In [9]:
```
#importing necessary libraries
import os
import gzip
import shutil
import json

import pandas as pd
from pandas import json_normalize

#plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns


#Supporting Function to extract the files and do some cleanup
def parse_json(filename: str):
    "Function to parse json files and return a pandas dataframe"

    with open(filename) as f:
        lines = f.read()
    if 'users' in filename:
        # remove the first and last line of the file of the "users.json" fil
        lines = lines.splitlines()[1:-1]

    else:
        lines = lines.splitlines()

    df_tmp = pd.DataFrame(lines)
    df_tmp.columns = ['json_data']
    df_tmp['json_data'].apply(json.loads)
    ret_json = pd.json_normalize(df_tmp['json_data'].apply(json.loads))

    return ret_json
```

Step 1 : Generates 4 data tables based on our data model

1. brand_table
2. user_table
3. receipt_table
4. receipt_item_table (extracted from "*rewardsReceiptItemList*" column)

Originally we had 3 datasets. In order to build a normalized datamodel, I have generated a new table from Receipts data called Receiptreceipt_item_table table which consists of receipt_id, and all the normalized fields from rewardsReceiptitemList field. This can be used to join between receipt and brand table enabling us to maintain a relational structure in our data model.

In [10]:
```
brand_table= parse_json('data/brands.json')
receipt_table= parse_json('data/receipts.json')
user_table= parse_json('data/users.json')
```

```
receipt_item_table = receipt_table[['_id.$oid', 'rewardsReceiptItemList']]
receipt_item_table = receipt_item_table.rename(columns={'_id.$oid': 'receipt
receipt_item_table = receipt_item_table.explode('rewardsReceiptItemList')

expanded_receipts = json_normalize(receipt_item_table['rewardsReceiptItemLis
receipt_item_table = pd.concat([ receipt_item_table.reset_index()['receipt_i
```

Step 3 : Null Value Check

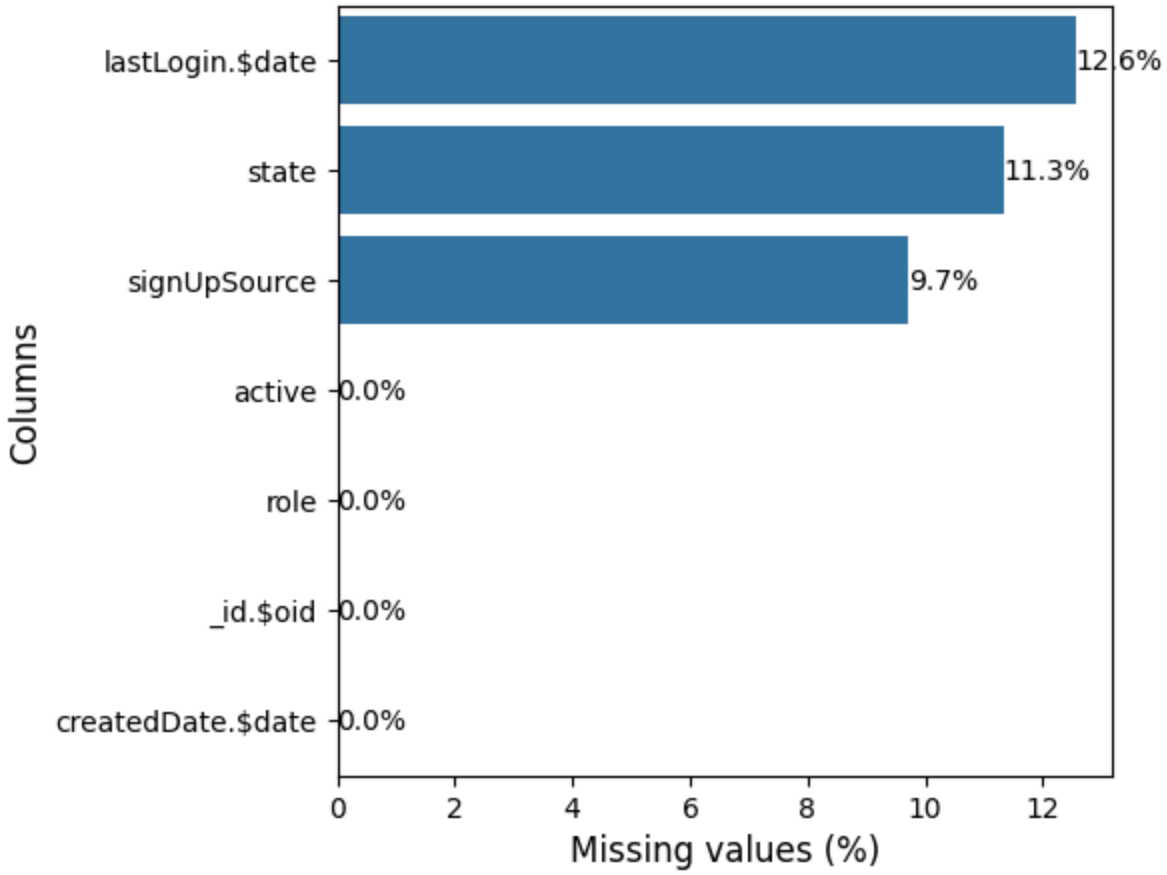
In this step, we will check for null values in the datasets. The findings from this step are summarized as follows:

1. The user table has the least amount of missing data, with *lastlogin* column having only 12% of null values, which is within acceptable standards.
2. The brand-table has significant missing columns, with *categorycode* and *topbrand* columns having more than 50% null values.
3. Receipts table has 9 data fields with more than 40% missing records. Some columns could be of concern such as *purchasedate*/*finisheddate* or bonus points earned which would be significant in user behavior analytics.
4. *Item* table is highly sparse, with 77% of the fields have more than 30% missing values.

