خواندند دیتافریم:

```
import pandas as pd

df=pd.read_excel('data_u.xlsx')
df.head()
```

تعداد ستون ها

```
len(df.columns)
```

حدف دو ستون اول

```
df_without_first_column = df.drop(columns=['obj'])
df_without_first_column.head()
df_without_first_and_second_column = df.drop(columns=['class','obj'])
df_without_first_and_second_column.head()
```

ماتریس همبستگی

```
correlation_matrix =
df_without_first_and_second_column.corr(method='spearman')
correlation_matrix
```

دستور ذخیره

```
correlation_matrix.to_csv('correlation_matrix.csv')
```

محاسبه همبستگی به صورتی که خواسته شده بود (بالا مثلثی)

```
import numpy as np

correlation_array = correlation_matrix.to_numpy()

upper_triangular_matrix = np.triu(correlation_array, k=1)

print(upper_triangular_matrix)
upper_triangular_df = pd.DataFrame(upper_triangular_matrix,
columns=correlation_matrix.columns, index=correlation_matrix.index)
upper_triangular_df
upper_triangular_df = upper_triangular_df.applymap(lambda x: np.nan if x
== 0 else x)
upper_triangular_df
upper_triangular_df.to_csv('upper_triangular_matrix.csv')
```

پایین مثلثی:

```
# ماتریس مثلثی پایین ماتریس کورلیشن را نگه داریم
lower_triangular_matrix = np.tril(correlation_array, k=-1)  # حفظ مثلث
پایین

# تبدیل ماتریس مثلثی پایین به دیتافریم پانداس و تعیین نام ستون‌ها
lower_triangular_df = pd.DataFrame(lower_triangular_matrix,
columns=correlation_matrix.columns, index=correlation_matrix.index)

lower_triangular_df = lower_triangular_df.applymap(lambda x: np.nan if x
== 0 else x)


lower_triangular_df
```

ماتریس کورلیشن قدر مطلق:

```
absolute_correlation_matrix = np.abs(correlation_matrix)
absolute_correlation_matrix
```

قدر مطلق بالا و پایین مثلثی:

```
absolute_lower_triangular_df = np.abs(lower_triangular_df)
absolute_lower_triangular_df
absolute_upper_triangular_df = np.abs(upper_triangular_df)
absolute_upper_triangular_df
```

نگهداری فقط ان هایی که همبستگی بالای حد استانه دارند

```
threshold = 0.9

# ایجاد یک دیتافریم جدید با مقادیر خالی برای مقادیر کمتر از حد استانه
absolute_lower_triangular_df_90 =
absolute_lower_triangular_df.applymap(lambda x: np.nan if x < threshold
else x)

(absolute_lower_triangular_df_90)
```

شمارش موارد باقی مانده و کل موارد و محاسبه نسبتشان

```
absolute_lower_triangular_df_90.count().sum()
df.count().sum()
absolute_lower_triangular_df_90.count().sum()/df.count().sum()
```

خواندن فایل پی ولیو ها به صورت دیکشنری:

```
import pandas as pd
```

```
def read_excel_to_dict(file_path):
    data = pd.read_excel(file_path)  # Assuming the first column is the
index
    data_dict = data.iloc[0].to_dict()
    return data_dict

file_path = "not_sort.xlsx"  # Replace with the actual file path
data_dict = read_excel_to_dict(file_path)

print(data_dict)
```

خواندن به صورت دیتافریم پانداس (جدول)

```
data = pd.read_excel(file_path )
data.head()
```

