Причинно-следственный анализ

Артём Копань

Задача

Дана таблица с данными о продажах сети кафе в промежуток времени между 5 месяцами до и 13 месяцами после введения в продажу нового напитка M. Им собираются заменить напиток S.

Требуется:

1) Проверить наличие причинно-следственной связи между переменными S_sales и M_sales
2) Восстановить как можно больше аутентичных причинно-следственных связей между переменными

Используемые алгоритмы

- Constraint-based:
  - Алгоритм Петера-Кларка (PC)
  - Fast Causal Inference (FCI)
- Score-based:
  - Алгоритм жадного поиска эквивалентности (Greedy equivalence search, GES)
- Permutation-based
  - Алгоритм Greedy relaxation of the sparsest permutation (GRaSP)

Предобработка данных

```
df = df.drop(['Unnamed: 0', 'id'], axis=1)
```

```
df = df.fillna(0)
```

```
address_values = list(df.address)
address_values = list(map(str, address_values))
address_number_values = list(map(lambda address: int(address.split()[0]), address_values))
```

```
df['address'] = address_number_values
```

Алгоритм Петера-Кларка

Пороговое значение p-уровня значимости было выбрано равным 0.2 (иначе граф получался

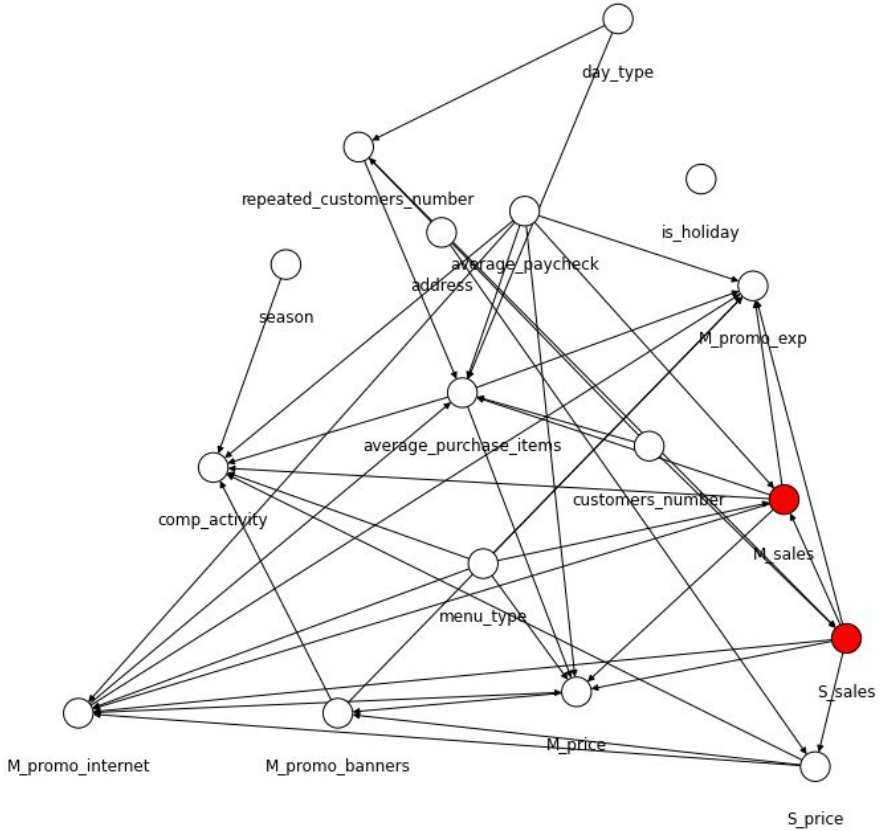
```
graph_pc_raw = pc(df.values, alpha=alpha, indep_test='fisherz')
```

```
Depth=10, working on node 16: 100% 17/17 [00:00<00:00, 1133.13it/s]
```

```
graph_pc_raw.to_nx_graph()

mapping = {node:i for node, i in enumerate(df.columns)}
mapping_r = {i:node for node, i in enumerate(df.columns)}

graph_pc = nx.relabel_nodes(graph_pc_raw.nx_graph, mapping)
```

```
edges_to_remove = [(u, v) for u, v in graph_pc.edges() if v == 'is_holiday']
edges_to_remove += [(u, v) for u, v in graph_pc.edges() if v == 'day_type']
edges_to_remove += [(u, v) for u, v in graph_pc.edges() if v == 'season']
edges_to_remove += [(u, v) for u, v in graph_pc.edges() if v == 'address']

graph_pc.remove_edges_from(edges_to_remove)
```

