In [21]:
```
#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


#prereq step for primary data unizpping
filenames = os.listdir('data/')
for filename in filenames:
    if filename.endswith('.gz'):
        with gzip.open(filename, 'rb') as f_in:
            with open(filename[:-3], 'wb') as f_out:
                shutil.copyfileobj(f_in, f_out)
                print(f'{filename} unzipped')
                os.remove(filename)
                print(f'{filename} removed')
    else:
        print(f'{filename} data already extracted')
```

```
users.json data already extracted
receipts.json data already extracted
brands.json data already extracted
```

Note : In this notebook, I will present a step by step guide to identify some common data quality concerns for given datasets and summarize it at the end of document.

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

1. brand_table
2. user_table
3. receipt_table
4. 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 [2]:
```
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"
        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


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': 'rec
receipt_item_table = receipt_item_table.explode('rewardsReceiptItemList'

expanded_receipts = json_normalize(receipt_item_table['rewardsReceiptIte
receipt_item_table = pd.concat([ receipt_item_table.reset_index()['recei
```

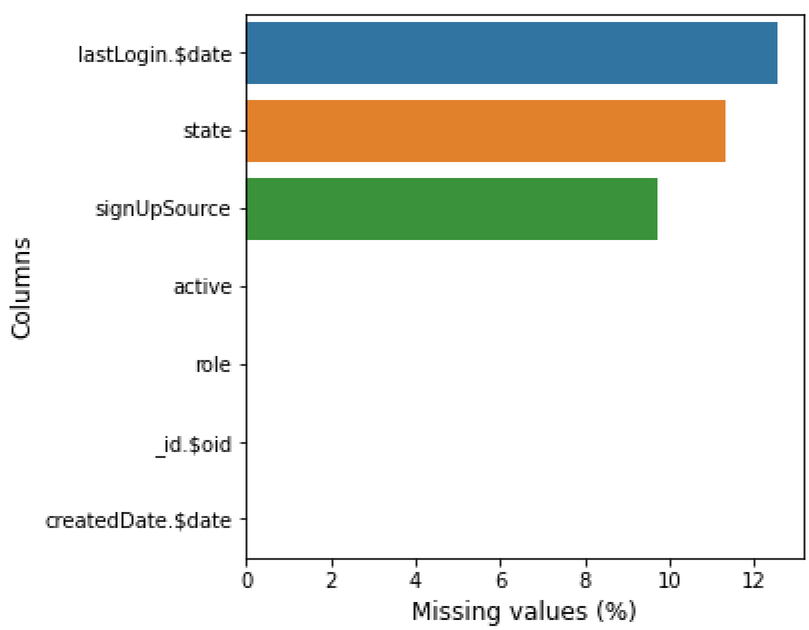
Step 2 : Null Value Check

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.

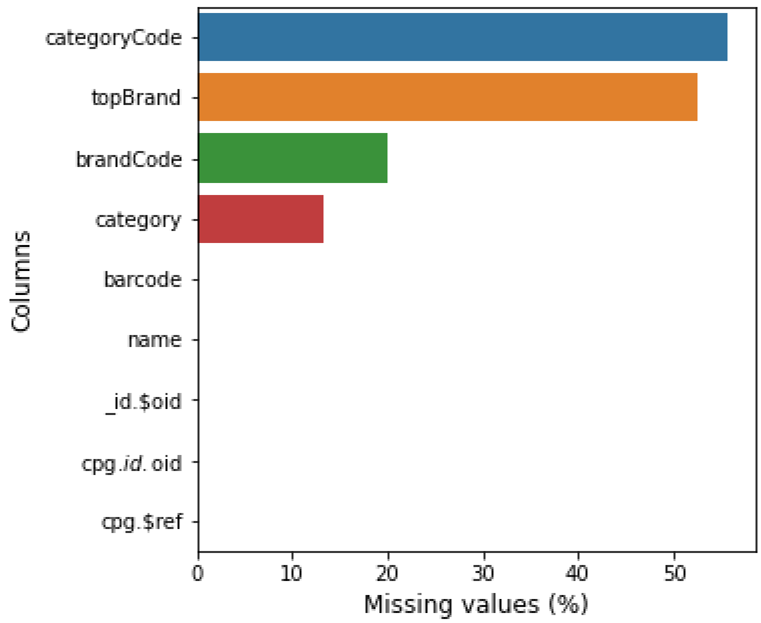
In [3]:
```
def null_value_check(input_df: pd.DataFrame,name:str=None,fig_width=5,fi
    ''' This function returns a bar plot for null values, ranked by perc
    '''

    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}:')
    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)
        #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:
```



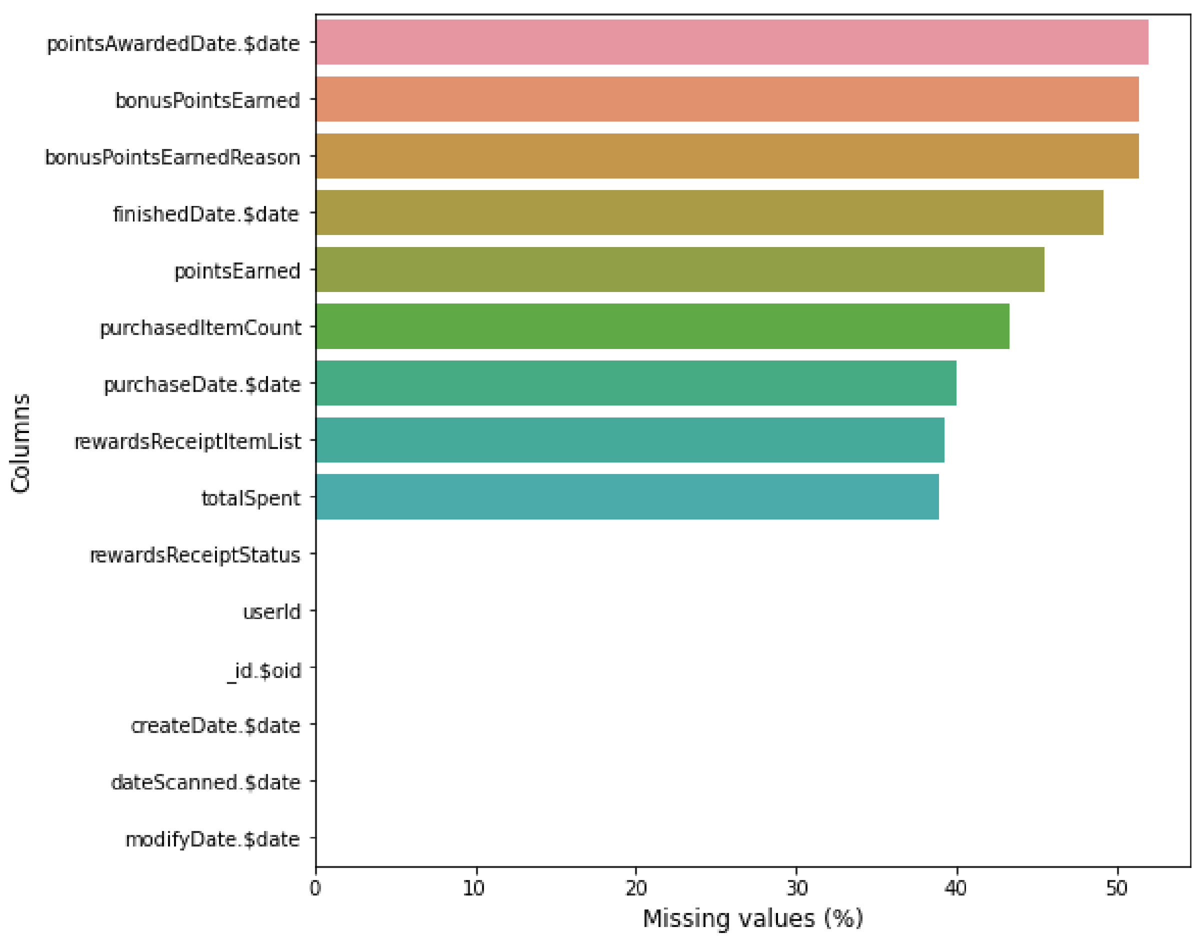
In [4]:
```
null_value_check(brand_table, "brands",orient='h')
```

Missing values by percentage for brands:



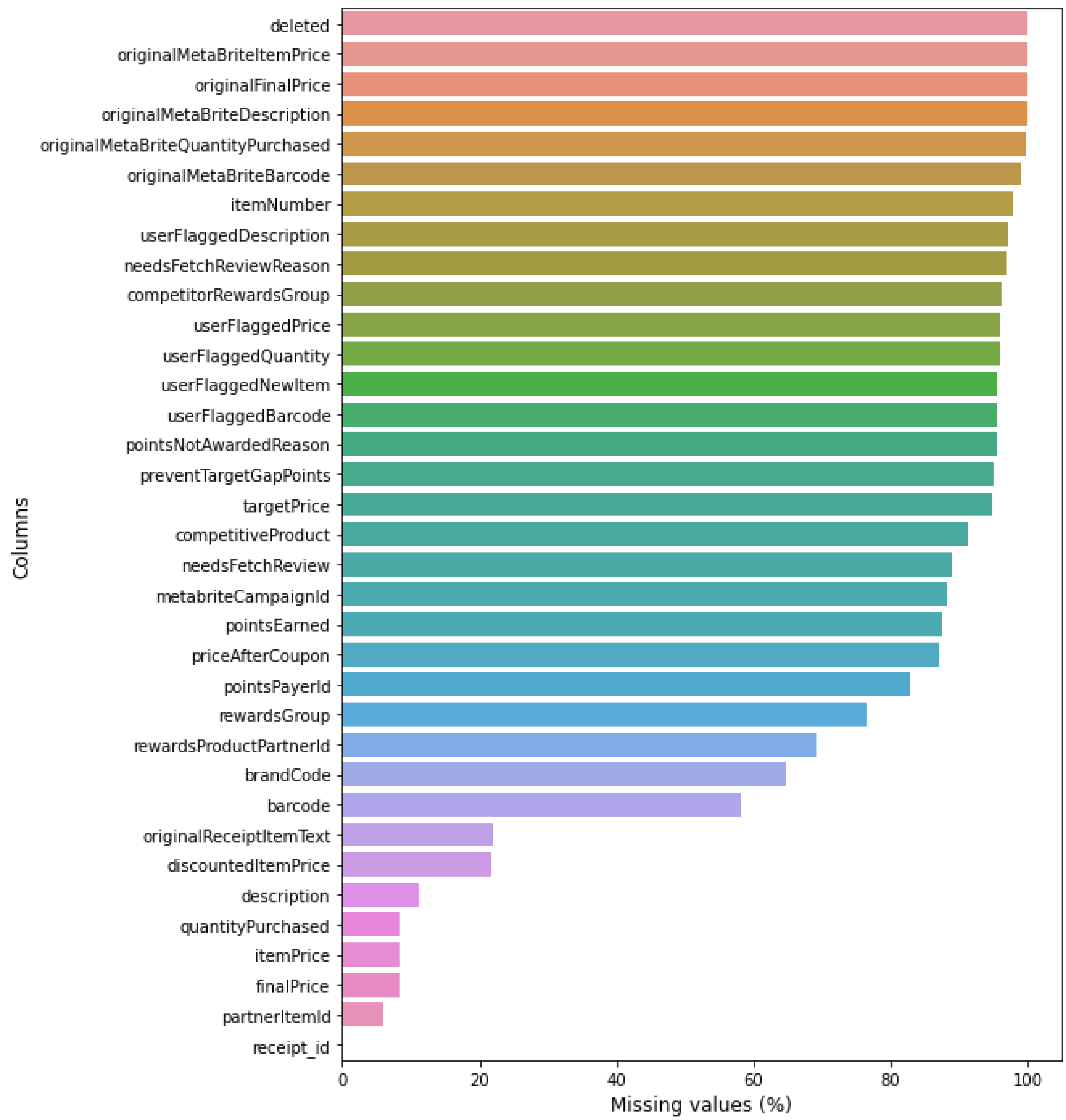
In [5]:
```
null_value_check(receipt_table, "receipts", fig_width=8,fig_height=8, or
```

Missing values by percentage for receipts:



In [6]:
```
null_value_check(receipt_item_table, "items",fig_width=8,fig_height=12,o
```

Missing values by percentage for items:



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

Step 3.1 : Duplicate user entries

In [7]:
```
print(f'Total Duplicate Rows in Users Data : {user_table.duplicated().sur
user_table[user_table.duplicated(subset=['active','role','_id.$oid','cre
```

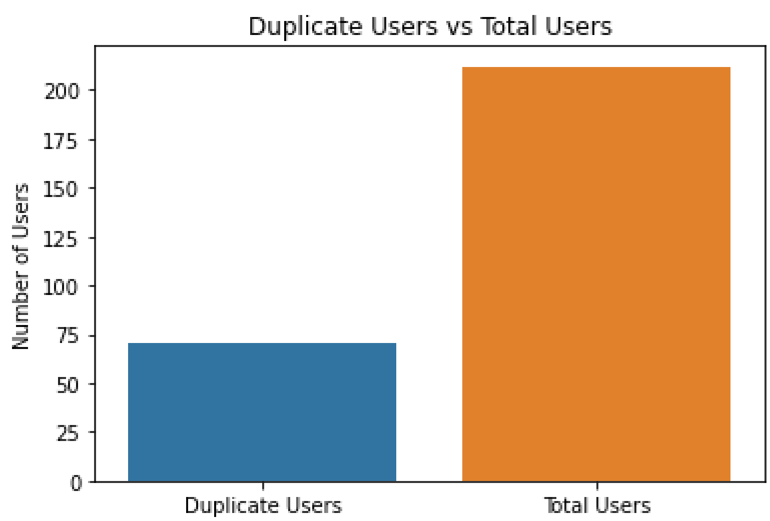
```
Total Duplicate Rows in Users Data : 282
```

Out[7]:

	active	role	signUpSource	state	_id.$oid	createdDate.$date	lastLogin.$
1	True	consumer	Email	WI	5ff1e194b6a9d73a3a9f1052	1609687444800	1.609688
3	True	consumer	Email	WI	5ff1e194b6a9d73a3a9f1052	1609687444800	1.609688
4	True	consumer	Email	WI	5ff1e194b6a9d73a3a9f1052	1609687444800	1.609688
7	True	consumer	Email	WI	5ff1e194b6a9d73a3a9f1052	1609687444800	1.609688
9	True	consumer	Email	WI	5ff1e194b6a9d73a3a9f1052	1609687444800	1.609688

In [8]:
```
# sns.barplot(user_table[user_table.duplicated()]['_id.$oid'].nunique(),
sns.barplot(x=['Duplicate Users', 'Total Users'], y=[user_table[user_tab
plt.ylabel('Number of Users')
plt.title('Duplicate Users vs Total Users')
```

Out[8]: `Text(0.5, 1.0, 'Duplicate Users vs Total Users')`



Step 3.2 : Duplicate Brand

In [9]:
```
brand_table[brand_table.duplicated(subset=['category','categoryCode','na
```

Out[9]:

	barcode	category	categoryCode	name	topBrand	_id.$oid	
126	511111104698	Baby	NaN	Pull-Ups	False	5bd201a990fa074576779a19	55
978	511111312949	Baby	NaN	Pull-Ups	True	5db3288aee7f2d6de4248977	55
64	511111805854	Health & Wellness	NaN	ONE A DAY® WOMENS	False	5da609991dda2c3e1416ae90	53e
339	511111914051	Health & Wellness	NaN	ONE A DAY® WOMENS	NaN	5e5ff265ee7f2d0b35b2a18f	53e
574	511111605546	Snacks	NaN	Baken-Ets	NaN	5d9d08d1a60b87376833e348	53
848	511111701781	Snacks	NaN	Baken-Ets	True	585a961fe4b03e62d1ce0e76	53

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

Type *Markdown* and LaTeX: α^2

In [10]:
```
brand_table[brand_table.duplicated(subset=['category','categoryCode','na
```

Out[10]:

	barcode	category	categoryCode	name	topBrand	_id.$oid
126	511111104698	Baby	NaN	Pull-Ups	False	5bd201a990fa074576779a19
978	511111312949	Baby	NaN	Pull-Ups	True	5db3288aee7f2d6de4248977
477	511111304616	Beverages	NaN	V8 Hydrate	NaN	5bcdfc5a965c7d66d92731e9
1025	511111804604	Beverages	NaN	V8 Hydrate	False	5bcdfc5990fa074576779a15
1081	511111206330	Breakfast & Cereal	NaN	Dippin Dots® Cereal	NaN	5dc2d9d4a60b873d6b0666d2
1163	511111706328	Breakfast & Cereal	NaN	Dippin Dots® Cereal	NaN	5dc1fca91dda2c0ad7da64ae
64	511111805854	Health & Wellness	NaN	ONE A DAY® WOMENS	False	5da609991dda2c3e1416ae90
339	511111914051	Health & Wellness	NaN	ONE A DAY® WOMENS	NaN	5e5ff265ee7f2d0b35b2a18f
574	511111605546	Snacks	NaN	Baken-Ets	NaN	5d9d08d1a60b87376833e348
848	511111701781	Snacks	NaN	Baken-Ets	True	585a961fe4b03e62d1ce0e76

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

Type *Markdown* and LaTeX: α^2

In [11]:
```
receipt_table[receipt_table.duplicated(subset=['_id.$oid',], keep=False)
```

Out[11]:

bonusPointsEarned	bonusPointsEarnedReason	pointsEarned	purchasedItemCount	rewardsRecei

All the receipt id's are unique - No Duplicates found

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

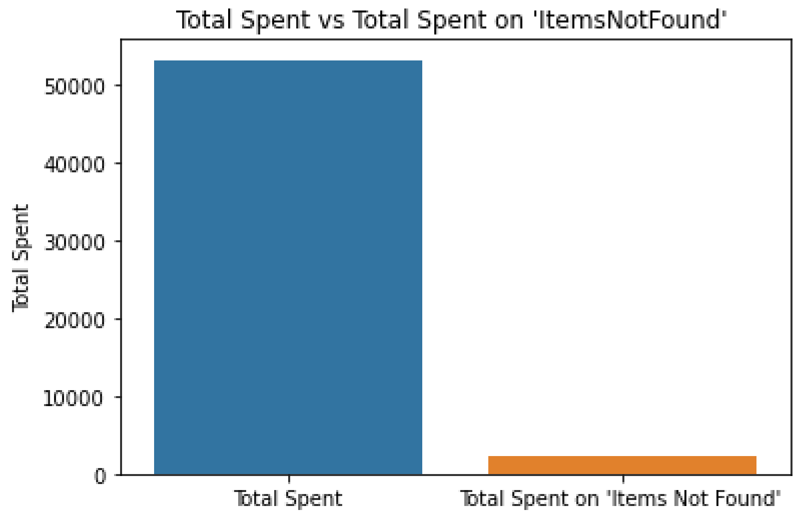
This will identify receipt entries with "ITEM NOT FOUND" in the description, which might indicate issues with barcode scanning or product data lookups. There are 6 different barcode's that don't have accurate descriptions. This needs to be highlighted.

In [12]:
```
print(receipt_item_table[receipt_item_table.description == 'ITEM NOT FOU
```

```
['4011' '686924155783' '22' '686924291290' '792851356565' '500011104752
4']
```

In [13]:
```
sns.barplot(x=['Total Spent', "Total Spent on 'Items Not Found' "], y=[r
plt.ylabel('Total Spent')
plt.title("Total Spent vs Total Spent on 'ItemsNotFound' ")
```

Out[13]: 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.

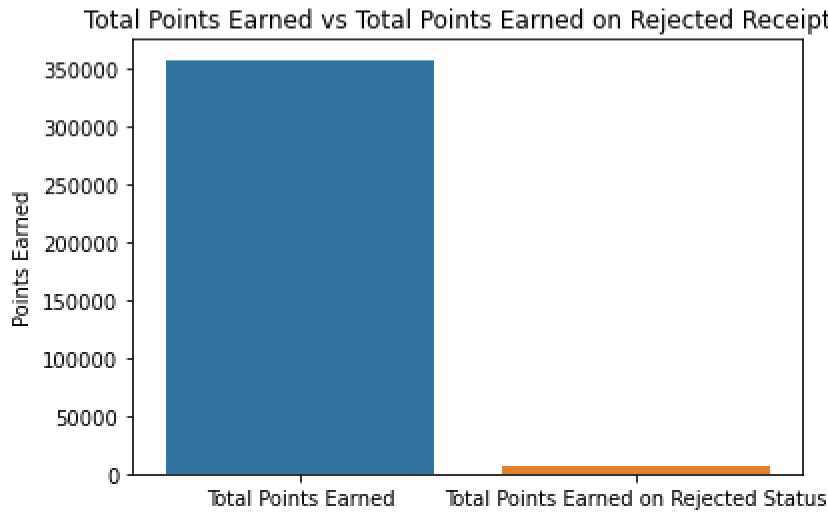
In [14]:
```
receipt_table.pointsEarned = receipt_table.pointsEarned.astype(float)
receipt_table[(receipt_table.rewardsReceiptStatus == 'REJECTED') & (rece
```

Out[14]:

	bonusPointsEarned	bonusPointsEarnedReason	pointsEarned	purchasedItemCount	rewardsRe
2	5.0	All-receipts receipt bonus	5.0	1.0	[{'needs False, 'pa
13	750.0	Receipt number 1 completed, bonus point schedu...	750.0	11.0	'0' 'comp
62	750.0	Receipt number 1 completed, bonus point schedu...	750.0	2.0	[{'descri austria
203	5.0	All-receipts receipt bonus	5.0	1.0	[{'ba 'finalPric
207	5.0	All-receipts receipt bonus	5.0	1.0	[{'ba 'finalPric

