aidancharles2004-arch / **aidan_app**

Code Issues Pull requests Actions Projects Wiki Securit

aidan_app / **new_pp.ipynb**

aidancharles2004-arch Add files via upload 5ae3bcb · 3 minutes ago

2895 lines (2895 loc) · 1.58 MB

aidan_app / **new_pp.ipynb** Top

Preview Code Blame Raw

Chapter 3:Implementation

3.1 Data Loading and Preprocessing

In [2]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import datetime

df = pd.read_csv("group_7.csv")
```

In [6]:
```python
df.columns
```

Out[6]:
```
Index(['Income', 'Age', 'Dependents', 'Occupation', 'City_Tier', 'Re
nt',
       'Loan_Repayment', 'Insurance', 'Groceries', 'Transport', 'Eat
ing_Out',
       'Entertainment', 'Utilities', 'Healthcare', 'Education',
       'Miscellaneous', 'Desired_Savings_Percentage', 'Desired_Savin
gs',
       'Disposable_Income', 'Potential_Savings_Groceries',
       'Potential_Savings_Transport', 'Potential_Savings_Eating_Out
',
       'Potential_Savings_Entertainment', 'Potential_Savings_Utiliti
es',
       'Potential_Savings_Healthcare', 'Potential_Savings_Education
',
       'Potential_Savings_Miscellaneous', 'Fake_date', 'days_since_s
tart',
       'weekday_sin', 'weekday_cos', 'month_sin', 'month_cos'],
      dtype='object')
```

In [4]:
```python
df.head()
```

Out[4]:

	Income	Age	Dependents	Occupation	City_Tier	Rent	Loan_
0	44637.249636	49	0	Self_Employed	Tier_1	13391.174891	
1	26858.596592	34	2	Retired	Tier_2	5371.719318	
2	50367.605084	35	1	Student	Tier_3	7555.140763	4
3	101455.600247	21	0	Self_Employed	Tier_3	15218.340037	6
4	24875.283548	52	4	Professional	Tier_2	4975.056710	3

5 rows × 27 columns

In [5]:
```python
df.columns
```

Out[5]:
```
Index(['Income', 'Age', 'Dependents', 'Occupation', 'City_Tier', 'Re
nt',
       'Loan_Repayment', 'Insurance', 'Groceries', 'Transport', 'Eat
ing_Out',
       'Entertainment', 'Utilities', 'Healthcare', 'Education',
       'Miscellaneous', 'Desired_Savings_Percentage', 'Desired_Savin
gs',
       'Disposable_Income', 'Potential_Savings_Groceries',
       'Potential_Savings_Transport', 'Potential_Savings_Eating_Out
',
       'Potential_Savings_Entertainment', 'Potential_Savings_Utiliti
es',
       'Potential_Savings_Healthcare', 'Potential_Savings_Education
',
       'Potential_Savings_Miscellaneous'],
      dtype='object')
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 27 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   Income                           20000 non-null  float64
 1   Age                              20000 non-null  int64  
 2   Dependents                       20000 non-null  int64  
 3   Occupation                       20000 non-null  object 
 4   City_Tier                        20000 non-null  object 
 5   Rent                             20000 non-null  float64
 6   Loan_Repayment                   20000 non-null  float64
 7   Insurance                        20000 non-null  float64
 8   Groceries                        20000 non-null  float64
 9   Transport                        20000 non-null  float64
 10  Eating_Out                       20000 non-null  float64
 11  Entertainment                    20000 non-null  float64
 12  Utilities                        20000 non-null  float64
 13  Healthcare                       20000 non-null  float64
 14  Education                        20000 non-null  float64
 15  Miscellaneous                    20000 non-null  float64
 16  Desired_Savings_Percentage       20000 non-null  float64
 17  Desired_Savings                  20000 non-null  float64
 18  Disposable_Income                20000 non-null  float64
 19  Potential_Savings_Groceries      20000 non-null  float64
 20  Potential_Savings_Transport      20000 non-null  float64
 21  Potential_Savings_Eating_Out     20000 non-null  float64
 22  Potential_Savings_Entertainment  20000 non-null  float64
 23  Potential_Savings_Utilities      20000 non-null  float64
 24  Potential_Savings_Healthcare     20000 non-null  float64
 25  Potential_Savings_Education      20000 non-null  float64
 26  Potential_Savings_Miscellaneous  20000 non-null  float64
dtypes: float64(23), int64(2), object(2)
memory usage: 4.1+ MB
```

In [7]:
```
df.describe()
```

Out[7]:

	Income	Age	Dependents	Rent	Loan_Repaym
count	2.000000e+04	20000.000000	20000.000000	20000.000000	20000.000
mean	4.158550e+04	41.031450	1.995950	9115.494629	2049.800

std	4.001454e+04	13.578725	1.417616	9254.228188	4281.789
min	1.301187e+03	18.000000	0.000000	235.365692	0.000
25%	1.760488e+04	29.000000	1.000000	3649.422246	0.000
50%	3.018538e+04	41.000000	2.000000	6402.751824	0.000
75%	5.176545e+04	53.000000	3.000000	11263.940492	2627.142
max	1.079728e+06	64.000000	4.000000	215945.674703	123080.682

8 rows × 25 columns

In [8]:
```
df.isnull()
```

Out[8]:

	Income	Age	Dependents	Occupation	City_Tier	Rent	Loan_Repayment
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
19995	False	False	False	False	False	False	False
19996	False	False	False	False	False	False	False
19997	False	False	False	False	False	False	False
19998	False	False	False	False	False	False	False
19999	False	False	False	False	False	False	False

20000 rows × 27 columns

In [9]:
```
df.duplicated()
```

Out[9]:
```
0        False
1        False
2        False
3        False
4        False
         ...
19995    False
19996    False
19997    False
19998    False
19999    False
Length: 20000, dtype: bool
```

In [10]:
```
df.head()
```

Out[10]:

	Income	Age	Dependents	Occupation	City_Tier	Rent	Loan_

0	44637.249636	49	0	Self_Employed	Tier_1	13391.174891	
1	26858.596592	34	2	Retired	Tier_2	5371.719318	
2	50367.605084	35	1	Student	Tier_3	7555.140763	4
3	101455.600247	21	0	Self_Employed	Tier_3	15218.340037	6
4	24875.283548	52	4	Professional	Tier_2	4975.056710	3

5 rows × 27 columns

In [11]:

```
df.shape
```

Out[11]: (20000, 27)

In [12]:

```
df.tail()
```

Out[12]:

	Income	Age	Dependents	Occupation	City_Tier	Rent	L
19995	40913.466178	51	4	Self_Employed	Tier_1	12274.039853	
19996	90295.772638	21	1	Student	Tier_2	18059.154528	
19997	40604.567373	30	1	Professional	Tier_2	8120.913475	
19998	118157.817240	27	2	Professional	Tier_1	35447.345172	
19999	8209.249769	62	3	Professional	Tier_1	2462.774931	

5 rows × 27 columns

2.3 Feature Engineering

In [13]:

```
# Generate daily fake date
today = datetime.datetime.today().date()
start_date = pd.to_datetime(today)
df['Fake_date'] = pd.date_range(start=start_date, periods=len(df), f

# Continuous time feature
df['days_since_start'] = (df['Fake_date'] - df['Fake_date'].min()).d

# Cyclical features for weekly and monthly seasonality
df['weekday_sin'] = np.sin(2 * np.pi * df['Fake_date'].dt.weekday /
df['weekday_cos'] = np.cos(2 * np.pi * df['Fake_date'].dt.weekday /
df['month_sin'] = np.sin(2 * np.pi * df['Fake_date'].dt.month / 12)
df['month_cos'] = np.cos(2 * np.pi * df['Fake_date'].dt.month / 12)
```

In [14]:

```
df.head()
```

Out[14]:

	Income	Age	Dependents	Occupation	City_Tier	Rent	Loan_
0	44637.249636	49	0	Self_Employed	Tier_1	13391.174891	
1	26858.596592	34	2	Retired	Tier_2	5371.719318	

2	50367.605084	35	1	Student	Tier_3	7555.140763	4
3	101455.600247	21	0	Self_Employed	Tier_3	15218.340037	6
4	24875.283548	52	4	Professional	Tier_2	4975.056710	3

5 rows × 33 columns

In [17]:

```
import warnings
warnings.filterwarnings('ignore')
```

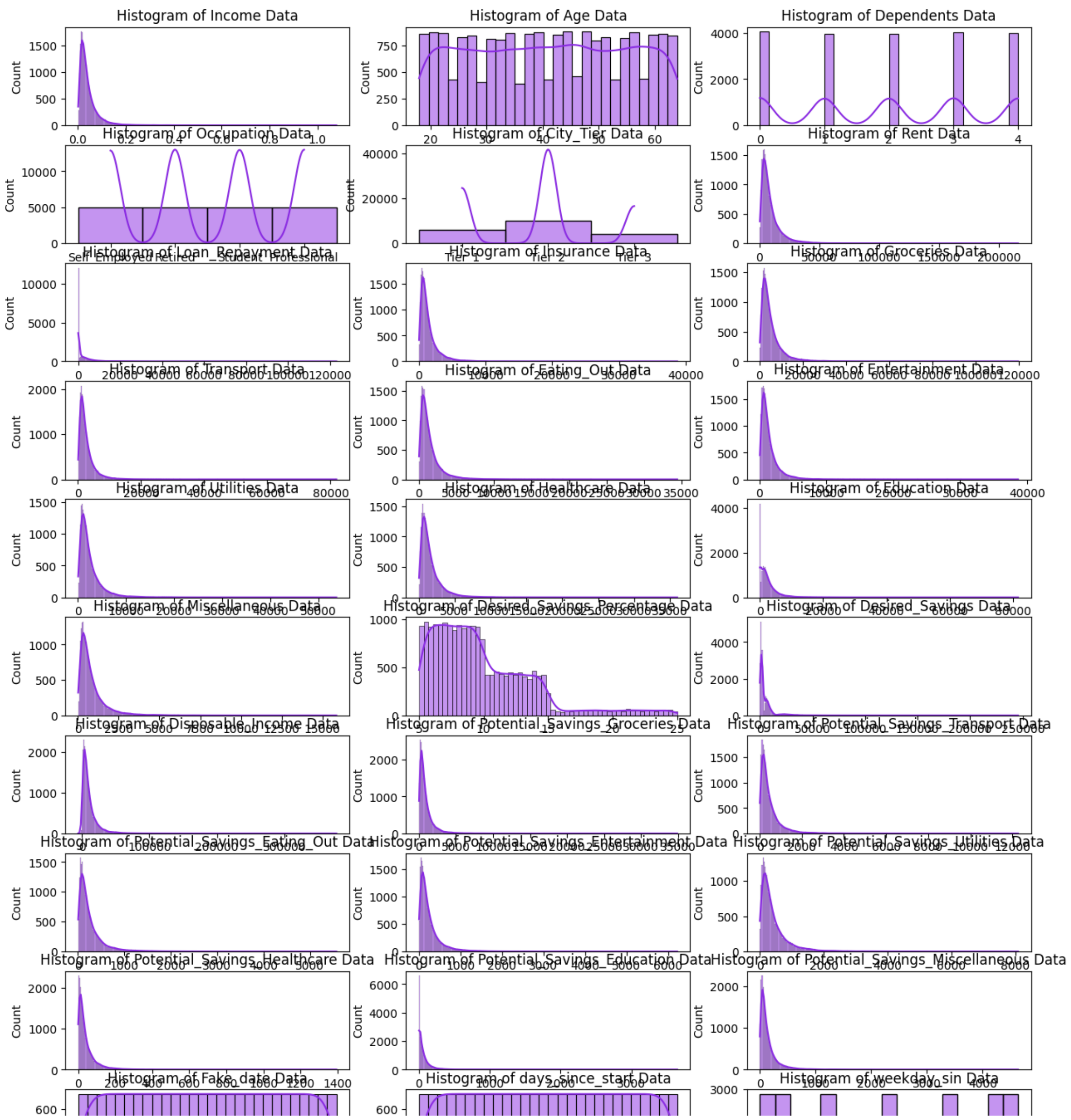
2.3 Exploratory Data Analysis(EDA)

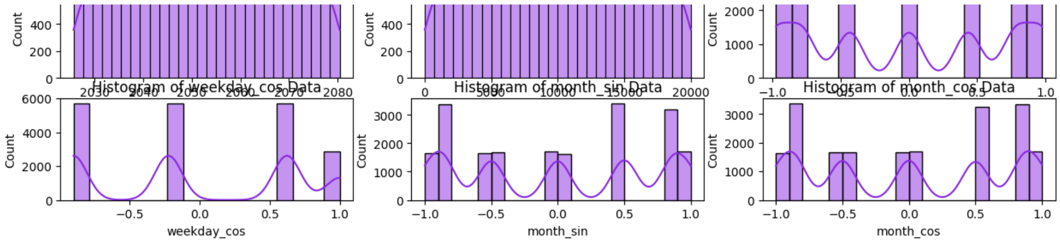
In [ ]:

```
```

In [18]:

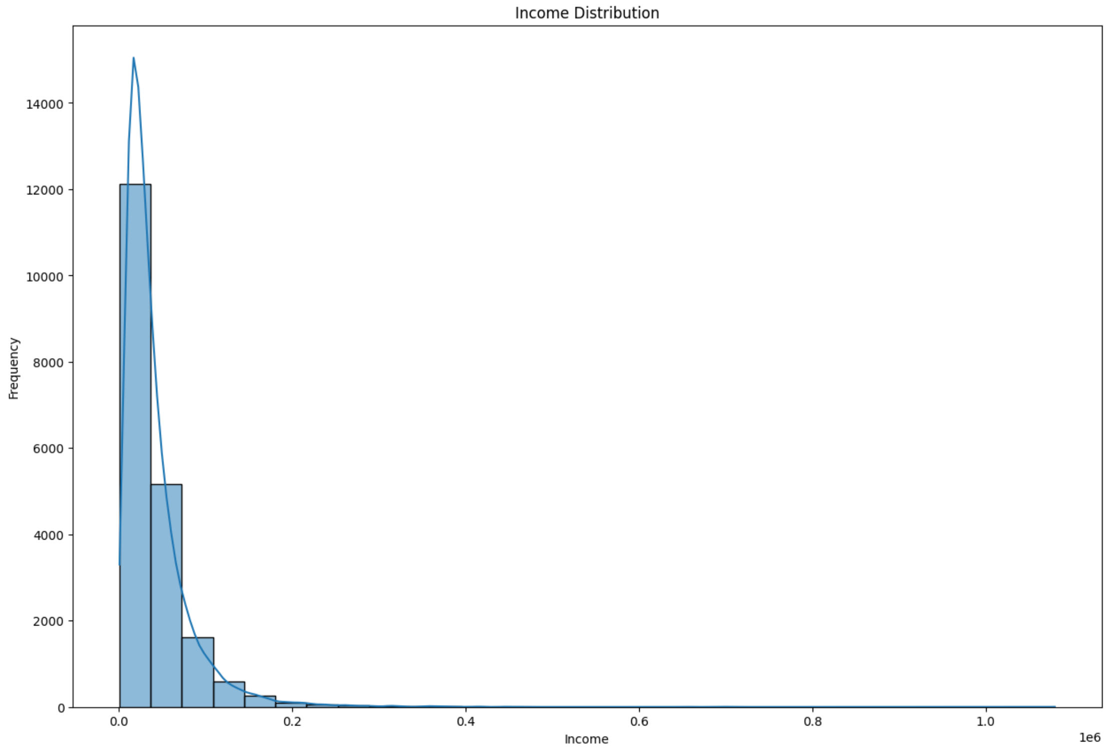
```
plt.figure(figsize = (15, 20))
for i, col in enumerate(df.columns, 1):
    plt.subplot(11,3 , i)
    sns.histplot(x = df[col],color="blueviolet",kde=True)
    plt.title(f"Histogram of {col} Data")
    plt.plot()
```





In [19]:

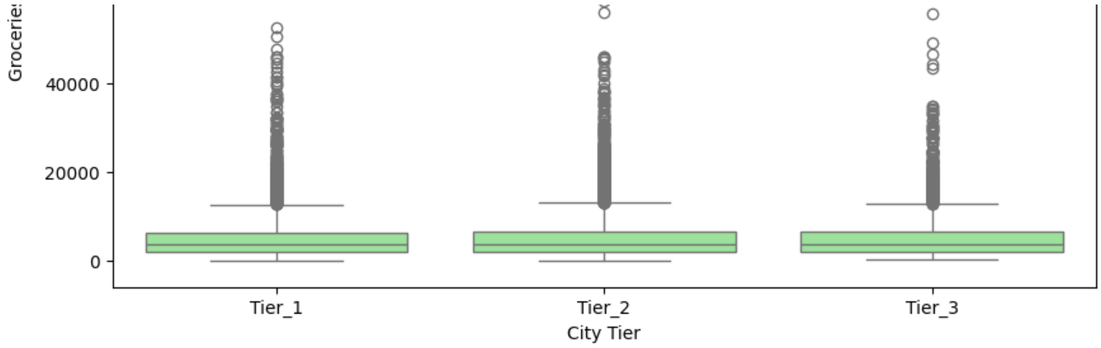
```
# Distribution of Income
plt.figure(figsize=(15, 10))
sns.histplot(df['Income'], bins=30, kde=True)
plt.title('Income Distribution')
plt.xlabel('Income')
plt.ylabel('Frequency')
plt.show()
```



In [ ]:

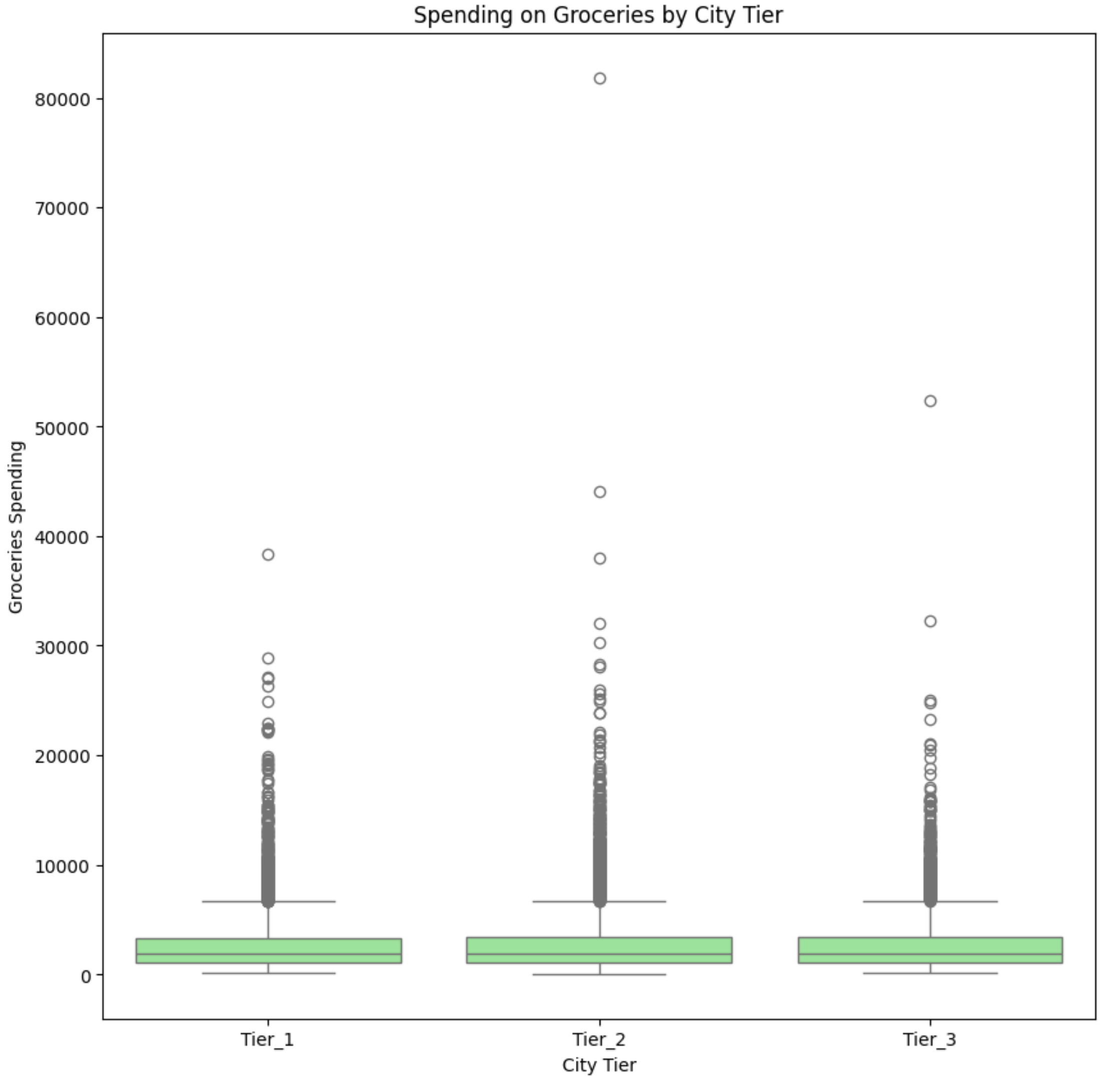
In [20]:

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='City_Tier', y='Groceries', data=df ,color="lightgreen
plt.title('Spending on Groceries by City Tier')
plt.xlabel('City Tier')
plt.ylabel('Groceries Spending')
plt.show()
```