Алгоритм Петера-Кларка



Алгоритм Петера-Кларка

Связь между S_sales и M_sales есть.

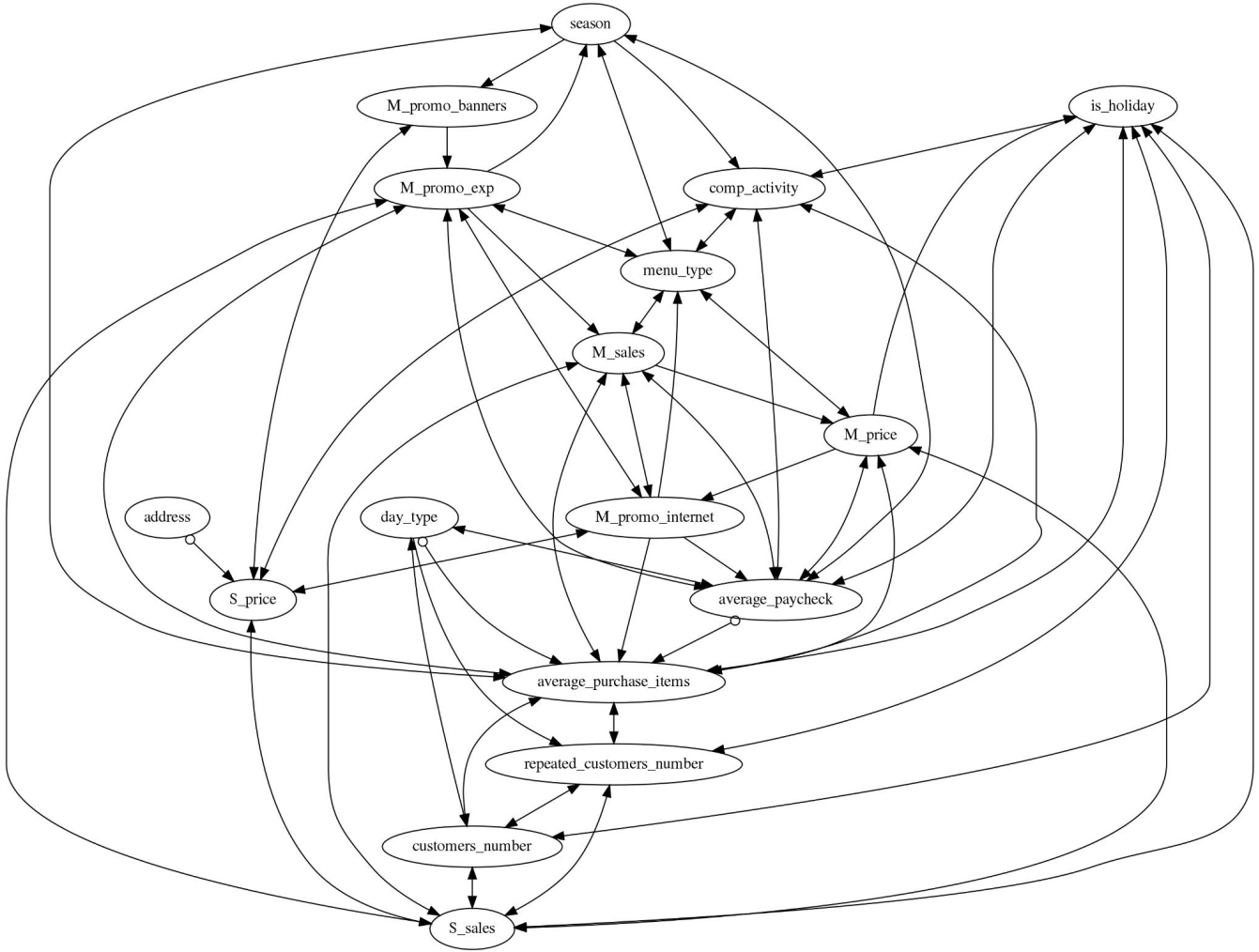
```
Оценка эффекта: -0.6276425022932131
```

