سه الگوریتم پیاده سازی شده است. الگوریتم ۳ و ۴ و ۵ موراد جدید هستند. در هر مورد برای سه استانه ۹۰ و ۹۵ و ۹۹ درصد نتایج ذخیره شده و در فایل های مرتبط ارائه شده اند. همچنین سه تصویر مربوط به ماتریس همبستگی برای هر حالت ارائه شده است که با پسوند cl برای کلاسترینگ و th برای نمایش موارد بالای ۷۰ درصد و cor برای کلیه موارد نام گذاری شده اند و پیشوند آن نیز مشخص کننده حالت ۹۰، ۹۹ و ۹۵ درصد حد استانه می باشد.

در هر مورد بردار مقدار مطلق همبستگی ها مشخص شده است که abs_correlation_df پیشوند ان است

در هر مورد بردار مقدار واقعی با پیشوند correlation_df مشخص شده است

در هر مورد ماتریس همبستگی با correlation_matrix_final_df مشخص شده است.

در هر حالت فایل خروجی با final_df نام گذاری شده است.

در هر حالت ستون هایی که هم بستگی پایین تر از استانه دارند (مشابه ۷۰ حالت) با no_corr_columns و ستون های انتخاب شده با selected_columns مشخص شده اند.

همچنین فایل هر الگوریتم در هر پوشه موجود است. در فایل مربوطه مقدار حد استانه به سادگی قابل تظیم توسط کاربر است.

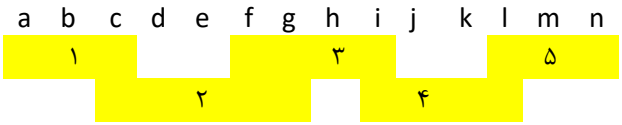
به طور کلی کدهای هر سه الگوریتم مشابه هستند اما الگوریتم ۵ داخل کدهایش موارد مربوط به ویژگی های مشابه یا ضریبی از یکدیگر را نیز دارد.

به دلیل تشابه الگوریتم ۳ تنها کامل توضیح داده میشد

سپس تفاوت های الگوریتم های ۴ و ۵ مجزا توضیح داده میشوند.

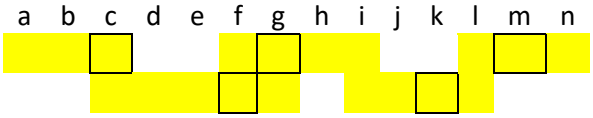
در مورد حلقه ی اعمال شده در الگوریتم ها نیز در زیر توضیح داده میشود:

فرض کنید حروف الفبا بیان گر ستون ها یا ویژگی ها باشند و خطوط زرد رنگ بیان گر گروه های همبستگی (نام هر گروه با یک عدد مشخص شده است):



یعنی a,b,c یک گروه و c,de,f,g یک گروه و الی اخر...

حال فرض کنید شکل زیر نماینده هر گروه را نمایش دهد:



یعنی c نماینده ی گروه a,b,c و f نماینده ی گروه c,d,e,f,g و g نماینده ی f,g,h,i,j و k و m نماینده ی دو گروه دیگر باشند.

بنابراین فیچرهای c,f,g,k,m انتخاب خواهند شد که به صورت زیر می‌باشند:

c f g k m

با توجه به گروه های همبستگی ابتدایی میدانیم f,g دارای همبستگی بالای استانه هستند چون در یک گروه قرار داشته اند همینطور c و f نیز در گروه دوم بوده اند و دارای همبستگی بالای حد استانه با یکدیگرند. پس فیچرهای انتخاب شده نیز همبستگی بالای حد استانه دارند که در شکل زیر نمایش داده شده است:

c f g k m

با توجه به مرحله پیشین دیدیم که در گروه دوم که شامل ویژگی c و f بود ویژگی f انتخاب شد پس f در مقایسه با c باید به عنوان نماینده انتخاب شود (پی ولیوی کمتری دارد) بنابراین مجدد f انتخاب میشود. همچنین بین f و g نیز g نماینده ی f میباشد. و انتخاب میشود و ستون های k و m نیز همبستگی ندارند و دست نخورده باقی میمانند. در پایان این حلقه ویژگی های انتخاب شده به صورت زیر میباشند:

f g k m

با توجه به توضیحات داده شده، در این مرحله نیز g انتخاب میشود.

