 100.43 | 1.0 | High Probability & High Spend |
| 709 | 0.933141 | 501.191406 | -2 | 51 | 2216.45 | 43.459804 | 285.85 | 1.0 | High Probability & High Spend |
| 1076 | 0.897347 | 487.034729 | -9 | 40 | 2847.72 | 71.193000 | 476.77 | 1.0 | High Probability & High Spend |
| 20916 | 0.926549 | 442.450226 | -6 | 48 | 3037.98 | 63.291250 | 39.47 | 1.0 | High Probability & High Spend |
| 22347 | 0.860961 | 414.604828 | -12 | 28 | 1967.80 | 70.278571 | 389.09 | 1.0 | High Probability & High Spend |

In [ ]: