1 Business Problem

1.1 Overview

In the era of online stores, It's become essential and mandatory to know the demand of any product in near future. It will help stores to stock the product which demand is going to be high in near future. If we are able to do so, it will directly improve revenue of company.
i.e. if we have any event in next week, definitely the sell will increase on some product depends on the event.

1.2 Objective:

Our main objective is to predict the product's sell for the next 28 days.

1.3 Performance Matrix:

RMSE (Root Mean Square Error) : RMSE is a good measure when we care more about prediction. It is a square root of the average squared error.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

RMSE gives more importance to the most significant error. If there is slighlty error in demand our RMSE will increase significantly.

1.4 Why ML approach, and Why not normal Statstical Method

Statstical method can perfrom good when we have univarite dataset or one step prediction. It has been seen that for multi step prediction , ML methods works well.

In [1]:
```python
import numpy as np
import pandas as pd
import seaborn as sns
import plotly
import matplotlib.pyplot as plt
import plotly.express as px
import warnings
warnings.filterwarnings('ignore')
```

executed in 2.47s, finished 15:50:26 2021-03-14

In [2]:
```python
import shutil

total, used, free = shutil.disk_usage("/")

print("Total: %d GiB" % (total // (2**30)))
print("Used: %d GiB" % (used // (2**30)))
print("Free: %d GiB" % (free // (2**30)))
```

```
Total: 58 GiB
Used: 16 GiB
Free: 40 GiB
```

In [4]:
```
! mkdir ~/.kaggle
```

In [5]:
```
! cp kaggle.json ~/.kaggle/
#Make directory named kaggle and copy kaggle.json file there.
```

In [9]:
```
! chmod 600 ~/.kaggle/kaggle.json
```

In [10]:
```
!kaggle competitions download -c m5-forecasting-accuracy
```

```
Downloading m5-forecasting-accuracy.zip to /home/srivastavrajivkumar
 46%|█████████████████▌                    | 21.0M/45.8M [00:00<00:00, 81.0MB/s]
100%|██████████████████████████████████████| 45.8M/45.8M [00:00<00:00, 172MB/s]
```

In [15]:
```
!unzip "m5-forecasting-accuracy.zip"
```

```
Archive:  m5-forecasting-accuracy.zip
  inflating: calendar.csv
  inflating: sales_train_evaluation.csv
  inflating: sales_train_validation.csv
  inflating: sample_submission.csv
  inflating: sell_prices.csv
```

2 Calendar

In [61]:
```
# calendar = pd.read_csv('calendar.csv')
calendar.head(5)#checking top
```

Out[61]:

	date	wm_yr_wk	weekday	wday	month	year	d	event_name_1	event_type_1	event_name_2	event_type_2	snap_CA
0	2011-01-29	11101	Saturday	1	1	2011	d_1	NaN	NaN	NaN	NaN	0
1	2011-01-30	11101	Sunday	2	1	2011	d_2	NaN	NaN	NaN	NaN	0
2	2011-01-31	11101	Monday	3	1	2011	d_3	NaN	NaN	NaN	NaN	0
3	2011-02-01	11101	Tuesday	4	2	2011	d_4	NaN	NaN	NaN	NaN	1
4	2011-02-02	11101	Wednesday	5	2	2011	d_5	NaN	NaN	NaN	NaN	1

In [17]:
```
#What is the size of the data  calendar
print(f'Calendar has {calendar.shape[0]} rows and {calendar.shape[1]} columns.')
```

```
Calendar has 1969 rows and 14 columns.
```

In [ ]:
```
#what are the columns present in calendar
print(calendar.columns)
```

```
Index(['date', 'wm_yr_wk', 'weekday', 'wday', 'month', 'year', 'd',
       'event_name_1', 'event_type_1', 'event_name_2', 'event_type_2',
       'snap_CA', 'snap_TX', 'snap_WI'],
      dtype='object')
```

2.1 Column Description

>>>>>>>>

Column Name	Description
date	The date in a “y-m-d” format
wm_yrwk	The id of the week the date belongs to.
weekday	The type of the day (Saturday, Sunday, …, Friday).
wday	The id of the weekday, starting from Saturday.
month	The month of the date.
year	The year of the date.
d	no of days count
event_name_1	If the date includes an event, the name of this event.
event_type_1	If the date includes an event, the type of this event.
event_name_2	If the date includes a second event, the name of this event.
event_type_2	If the date includes a second event, the type of this event.
snap_CA snap_TX and snap_WI	A binary variable (0 or 1) indicating whether the stores of CA, TX or WI allow SNAP3 purchases on the examined date. 1 indicates that SNAP purchases areallowed.

In [ ]:
```
#checking types of data
calendar.dtypes
```

Out[18]:
```
date            object
wm_yr_wk         int64
weekday         object
wday             int64
month            int64
year             int64
d               object
event_name_1    object
event_type_1    object
event_name_2    object
event_type_2    object
snap_CA          int64
snap_TX          int64
snap_WI          int64
dtype: object
```

We have only two types of data in Calendar : int64 and object

int64 : 'wm_yr_wk', 'wday', 'month', 'year', 'snap_CA', 'snap_TX', 'snap_WI'

object64 : 'date', 'weekday', 'd', 'event_name_1', 'event_type_1', 'event_name_2', 'event_type_2',

In [64]:
```python
#checking null values

def missing_values(df):
    missing_count = df.isnull().sum()
    percent_missing = missing_count * 100 / len(df)
    values_available = len(df) - missing_count

    missing_value_df = pd.DataFrame({
        'column_name': df.columns,
        "non_misisng_count": values_available,
        'missing_count': missing_count,

        'percent_missing': percent_missing
    })
    missing_value_df.sort_values('percent_missing', inplace=True)

    return missing_value_df


missing_values(calendar)
```

Out[64]:

	column_name	non_misisng_count	missing_count	percent_missing
date	date	1969	0	0.000000
wm_yr_wk	wm_yr_wk	1969	0	0.000000
weekday	weekday	1969	0	0.000000
wday	wday	1969	0	0.000000
month	month	1969	0	0.000000
year	year	1969	0	0.000000
d	d	1969	0	0.000000
snap_CA	snap_CA	1969	0	0.000000
snap_TX	snap_TX	1969	0	0.000000
snap_WI	snap_WI	1969	0	0.000000
event_3	event_3	1969	0	0.000000
event_type_3	event_type_3	1969	0	0.000000
event_name_1	event_name_1	162	1807	91.772473
event_type_1	event_type_1	162	1807	91.772473
event_name_2	event_name_2	5	1964	99.746064

	column_name	non_misisng_count	missing_count	percent_missing
event_type_2	event_type_2	5	1964	99.746064

we have only 4 columns which has missing value in it
event_name_1 and event_type_1 has 91% missing values
event_name_2 and event_type_2 has 99% missing values

In [ ]:
```
#weekdays
print(f' weekdays {calendar.weekday.unique()}')

#wday
print(f' wday {calendar.wday.unique()}')
```

```
 weekdays ['Saturday' 'Sunday' 'Monday' 'Tuesday' 'Wednesday' 'Thursday' 'Friday']
 wday [1 2 3 4 5 6 7]
```

We have 7 days saturady to friday where 1 donates to saturday , 2 donates to sunday and so on.

[What is snap Purchasing (https://www.kaggle.com/c/m5-forecasting-accuracy/discussion/133614)](https://www.kaggle.com/c/m5-forecasting-accuracy/discussion/133614)

The United States federal government provides a nutrition assistance benefit called the Supplement Nutrition Assistance Program (SNAP). SNAP provides low income families and individuals with an Electronic Benefits Transfer debit card to purchase food products. In many states, the monetary benefits are dispersed to people across 10 days of the month and on each of these days 1/10 of the people will receive the benefit on their card.

ref: [https://www.kaggle.com/c/m5-forecasting-accuracy/discussion/133614 (https://www.kaggle.com/c/m5-forecasting-accuracy/discussion/133614)](https://www.kaggle.com/c/m5-forecasting-accuracy/discussion/133614)

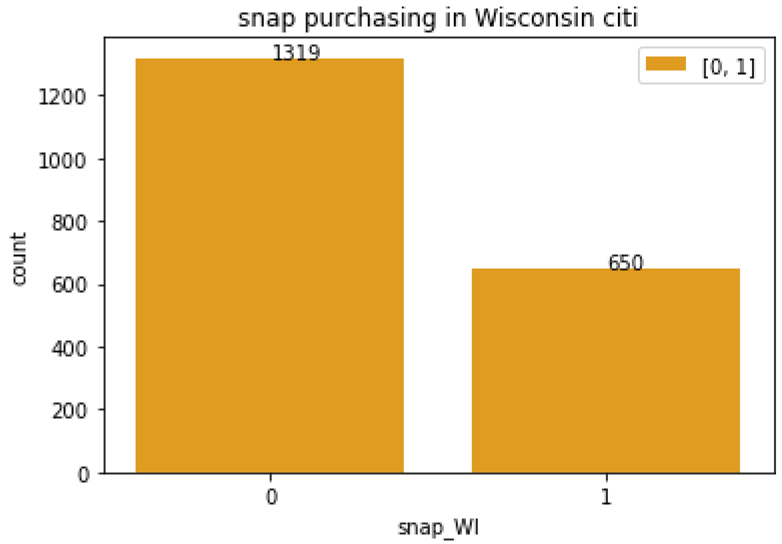
In [ ]:
```
#unique values in snap coloumns
print(f' California : {calendar.snap_CA.unique()}')
print(f' Texas : {calendar.snap_TX.unique()}')
print(f' Wisconsin : {calendar.snap_WI.unique()}')
```

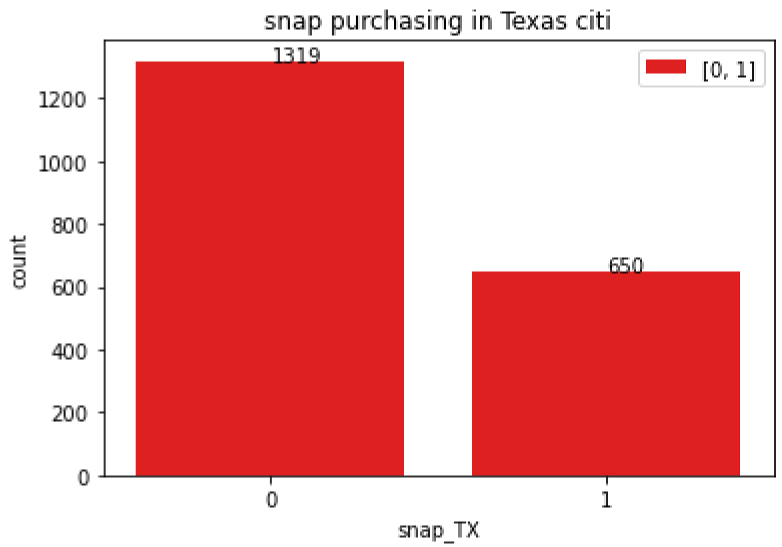
```
 California : [0 1]
 Texas : [0 1]
 Wisconsin : [0 1]
```

We have two values 0 and 1 for each state in snap columns. Where 1 indicates that snap purcahes is allowed and 0 indicates that purchase is not allowed.

In [ ]:

```
#Wisconsin Citi
plots = sns.countplot(calendar.snap_WI, label = [0,1], color = 'orange')

for bar in plots.patches:
    plots.annotate(bar.get_height() ,
                   (bar.get_x()  + bar.get_width() / 2,
                    bar.get_height()))
plt.legend()
plt.title('snap purchasing in Wisconsin citi')
plt.show()



#Texas Citi
plots = sns.countplot(calendar.snap_TX, label = [0,1], color = 'red')

for bar in plots.patches:
    plots.annotate(bar.get_height(),
                   (bar.get_x()  +  bar.get_width() / 2,
                    bar.get_height()))
plt.legend()
plt.title('snap purchasing in Texas citi')
plt.show()


#California
plots = sns.countplot(calendar.snap_CA,  color = 'green')

for bar in plots.patches:
    plots.annotate(bar.get_height(),
                   (bar.get_x() + bar.get_width() / 2,
                    bar.get_height()))
plt.legend()
plt.title('snap purchasing in California citi')
plt.show()
```

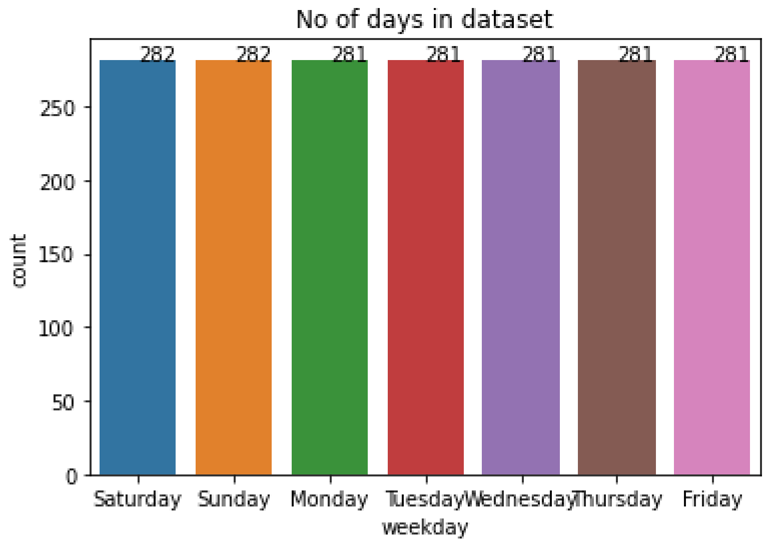





```
No handles with labels found to put in legend.
```

🛒 1316 purchase made without snap and 650 wit snap for all the three CA, TX and WI states,

In [ ]:
```
plots = sns.countplot(calendar.weekday )

for bar in plots.patches:
    plots.annotate(bar.get_height(),
                   (bar.get_x() + bar.get_width() / 2,
                    bar.get_height()))
plt.title("No of days in dataset")
plt.show()
```

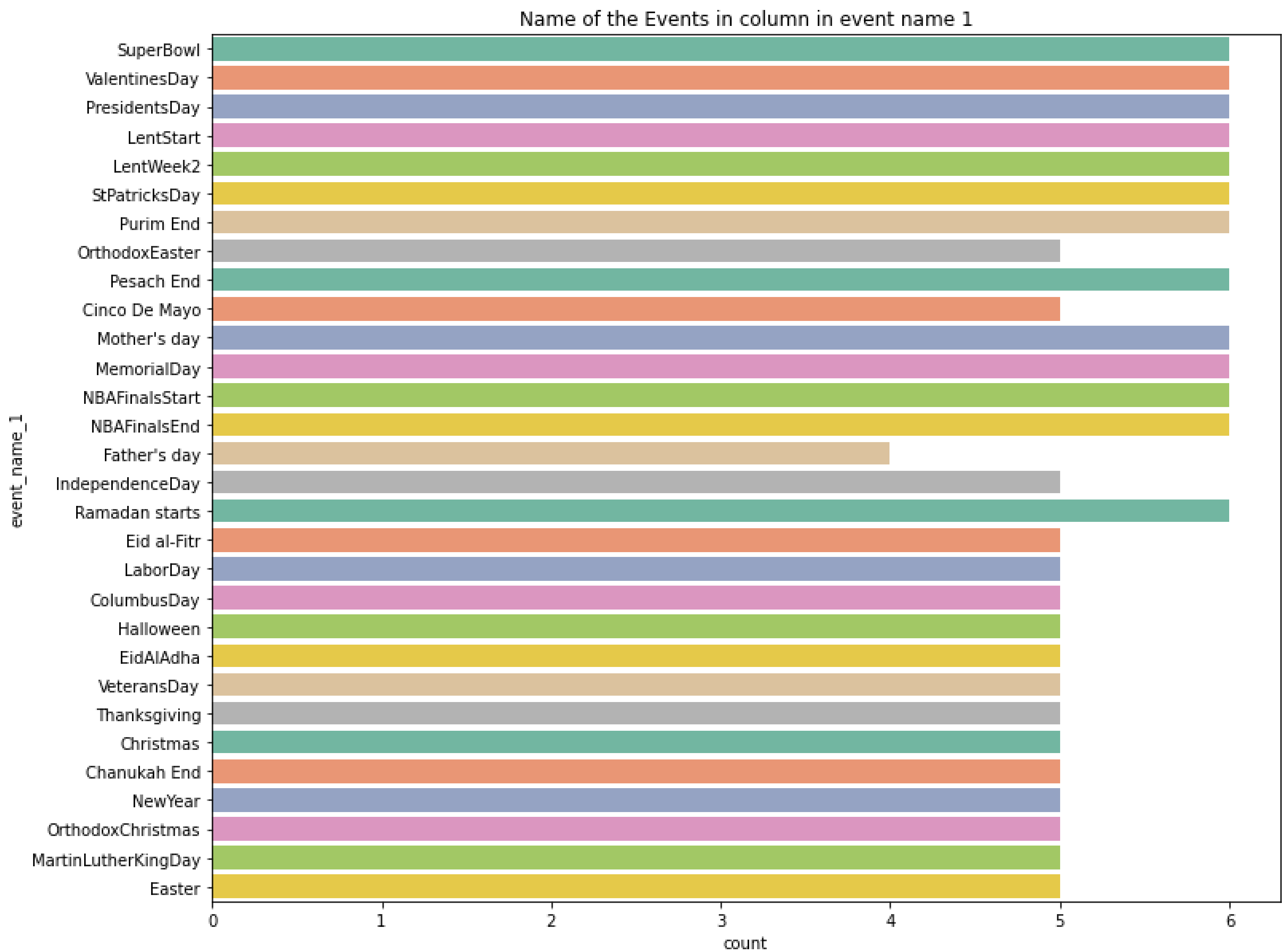


- We have balanced date for each day

In [ ]:
```
print(f' all unique values in  event_name_1 column {calendar.event_name_1.nunique()}')
```

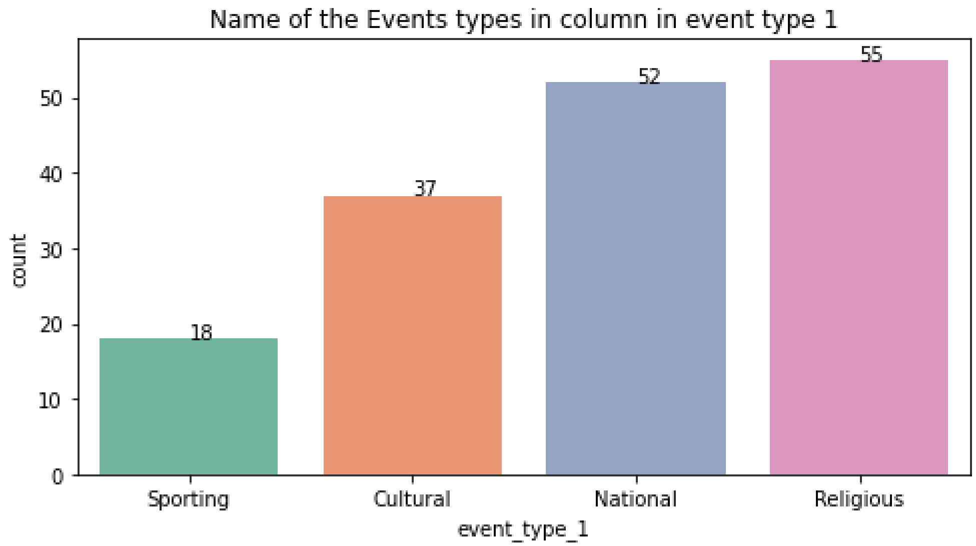
```
all unique values in  event_name_1 column 30
```

In [ ]:
```
plt.figure(figsize=(12, 10))
plots = sns.countplot(y=calendar.event_name_1,  palette = "Set2")
plt.title("Name of the Events in column in event name 1")
plt.show()
```



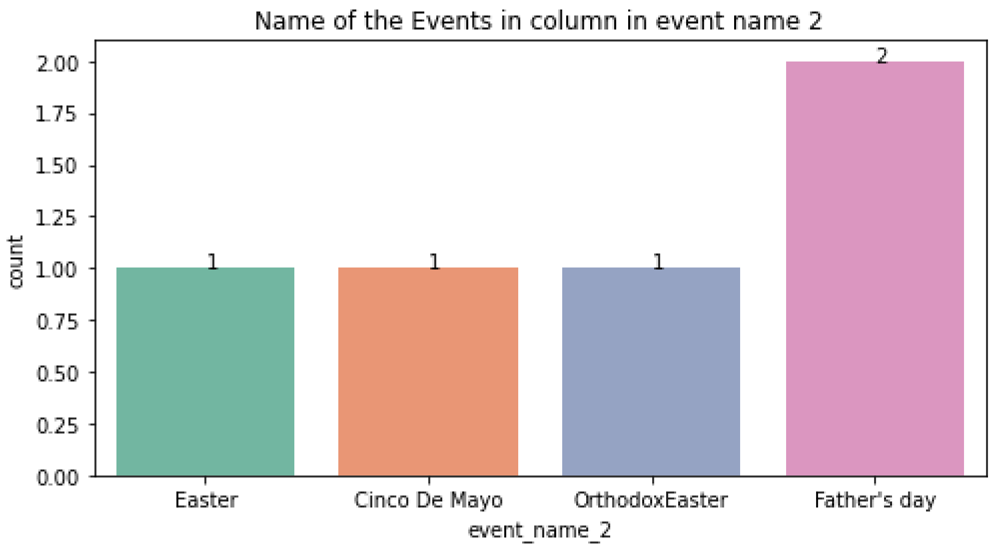
In [ ]:

```
plt.figure(figsize=(8, 4))
plots = sns.countplot(x=calendar.event_type_1,  palette = "Set2")
for bar in plots.patches:
    plots.annotate(bar.get_height(),
                   (bar.get_x() + bar.get_width() / 2,
                    bar.get_height()))
plt.title("Name of the Events types in column in event type 1")
plt.show()
```



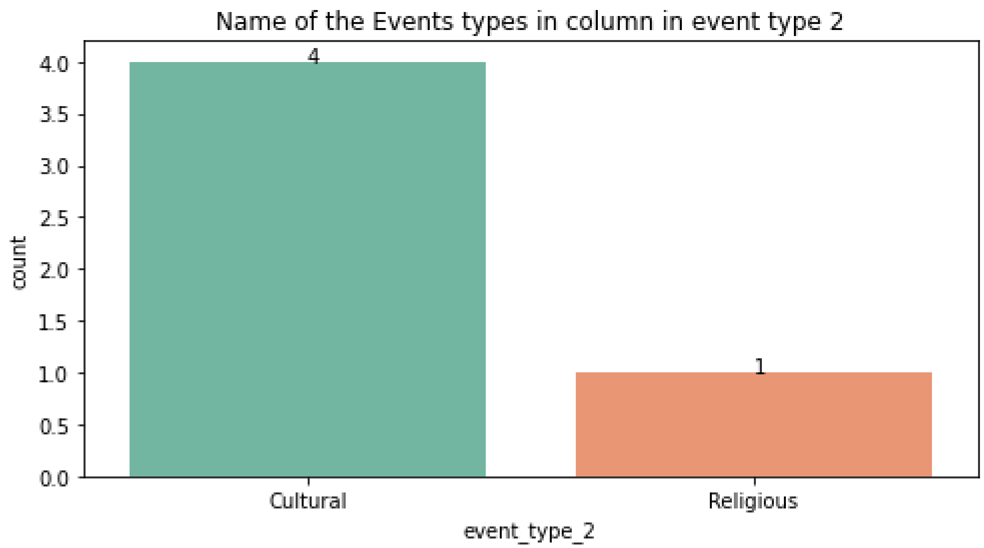
In [ ]:

```
plt.figure(figsize=(8, 4))
plots = sns.countplot(x=calendar.event_name_2,  palette = "Set2")
for bar in plots.patches:
    plots.annotate(bar.get_height(),
                   (bar.get_x() + bar.get_width() / 2,
                    bar.get_height()))
plt.title("Name of the Events in column in event name 2")
plt.show()
```



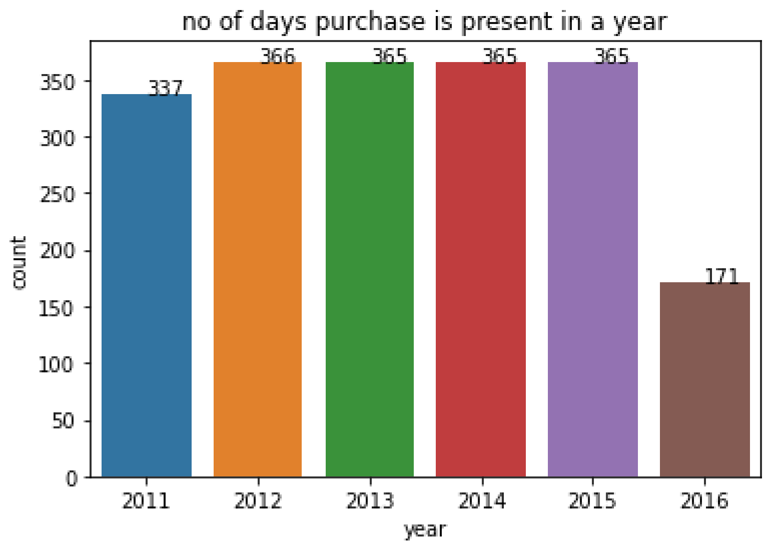
In [ ]:

```
#
plt.figure(figsize=(8, 4))
plots = sns.countplot(x=calendar.event_type_2,  palette = "Set2")
for bar in plots.patches:
    plots.annotate(bar.get_height(),
                   (bar.get_x() + bar.get_width() / 2,
                    bar.get_height()))
plt.title("Name of the Events types in column in event type 2")
plt.show()
```

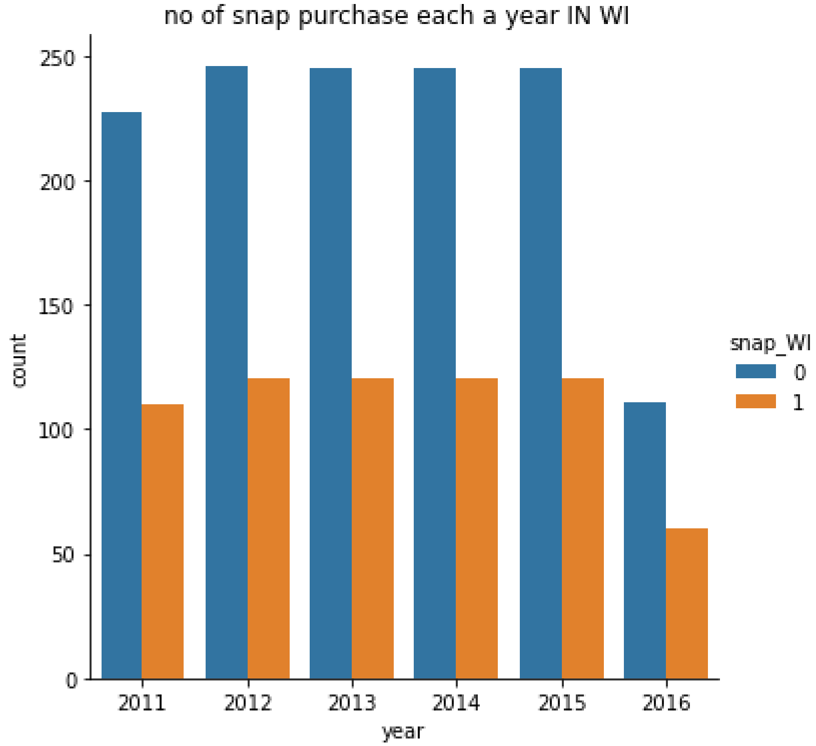


In [ ]:

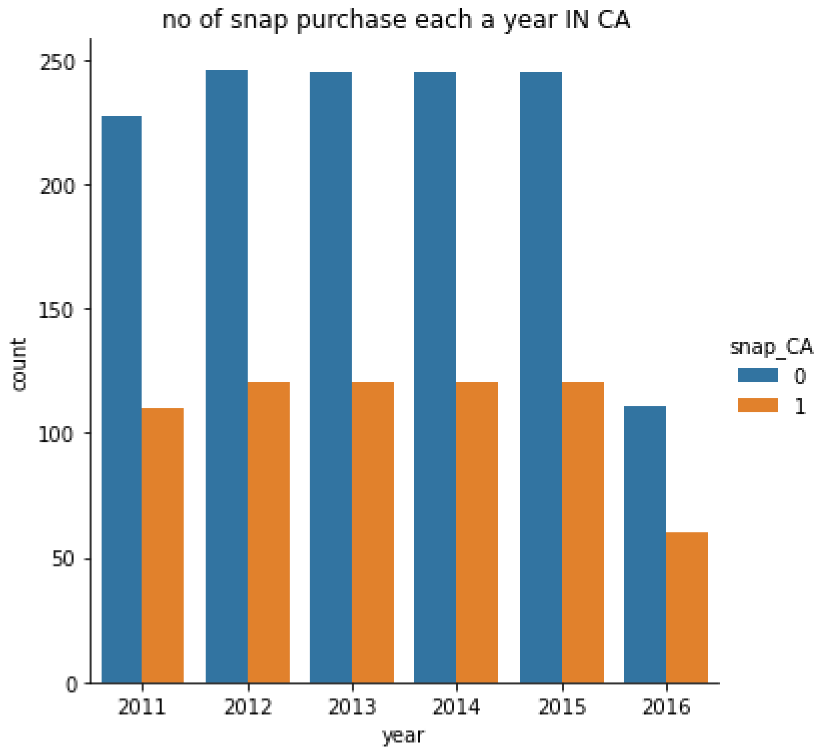
```
#no of days purchase is present in a year
plots = sns.countplot(data = calendar, x = 'year' )
for bar in plots.patches:
    plots.annotate(bar.get_height(),
                   (bar.get_x() + bar.get_width() / 2,
                    bar.get_height()))
plt.title("no of days purchase is present in a year")

plt.show()
```



In [ ]:
```
#no of snap purchase each a year WI
sns.catplot(x='year', data=calendar, kind = 'count', hue = 'snap_WI')
plt.title("no of snap purchase each a year IN WI  ")
plt.show()
```

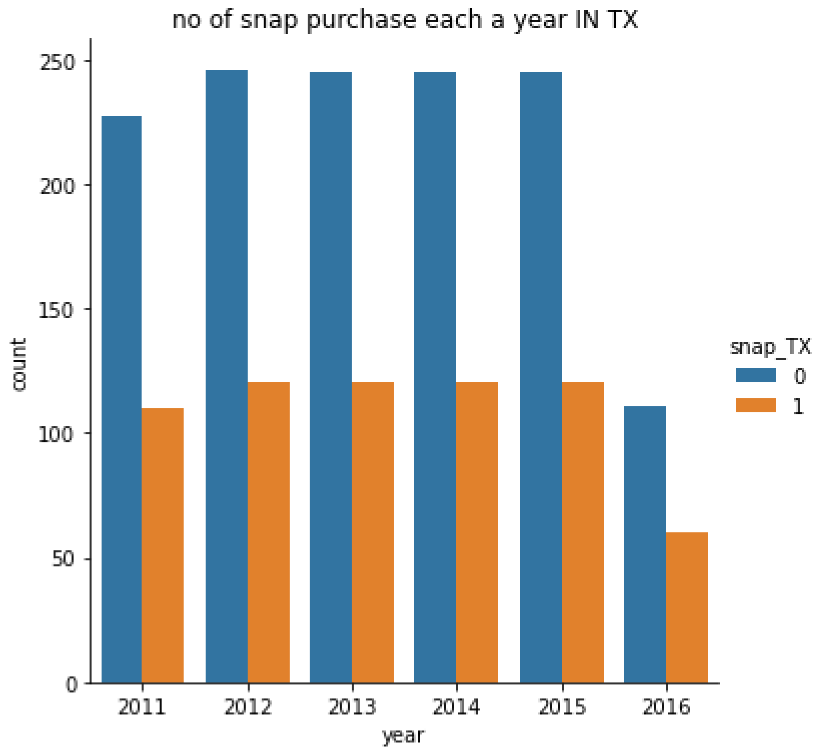


In [ ]:
```
#no of snap purchase each a year CA
sns.catplot(x='year', data=calendar, kind = 'count', hue = 'snap_CA')
plt.title("no of snap purchase each a year IN CA  ")
plt.show()
```



In [ ]:

```
#no of snap purchase each a year
sns.catplot(x='year', data=calendar, kind = 'count', hue = 'snap_TX')
plt.title("no of snap purchase each a year IN TX")
plt.show()
```



2.1.1 Feature enginerring

In [65]:

```
#creating two new columns
calendar['event_3'] = calendar[['event_name_1', 'event_name_2']].apply(lambda x: ', '.join(x[x.notnull()]), axis = 1)
calendar['event_type_3'] = calendar[['event_type_1', 'event_type_2']].apply(lambda x: ', '.join(x[x.notnull()]), axis = 1)
```

3 Sales Train Evaluation

In [18]:
```
sales_train_eval = pd.read_csv("sales_train_evaluation.csv")
sales_train_eval.head()
```

Out[18]:

	id	item_id	dept_id	cat_id	store_id	state_id	d_1	d_2	d_3	d_4	...	d_1932	d_19
0	HOBBIES_1_001_CA_1_evaluation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	2	
1	HOBBIES_1_002_CA_1_evaluation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	0	
2	HOBBIES_1_003_CA_1_evaluation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	1	
3	HOBBIES_1_004_CA_1_evaluation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	1	
4	HOBBIES_1_005_CA_1_evaluation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	0	

5 rows × 1947 columns

In [19]:
```
sales_train_val = pd.read_csv("sales_train_validation.csv")
sales_train_val.head()
```

Out[19]:

	id	item_id	dept_id	cat_id	store_id	state_id	d_1	d_2	d_3	d_4	...	d_1904	d_190
0	HOBBIES_1_001_CA_1_validation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	1	
1	HOBBIES_1_002_CA_1_validation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	0	
2	HOBBIES_1_003_CA_1_validation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	2	
3	HOBBIES_1_004_CA_1_validation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	1	
4	HOBBIES_1_005_CA_1_validation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	...	2	

5 rows × 1919 columns

In [ ]:
```
#What is the size of the data  sales_train_eval
print(f'sales_train_eval has {sales_train_eval.shape[0]} rows and {sales_train_eval.shape[1]} columns.')
```

```
sales_train_eval has 30490 rows and 1947 columns.
```

In [ ]:
```
#column names
sales_train_eval.columns
```

Out[35]:
```
Index(['id', 'item_id', 'dept_id', 'cat_id', 'store_id', 'state_id', 'd_1',
       'd_2', 'd_3', 'd_4',
       ...
       'd_1932', 'd_1933', 'd_1934', 'd_1935', 'd_1936', 'd_1937', 'd_1938',
       'd_1939', 'd_1940', 'd_1941'],
      dtype='object', length=1947)
```

3.1 Column Description

Column Name	Description
item_id	The id of the product.
dept_id	The id of the department the product belongs to.
cat_id	The id of the category the product belongs to.
store_id	The id of the store where the product is sold.
state_id	The State where the store is located.
d_1, d_2, …, d_i, … d_1941	The number of units sold at day i, starting from 2011-01-29.

In [ ]:
```
#checking null values
sales_train_eval.isnull().any()
```

Out[36]:
```
id          False
item_id     False
dept_id     False
cat_id      False
store_id    False
            ...
d_1937      False
d_1938      False
d_1939      False
d_1940      False
d_1941      False
Length: 1947, dtype: bool
```

In [ ]:
```
#checking null values
missing_values(sales_train_eval)
```

Out[37]:

	column_name	non_misisng_count	missing_count	percent_missing
id	id	30490	0	0.0
d_1300	d_1300	30490	0	0.0
d_1299	d_1299	30490	0	0.0
d_1298	d_1298	30490	0	0.0
d_1297	d_1297	30490	0	0.0
...	...	...	...	...
d_637	d_637	30490	0	0.0
d_636	d_636	30490	0	0.0
d_635	d_635	30490	0	0.0
d_663	d_663	30490	0	0.0
d_1941	d_1941	30490	0	0.0

1947 rows × 4 columns

We don't have any null values in any column

In [ ]:
```
#types of columns
sales_train_eval.dtypes
```

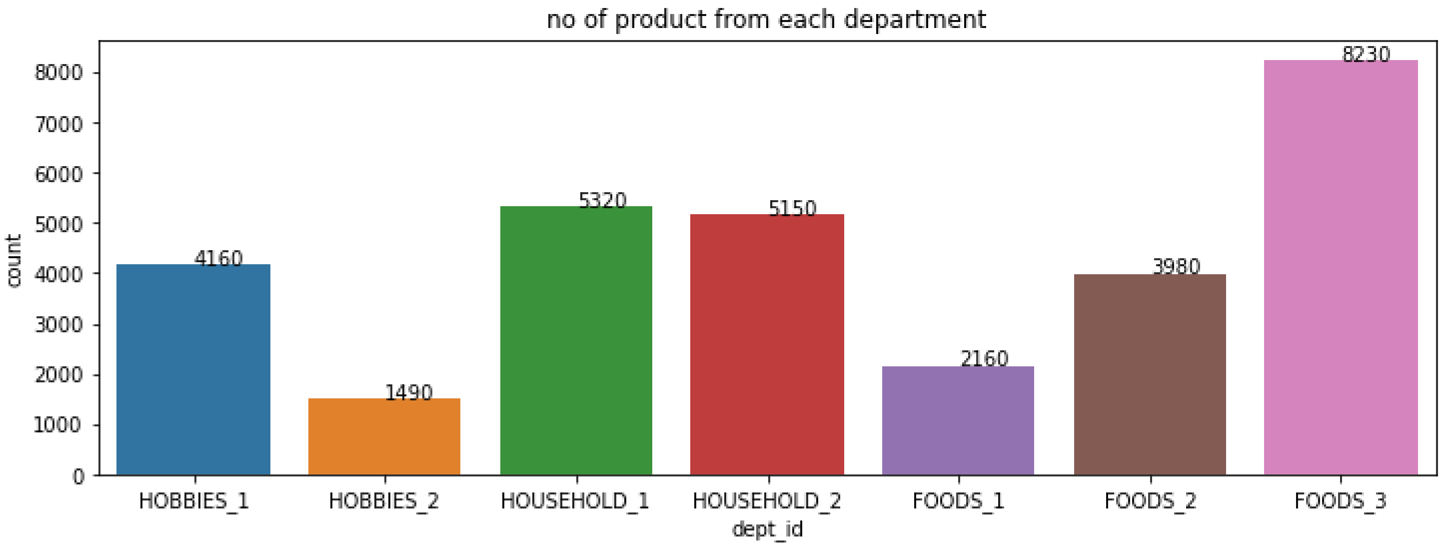
Out[38]:
```
id          object
item_id     object
dept_id     object
cat_id      object
store_id    object
             ...
d_1937       int64
d_1938       int64
d_1939       int64
d_1940       int64
d_1941       int64
Length: 1947, dtype: object
```

In [ ]:
```
#no of unique itemid
print(f'no of unique item: { sales_train_eval.item_id.nunique()}')
```

```
no of unique item: 3049
```

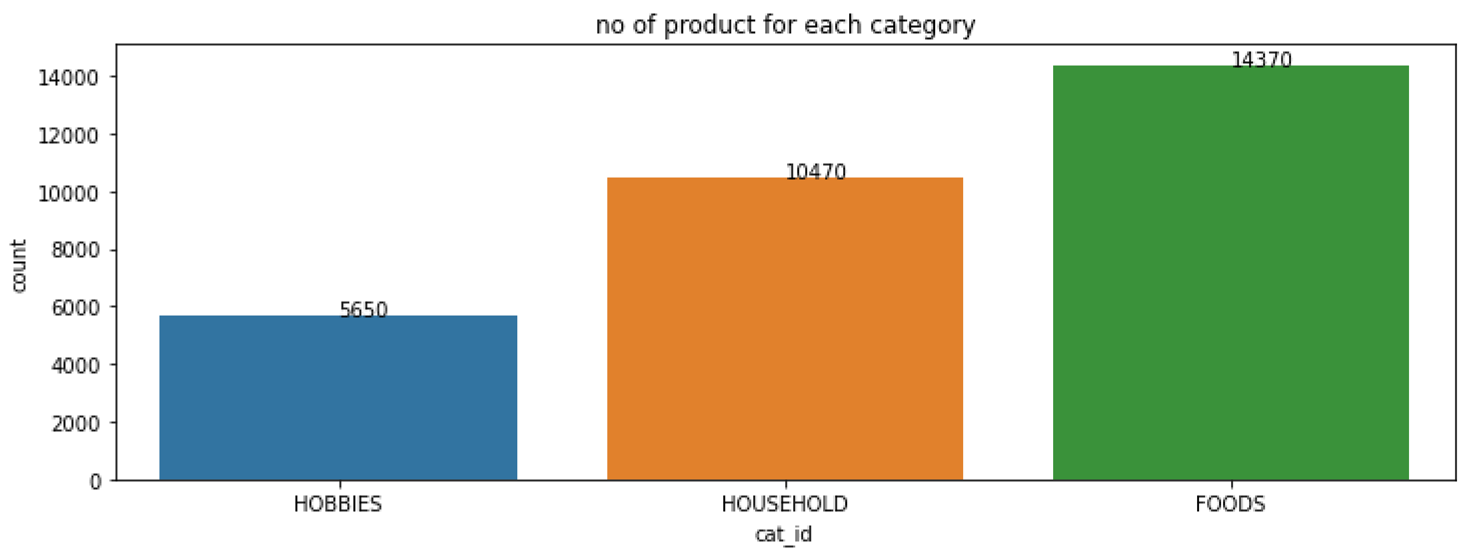
In [ ]:
```
#no of unique department:
print(f'no of unique department: { sales_train_eval.dept_id.nunique()}')
plt.figure(figsize=(12, 4))
plots = sns.countplot(x=sales_train_eval.dept_id)

for bar in plots.patches:
    plots.annotate(bar.get_height(),
                   (bar.get_x() + bar.get_width() / 2,
                    bar.get_height()))
plt.title('no of product from each department')
plt.show()
```

```
no of unique department: 7
```



In [ ]:
```
print(f' no of unique values in category : {sales_train_eval.cat_id.nunique()}')

plt.figure(figsize=(12, 4))
plots = sns.countplot(x=sales_train_eval.cat_id)

for bar in plots.patches:
    plots.annotate(bar.get_height(),
                   (bar.get_x() + bar.get_width() / 2,
                    bar.get_height()))
plt.title('no of product for each category')
plt.show()
```

```
no of unique values in category : 3
```



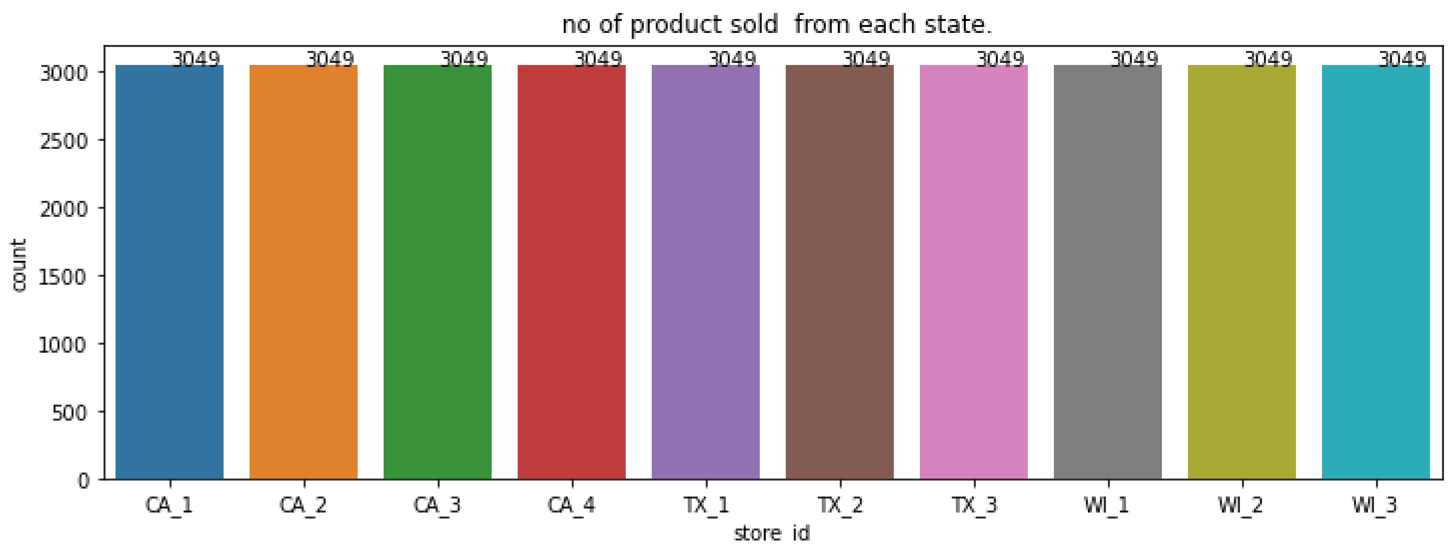
In [ ]:
```
print(f' no of unique state : {sales_train_eval.state_id.nunique()}')

plt.figure(figsize=(12, 4))
plots = sns.countplot(x=sales_train_eval.state_id)

for bar in plots.patches:
    plots.annotate(bar.get_height(),
                   (bar.get_x() + bar.get_width() / 2,
                    bar.get_height()))

plt.title('no of store located in each state')
plt.show()
```

```
no of unique state : 3
```

CA has 12196 , TX has 9147 ad WI has 9147 store.

In [ ]:
```
print(f' no of unique store : {sales_train_eval.store_id.nunique()}')

plt.figure(figsize=(12, 4))
plots = sns.countplot(x=sales_train_eval.store_id)

for bar in plots.patches:
    plots.annotate(bar.get_height(),
                   (bar.get_x() + bar.get_width() / 2,
                    bar.get_height()))

plt.title(' no of product sold  from each state.')
plt.show()
```

```
 no of unique store : 10
```



We have 3049*3 = 9147 products sold from every state

4 Sell Price

In [20]:
```
sell_price = pd.read_csv("sell_prices.csv")
sell_price.head()
```

Out[20]:

	store_id	item_id	wm_yr_wk	sell_price
0	CA_1	HOBBIES_1_001	11325	9.58
1	CA_1	HOBBIES_1_001	11326	9.58
2	CA_1	HOBBIES_1_001	11327	8.26
3	CA_1	HOBBIES_1_001	11328	8.26
4	CA_1	HOBBIES_1_001	11329	8.26

In [ ]:
```
#What is the size of the data  sell_price
print(f'sell_price has {sell_price.shape[0]} rows and {sell_price.shape[1]} columns.')
```

```
sell_price has 6841121 rows and 4 columns.
```

4.1 Description Column

Column Name	Description
store_id	**The id of the store where the product is sold.**
item_id	**The id of the product.**
wm_yr_wk	**The id of the week.**
sell_price	**The price of the product for the given week/store. The price is provided per week (average across seven days). If not available, this means that the product was not sold during the examined week. Note that although prices are constant at weekly basis, they may change through time (both training and test set).**

In [ ]:
```
# What is the type of each column
sell_price.dtypes
```

Out[46]:
```
store_id       object
item_id        object
wm_yr_wk        int64
sell_price    float64
dtype: object
```

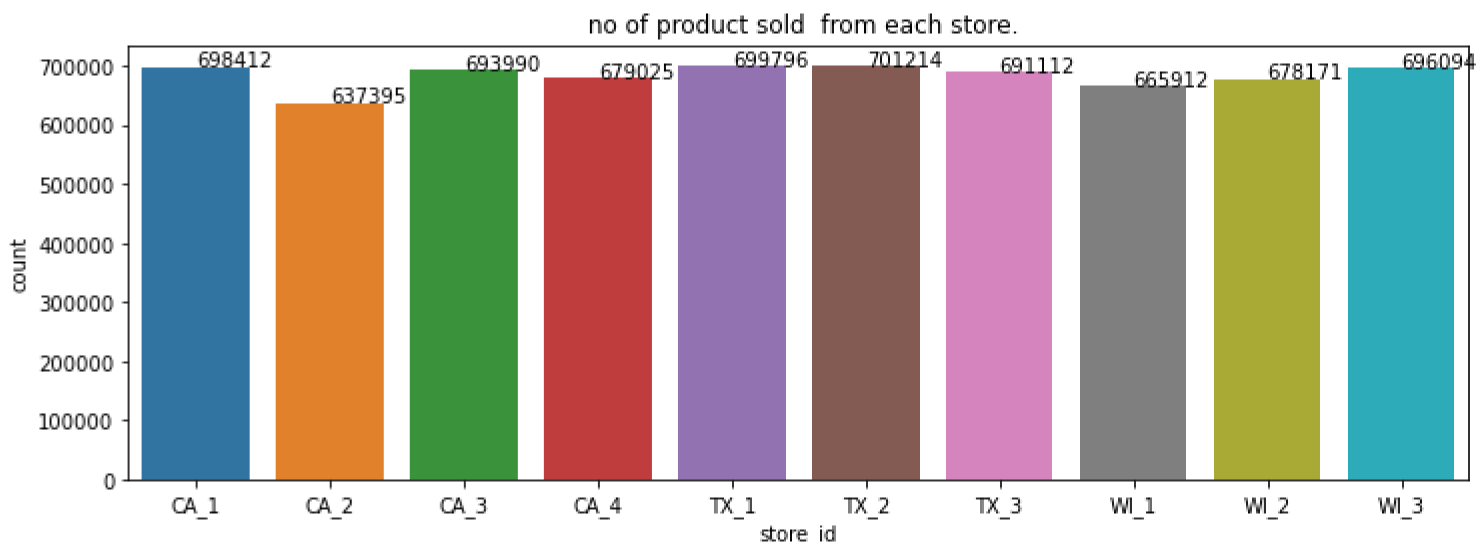
In [ ]:
```
#checking null values
missing_values(sell_price)
```

Out[47]:

	column_name	non_misisng_count	missing_count	percent_missing
store_id	store_id	6841121	0	0.0
item_id	item_id	6841121	0	0.0
wm_yr_wk	wm_yr_wk	6841121	0	0.0
sell_price	sell_price	6841121	0	0.0

In [ ]:
```
#no of unique each store in store_id
print(f' no of unique store : {sell_price.store_id.nunique()}')

plt.figure(figsize=(12, 4))
plots = sns.countplot(x=sell_price.store_id)

for bar in plots.patches:
    plots.annotate(bar.get_height(),
                   (bar.get_x() + bar.get_width() / 2,
                    bar.get_height()))
plt.title(' no of product sold  from each store.')
plt.show()
```

```
no of unique store : 10
```



In [ ]:
```
#no of unique item in item_id
print(f' no of unique item: {sell_price.item_id.nunique()}')
```

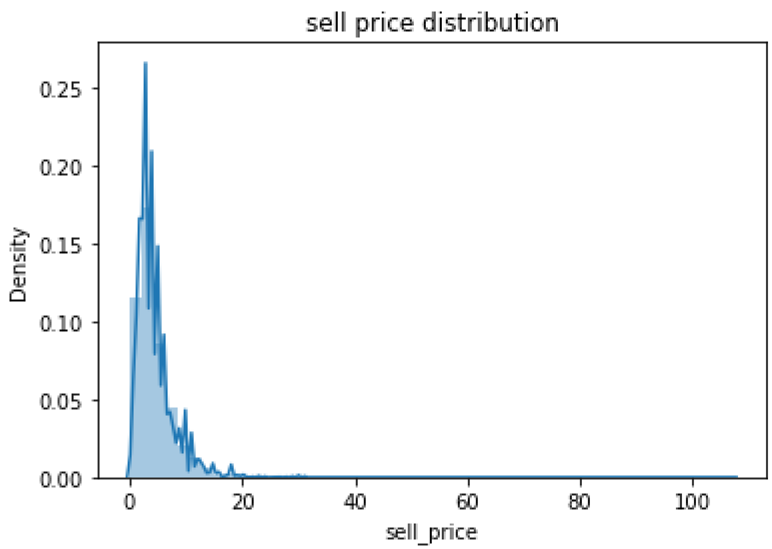
```
no of unique item: 3049
```

In [ ]:
```
#sell price
sell_price.sell_price.describe().apply(lambda x: format(x, 'f')) # lambda function is used here to avoid scintific notatic
```

Out[50]:
```
count    6841121.000000
mean           4.410952
std            3.408814
min            0.010000
25%            2.180000
50%            3.470000
75%            5.840000
max          107.320000
Name: sell_price, dtype: object
```

we have minimum value 0.010 and maximum 107.32

In [ ]:
```
sns.distplot(sell_price.sell_price)
plt.title("sell price distribution")
plt.show()
```



In [ ]:
```python
#calculating 0-100th percentile to find a the correct percentile value for removal of outliers
for i in range(0,100,10):
    var =sell_price.sell_price.values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print ("100 percentile value is ",var[-1])
```

```
0 percentile value is 0.01
10 percentile value is 1.24
20 percentile value is 1.97
30 percentile value is 2.48
40 percentile value is 2.88
50 percentile value is 3.47
60 percentile value is 3.98
70 percentile value is 4.98
80 percentile value is 6.27
90 percentile value is 8.64
100 percentile value is  107.32
```

In [ ]:
```python
#looking into 90-100th percentile
for i in range(90, 100, 1):
    var =sell_price.sell_price.values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print ("100 percentile value is ",var[-1])
```

```
90 percentile value is 8.64
91 percentile value is 8.97
92 percentile value is 9.47
93 percentile value is 9.92
94 percentile value is 9.98
95 percentile value is 10.98
96 percentile value is 11.88
97 percentile value is 12.87
98 percentile value is 14.48
99 percentile value is 17.92
100 percentile value is  107.32
```

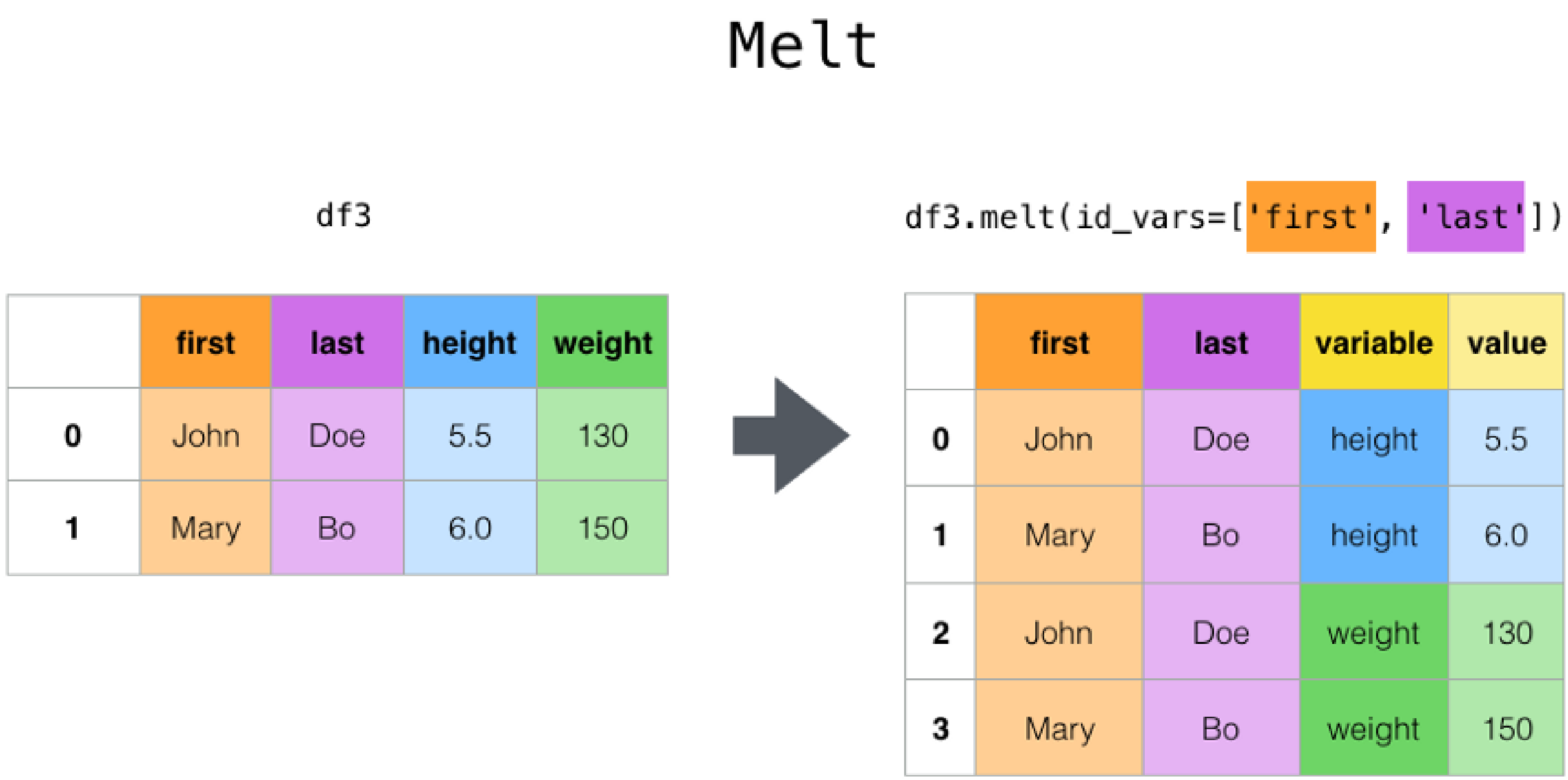
We have a outlier at 100th percentile, so we can remove it

5 Merge all the data

→ first down cast all the dataframe to reduce the size of our data.
→ melt the sell_train eval.
→ merge the all 3 dataframe sell price, sell train eval and calendar.

5.0.0.1 What is melting

Melting : It is used to change the DataFrame format from wide to long.



5.0.0.2 Why melting is required here?

In our case, there is no unique column which can used to merge sell train eval with sell price and caledar. So here, we will change all columns d_1 to d_1941 to row which will be treat as one column. This will give us a unique column

In [2]:
```
#https://www.kaggle.com/anshuls235/time-series-forecasting-eda-fe-modelling/notebook#highlight%20hl-lexer_wrapper

from tqdm import tqdm


def downcast(df):
    cols = df.dtypes.index.tolist()
    types = df.dtypes.values.tolist()
    for i, t in tqdm(enumerate(types)):
        if 'int' in str(t):
            if df[cols[i]].min() > np.iinfo(
                    np.int8).min and df[cols[i]].max() < np.iinfo(np.int8).max:
                df[cols[i]] = df[cols[i]].astype(np.int8)
            elif df[cols[i]].min() > np.iinfo(
                    np.int16).min and df[cols[i]].max() < np.iinfo(
                        np.int16).max:
                df[cols[i]] = df[cols[i]].astype(np.int16)
            elif df[cols[i]].min() > np.iinfo(
                    np.int32).min and df[cols[i]].max() < np.iinfo(
                        np.int32).max:
                df[cols[i]] = df[cols[i]].astype(np.int32)
            else:
                df[cols[i]] = df[cols[i]].astype(np.int64)
        elif 'float' in str(t):
            if df[cols[i]].min() > np.finfo(
                    np.float16).min and df[cols[i]].max() < np.finfo(
                        np.float16).max:
                df[cols[i]] = df[cols[i]].astype(np.float16)
            elif df[cols[i]].min() > np.finfo(
                    np.float32).min and df[cols[i]].max() < np.finfo(
                        np.float32).max:
                df[cols[i]] = df[cols[i]].astype(np.float32)
            else:
                df[cols[i]] = df[cols[i]].astype(np.float64)
        elif t == np.object:
            if cols[i] == 'date':
                df[cols[i]] = pd.to_datetime(df[cols[i]], format='%Y-%m-%d')
            else:
                df[cols[i]] = df[cols[i]].astype('category')
    return df


# sales_train_eval = downcast(sales_train_eval)
# sell_price = downcast(sell_price)
# calendar = downcast(calendar)
```

In [67]:
```
df = pd.melt(
    sales_train_eval,
    id_vars=['id', 'item_id', 'dept_id', 'cat_id', 'store_id', 'state_id'],
    var_name='d',
    value_name='demand').dropna()
df = pd.merge(df, calendar, on='d', how='left')
df = pd.merge(df,
              sell_price,
              on=['store_id', 'item_id', 'wm_yr_wk'],
              how='left')
# df.to_csv("final_validation.csv",  index = False) #save the data
# df.to_pickle("final_pickle.pkl")
df.head()
```

Out[67]:

	id	item_id	dept_id	cat_id	store_id	state_id	d	demand	date	wm_yr_wk	...	event_r
0	HOBBIES_1_001_CA_1_validation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0	2011-01-29	11101	...	
1	HOBBIES_1_002_CA_1_validation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0	2011-01-29	11101	...	
2	HOBBIES_1_003_CA_1_validation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0	2011-01-29	11101	...	
3	HOBBIES_1_004_CA_1_validation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0	2011-01-29	11101	...	
4	HOBBIES_1_005_CA_1_validation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0	2011-01-29	11101	...	

5 rows × 24 columns

- we are creating two new columns , Demand and D with melting the data.
- D column contains data of days from d1 to d1941.
- Demand contains number of product sell on that day.
- So total 30490*1941 = 59,181,090 rows will there in our dataset.

In [4]:
```
import pickle
df = pd.read_pickle("final_pickle.pkl")
df.head()
```

Out[4]:

	id	item_id	dept_id	cat_id	store_id	state_id	d	demand	date	wm_yr_wk	...	event_
0	HOBBIES_1_001_CA_1_evaluation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0	2011-01-29	11101	...	
1	HOBBIES_1_002_CA_1_evaluation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0	2011-01-29	11101	...	
2	HOBBIES_1_003_CA_1_evaluation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0	2011-01-29	11101	...	
3	HOBBIES_1_004_CA_1_evaluation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0	2011-01-29	11101	...	
4	HOBBIES_1_005_CA_1_evaluation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	CA	d_1	0	2011-01-29	11101	...	

5 rows × 24 columns

In [6]:
```
#What is the size of the data  final data
print(f'final file has {df.shape[0]} rows and {df.shape[1]} columns.')
```

```
final file has 59181090 rows and 24 columns.
```

In [69]:
```
missing_values(df)
```

Out[69]:

	column_name	non_misisng_count	missing_count	percent_missing
id	id	58327370	0	0.000000
event_3	event_3	58327370	0	0.000000
snap_WI	snap_WI	58327370	0	0.000000
snap_TX	snap_TX	58327370	0	0.000000
snap_CA	snap_CA	58327370	0	0.000000
year	year	58327370	0	0.000000
month	month	58327370	0	0.000000
event_type_3	event_type_3	58327370	0	0.000000
weekday	weekday	58327370	0	0.000000
wday	wday	58327370	0	0.000000
date	date	58327370	0	0.000000
demand	demand	58327370	0	0.000000
d	d	58327370	0	0.000000
state_id	state_id	58327370	0	0.000000
store_id	store_id	58327370	0	0.000000
cat_id	cat_id	58327370	0	0.000000
dept_id	dept_id	58327370	0	0.000000
item_id	item_id	58327370	0	0.000000
wm_yr_wk	wm_yr_wk	58327370	0	0.000000
sell_price	sell_price	46027957	12299413	21.086864
event_name_1	event_name_1	4695460	53631910	91.949817
event_type_1	event_type_1	4695460	53631910	91.949817
event_name_2	event_name_2	121960	58205410	99.790904
event_type_2	event_type_2	121960	58205410	99.790904

- **sell price** has 20% missing values
- **event name 1 and event type 1 has** has 91.85% missing values
- **event name 2 and event type 2 has** has 99.79% missing values

In [5]:

```
#filling null values in event_3 and event_type_3
df["event_3"] = df['event_3'].cat.add_categories("no_event")
df["event_3"] = df['event_3'].fillna("no_event")
df["event_type_3"] = df['event_type_3'].cat.add_categories("no_event")
df["event_type_3"] = df['event_type_3'].fillna("no_event")

#drop the values
df = df.drop(["event_name_1", "event_name_2", "event_type_2" , "event_type_1"], axis=1)
```

In [ ]:

```
#demand column
df.demand.describe().apply(lambda x: format(x, 'f'))
```

Out[15]:

```
count    59181090.000000
mean            1.130888
std             3.870038
min             0.000000
25%             0.000000
50%             0.000000
75%             1.000000
max           763.000000
Name: demand, dtype: object
```

Demand has minimum value 0 which means no sell and maximum 763 sell.

In [ ]:
```
#checking demand values
#calculating 0-100th percentile to find a the correct percentile value for removal of outliers
for i in range(0,100,10):
    var =df.demand.values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print ("100 percentile value is ",var[-1])
```

```
0 percentile value is 0
10 percentile value is 0
20 percentile value is 0
30 percentile value is 0
40 percentile value is 0
50 percentile value is 0
60 percentile value is 0
70 percentile value is 1
80 percentile value is 1
90 percentile value is 3
100 percentile value is  763
```

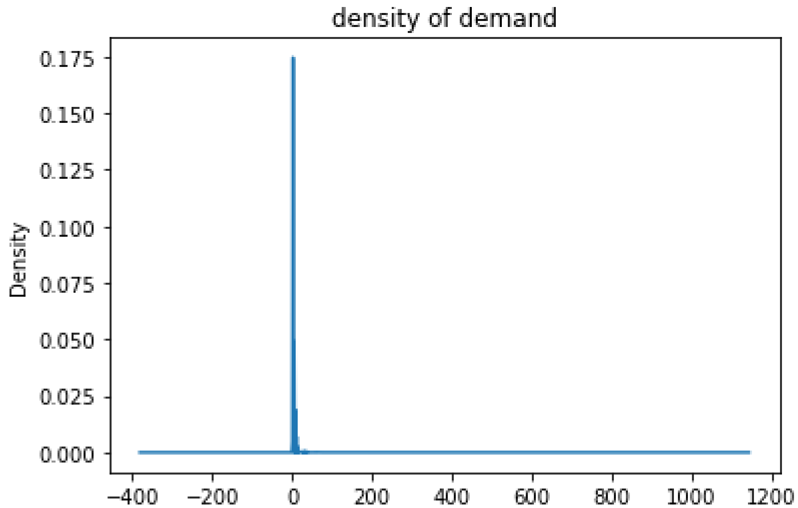
Demand has 1 value till 90 % and at 100% it has 763 values

In [ ]:
```

#calculating 0-100th percentile to find a the correct percentile value for removal of outliers
for i in range(90, 100):
    var =df.demand.values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print ("100 percentile value is ",var[-1])
```

```
90 percentile value is 3
91 percentile value is 3
92 percentile value is 4
93 percentile value is 4
94 percentile value is 5
95 percentile value is 5
96 percentile value is 6
97 percentile value is 8
98 percentile value is 10
99 percentile value is 15
100 percentile value is  763
```

Demand has 15 value till 99 % and at 100% it has 763 values

In [ ]:
```
df['demand'].plot(kind='density')
plt.title("density of demand")
plt.show()
```



Density plot of demand is cleary showing we have sudden spike in sell between 0 to 15

In [3]:
```
#sell price column
df.sell_price.describe().apply(lambda x: format(x, 'f'))
```

Out[3]:
```
count    46881677.000000
mean            4.409435
std             3.406079
min             0.010000
25%             2.180000
50%             3.470000
75%             5.840000
max           107.300000
Name: sell_price, dtype: object
```

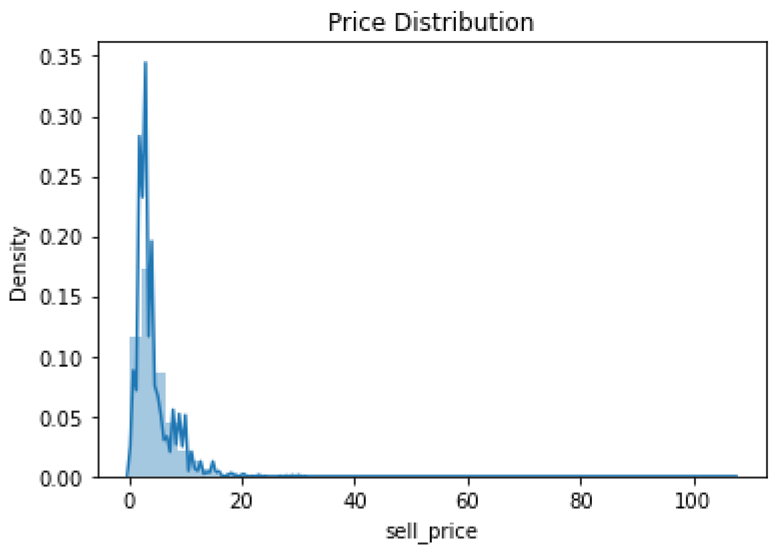
price has minimum value 0.010 and max is 107.312.

In [ ]:
```
#checking sell values
#calculating 0-100th percentile to find a the correct percentile value for removal of outliers
for i in range(0,100,10):
    var =df.sell_price.values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(i,var[int(len(var)*(float(i)/100))]))
print ("100 percentile value is ",var[-1])
```

```
0 percentile value is 0.01000213623046875
10 percentile value is 1.5
20 percentile value is 2.1796875
30 percentile value is 2.8203125
40 percentile value is 3.48046875
50 percentile value is 4.46875
60 percentile value is 5.94140625
70 percentile value is 7.98046875
80 percentile value is nan
90 percentile value is nan
100 percentile value is  nan
```

Price has 7.98 at 70 % and after that it's nan

In [4]:
```
sns.distplot(df.sell_price)
plt.title("Price Distribution")
plt.show()
```



In [ ]:

```
#how many values are there with more than x in sell_price
print("values more than 18 :" , (df[df.sell_price>18]).shape)

print("values more than 40 :" , (df[df.sell_price>40]).shape)

print("values more than 50 :" , (df[df.sell_price>50]).shape)

print("values more than 60 :" , (df[df.sell_price>60]).shape)

print("values more than 70 :" , (df[df.sell_price>70]).shape)

print("values more than 80 :" , (df[df.sell_price>80]).shape)
```

```
values more than 18 : (311786, 22)
values more than 40 : (350, 22)
values more than 50 : (119, 22)
values more than 60 : (112, 22)
values more than 70 : (21, 22)
values more than 80 : (21, 22)
```

we can clearly see, we have very few sells which has more price more than 40

In [ ]:

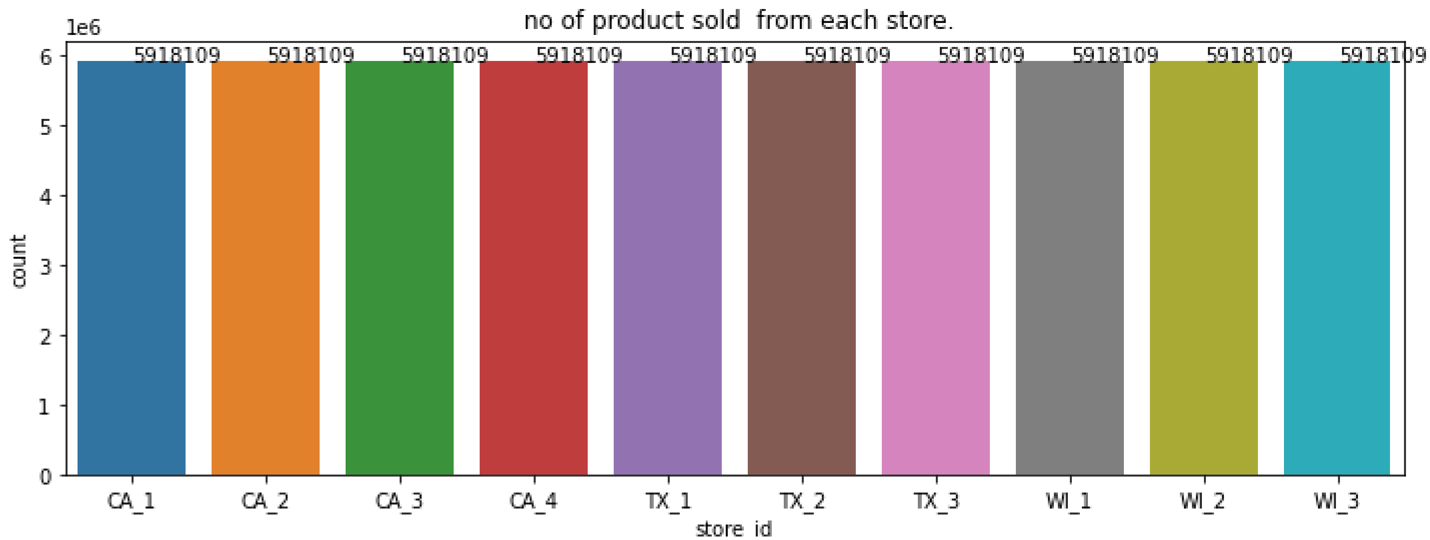
```
#how many store located in each state
plots = sns.countplot(x = df.state_id)
for bar in plots.patches:
    plots.annotate(bar.get_height(),
                   (bar.get_x() + bar.get_width() / 2,
                    bar.get_height()))
plt.title('no of store located in each state')
plt.show()
```

We have 23672436 , 17754327, 17754327 stores data in CA, TX, and WI

In [ ]:

```
##no of unique each store in store_id
print(f' no of unique store : {df.store_id.nunique()}')

plt.figure(figsize=(12, 4))
plots = sns.countplot(x=df.store_id)

for bar in plots.patches:
    plots.annotate(bar.get_height(),
                   (bar.get_x() + bar.get_width() / 2,
                    bar.get_height()))
plt.title(' no of product sold  from each store.')
plt.show()
```

```
no of unique store : 10
```

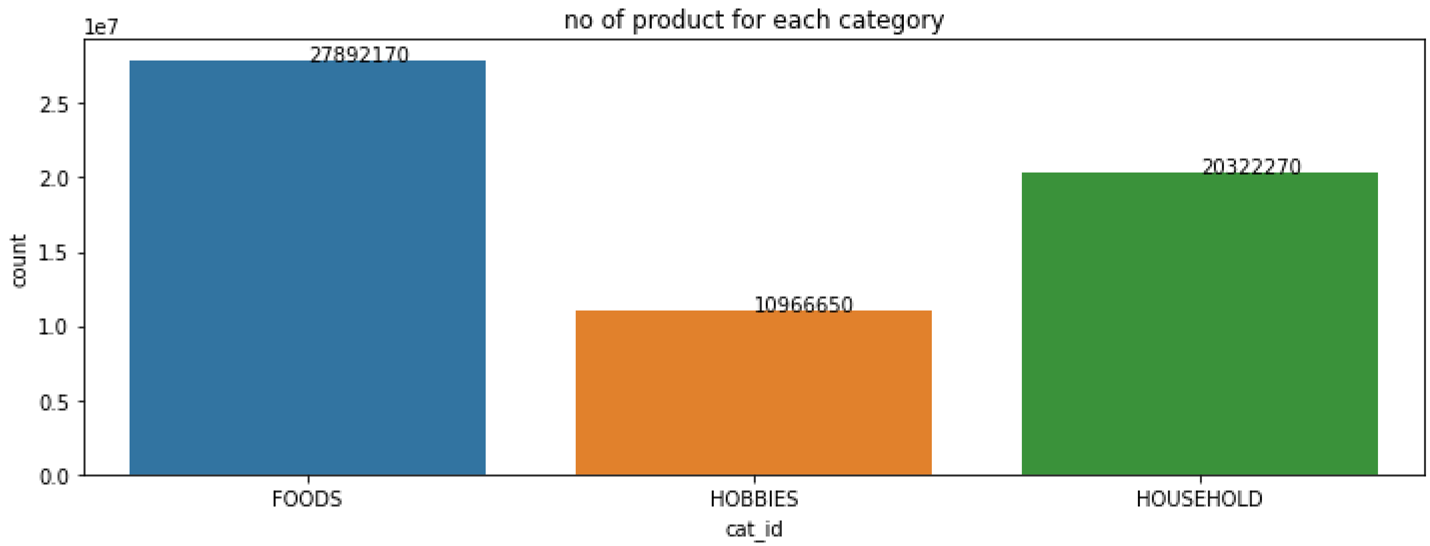


We have total 10 store in 3 states in which 4 are in CA, 3 are in TX and 3 are in WI

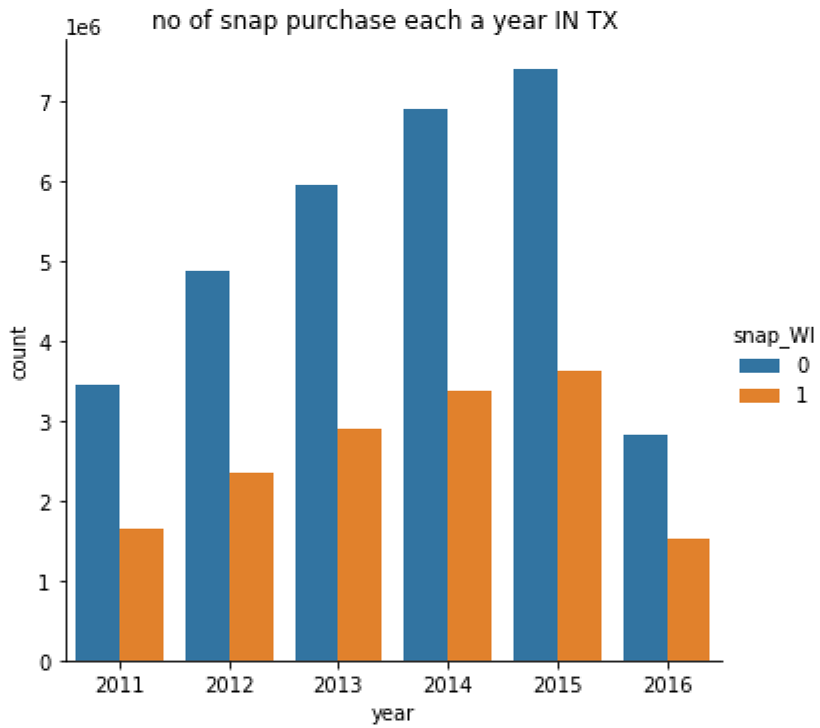
In [ ]:

```
print(f' no of unique values in category : {df.cat_id.nunique()}')

plt.figure(figsize=(12, 4))
plots = sns.countplot(x=df.cat_id)

for bar in plots.patches:
    plots.annotate(bar.get_height(),
                   (bar.get_x() + bar.get_width() / 2,
                    bar.get_height()))
plt.title('no of product for each category')
plt.show()
```

```
no of unique values in category : 3
```



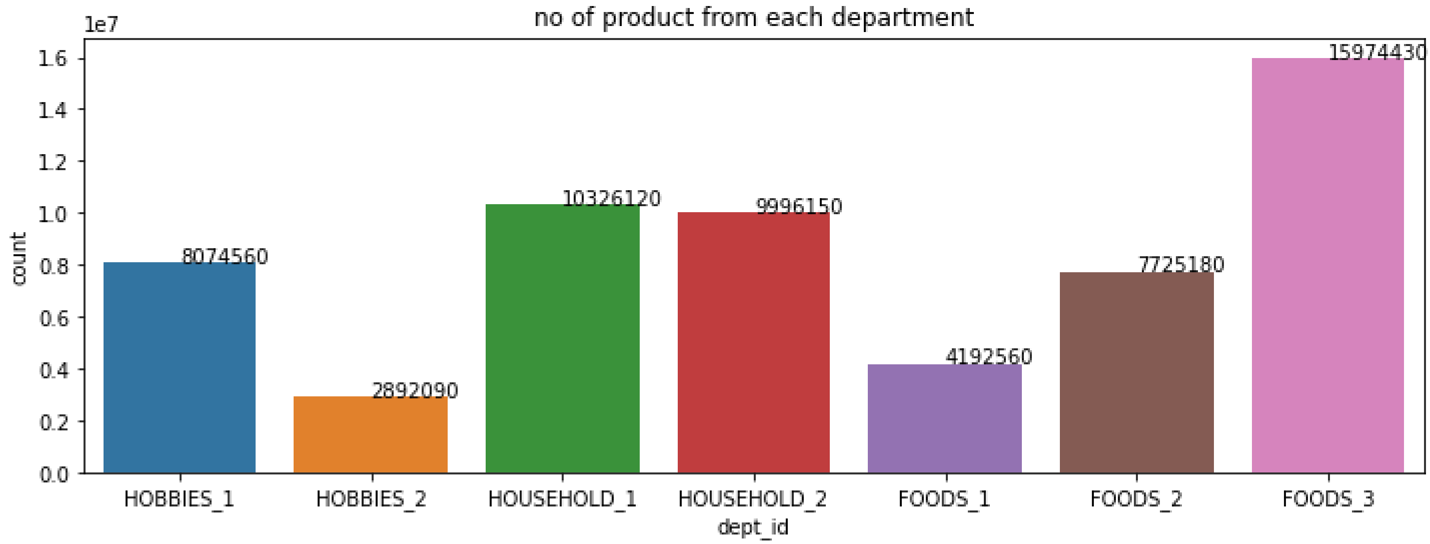
In [ ]:
```
#no of snap purchase each a year TX
plots = sns.catplot(x='year', data=df, kind = 'count', hue = 'snap_WI')

plt.title("no of snap purchase each a year IN TX  ")

plt.show()
```



We can see in each year we have very few days when snap was allowed.

In [ ]:
```
#no of unique department:
print(f'no of unique department: { df.dept_id.nunique()}')
plt.figure(figsize=(12, 4))
plots = sns.countplot(x=df.dept_id)

for bar in plots.patches:
    plots.annotate(bar.get_height(),
                   (bar.get_x() + bar.get_width() / 2,
                    bar.get_height()))
plt.title('no of product from each department')
plt.show()
```

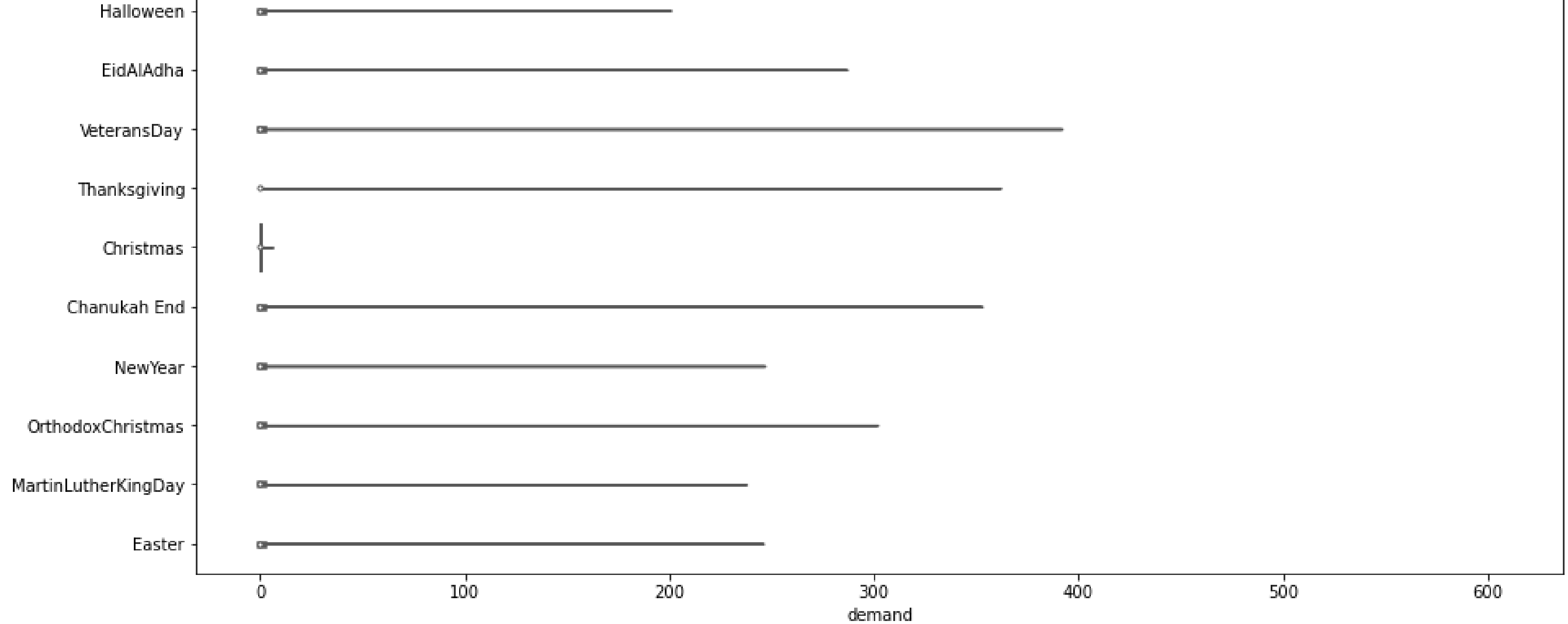
```
no of unique department: 7
```



6 Univariate Analysis

6.1 Demand With Event_name_1

In [31]:
```
plt.figure(figsize=(15, 20))
sns.violinplot(y = df['event_name_1'], x = df.demand)
plt.title(" demand on event_name 1", loc='left')
plt.show()
```



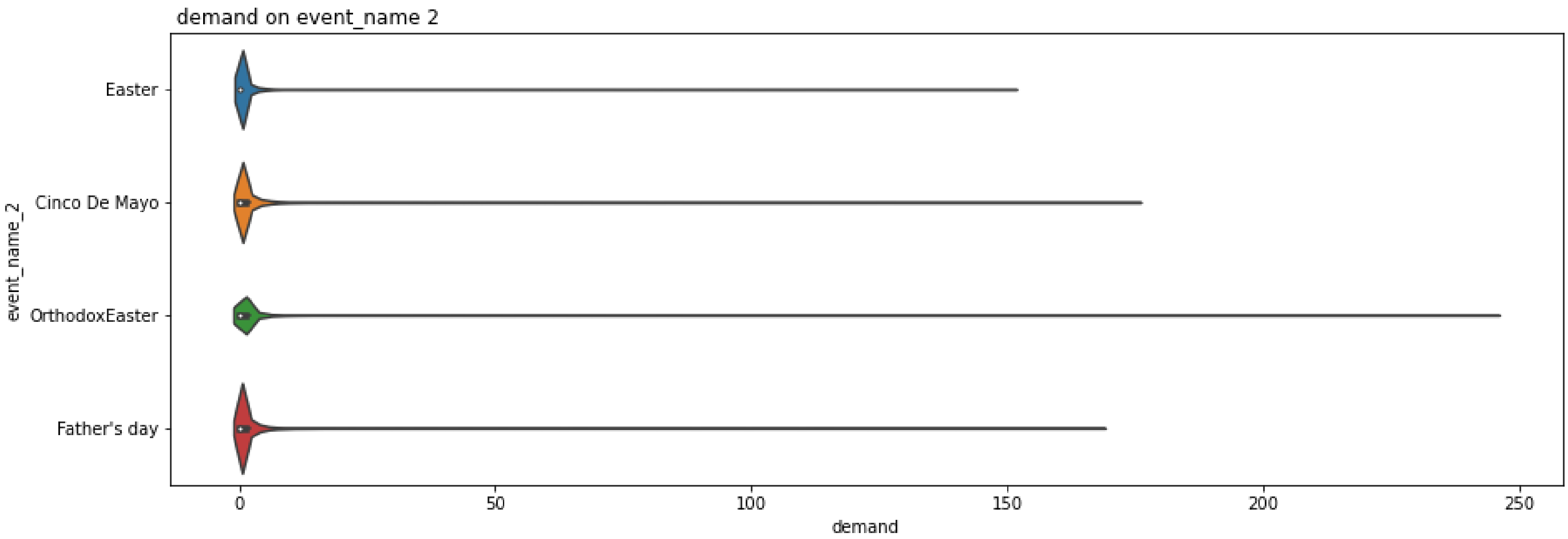
Observation

1. NBA final start and labour day has more demand as compare to other events.
2. After that Chanuakh end , thanksgiving and Indepndece has more sell.

6.2 Demand With Event_name_2

In [54]:

```
plt.figure(figsize=(15, 5))
sns.violinplot(y = df['event_name_2'], x = df.demand)
plt.title(" demand on event_name 2", loc='left')
plt.show()
```

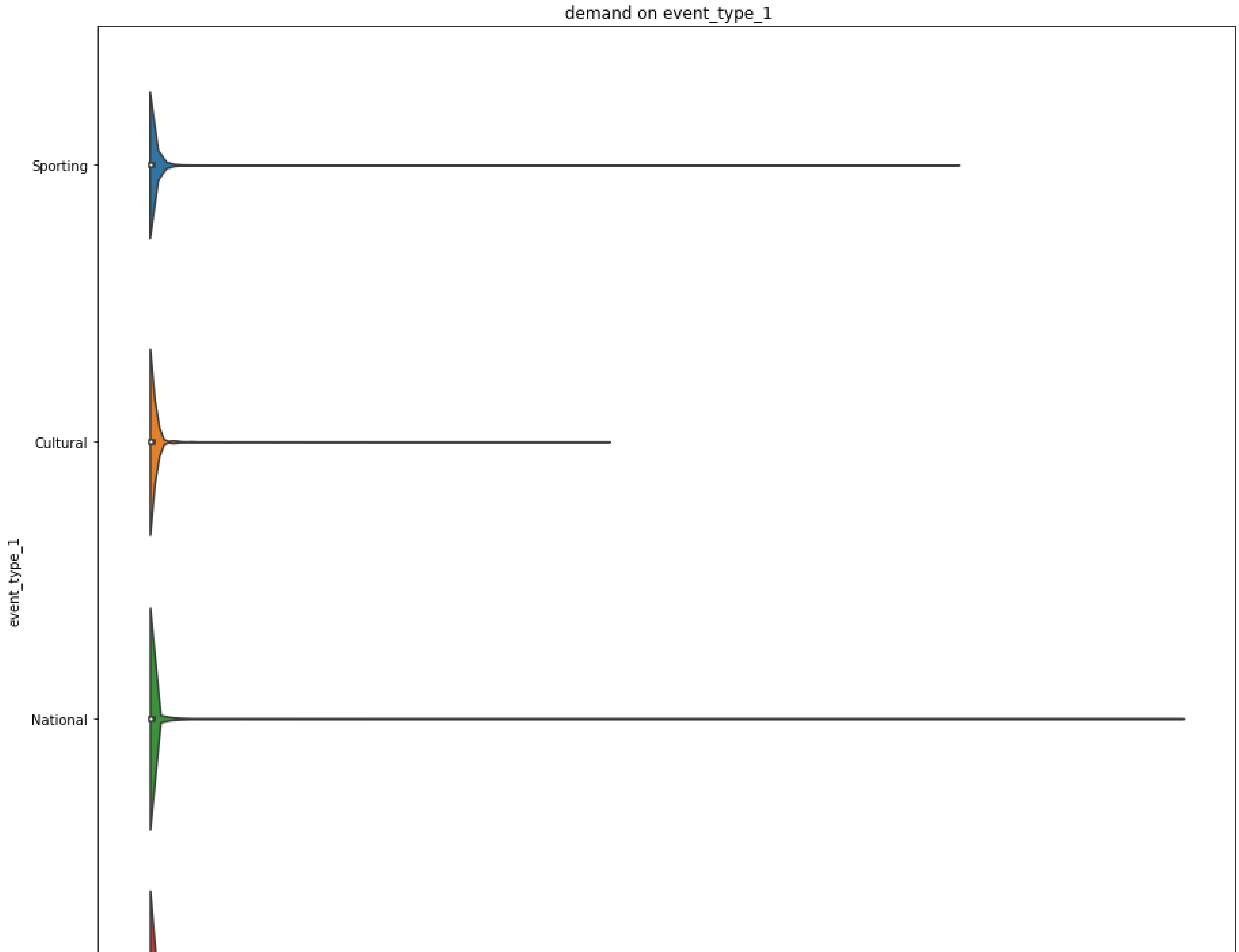


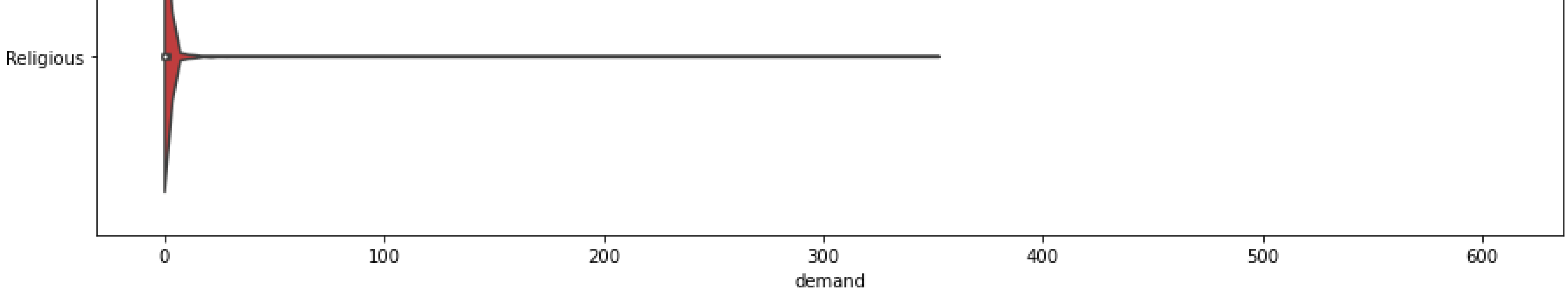
Observation

1. Orthadox event has more demand, after that Father's day, Easter and Cinco De Mayo

6.3 Demand With Event_Type_1

In [51]:
```
plt.figure(figsize=(15, 15))
sns.violinplot(y = df['event_type_1'], x = df.demand
              )
plt.title(" demand on event_type_1")
plt.show()
```





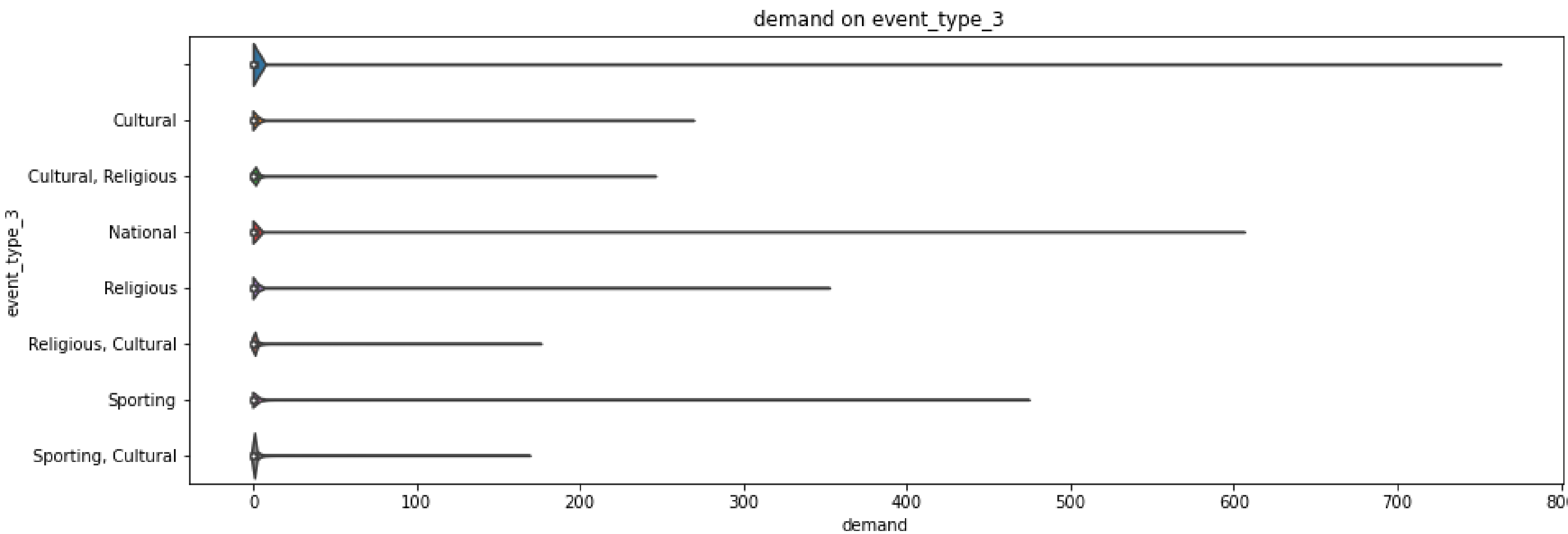
Observation

1. National Event has more sell, then sporting , religious and cultural.

6.4 Demand With Event_Type_2

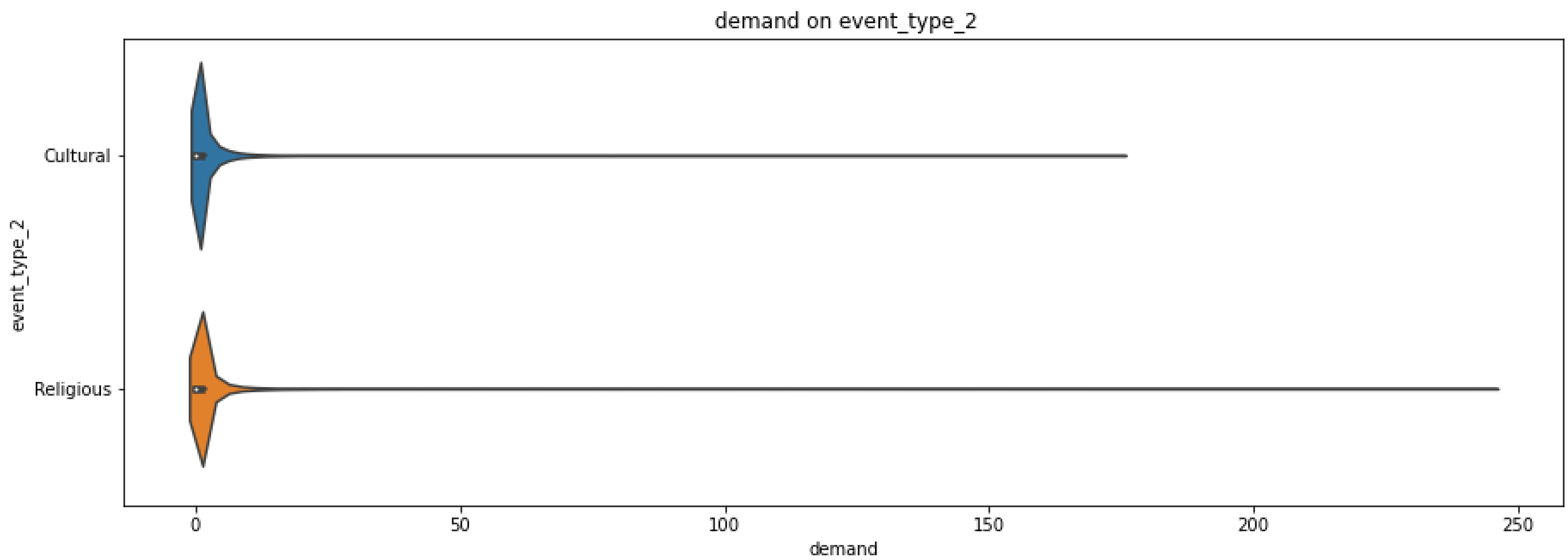
In [70]:

```
plt.figure(figsize=(15, 5))
sns.violinplot(y = df['event_type_3'], x = df.demand)
plt.title(" demand on event_type_3")
plt.show()
```



In [50]:

```
plt.figure(figsize=(15, 5))
sns.violinplot(y = df['event_type_2'], x = df.demand)
plt.title(" demand on event_type_2")
plt.show()
```

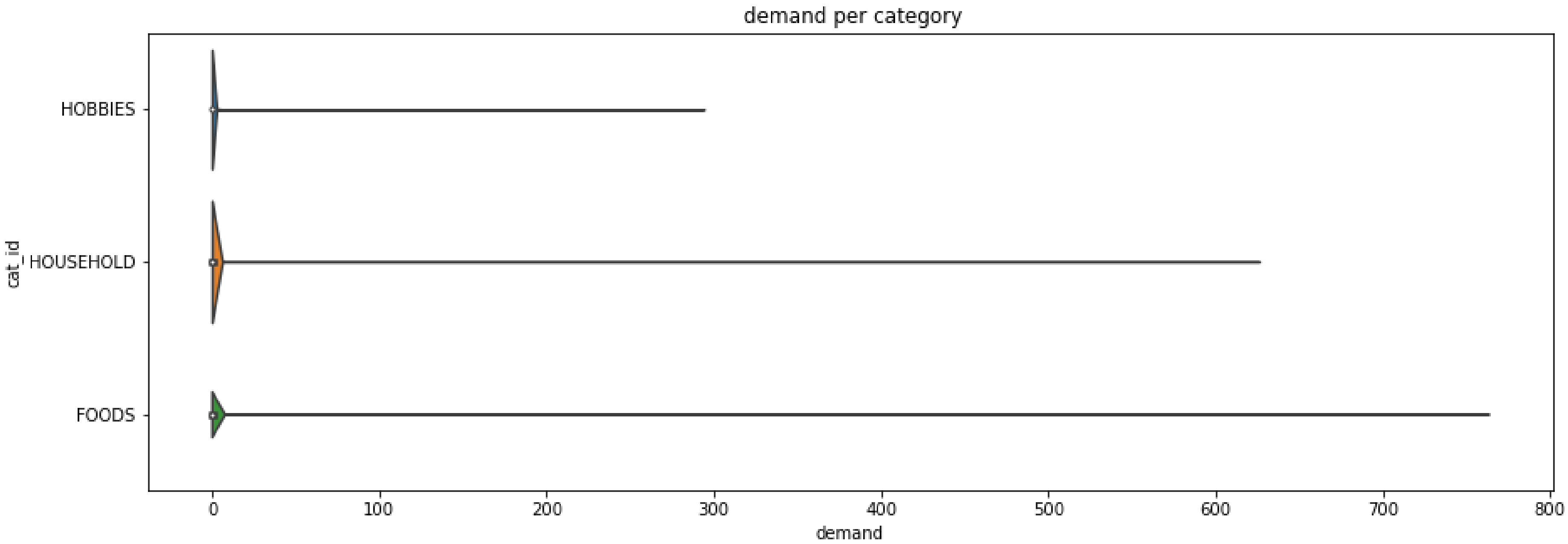


Observation

1. Religious event has more demand as compare to Cultural event.

6.5 Demand per Category

In [52]:
```
plt.figure(figsize=(15, 5))
sns.violinplot(y = df['cat_id'], x = df.demand)
plt.title(" demand per category")
plt.show()
```

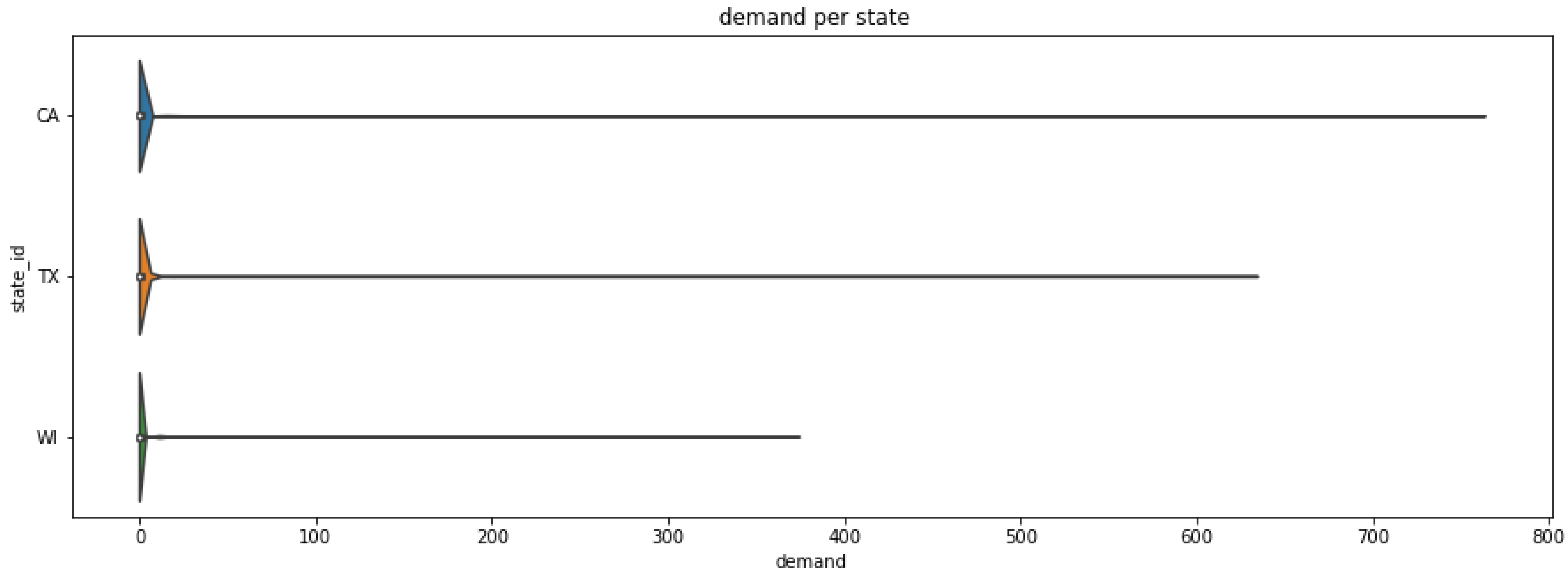


Observation

1. Foods category has more demand then Householdes , and Hobbies .

6.6 Demand per State

In [53]:
```
plt.figure(figsize=(15, 5))
sns.violinplot(y = df['state_id'], x = df.demand)
plt.title(" demand per state")
plt.show()
```

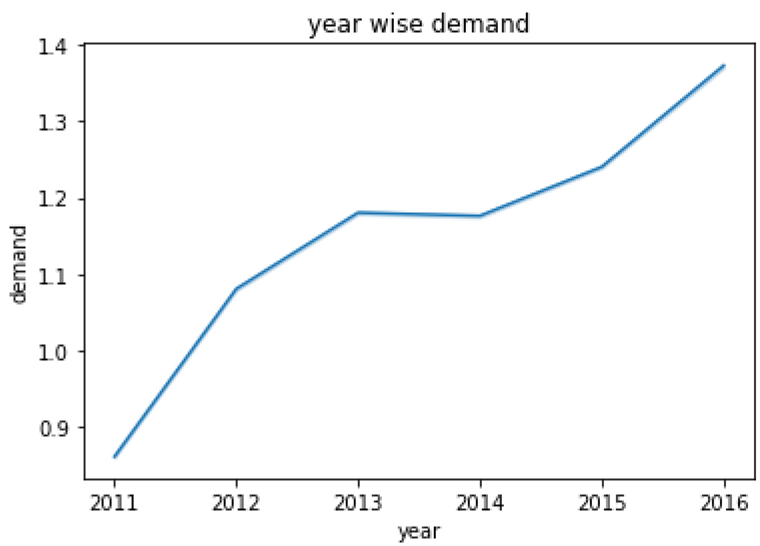


Observation

1. California has more demand , then Texas and Wisconsin.

6.7 Demand Per Year

In [35]:
```
#year wise demand
sns.lineplot(x=df['year'] , y=df['demand'])
plt.title("year wise demand")
plt.show()
```

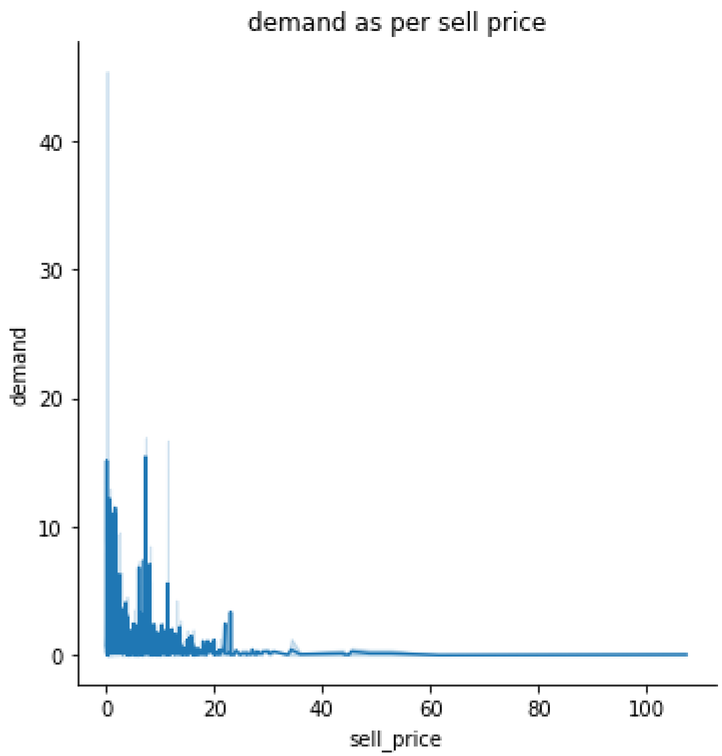


Observation

1. Demand is incresing from 2011 to 2016.
2. Increment is non-linear.
3. There is some low trend from mid 2013 - 2014.

6.8 Demand with Sell Price

In [41]:
```
#sell price trend in demand

sns.relplot(
    data=df, x="sell_price", y="demand",
    kind="line",
)
plt.title("demand as per sell price")
plt.show()
```



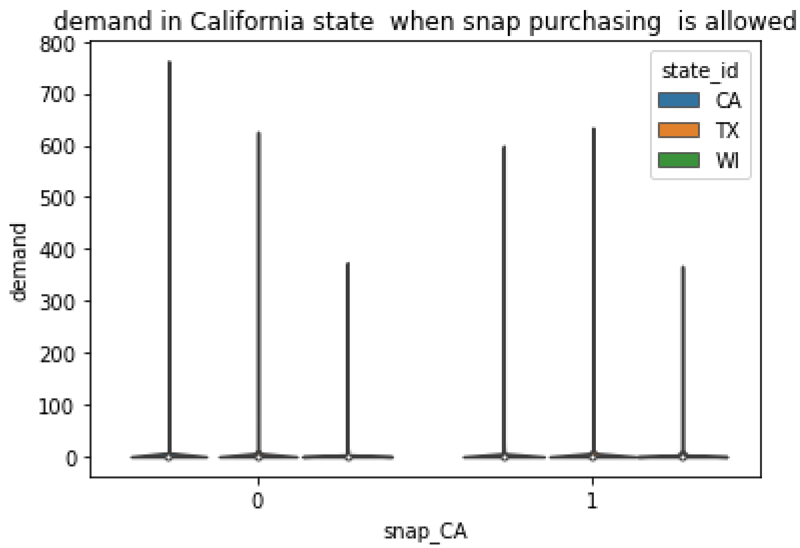
Observation

1. There is more demand when price is low.
2. When product value is less than 20 $ it has more demand.
3. As product's price increased more than 20 $, demand of the product has decreased significantly.

