```
import pandas as pd
import numpy as np
np.random.seed(130)
cols = pd.read_csv("var_names.csv")
data = pd.read_csv("CSCS_data_anon.csv",
                   na_values=["9999", "", " ", "Presented but no response",
"NA"])
empty = (data.isna().sum()==data.shape[0])
data = data[empty.index[~empty]] # keep non empty columns only

data.shape
```

```
/tmp/ipykernel_11595/4148703477.py:5: DtypeWarning: Columns
(129,408,630,671,689,978,1001,1002,1006,1007,1008,1080,1113,1115,1116,1117,1118,1119,1120,1121
have mixed types. Specify dtype option on import or set low_memory=False.
  data = pd.read_csv("CSCS_data_anon.csv",
```

```
(11431, 1779)
```

```
dataV2 = data[data.REMOVE_case=='No'].copy()
dataV2.shape
```

```
(10018, 1779)
```

```
data2021 = dataV2[dataV2['SURVEY_collection_year'] == 2021].copy()
data2021.shape
```

```
(3186, 1779)
```

```
p_1 = 1480/(1480 + 1237 + 75)
p_0 = (1237 + 75)/(1480 + 1237 + 75)
mask = data2021['CONNECTION_time_with_others_satisfied'].isna()

# Generate random "Yes" or "No" for missing values
random_values = np.random.choice(["Yes", "No"], size=mask.sum(), p=[p_1, p_0])

# Assign random values to missing entries
data2021.loc[mask, 'CONNECTION_time_with_others_satisfied'] = random_values
```

```
data2021['CONNECTION_time_with_others_satisfied_yes'] =
data2021['CONNECTION_time_with_others_satisfied'] == 'Yes'
```

```
data2021['WELLNESS_life_satisfaction'].fillna(data2021[
'WELLNESS_life_satisfaction'].mean(), inplace=True)
```

```
/tmp/ipykernel_11595/291567658.py:1: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.
```

```
  data2021['WELLNESS_life_satisfaction'].fillna(data2021['WELLNESS_life_satisfaction'].mean(),
  inplace=True)
```

```
data2021['CONNECTION_social_media_visits_per_day'].fillna('Missing',
inplace=True)
data2021['CONNECTION_social_media_time_per_day'].fillna('Missing', inplace=True)
data2021['LIFESTYLE_time_use_balance_media'].fillna('Missing', inplace=True)
```

```
/tmp/ipykernel_11595/1275458639.py:1: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.
```

```
  data2021['CONNECTION_social_media_visits_per_day'].fillna('Missing',
  inplace=True)
/tmp/ipykernel_11595/1275458639.py:2: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.
```

```
  data2021['CONNECTION_social_media_time_per_day'].fillna('Missing',
  inplace=True)
/tmp/ipykernel_11595/1275458639.py:3: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  data2021['LIFESTYLE_time_use_balance_media'].fillna('Missing', inplace=True)
```

```
data2021['WELLNESS_above_5'] = data2021['WELLNESS_life_satisfaction'] > 5
```

```
from sklearn import model_selection
train, test = model_selection.train_test_split(data2021, train_size=0.8)
```

```
import statsmodels.formula.api as smf
formula = '''I(CONNECTION_time_with_others_satisfied_yes.astype(int)) ~
                C(CONNECTION_social_media_visits_per_day, Treatment(reference =
'1-3 times per day'))
'''
logreg = smf.logit(formula, data=train)
logreg_fit = logreg.fit()
logreg_fit.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.671967
         Iterations 5
```

Dep. Variable:	I(CONNECTION_time_with_others_satisfied_yes.astype(int))	**No. Observations:**	
Model:	Logit	**Df Residuals:**	
Method:	MLE	**Df Model:**	
Date:	Sat, 02 May 2026	**Pseudo R-squ.:**	
Time:	07:18:48	**Log-Likelihood:**	
converged:	True	**LL-Null:**	
Covariance Type:	nonrobust	**LLR p-value:**	