f g k m

در نهایت سه ویژگی زیر پس از سه حلقه بررسی همبستگی ها انتخاب میشوند.

g k m

توضیح اینکه در حلقه ی نخست ویژگی c به عنوان نماینده ی a,b حضور پیدا میکند. در مرحله بعد با توجه به اینکه ویژگی f نماینده c میباشد (از گروه ۲، f به عنوان نماینده همه ویژگی ها مشخص میشود) در مرحله ی دوم f به جای c مینشیند و به حلقه ی سوم فرستاده میشود. در نهایت g نیز به عنوان نماینده ی f مشخص میشود. این توضیح در مورد سایر ویژگی های انتخاب شده و حذف شده نیز صدق میکند.

تفاوت کار فعلی با کار پیشین انجام شده توسط بنده این بود که در کار پیشین از pca استفاده میشد و در مرحله نخست با pca یک بردار جدید به جای کلیه بردار های یک گروه ساخته میشد و نیاز به اعمال حلقه نبود تا تمام ابعاد همبستگی ها کشف شود.

دیگر اینکه در الگوریتم سوم، از انتخاب تصادفی استفاده میشود، برای سادگی اولین ویژگی بین ویژگی هایی که پی ولیوی کمینه و برابر دارند انتخاب شدند.

در الگوریتم ۴ مواردی که پی ولیوی برابر وکمینه دارند همبستگی چک میشود و همبستگی بیشینه با لیبل معیار انتخاب است. اگر همبستگی نیز برابر باشد همه موارد پی ولیوی کمینه که همبستگی برابر دارند انتخاب میشوند.

در الگوریتم ۵ نیز مشابه الگوریتم ۴ رفتار میشود. با این تفاوت که در ابتدای برنامه کدهایی وجود دارند که اگر در جدول ویژگی هایی با پی ولیوی برابر وجود داشته باشند، کاربر را اگاه میسازند تا به طور دستی اقدام به انتخاب بین آن ها بکند و جدول را اصلاح کند.

توضیحات کدها:

کد فایل ۳:

فراخوانی کتابخانه و تعیین حد استانه و خواندن دیتاست با عنوان دیتافریم پانداس:

```
[ ] 1 import pandas as pd
    2
```

مقدار حد آستانه را وارد کنید یا اصلاح کنید

```
[ ] 1 threshold=0.9
```

```
1 df=pd.read_excel('data_u6.xlsx')
2 df.head()
```

	obj	class	f1	f2	f3	f5	f6	f7	f8	f9	...	f2541	f2545	f2546	f2549	f2550	f2551	f2553	f2555	f2556	f2557
0	1	1	0.778021	0.370202	40.129490	105.621967	75.000000	119.854078	120.623381	3406.541667	...	2709.139464	0.000382	7.584146	0.021298	0.002399	294890.01230	44.432247	36	1.0	3.0
1	2	1	0.821474	0.245085	22.974936	92.439169	72.090221	73.348483	99.206855	2486.166667	...	1797.152900	0.000725	7.572633	0.014414	0.003574	196790.75310	47.867235	28	1.0	5.0
2	3	1	0.223832	0.178680	24.034978	71.400280	42.296572	78.447435	94.323910	1298.166667	...	824.070347	0.001518	7.419445	0.012748	0.007548	93491.09881	58.766584	27	1.0	6.0
3	4	1	0.825111	0.059789	7.531322	64.761099	55.362442	6.082763	90.255194	70.250000	...	45.985507	0.015250	5.756863	0.015342	0.030374	28001.93655	106.215785	29	0.0	3.0
4	5	1	0.311822	0.134679	13.241971	50.447993	8.246211	17.464249	87.960218	479.125000	...	388.739696	0.002676	7.166653	0.010228	0.011863	165579.87010	116.725737	31	1.0	3.0

5 rows × 1336 columns

خواندن پی ولیو ها به صورت دیکشنری :

```
[ ] 1 def read_excel_to_dict(file_path):
    2     data = pd.read_excel(file_path)  # Assuming the first column is the index
    3     data_dict = data.iloc[0].to_dict()
    4     return data_dict
    5
    6 file_path = "not_sort2.xlsx"  # Replace with the actual file path
    7 data_dict = read_excel_to_dict(file_path)
    8
    9
```

تعریف توابع مورد استفاده:

تابع کاهش ستون ها:

```
#import random
def reduce_correlated_columns3(df, correlated_groups, p_values):
    reduced_columns = []
    selected_columns=[]
    p_val_columns=[]
    corr_columns=[]
    rand_columns=[]
    dff=df
    all_columns=[]
    for group in correlated_groups:
        candidates=[]
        fcandidates=[]
        min_pvalue_column,min_p_value = find_min_pvalue_column(group,
p_values)

        for col in group:
```

```
            all_columns.append(col)
            if p_values[col]==min_p_value:
              candidates.append(col)
          if len(candidates)==1:
            selected_columns.append(candidates[0])
            p_val_columns.append(candidates[0])

          if len(candidates)>1:
              #select on
              #random_index = random.randint(0, len(candidates) - 1)
              #random_candidate = candidates[random_index]
              #selected_columns.append(random_candidate)
              random_candidate=candidates[0]
              selected_columns.append(random_candidate)
              #print(candidates)
              #print("Randomly selected candidate:", random_candidate)
              rand_columns.append(random_candidate)
      print(len(set(selected_columns)),'out of ',len(set(all_columns)), '
columns have been selected')
      return set(selected_columns) , set(corr_columns),set(p_val_columns)
```

تابع تشخیص گروه های همبستگی:

```
def detect_correlation_groups(df, threshold):
    corr_matrix = df.corr(method='spearman').abs()
    column_correlation_groups = {}  # To store the correlated groups for
each column

    for column in corr_matrix.columns:
        correlated_group = set()

        if column not in column_correlation_groups:
            correlated_group.add(column)

            for other_column in corr_matrix.columns:
                if other_column != column:
                    if other_column in column_correlation_groups and
column in column_correlation_groups[other_column]:
                        correlated_group.update(column_correlation_groups[
other_column])

                    elif corr_matrix.loc[column, other_column] >
threshold:
                        correlated_group.add(other_column)
```

```
            if len(correlated_group) > 1:
                #print(correlated_group)
                for col in correlated_group:
                    column_correlation_groups[col] = correlated_group

    return list(column_correlation_groups.values())
```

تابع یافتن کمترین پی ولیو در هر گروه:

```
def find_min_pvalue_column(correlation_group, p_values):
    min_pvalue = np.inf
    min_pvalue_column = None
    for col in correlation_group:
        if p_values[col] < min_pvalue:
            min_pvalue = p_values[col]
            min_pvalue_column = col
    return min_pvalue_column,min_pvalue
```

اعمال الگوریتم:

```
from natsort import natsorted
import numpy as np
import pandas as pd

final_df=df
i=0
p_values_dict=data_dict
final_df_without_first_and_second_column=final_df.drop(columns=['class','o
bj'])
correlation_matrix =
final_df_without_first_and_second_column.corr(method='spearman')
correlation_array = correlation_matrix.to_numpy()
upper_triangular_matrix = np.triu(correlation_array, k=1)
upper_triangular_df = pd.DataFrame(upper_triangular_matrix,
columns=correlation_matrix.columns, index=correlation_matrix.index)
absolute_upper_triangular_df = np.abs(upper_triangular_df)
absolute_upper_triangular_df =
absolute_upper_triangular_df.applymap(lambda x: float(x))
absolute_upper_triangular_df =
absolute_upper_triangular_df.applymap(lambda x: np.nan if x < threshold
else 1)
```

شمارش تعداد همبستگی های موجود در دیتاست اولیه:

```
N=absolute_upper_triangular_df.count().sum()
```

شروع حلقه:

```
while N!=0:
  print('-------')

  print(N,' correlated feature exist')
```

استخراج گروه ها

```
  correlated_groups =
detect_correlation_groups(final_df_without_first_and_second_column,
threshold)
```

اعمال تابع کاهش ستون ها:

```
  columns,corr_columns,p_val_columns=reduce_correlated_columns3(final_df.c
opy(), correlated_groups, p_values_dict)
  column2remove=[]
  for group in correlated_groups:
    for col in group:
      column2remove.append(col)
  keptcolumns=[]
  for col in final_df.columns:
    if col not in set(column2remove):
      keptcolumns.append(col)
  removed_columns=[]
  for col in set(column2remove):
    if col not in columns:
      removed_columns.append(col)
```

