Appendix B: Data analysis script

Data processing

Import data

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Import libraries
import pandas as pd
import numpy as np
```

```
# Load the dataset
folder_path = '/content/drive/MyDrive/Work space/CHI paper/Data analysis/'

file_path1 = folder_path + '1st_study.csv'
df1 = pd.read_csv(file_path1)

file_path2 = folder_path + '2rd_study.csv'
df2 = pd.read_csv(file_path2)
display(df2.head())
```

	CODE	AGE	N_CHILDREN_ADOLESCENTS	SUS_1	SUS_2	SUS_3	SUS_4	SUS_5	SUS_6	SUS_7	...	PRESENCE_5	PRESENCE_6	INVOLVEMENT_7	INVOLVEMENT_8	INVOLVEMENT_9	INVOLVEMENT_10	REALISM_11	RE/
0	AABA1702	15.17	2	0	2	2	4	1	3	3	...	2	3	2	6	6	4	3	
1	AABA2112	10.39	1	1	3	4	3	3	3	3	...	1	1	4	4	2	1	4	
2	AACI2309	14.55	2	0	4	3	1	4	2	4	...	2	1	3	1	2	1	2	
3	AAFI3110	10.50	1	2	3	1	3	3	2	3	...	6	5	4	3	4	6	3	
4	AAFO1112	11.37	1	4	2	4	3	3	3	4	...	4	7	7	2	7	7	7	

5 rows × 43 columns

```
# Add IS_CHILDREN column
df1['IS_CHILDREN'] = df1['N_CHILDREN_ADOLESCENTS'].apply(lambda x: True if x == 1 else False)
df1 = df1.drop('N_CHILDREN_ADOLESCENTS', axis=1)

df2['IS_CHILDREN'] = df2['N_CHILDREN_ADOLESCENTS'].apply(lambda x: True if x == 1 else False)
df2 = df2.drop('N_CHILDREN_ADOLESCENTS', axis=1)
```

Calculation

```
# Calculate relevant variables for Study 1
df1_processed = pd.DataFrame()
df1_processed['Age'] = df1['Age']
df1_processed['is_children'] = df1['IS_CHILDREN']

#Perception
df1_processed['Immersion'] = df1['IMMERSION_VR']
df1_processed['Experienced Realism'] = df1['PERCEPTIONOFREALISM_VR']
df1_processed['Involvement'] = df1['INVOLVEMENT_VR']
df1_processed['Presence'] = df1['PRESENCE_VR']

#Exclusion
df1_processed['Exclusion'] = df1['MANIPULATION_CHECK_1_VR'] + df1['MANIPULATION_CHECK_2_VR']

df1_processed.head()
```

	Age	is_children	Immersion	Experienced Realism	Involvement	Presence	Exclusion
0	13.250000	False	1	1	1	1	10
1	11.250000	True	2	3	1	3	9
2	12.750000	True	1	1	1	1	10
3	14.416667	False	2	1	1	2	8
4	9.583333	True	3	3	1	3	9

Next steps: Generate code with df1_processed | View recommended plots | New interactive sheet

```
# Calculate relevant variables for Study 2
df2_processed = pd.DataFrame()
df2_processed['Age'] = df2['AGE']
df2_processed['is_children'] = df2['IS_CHILDREN']

#Perception (IPQ)
presence_columns = ['PRESENCE_1','PRESENCE_2', 'PRESENCE_4', 'PRESENCE_5', 'PRESENCE_6']
involvement_columns = ['INVOLVEMENT_7', 'INVOLVEMENT_8', 'INVOLVEMENT_9', 'INVOLVEMENT_10']
experienced_realism_columns = ['REALISM_3', 'REALISM_11', 'REALISM_12', 'REALISM_13', 'REALISM_14']

reverse_score_columns = ['PRESENCE_4', 'INVOLVEMENT_7', 'INVOLVEMENT_9', 'REALISM_3']
for col in reverse_score_columns:
    if col in df2.columns:
        df2[col] = 7 + 1 - df2[col]
    else:
        print(f"Warning: Column '{col}' not found in DataFrame. Skipping reverse scoring for this column.")
df2_processed['Experienced Realism'] = df2[experienced_realism_columns].sum(axis=1)
df2_processed['Involvement'] = df2[involvement_columns].sum(axis=1)
df2_processed['Presence'] = df2[presence_columns].sum(axis=1)