In [19]:

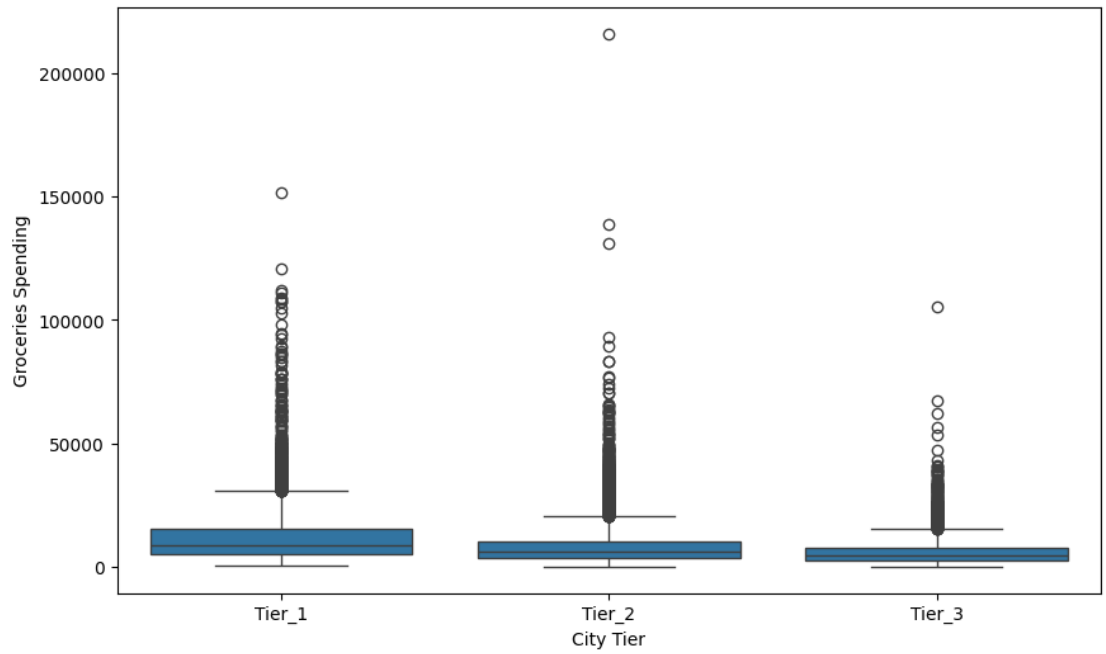
```
plt.figure(figsize=(10, 10))
sns.boxplot(x='City_Tier', y='Transport', data=df,color="lightgreen"
plt.title('Spending on Groceries by City Tier')
plt.xlabel('City Tier')
plt.ylabel('Groceries Spending')
plt.show()
```



In [19]:

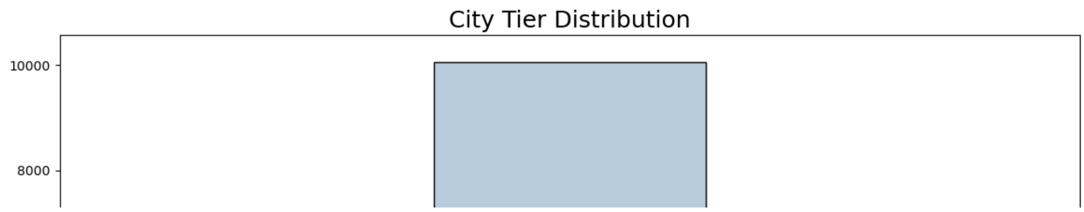
```
plt.figure(figsize=(10, 6))
sns.boxplot(x='City_Tier', y='Rent', data=df)
plt.title('Spending on Groceries by City Tier')
plt.xlabel('City Tier')
plt.ylabel('Groceries Spending')
plt.show()
```