Intercept
C(CONNECTION_social_media_visits_per_day, Treatment(reference='1-3 times per day'))[T.
C(CONNECTION_social_media_visits_per_day, Treatment(reference='1-3 times per day'))[T.
C(CONNECTION_social_media_visits_per_day, Treatment(reference='1-3 times per day'))[T.
C(CONNECTION_social_media_visits_per_day, Treatment(reference='1-3 times per day'))[T.
C(CONNECTION_social_media_visits_per_day, Treatment(reference='1-3 times per day'))[T.

```
((train.CONNECTION_time_with_others_satisfied ==
'Yes')==(logreg_fit.predict(train) > 0.5)).sum()/train.shape[0]
```

```
np.float64(0.5784929356357927)
```

```
((test.CONNECTION_time_with_others_satisfied == 'Yes')==(logreg_fit.predict(test)
> 0.5)).sum()/test.shape[0]
```

```
np.float64(0.5611285266457681)
```

```
formula = '''I(CONNECTION_time_with_others_satisfied_yes.astype(int)) ~
                C(CONNECTION_social_media_time_per_day, Treatment(reference =
'10-30 minutes per day'))
'''
logreg = smf.logit(formula, data=train)
logreg_fit = logreg.fit()
logreg_fit.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.682033
         Iterations 4
```

Dep. Variable:	I(CONNECTION_time_with_others_satisfied_yes.astype(int))	**No. Observations:**	
Model:	Logit	**Df Residuals:**	
Method:	MLE	**Df Model:**	
Date:	Sat, 02 May 2026	**Pseudo R-squ.:**	
Time:	07:18:49	**Log-Likelihood:**	
converged:	True	**LL-Null:**	
Covariance Type:	nonrobust	**LLR p-value:**	

Intercept
C(CONNECTION_social_media_time_per_day, Treatment(reference='10-30 minutes per day'))
C(CONNECTION_social_media_time_per_day, Treatment(reference='10-30 minutes per day'))
C(CONNECTION_social_media_time_per_day, Treatment(reference='10-30 minutes per day'))
C(CONNECTION_social_media_time_per_day, Treatment(reference='10-30 minutes per day'))
C(CONNECTION_social_media_time_per_day, Treatment(reference='10-30 minutes per day'))
C(CONNECTION_social_media_time_per_day, Treatment(reference='10-30 minutes per day'))

```
((train.CONNECTION_time_with_others_satisfied ==
'Yes')==(logreg_fit.predict(train) > 0.5)).sum()/train.shape[0]
```

```
np.float64(0.5624018838304553)
```

```
((test.CONNECTION_time_with_others_satisfied == 'Yes')==(logreg_fit.predict(test)
> 0.5)).sum()/test.shape[0]
```

```
np.float64(0.5736677115987461)
```

```
import statsmodels.formula.api as smf
formula = '''I(CONNECTION_time_with_others_satisfied_yes.astype(int)) ~
                C(LIFESTYLE_time_use_balance_media, Treatment(reference='Just the
right amount'))
'''
logreg = smf.logit(formula, data=train)
logreg_fit = logreg.fit()
logreg_fit.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.676356
         Iterations 4
```

Dep. Variable:	I(CONNECTION_time_with_others_satisfied_yes.astype(int))	**No. Observations:**	
Model:	Logit	**Df Residuals:**	
Method:	MLE	**Df Model:**	
Date:	Sat, 02 May 2026	**Pseudo R-squ.:**	
Time:	07:18:49	**Log-Likelihood:**	
converged:	True	**LL-Null:**	
Covariance Type:	nonrobust	**LLR p-value:**	