#System Usability
sus_columns = [f'SUS_{i}' for i in range(1, 11)]
temp = 0
```

```
for i, col in enumerate(sus_columns):
    if (i + 1) % 2 != 0: # Odd-numbered questions
        temp += df2[col] - 1
    else: # Even-numbered questions
        temp += 5 - df2[col]
df2_processed['System Usability'] = temp * 2.5

#Simulator Sickness
nausea_columns = ['DISCOMFORT', 'SALIVATION','SWEAT','NAUSEA','CONCENTRATION']
oculomotor_columns = ['DISCOMFORT', 'TIREDNESS','HEADACHE', 'EYESTRAIN', 'FOCUSING','CONCENTRATION', 'BLURRY']
disorientation_columns = ['FOCUSING','NAUSEA','HEADHEAVINESS', 'BLURRY','OPENEDEYESDIZZINESS', 'CLOSEDEYESDIZZINESS', 'VERTIGO' ]
df2_processed['Simulator Sickness'] = (df2[nausea_columns].sum(axis=1) + df2[oculomotor_columns].sum(axis=1) + df2[disorientation_columns].sum(axis=1)) * 3.74

#Exclusion
df2_processed['Exclusion'] = df2['MANIPULATION_CHECK_1'] + df2['MANIPULATION_CHECK_2']

df2_processed.head()
```

	Age	is_children	Experienced Realism	Involvement	Presence	System Usability	Simulator Sickness	Exclusion
0	15.17	False	14	18	18	35.0	41.14	4
1	10.39	True	12	15	18	47.5	67.32	10
2	14.55	False	14	13	15	60.0	108.46	8
3	10.50	True	18	17	22	47.5	0.00	10
4	11.37	True	30	11	26	62.5	59.84	2

Next steps: Generate code with df2_processed | View recommended plots | New interactive sheet

```
df1_variables_list = df1_processed.select_dtypes(include=np.number).columns.tolist()
df1_variables_list.remove('Age')
df2_variables_list = df2_processed.select_dtypes(include=np.number).columns.tolist()
df2_variables_list.remove('Age')
```

Mean comparison

Define function

```
from scipy.stats import shapiro, ttest_ind, mannwhitneyu
from statsmodels.stats.power import TTestIndPower
analysis = TTestIndPower()

def compare_means_func(study: int, df: pd.DataFrame, column_name: str):
    alpha = 0.05

    # Align the indices of the column and group_column
    column = df[column_name]
    aligned_column, aligned_group_column = column.align(df['is_children'], join='inner')
```

```
# Get the two unique group values (assuming True is children and False is adolescents)
group1_label = 'children'
group2_label = 'adolescents'

# Separate the column values based on the group column using the aligned index
group1_values = aligned_column.loc[aligned_group_column == True].dropna()  # True corresponds to children
group2_values = aligned_column.loc[aligned_group_column == False].dropna() # False corresponds to adolescents

n1, n2 = len(group1_values), len(group2_values)

# Check normality for each group using Shapiro-Wilk test
shapiro_pvalue_1 = shapiro(group1_values).pvalue
shapiro_pvalue_2 = shapiro(group2_values).pvalue

# Determine the appropriate statistical test based on normality
if shapiro_pvalue_1 > alpha and shapiro_pvalue_2 > alpha:
    method = "Independent samples t-test (Welch's)"
    statistic, p_value = ttest_ind(group1_values, group2_values, equal_var=False)

    dof = n1 + n2 - 2
    pooled_std = np.sqrt(((n1-1)*np.var(group1_values, ddof=1) + (n2-1)*np.var(group2_values, ddof=1)) / dof)
    effect = (np.mean(group1_values) - np.mean(group2_values)) / pooled_std
    power = analysis.solve_power(effect_size=effect, nobs1=len(group1_values), alpha=alpha, ratio=len(group2_values)/len(group1_values))

else:
    method = "Mann-Whitney U test"
    statistic, p_value = mannwhitneyu(group1_values, group2_values)

    mean_U = n1 * n2 / 2
    std_U = np.sqrt(n1 * n2 * (n1 + n2 + 1) / 12)
    z = (statistic - mean_U) / std_U
    effect = z / np.sqrt(n1 + n2)

    simulations = 5000
    count_sig = 0
    for _ in range(simulations):
        x_sample = np.random.choice(group1_values, n1, replace=True)
        y_sample = np.random.choice(group2_values, n2, replace=True)
        stat, p = mannwhitneyu(x_sample, y_sample)
        if p < alpha:
            count_sig += 1
    power = count_sig / simulations

results = {
    'study': study,
    'variable': column_name,
    'children_mean': np.mean(group1_values),
    'children_stdDev': np.std(group1_values),
    'adolescents_mean':  np.mean(group2_values),
    'adolescents_stdDev': np.std(group2_values),
    'effect_size_d': effect,
    'power': power,
    'method': method,
    'statistic': statistic,
    'pValue': p_value,
    'is_significant': 'Yes' if p_value < alpha else 'No'
```

