In [1]:
```
#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} is not a gz file')
```

```
users.json is not a gz file
receipts.json is not a gz file
brands.json is not a gz file
```

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

i.e. (brand_table, receipt_table, user_table,item_table)

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')

item_table = receipt_table[['_id.$oid', 'rewardsReceiptItemList']]
item_table = item_table.rename(columns={'_id.$oid': 'receipt_id'})
item_table = item_table.explode('rewardsReceiptItemList')

expanded_receipts = json_normalize(item_table['rewardsReceiptItemList'])
item_table = pd.concat([ item_table.reset_index()['receipt_id'],expanded_
```

In [3]:
```
brand_table.head()
```

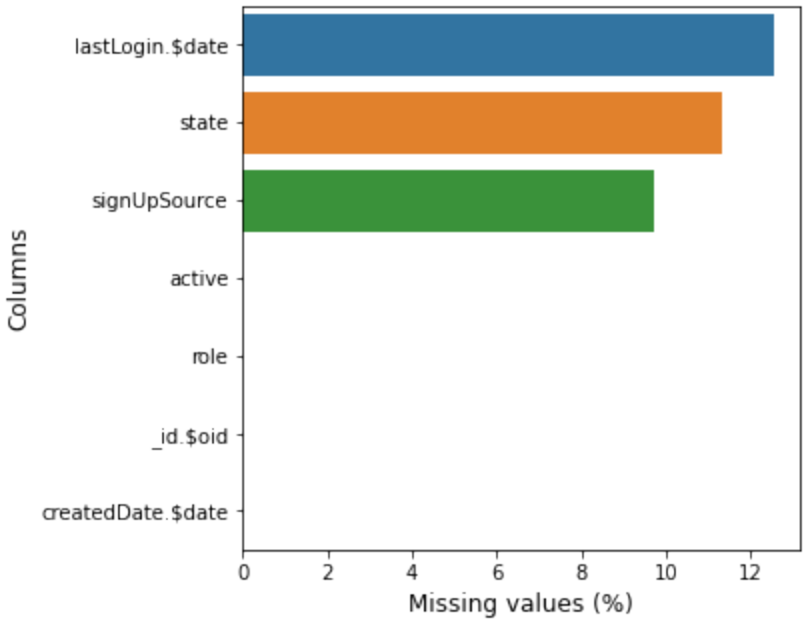
Out[3]:

	barcode	category	categoryCode	name	topBrand	
0	511111019862	Baking	BAKING	test brand @1612366101024	False	601ac115be37ce2ea
1	511111519928	Beverages	BEVERAGES	Starbucks	False	601c5460be37ce2ea
2	511111819905	Baking	BAKING	test brand @1612366146176	False	601ac142be37ce2ea
3	511111519874	Baking	BAKING	test brand @1612366146051	False	601ac142be37ce2ea
4	511111319917	Candy & Sweets	CANDY_AND_SWEETS	test brand @1612366146827	False	601ac142be37ce2ea

Step 2 : Null Value Check

In [116]:
```
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')
```

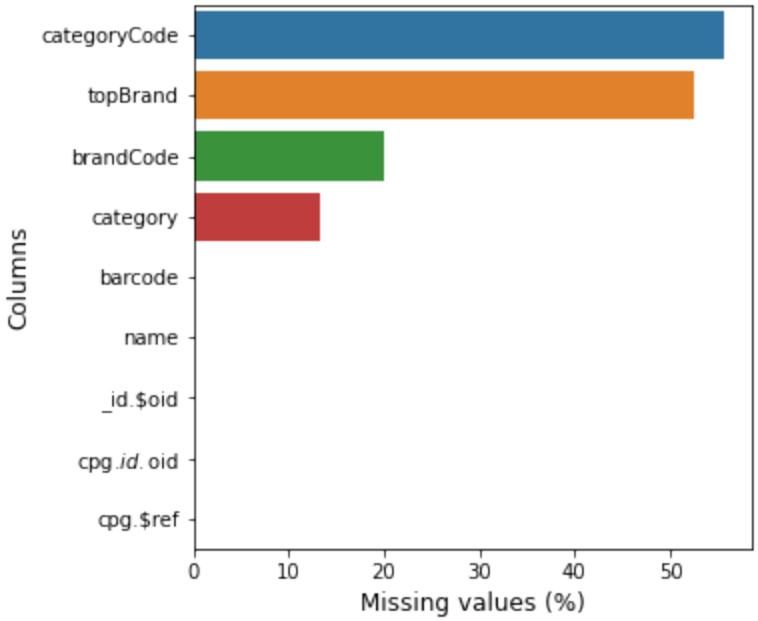
```
Missing values by percentage for Users:
```



Here, we can easily see that user-table has very less number of null data, with lastlogin column having only 12% of null values, which is within acceptable standards.

In [118]:
```
null_value_check(brand_table, "brands",orient='h')
```

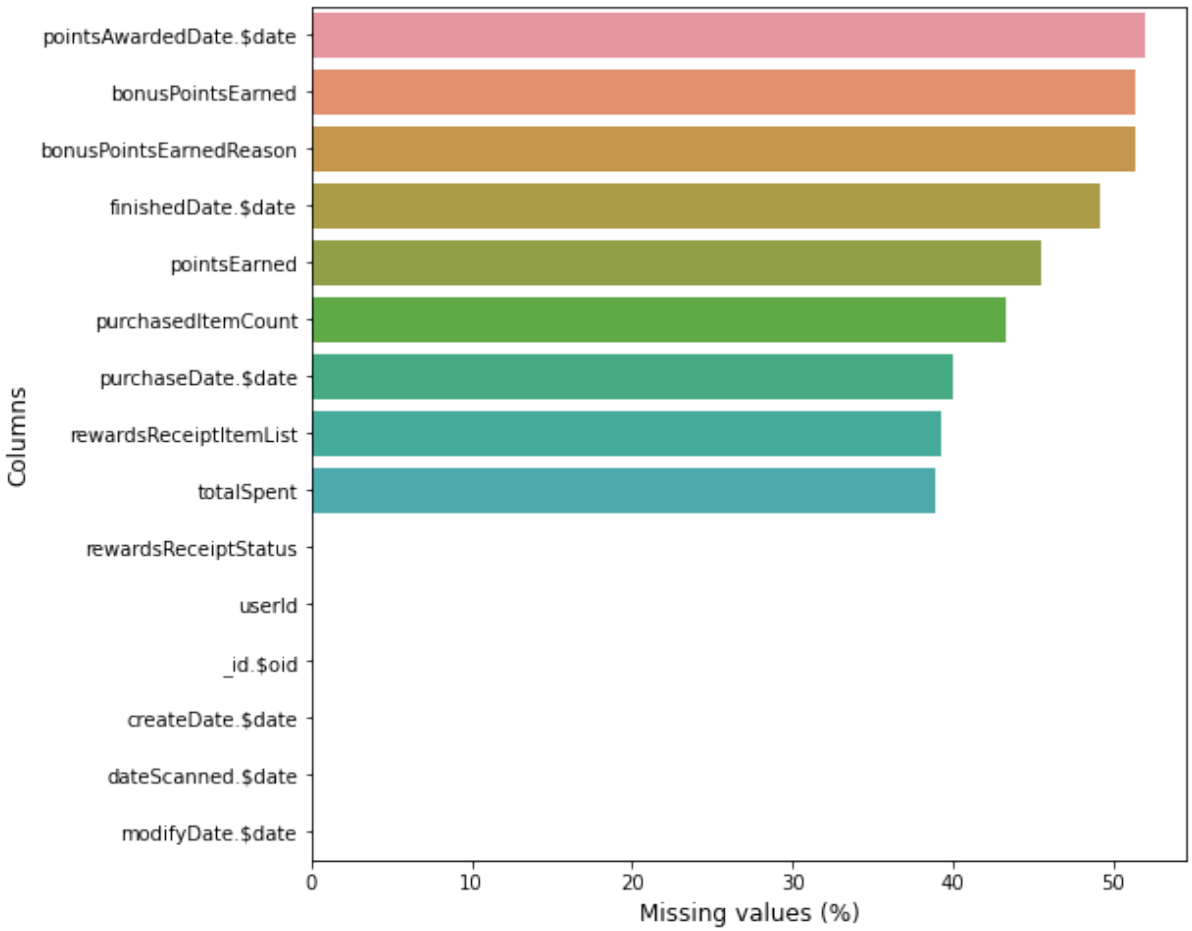
```
Missing values by percentage for brands:
```