Intercept					
C(LIFESTYLE_time_use_balance_media, Treatment(reference='Just the right amount'))[T.Mi					
C(LIFESTYLE_time_use_balance_media, Treatment(reference='Just the right amount'))[T.To					
C(LIFESTYLE_time_use_balance_media, Treatment(reference='Just the right amount'))[T.To					

```
((train.CONNECTION_time_with_others_satisfied ==
'Yes')==(logreg_fit.predict(train) > 0.5)).sum()/train.shape[0]
```

```
np.float64(0.5828100470957613)
```

```
((test.CONNECTION_time_with_others_satisfied == 'Yes')==(logreg_fit.predict(test)
> 0.5)).sum()/test.shape[0]
```

```
np.float64(0.5909090909090909)
```

```
import statsmodels.formula.api as smf
formula = '''I(WELLNESS_above_5.astype(int)) ~
                    C(CONNECTION_social_media_visits_per_day,
Treatment(reference='1-3 times per day'))+
                    C(CONNECTION_social_media_time_per_day) +
                    C(LIFESTYLE_time_use_balance_media)
'''
logreg = smf.logit(formula, data=train)
logreg_fit2 = logreg.fit()
logreg_fit2.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.630624
         Iterations 5
```

Dep. Variable:	I(WELLNESS_above_5.astype(int))	**No. Observations:**	2548
Model:	Logit	**Df Residuals:**	2533
Method:	MLE	**Df Model:**	14
Date:	Sat, 02 May 2026	**Pseudo R-squ.:**	0.02495
Time:	07:18:49	**Log-Likelihood:**	-1606.8
converged:	True	**LL-Null:**	-1648.0
Covariance Type:	nonrobust	**LLR p-value:**	1.082e-11

Intercept
C(CONNECTION_social_media_visits_per_day, Treatment(reference='1-3 times per day'))[T.
C(CONNECTION_social_media_visits_per_day, Treatment(reference='1-3 times per day'))[T.
C(CONNECTION_social_media_visits_per_day, Treatment(reference='1-3 times per day'))[T.
C(CONNECTION_social_media_visits_per_day, Treatment(reference='1-3 times per day'))[T.
C(CONNECTION_social_media_visits_per_day, Treatment(reference='1-3 times per day'))[T.
C(CONNECTION_social_media_time_per_day)[T.10-30 minutes per day]
C(CONNECTION_social_media_time_per_day)[T.2-3 hours per day]
C(CONNECTION_social_media_time_per_day)[T.31-60 minutes per day]
C(CONNECTION_social_media_time_per_day)[T.Less than 10 minutes per day]
C(CONNECTION_social_media_time_per_day)[T.Missing]
C(CONNECTION_social_media_time_per_day)[T.More than 3 hours per day]
C(LIFESTYLE_time_use_balance_media)[T.Missing]
C(LIFESTYLE_time_use_balance_media)[T.Too little]
C(LIFESTYLE_time_use_balance_media)[T.Too much]