In [15]:
```
sns.barplot(x=['Total Points Earned', "Total Points Earned on Rejected St
plt.ylabel('Points Earned')
plt.title("Total Points Earned vs Total Points Earned on Rejected Receipt
```

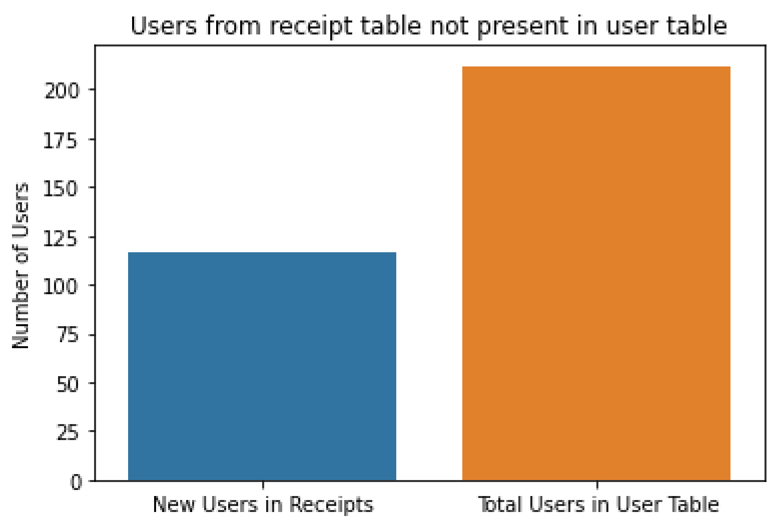
Out[15]:
```
Text(0.5, 1.0, 'Total Points Earned vs Total Points Earned on Rejected
Receipts ')
```



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

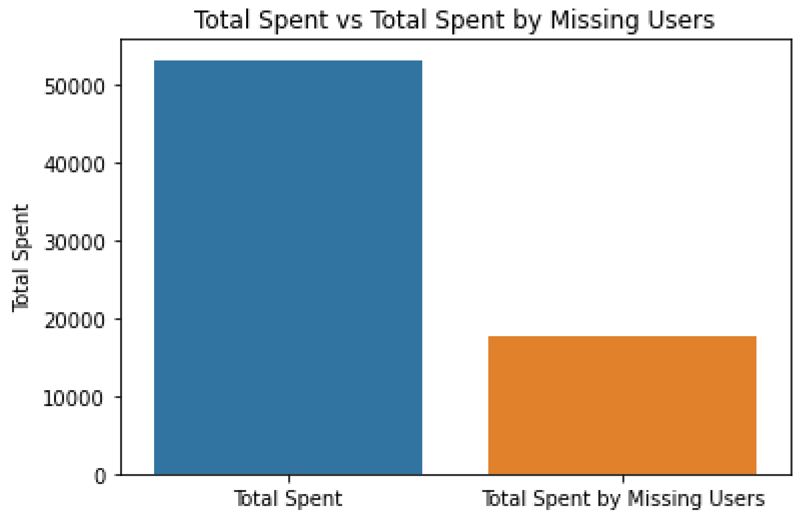
In [16]:
```
#list of user id from receipt table which are not present in user table
left_out_users = len(receipt_table[~receipt_table.userId.isin(user_table

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

Out[16]:
```
Text(0.5, 1.0, 'Users from receipt table not present in user table')
```



In [17]:
```
sns.barplot(x=['Total Spent', 'Total Spent by Missing Users'], y=[receip
plt.ylabel('Total Spent')
plt.title('Total Spent vs Total Spent by Missing Users')
```

Out[17]: Text(0.5, 1.0, 'Total Spent vs Total Spent by Missing Users')



Step 4.4 : Inconsistency between userFlaggedBarcode and originalMetaBriteBarcode fields

Proper documentation is needed in order to understand the discrepancies between userFlaggedBarcode, barcode and originalMetaBriteBarcode fields.

In [18]:
```
receipt_item_table[(receipt_item_table.barcode!= receipt_item_table.orig
```

Out[18]:

	userFlaggedBarcode	originalMetaBriteBarcode	barcode
7	4011	028400642255	4011
68	079400066619	080878042197	079400066619
175	079400066619	080878042197	079400066619
374	079400066619	080878042197	079400066619
392	4011	028400642255	4011

Once these issues are resolved, we can proceed with data transformation and even use more advanced statistical and machine learning methods to identify data quality issues.