Indian Personal Finance and Spending Habits Project

Brief Description of the Project: Developed and evaluated predictive regression models using deep learning Neural Networks, Machine Learning bagging and boosting algorithms like Random Forest and XGBoost on a large financial dataset. Applied data preprocessing, feature engineering and model tuning. Utilized performance metrics (MSE, MAE, R^2) and visualizations for model comparison. Demonstrated skills in Python, machine learning, data analysis, data science, model evaluation and visualization using libraries such as numpy, pandas, matplotlib, seaborn, scikit-learn, TensorFlow etc.

Importing necessary libraries

In [4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import ScalarFormatter, MaxNLocator
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestRegressor
from xgboost.sklearn import XGBRegressor
import xgboost as xgb
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Model
from tensorflow.keras.layers import (
    Dense, Dropout, LayerNormalization, MultiHeadAttention,Layer, GlobalAveragePooling1D, Input,Reshape, Lambda,Attention)
from tensorflow.keras.optimizers import Adam

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

Loading the dataset

In [6]:
```
df=pd.read_csv(r"D:\Data Science\Projects\1. Indian Personal Finance and Spending Habits Project\data.csv")
df.head()
```

Out[6]:

	Income	Age	Dependents	Occupation	City_Tier	Rent	Loan_Repayment	Insurance	Groceries	Transport	...	Desi
0	44637.249636	49	0	Self_Employed	Tier_1	13391.174891	0.000000	2206.490129	6658.768341	2636.970696	...	
1	26858.596592	34	2	Retired	Tier_2	5371.719318	0.000000	869.522617	2818.444460	1543.018778	...	
2	50367.605084	35	1	Student	Tier_3	7555.140763	4612.103386	2201.800050	6313.222081	3221.396403	...	
3	101455.600247	21	0	Self_Employed	Tier_3	15218.340037	6809.441427	4889.418087	14690.149363	7106.130005	...	1
4	24875.283548	52	4	Professional	Tier_2	4975.056710	3112.609398	635.907170	3034.329665	1276.155163	...	

5 rows × 27 columns

About Dataset

The dataset used contains detailed financial and demographic data for 20,000 individuals, focusing on income, expenses, and potential savings across various categories. The data aims to provide insights into personal financial management and spending patterns using the following:

Income: Monthly income in currency units (INR).
Age: Age of the individual.
Dependents: Number of dependents supported by the individual.
Occupation: Type of employment or job role.
City_Tier: A categorical variable representing the living area tier (e.g., Tier 1, Tier 2).
Monthly Expenses: Categories like Rent, Loan_Repayment, Insurance, Groceries, Transport, Eating_Out, Entertainment, Utilities, Healthcare, Education, and Miscellaneous record various monthly expenses.
Desired_Savings_Percentage and Desired_Savings: Targets for monthly savings.
Disposable_Income: Income remaining after all expenses are accounted for.
Potential Savings: Includes estimates of potential savings across different spending areas such as Groceries, Transport, Eating_Out, Entertainment, Utilities, Healthcare, Education, and Miscellaneous.

In [8]:
```
df.round(2)
```

Out[8]:

	Income	Age	Dependents	Occupation	City_Tier	Rent	Loan_Repayment	Insurance	Groceries	Transport	...	Desired_Savings
0	44637.25	49	0	Self_Employed	Tier_1	13391.17	0.00	2206.49	6658.77	2636.97	...	6200.54
1	26858.60	34	2	Retired	Tier_2	5371.72	0.00	869.52	2818.44	1543.02	...	1923.18
2	50367.61	35	1	Student	Tier_3	7555.14	4612.10	2201.80	6313.22	3221.40	...	7050.36
3	101455.60	21	0	Self_Employed	Tier_3	15218.34	6809.44	4889.42	14690.15	7106.13	...	16694.97
4	24875.28	52	4	Professional	Tier_2	4975.06	3112.61	635.91	3034.33	1276.16	...	1874.10
...	...	...	...	...	...	...	...	...	...	...	...	...
19995	40913.47	51	4	Self_Employed	Tier_1	12274.04	7703.85	1646.80	5477.40	2084.23	...	1163.32
19996	90295.77	21	1	Student	Tier_2	18059.15	0.00	2770.59	13118.22	4633.11	...	10613.59
19997	40604.57	30	1	Professional	Tier_2	8120.91	8089.61	1548.56	6018.28	2493.56	...	2267.91
19998	118157.82	27	2	Professional	Tier_1	35447.35	12345.91	4735.67	16392.44	8248.24	...	10603.68
19999	8209.25	62	3	Professional	Tier_1	2462.77	1120.88	276.38	969.92	460.44	...	531.04

20000 rows × 27 columns

In [9]:
```
df.shape
```

Out[9]: (20000, 27)

In [10]:
```
df.size
```

Out[10]: 540000

In [11]:
```
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 [12]:
```
df.describe()
```

Out[12]:

	Income	Age	Dependents	Rent	Loan_Repayment	Insurance	Groceries	Transport	Eating_Out	E
count	2.000000e+04	20000.000000	20000.000000	20000.000000	20000.000000	20000.000000	20000.000000	20000.000000	20000.000000	
mean	4.158550e+04	41.031450	1.995950	9115.494629	2049.800292	1455.028761	5205.667493	2704.466685	1461.856982	
std	4.001454e+04	13.578725	1.417616	9254.228188	4281.789941	1492.938435	5035.953689	2666.345648	1481.660811	
min	1.301187e+03	18.000000	0.000000	235.365692	0.000000	30.002012	154.078240	81.228584	39.437523	
25%	1.760488e+04	29.000000	1.000000	3649.422246	0.000000	580.204749	2165.426419	1124.578012	581.011801	
50%	3.018538e+04	41.000000	2.000000	6402.751824	0.000000	1017.124681	3741.091535	1933.845509	1029.109726	
75%	5.176545e+04	53.000000	3.000000	11263.940492	2627.142320	1787.160895	6470.892718	3360.597508	1807.075251	
max	1.079728e+06	64.000000	4.000000	215945.674703	123080.682009	38734.932935	119816.898124	81861.503457	34406.100166	

8 rows × 25 columns

Handling missing values

In [14]:
```
df.isna().sum()
```

Out[14]:
```
Income                           0
Age                              0
Dependents                       0
Occupation                       0
City_Tier                        0
Rent                             0
Loan_Repayment                   0
Insurance                        0
Groceries                        0
Transport                        0
Eating_Out                       0
Entertainment                    0
Utilities                        0
Healthcare                       0
Education                        0
Miscellaneous                    0
Desired_Savings_Percentage       0
Desired_Savings                  0
Disposable_Income                0
Potential_Savings_Groceries      0
Potential_Savings_Transport      0
Potential_Savings_Eating_Out     0
Potential_Savings_Entertainment  0
Potential_Savings_Utilities      0
Potential_Savings_Healthcare     0
Potential_Savings_Education      0
Potential_Savings_Miscellaneous  0
dtype: int64
```

Handling duplicate values

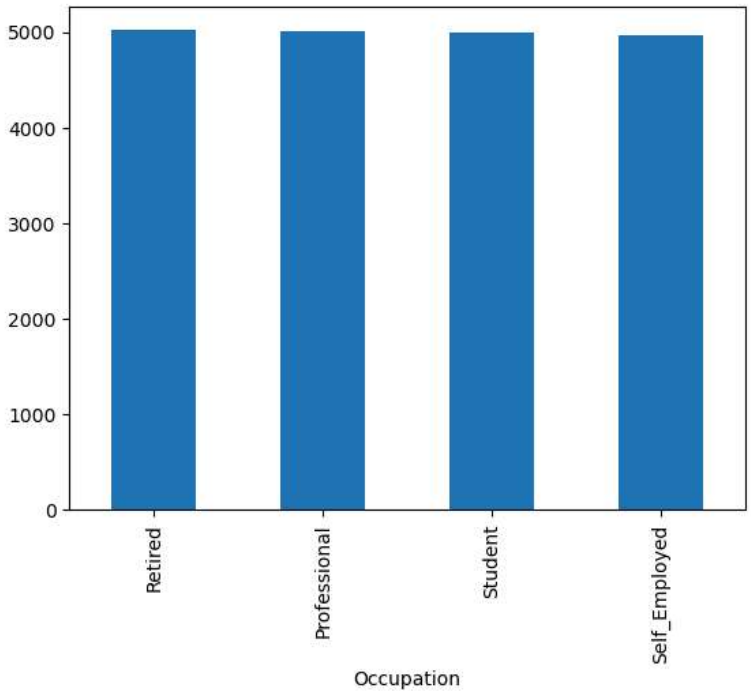
In [16]:
```
df.duplicated().any()
```

Out[16]:
```
False
```

EDA with visualizations

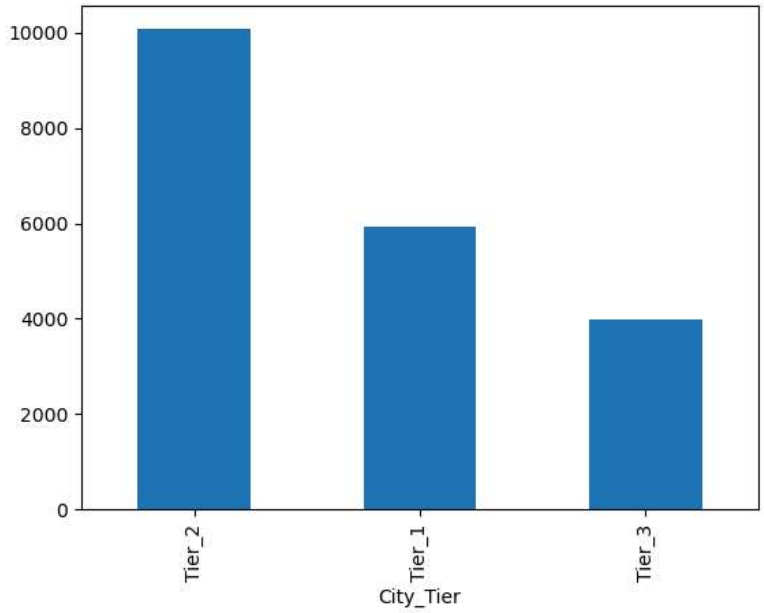
In [18]:
```
df['Occupation'].value_counts().plot(kind='bar')
```

Out[18]:
```
<Axes: xlabel='Occupation'>
```

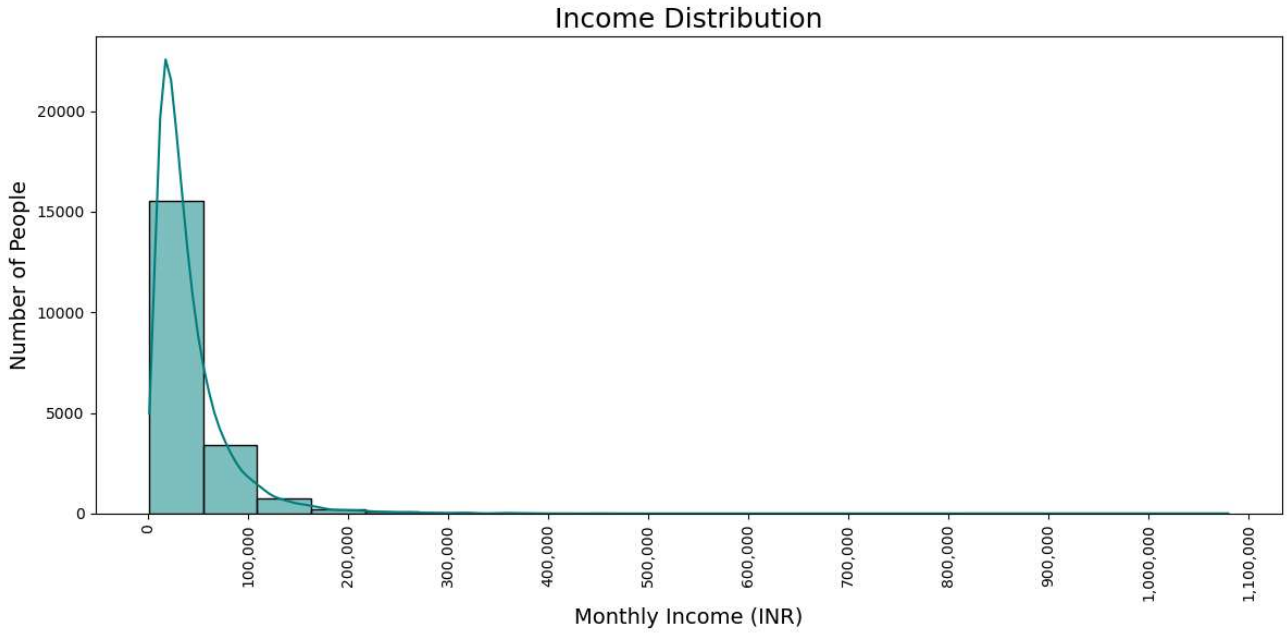


In [19]:
```
df['City_Tier'].value_counts().plot(kind='bar')
```

Out[19]: <Axes: xlabel='City_Tier'>



In [20]:
```
plt.figure(figsize=(12, 6))
sns.histplot(df['Income'], bins=20, kde=True, color='teal')

ax = plt.gca()
ax.xaxis.set_major_formatter(ScalarFormatter())  # Disable scientific notation
ax.ticklabel_format(style='plain', axis='x')
ax.xaxis.set_major_locator(MaxNLocator(nbins=12))  # Increase number of x-ticks

# Rotate x-axis labels vertically and round them
xticks = ax.get_xticks()
ax.set_xticklabels([f"{int(x):,}" for x in xticks], rotation=90)

plt.title('Income Distribution', fontsize=18)
plt.xlabel('Monthly Income (INR)', fontsize=14)
plt.ylabel('Number of People', fontsize=14)
plt.tight_layout()
plt.show()
```



In [21]:
```
# Change type of categorical features with numerical ones
le = LabelEncoder()

df['Occupation'] = le.fit_transform(df['Occupation'])
print("Occupation mapping:", dict(zip(le.classes_, le.transform(le.classes_))))

df['City_Tier'] = le.fit_transform(df['City_Tier'])
print("City_Tier mapping:", dict(zip(le.classes_, le.transform(le.classes_))))

df.head()
```

```
Occupation mapping: {'Professional': 0, 'Retired': 1, 'Self_Employed': 2, 'Student': 3}
City_Tier mapping: {'Tier_1': 0, 'Tier_2': 1, 'Tier_3': 2}
```

Out[21]:

	Income	Age	Dependents	Occupation	City_Tier	Rent	Loan_Repayment	Insurance	Groceries	Transport	...	Desire
0	44637.249636	49	0	2	0	13391.174891	0.000000	2206.490129	6658.768341	2636.970696	...	62
1	26858.596592	34	2	1	1	5371.719318	0.000000	869.522617	2818.444460	1543.018778	...	19
2	50367.605084	35	1	3	2	7555.140763	4612.103386	2201.800050	6313.222081	3221.396403	...	70
3	101455.600247	21	0	2	2	15218.340037	6809.441427	4889.418087	14690.149363	7106.130005	...	166
4	24875.283548	52	4	0	1	4975.056710	3112.609398	635.907170	3034.329665	1276.155163	...	18

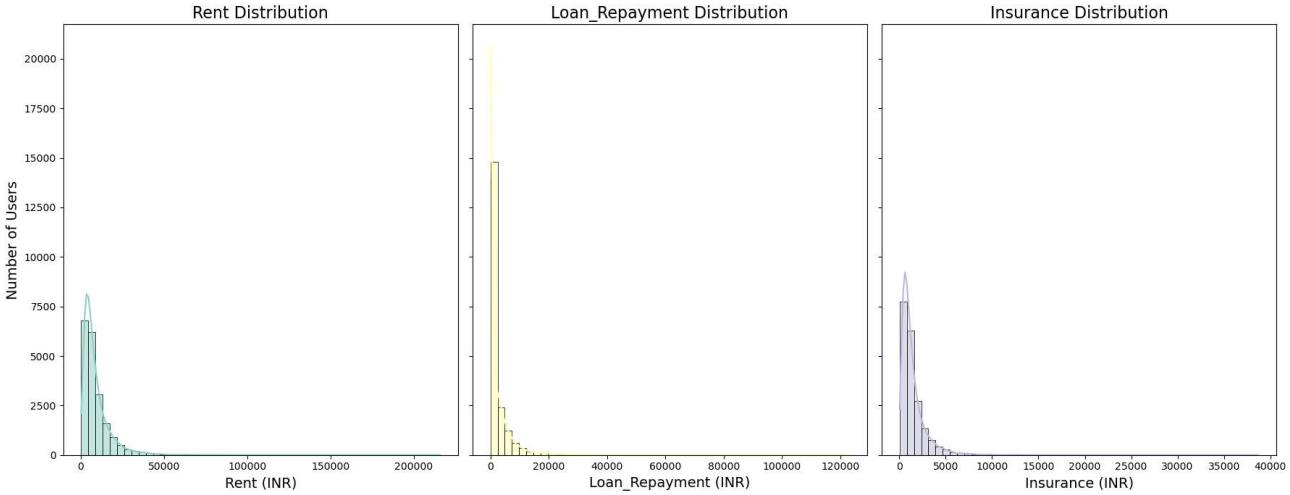
5 rows × 27 columns

Expense Analysis

1. Fixed Expenses

Fixed expenses include Rent, Loan Repayment, and Insurance. Let's analyze their distributions.

In [24]:
```
fixed_expenses = ['Rent', 'Loan_Repayment', 'Insurance']
num_fixed = len(fixed_expenses)

fixed_palette = sns.color_palette("Set3", num_fixed)

fig, axes = plt.subplots(1, num_fixed, figsize=(18, 7), sharey=True)

for ax, expense, color in zip(axes, fixed_expenses, fixed_palette):
    sns.histplot(df[expense], bins=50, kde=True, color=color, ax=ax)
    ax.set_title(f'{expense} Distribution', fontsize=16)
    ax.set_xlabel(f'{expense} (INR)', fontsize=14)
    ax.set_ylabel('Number of Users', fontsize=14)

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



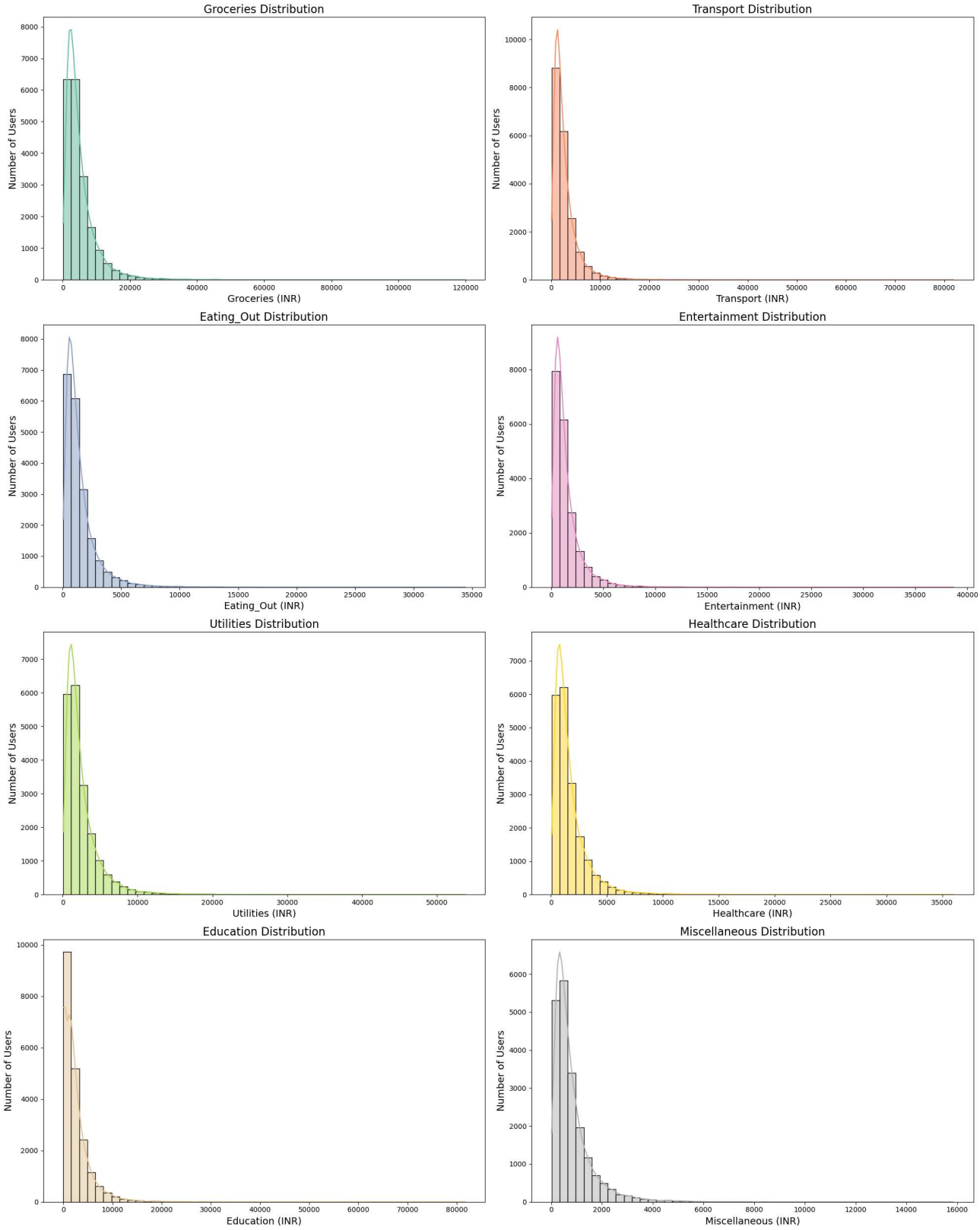
Observations:

- **Rent Distribution**:
  - The majority of users pay a relatively low rent, with a sharp peak at lower rent values (likely below 20,000 INR).
  - The distribution is right-skewed, with a small number of users paying higher rents, extending up to 200,000 INR.
  - This could suggest that a significant portion of users live in less expensive regions or that housing affordability varies widely, possibly due to differences between urban and rural areas.
- **Loan Repayment Distribution**:
  - A prominent spike at or near zero indicates that a substantial proportion of users have no loan repayment obligations.
  - For those with loan repayments, the amounts are relatively low, with a rapid decline in the number of users as repayment amounts increase.
  - The overall distribution is highly right-skewed, showing that few users have high monthly loan repayments (up to 120,000 INR).
- **Insurance Distribution**:
  - Insurance premiums are generally low, with the majority of users paying minimal amounts (likely below 5,000 INR).
  - Similar to the other distributions, this data is also right-skewed, indicating that while most users pay small amounts for insurance, a few pay significantly higher premiums, extending up to 40,000 INR.
  - This suggests that a large proportion of users may have basic or minimal insurance coverage, with fewer opting for comprehensive or higher-value plans.

2. Variable Expenses

Variable expenses include Groceries, Transport, Eating Out, Entertainment, Utilities, Healthcare, Education, and Miscellaneous.