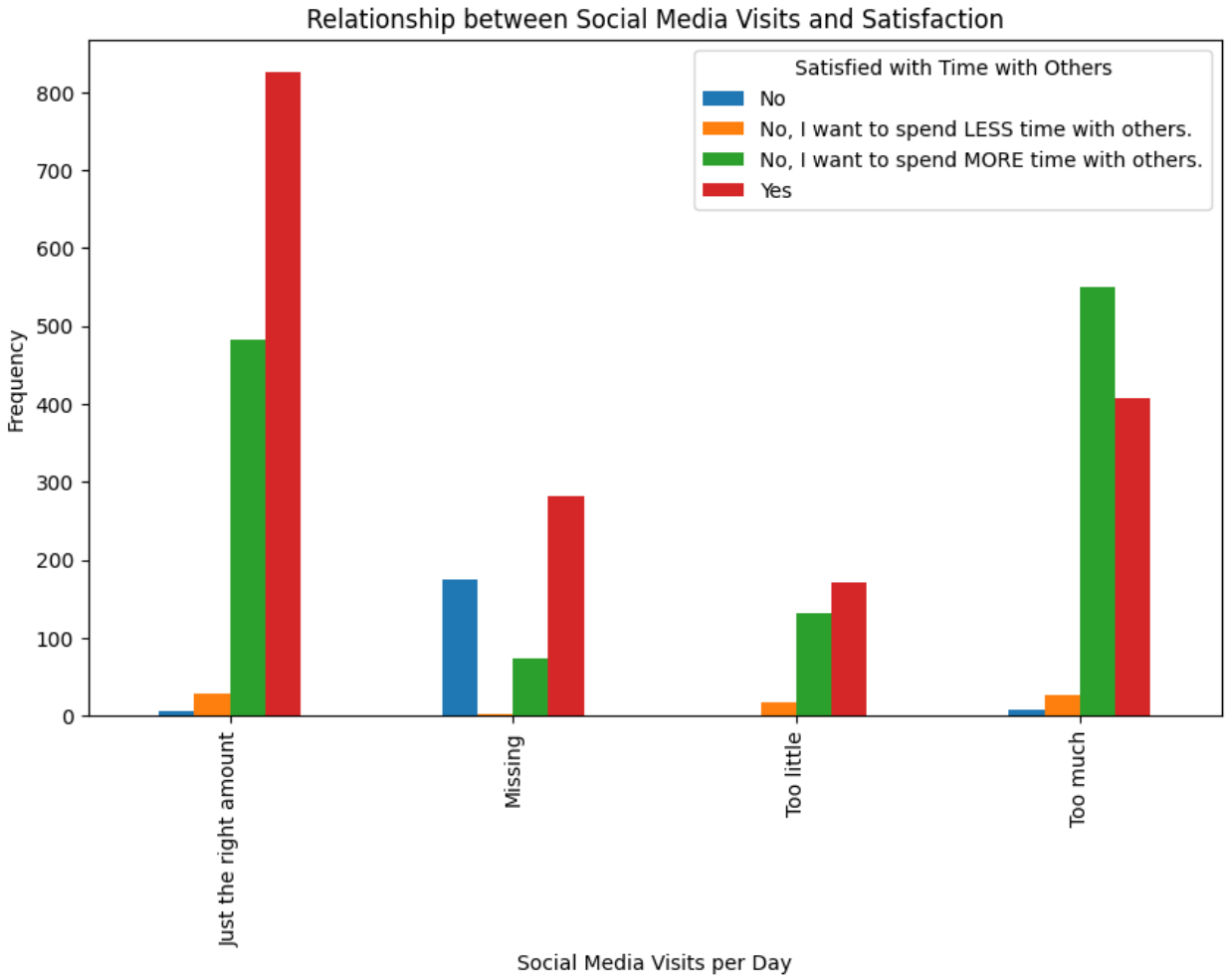
```
((train.CONNECTION_time_with_others_satisfied ==
'Yes')==(logreg_fit2.predict(train) > 0.5)).sum()/train.shape[0]
```

```
np.float64(0.5498430141287284)
```