In [11]:
```
def null_value_check(input_df: pd.DataFrame,name:str=None,fig_width=5,fig_he
    ''' This function returns a bar plot for null values, ranked by percenta
    '''

    missing_data = input_df.isnull().sum()
    missing_data = missing_data[missing_data >= 0]
    missing_data = (missing_data / len(input_df)) * 100
    print(f'Missing values by percentage for {name} table:')
    missing_df =  missing_data.sort_values(ascending=False)
    if not orient:
        plt.figure(figsize=(fig_width, fig_height))
        fig =  sns.barplot(x=missing_df.index, y=missing_df)
        #horizontal bar plot in sns
        fig.set_xticklabels(fig.get_xticklabels(), rotation=70)

        fig.set_ylabel('Missing values (%)', fontsize=12)
        fig.set_xlabel('Columns', fontsize=12)

    if orient == 'h':
        plt.figure(figsize=(fig_width,fig_height))
        fig =  sns.barplot(y=missing_df.index, x=missing_df)

        for index, value in enumerate(missing_df):
            plt.text(value , index, f"{value:.1f}%", va='center', fontsize=1
        #horizontal bar plot in sns
        fig.set_ylabel('Columns', fontsize=12)
        fig.set_xlabel('Missing values (%)', fontsize=12)


    return
```

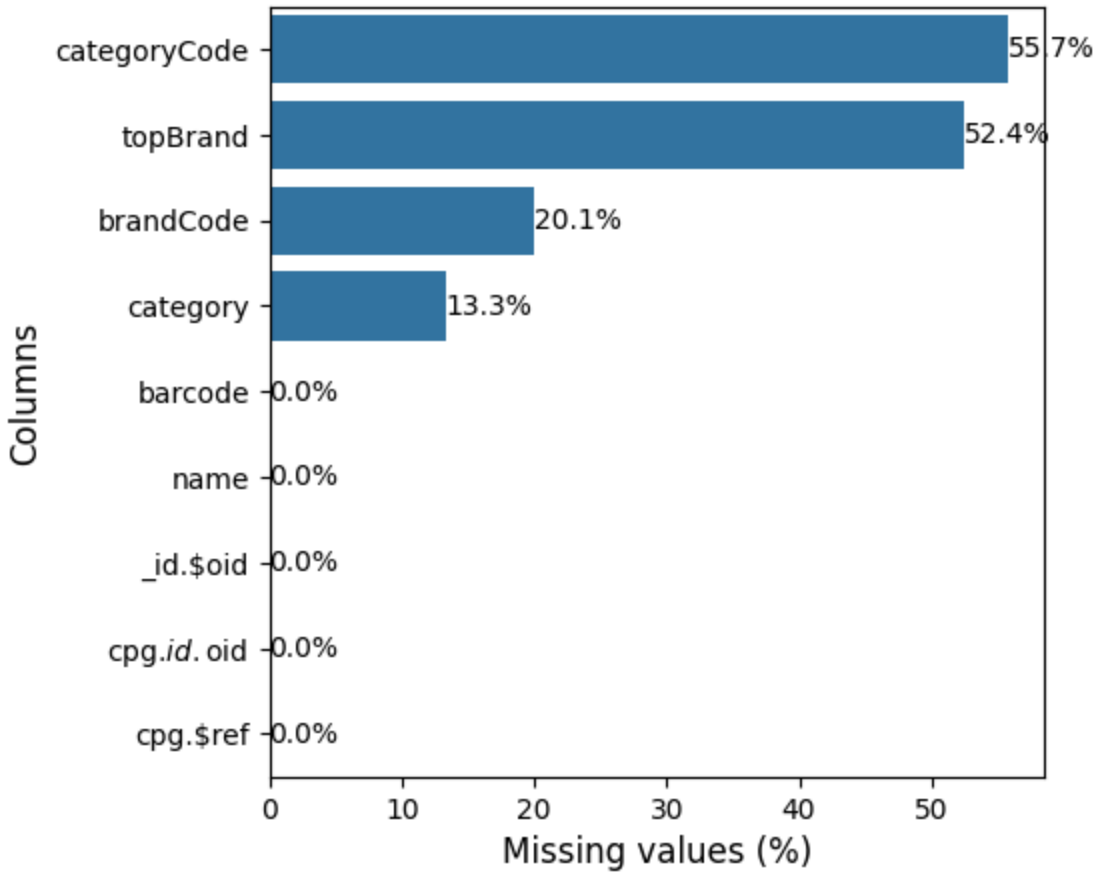
```
null_value_check(user_table, "Users",orient='h')
```

```
Missing values by percentage for Users table:
```