ساخت دیتاست جدید از ستون های انتخاب شده و دست نخورده:

```
  final_columns=list(keptcolumns)+list(columns)
  sorted_list = sorted(final_columns)
  sorted_list = natsorted(sorted_list)
  item_to_move = 'obj'
  index_to_move = sorted_list.index(item_to_move)
  sorted_list.pop(index_to_move)
  sorted_list.insert(0, item_to_move)
  final_df = final_df[sorted_list]
  final_df_without_first_and_second_column=final_df.drop(columns=['class',
'obj'])
  correlation_matrix =
final_df_without_first_and_second_column.corr(method='spearman')
  correlation_array = correlation_matrix.to_numpy()
  upper_triangular_matrix = np.triu(correlation_array, k=1)
```

```
  upper_triangular_df = pd.DataFrame(upper_triangular_matrix,
columns=correlation_matrix.columns, index=correlation_matrix.index)
  absolute_upper_triangular_df = np.abs(upper_triangular_df)
  absolute_upper_triangular_df =
absolute_upper_triangular_df.applymap(lambda x: float(x))
  absolute_upper_triangular_df =
absolute_upper_triangular_df.applymap(lambda x: np.nan if x < threshold
else 1)
  old_N=N
```

شمارش مجدد ویژگی هایی که همبستگی دارند

```
  N=absolute_upper_triangular_df.count().sum()
  print('itteration number ',i)
  print('Number of correlated features reduced from ',old_N, ' to ',N)
  i=i+1
  #p_values_dict=
print('*******************************')
print('total itterations: ',i)
print(N,' correlated feature exist')

final_df.head()
```

بازگشت به شروع حلقه و تکرار عملیات روی دیتاست جدید با بررسی ستون هایی که همچنان همبستگی دارند، تا زمانی که هیچ همبستگی باقی نماند

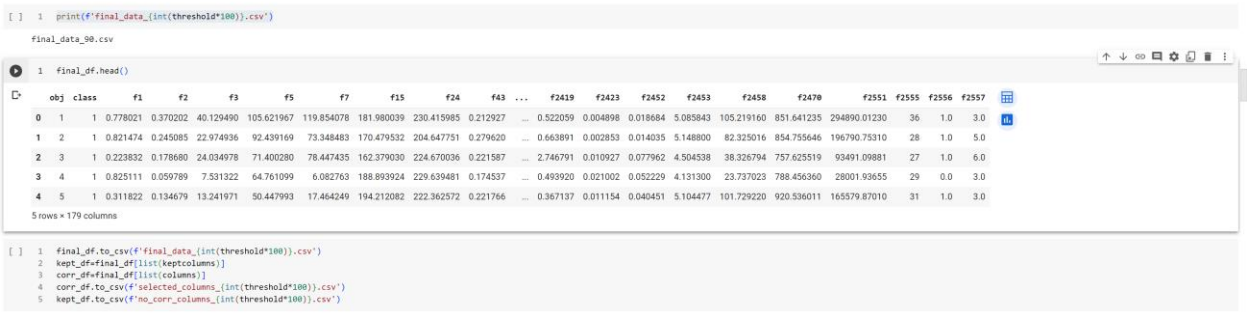
نمونه نتیجه:

```
-------
54295  correlated feature exist
190 out of  1261  columns have been selected
itteration number  0
Number of correlated features reduced from  54295  to  513
-------
513  correlated feature exist
46 out of  119  columns have been selected
itteration number  1
Number of correlated features reduced from  513  to  39
-------
39  correlated feature exist
10 out of  21  columns have been selected
itteration number  2
Number of correlated features reduced from  39  to  5
-------
5  correlated feature exist
3 out of  5  columns have been selected
itteration number  3
Number of correlated features reduced from  5  to  0
*******************************
total itterations:  4
0  correlated feature exist
```

	obj	class	f1	f2	f3	f5	f7	f15	f24	f43	...	f2419	f2423	f2452	f2453	f2458	f2470	f2551	f2555	f2556	f2557
0	1	1	0.778021	0.370202	40.129490	105.621967	119.854078	181.980039	230.415985	0.212927	...	0.522059	0.004898	0.018684	5.085843	105.219160	851.641235	294890.01230	36	1.0	3.0
1	2	1	0.821474	0.245085	22.974936	92.439169	73.348483	170.479532	204.647751	0.279620	...	0.663891	0.002853	0.014035	5.148800	82.325016	854.755646	196790.75310	28	1.0	5.0
2	3	1	0.223832	0.178680	24.034978	71.400280	78.447435	162.379030	224.670036	0.221587	...	2.746791	0.010927	0.077962	4.504538	38.326794	757.625519	93491.09881	27	1.0	6.0
3	4	1	0.825111	0.059789	7.531322	64.761099	6.082763	188.893924	229.639481	0.174537	...	0.493920	0.021002	0.052229	4.131300	23.737023	788.456360	28001.93655	29	0.0	3.0
4	5	1	0.311822	0.134679	13.241971	50.447993	17.464249	194.212082	222.362572	0.221766	...	0.367137	0.011154	0.040451	5.104477	101.729220	920.536011	165579.87010	31	1.0	3.0