```
    }

    return results
```

Calculate results

```
mean_comparision_results = pd.DataFrame()
#Loop through all variables to run the function
for col in df1_variables_list:
    comparison_result = compare_means_func(1, df1_processed, col)
    mean_comparision_results = pd.concat([mean_comparision_results, pd.DataFrame([comparison_result])], ignore_index=True)
for col in df2_variables_list:
    comparison_result = compare_means_func(2, df2_processed, col)
    mean_comparision_results = pd.concat([mean_comparision_results, pd.DataFrame([comparison_result])], ignore_index=True)

display(mean_comparision_results)
```

	study	variable	children_mean	children_stdDev	adolescents_mean	adolescents_stdDev	effect_size_d	power	method	statistic	pValue	is_significant
0	1	Immersion	3.394737	1.424217	2.310811	1.218391	0.364935	0.997600	Mann-Whitney U test	4001.000000	4.819993e-06	Yes
1	1	Experienced Realism	3.289474	1.403152	2.175676	1.200879	0.375524	0.998400	Mann-Whitney U test	4035.500000	2.469854e-06	Yes
2	1	Involvement	2.394737	1.367662	1.797297	0.999726	0.208096	0.779400	Mann-Whitney U test	3490.000000	7.178250e-03	Yes
3	1	Presence	3.723684	1.333759	2.513514	1.317689	0.405910	0.999600	Mann-Whitney U test	4134.500000	3.803798e-07	Yes
4	1	Exclusion	6.710526	3.029746	8.135135	2.170592	-0.213467	0.772600	Mann-Whitney U test	2116.500000	7.355333e-03	Yes
5	2	Experienced Realism	18.670886	5.406797	16.404762	3.830078	0.195717	0.717400	Mann-Whitney U test	4070.500000	1.225154e-02	Yes
6	2	Involvement	17.886076	4.319480	16.226190	3.406406	0.182712	0.664200	Mann-Whitney U test	4020.500000	1.928485e-02	Yes

Next steps: Generate code with `mean_comparision_results` | View recommended plots | New interactive sheet

Correlation

Define pairwise function

```
from scipy.stats import shapiro, pearsonr, spearmanr

def correlation_func(study:int, dataset1: pd.Series, dataset2: pd.Series, variable_name1: str, variable_name2: str):
  alpha = 0.05
```

```
  # Check normality for each group using Shapiro-Wilk test
  shapiro_pvalue_1 = shapiro(dataset1).pvalue
  shapiro_pvalue_2 = shapiro(dataset2).pvalue

  # Determine the appropriate statistical test based on normality
  if shapiro_pvalue_1 > alpha and shapiro_pvalue_2 > alpha:
      method = "Pearson's correlation"
      statistic, p_value = pearsonr(dataset1, dataset2)
  else:
      method = "Spearman's correlation"
      statistic, p_value = spearmanr(dataset1, dataset2)

  results = {
      'study': study,
      'variable1': variable_name1,
      'variable2': variable_name2,
      'method': method,
      'pValue': p_value,
      'statistic': statistic,
      'is_significant': 'Yes' if p_value < alpha else 'No'
  }

  return results
```

Run Pairwise Correlations

```
paiwise_correlation_results = pd.DataFrame()

# Pairwise correlation for df1_processed
df1_numerical_vars = df1_processed.select_dtypes(include=np.number).columns.tolist()
for i in range(len(df1_numerical_vars)):
    for j in range(i + 1, len(df1_numerical_vars)):
        var1 = df1_numerical_vars[i]
        var2 = df1_numerical_vars[j]
        temp = correlation_func(1, df1_processed[var1], df1_processed[var2], var1, var2)
        paiwise_correlation_results = pd.concat([paiwise_correlation_results, pd.DataFrame([temp])], ignore_index=True)