6.9 Demand when Snap purchased allowed in California

6.9 Demand when Snap purchased allowed in California

In [55]:
```
#snap_ca and demand
sns.violinplot(x=df['snap_CA'] , y=df['demand'], hue=df['state_id'])
plt.title("demand in California state  when snap purchasing  is allowed")
plt.show()
```

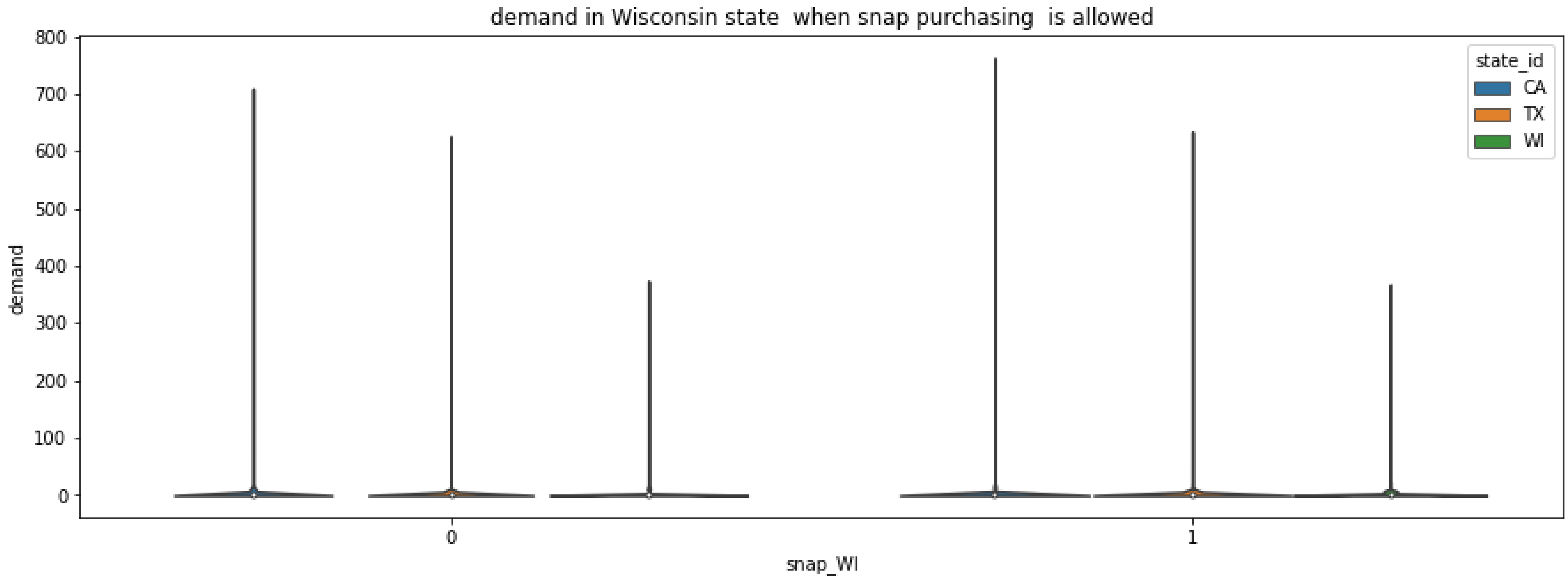


Observation

1. There is no significant impact of demand in Californnia when snap purchasing was allowed

6.10 Demand when Snap purchased allowed in Wisconsin

In [56]:
```
# snap_wi and demand
plt.figure(figsize=(15, 5))
sns.violinplot(x=df['snap_WI'] , y=df['demand'], hue=df['state_id'])
plt.title("demand in Wisconsin state  when snap purchasing  is allowed")
plt.show()
```

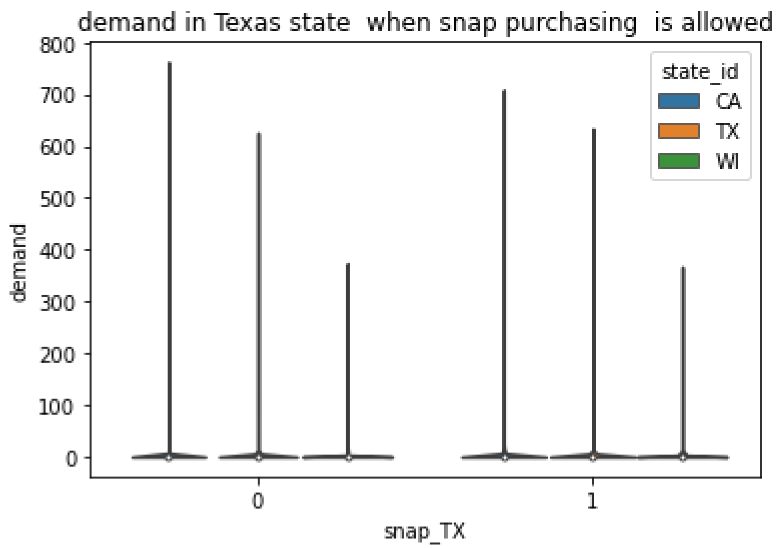


Observation

1. There is no significant impact of demand in Wisconsin when snap purchasing was allowed

6.11 Demand when Snap purchased allowed in Texas

In [57]:
```
#snap_ta and demand
sns.violinplot(x=df['snap_TX'] , y=df['demand'], hue=df['state_id'])
plt.title("demand in Texas state  when snap purchasing  is allowed")
plt.show()
```

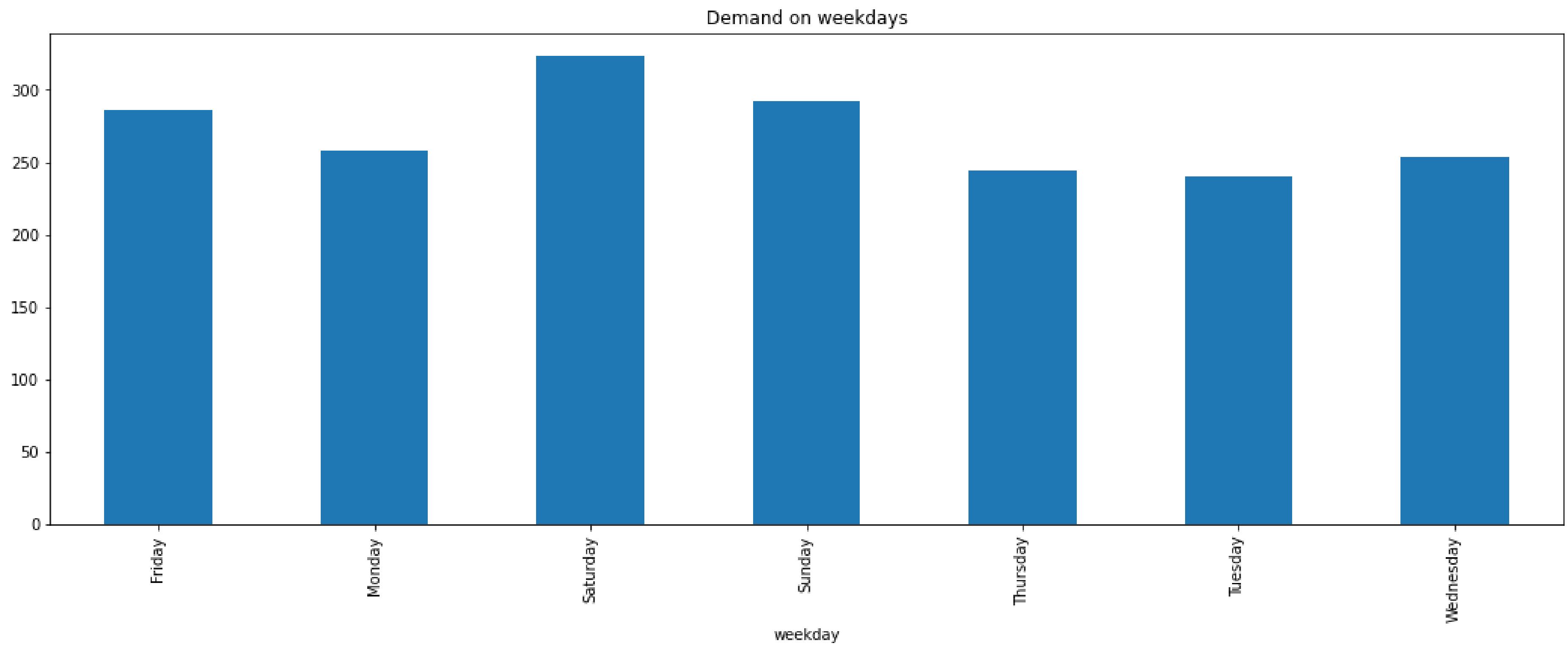


Observation

1. There is no significant impact of demand in Texas when snap purchasing was allowed

6.12 Demand on Weekday

In [191]:
```
week = df.groupby("weekday")
week["demand"].nunique().plot(kind ="bar", figsize = (18,6))
plt.title("Demand on weekdays")
plt.show()
```

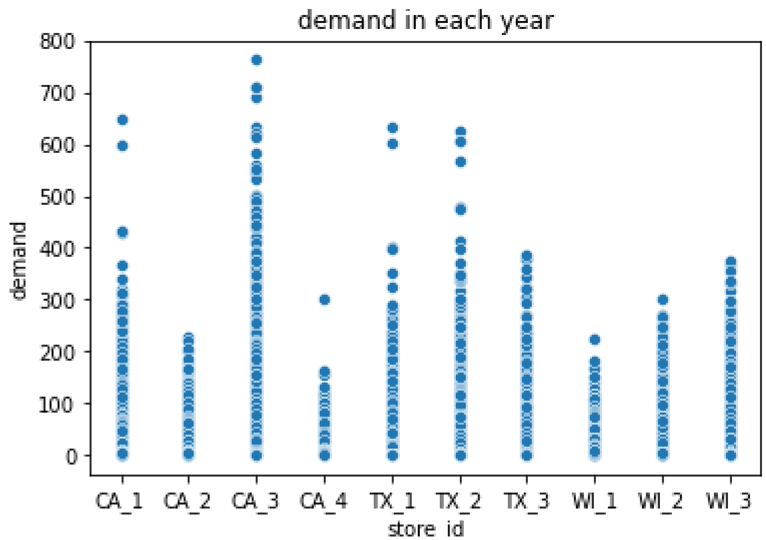


Observation

1. We have more demand on Saturday and Sunday as compare to other days.

6.13 Demand By each Store

In [ ]:
```
#demand by each store
sns.violinplot(x = df.store_id, y = df.demand, hue = df.state_id)
plt.title("demand in each year")
plt.show()
```



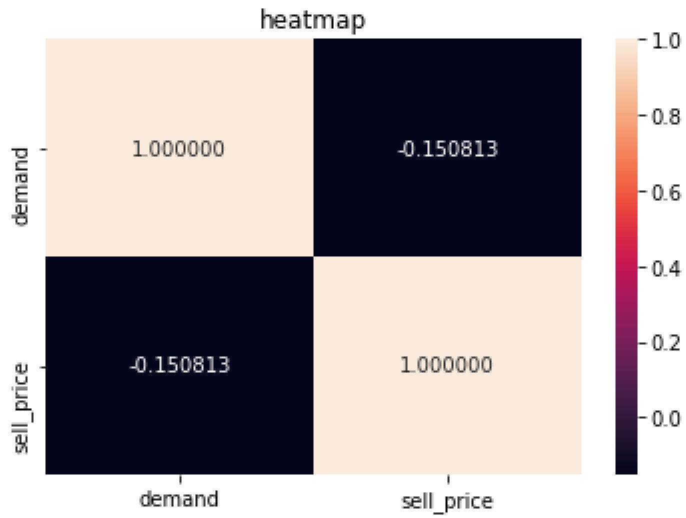
Observation

1. Stores which are located in CA and TX has more demand as compare to WI

6.14 Demand ad Sell price Relationship

In [ ]:

```
#sell price and demand co releation p
d = df[['demand', 'sell_price']]
d = d.corr()
sns.heatmap(d, annot = True, fmt = 'f')
plt.title("heatmap")
plt.show()
```



Observation

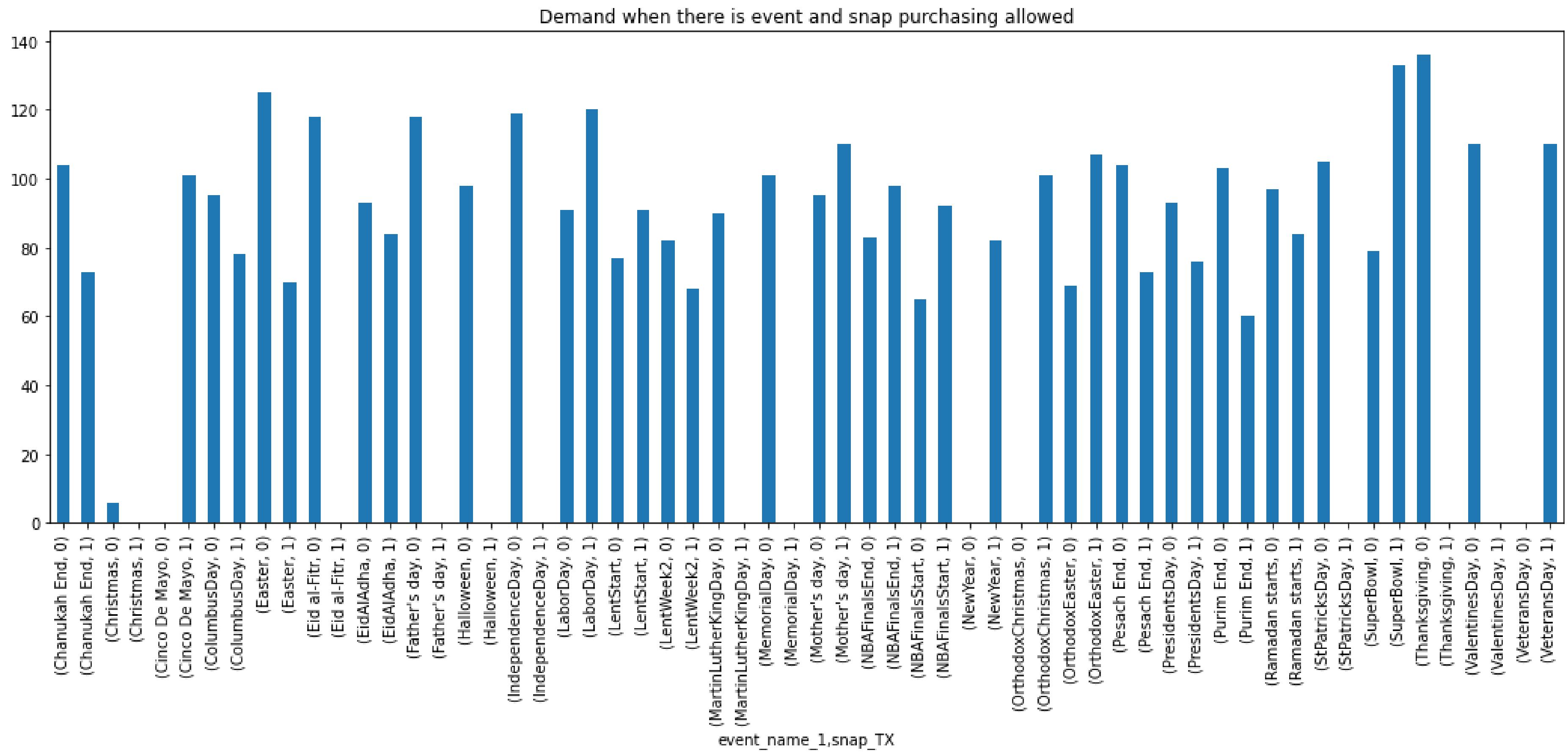
1.There is negative relationship betweenn demand and sell price.

2.We can state that if we have price high then demand will be low , vice versa

7 Bivariate Analysis

7.1 Demand with event_name_1 and snap purchasing in Texas

In [148]:
```
Texas = df.groupby(['event_name_1', "snap_TX"])
Texas["demand"].nunique().plot(kind = "bar", figsize=(18,6))
plt.title("Demand when there is event and snap purchasing allowed")
plt.show()
```

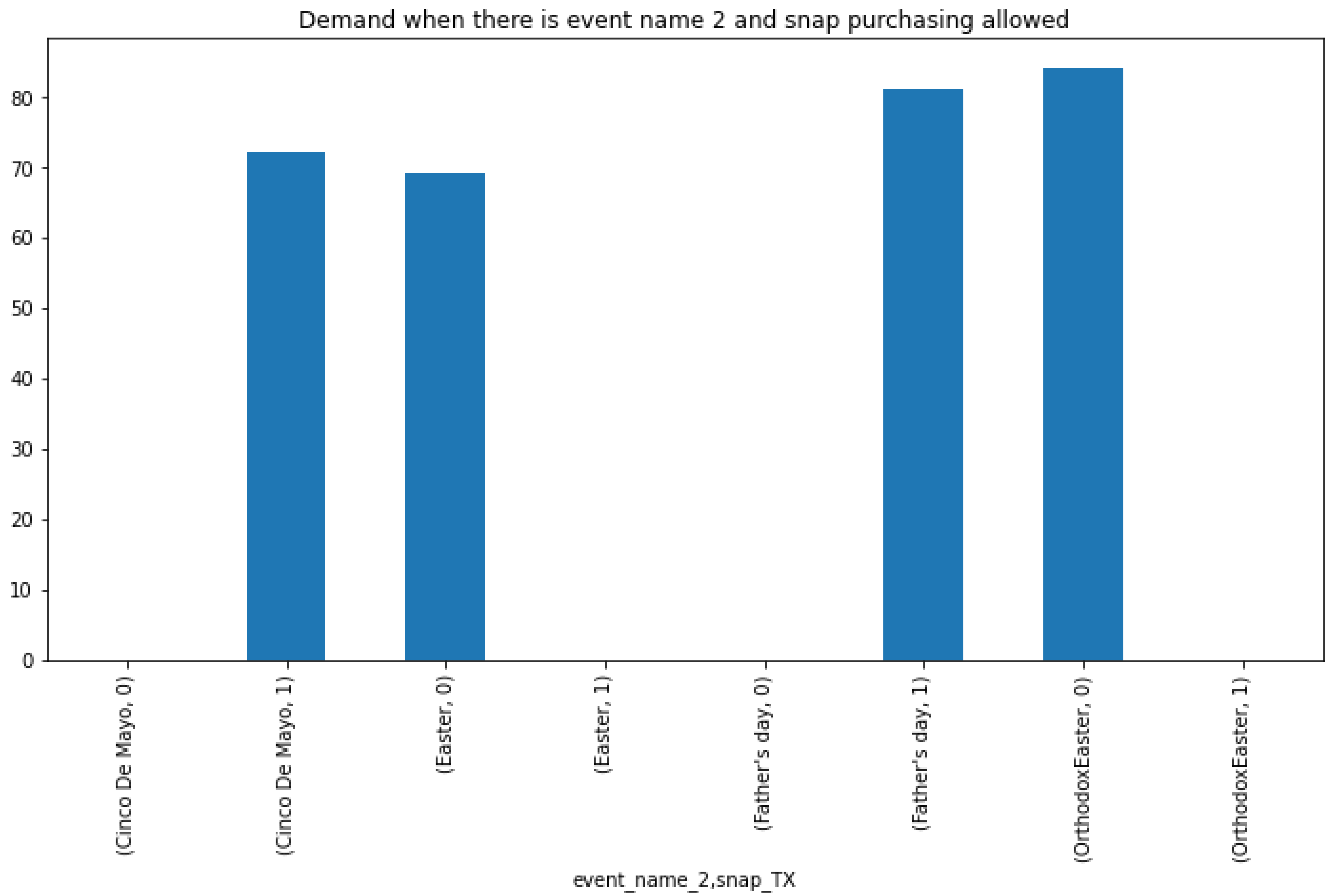


Observation

1. Labour day and Superbowl, have demand on snap purchasing

7.2 Demand with event_name_2 and snap purchasing in Texas

In [150]:
```
Texas = df.groupby(["event_name_2" ,"snap_TX",])
Texas["demand"].nunique().plot(kind = "bar", figsize=(12,6))
plt.title("Demand when there is event name 2 and snap purchasing allowed")
plt.show()
```

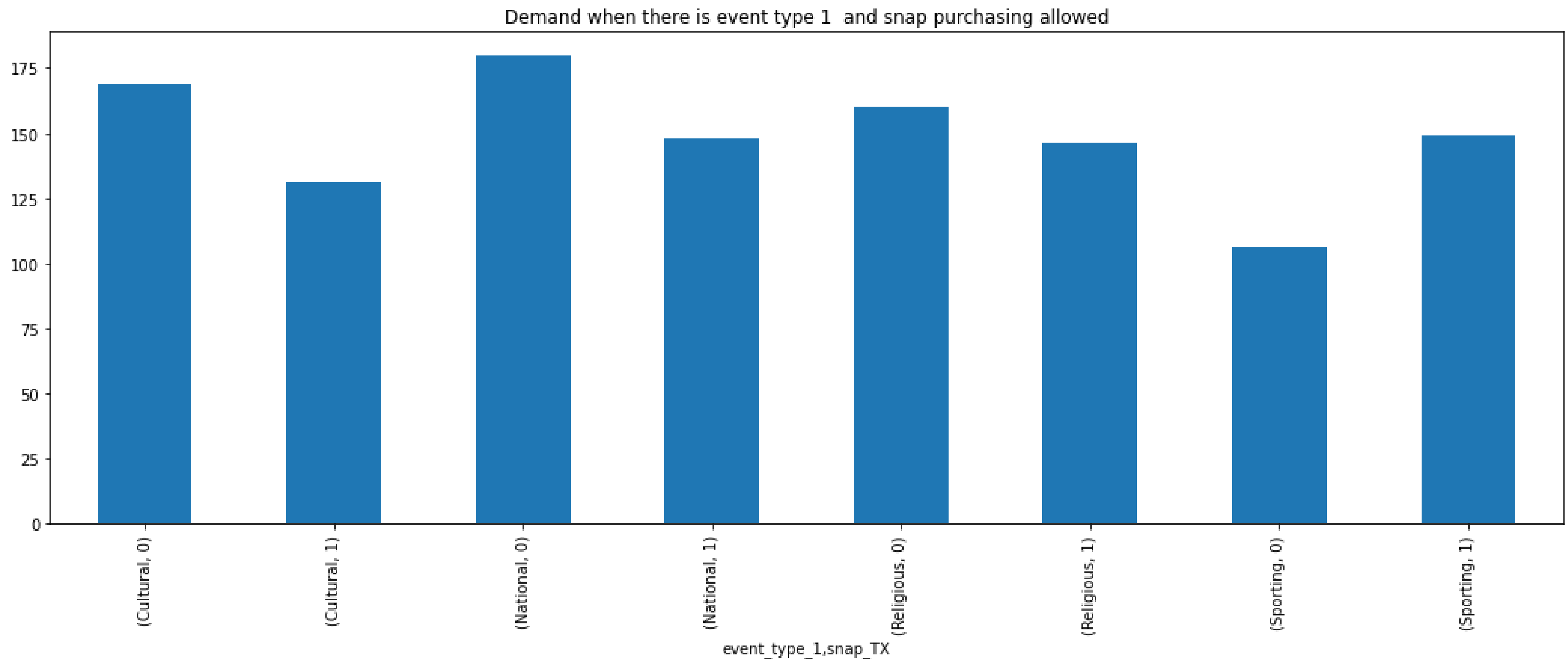


Observation

1. We can definetly see by Cinco De Mayo and father's day, if we have event and snap purchasing is allowed then there is high sell.
2. Orthadox Easter has highest sell even without any snap purchasing

7.3 Demand with even_type_1 and snap purchasing in Texas

In [151]:
```
Texas = df.groupby(["event_type_1" ,"snap_TX",])
Texas["demand"].nunique().plot(kind = "bar", figsize=(18,6))
plt.title("Demand when there is event type 1  and snap purchasing allowed")
plt.show()
```

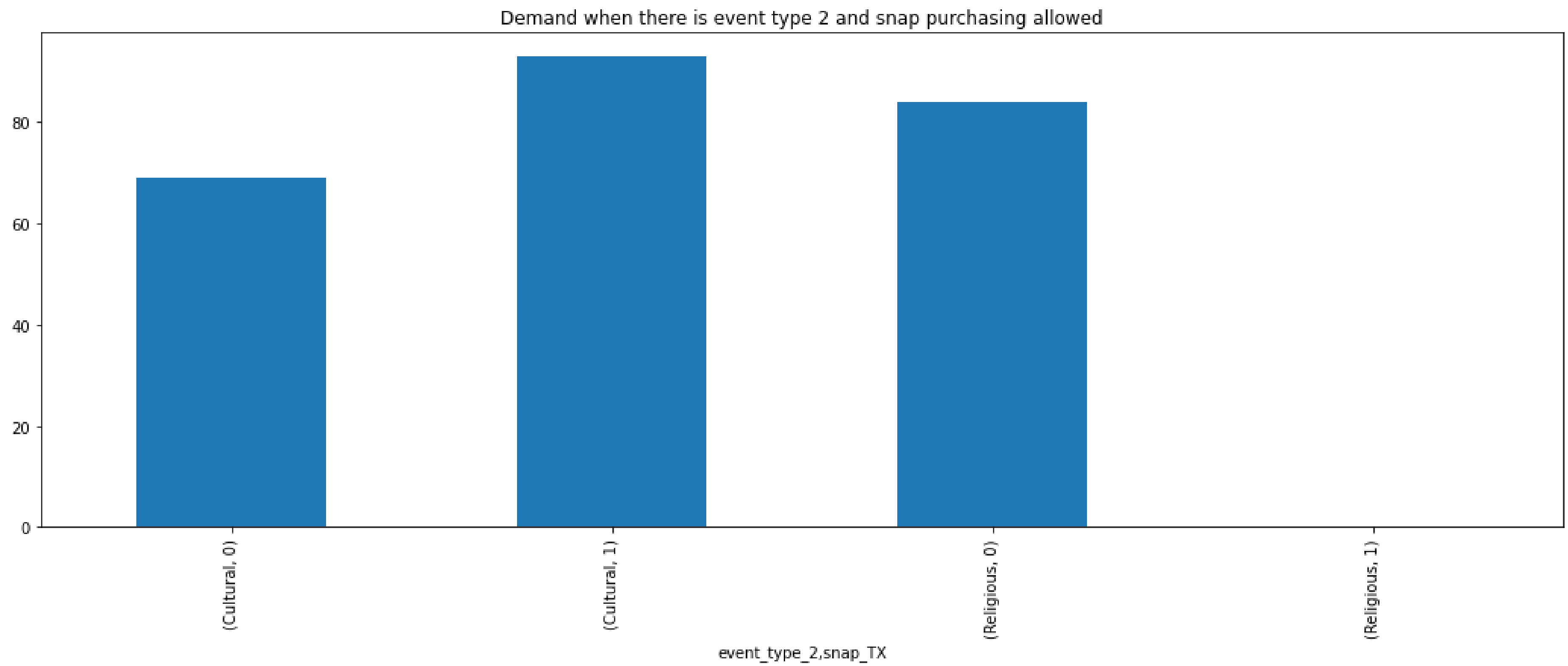


Observation

1. We can see even the non-snap purchasing days has more demand .

7.4 Demand with even_type_2 and snap purchasing in Texas

In [152]:
```
Texas = df.groupby(["event_type_2" ,"snap_TX",])
Texas["demand"].nunique().plot(kind = "bar", figsize=(18,6))
plt.title("Demand when there is event type 2 and snap purchasing allowed")
plt.show()
```



Observation

1. We have more demand on Cultural Event and snap Purchasing was allowed

7.5 Demand from each store id when Texas had snap purchasing

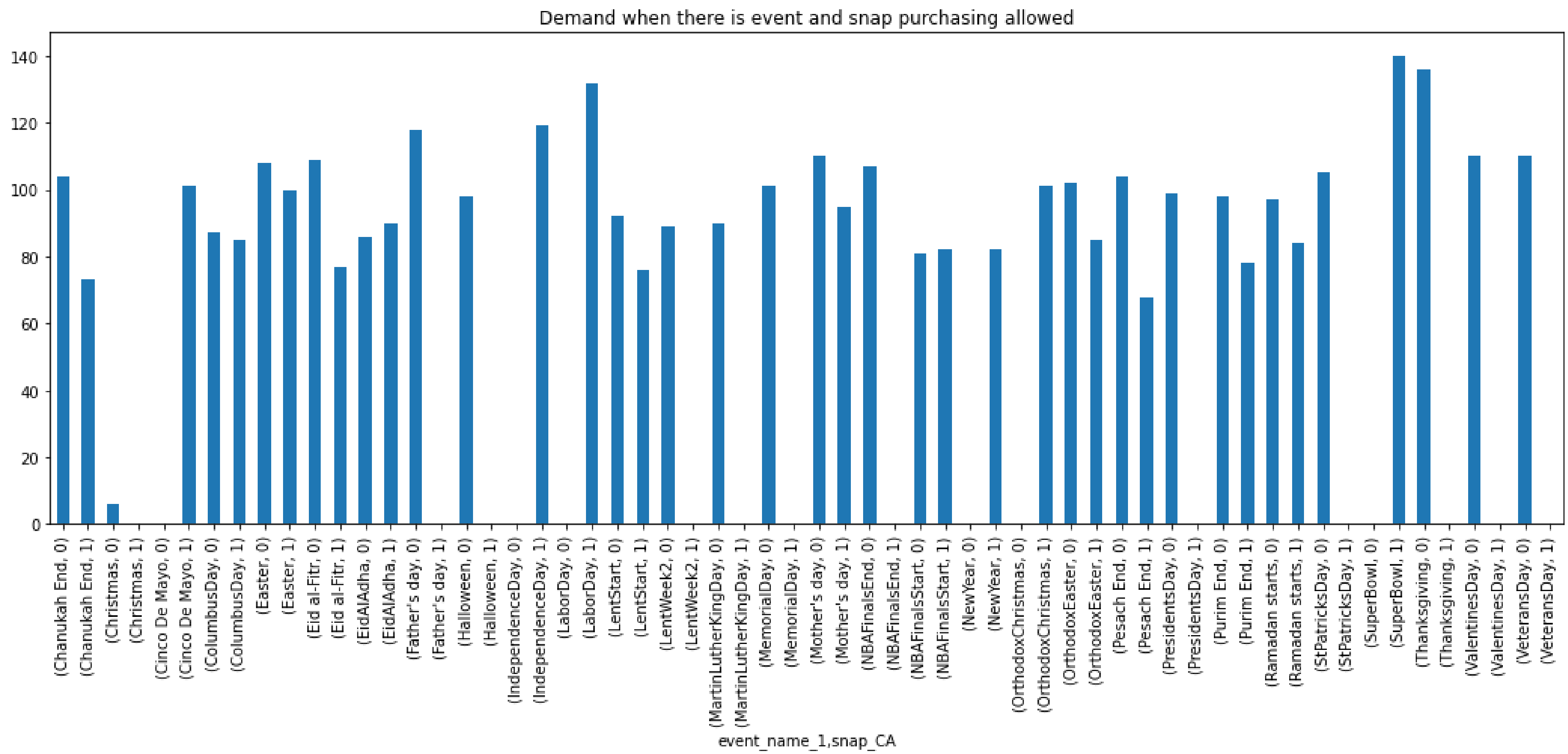
In [172]:
```
event = df.groupby(["store_id" ,"snap_TX",])
event["demand"].nunique().plot(kind = "bar", figsize=(18,6))
plt.title("Demand of product in each store when snap_purchase allowed")
plt.show()
```

Observation

1. CA_3 has more demand on snap purchasing and non snap purchasing days.

7.6 Demand with event_name_1 and snap purchasing in California

In [153]:
```
California = df.groupby(['event_name_1', "snap_CA"])
California["demand"].nunique().plot(kind = "bar", figsize=(18,6))
plt.title("Demand when there is event and snap purchasing allowed")
plt.show()
```