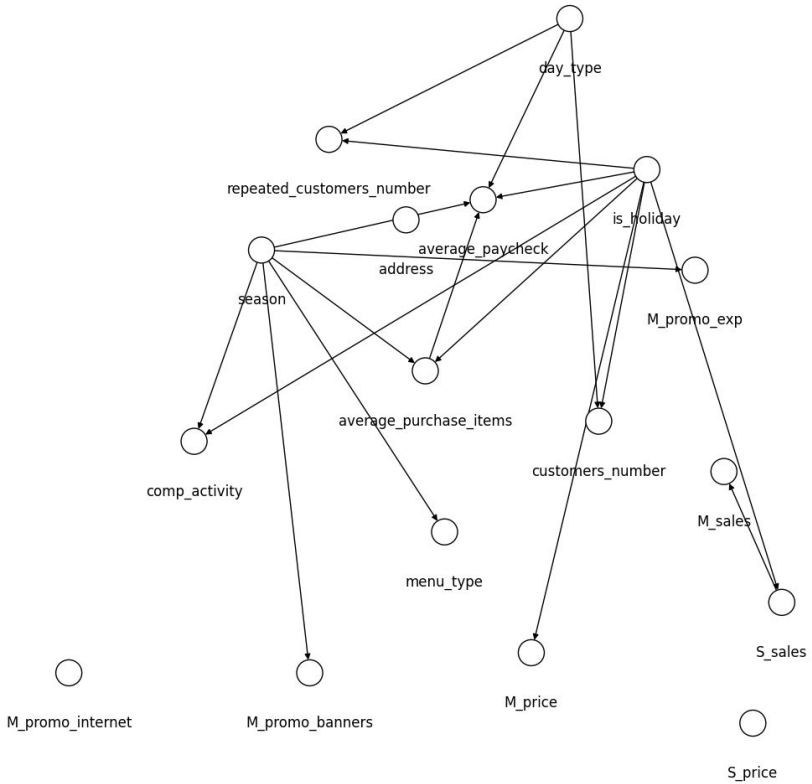
Алгоритм FCI

```
from causallearn.search.ConstraintBased.FCI import fci

fci_graph, fci_edges = fci(df.values, alpha=alpha, indep_test='fisherz')
```

```
  0%|          | 0/17 [00:00<?, ?it/s]

X10 --> X2
X3 --> X14
X4 --> X5
X6 --> X4
X4 --> X8
X8 --> X6
X6 --> X16
X7 --> X12
X16 --> X10
```

```
pdy = GraphUtils.to_pydot(fci_graph, labels=df.columns)
pdy.write_png('fci_graph.png')
```


season
M_promo_banners
is_holiday
M_promo_exp
comp_activity
menu_type
M_sales
M_price
address
day_type
M_promo_internet
S_price
average_paycheck
average_purchase_items
repeated_customers_number
customers_number
S_sales


day_type
repeated_customers_number
is_holiday
average_paycheck
address
season
M_promo_exp
average_purchase_items
customers_number
comp_activity
M_sales
menu_type
S_sales
M_price
M_promo_internet
M_promo_banners
S_price