# Pairwise correlation for df2_processed
df2_numerical_vars = df2_processed.select_dtypes(include=np.number).columns.tolist()
for i in range(len(df2_numerical_vars)):
    for j in range(i + 1, len(df2_numerical_vars)):
        var1 = df2_numerical_vars[i]
        var2 = df2_numerical_vars[j]
        temp = correlation_func(2, df2_processed[var1], df2_processed[var2], var1, var2)
        paiwise_correlation_results = pd.concat([paiwise_correlation_results, pd.DataFrame([temp])], ignore_index=True)

display(paiwise_correlation_results)
```

	study	variable1	variable2	method	pValue	statistic	is_significant
0	1	Age	Immersion	Spearman's correlation	9.764005e-06	-0.352379	Yes
1	1	Age	Experienced Realism	Spearman's correlation	7.945426e-06	-0.355678	Yes
2	1	Age	Involvement	Spearman's correlation	1.713640e-04	-0.302153	Yes
3	1	Age	Presence	Spearman's correlation	2.419621e-08	-0.436151	Yes
4	1	Age	Exclusion	Spearman's correlation	4.036420e-04	0.285214	Yes
5	1	Immersion	Experienced Realism	Spearman's correlation	7.441654e-31	0.771418	Yes
6	1	Immersion	Involvement	Spearman's correlation	1.362790e-05	0.346963	Yes
7	1	Immersion	Presence	Spearman's correlation	1.024456e-23	0.703551	Yes
8	1	Immersion	Exclusion	Spearman's correlation	2.171341e-03	-0.248458	Yes
9	1	Experienced Realism	Involvement	Spearman's correlation	2.837772e-04	0.292307	Yes
10	1	Experienced Realism	Presence	Spearman's correlation	1.065843e-14	0.577183	Yes
11	1	Experienced Realism	Exclusion	Spearman's correlation	9.358690e-03	-0.211543	Yes
12	1	Involvement	Presence	Spearman's correlation	3.286277e-08	0.432388	Yes
13	1	Involvement	Exclusion	Spearman's correlation	6.934464e-19	-0.643245	Yes
14	1	Presence	Exclusion	Spearman's correlation	2.136272e-03	-0.248839	Yes
15	2	Age	Experienced Realism	Spearman's correlation	2.237750e-03	-0.237806	Yes
16	2	Age	Involvement	Spearman's correlation	5.437110e-03	-0.216813	Yes
17	2	Age	Presence	Spearman's correlation	1.331701e-01	-0.118120	No
18	2	Age	System Usability	Spearman's correlation	1.641087e-02	-0.187725	Yes
19	2	Age	Simulator Sickness	Spearman's correlation	1.526918e-01	0.112523	No
20	2	Age	Exclusion	Spearman's correlation	1.823686e-01	0.104967	No
21	2	Experienced Realism	Involvement	Spearman's correlation	1.877143e-04	0.288522	Yes
22	2	Experienced Realism	Presence	Spearman's correlation	1.774861e-12	0.516106	Yes
23	2	Experienced Realism	System Usability	Spearman's correlation	7.163455e-07	0.376749	Yes
24	2	Experienced Realism	Simulator Sickness	Spearman's correlation	6.277385e-01	-0.038262	No
25	2	Experienced Realism	Exclusion	Spearman's correlation	2.058079e-01	-0.099616	No
26	2	Involvement	Presence	Spearman's correlation	3.527891e-07	0.386279	Yes
27	2	Involvement	System Usability	Spearman's correlation	8.120078e-02	0.136992	No
28	2	Involvement	Simulator Sickness	Spearman's correlation	3.150510e-02	-0.168541	Yes
29	2	Involvement	Exclusion	Spearman's correlation	4.435444e-02	0.157719	Yes
30	2	Presence	System Usability	Spearman's correlation	6.689008e-04	0.263762	Yes
31	2	Presence	Simulator Sickness	Spearman's correlation	5.403892e-01	-0.048297	No
32	2	Presence	Exclusion	Spearman's correlation	1.719663e-01	0.107504	No

33	2	System Usability	Simulator Sickness	Spearman's correlation	7.077987e-01	0.029579	No
34	2	System Usability	Exclusion	Spearman's correlation	7.983142e-01	0.020168	No
35	2	Simulator Sickness	Exclusion	Spearman's correlation	4.043471e-01	0.065750	No

Next steps: Generate code with `paiwise_correlation_results` View recommended plots New interactive sheet

Run Multivarible regression

```
import statsmodels.api as sm
import pandas as pd
from IPython.display import display

df1_numeric = df1_processed.select_dtypes(include=np.number)
df2_numeric = df2_processed.select_dtypes(include=np.number)

def perform_regression(study: int, df_numeric: pd.DataFrame):
    results = pd.DataFrame(index=df_numeric.columns, columns=df_numeric.columns)
    r2 = {}
    adjusted_r2 = {}

    for dependent_var in df_numeric.columns:
        X = df_numeric.drop(dependent_var, axis=1)
        y = df_numeric[dependent_var]