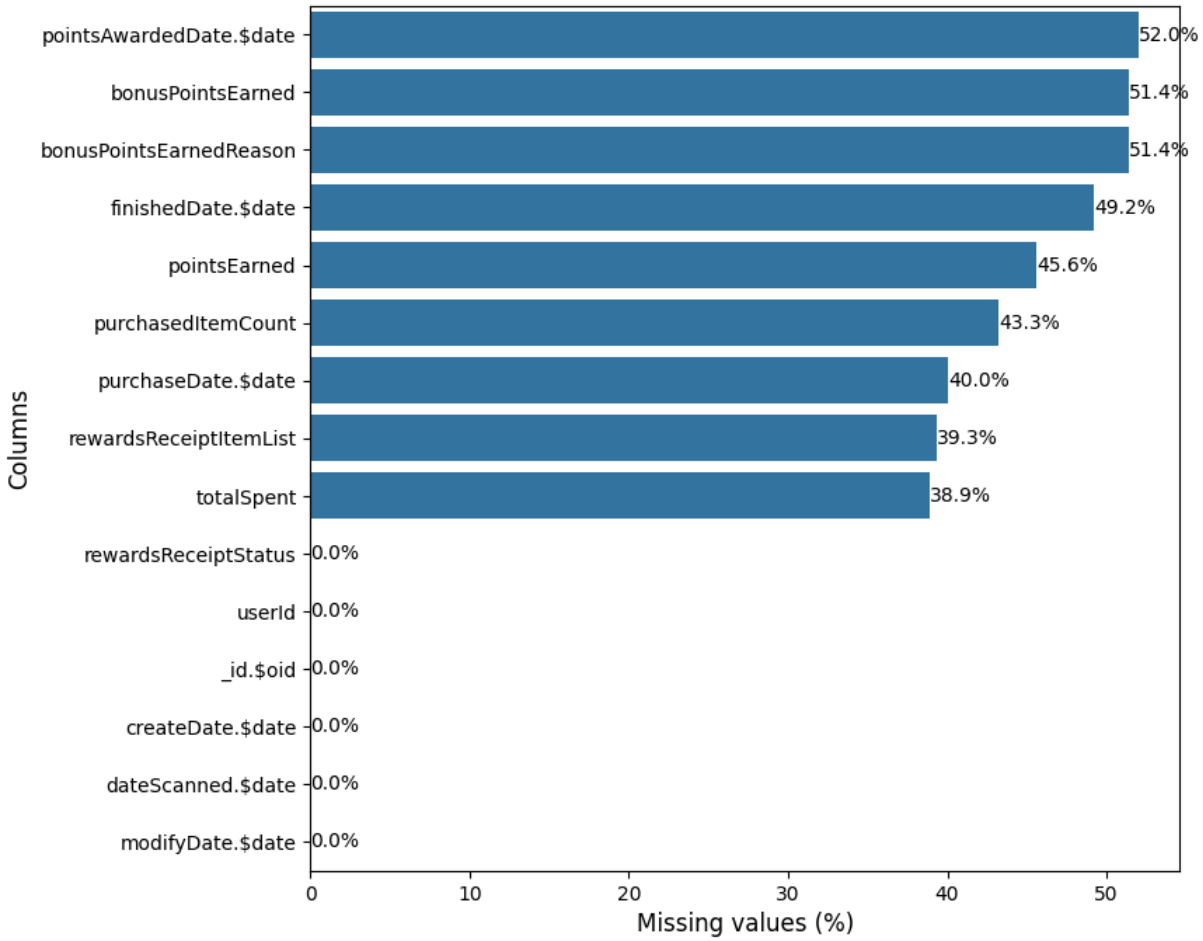
In [12]:
```
null_value_check(brand_table, "brands",orient='h')
```

```
Missing values by percentage for brands table:
```