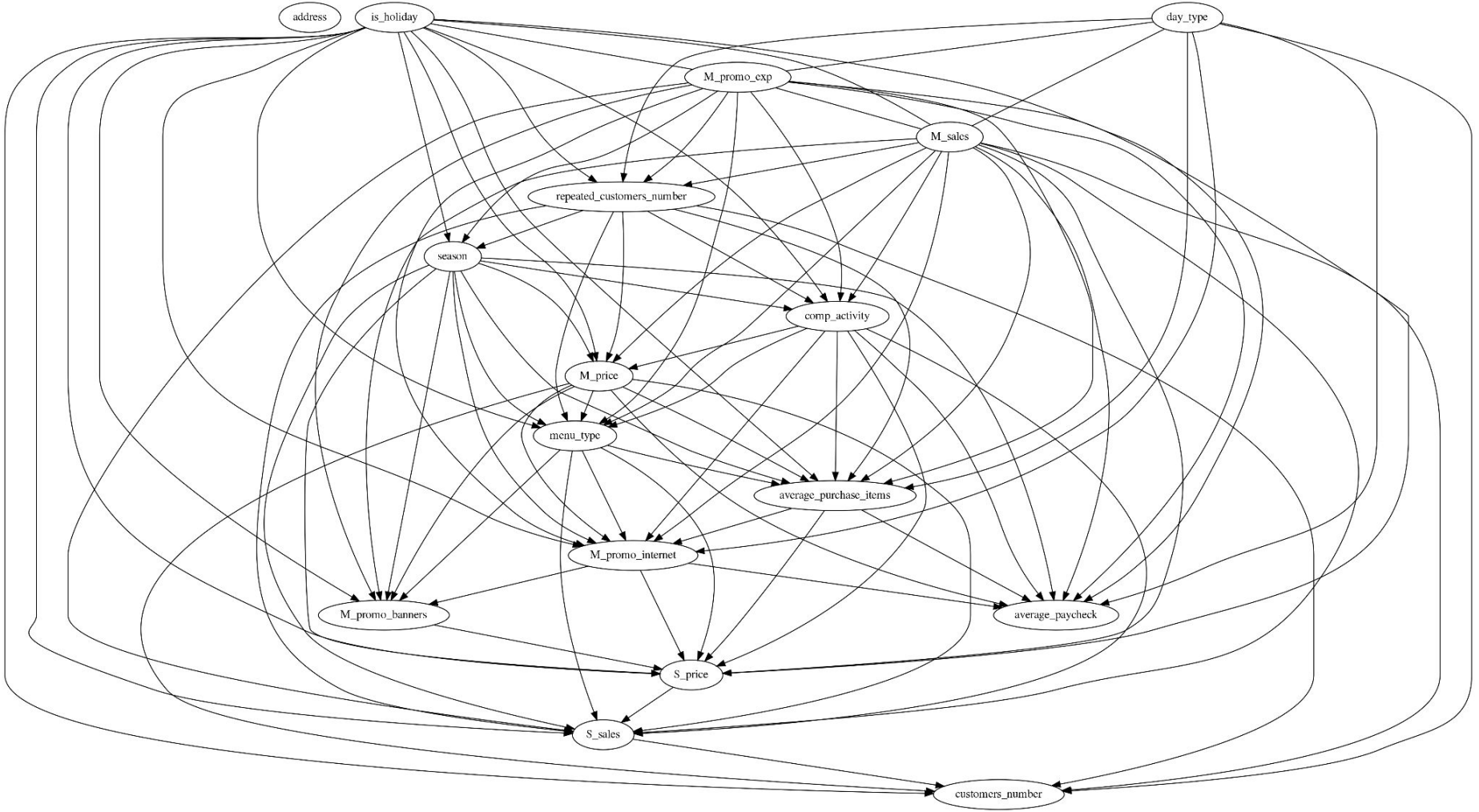
Алгоритм FCI

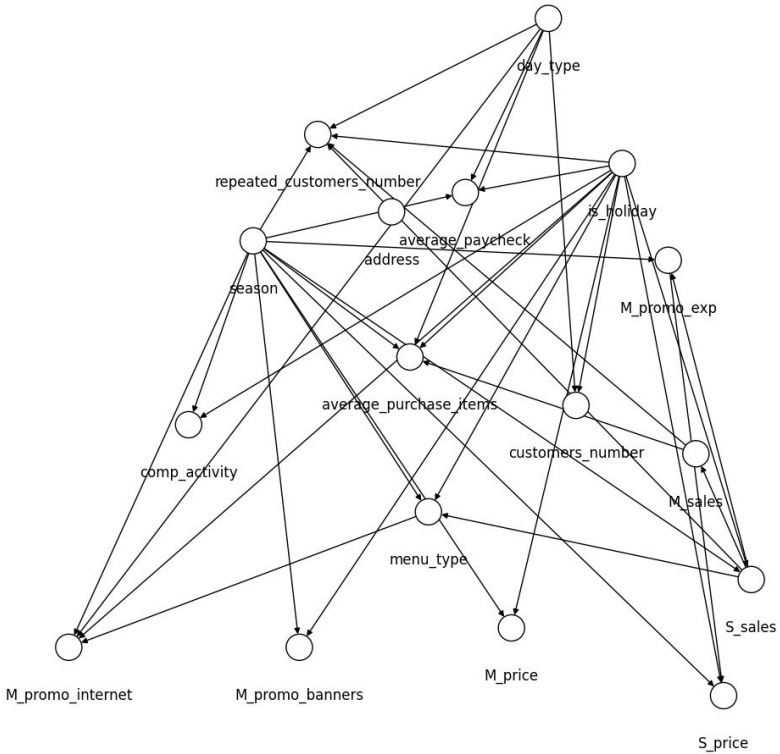
- Есть связь между S_sales и M_sales, но её характер из графа неясен (ребро двунаправленное)
- Оценка эффекта: -0.9311992463210004

Алгоритм GES

```
from causallearn.search.ScoreBased.GES import ges

graph_ges = ges(df, score_func='local_score_BIC')
```

```
import io
from causallearn.utils.GraphUtils import GraphUtils
import matplotlib.image as mpimg

pyd = GraphUtils.to_pydot(graph_ges['G'], labels=df.columns)
pyd.write_png('graph_ges.png')
```


address
is_holiday
day_type
M_promo_exp
M_sales
repeated_customers_number
season
comp_activity
M_price
menu_type
average_purchase_items
M_promo_internet
M_promo_banners
average_paycheck
S_price
S_sales
customers_number


day_type
repeated_customers_number
is_holiday
average_paycheck
address
season
M_promo_exp
average_purchase_items
customers_number
comp_activity
M_sales
menu_type
S_sales
M_price
M_promo_internet
M_promo_banners
S_price