        X = sm.add_constant(X)
        model = sm.OLS(y, X).fit()

        # Store R-squared for this model
        r2[dependent_var] = model.rsquared
        adjusted_r2[dependent_var] = model.rsquared_adj

        # Store the formatted coefficient and significance for each predictor
        for predictor in X.columns:
            if predictor == 'const':
                continue # Skip the constant

            coef = model.params.get(predictor)
            p_value = model.pvalues.get(predictor)

            # Format the result with a significance indicator
            if p_value is not None and p_value < 0.05:
                formatted_result = f"{coef:.3f}*"
            elif coef is not None:
                formatted_result = f"{coef:.3f}"
            else:
                formatted_result = None


            # Fill the table with the result
            results.loc[predictor, dependent_var] = formatted_result

    # Create the final table within the function
```

```
    r2_row = pd.DataFrame([r2], index=['R-squared'])
    adjusted_r2_row = pd.DataFrame([adjusted_r2], index=['Adjusted R-squared'])
    # Drop the 'Age' column from results before concatenating if it exists
    final_table = pd.concat([results.drop(columns='Age', errors='ignore'), r2_row, adjusted_r2_row], axis=0)

    print(f"Multiple Regression Results for Study {study}")
    final_table = final_table.drop(columns='Age', errors='ignore').T # Use .T to transpose and get the desired format
    return final_table # Return the final table


# Perform regression for each study and store the results
final_table1 = perform_regression(1, df1_numeric)
display(final_table1)
final_table2 = perform_regression(2, df2_numeric)
display(final_table2)
```

```
Multiple Regression Results for Study 1
```

	Age	Immersion	Experienced Realism	Involvement	Presence	Exclusion	R-squared	Adjusted R-squared
Immersion	0.020	NaN	0.546*	0.034	0.371*	-0.022	0.69774	0.687245
Experienced Realism	-0.051	0.704*	NaN	-0.044	0.060	-0.008	0.603067	0.589284
Involvement	-0.016	0.039	-0.040	NaN	0.198*	-0.270*	0.527519	0.511113
Presence	-0.120*	0.549*	0.069	0.252*	NaN	0.041	0.567228	0.552201
Exclusion	0.203*	-0.134	-0.037	-1.412*	0.169	NaN	0.495425	0.477905

```
Multiple Regression Results for Study 2
```

	Age	Experienced Realism	Involvement	Presence	System Usability	Simulator Sickness	Exclusion	R-squared	Adjusted R-squared
Experienced Realism	-0.312	NaN	0.192*	0.408*	0.125*	0.002	-0.273*	0.379648	0.355789
Involvement	-0.317*	0.161*	NaN	0.235*	-0.006	-0.014	0.261*	0.238321	0.209026
Presence	0.091	0.363*	0.249*	NaN	0.023	0.003	0.184	0.305029	0.2783
System Usability	-0.569	0.692*	-0.040	0.141	NaN	0.010	0.338	0.173369	0.141575
Simulator Sickness	1.543	0.133	-1.424	0.272	0.154	NaN	1.728	0.046256	0.009574
Exclusion	0.170	-0.093*	0.106*	0.070	0.021	0.007	NaN	0.088718	0.053669

Next steps: Generate code with `final_table1` | View recommended plots | New interactive sheet | Generate code with `final_table2` | View recommended plots | New interactive sheet

Run Variance inflation factor

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

#Define
def calculate_vif(df):
    df = add_constant(df)
    vif_data = pd.DataFrame()
    vif_data["Feature"] = df.columns
    vif_data["VIF"] = [variance_inflation_factor(df.values, i)
```

```
                for i in range(df.shape[1])]
    vif_data = vif_data[vif_data["Feature"] != "const"]
    return vif_data

# Calculate VIF for df1_processed
vif_df1 = calculate_vif(df1_numeric)
print("VIF for Study 1:")
display(vif_df1.T)

# Calculate VIF for df2_processed
vif_df2 = calculate_vif(df2_numeric)
print("\n\nVIF for Study 2:")
display(vif_df2.T)
```

```
VIF for Study 1:
```

	1	2	3	4	5	6
Feature	Age	Immersion	Experienced Realism	Involvement	Presence	Exclusion
VIF	1.34634	3.308415	2.519314	2.116487	2.310683	1.981866

```
VIF for Study 2:
```

	1	2	3	4	5	6	7
Feature	Age	Experienced Realism	Involvement	Presence	System Usability	Simulator Sickness	Exclusion
VIF	1.166385	1.611989	1.312889	1.43891	1.20973	1.0485	1.097356