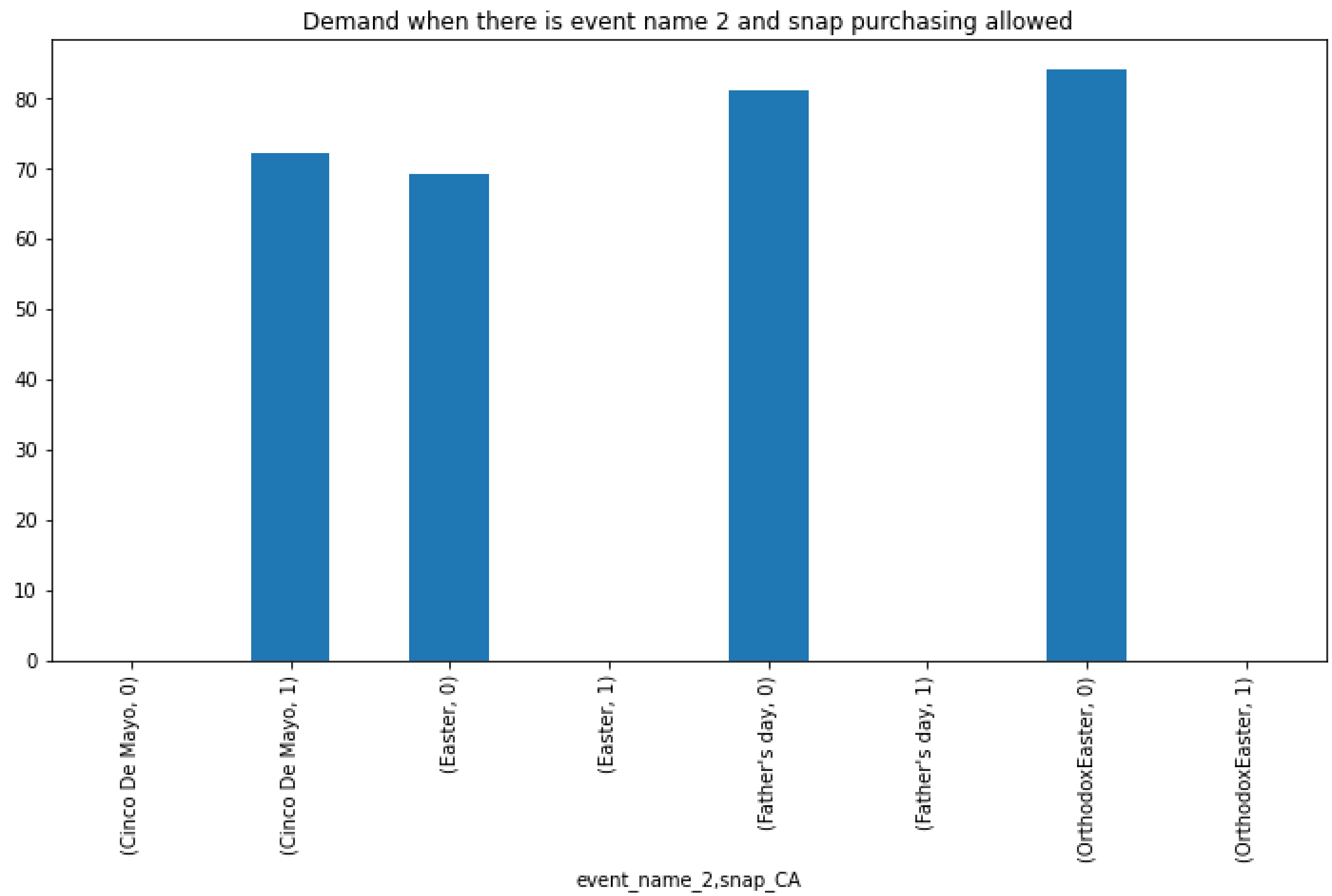


Observation

1. Labour day and Superbowl, have demand on snap purchasing

7.7 Demand with event_name_2 and snap purchasing in California

In [154]:
```
California = df.groupby(["event_name_2" ,"snap_CA",])
California["demand"].nunique().plot(kind = "bar", figsize=(12,6))
plt.title("Demand when there is event name 2 and snap purchasing allowed")
plt.show()
```

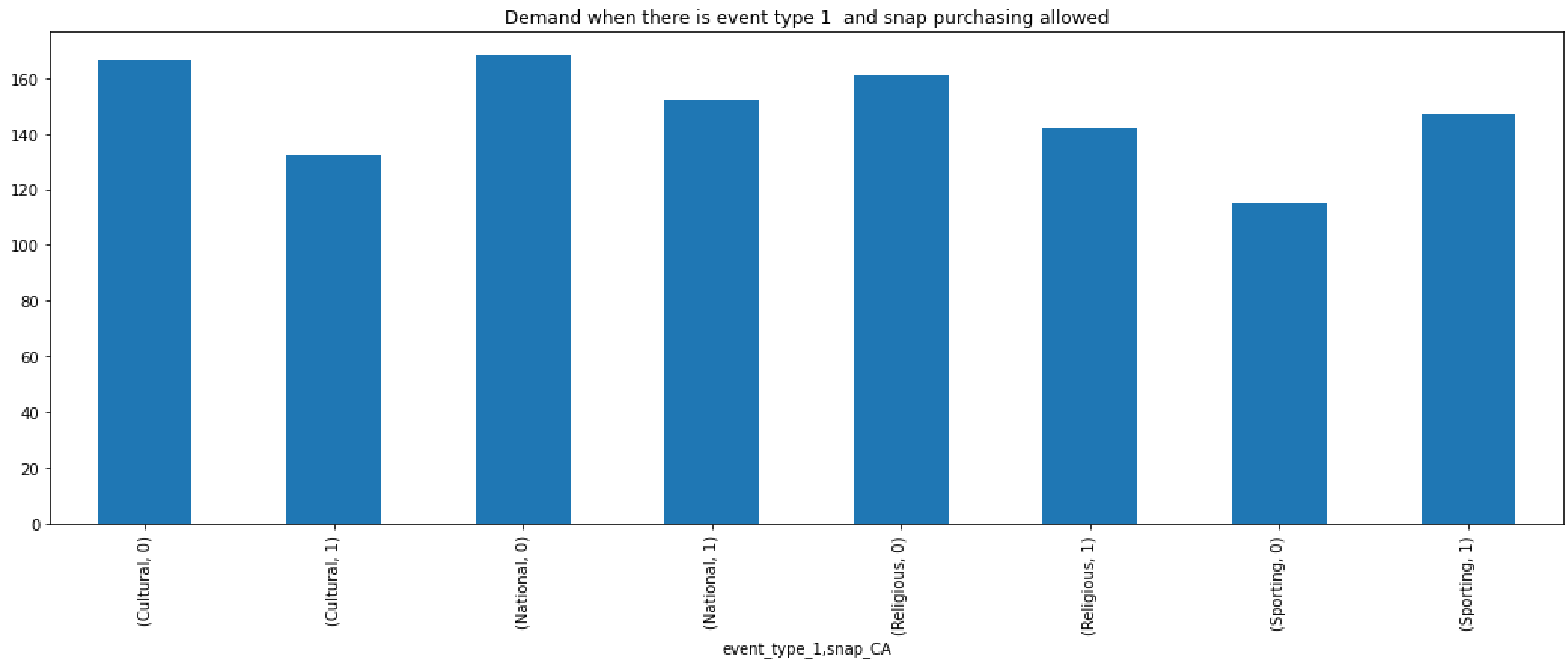


Observation

1. We can definetly see by Cinco De Mayo and father's day, if we have event and snap purchasing is allowed then there is high sell.
2. Orthadox Easter has highest sell even without any snap purchasing

7.8 Demand with even_type_1 and snap purchasing in California

In [155]:
```
California = df.groupby(["event_type_1" ,"snap_CA",])
California["demand"].nunique().plot(kind = "bar", figsize=(18,6))
plt.title("Demand when there is event type 1  and snap purchasing allowed")
plt.show()
```

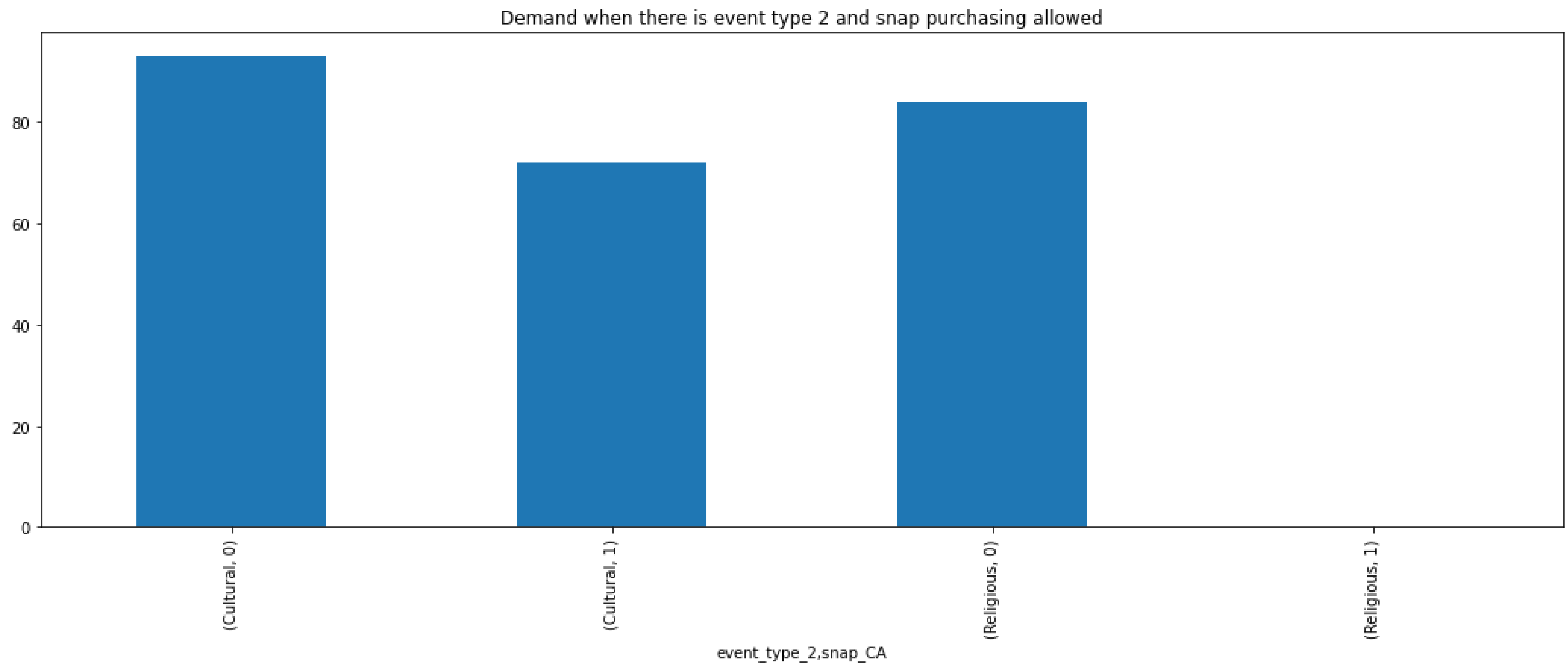


Observation

1. We can see even the non-snap purchasing days has more demand .

7.9 Demand with even_type_2 and snap purchasing in California

In [156]:
```
California = df.groupby(["event_type_2" ,"snap_CA",])
California["demand"].nunique().plot(kind = "bar", figsize=(18,6))
plt.title("Demand when there is event type 2 and snap purchasing allowed")
plt.show()
```

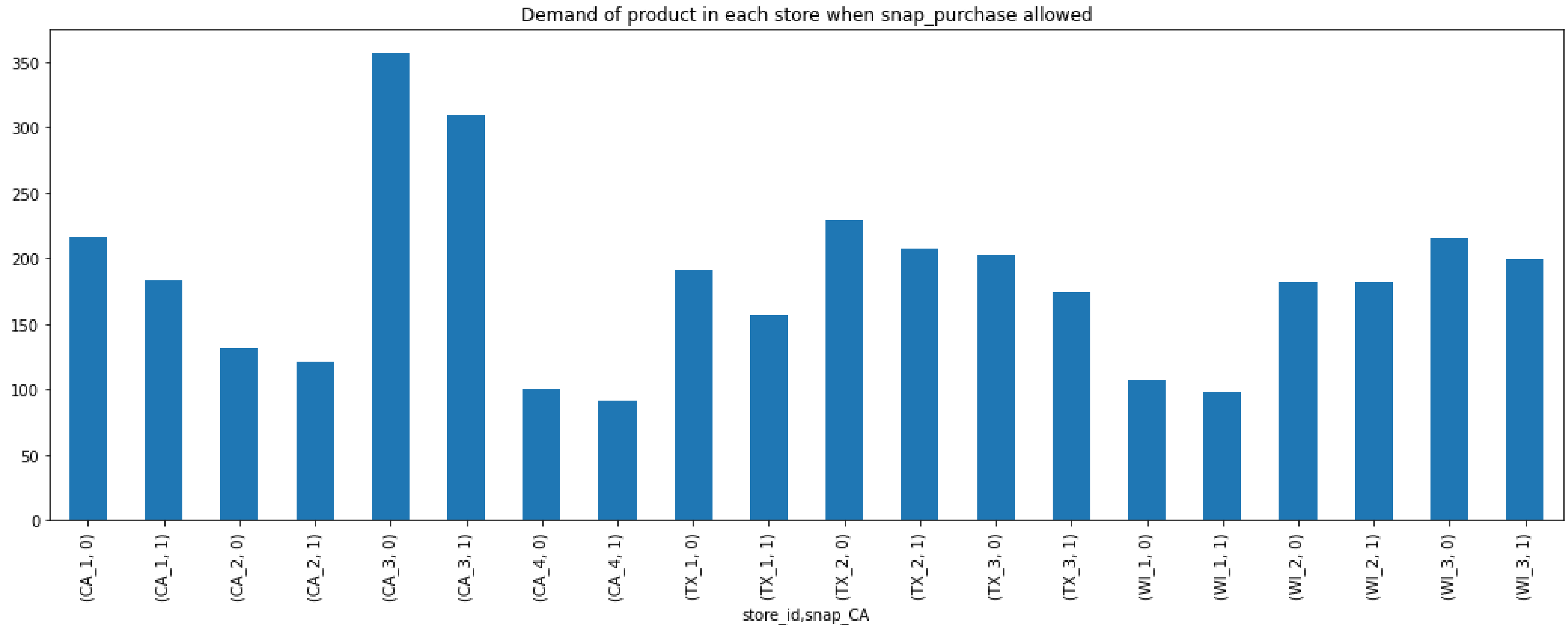


Observation

1. We have more demand on Cultural Event and snap Purchasing was allowed

7.10 Demand from each store id when California had snap purchasing

In [173]:
```
event = df.groupby(["store_id" ,"snap_CA",])
event["demand"].nunique().plot(kind = "bar", figsize=(18,6))
plt.title("Demand of product in each store when snap_purchase allowed")
plt.show()
```



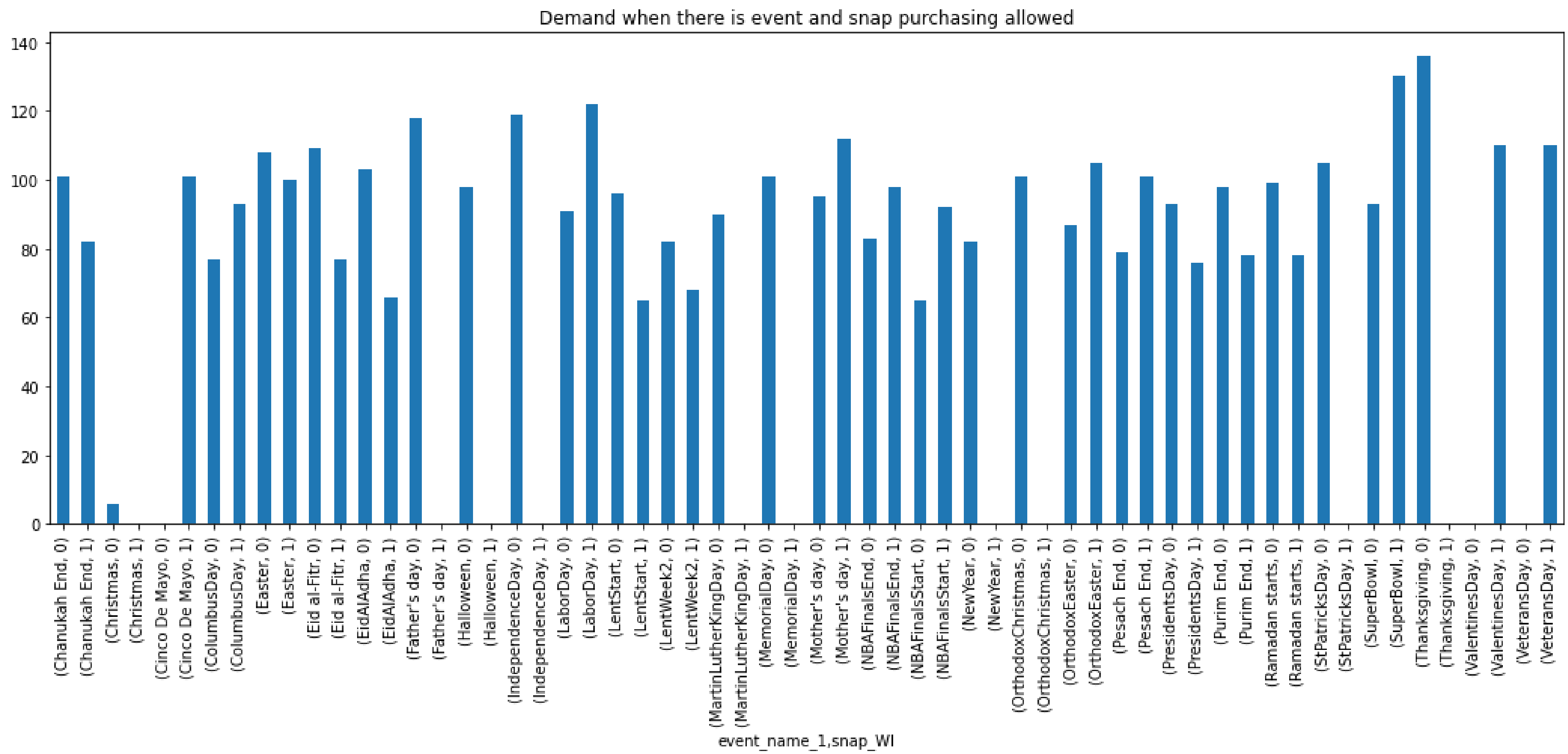
Observation

1. CA_3 has more demand on snap purchasing and non snap purchasing days.

7.11 Demand with event_name_1 and snap purchasing in Wisconsin

In [157]:

```
Wisconsin = df.groupby(['event_name_1', "snap_WI"])
Wisconsin["demand"].nunique().plot(kind = "bar", figsize=(18,6))
plt.title("Demand when there is event and snap purchasing allowed")
plt.show()
```

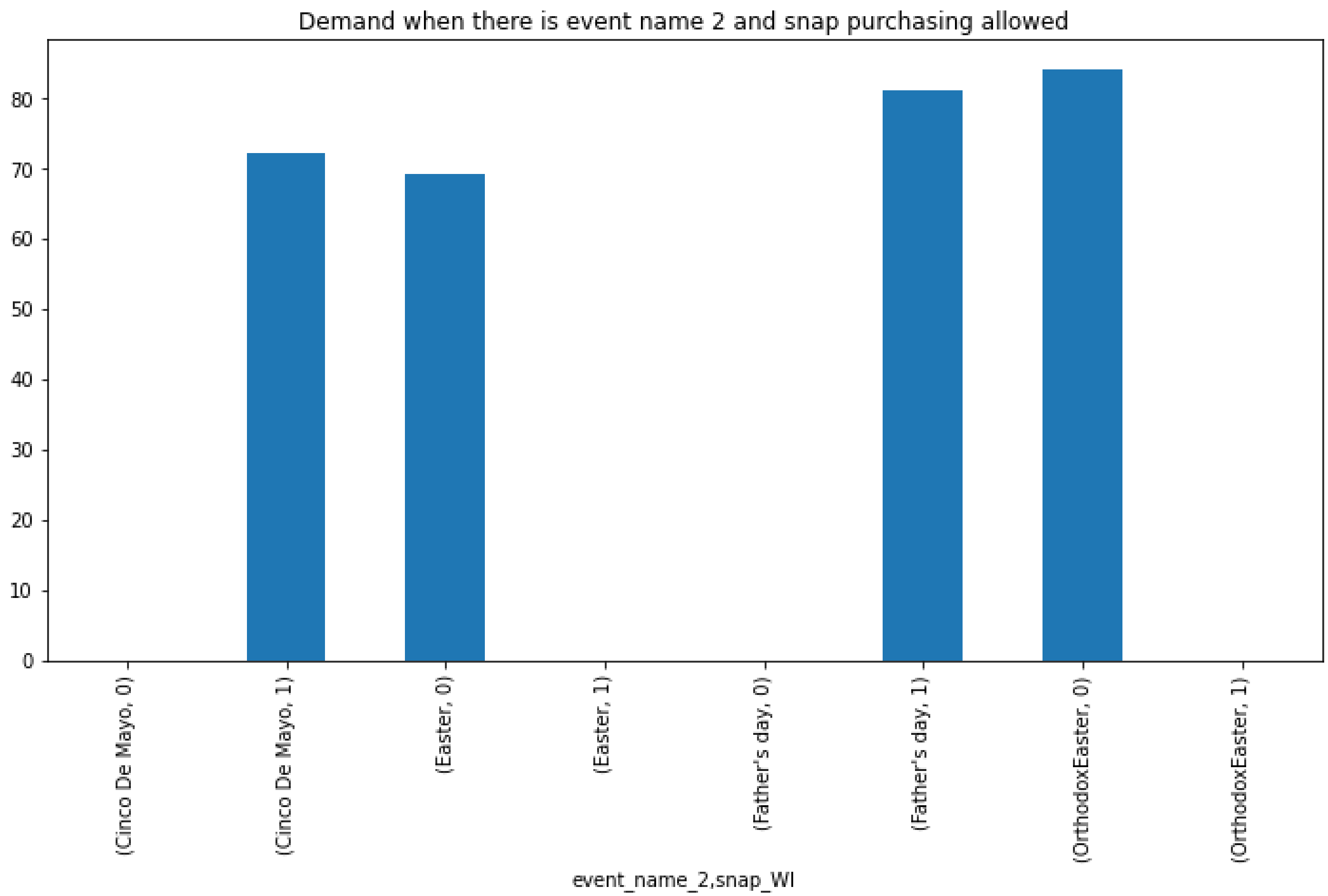


Observation

1. Labour day and Superbowl, have demand on snap purchasing

7.12 Demand with event_name_2 and snap purchasing in Wisconsin

In [158]:
```
Wisconsin = df.groupby(["event_name_2" ,"snap_WI",])
Wisconsin["demand"].nunique().plot(kind = "bar", figsize=(12,6))
plt.title("Demand when there is event name 2 and snap purchasing allowed")
plt.show()
```

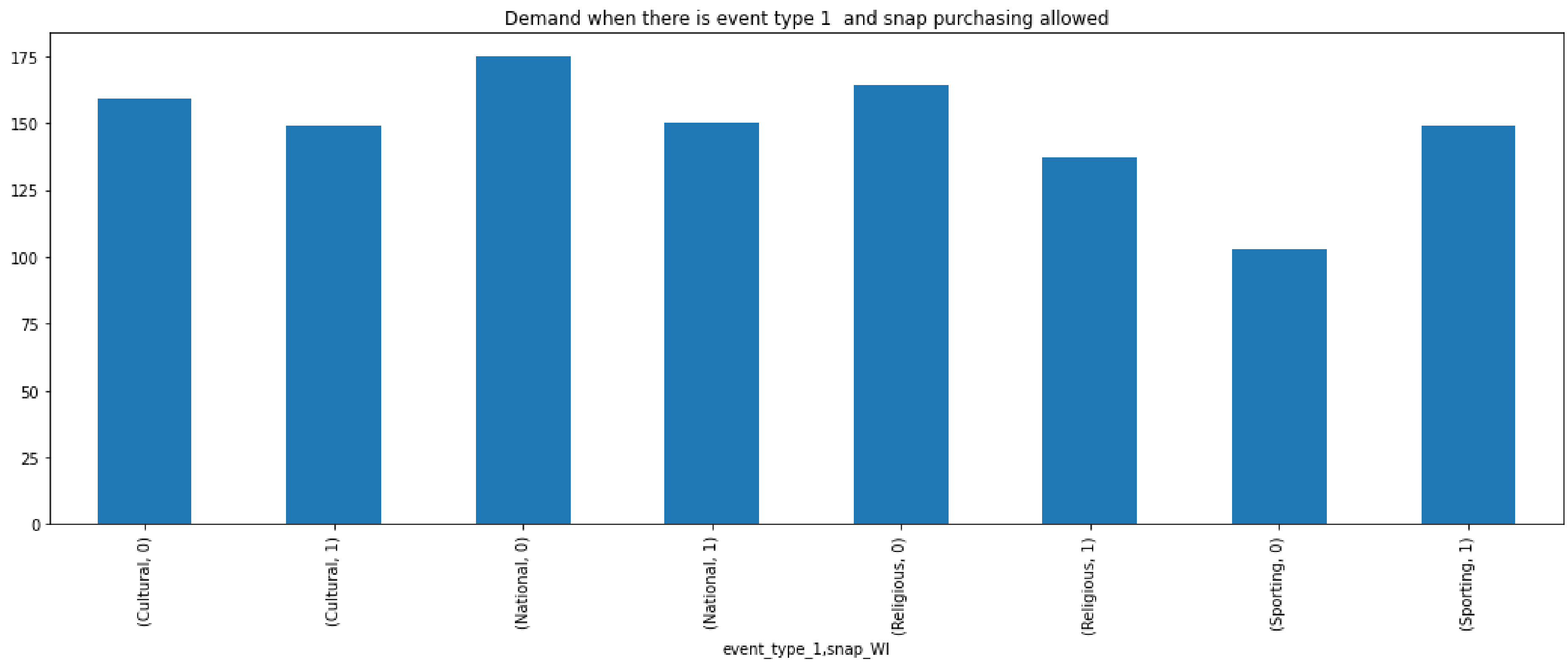


Observation

1. We can see even the non-snap purchasing days has more demand .

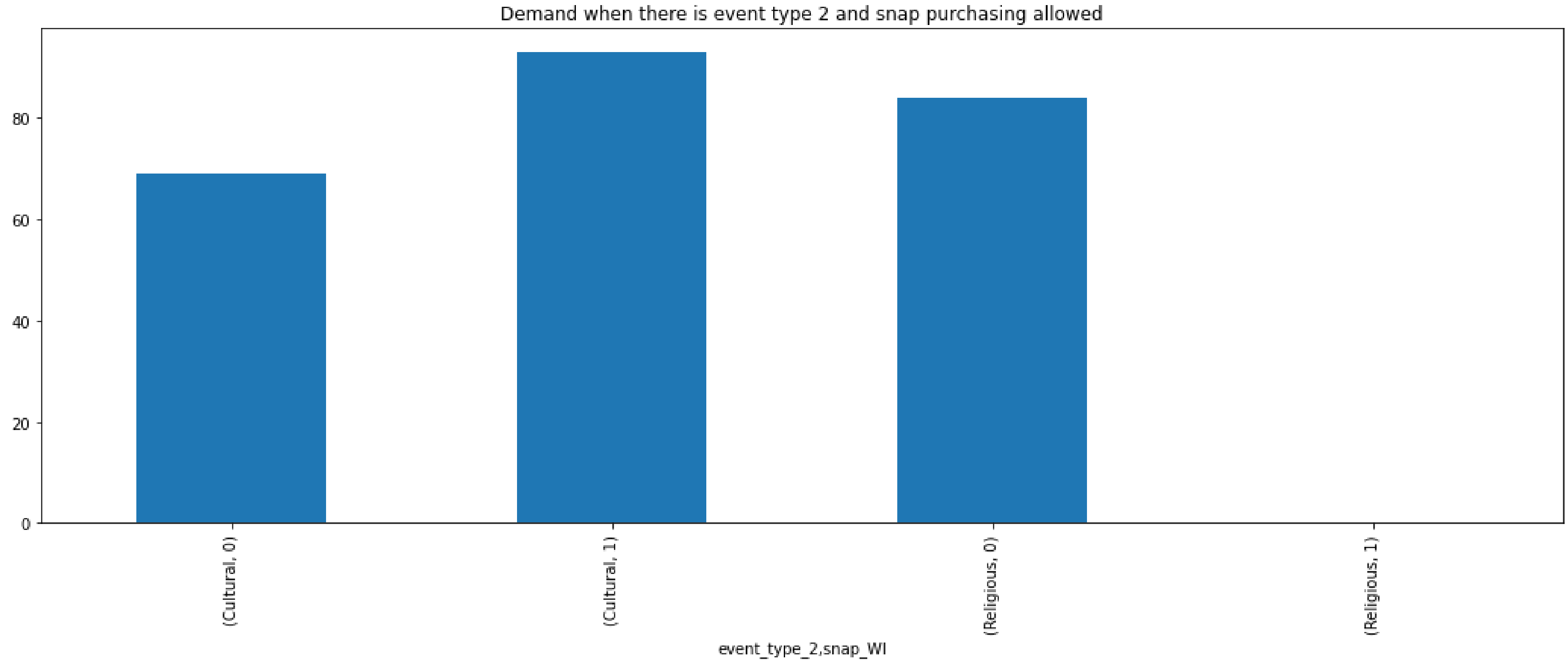
7.13 Demand with even_type_1 and snap purchasing in Wisconsin