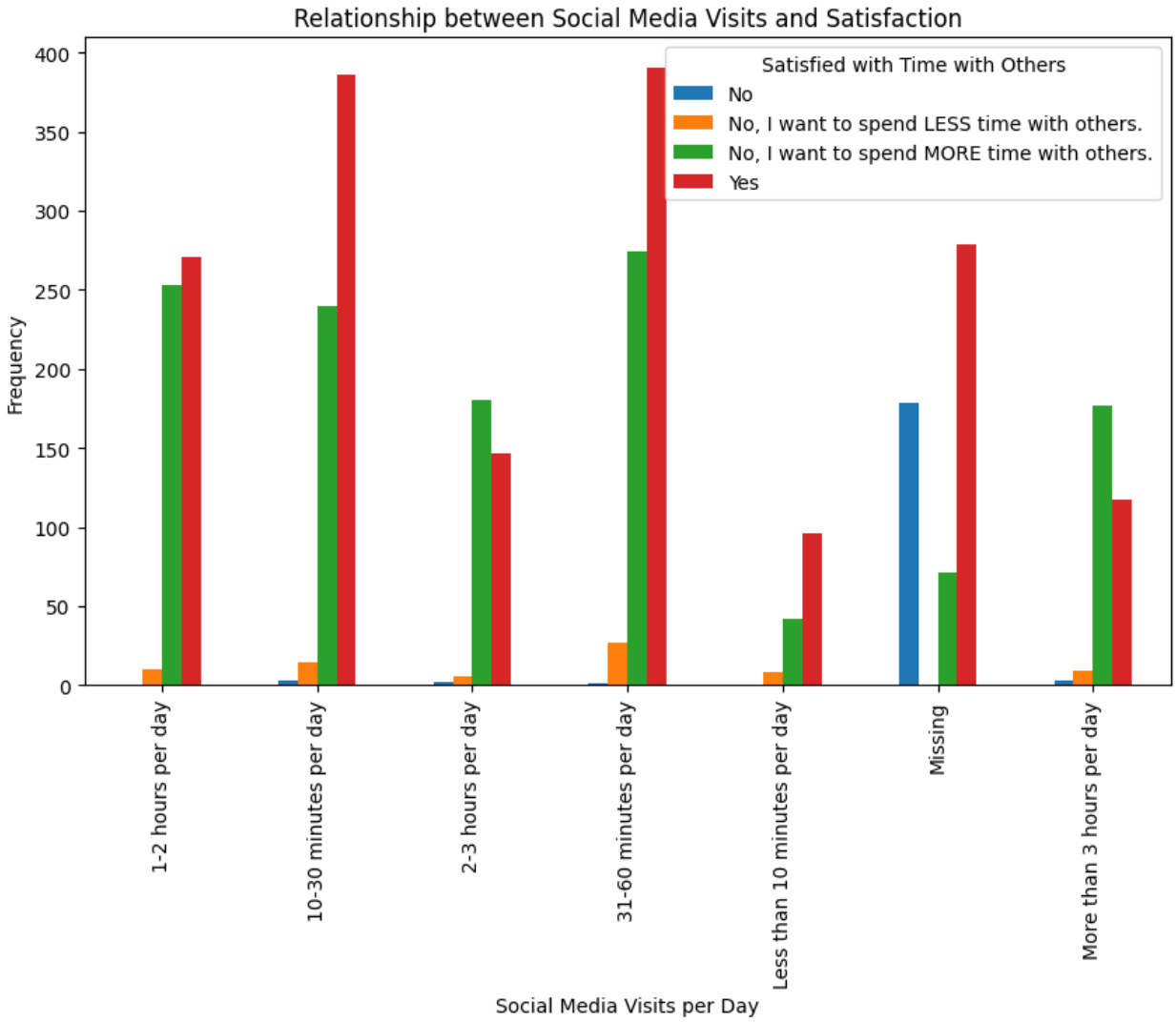
```
((test.CONNECTION_time_with_others_satisfied ==
'Yes')==(logreg_fit2.predict(test) > 0.5)).sum()/test.shape[0]
```

```
np.float64(0.5532915360501567)
```