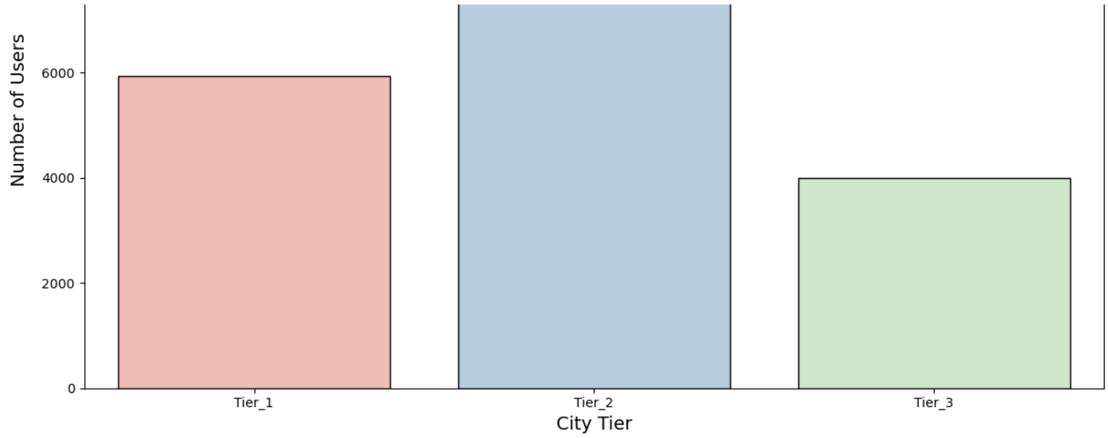
Spending on Groceries by City Tier



In [20]:
```
plt.figure(figsize=(14, 7))
sns.histplot(df['Age'], bins=50, kde=True, color='blueviolet')
plt.title('Age Distribution', fontsize=18)
plt.xlabel('Age', fontsize=14)
plt.ylabel('Number of Users', fontsize=14)
plt.tight_layout()
plt.show()
```

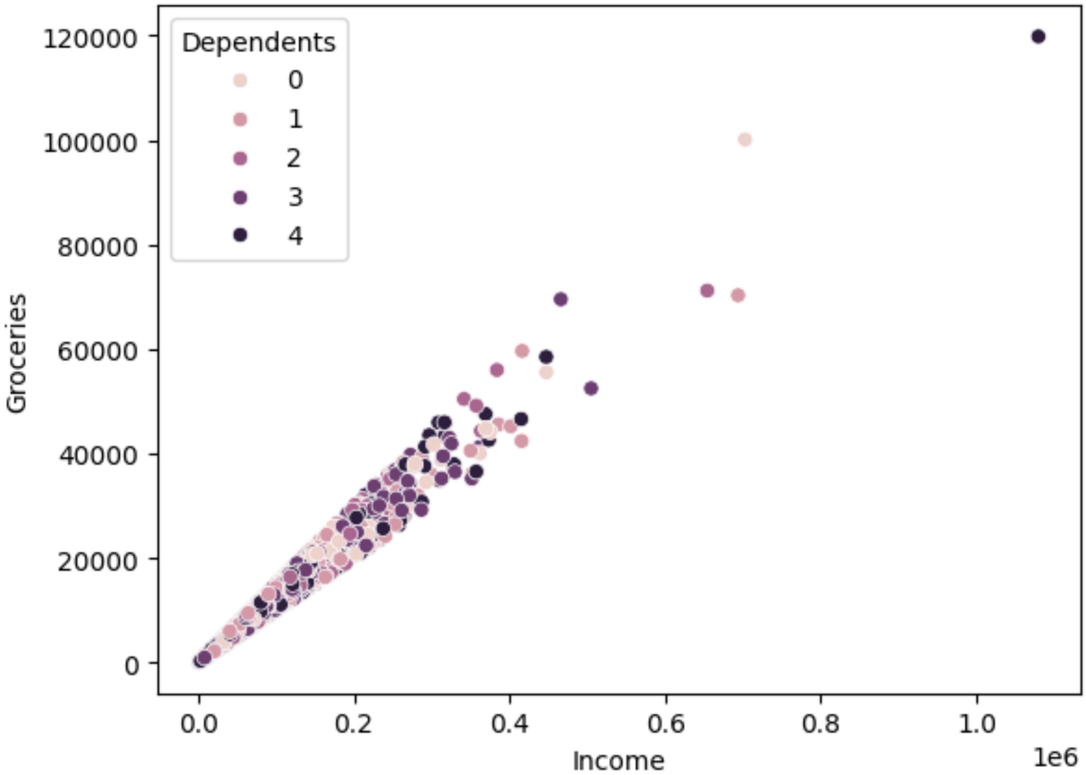
In [ ]:
```
plt.figure(figsize=(12, 7))
sns.countplot(x='City_Tier', data=df, palette='Pastel1', edgecolor='
plt.title('City Tier Distribution', fontsize=18)
plt.xlabel('City Tier', fontsize=14)
plt.ylabel('Number of Users', fontsize=14)
plt.tight_layout()
plt.show()
```





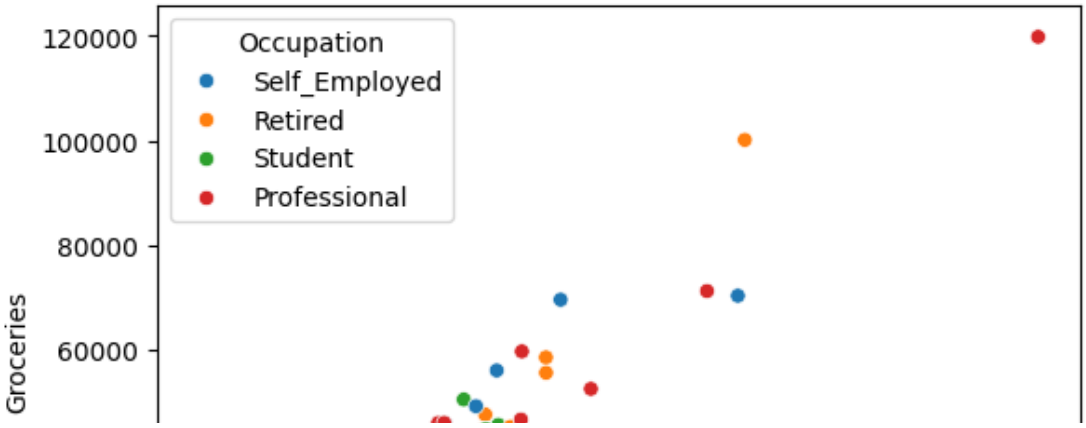
In [21]:

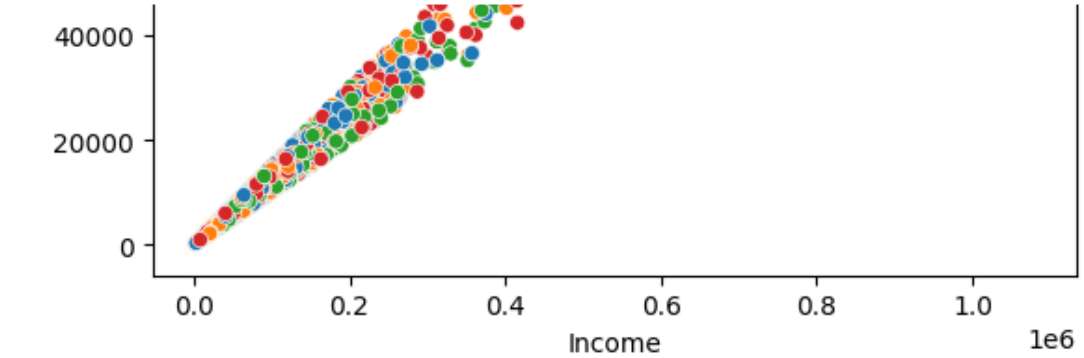
```
# Top correlation of target by "Dependents"
sns.scatterplot(x = df["Income"], y = df["Groceries"], hue = df["Dep
plt.show()
```



In [22]:

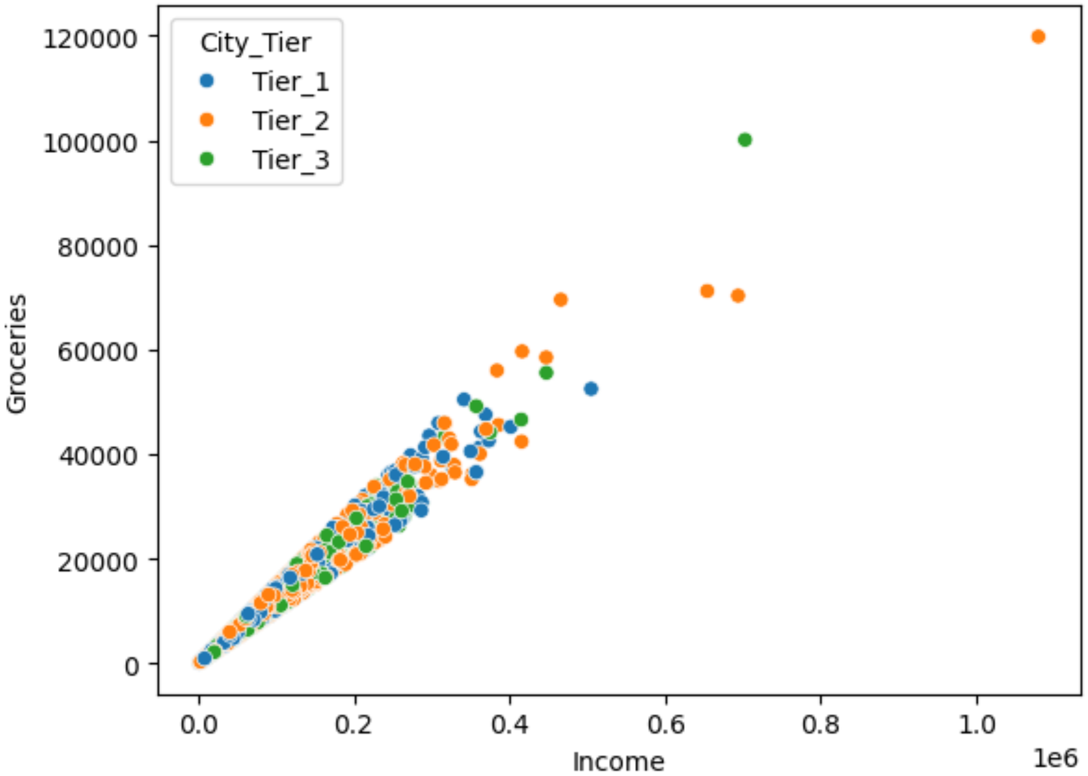
```
# Top correlation of target by "Occuptaion"
sns.scatterplot(x = df["Income"], y = df["Groceries"], hue = df["Occ
plt.show()
```





In [28]:

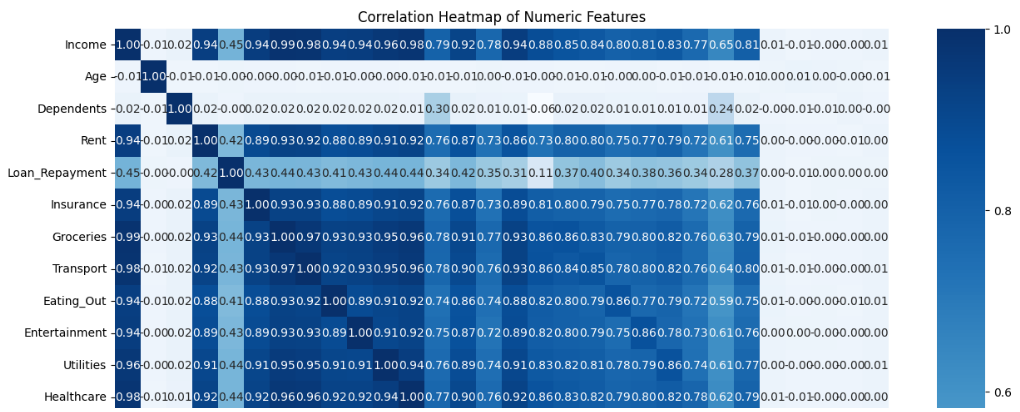
```
# Top correlation of target by "City_Tier"
sns.scatterplot(x = df["Income"], y = df["Groceries"], hue = df["Cit
plt.show()
```