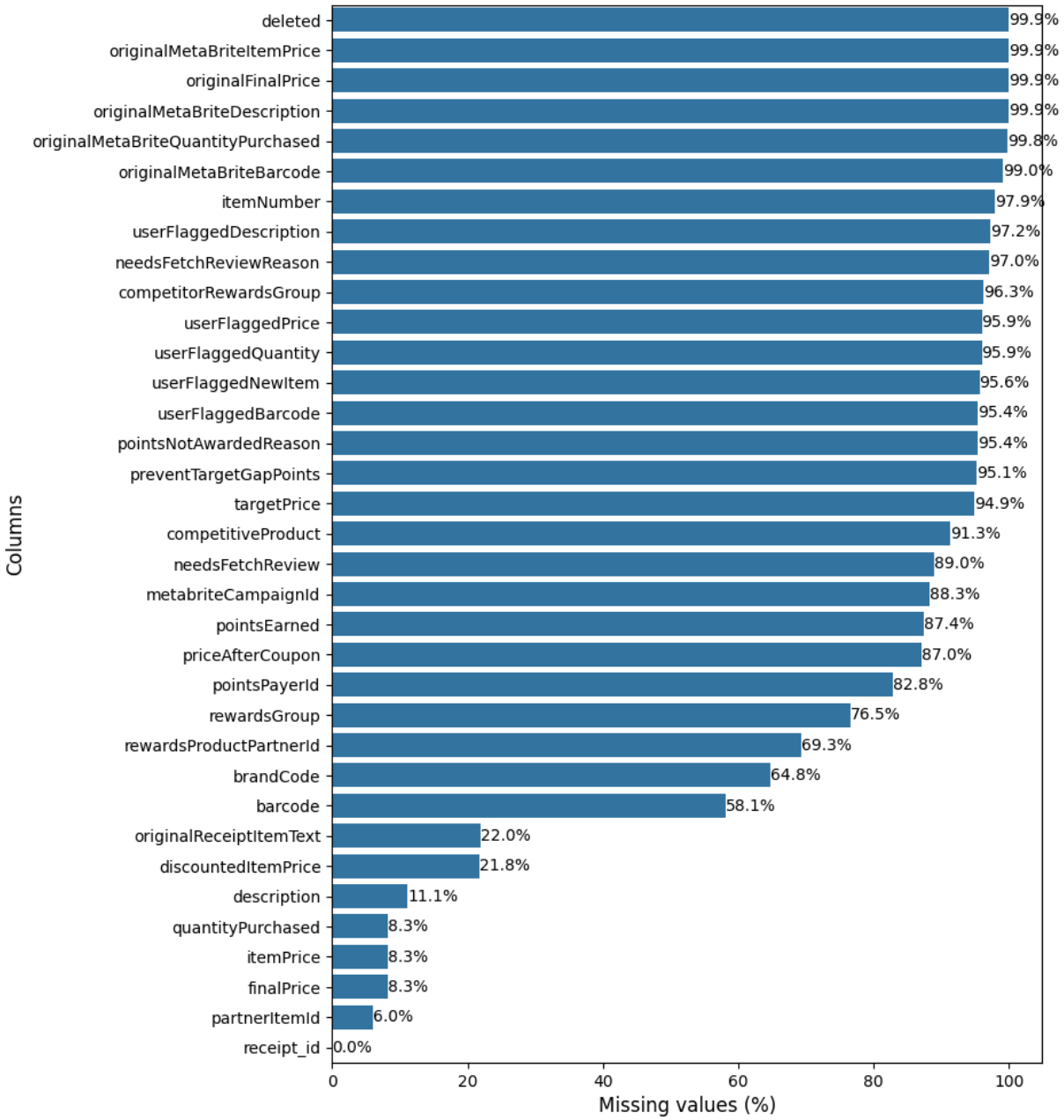
In [13]:
```
null_value_check(receipt_table, "receipts", fig_width=8,fig_height=8, orient
```

```
Missing values by percentage for receipts table:
```



In [14]:
```
null_value_check(receipt_item_table, "items",fig_width=8,fig_height=12,orier
```

```
Missing values by percentage for items table:
```



Step 3 : Data Duplication Check

In this step, we will primarily check for duplicate records across all data sources. The findings from this step are summarized as follows:

1. User Table has 282 duplicate rows corresponding to 70 users having the same *id*, *active status*, and *creation date*. This could possibly point to a situation where the data logic for updating the "active" status or updating the user-id has failed to operate properly.
2. For the brand table, there is data duplicaton based on *category*, *categoryCode*, *name, and cpg-id* fields, resulting in data redundancy. So we need to logically evaluate the requirements for these data fields to remove redundant information or possibly combine multiple fields to get accurate data fields.