In [159]:
```
Wisconsin = df.groupby(["event_type_1" ,"snap_WI",])
Wisconsin["demand"].nunique().plot(kind = "bar", figsize=(18,6))
plt.title("Demand when there is event type 1  and snap purchasing allowed")
plt.show()
```



7.14 Demand with even_type_2 and snap purchasing in Wisconsin

In [160]:
```
Wisconsin = df.groupby(["event_type_2" ,"snap_WI",])
Wisconsin["demand"].nunique().plot(kind = "bar", figsize=(18,6))
plt.title("Demand when there is event type 2 and snap purchasing allowed")
plt.show()
```

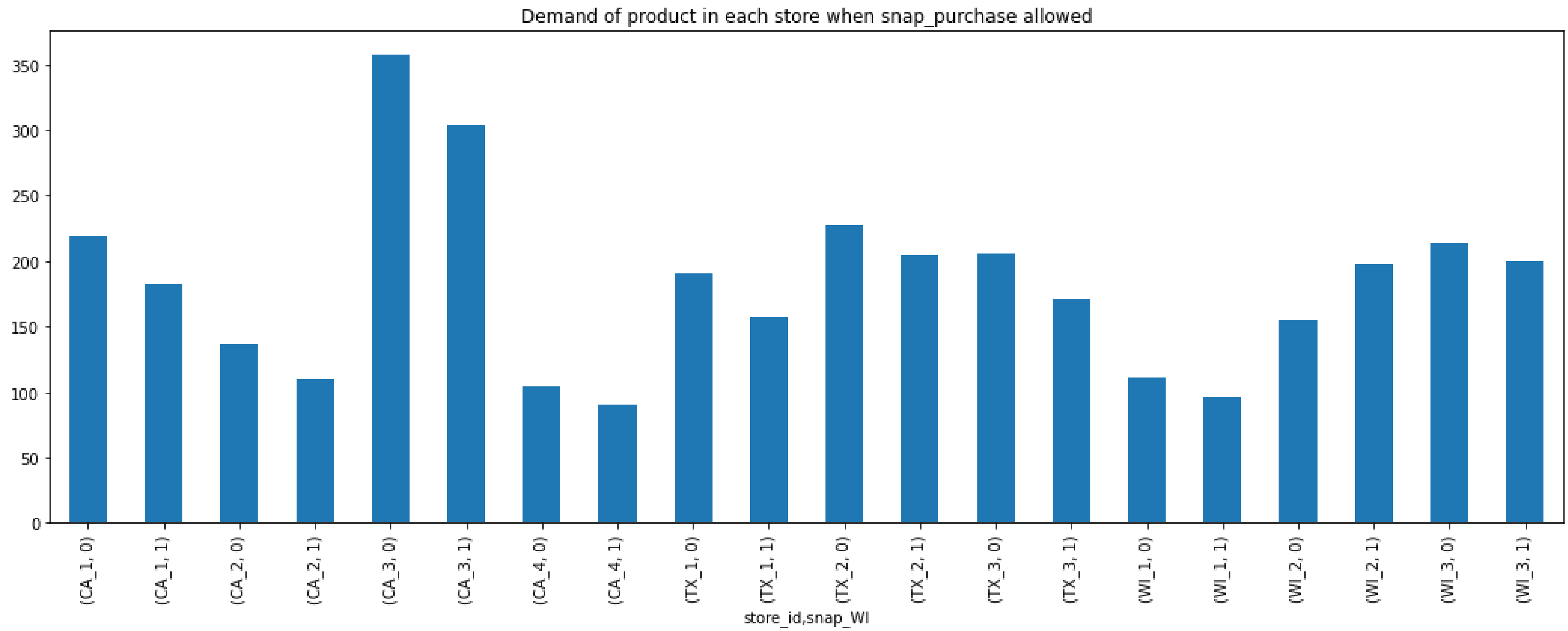


Observation

1. We have more demand on Cultural Event and snap Purchasing was allowed

7.15 Demand in each store when snap purchasing allowed in Wisconsin

In [174]:
```
event = df.groupby(["store_id" ,"snap_WI",])
event["demand"].nunique().plot(kind = "bar", figsize=(18,6))
plt.title("Demand of product in each store when snap_purchase allowed")
plt.show()
```

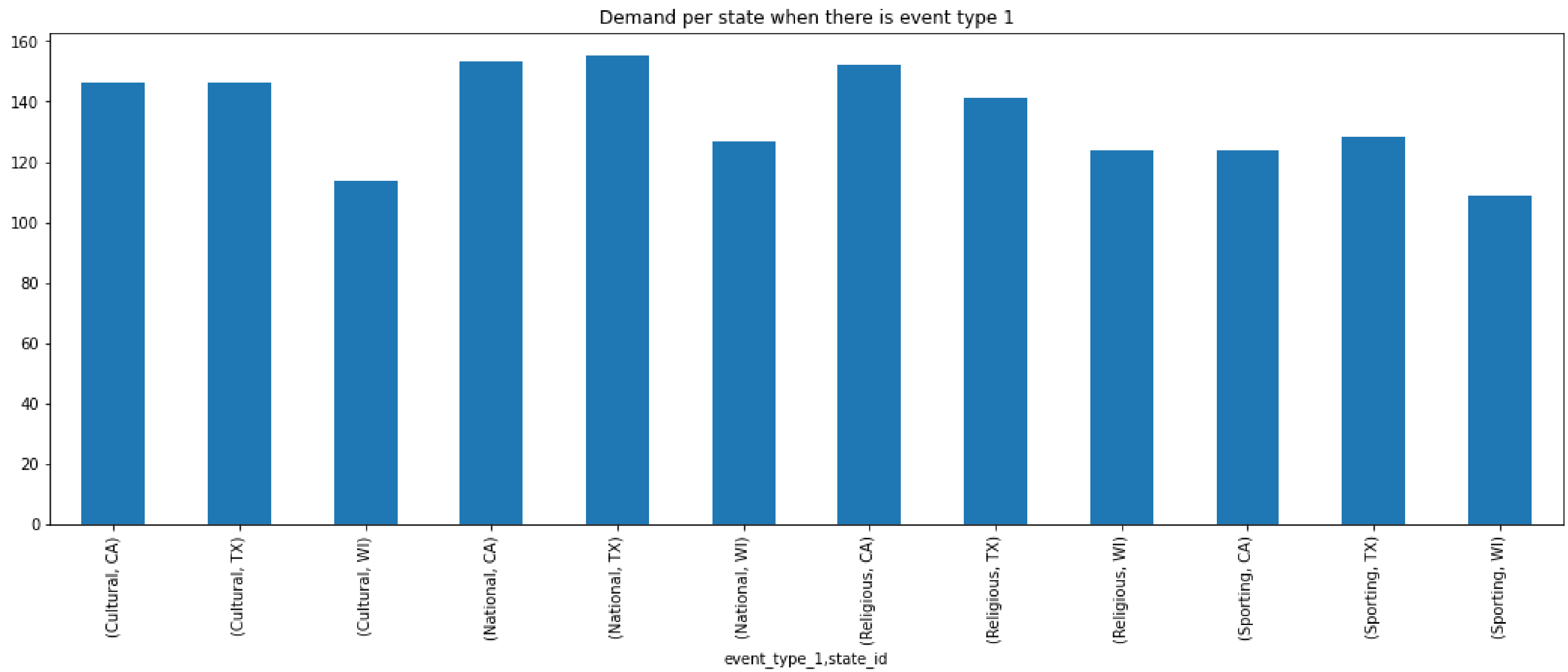


Observation

1. CA_3 has more demand on snap purchasing and non snap purchasing days.

7.16 Demand per state when there is an event

```
In [163]: event = df.groupby(["event_type_1" ,"state_id",])
event["demand"].nunique().plot(kind = "bar", figsize=(18,6))
plt.title("Demand per state when there is event type 1")
plt.show()
```

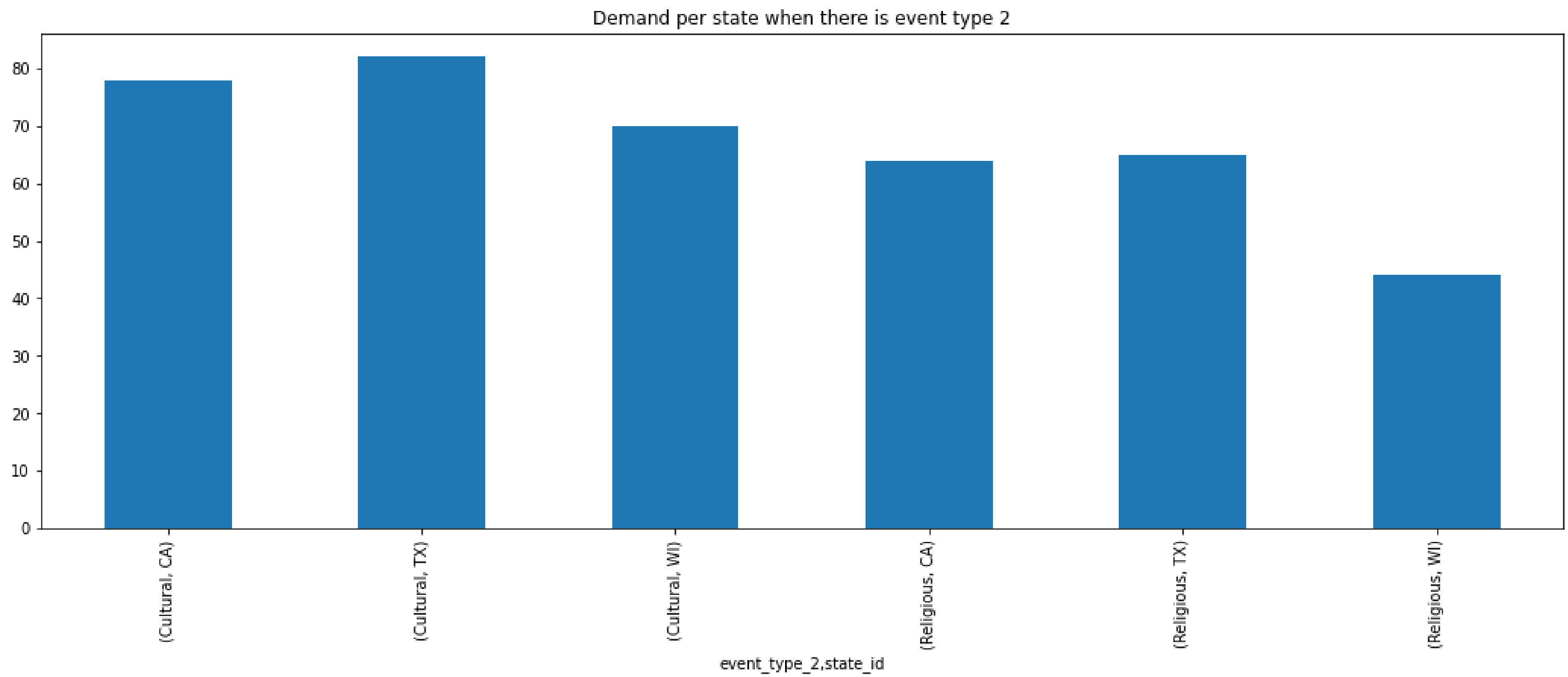


Observation

1. California and Texas have more demand on National Event
2. California and Texas have also second highest demand on Cultural Event

7.17 Demand per state when event type 2

In [165]:
```
event = df.groupby(["event_type_2" ,"state_id",])
event["demand"].nunique().plot(kind = "bar", figsize=(18,6))
plt.title("Demand per state when there is event type 2")
plt.show()
```

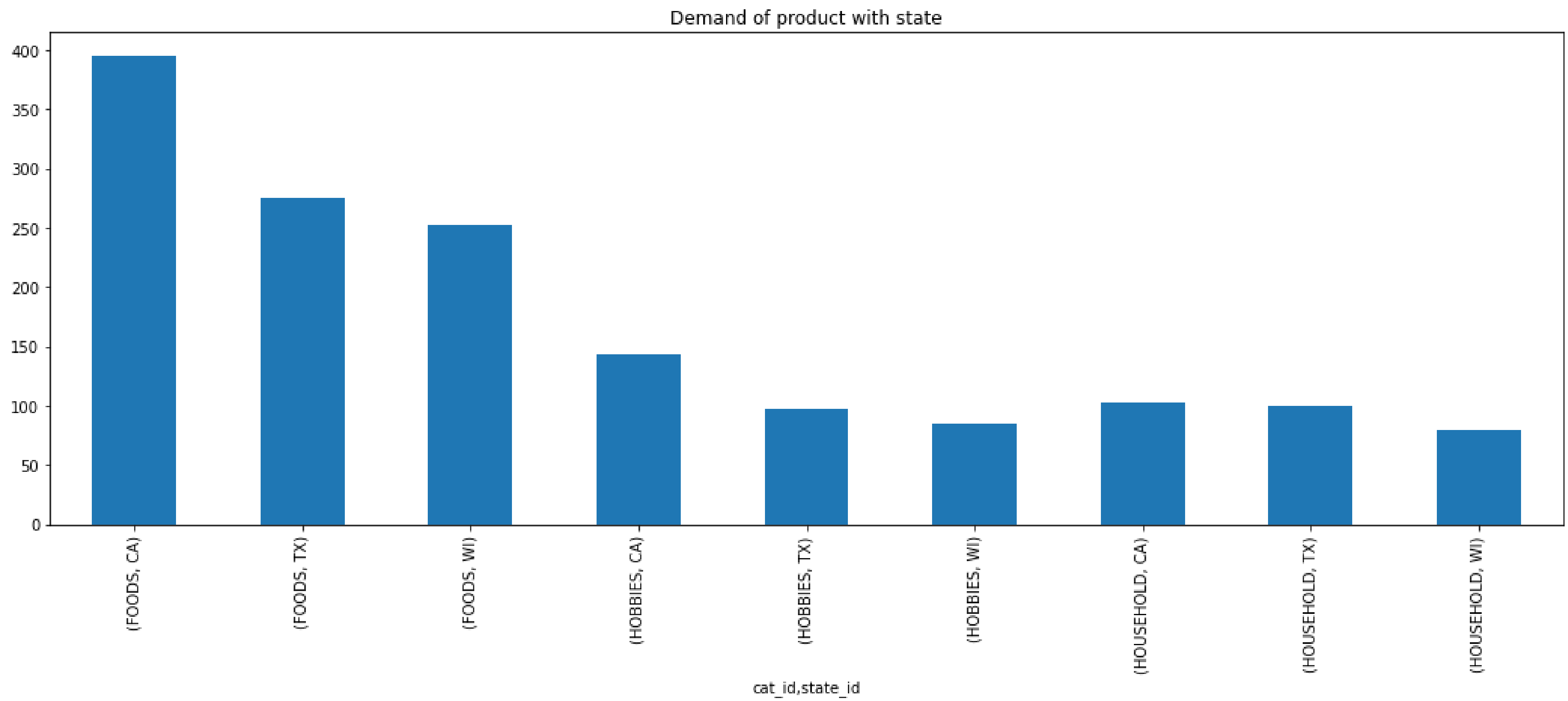


Observation

1. Cultural event has more demand.
2. Texas has more demand on cultural event. 3.California has second highest demand on cultural event

7.18 Demand of product with state

In [167]:
```
event = df.groupby(["cat_id" ,"state_id",])
event["demand"].nunique().plot(kind = "bar", figsize=(18,6))
plt.title("Demand of product with state")
plt.show()
```

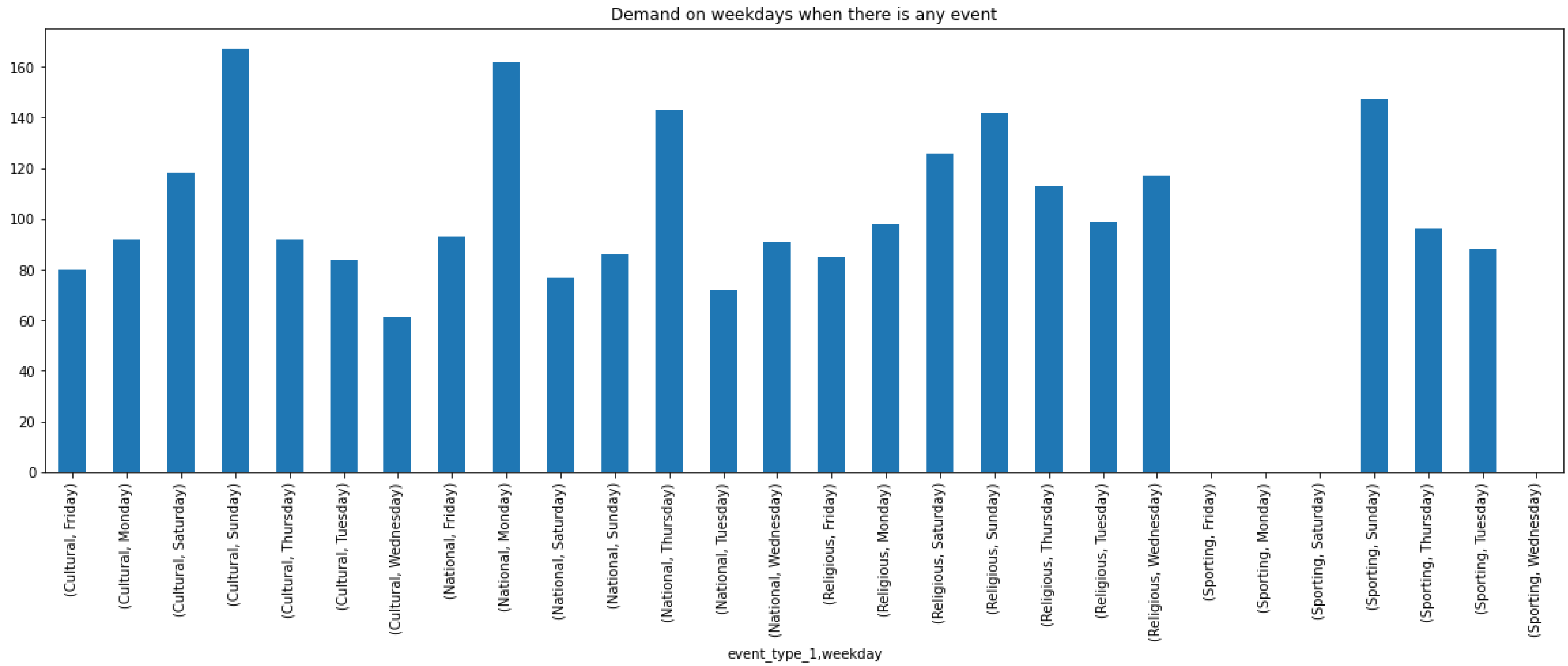


Observation

1. Food category has more demand.
2. California has highest demand on food.
3. Texas has second highest demand on food.

7.19 Sell on weekdays and when there is an event type 1

In [210]:
```
a = df.groupby(["event_type_1" ,"weekday"])
a["demand"].nunique().plot.bar(figsize=(20,6))
plt.title("Demand on weekdays when there is any event")
plt.show()
```



Observation

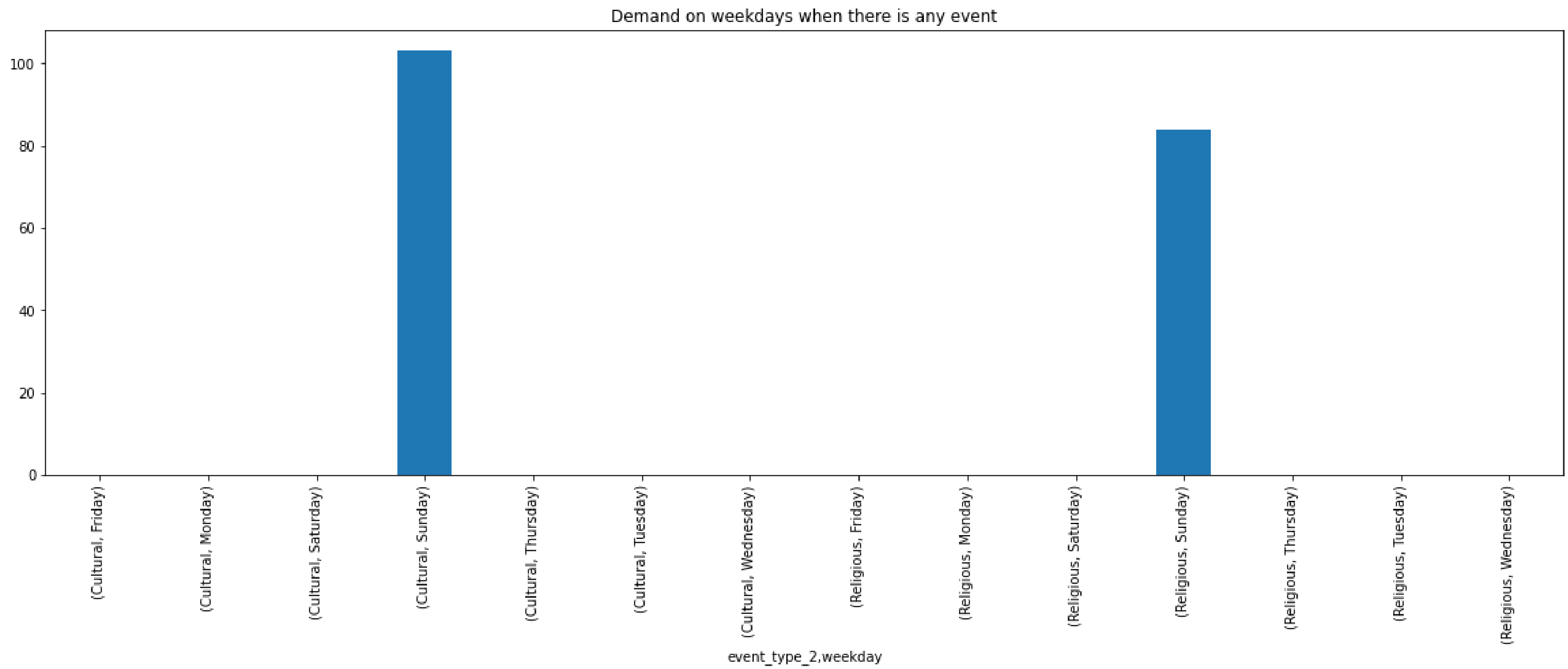
1.Cultural and Religios Event have more demand on Sunday.

2.National Event has more demand on Monday and Thrusday.

7.20 Demand on weekdays when there is an event type 2

In [209]:

```
a = df.groupby(["event_type_2" ,"weekday"])
a["demand"].nunique().plot.bar(figsize=(20,6))
plt.title("Demand on weekdays when there is any event")
plt.show()
```

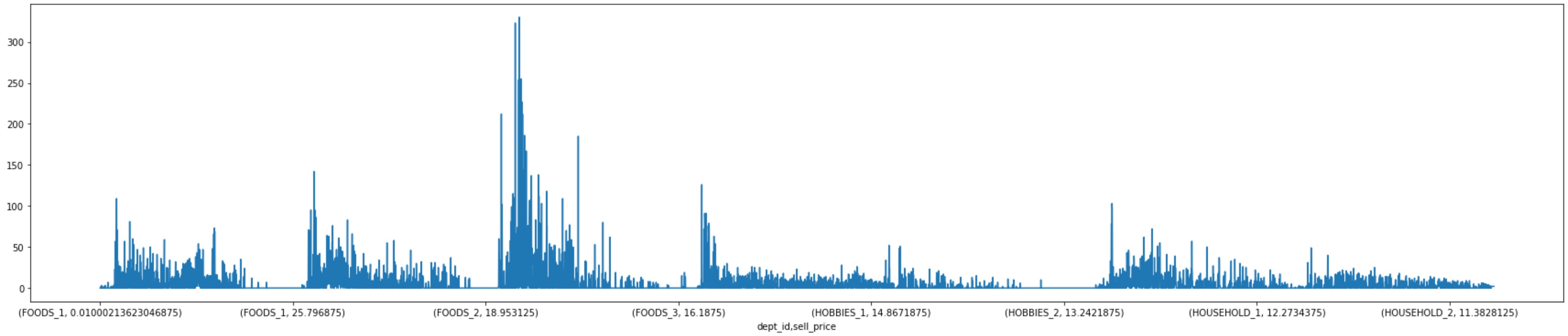


Observation

1. Cultural and Religious event has more demand on Sunday

7.21 Demand with Department id when sell price is high or low

In [217]:
```
a = df.groupby(["dept_id" ,"sell_price"])
a["demand"].nunique().plot( figsize=(30,6))
plt.show()
```



In [ ]:
```

#which store has maximum price sell on which date
dataset_1 = df.groupby(["dept_id" ,"demand", "state_id","store_id","date", ])['sell_price'].max()
dataset_1 = pd.DataFrame(dataset_1.reset_index())
fig = px.line(dataset_1, x = 'dept_id', y='sell_price', color = "dept_id",
              hover_data = [ 'demand', "store_id"])
fig.show()
```

Observation

1. Food_2 and price between 18 o 17 has more demand.

In [4]:
```
# taking data year wise
df_2011 = df[df.year==2011]
df_2012  = df[df.year==2012]
df_2013 = df[df.year==2013]
df_2014 = df[df.year==2014]
df_2015  = df[df.year==2015]
df_2016  = df[df.year==2016]


#What is the size of the data  final data
print(f'df_2011 file has {df_2011.shape[0]} rows and {df_2011.shape[1]} columns.')
print(f'df_2012 file has {df_2012.shape[0]} rows and {df_2012.shape[1]} columns.')
print(f'df_2013 file has {df_2013.shape[0]} rows and {df_2013.shape[1]} columns.')
print(f'df_2014 file has {df_2014.shape[0]} rows and {df_2014.shape[1]} columns.')
print(f'df_2015 file has {df_2015.shape[0]} rows and {df_2015.shape[1]} columns.')
print(f'df_2016 file has {df_2016.shape[0]} rows and {df_2016.shape[1]} columns.')
```

```
df_2011 file has 10275130 rows and 24 columns.
df_2012 file has 11159340 rows and 24 columns.
df_2013 file has 11128850 rows and 24 columns.
df_2014 file has 11128850 rows and 24 columns.
df_2015 file has 11128850 rows and 24 columns.
df_2016 file has 4360070 rows and 24 columns.
```

In [5]:
```
df_2011 = downcast(df_2011)
df_2012  = downcast(df_2012)
df_2013 = downcast(df_2013)
df_2014 = downcast(df_2014)
df_2015  = downcast(df_2015)
df_2016  = downcast(df_2016)
```

```
24it [00:00, 40.07it/s]
24it [00:00, 37.87it/s]
24it [00:00, 40.14it/s]
24it [00:00, 43.10it/s]
24it [00:00, 44.52it/s]
24it [00:00, 115.82it/s]
```

7.22 Individual year analysis

In [8]:

```
def demand_year_wise(dataset_1, Title):


    dataset_1 = dataset_1.groupby([ 'date', 'month'])['demand'].sum()
    dataset_1 = pd.DataFrame(dataset_1.reset_index())
    fig = px.line(dataset_1, x = 'date', y='demand', color='month', title = Title)



    fig.show()

demand_year_wise(df_2011, "Demand in 2011")
```

In [9]:
```
demand_year_wise(df_2012, "Demand in 2012")
```

In [10]:

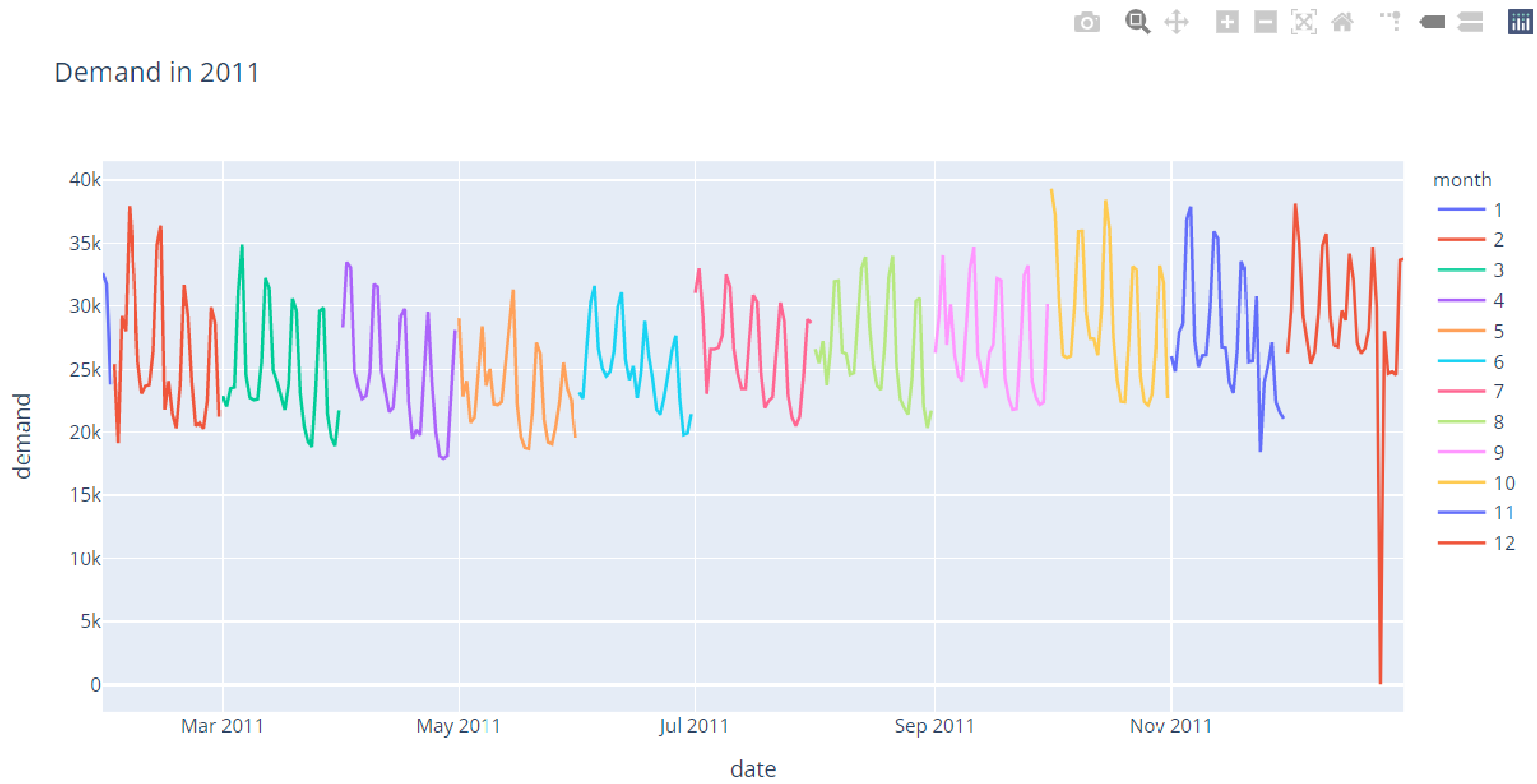
```
demand_year_wise(df_2013, "Demand in 2013")
```

In [11]:
```
demand_year_wise(df_2014, "Demand in 2014")
```

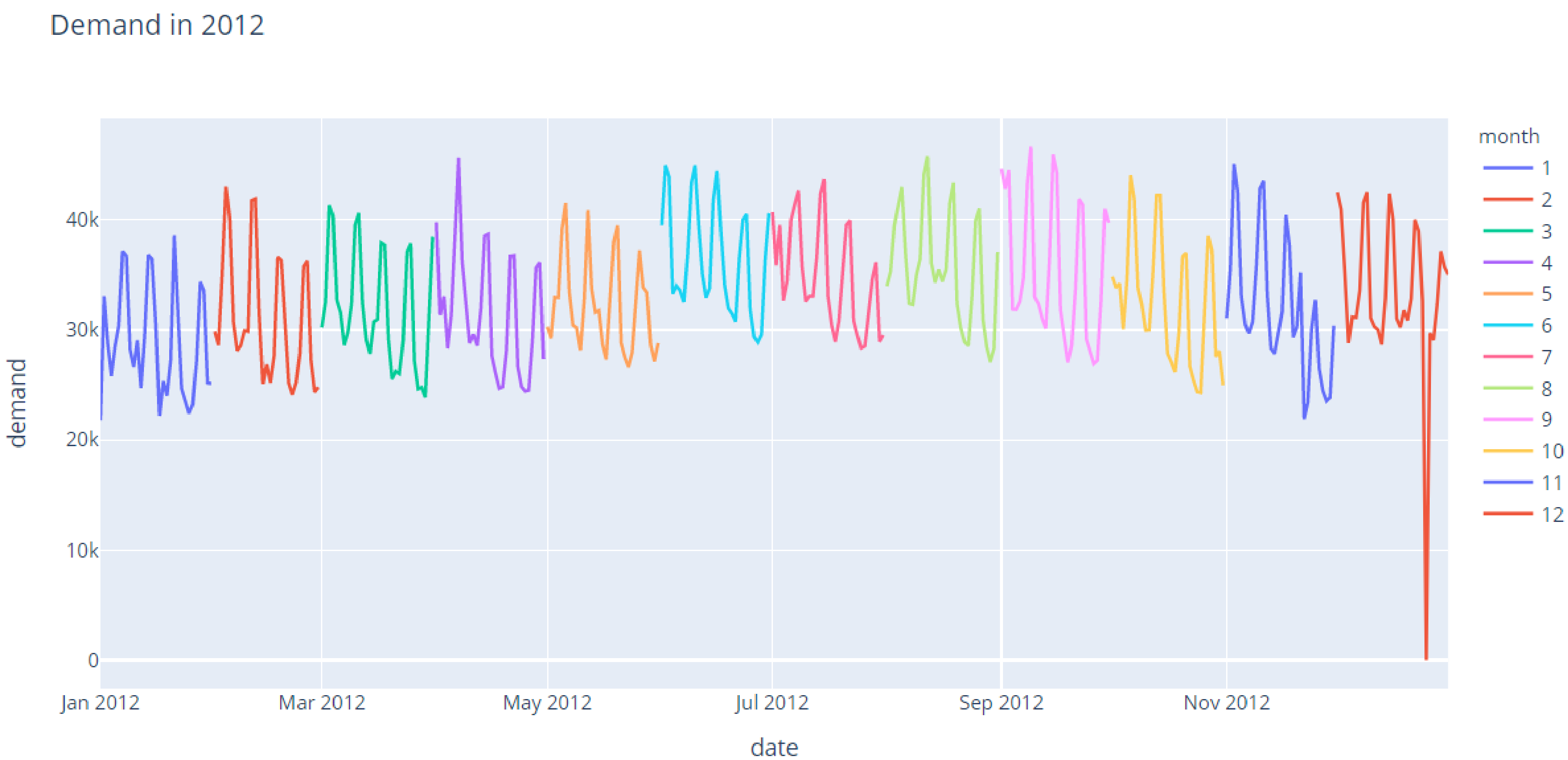
In [12]:
```
demand_year_wise(df_2015, "Demand in 2015")
```

In [13]:
```
demand_year_wise(df_2016, "Demand in 2016")
```

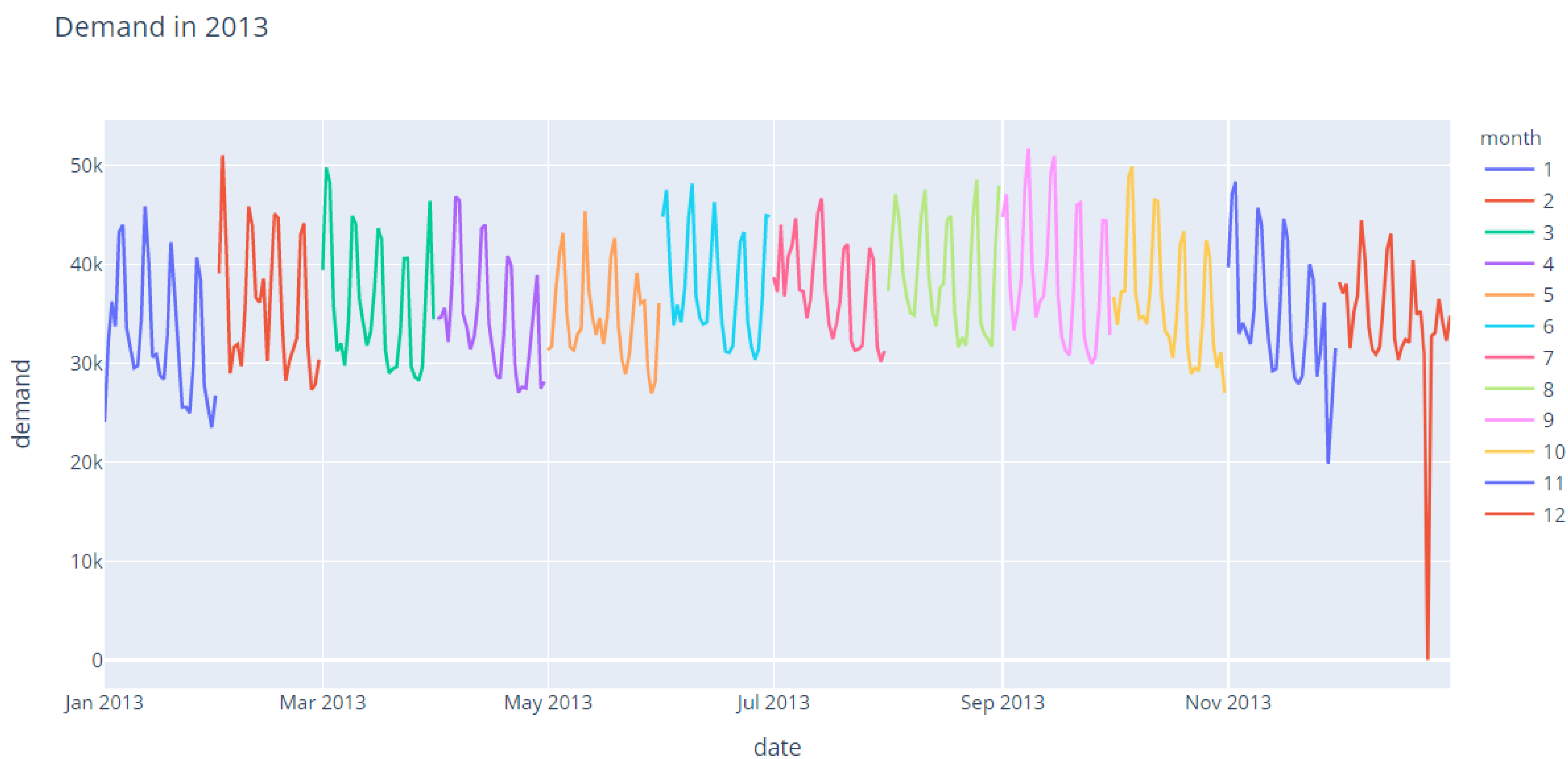
plotly doesn't render plots after closing the kernel, so adding it as a image



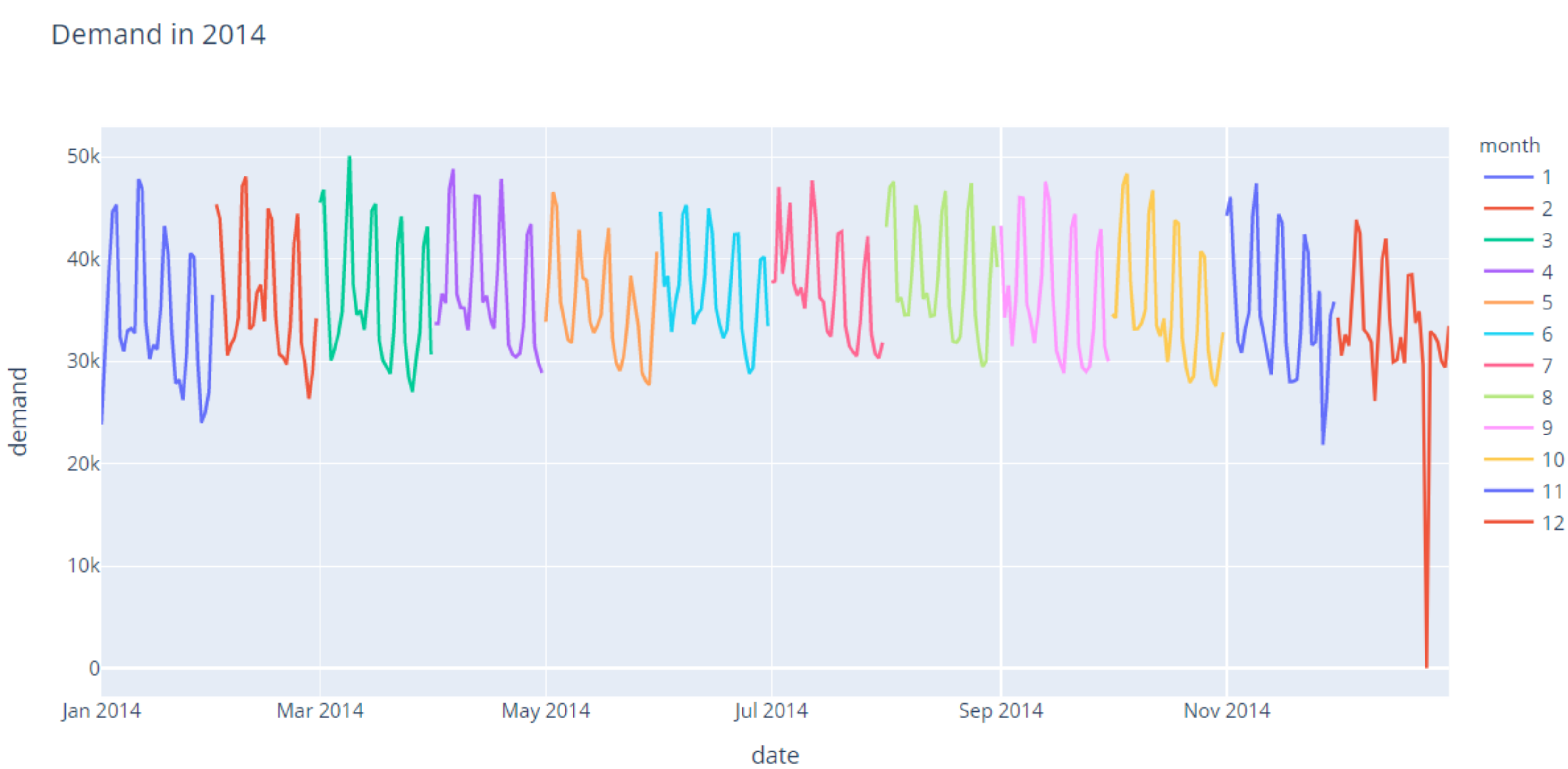
Demand in 2011
demand
40k
35k
30k
25k
20k
15k
10k
5k
0
Mar 2011
May 2011
Jul 2011
Sep 2011
Nov 2011
date
month
1
2
3
4
5
6
7
8
9
10
11
12

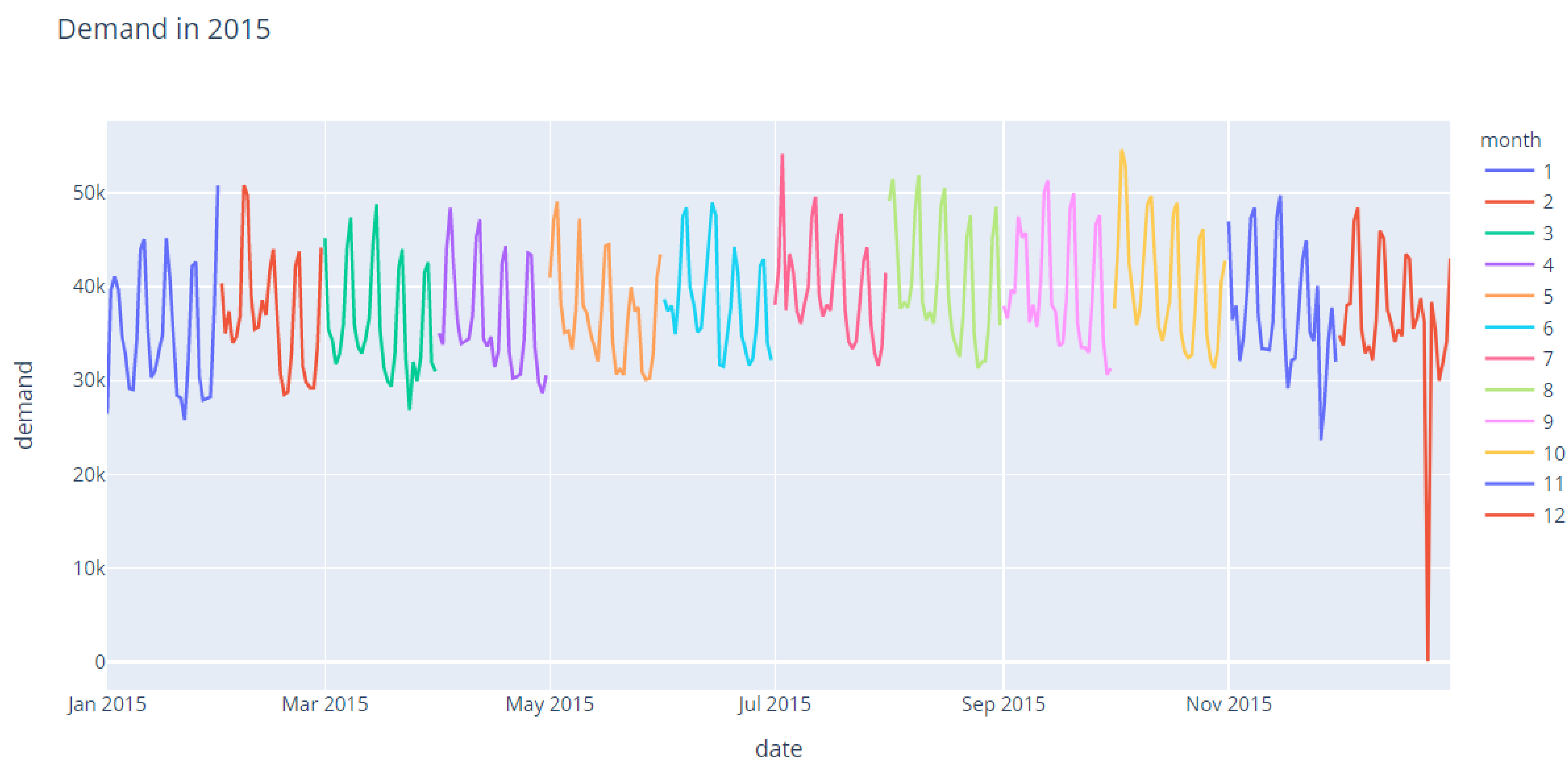


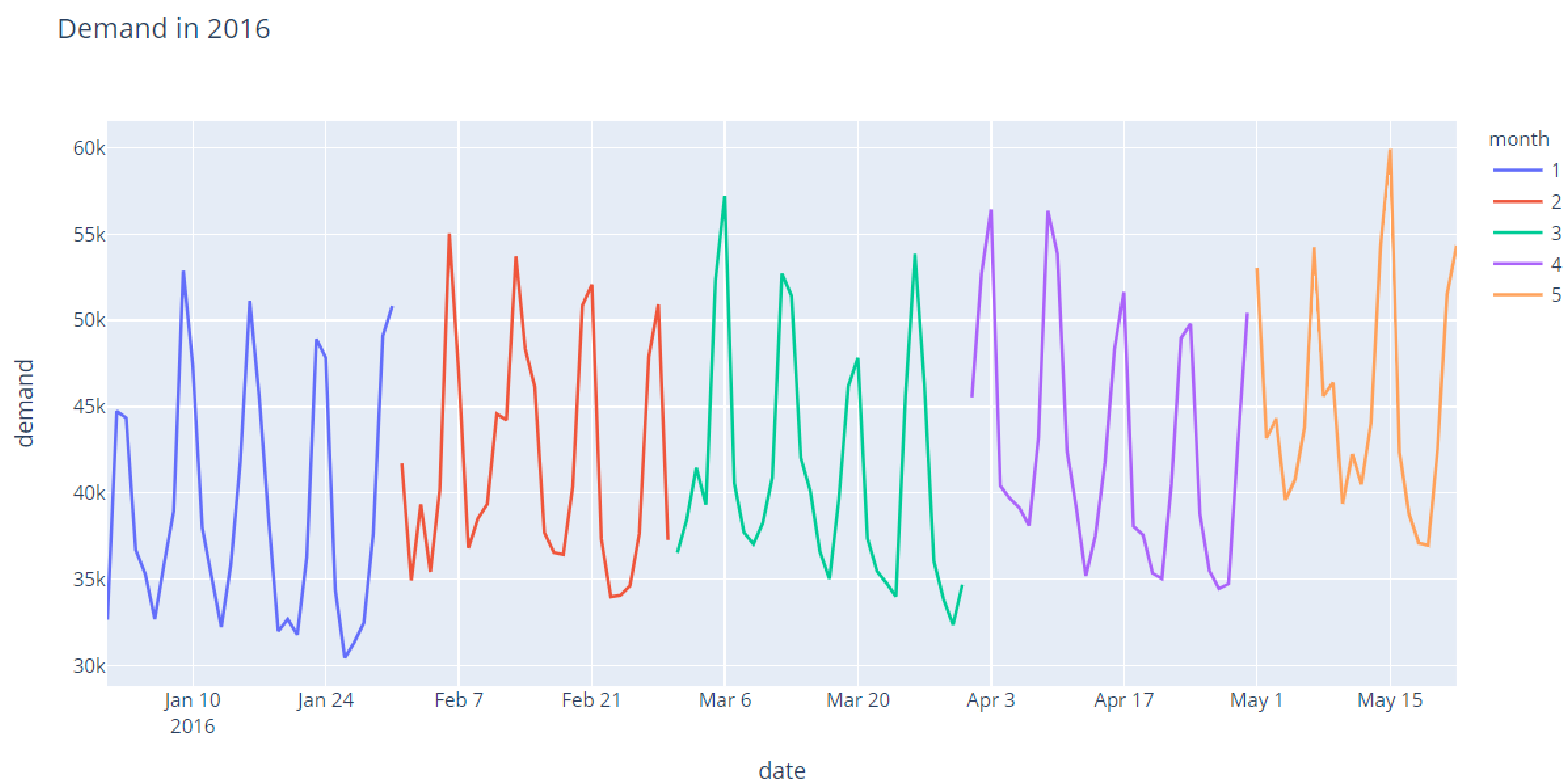
Demand in 2012
demand
0
10k
20k
30k
40k
date
Jan 2012
Mar 2012
May 2012
Jul 2012
Sep 2012
Nov 2012
month
1
2
3
4
5
6
7
8
9
10
11
12



Demand in 2013
demand
50k
40k
30k
20k
10k
0
Jan 2013
Mar 2013
May 2013
Jul 2013
Sep 2013
Nov 2013
date
month
1
2
3
4
5
6
7
8
9
10
11
12


Demand in 2014
demand
50k
40k
30k
20k
10k
0
Jan 2014
Mar 2014
May 2014
Jul 2014
Sep 2014
Nov 2014
date
month
1
2
3
4
5
6
7
8
9
10
11
12


Demand in 2015
demand
50k
40k
30k
20k
10k
0
Jan 2015
Mar 2015
May 2015
Jul 2015
Sep 2015
Nov 2015
date
month
1
2
3
4
5
6
7
8
9
10
11
12



7.22.1 Observation

1. There is no or very less on christmas in every year.
2. There is seasonality in our data. Every year it repeats .

7.23 Demand on Weekend

In [28]:
```
df_2011 = df[df.year==2011]
# df_2012  = df[df.year==2012]
# df_2013 = df[df.year==2013]
# df_2014 = df[df.year==2014]
# df_2015  = df[df.year==2015]
# df_2016  = df[df.year==2016]

df_2011 = downcast(df_2011)
# df_2012  = downcast(df_2012)
# df_2013 = downcast(df_2013)
# df_2014 = downcast(df_2014)
# df_2015  = downcast(df_2015)
# df_2016  = downcast(df_2016)
```

```
27it [00:01, 13.77it/s]
```

7.24 demand when snap purchase allowed and has an event on weekend and weekdays

In [33]:
```
def demand_year_snap__days(dataset, year):
    dataset_weekend_event_snap_allowed = dataset [ (dataset[ 'weekday'] == "Sunday") | (df_2016[ 'weekday'] == "Sunday") |
                              | (dataset['snap_WI'] == 1) | (dataset['snap_CA'] == 1)]
    dataset_weekday = dataset [ (dataset[ 'weekday'] != "Sunday") | (df_2016[ 'weekday'] != "Sunday") | (dataset['snap_TX'
                              | (dataset['snap_WI'] != 1) | (dataset['snap_CA'] != 1)]

    print(f" demand in {year} when event was on weekend and snap purchasing was allowed \n{dataset_weekend_event_snap_allo
    print("***" *20)
    print(f" demand in {year} on weekdays \n{dataset_weekday.demand.describe().apply(lambda x: format(x, 'f'))}" )
```

In [39]:
```
demand_year_snap__days(df_2011, 2011)
```

```
 demand in 2011 when event was on weekend and snap purchasing was allowed
count    5793100.000000
mean           0.913046
std            3.867120
min            0.000000
25%            0.000000
50%            0.000000
75%            0.000000
max          634.000000
Name: demand, dtype: object
************************************************************
 demand in 2011 on weekdays
count    10061700.000000
mean            0.856844
std             3.675353
min             0.000000
25%             0.000000
50%             0.000000
75%             0.000000
max           693.000000
Name: demand, dtype: object
```

```
In [35]: demand_year_snap__days(df_2012, 2012)
```

```
 demand in 2012 when event was on weekend and snap purchasing was allowed
count    6280940.000000
mean           1.158031
std            4.458339
min            0.000000
25%            0.000000
50%            0.000000
75%            1.000000
max          498.000000
Name: demand, dtype: object
************************************************************
 demand in 2012 on weekdays
count    10976400.000000
mean            1.075411
std             4.183434
min             0.000000
25%             0.000000
50%             0.000000
75%             1.000000
max           648.000000
Name: demand, dtype: object
```

```
In [34]: demand_year_snap__days(df_2013, 2013)
```

```
 demand in 2013 when event was on weekend and snap purchasing was allowed
count    6280940.000000
mean           1.258419
std            4.457750
min            0.000000
25%            0.000000
50%            0.000000
75%            1.000000
max          763.000000
Name: demand, dtype: object
************************************************************
 demand in 2013 on weekdays
count    10915420.000000
mean            1.173782
std             4.201005
min             0.000000
25%             0.000000
50%             0.000000
75%             1.000000
max           763.000000
Name: demand, dtype: object
```

In [36]:
```
demand_year_snap__days(df_2014, 2014)
```

```
 demand in 2014 when event was on weekend and snap purchasing was allowed
count    6311430.000000
mean           1.257649
std            3.944645
min            0.000000
25%            0.000000
50%            0.000000
75%            1.000000
max          567.000000
Name: demand, dtype: object
************************************************************
 demand in 2014 on weekdays
count    10884930.000000
mean            1.167551
std             3.658522
min             0.000000
25%             0.000000
50%             0.000000
75%             1.000000
max           606.000000
Name: demand, dtype: object
```

In [37]:
```
demand_year_snap__days(df_2015, 2015)
```

```
 demand in 2015 when event was on weekend and snap purchasing was allowed
count    6250450.000000
mean           1.318362
std            3.758913
min            0.000000
25%            0.000000
50%            0.000000
75%            1.000000
max          349.000000
Name: demand, dtype: object
************************************************************
 demand in 2015 on weekdays
count    10945910.000000
mean            1.235248
std             3.545168
min             0.000000
25%             0.000000
50%             0.000000
75%             1.000000
max           349.000000
Name: demand, dtype: object
```

In [38]:
```
demand_year_snap__days(df_2016, 2016)
```

```
 demand in 2016 when event was on weekend and snap purchasing was allowed
count    2591650.000000
mean           1.437597
std            3.768596
min            0.000000
25%            0.000000
50%            0.000000
75%            2.000000
max          248.000000
Name: demand, dtype: object
************************************************************
 demand in 2016 on weekdays
count    4268600.000000
mean           1.364482
std            3.586056
min            0.000000
25%            0.000000
50%            0.000000
75%            1.000000
max          248.000000
Name: demand, dtype: object
```

8 Final Observation

- CA and TX has more demand on product as compare to WI.
- We have very few days in a year when snap purchasing is allowed.
- There is negative trend in demand and sell price.
- We have higher demand on National and Religious event.
- We have more product in Food department and less product in Hobbies department.
- Saturday and Sunday have high demand.
- Cultural Event and National event have more demand.
- Snap_purchasing does not make any significant impact.
- If there is any event, demand is going to be increase certainly.
- There is no or very less sell on christmas in every year.
- There is seasonality in our data. Every year it repeats

9 Data Preperation