In [27]:
```
variable_expenses = [
    'Groceries', 'Transport', 'Eating_Out', 'Entertainment',
    'Utilities', 'Healthcare', 'Education', 'Miscellaneous'
]
num_variable = len(variable_expenses)

variable_palette = sns.color_palette("Set2", num_variable)

fig, axes = plt.subplots(4, 2, figsize=(20, 25))
axes = axes.flatten()

for ax, expense, color in zip(axes, variable_expenses, variable_palette):
    sns.histplot(df[expense], bins=50, kde=True, color=color, ax=ax)
    ax.set_title(f'{expense} Distribution', fontsize=16)
    ax.set_xlabel(f'{expense} (INR)', fontsize=14)
    ax.set_ylabel('Number of Users', fontsize=14)

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



Observations:

- **Groceries**: Most users spend less than 20,000 INR on groceries, with a few outliers spending significantly more. This indicates that the majority of users maintain moderate grocery expenses.
- **Transport**: The majority of users have transport expenses below 10,000 INR, with a sharp decline beyond this point, suggesting limited high transport expenditures among users.

- **Eating Out**: Most users spend below 5,000 INR on eating out, with a few spending more, indicating that dining out is not a significant expense for most.
- **Entertainment**: Entertainment expenses are generally below 5,000 INR, with the majority of users spending minimal amounts in this category, reflecting lower discretionary spending.
- **Utilities**: Users primarily have utility expenses under 10,000 INR, with a few outliers, suggesting stable and predictable utility costs for most.
- **Healthcare**: Healthcare expenses are generally low, with most users spending less than 5,000 INR, indicating limited spending on medical needs among the majority.
- **Education**: Education-related expenses are primarily below 10,000 INR, indicating that high educational costs are rare among users.
- **Miscellaneous**: Most users have miscellaneous expenses below 2,000 INR, with a few higher expenditures, indicating that additional unclassified expenses are typically small.

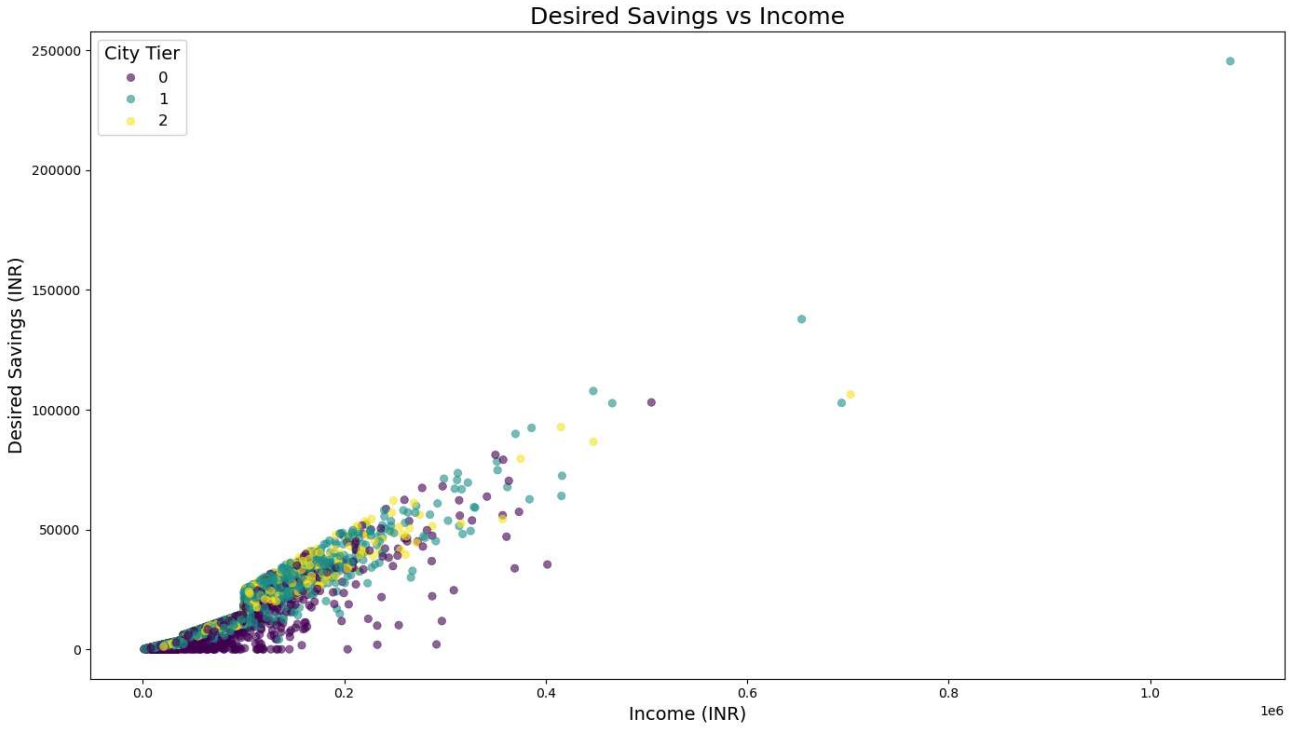
Each of these expense categories shows a right-skewed distribution, with the majority of users having relatively low expenditures and a smaller number spending significantly more. This suggests that while most users manage their expenses conservatively, there are outliers with higher spending in each category.

Savings Analysis

1. Desired Savings vs Income

Analyzing how desired savings correlate with income levels.

In [31]:
```
plt.figure(figsize=(14, 8))
sns.scatterplot(
    data=df,
    x='Income',
    y='Desired_Savings',
    hue='City_Tier',
    palette='viridis',
    alpha=0.6,
    edgecolor=None
)
plt.title('Desired Savings vs Income', fontsize=18)
plt.xlabel('Income (INR)', fontsize=14)
plt.ylabel('Desired Savings (INR)', fontsize=14)
plt.legend(title='City Tier', fontsize=12, title_fontsize=14)
plt.tight_layout()
plt.show()
```



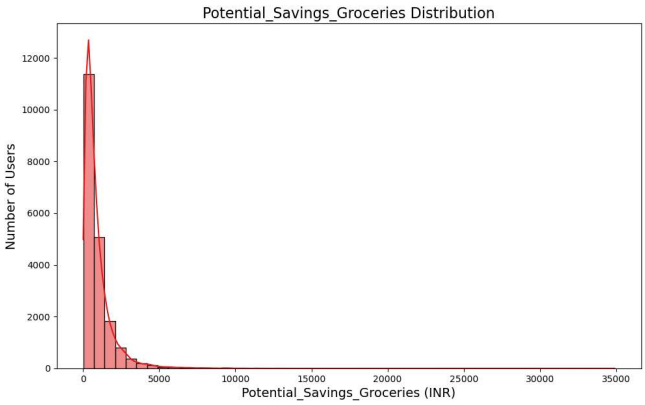
Observations:

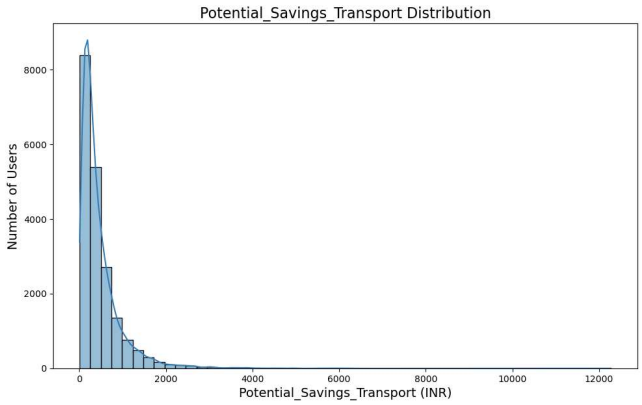
- There is a positive correlation between income and desired savings, with higher incomes leading to higher savings goals.
- Users across all city tiers follow similar savings trends, but higher-income users show more variability in their savings.
- Most users are concentrated at lower income levels, with a few outliers having significantly higher incomes and desired savings.

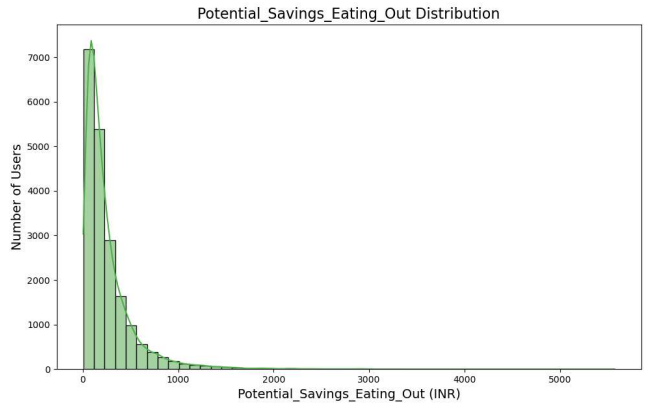
2. Potential Savings Across Categories

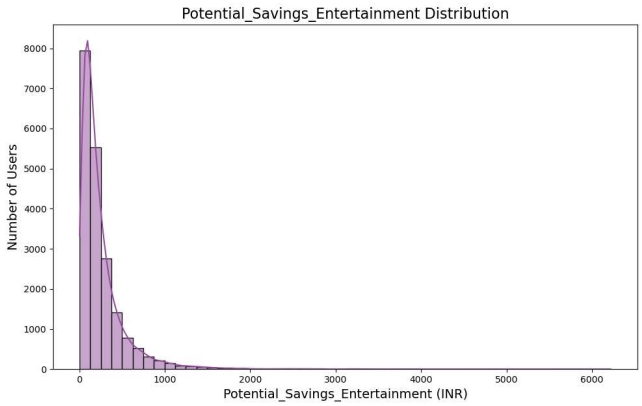
Understanding how much users can potentially save in each expense category.

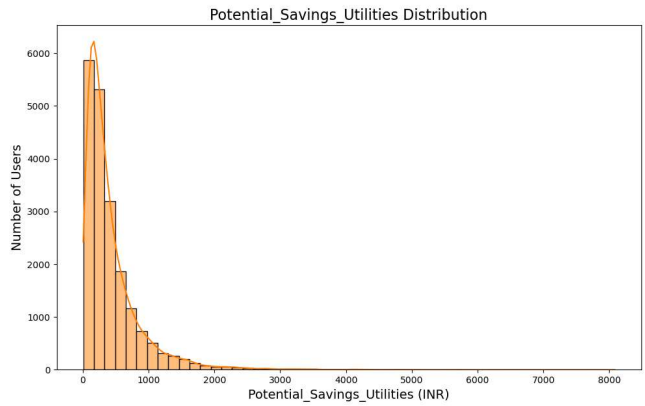
In [34]:
```
potential_savings_columns = [col for col in df.columns if col.startswith('Potential_Savings')]