Алгоритм GES

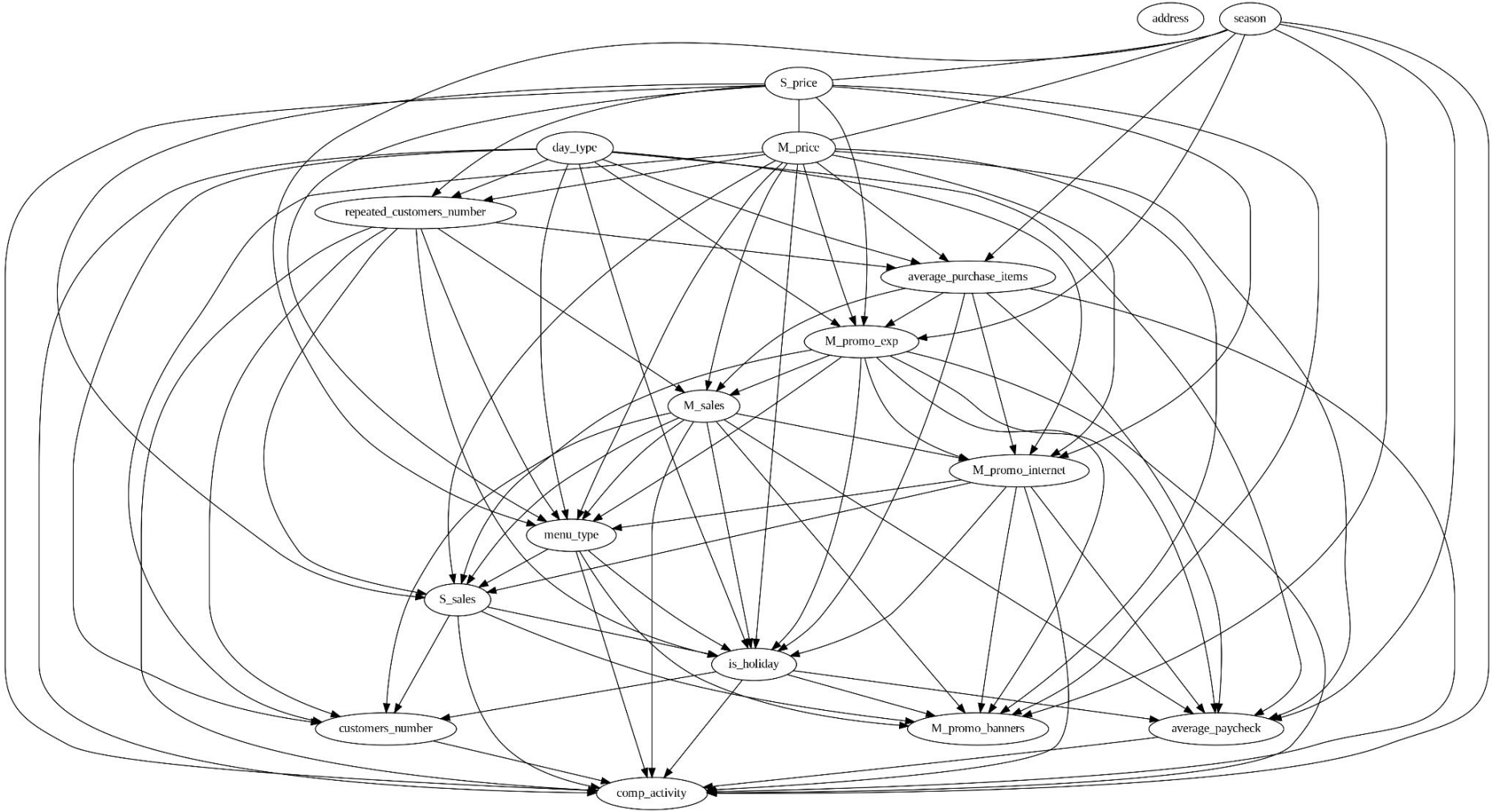
- Причинно-следственная связь между S_sales и M_sales есть
- Оценка эффекта: -0.9311992463210004

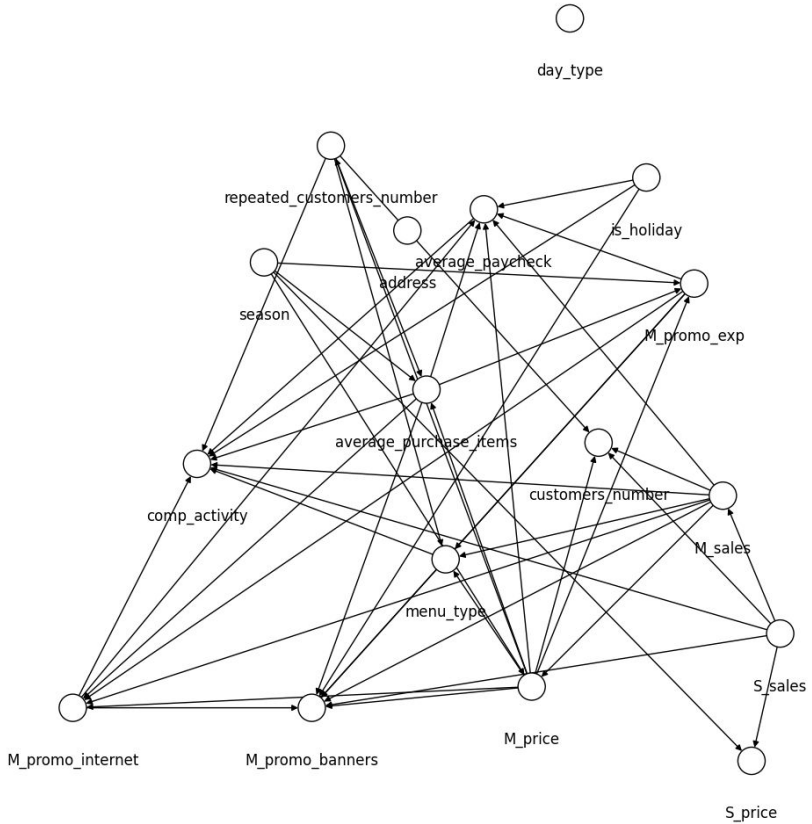
Алгоритм GRaSP

```
from causallearn.search.PermutationBased.GRaSP import grasp

G = grasp(df.values)
```

```
GRaSP edge count: 85
GRaSP completed in: 722.11s
```

```
import io
from causallearn.utils.GraphUtils import GraphUtils

pyd = GraphUtils.to_pydot(G, labels=df.columns)
pyd.write_png('graph_grasp.png')
```