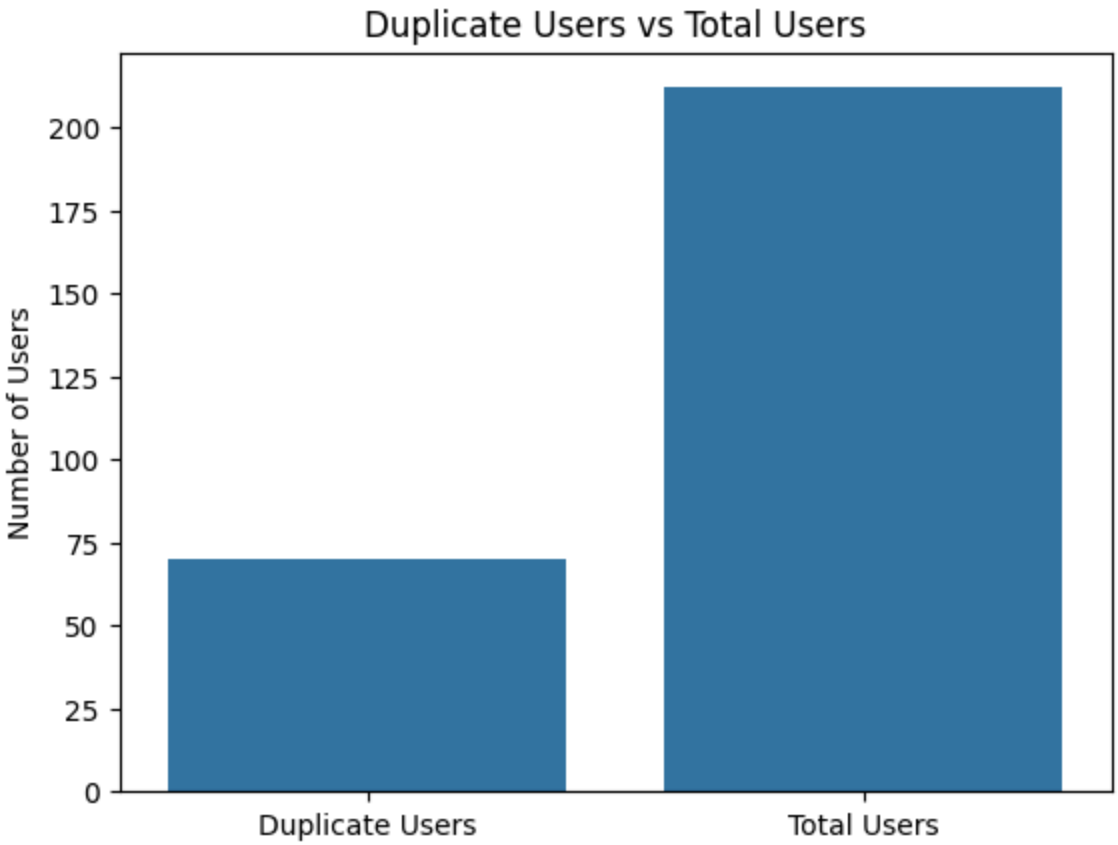
Step 3.1 : Duplicate user entries

In [15]:
```
print(f'Total Duplicate Rows in Users Data : {user_table.duplicated().sum()}
```

```
Total Duplicate Rows in Users Data : 282
```

In [16]:
```
# sns.barplot(user_table[user_table.duplicated()]['_id.$oid'].nunique(), use
sns.barplot(x=['Duplicate Users', 'Total Users'], y=[user_table[user_table.d
plt.ylabel('Number of Users')
plt.title('Duplicate Users vs Total Users')
```

Out[16]:
```
Text(0.5, 1.0, 'Duplicate Users vs Total Users')
```



Step 3.2 : Duplicate Brand

In [17]:
```
brand_table[brand_table.duplicated(subset=['category','categoryCode','name',
```

Out[17]:

	barcode	category	categoryCode	name	topBrand	
126	511111104698	Baby	NaN	Pull-Ups	False	5bd201a990fa0745
978	511111312949	Baby	NaN	Pull-Ups	True	5db3288aee7f2d6de
64	511111805854	Health & Wellness	NaN	ONE A DAY® WOMENS	False	5da609991dda2c3e
339	511111914051	Health & Wellness	NaN	ONE A DAY® WOMENS	NaN	5e5ff265ee7f2d0b
574	511111605546	Snacks	NaN	Baken-Ets	NaN	5d9d08d1a60b87376
848	511111701781	Snacks	NaN	Baken-Ets	True	585a961fe4b03e62

Table : Duplicate entries with same category, categoryCode, name and brand name

Step 4 : Data Consistency

This section is primarily aimed at finding out data discrepancies that needs to be corrected for accurate analytics, such as incomaptible fields, invalid product numbers etc. The findings from this step are summarized as follows:

1. For *receipt_item* table, there are 6 different barcodes that have '**ITEM NOT FOUND**' descriptions. The criteria for these descriptions and the expectations for corresponding reports regarding these barcodes needs to be properly specified.
2. There are Receipts with *rewardsReceiptStatus* marked as "**REJECTED**" but having *pointsEarned* greater than 0. This could point towards an issue in application logic while populating the *pointsEarned* field. If this is by design, it should be properly documented such that BI applications integrate the logic to handle these cases for accurate reporting.
3. Receipt Table has 117 unique users, not present in User Table, which accounts for nearly **30% of transcation amount** by *totalSpent*. This needs to resolved as it negatively affects user segmentation analytics.