num_potential = len(potential_savings_columns)
palette_potential = sns.color_palette("Set1", num_potential)

fig, axes = plt.subplots(4, 2, figsize=(20, 25))
axes = axes.flatten()

for ax, column, color in zip(axes, potential_savings_columns, palette_potential):
    sns.histplot(df[column], bins=50, kde=True, color=color, ax=ax)
    ax.set_title(f'{column} Distribution', fontsize=16)
    ax.set_xlabel(f'{column} (INR)', fontsize=14)
    ax.set_ylabel('Number of Users', fontsize=14)

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

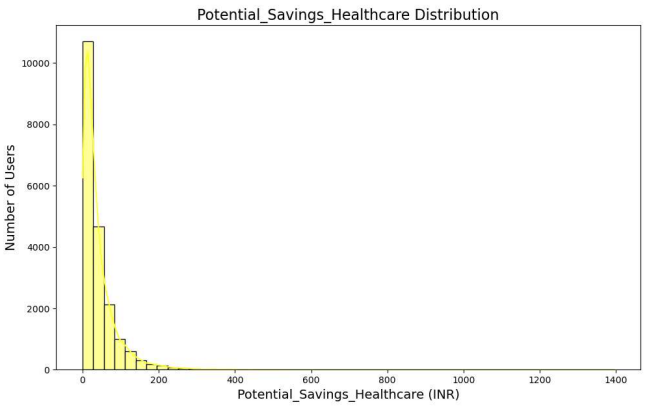


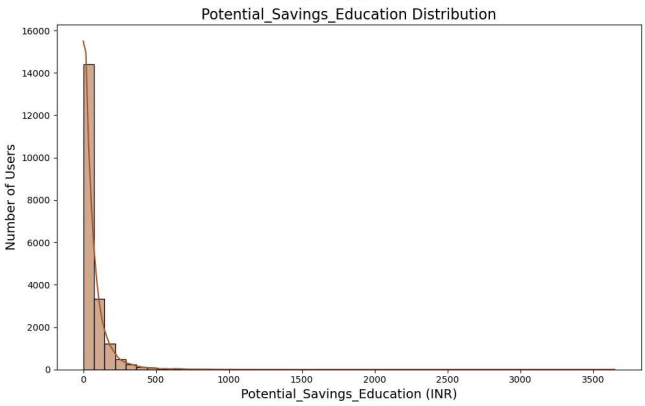


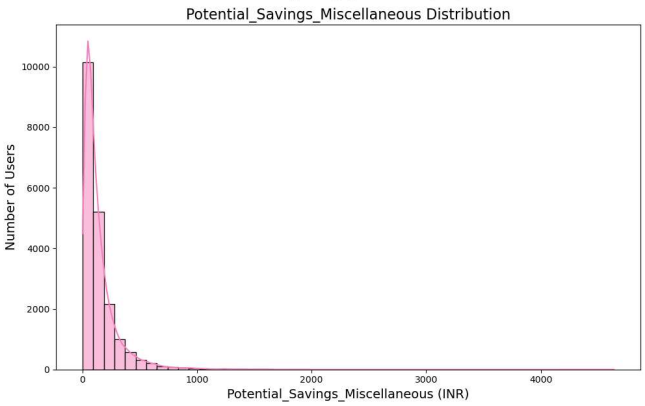












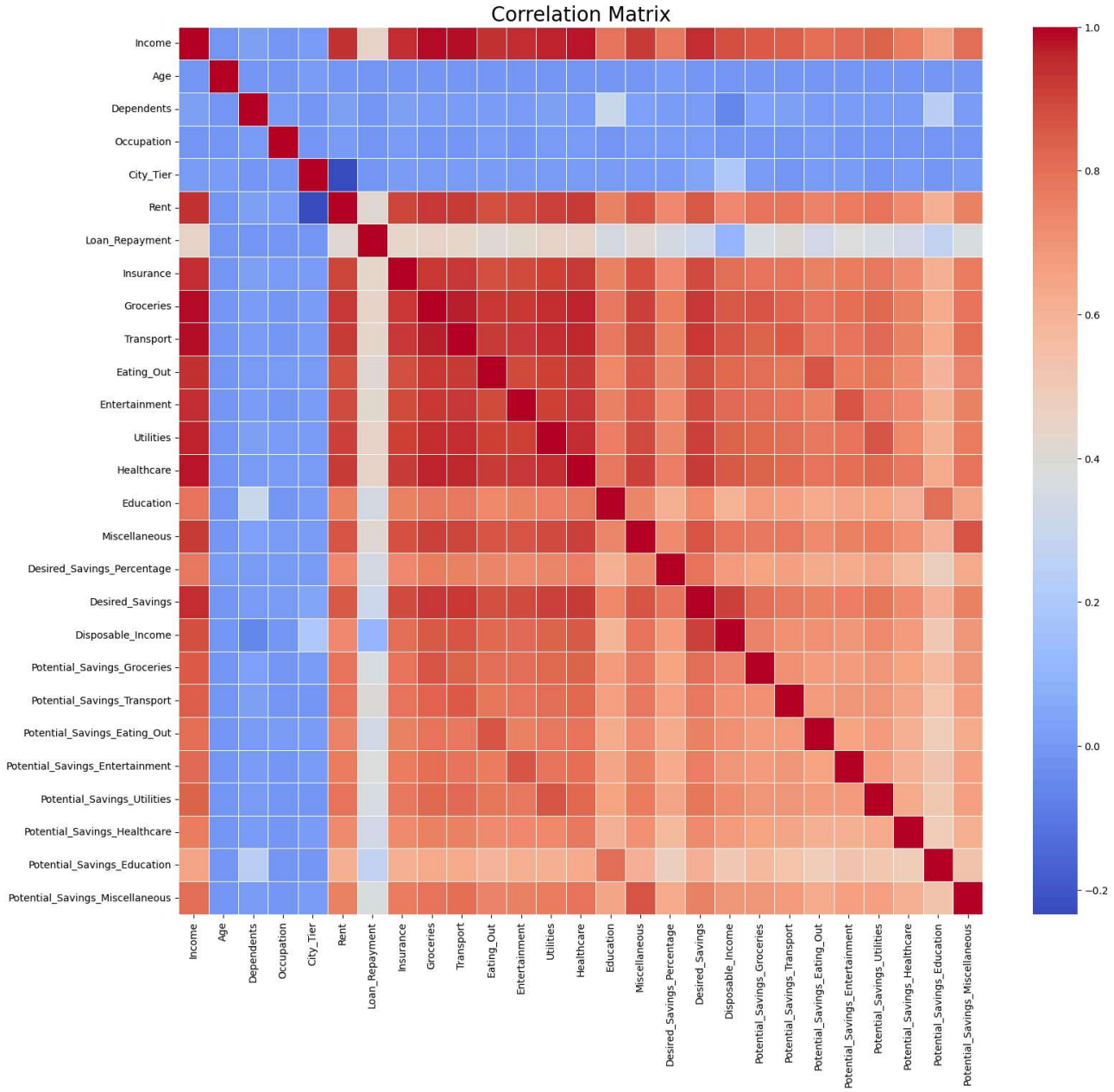
Observations:

- **Potential Savings Distribution**: Across all expense categories, potential savings are highly skewed towards lower values, indicating that most users have limited savings potential in each category.
- **Groceries and Transport**: These categories show relatively higher potential savings compared to others, suggesting room for users to optimize spending in these areas.
- **Healthcare and Education**: Potential savings are minimal, reflecting the essential nature of these expenses, where users have less flexibility to cut costs.