The brands table is a bit problematic, with category code and topbrand columns having more than 50% null values, this requires us to carefully analyze data source.

In [120]:
```
null_value_check(receipt_table, "receipts", fig_width=8,fig_height=8, or
```

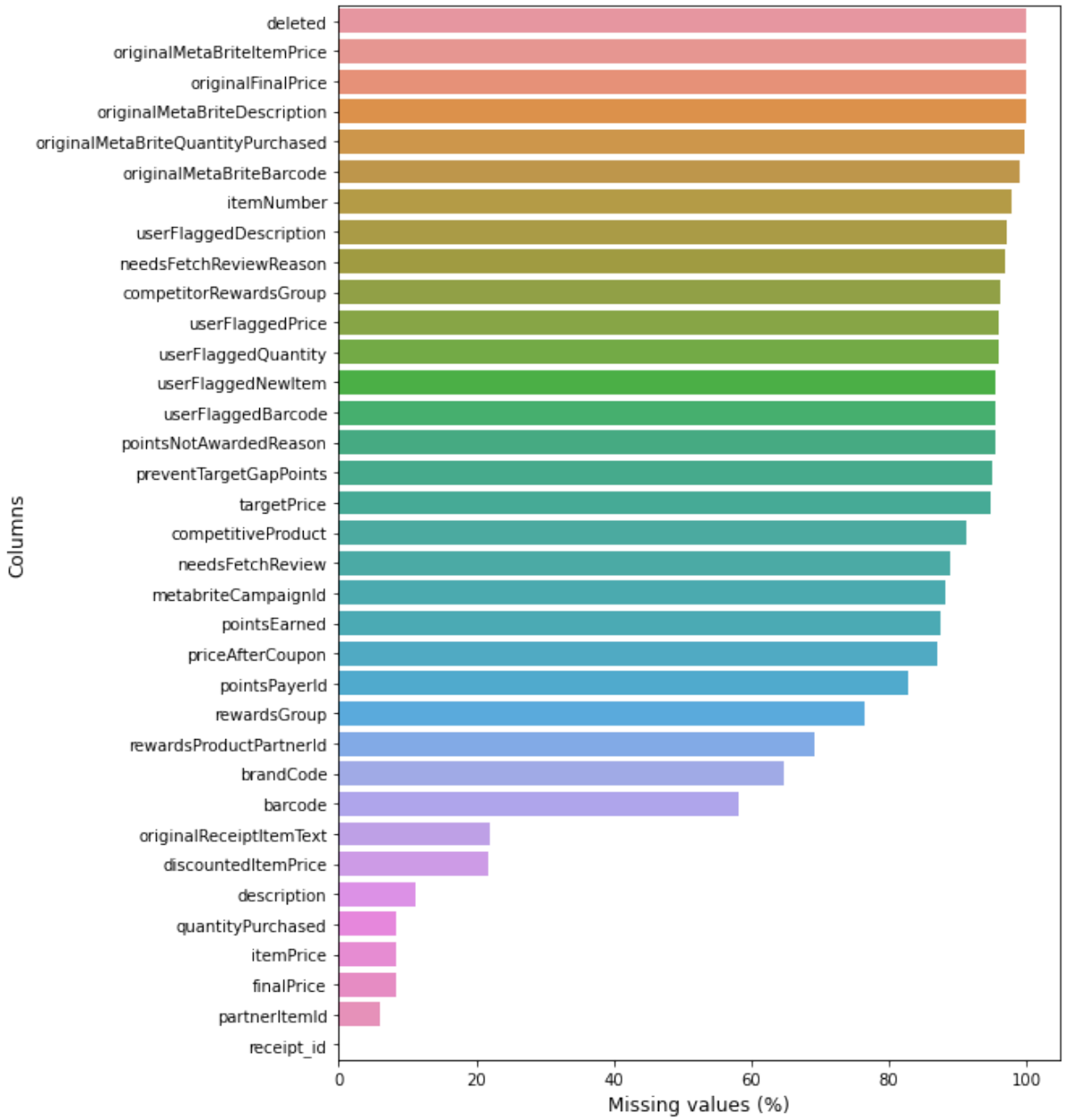
```
Missing values by percentage for receipts:
```



Receipts table has 9 data fields with more than 40% missing records. Some important columns are of concern such as purchasedate/finisheddate or bonus points earned.

In [123]:
```
null_value_check(item_table, "items",fig_width=8,fig_height=12,orient='h
```

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



Similarly, for the ReceiptItem table has majority of fields missing, 77% of the fields have more than 30% missing values.

Step 3 : Data Duplication Check

In this step, we will primarily check for duplicate records across all data sources.

In [161]:
```
#Duplicate user entries