Step 4.1 : "ITEM NOT FOUND" Descriptions

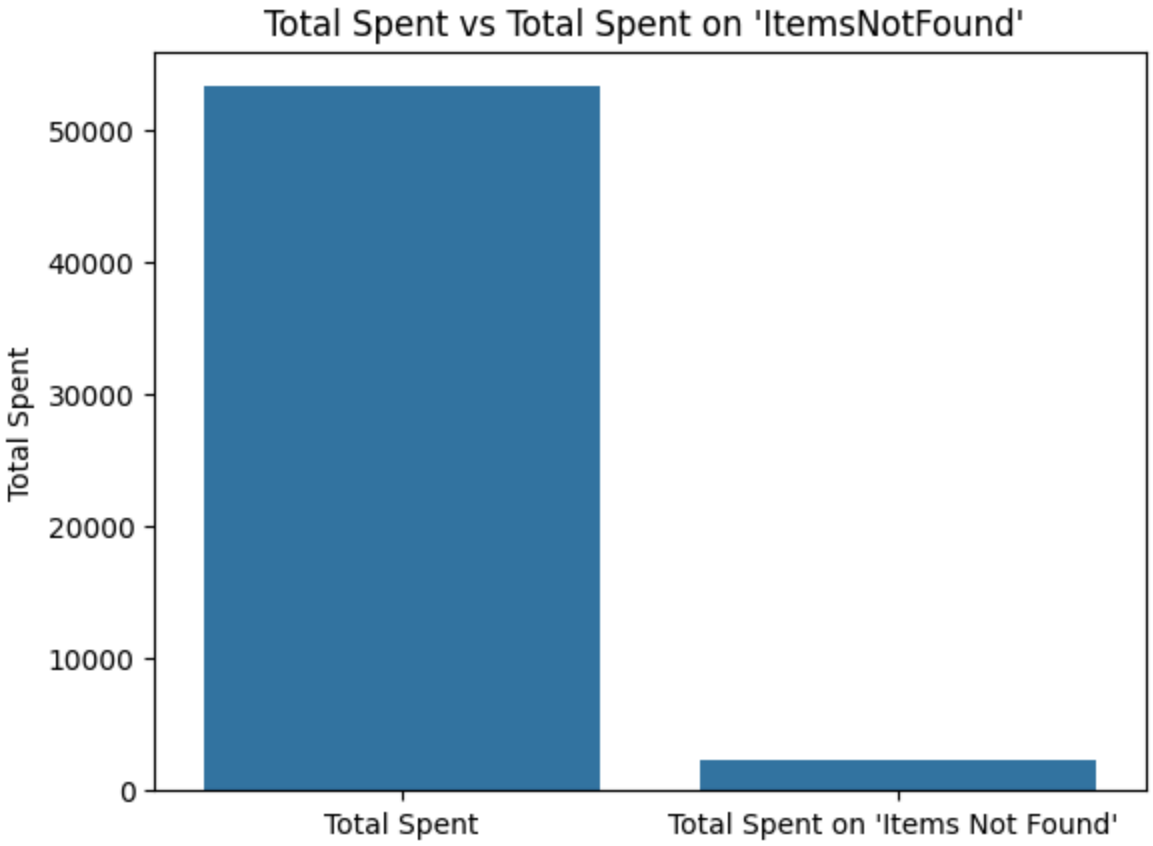
This will identify receipt entries with "ITEM NOT FOUND" in the description, which might indicate issues with barcode scanning or product data lookups. There are 6 different barcode's that don't have accurate descriptions. This needs to be highlighted.

In [18]:
```
print(receipt_item_table[receipt_item_table.description == 'ITEM NOT FOUND']
```

```
['4011' '686924155783' '22' '686924291290' '792851356565' '5000111047524']
```

In [19]:
```
sns.barplot(x=['Total Spent', "Total Spent on 'Items Not Found' "], y=[recei
plt.ylabel('Total Spent')
plt.title("Total Spent vs Total Spent on 'ItemsNotFound' ")
```

Out[19]:
```
Text(0.5, 1.0, "Total Spent vs Total Spent on 'ItemsNotFound' ")
```



Step 4.2 : "REJECTED" Receipts with pointsEarned > 0

Receipts with rewardsReceiptStatus marked as "REJECTED" but having pointsEarned greater than 0. Clarification is needed on how these values should be handled while analyzing rewards data.