Correlation Analysis

Analyzing correlations between different financial aspects.

In [37]:
```
numeric_df = df.select_dtypes(include=[np.number])

correlation_matrix = numeric_df.corr()

plt.figure(figsize=(18, 16))
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix', fontsize=20)
plt.show()
```



Observations:

- **Income Correlations**: Income shows a strong positive correlation with expenses like rent, loan repayment, and insurance, indicating that higher earners tend to spend more on these fixed costs.
- **Desired Savings**: Desired savings correlate positively with income and disposable income, suggesting that as income increases, so do savings goals.
- **Expense Relationships**: There are strong correlations among different expense categories (e.g., groceries, transport, and utilities), reflecting that higher spending in one area is often associated with increased spending in others.

MODEL

The following code is written to prepare the dataset so as to train a regression model to predict a person's income based on their age, dependents, spending patterns (rent, groceries, healthcare, etc.), and categorical details like occupation and city tier — without using any features that would directly reveal the income.

In [40]:
```
# Drop target and leakage columns
leakage_cols = ['Income', 'Desired_Savings', 'Disposable_Income'] + \
               [col for col in df.columns if col.startswith('Potential_Savings_')]

X = df.drop(columns=leakage_cols)
y = df['Income']

# Handle categorical variables (if not done already)
X = pd.get_dummies(X, drop_first=True)

# Split data: 80% train, 20% test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

print("Training set shape:", X_train.shape)
print("Testing set shape:", X_test.shape)
```

```
Training set shape: (16000, 16)
Testing set shape: (4000, 16)
```

In [41]:
```
# Basic architecture example in Keras
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1)  # Output layer for regression
])
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2)
```

```
Epoch 1/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 827038720.0000 - val_loss: 18759602.0000
Epoch 2/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 15755330.0000 - val_loss: 12280856.0000
Epoch 3/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 12220342.0000 - val_loss: 14209585.0000
Epoch 4/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 13005768.0000 - val_loss: 11015941.0000
Epoch 5/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 11551525.0000 - val_loss: 10087413.0000
Epoch 6/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 9795587.0000 - val_loss: 9531169.0000
Epoch 7/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 10892719.0000 - val_loss: 9663580.0000
Epoch 8/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 11083889.0000 - val_loss: 10292575.0000
Epoch 9/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 10178866.0000 - val_loss: 8887091.0000
Epoch 10/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 10071924.0000 - val_loss: 9409807.0000
Epoch 11/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 10400669.0000 - val_loss: 8916404.0000
Epoch 12/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 11777877.0000 - val_loss: 8394945.0000
Epoch 13/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 9885307.0000 - val_loss: 9577307.0000
Epoch 14/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 10306809.0000 - val_loss: 11529548.0000
Epoch 15/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 9734438.0000 - val_loss: 8508326.0000
Epoch 16/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 9245658.0000 - val_loss: 8442904.0000
Epoch 17/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 8857223.0000 - val_loss: 7587039.5000
Epoch 18/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 8712387.0000 - val_loss: 11923117.0000
Epoch 19/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 9606027.0000 - val_loss: 10375674.0000
Epoch 20/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 8440605.0000 - val_loss: 7940784.5000
Epoch 21/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 8843249.0000 - val_loss: 7305122.0000
Epoch 22/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 8291954.0000 - val_loss: 7861088.5000
Epoch 23/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 8562103.0000 - val_loss: 7130211.0000
Epoch 24/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 8032102.5000 - val_loss: 7847166.5000
Epoch 25/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 8284518.5000 - val_loss: 7211447.0000
Epoch 26/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 8558352.0000 - val_loss: 7644301.5000
Epoch 27/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 8329249.5000 - val_loss: 6710542.5000
Epoch 28/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 7091603.0000 - val_loss: 6712055.5000
Epoch 29/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 7082339.0000 - val_loss: 5825315.0000
Epoch 30/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 7112308.0000 - val_loss: 6352855.0000
Epoch 31/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 7455394.5000 - val_loss: 7186653.0000
Epoch 32/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 6590338.5000 - val_loss: 5401408.0000
Epoch 33/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 6481160.0000 - val_loss: 5010552.5000
Epoch 34/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 6729386.5000 - val_loss: 5444295.0000
Epoch 35/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 6536731.0000 - val_loss: 4733209.5000
Epoch 36/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5968431.0000 - val_loss: 4544226.5000
Epoch 37/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5789643.0000 - val_loss: 4627905.5000
Epoch 38/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5396998.5000 - val_loss: 4371063.5000
Epoch 39/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5193629.0000 - val_loss: 4875579.5000
Epoch 40/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5572034.5000 - val_loss: 4089654.5000
Epoch 41/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5442153.5000 - val_loss: 4758617.5000
Epoch 42/100
```

```
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5475272.5000 - val_loss: 3914808.2500
Epoch 43/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5141782.5000 - val_loss: 3962231.0000
Epoch 44/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5979738.0000 - val_loss: 3777538.5000
Epoch 45/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4508234.5000 - val_loss: 4965709.5000
Epoch 46/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5564504.0000 - val_loss: 3766207.7500
Epoch 47/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4641828.0000 - val_loss: 3784103.2500
Epoch 48/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4328012.0000 - val_loss: 4598613.5000
Epoch 49/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5723938.0000 - val_loss: 3994444.5000
Epoch 50/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4838213.0000 - val_loss: 3624068.5000
Epoch 51/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4825350.0000 - val_loss: 4348800.5000
Epoch 52/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4698818.0000 - val_loss: 3450180.5000
Epoch 53/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4379193.5000 - val_loss: 3654018.2500
Epoch 54/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4137558.5000 - val_loss: 5760608.0000
Epoch 55/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4602630.5000 - val_loss: 4454713.5000
Epoch 56/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 6052817.0000 - val_loss: 3695358.0000
Epoch 57/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 4757751.0000 - val_loss: 3275918.5000
Epoch 58/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5088926.0000 - val_loss: 4282373.0000
Epoch 59/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 4386289.5000 - val_loss: 3548566.5000
Epoch 60/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 4250786.5000 - val_loss: 3823850.5000
Epoch 61/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5097887.5000 - val_loss: 4170035.2500
Epoch 62/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4799607.5000 - val_loss: 3889879.0000
Epoch 63/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4489378.0000 - val_loss: 3273250.0000
Epoch 64/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4074651.2500 - val_loss: 3793309.5000
Epoch 65/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4620872.5000 - val_loss: 4399362.0000
Epoch 66/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 3732072.0000 - val_loss: 3533295.2500
Epoch 67/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5137132.0000 - val_loss: 3505332.7500
Epoch 68/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4085934.7500 - val_loss: 3267521.5000
Epoch 69/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4287116.5000 - val_loss: 4376866.5000
Epoch 70/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4685362.0000 - val_loss: 4198842.0000
Epoch 71/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 3957146.0000 - val_loss: 3801147.5000
Epoch 72/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4313282.0000 - val_loss: 3810263.7500
Epoch 73/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4335373.5000 - val_loss: 3796889.0000
Epoch 74/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 3830995.5000 - val_loss: 3669993.5000
Epoch 75/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4072954.2500 - val_loss: 4276164.0000
Epoch 76/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4741374.5000 - val_loss: 3283898.2500
Epoch 77/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4089191.7500 - val_loss: 3173820.7500
Epoch 78/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4033848.0000 - val_loss: 3979772.2500
Epoch 79/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4157859.2500 - val_loss: 4386113.0000
Epoch 80/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4425901.0000 - val_loss: 3196920.7500
Epoch 81/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 3600253.7500 - val_loss: 7717712.5000
Epoch 82/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5927382.0000 - val_loss: 3309642.5000
Epoch 83/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4580764.0000 - val_loss: 3505535.7500
```