address
season
S_price
day_type
M_price
repeated_customers_number
average_purchase_items
M_promo_exp
M_sales
M_promo_internet
menu_type
S_sales
is_holiday
customers_number
M_promo_banners
average_paycheck
comp_activity


day_type
repeated_customers_number
is_holiday
average_paycheck
address
season
M_promo_exp
average_purchase_items
customers_number
comp_activity
M_sales
menu_type
S_sales
M_price
M_promo_internet
M_promo_banners
S_price

Алгоритм GRaSP

- Имеется связь между S_sales и M_sales
- Оценка эффекта: -0.9311992463210004

Desbordante

Desbordante — это библиотека для поиска функциональных зависимостей в данных. Я решил применить её в данной задаче.

Находим функциональные зависимости при помощи Desbordante, составляем на их основе граф и применяем background knowledge.

Desbordante

```
import desbordante

TABLE = 'ci_data_internship_2.csv'

algo = desbordante.fd.algorithms.Default()
algo.load_data(table=(TABLE, ',', True))
algo.execute()
result = algo.get_fds()
print('FDs:')
for fd in result:
    print(fd)
```

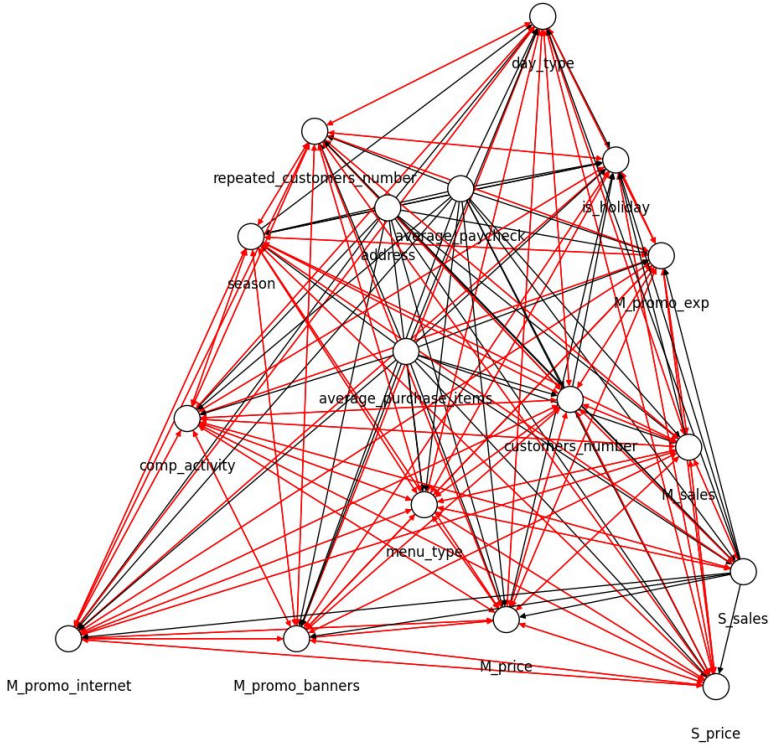
```
adjacency_matrix = [[0 for j in range(len(df.columns))] for i in range(len(df.columns))]
nodes = list(df.columns)
d = {nodes[i]: i for i in range(len(df.columns))}

for fd in result:
    lhs, rhs = str(fd).split(' -> ')
    lhs = lhs[1:-1].split()
    if rhs != ' ' and rhs != '' and len(lhs) != 0:
        for node in lhs:
            if node == '':
                continue
            adjacency_matrix[d[node]][d[rhs]] = 1

graph = nx.from_numpy_array(np.array(adjacency_matrix), create_using=nx.DiGraph)
```