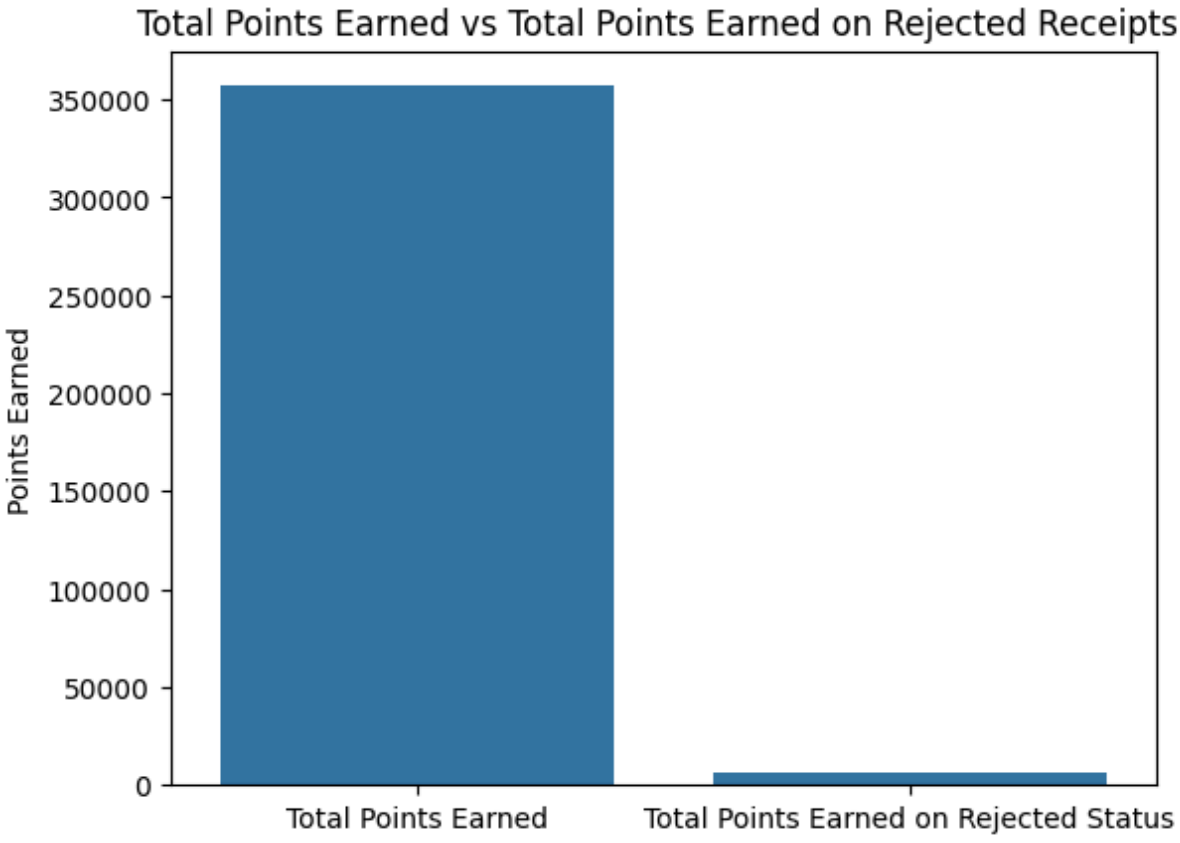
In [20]:
```
receipt_table.pointsEarned = receipt_table.pointsEarned.astype(float)
receipt_table[(receipt_table.rewardsReceiptStatus == 'REJECTED') & (receipt_
```

Out[20]:

	bonusPointsEarned	bonusPointsEarnedReason	pointsEarned	purchasedItemCour
2	5.0	All-receipts receipt bonus	5.0	1.
13	750.0	Receipt number 1 completed, bonus point schedu...	750.0	11.
62	750.0	Receipt number 1 completed, bonus point schedu...	750.0	2.
203	5.0	All-receipts receipt bonus	5.0	1.
207	5.0	All-receipts receipt bonus	5.0	1.

In [21]:
```
sns.barplot(x=['Total Points Earned', "Total Points Earned on Rejected Statu
plt.ylabel('Points Earned')
plt.title("Total Points Earned vs Total Points Earned on Rejected Receipts "
```

Out[21]:
```
Text(0.5, 1.0, 'Total Points Earned vs Total Points Earned on Rejected Rece
ipts ')
```