5 rows × 179 columns

ذخیره نتایج:



تشکیل جدول همبستگی های دیتاست اولیه و سپس مرتب سازی بر اساس کاهشی و سپس ذخیره

```
import pandas as pd
df_without_first_and_second_column=df.drop(columns=['class','obj'])
# Assuming correlation_matrix is your calculated correlation matrix
correlation_matrix =
df_without_first_and_second_column.corr(method='spearman')

# Get the names of the features
feature_names = correlation_matrix.columns.tolist()

# Reshape the correlation matrix into a vector
correlation_vector = correlation_matrix.values.flatten()

# Create a DataFrame with feature names and correlation values
correlation_df = pd.DataFrame({
    'Feature 1': [feature_names[i] for i in range(len(feature_names)) for
_ in range(len(feature_names))],
    'Feature 2': feature_names * len(feature_names),
    'Correlation': correlation_vector
})
correlation_df = correlation_df[correlation_df['Feature 1'] <
correlation_df['Feature 2']]
correlation_df
correlation_df = correlation_df.sort_values(by='Correlation',
ascending=False)
correlation_df
correlation_df.to_csv('correlation_df_vec.csv')
```

همین جدول همبستگی نهایی با اندازه قدر مطلق:

```
abs_correlation_df = correlation_df.sort_values(by='Correlation', key=abs,
ascending=False)
abs_correlation_df
abs_correlation_df.to_csv('abs_correlation_df_vec.csv')
```

همبستگی های جدول نهایی به شکل ماتریس و معیار اسپیرمن

```
correlation_matrix_final_df = final_df.corr(method='spearman')
```

```
correlation_matrix_final_df.head()
correlation_matrix_final_df.to_csv(f'correlation_matrix_final_df_{int(thre
shold*100)}.csv')
```

جدول همبستگی های دیتاست نهایی و سپس مرتب سازی بر اساس کاهشی و سپس ذخیره

```
# Get the names of the features
feature_names = correlation_matrix_final_df.columns.tolist()

# Reshape the correlation matrix into a vector
correlation_vector = correlation_matrix_final_df.values.flatten()

# Create a DataFrame with feature names and correlation values
correlation_df = pd.DataFrame({
    'Feature 1': [feature_names[i] for i in range(len(feature_names)) for
_ in range(len(feature_names))],
    'Feature 2': feature_names * len(feature_names),
    'Correlation': correlation_vector
})
correlation_df = correlation_df[correlation_df['Feature 1'] <
correlation_df['Feature 2']]
correlation_df = correlation_df.sort_values(by='Correlation',
ascending=False)
correlation_df.to_csv(f'correlation_df_vec_{int(threshold*100)}.csv')
correlation_df
abs_correlation_df = correlation_df.sort_values(by='Correlation', key=abs,
ascending=False)
abs_correlation_df.to_csv(f'abs_correlation_df_vec_{int(threshold*100)}.cs
v')
abs_correlation_df
```

نمایش همبستگی های جدول نهایی

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

numeric_columns = final_df.select_dtypes(include=['float64', 'int64'])

correlation_matrix = numeric_columns.corr()

plt.figure(figsize=(30, 30))

sns.heatmap(correlation_matrix, center=0)

plt.title(f'Correlation Heatmap of final_df_{int(threshold*100)}')
```

```
plt.show()
```

نمایش تنها همبستگی های بالای ۷۰ درصد جدول نهایی

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming final_df is your dataframe
# If your dataframe contains non-numeric columns, you might want to select
only numeric columns for correlation
numeric_columns = final_df.select_dtypes(include=['float64', 'int64'])

# Calculate the correlation matrix
correlation_matrix = numeric_columns.corr()

# Set a correlation threshold
correlation_threshold = 0.7
correlation_matrix_filtered = correlation_matrix[correlation_matrix.abs()
> correlation_threshold]

# Set up the matplotlib figure with a larger size
plt.figure(figsize=(30, 30))

# Create a heatmap using seaborn
sns.heatmap(correlation_matrix_filtered, cmap='coolwarm', center=0)

# Add a title
plt.title(f'Correlation Heatmap (Threshold: {correlation_threshold})')

# Show the plot
plt.show()
```