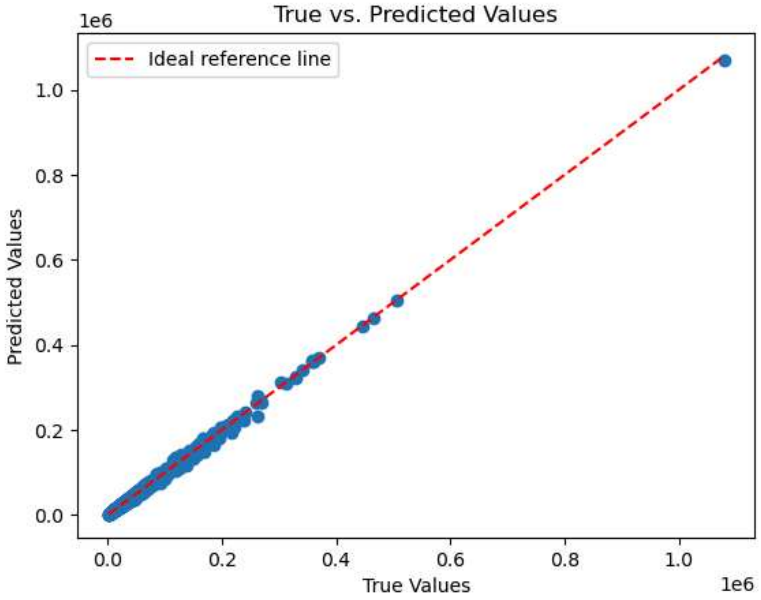
```
Epoch 84/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4558149.0000 - val_loss: 3138563.7500
Epoch 85/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4535522.0000 - val_loss: 3228584.2500
Epoch 86/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4749522.0000 - val_loss: 3276563.2500
Epoch 87/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 3588673.5000 - val_loss: 4399883.0000
Epoch 88/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4731359.5000 - val_loss: 3557430.7500
Epoch 89/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4161584.0000 - val_loss: 3965615.0000
Epoch 90/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4882314.0000 - val_loss: 3413479.2500
Epoch 91/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4490862.5000 - val_loss: 4653604.0000
Epoch 92/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4422483.0000 - val_loss: 3375016.0000
Epoch 93/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 5323550.5000 - val_loss: 4144141.5000
Epoch 94/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4141722.2500 - val_loss: 3284763.7500
Epoch 95/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4366565.0000 - val_loss: 3525432.0000
Epoch 96/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4090941.0000 - val_loss: 3772088.2500
Epoch 97/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 3908858.2500 - val_loss: 4423566.0000
Epoch 98/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 4115745.0000 - val_loss: 3687363.5000
Epoch 99/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 4434126.5000 - val_loss: 5093612.0000
Epoch 100/100
400/400 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 4007515.2500 - val_loss: 3412786.0000
```

Out[41]: <keras.src.callbacks.history.History at 0x26620ec3560>

In [42]:
```python
# Predict on test set
nn_y_pred = model.predict(X_test)

# Compute metrics
nn_mse = mean_squared_error(y_test, nn_y_pred)
nn_mae = mean_absolute_error(y_test, nn_y_pred)
nn_r2 = r2_score(y_test, nn_y_pred)

print(f'Neural Network Mean Squared Error: {nn_mse}')
print(f'Neural Network Mean Absolute Error: {nn_mae}')
print(f'Neural Network R-squared: {nn_r2}')
```

```
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Neural Network Mean Squared Error: 3833206.910974666
Neural Network Mean Absolute Error: 790.0579518563894
Neural Network R-squared: 0.9978806291165092
```

In [43]:
```python
# Scatter plot of actual vs predicted values for Neural Network
plt.scatter(y_test, nn_y_pred)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--', label= 'Ideal reference line')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.title('True vs. Predicted Values')
plt.legend()
plt.show()
```



Key points:

If the model is perfect, all the points will lie on the ideal refrence diagonal line which is plotted as a dashed red line.

Points above the line: Model over-predicts.

Points below the line: Model under-predicts.

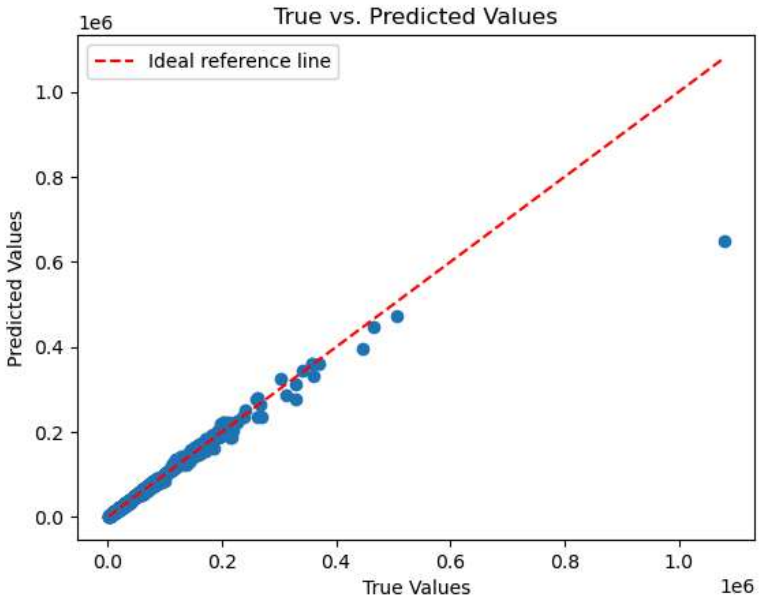
The closer the points are to the diagonal, the better the model's predictions match the true values.

Random forest regressor

In [46]:
```
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_y_pred = rf_model.predict(X_test)

rf_mse = mean_squared_error(y_test, rf_y_pred)
rf_mae = mean_absolute_error(y_test, rf_y_pred)
rf_r2 = r2_score(y_test, rf_y_pred)

print(f"Random Forest MSE: {rf_mse}")
print(f"Random Forest MAE: {rf_mae}")
print(f"Random Forest R²: {rf_r2}")
```

```
Random Forest MSE: 53652104.33061681
Random Forest MAE: 1088.7114040279664
Random Forest R²: 0.9703358805310596
```

In [47]:
```
# Scatter plot of actual vs predicted values for Random Forest
plt.scatter(y_test, rf_y_pred)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--', label= 'Ideal reference line')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.title('True vs. Predicted Values')
plt.legend()
plt.show()
```

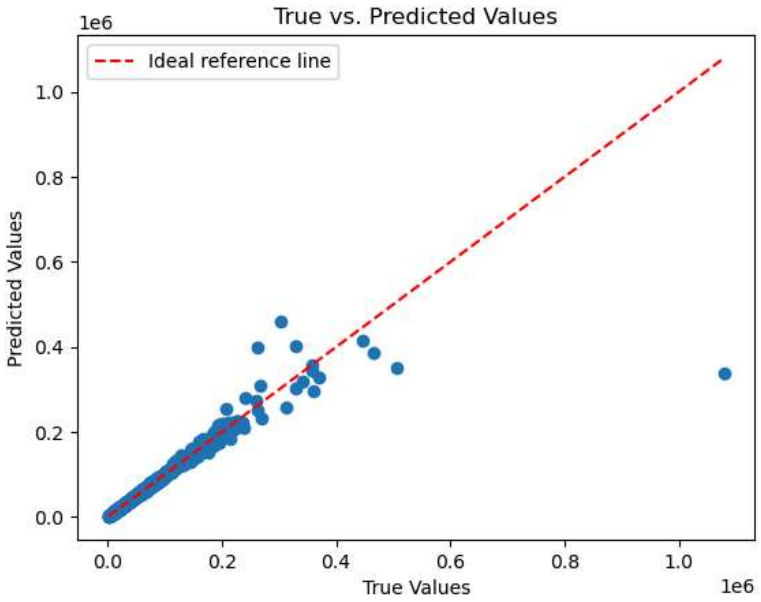


XGBoost model

In [49]:
```
# Create XGBoost model
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)

# Train the model
xgb_model.fit(X_train, y_train)

# Predict on the test set
xgb_y_pred = xgb_model.predict(X_test)

# Compute metrics for XGBoost
xgb_mse = mean_squared_error(y_test, xgb_y_pred)
xgb_mae = mean_absolute_error(y_test, xgb_y_pred)
xgb_r2 = r2_score(y_test, xgb_y_pred)

print(f"XGBoost MSE: {xgb_mse}")
print(f"XGBoost MAE: {xgb_mae}")
print(f"XGBoost R²: {xgb_r2}")
```

```
XGBoost MSE: 165775310.85695145
XGBoost MAE: 1485.5133702652413
XGBoost R²: 0.9083432292616506
```