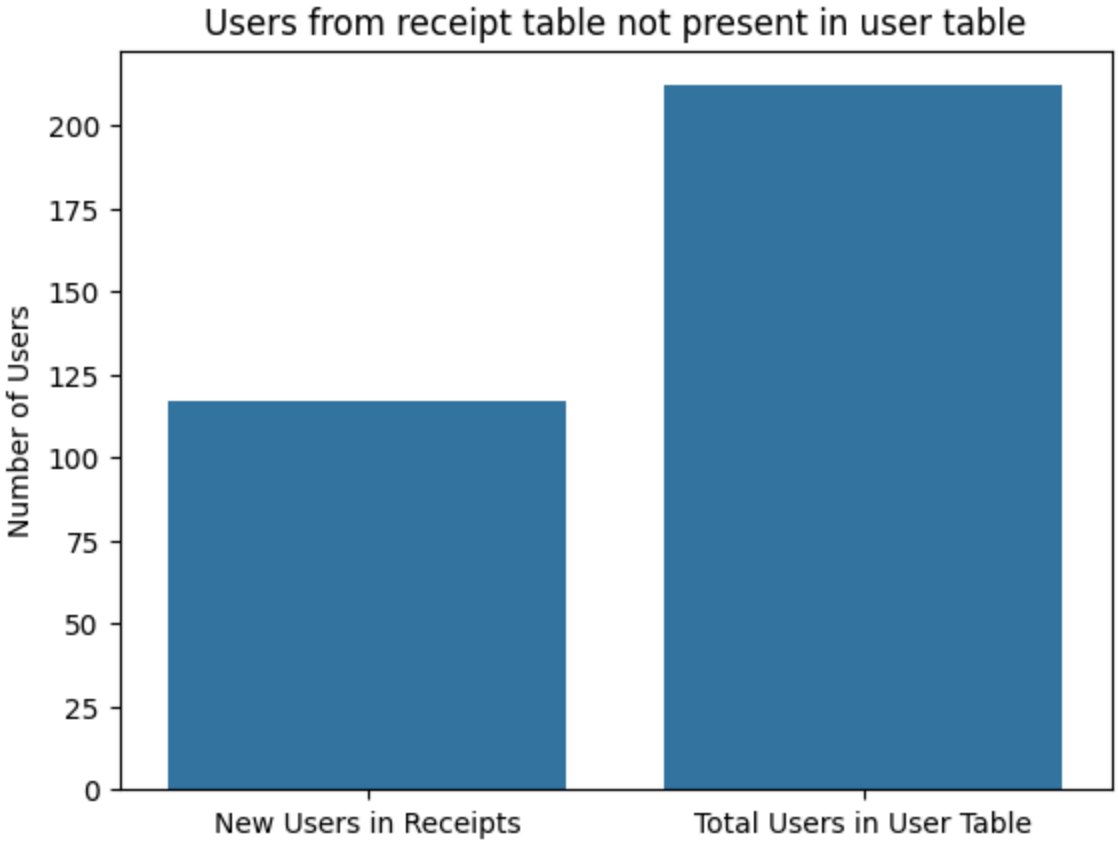


Step 4.3 : User_id's from receipt data that are not present in User Table

In [22]:
```
#list of user id from receipt table which are not present in user table
left_out_users = len(receipt_table[~receipt_table.userId.isin(user_table['_i
```

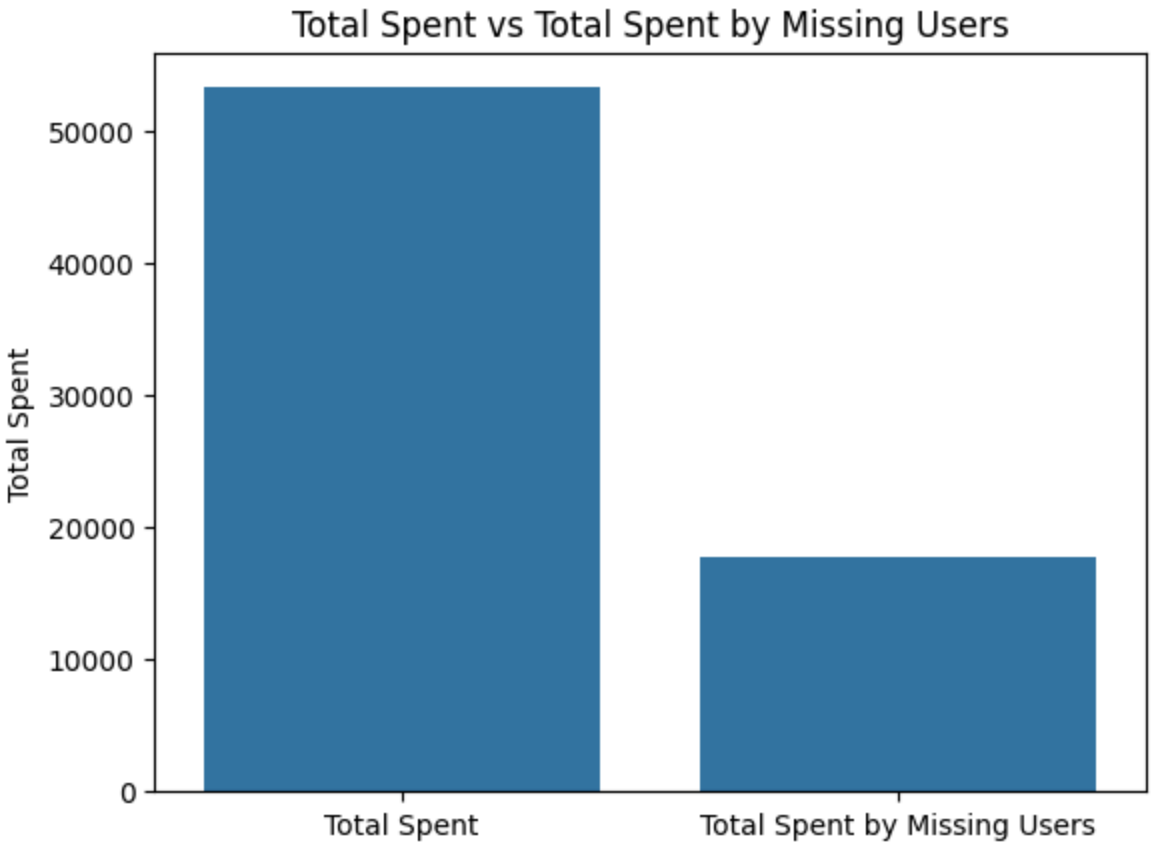
```
sns.barplot(x=['New Users in Receipts', 'Total Users in User Table'], y=[lef
plt.ylabel('Number of Users')
plt.title('Users from receipt table not present in user table')
```

Out[22]: Text(0.5, 1.0, 'Users from receipt table not present in user table')



In [31]:
```
sns.barplot(x=['Total Spent', 'Total Spent by Missing Users'], y=[receipt_ta
plt.ylabel('Total Spent')
plt.title('Total Spent vs Total Spent by Missing Users')
```

Out[31]: Text(0.5, 1.0, 'Total Spent vs Total Spent by Missing Users')


Total Spent vs Total Spent by Missing Users
Total Spent
50000
40000
30000
20000
10000
0
Total Spent
Total Spent by Missing Users