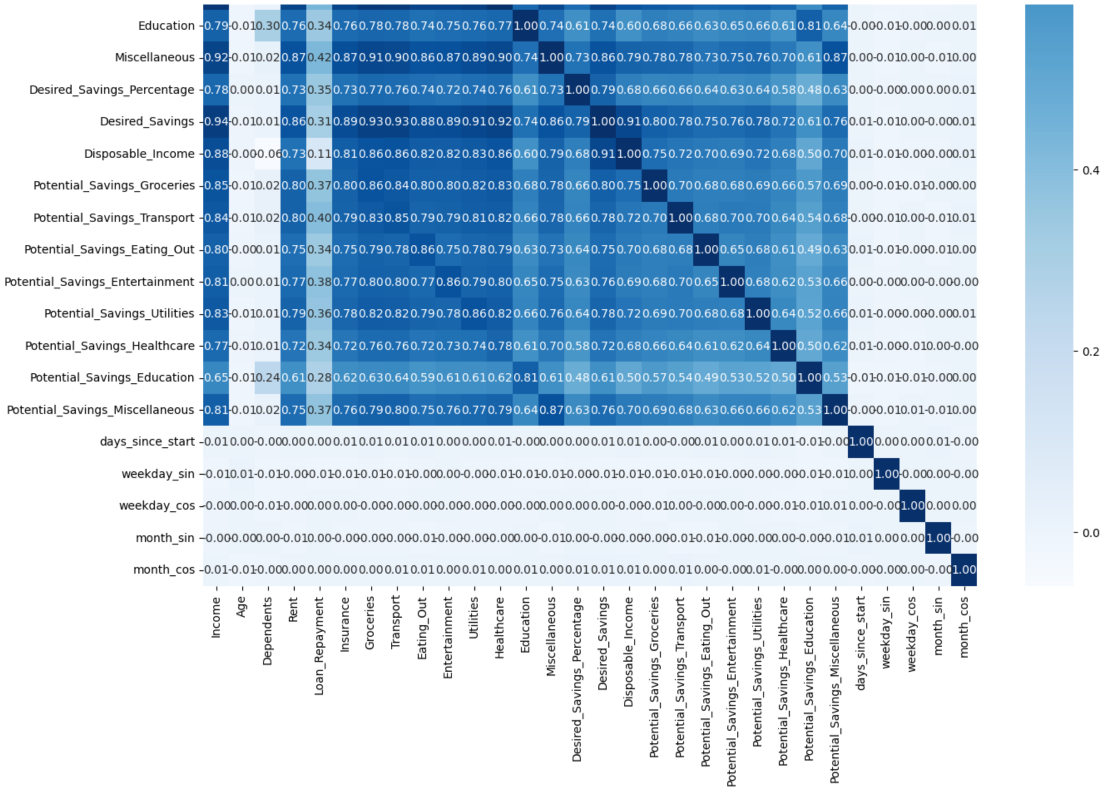


In [22]:

```
df_corr = df.select_dtypes(include=["number"]).corr()

plt.figure(figsize=(15, 15))
sns.heatmap(df_corr, fmt=".2f", annot=True, cmap="Blues")
plt.title("Correlation Heatmap of Numeric Features")
plt.show()
```





In [ ]:

2.4 Model Selection .

In [ ]:

In [23]:

```
import joblib

from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor, GradientBoosting
from sklearn.linear_model import LinearRegression
from sklearn.multioutput import MultiOutputRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error,
```

In [25]:

```
variable_expenses = [
    'Groceries', 'Transport', 'Eating_Out', 'Entertainment',
    'Utilities', 'Healthcare', 'Education', 'Miscellaneous'
]

target_columns = [f'Potential_Savings_{cat}' for cat in variable_exp

# Create mock target columns if not exist (10% of expense)
for i, col in enumerate(target_columns):
    if col not in df.columns:
        # backup model once (optional)
        shutil.copy("aidan.joblib", "aidan_old.joblib")
        df[col] = df[variable_expenses[i]] * 0.1

numerical_features = ['Income', 'Age', 'Dependents', 'Disposable_Inc
categorical_features = ['Occupation', 'City_Tier']
```

```
numerical_features = [col for col in numerical_features if col in df
categorical_features = [col for col in categorical_features if col i
target_columns = [col for col in target_columns if col in df.columns
```

In [32]:

```
# ==============================
encoder = OneHotEncoder(drop='first', sparse_output=False)
encoded_cats = encoder.fit_transform(df[categorical_features])
encoded_cat_columns = encoder.get_feature_names_out(categorical_feat
df_encoded_cats = pd.DataFrame(encoded_cats, columns=encoded_cat_col

# Combine numerical + encoded categorical
df_features = pd.concat([df[numerical_features], df_encoded_cats], a

# Scale numerical columns
scaler = StandardScaler()
df_scaled_numerical = pd.DataFrame(scaler.fit_transform(df_features[
                                   columns=numerical_features, index
df_features.update(df_scaled_numerical)
```

In [34]:

```
df_features.columns
```

Out[34]:

```
Index(['Income', 'Age', 'Dependents', 'Disposable_Income', 'Desired_
Savings',
       'Groceries', 'Transport', 'Eating_Out', 'Entertainment', 'Uti
lities',
       'Healthcare', 'Education', 'Miscellaneous', 'Occupation_Retir
ed',
       'Occupation_Self_Employed', 'Occupation_Student', 'City_Tier_
Tier_2',
       'City_Tier_Tier_3'],
      dtype='object')
```

In [35]:

```
X = df_features
y = df[target_columns]

# ==============================
# 4️⃣ Train / test split
# ==============================
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
```