print(f'Total Duplicate Rows in Users Data : {user_table.duplicated().su
user_table[user_table.duplicated(subset=['active','role','_id.$oid','cre
```

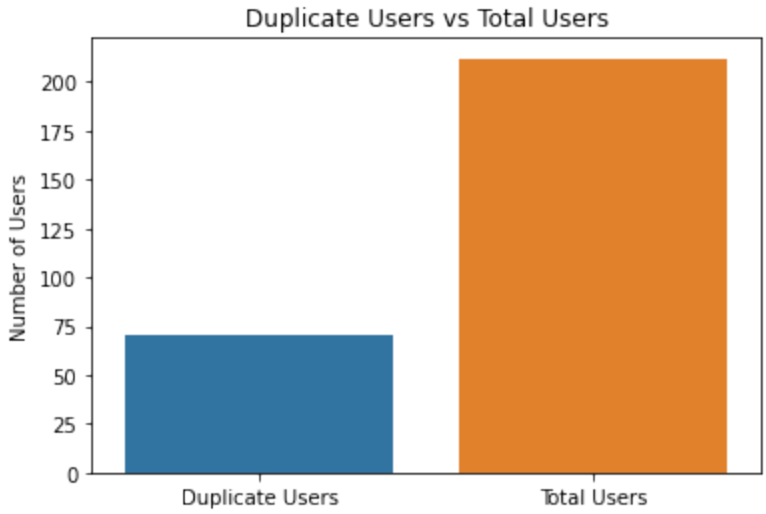
```
Total Duplicate Rows in Users Data : 282
```

Out[161]:

	active	role	signUpSource	state	_id.$oid	createdDate.$date	lastLogin.
1	True	consumer	Email	WI	5ff1e194b6a9d73a3a9f1052	1970-01-01 00:26:49.687444800	1.60968
3	True	consumer	Email	WI	5ff1e194b6a9d73a3a9f1052	1970-01-01 00:26:49.687444800	1.60968
4	True	consumer	Email	WI	5ff1e194b6a9d73a3a9f1052	1970-01-01 00:26:49.687444800	1.60968
7	True	consumer	Email	WI	5ff1e194b6a9d73a3a9f1052	1970-01-01 00:26:49.687444800	1.60968
9	True	consumer	Email	WI	5ff1e194b6a9d73a3a9f1052	1970-01-01 00:26:49.687444800	1.60968

In [168]:
```
# 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[168]:
```
Text(0.5, 1.0, 'Duplicate Users vs Total Users')
```