نمایش همبستگی های کلاسترینگ شده جدول نهایی

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.cluster import hierarchy

# Assuming final_df is your dataframe
# If your dataframe contains non-numeric columns, you might want to select
only numeric columns for correlation
numeric_columns = final_df.select_dtypes(include=['float64', 'int64'])
```

```
# Calculate the correlation matrix
correlation_matrix = numeric_columns.corr()

# Calculate linkage matrix for hierarchical clustering
linkage = hierarchy.linkage(correlation_matrix, method='average')

# Set up the matplotlib figure with a larger size
plt.figure(figsize=(300, 300))

# Create a clustered heatmap using seaborn
sns.clustermap(correlation_matrix, method='average', cmap='coolwarm',
center=0, figsize=(20, 18), row_linkage=linkage, col_linkage=linkage)

# Add a title
plt.title('Clustered Correlation Heatmap of final_df')

# Show the plot
plt.show()
```

بخش های اضافه شده در فایل چهار:

```
import pandas as pd

# Load your DataFrame
# df = pd.read_csv("your_data.csv")

correlation_matrix = df.corr()

correlated_pairs = []

for i in range(len(correlation_matrix.columns)):
    for j in range(i + 1, len(correlation_matrix.columns)):
        if abs(correlation_matrix.iloc[i, j]) >= 1:
            correlated_pairs.append((correlation_matrix.columns[i],
correlation_matrix.columns[j]))
print("Highly Correlated Pairs:")
for pair in correlated_pairs:
    print(pair)
    print(pair[0],' =
',round(df[list(pair)[0]][0]/df[list(pair)[1]][0],2),' X ',pair[1])
    print(df[list(pair)].head())
    print('***************')
print(len(correlated_pairs),' correlated_pairs exist')
```

بخش بالا برای محاسبه مواردی است که همبستگی ۱ دارند و معیارش مشخص نشده که پیرسون است و خروجی ان ستون هایی که ضریب از همدیگرند میباشند.

همچنین الگوریتم کاهش ابعاد کمی تغییر داده شد که توضیحاتش داده شده است:

```
#import random
def reduce_correlated_columns(df, correlated_groups, p_values,
class_abs_correlation):
    reduced_columns = []
    selected_columns=[]
    p_val_columns=[]
    corr_columns=[]
    dff=df
    old_fcandidates=[]
    for group in correlated_groups:
        candidates=[]
        fcandidates=[]
        min_pvalue_column,min_p_value = find_min_pvalue_column(group,
p_values)

        for col in group:
```

```
            if p_values[col]==min_p_value:
              candidates.append(col)
          if len(candidates)==1:
            selected_columns.append(candidates[0])
            p_val_columns.append(candidates[0])

          if len(candidates)>1:
              max_cor_column,max_cor = find_max_corr_column(candidates,
abs_correlations_dict)

              for cl in candidates:

                if abs_correlations_dict[cl]==max_cor:
                  fcandidates.append(cl)
              if fcandidates!=[] and old_fcandidates!=fcandidates and
len(fcandidates)!=1 :
                print(fcandidates)
                print(df[fcandidates])
                print('***')


              old_fcandidates=fcandidates
              if len(fcandidates)==1:
                selected_columns.append(fcandidates[0])
                corr_columns.append(fcandidates[0])
              if len(fcandidates)>1:
                for i in fcandidates:

                  selected_columns.append(i)
                  corr_columns.append(i)


                  #print(i)

    print(len(set(selected_columns)), ' columns have been selected')
    return set(selected_columns) , set(corr_columns),set(p_val_columns),
```

در این موارد مشخص شده، ماکزیمم کورلیشن معیار انتخاب بین مواردی که پی ولیو کمینه و برابر دارند است. در مواردی که کورلیشن نیز برابر باشد، همه موارد انتخاب میشوند.

در کد اصلی الگوریتم نیز یک بخش اضافه شد که اگر تعداد ویژگی هایی که هبستگی دارند از تعدادی بیشتر امکان پذیر نبود حلقه متوقف شود. از انجایی که در این الگوریتم گفته شده اگر همبستگی ها برابر بود همگی را انتخاب کن، این امکان وجود داشت که در جدول نهایی مشاهده شود که همبستگی بین ویژگی هایی بالای استانه باشد ولی چون پی ولیو و همبستگی یکسان با کلاس دارند طبق این الگوریتم کاهش پذیر نباشند، بنابراین یک شرط توقف برای حلقه لازم میشد که در این مرحله به بدنه اصلی الگوریتم اضافه شد:

```
from natsort import natsorted
import numpy as np
import pandas as pd

final_df=df
i=0
p_values_dict=data_dict
final_df_without_first_and_second_column=final_df.drop(columns=['class','o
bj'])
correlation_matrix =
final_df_without_first_and_second_column.corr(method='spearman')
correlation_array = correlation_matrix.to_numpy()
upper_triangular_matrix = np.triu(correlation_array, k=1)
upper_triangular_df = pd.DataFrame(upper_triangular_matrix,
columns=correlation_matrix.columns, index=correlation_matrix.index)
absolute_upper_triangular_df = np.abs(upper_triangular_df)
absolute_upper_triangular_df =
absolute_upper_triangular_df.applymap(lambda x: float(x))
absolute_upper_triangular_df =
absolute_upper_triangular_df.applymap(lambda x: np.nan if x < threshold
else 1)
N=absolute_upper_triangular_df.count().sum()
while N!=0:
  print('-------------------------------------------------')

  print(N,' correlated feature exist')

  correlated_groups =
detect_correlation_groups(final_df_without_first_and_second_column,
threshold)
  columns,corr_columns,p_val_columns=reduce_correlated_columns(final_df.co
py(), correlated_groups, p_values_dict,abs_correlations_dict)
  column2remove=[]
  for group in correlated_groups:
    for col in group:
      column2remove.append(col)
  keptcolumns=[]
  for col in final_df.columns:
    if col not in set(column2remove):
      keptcolumns.append(col)
  removed_columns=[]
  for col in set(column2remove):
    if col not in columns:
      removed_columns.append(col)
  final_columns=list(keptcolumns)+list(columns)
```

```
  sorted_list = sorted(final_columns)
  sorted_list = natsorted(sorted_list)
  item_to_move = 'obj'
  index_to_move = sorted_list.index(item_to_move)
  sorted_list.pop(index_to_move)
  sorted_list.insert(0, item_to_move)
  final_df = final_df[sorted_list]
  final_df_without_first_and_second_column=final_df.drop(columns=['class',
'obj'])
  correlation_matrix =
final_df_without_first_and_second_column.corr(method='spearman')
  correlation_array = correlation_matrix.to_numpy()
  upper_triangular_matrix = np.triu(correlation_array, k=1)
  upper_triangular_df = pd.DataFrame(upper_triangular_matrix,
columns=correlation_matrix.columns, index=correlation_matrix.index)
  absolute_upper_triangular_df = np.abs(upper_triangular_df)
  absolute_upper_triangular_df =
absolute_upper_triangular_df.applymap(lambda x: float(x))
  absolute_upper_triangular_df =
absolute_upper_triangular_df.applymap(lambda x: np.nan if x < threshold
else 1)
  old_N=N
  N=absolute_upper_triangular_df.count().sum()
  print('itteration number ',i)
  print('Number of correlated features reduced from ',old_N, ' to ',N)
  i=i+1
  if N==old_N:
    print('number of correlated features can not reduce more, ')
    print('due to the existing of features with same p-value and same
correllation with each other')

    break
  #p_values_dict=
print('*******************************')
print('total itterations: ',i)
print(N,' correlated feature exist')
kept_df=final_df[list(keptcolumns)]
corr_df=final_df[list(columns)]
final_df.head()
```

فایل کد پنج:

در این فایل همبستگی های برابر ۱ هم از نظر پیرسون و هم اسپیر من بررسی شد که فیچرهایی که ضریبی از یکدیگرند کاملا شناسایی شوند:

پیرسون:

```
import pandas as pd

# Load your DataFrame
# df = pd.read_csv("your_data.csv")

#correlation_matrix = df.corr(method='spearman')
correlation_matrix = df.corr(method='pearson')

correlated_pairs = []

for i in range(len(correlation_matrix.columns)):
    for j in range(i + 1, len(correlation_matrix.columns)):
        if abs(correlation_matrix.iloc[i, j]) >= 1:
            correlated_pairs.append((correlation_matrix.columns[i],
correlation_matrix.columns[j]))
print("Highly Correlated Pairs:")
for pair in correlated_pairs:
    print(pair)
    print(pair[0],' =
',round(df[list(pair)[0]][0]/df[list(pair)[1]][0],2),' X ',pair[1])
    print(df[list(pair)].head())
    print('***************')
print(len(correlated_pairs),' correlated_pairs exist')
```

اسپیرمن:

```
import pandas as pd

# Load your DataFrame
# df = pd.read_csv("your_data.csv")

#correlation_matrix = df.corr(method='spearman')
correlation_matrix = df.corr(method='spearman')

correlated_pairs = []

for i in range(len(correlation_matrix.columns)):
    for j in range(i + 1, len(correlation_matrix.columns)):
        if abs(correlation_matrix.iloc[i, j]) >= 1:
            correlated_pairs.append((correlation_matrix.columns[i],
correlation_matrix.columns[j]))
```

```
print("Highly Correlated Pairs:")
for pair in correlated_pairs:
    print(pair)
    print(pair[0],' =
',round(df[list(pair)[0]][0]/df[list(pair)[1]][0],2),' X ',pair[1])
    print(df[list(pair)].head())
    print('***************')
print(len(correlated_pairs),' correlated_pairs exist')
```

در کد زیر در صورتی که مواردی پی ولیوی یکسان داشته باشند، تابعی تعریف شده که همبستگی شان را خروجی دهد:

```
import pandas as pd
import numpy as np

def print_correlation_with_class(p_values_dict, data_frame,Method):
    # Convert the dictionary to a list of tuples for easier sorting
    p_value_items = list(p_values_dict.items())

    # Sort the list of tuples based on p-values
    sorted_p_values = sorted(p_value_items, key=lambda x: x[1])

    # Initialize a dictionary to store columns with equal p-values
    equal_p_value_columns = {}

    # Group columns with equal p-values
    for i in range(len(sorted_p_values) - 1):
        if np.isclose(sorted_p_values[i][1], sorted_p_values[i + 1][1],
rtol=1e-6):
            col1 = sorted_p_values[i][0]
            col2 = sorted_p_values[i + 1][0]

            if col1 not in equal_p_value_columns:
                equal_p_value_columns[col1] = [col1, col2]
            else:
                equal_p_value_columns[col1].append(col2)

    # Calculate correlation with 'class' for columns with equal p-values
    for columns in equal_p_value_columns.values():
        print("Columns with equal p-values:", columns)
        for col in columns:
            correlation =
data_frame[col].corr(data_frame['class'],method=Method)
            print(f"{col} correlation with 'class': {correlation}")

# Example usage:.corr(method='spearman')
```

```
# Assuming you have a DataFrame called 'df' with columns and 'class'
column
# and a dictionary 'p_values' with column names as keys and p-values as
values
# print_correlation_with_class(p_values, df)
```

در اینجا همبستگی بر اساس اسپیرمن برای مواردی که پی ولیوی یکسان دارند خروجی داده میشود:

```
Df=pd.read_excel('data_u6.xlsx')

print_correlation_with_class(data_dict, Df,'spearman')
```

و در این جا بر اساس پیرسون:

```
print_correlation_with_class(data_dict, Df,'pearson')
```

فایل ۶: الگوریتم دیگری معرفی شد که سعی شد طبق سایت پیاده سازی شود، به دلیل نامشخص پیاده سازی آن خروجی نامناسب میداد و نافرجام ماند.

https://johfischer.com/2021/08/06/correlation-based-feature-selection-in-python-from-scratch/

این فایل با نام زیر موجود است:

6th_algorithm.ipynb

علت نافرجام ماندن پیاده سازی این کد این بود که مقدار زیر که خروجی بررسی pointbiserialr بین کلاس و ویژگی هاست،

```
    coeff = pointbiserialr( df[label], df[feature] )
```

و از کتابخانه ی زیر استفاده میکند،

```
from scipy.stats import pointbiserialr
```

مقدار خالی برمیگرداند:

```
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
nan
```

پیش بینی بنده این است که داده های مورد استفاده در این پژوهش مناسب استفاده از این روش نیستند.

بنده اطلاعای از روش و کدها ندارم و صرفا کدهای سایت را پیاده سازی کردم.