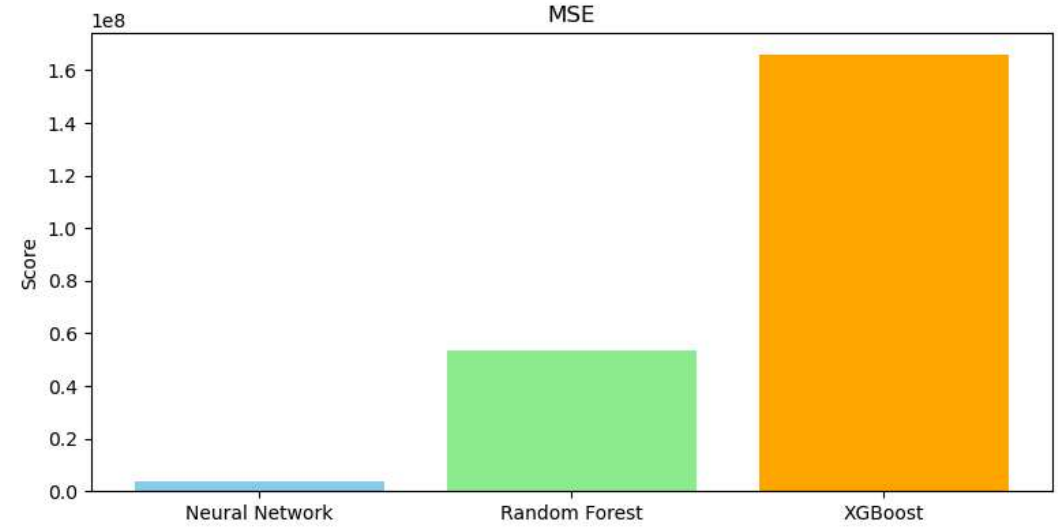
In [50]:
```
# Scatter plot of actual vs predicted values for XGBoost
plt.scatter(y_test, xgb_y_pred)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--', label= 'Ideal reference line')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.title('True vs. Predicted Values')
plt.legend()
plt.show()
```

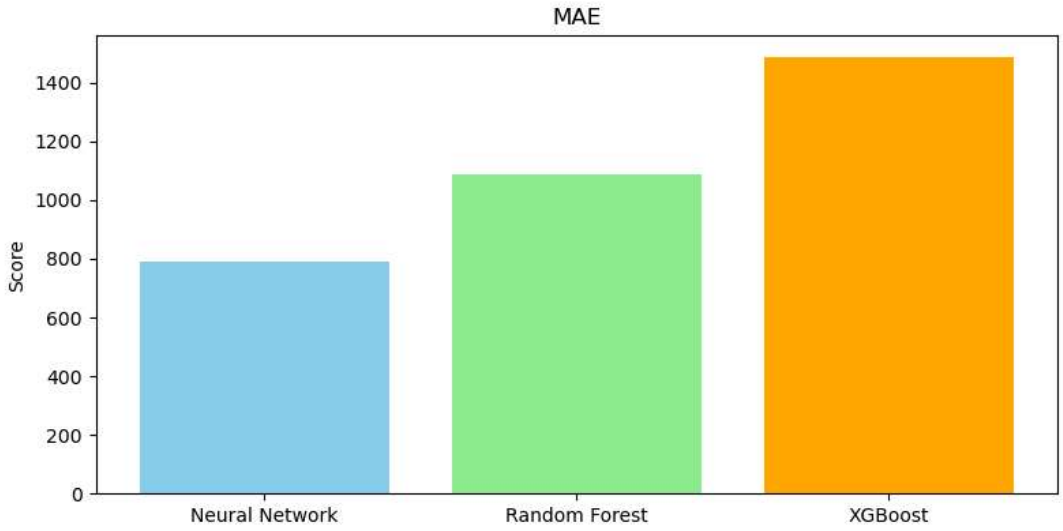


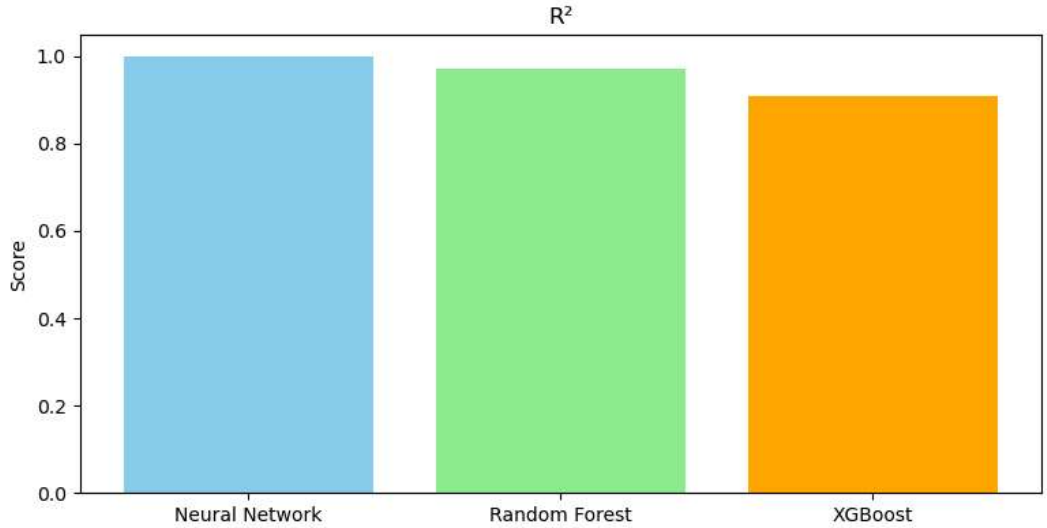
Performance comparision of the models

In [52]:
```
# Print the comparison of the models
print("Model Comparison:")
print(f"Neural Network MSE: {nn_mse}")
print(f"Random Forest MSE: {rf_mse}")
print(f"XGBoost MSE: {xgb_mse}")
print()
print(f"Neural Network MAE: {nn_mae}")
print(f"Random Forest MAE: {rf_mae}")
print(f"XGBoost MAE: {xgb_mae}")
print()
print(f"Neural Network R²: {nn_r2}")
print(f"Random Forest R²: {rf_r2}")
print(f"XGBoost R²: {xgb_r2}")

# Plot the results
metrics = ['MSE', 'MAE', 'R²']
nn_scores = [nn_mse, nn_mae, nn_r2]
rf_scores = [rf_mse, rf_mae, rf_r2]
xgb_scores = [xgb_mse, xgb_mae, xgb_r2]

models = ['Neural Network', 'Random Forest', 'XGBoost']
scores = [nn_scores, rf_scores, xgb_scores]

fig, axs = plt.subplots(3,1, figsize=(8, 12))

for i, metric in enumerate(metrics):
    axs[i].bar(models, [nn_scores[i], rf_scores[i], xgb_scores[i]], color=['skyblue', 'lightgreen', 'orange'])
    axs[i].set_title(metric)
    axs[i].set_ylabel('Score')

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

```
Model Comparison:
Neural Network MSE: 3833206.910974666
Random Forest MSE: 53652104.33061681
XGBoost MSE: 165775310.85695145

Neural Network MAE: 790.0579518563894
Random Forest MAE: 1088.7114040279664
XGBoost MAE: 1485.5133702652413

Neural Network R²: 0.9978806291165092
Random Forest R²: 0.9703358805310596
XGBoost R²: 0.9083432292616506
```







Observation:

Based on these results, the Neural Network model is the most accurate and reliable for this particular regression task, achieving substantially lower prediction errors (low MSE and MAE)0 and explaining nearly all variance (high R2 score) in the target variable compared to Random Forest and XGBoost.