In [36]:

```



models = {
    "Random Forest": MultiOutputRegressor(RandomForestRegressor(n_es
    "Linear Regression": MultiOutputRegressor(LinearRegression()),
    "Gradient Boosting": MultiOutputRegressor(GradientBoostingRegres
}

trained_pipelines = {}
results_metrics = {}

for name, model in models.items():
    print(f"\n🚀 Training {name}...")
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
```

```
    metrics = {}
    for i, target in enumerate(target_columns):
        mse = mean_squared_error(y_test[target], y_pred[:, i])
        mae = mean_absolute_error(y_test[target], y_pred[:, i])
        r2 = r2_score(y_test[target], y_pred[:, i])
        metrics[target] = {"MSE": mse, "MAE": mae, "R2": r2}

Skip to Main
new_pp
Last Checkpoint: 2 days ago
[Python 3 (ipykernel)]
Selection deleted
variable_expenses = [
    'Groceries', 'Transport', 'Eating_Out', 'Entertainment',
    'Utilities', 'Healthcare', 'Education', 'Miscellaneous'
]

target_columns = [f'Potential_Savings_{cat}' for cat in variable_exp

# Create mock target columns if not exist (10% of expense)
for i, col in enumerate(target_columns):
    if col not in df.columns:
        # backup model once (optional)
        shutil.copy("aidan.joblib", "aidan_old.joblib")
        df[col] = df[variable_expenses[i]] * 0.1

numerical_features = ['Income', 'Age', 'Dependents', 'Disposable_Inc
categorical_features = ['Occupation', 'City_Tier']

numerical_features = [col for col in numerical_features if col in df
categorical_features = [col for col in categorical_features if col i
target_columns = [col for col in target_columns if col in df.columns

Selection deleted
# ==============================
encoder = OneHotEncoder(drop='first', sparse_output=False)
encoded_cats = encoder.fit_transform(df[categorical_features])
encoded_cat_columns = encoder.get_feature_names_out(categorical_feat
df_encoded_cats = pd.DataFrame(encoded_cats, columns=encoded_cat_col

# Combine numerical + encoded categorical
df_features = pd.concat([df[numerical_features], df_encoded_cats], a

# Scale numerical columns
scaler = StandardScaler()
df_scaled_numerical = pd.DataFrame(scaler.fit_transform(df_features[
                                   columns=numerical_features, index
df_features.update(df_scaled_numerical)

df_scale_numerical.columns

---------------------------------------------------------------------
NameError                                 Traceback (most recent cal
Cell In[33], line 1
----> 1 df_scale_numerical.columns

Skip to Main
new_pp
Last Checkpoint: 2 days ago
[Python 3 (ipykernel)]
Selection deleted
variable_expenses = [
    'Groceries', 'Transport', 'Eating_Out', 'Entertainment',
```

```
    'Utilities', 'Healthcare', 'Education', 'Miscellaneous'
]

target_columns = [f'Potential_Savings_{cat}' for cat in variable_exp

# Create mock target columns if not exist (10% of expense)
for i, col in enumerate(target_columns):
    if col not in df.columns:
        # backup model once (optional)
        shutil.copy("aidan.joblib", "aidan_old.joblib")
        df[col] = df[variable_expenses[i]] * 0.1

numerical_features = ['Income', 'Age', 'Dependents', 'Disposable_Inc
categorical_features = ['Occupation', 'City_Tier']

numerical_features = [col for col in numerical_features if col in df
categorical_features = [col for col in categorical_features if col i
target_columns = [col for col in target_columns if col in df.columns

Selection deleted
# ==============================
encoder = OneHotEncoder(drop='first', sparse_output=False)
encoded_cats = encoder.fit_transform(df[categorical_features])
encoded_cat_columns = encoder.get_feature_names_out(categorical_feat
df_encoded_cats = pd.DataFrame(encoded_cats, columns=encoded_cat_col

# Combine numerical + encoded categorical
df_features = pd.concat([df[numerical_features], df_encoded_cats], a

# Scale numerical columns
scaler = StandardScaler()
df_scaled_numerical = pd.DataFrame(scaler.fit_transform(df_features[
                                   columns=numerical_features, index
df_features.update(df_scaled_numerical)

df_scale_numerical.columns

---------------------------------------------------------------------
NameError                                 Traceback (most recent cal
Cell In[33], line 1
----> 1 df_scale_numerical.columns

    df_metrics = pd.DataFrame(metrics).T
    results_metrics[name] = df_metrics
    trained_pipelines[name] = model

    print(f"✅ {name} training done. Metrics table:")
    display(df_metrics)  # Hii inaonyesha table kwa kila model

# ==============================
# 6️⃣ Save one pipeline for Streamlit
# ==============================
joblib.dump(trained_pipelines["Random Forest"], "aidan.joblib")
print("✅ Random Forest pipeline saved as 'aidan.joblib'. Ready for S
```

🚀 Training Random Forest...
✅ Random Forest training done. Metrics table:

	MSE	MAE	R2
Potential_Savings_Groceries	366761.750028	333.306220	0.701009
Potential_Savings_Transport	94107.279371	178.285721	0.689181
Potential_Savings_Eating_Out	23204.340636	91.541455	0.758104

Potential_Savings_Eating_Out	23204.340636	91.341455	0.738104
Potential_Savings_Entertainment	21526.562887	89.103868	0.748364
Potential_Savings_Utilities	69911.164855	156.839509	0.744183
Potential_Savings_Healthcare	1314.025237	21.022554	0.587155
Potential_Savings_Education	4126.500872	31.293443	0.680100
Potential_Savings_Miscellaneous	8798.768296	51.860658	0.737764

```
🚀 Training Linear Regression...
✅ Linear Regression training done. Metrics table:
```

	MSE	MAE	R2
Potential_Savings_Groceries	323128.434559	324.357258	0.736580
Potential_Savings_Transport	103468.419468	174.934140	0.658263
Potential_Savings_Eating_Out	22358.723054	89.985217	0.766920
Potential_Savings_Entertainment	23887.904097	88.465188	0.720760
Potential_Savings_Utilities	61589.369395	152.687398	0.774633
Potential_Savings_Healthcare	1163.252438	20.646174	0.634525
Potential_Savings_Education	4173.930660	31.353433	0.676424
Potential_Savings_Miscellaneous	8090.982135	51.426241	0.758859

```
🚀 Training Gradient Boosting...
✅ Gradient Boosting training done. Metrics table:
```

	MSE	MAE	R2
Potential_Savings_Groceries	356743.950600	331.353542	0.709176
Potential_Savings_Transport	89715.892076	174.999529	0.703685
Potential_Savings_Eating_Out	24930.382357	91.580103	0.740111
Potential_Savings_Entertainment	20341.073519	87.510251	0.762221
Potential_Savings_Utilities	71956.732813	156.728399	0.736697
Potential_Savings_Healthcare	1337.578433	21.004066	0.579755
Potential_Savings_Education	4210.798004	31.458101	0.673565
Potential_Savings_Miscellaneous	8773.465243	51.856419	0.738518

```
✅ Random Forest pipeline saved as 'aidan.joblib'. Ready for Streamlit
app.
```

In [ ]:

```

```

In [ ]:

```

```