استخراج مواردی که همبستگی بالای حد استانه دارند و شمارش دفعاتی که هر ستون بیش از حد استانه با موارد دیگر همبستگی داشته است. (کد اولیه)

```
import pandas as pd

def detect_correlation_groups(df, threshold):
    corr_matrix = df.corr().abs()
    correlated_groups = []
    #checked_columns = set()
    column_count = {}  # To store the count of appearances in correlated
groups
    min_value_columns = {}  # To store columns with minimum values in each
group

    for column in corr_matrix.columns:
        #if column not in checked_columns:
        if column  in df.columns:

            correlated_group = set([column])
            #checked_columns.add(column)

            for other_column in corr_matrix.columns:
                if other_column != column: #and other_column not in
checked_columns:
                    if corr_matrix.loc[column, other_column] > threshold:
                        correlated_group.add(other_column)
                        #checked_columns.add(other_column)
```

```
            if len(correlated_group) > 1:
                correlated_groups.append(correlated_group)
                for col in correlated_group:
                    column_count[col] = column_count.get(col, 0) + 1

                # Find column(s) with minimum value in the correlated
group
                #min_value_col = min(correlated_group, key=lambda col:
data_dict[col])
                #min_value_columns[tuple(correlated_group)] =
min_value_col

    return correlated_groups, column_count


# Set your correlation threshold
threshold = 0.9  # Adjust the threshold as needed

# Call the function
correlated_groups, column_count = detect_correlation_groups(df, threshold)

# Print the results
print("Correlated Groups:")
#for group in correlated_groups:
#    print(group)

print("\nColumn Appearances:")
for column, count in column_count.items():
  if count>1:
    print(f"{column}: {count} times")
```

محاسبه کورلیشن هر ستون با کلاس

```
def calculate_column_correlations(df, target_column):
    correlations = {}

    for column in df.columns:
        if column != target_column:
            correlation = df[column].corr(df[target_column])
            correlations[column] = correlation

    return correlations

# Assuming your target column name is 'class'
```

```
correlations_dict = calculate_column_correlations(df, 'class')

for column, correlation in correlations_dict.items():
    print(f"Correlation of '{column}' with 'class': {correlation}")
```

ذخیره همبستگی با کلاس

```
# Convert correlations_dict to a DataFrame for easy saving
correlations_df = pd.DataFrame(list(correlations_dict.items()),
columns=['Column', 'Correlation'])
correlations_df = correlations_df.iloc[1:]

# Save correlations_df to a CSV file
correlations_df.to_csv('correlations.csv', index=False)
```

استفاده از قدرمطلق همبستگی با کلاس

```
def calculate_column_abs_correlations(df, target_column):
    correlations = {}

    for column in df.columns:
        if column != target_column:
            correlation = abs(df[column].corr(df[target_column]))  #
Calculate absolute correlation
            correlations[column] = correlation

    return correlations

# Assuming your target column name is 'class'
abs_correlations_dict = calculate_column_abs_correlations(df, 'class')

for column, correlation in abs_correlations_dict.items():
    print(f"Absolute Correlation of '{column}' with 'class':
{correlation}")
```

ذخیره:

```
abs_correlations_df = pd.DataFrame(list(abs_correlations_dict.items()),
columns=['Column', 'Correlation'])
abs_correlations_df = correlations_df.iloc[1:]


abs_correlations_df.to_csv('abs_correlations_dict.csv')
```

محاسبه گروه هایی که همبستگی بالای حد استانه با هم دارند (مورد استفاده در مراحل بعد **(تکامل یافته ی کد اولیه)**):

```
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())

# Example usage
#correlation_matrix =
df_without_first_and_second_column.corr(method='spearman')

correlated_groups =
detect_correlation_groups(df_without_first_and_second_column, threshold)
len(correlated_groups)
```

تغییر نام دیکشنری که پی ولیو ها در آن قرار دارد و خواندن کتابخوانه های مورد نیاز

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

```
p_values=data_dict
```

نوشتن تابعی که گروهی از ستون ها را دریافت کند و فایل پی ولیو ها را هم دریافت کند و مقدار پی ولیو مینیمم آن گروه را بدهد. در توابع بعدی استفاده شده است:

```
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
```

تابعی که مقدار ماکزیمم را بدهد، مشابه بالا ولی برای فایل کورلیشن ها:

```
def find_max_corr_column(candidates, abs_correlations_dict):
    max_corr = -np.inf
    max_corr_column = None
    for col in candidates:
        if p_values[col] > max_corr:
            max_corr = p_values[col]
            max_corr_column = col
    return max_corr_column,max_corr
```