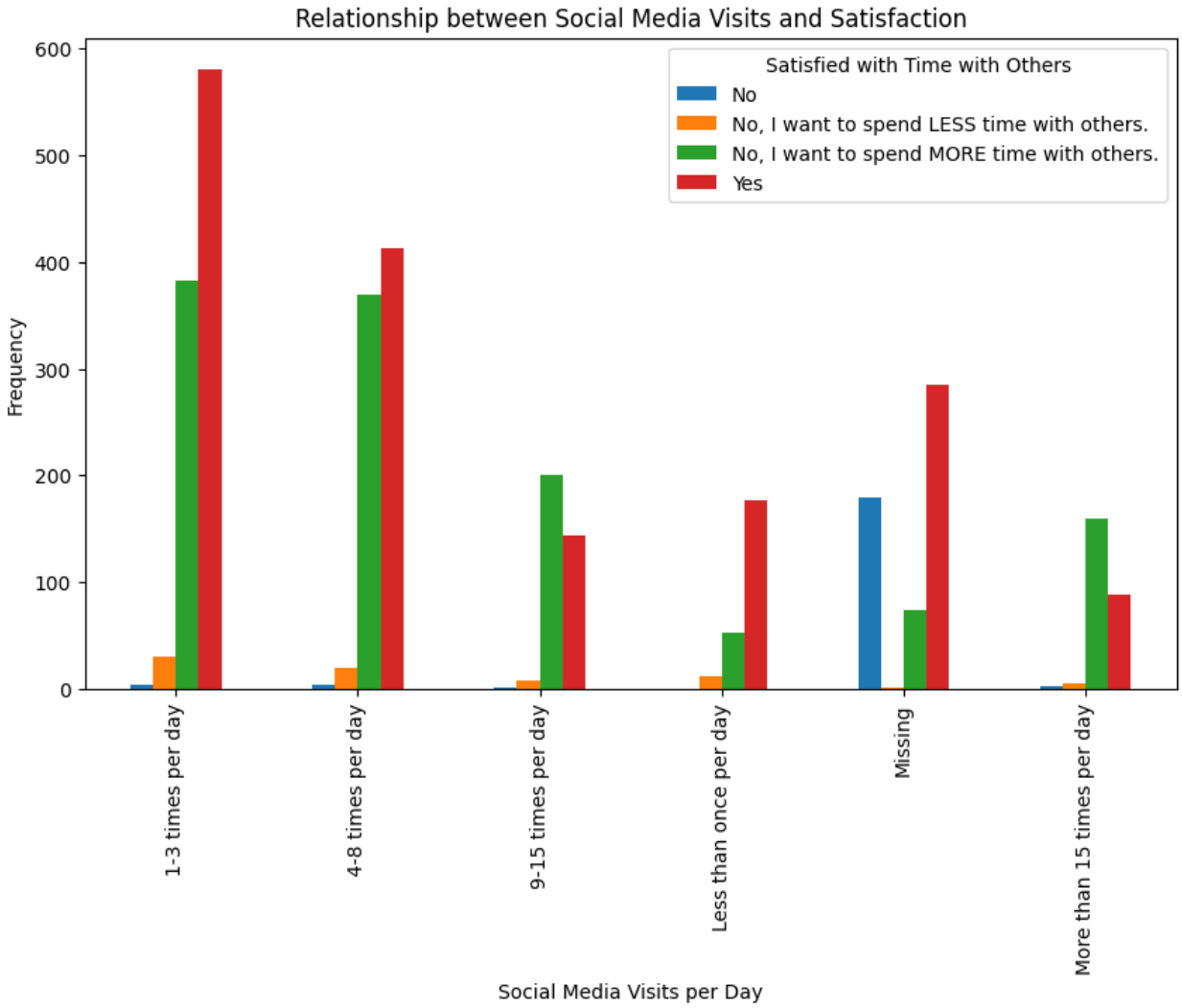
```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a crosstab of the two categorical variables
crosstab = pd.crosstab(data2021['LIFESTYLE_time_use_balance_media'],
                       data2021['CONNECTION_time_with_others_satisfied'])

# Plot the grouped bar chart
crosstab.plot(kind='bar', figsize=(10, 6))
plt.title('Relationship between Social Media Visits and Satisfaction')
plt.xlabel('Social Media Visits per Day')
plt.ylabel('Frequency')
plt.legend(title='Satisfied with Time with Others')
plt.show()
```




```
crosstab = pd.crosstab(data2021['CONNECTION_social_media_time_per_day'],
                       data2021['CONNECTION_time_with_others_satisfied'])

# Plot the grouped bar chart
crosstab.plot(kind='bar', figsize=(10, 6))
plt.title('Relationship between Social Media Visits and Satisfaction')
plt.xlabel('Social Media Visits per Day')
plt.ylabel('Frequency')
plt.legend(title='Satisfied with Time with Others')
plt.show()
```




```
crosstab = pd.crosstab(data2021['CONNECTION_social_media_visits_per_day'],
                       data2021['CONNECTION_time_with_others_satisfied'])

# Plot the grouped bar chart
crosstab.plot(kind='bar', figsize=(10, 6))
plt.title('Relationship between Social Media Visits and Satisfaction')
plt.xlabel('Social Media Visits per Day')
plt.ylabel('Frequency')
plt.legend(title='Satisfied with Time with Others')
plt.show()
```


Relationship between Social Media Visits and Satisfaction
Satisfied with Time with Others
No
No, I want to spend LESS time with others.
No, I want to spend MORE time with others.
Yes
Frequency
600
500
400
300
200
100
0
1-3 times per day
4-8 times per day
9-15 times per day
Less than once per day
Missing
More than 15 times per day
Social Media Visits per Day

```
colab2pdf()
```