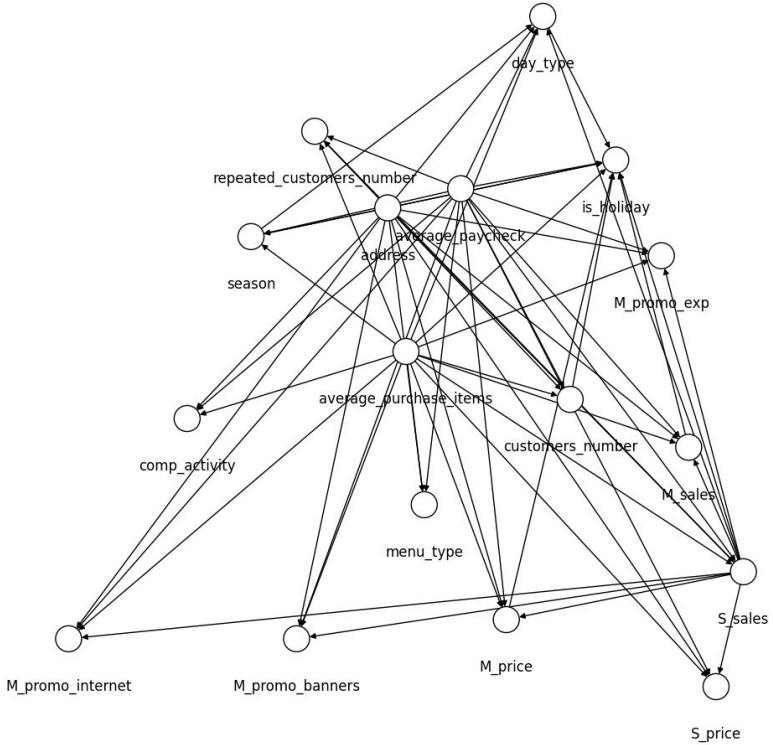
Desbordante

Background knowledge:

```
edges_to_remove = [(u, v) for u, v in graph.edges() if v == 'is_holiday']
edges_to_remove += [(u, v) for u, v in graph.edges() if v == 'day_type']
edges_to_remove += [(u, v) for u, v in graph.edges() if v == 'season']
edges_to_remove += [(u, v) for u, v in graph.edges() if v == 'address']
graph.remove_edges_from([('M_sales', 'S_sales')])
edges_to_remove = [(u, v) for u, v in graph.edges if (u, v) in get_non_directed_edges(graph)]

graph.remove_edges_from(edges_to_remove)
```

- Имеется связь между S_sales и M_sales
- `Оценка эффекта: -0.6085022649245388`


day_type
repeated_customers_number
is_holiday
average_paycheck
address
season
M_promo_exp
average_purchase_items
customers_number
comp_activity
M_sales
menu_type
S_sales
M_price
M_promo_internet
M_promo_banners
S_price


day_type
repeated_customers_number
is_holiday
average_paycheck
address
season
M_promo_exp
average_purchase_items
customers_number
comp_activity
M_sales
menu_type
S_sales
M_price
M_promo_internet
M_promo_banners
S_price

Выводы

- Все алгоритмы обнаруживают причинно-следственную связь между S_sales и M_sales
- Эта связь отрицательна, а оценка её эффекта при использовании разных алгоритмов колеблется в диапазоне [-0.932, -0.609].
- Алгоритмы в большинстве случаев хорошо восстанавливают причинно-следственные связи между различными переменными, но требуется удаление нелогичных связей с помощью background knowledge.