انتخاب ستون های نماینده‌ی گروه های همبستگی بالای حد استانه

```
def reduce_correlated_columns(df, correlated_groups, p_values,
class_abs_correlation):
    reduced_columns = []
    selected_columns=[]
    dff=df
    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])
          #print(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 len(fcandidates)==1:
              selected_columns.append(fcandidates[0])
            if len(fcandidates)>1:
              for i in fcandidates:
                selected_columns.append(i)

            #print(min_p_value)
            #print(group)
            #print(candidates)
            #print(max_cor_column)
            #print(max_cor)
            #print('-----')
    print(selected_columns)
    print(len(selected_columns))
    print(len(set(selected_columns)))
    return set(selected_columns)



columns=reduce_correlated_columns(df.copy(), correlated_groups, p_values,
abs_correlations_dict)
```

مشخص کردن ستون هایی که باید حذف شوند:

```
column2remove=[]
for group in correlated_groups:
  for col in group:
    column2remove.append(col)
print(set(column2remove))
```

مشخص کردن ستون هایی که از ابتدا دست زده نشدند چون همبستگی کمتر از حد استانه داشتند

```
keptcolumns=[]
for col in df.columns:
  if col not in set(column2remove):
    keptcolumns.append(col)
print(keptcolumns)
```

ادغام لیست ستون های دست زده نشده از ابتدا و ستون های نماینده‌ی موارد دارای همبستگی بالای حد استانه

```
final_columns=list(keptcolumns)+list(columns)
print(final_columns)
```

تشکیل جدول نهایی

```
final_df = df[final_columns]
final_df.head()
```

	obj	class	f100	f108	f116	f117	f131	f145	
0	1	1	6.099861	-52.035064	-12.935353	-11.163923	0.206524	0.597291	4
1	2	1	6.117552	-38.378041	-10.285282	-8.294217	0.324859	0.582532	5
2	3	1	6.352797	-38.115952	-8.392629	-6.119442	0.285934	0.633064	5
3	4	1	4.909277	-52.328724	-7.105880	-0.210757	-0.051216	0.974662	3
4	5	1	5.697579	-44.477188	-11.522002	-11.732962	0.118117	0.489198	4

ترتیب نام ستون ها کمی به هم ریخته بود بعد از این تلاش شد ترتیب اصلاح شود

مرتب کردن از نظر رشته ای

```
sorted_list = sorted(final_columns)

print(sorted_list)  # Output: [1, 2, 3, 4, 5]
```

انتقال ابجکت به ابتدای لیست ستون ها

```
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)
print(sorted_list)
```

ساخت مجدد جدول

```
final_df = df[sorted_list]
final_df.head()
```

	obj	class	f100	f1009	f1010	f1034	f1038
0	1	1	6.099861	4318.645073	0.005769	1006.228882	0.002118
1	2	1	6.117552	3954.552290	0.008657	925.471396	0.003007
2	3	1	6.352797	8806.721810	0.014633	1041.271212	0.003231
3	4	1	4.909277	646.505882	0.105870	331.154930	0.073884
4	5	1	5.697579	4449.770751	0.007424	1223.866848	0.005746

هنوز به هم ریخته است

مرتب کردن به صورت الفبایی

```
from natsort import natsorted

sorted_list = natsorted(sorted_list)

print(sorted_list)
```

انتقال ابجکت به ابتدای لیست

```
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)
print(sorted_list)
```

ساخت جدول نهایی

```
final_df = df[sorted_list]
final_df.head()
```

	obj	class	f2	f7	f15	f24	f41	f43
0	1	1	0.370202	119.854078	181.980039	230.415985	16.006034	0.212927
1	2	1	0.245085	73.348483	170.479532	204.647751	9.273114	0.279620
2	3	1	0.178680	78.447435	162.379030	224.670036	13.011379	0.221587
3	4	1	0.059789	6.082763	188.893924	229.639481	18.266011	0.174537
4	5	1	0.134679	17.464249	194.212082	222.362572	11.370417	0.221766

5 rows × 250 columns

ذخیره:

```
final_df.to_csv('final_data.csv')
```

موفق باشد.