As we can see, we have a total of 70 duplicate users in the User table having same id, same active status and same 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.

In [126]:
```
brand_table[brand_table.duplicated(subset=['category','categoryCode','nar
```

Out[126]:

	barcode	category	categoryCode	name	topBrand	_id.$oid	
126	511111104698	Baby	NaN	Pull-Ups	False	5bd201a990fa074576779a19	550
978	511111312949	Baby	NaN	Pull-Ups	True	5db3288aee7f2d6de4248977	550
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

In [70]:
```
brand_table[brand_table.duplicated(subset=['category','categoryCode','nar
```

Out[70]:

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

For the brand table, we can see there is data duplicaton based on category, categoryCode, name and cpg-id fields. There is data redundancy, as these fields represent similar information in most of the cases. 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.

In [71]:
```
receipt_table[receipt_table.duplicated(subset=['_id.$oid',], keep=False)
```

Out[71]:

	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.

In [72]:
```
#This will identify receipt entries with "ITEM NOT FOUND" in the descrip
item_table[item_table.description == 'ITEM NOT FOUND'].barcode.unique()
```

Out[72]:
```
array(['4011', '686924155783', '22', '686924291290', '792851356565',
       '5000111047524'], dtype=object)
```

There are 6 different barcode's that don't have accurate descriptions. This needs to be highlighted.

In [53]:
```
#Receipts with rewardsReceiptStatus marked as "REJECTED" but having poin

receipt_table.pointsEarned = receipt_table.pointsEarned.astype(float)
receipt_table[(receipt_table.rewardsReceiptStatus == 'REJECTED') & (rece
```

Out[53]:

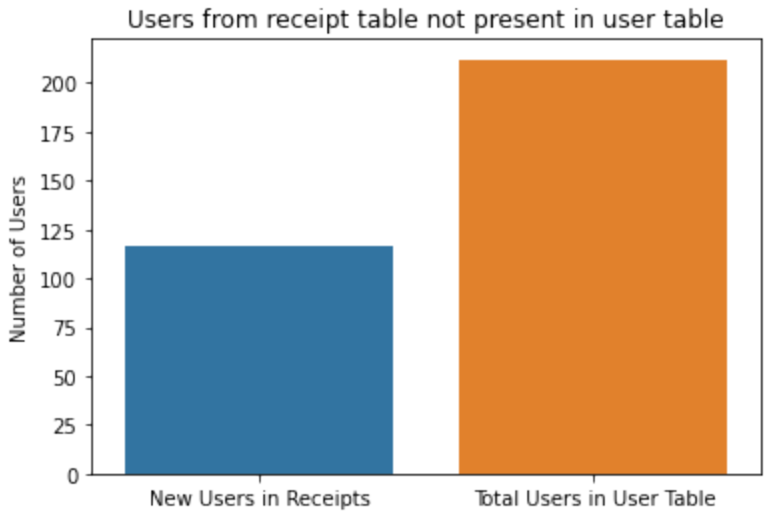
	bonusPointsEarned	bonusPointsEarnedReason	pointsEarned	purchasedItemCount	rewardsRe
2	5.0	All-receipts receipt bonus	5.0	1.0	[{'needs False, 'pa
13	750.0	Receipt number 1 completed, bonus point schedu...	750.0	11.0	'0 'comp
62	750.0	Receipt number 1 completed, bonus point schedu...	750.0	2.0	[{'descri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

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

In [172]:
```
#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[172]:
```
Text(0.5, 1.0, 'Users from receipt table not present in user table')
```



User Table has **117** missing users from receipt data, which accounts for nearly 30% of transcation amount by totalSpent. This might hamper related user segmentation studies.

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.