In [ ]:

```python
import numpy as np
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm

# Data
C = [16.30, 11.90, 19.30, 23.60, 15.10, 16.80, 8.50, 18.70, 15.20, 9.80, 23.
F1 = [22.60, 13.00, 30.10, 14.10, 21.00, 14.20, 19.80, 31.80, 27.00, 17.20,
F2 = [14.60, 16.80, 16.90, 18.10, 20.20, 15.30, 17.60, 28.10, 19.10, 19.90,
F3 = [26.10, 27.80, 19.70, 16.90, 23.30, 26.00, 23.90, 20.50, 21.00, 27.80,

data = {
    'WeightLoss': C + F1 + F2 + F3,
    'Group': ['C']*len(C) + ['F1']*len(F1) + ['F2']*len(F2) + ['F3']*len(F3)
}

df = pd.DataFrame(data)
model = ols('WeightLoss ~ Group', data=df).fit()

anova_table = anova_lm(model)
print(anova_table)
```

```
            df       sum_sq     mean_sq          F    PR(>F)
Group      3.0   746.352375  248.784125  10.363037  0.000009
Residual  76.0  1824.522500   24.006875        NaN       NaN
```

In [ ]:

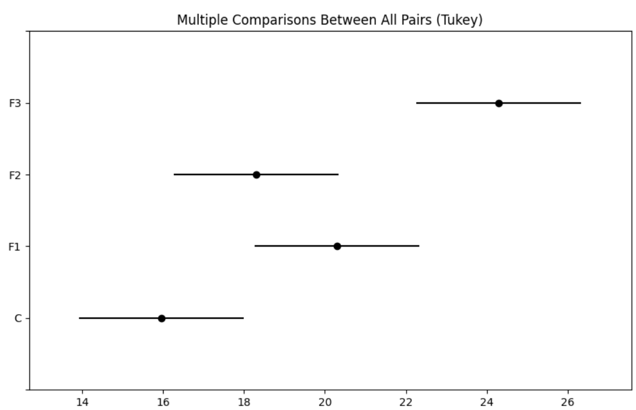
```python
import pandas as pd
import numpy as np
from statsmodels.stats.multicomp import pairwise_tukeyhsd
import matplotlib.pyplot as plt

# Data
data = {
    'C': [16.30, 11.90, 19.30, 23.60, 15.10, 16.80, 8.50, 18.70, 15.20, 9.80
          23.60, 9.10, 18.30, 13.20, 12.00, 15.30, 23.50, 9.50, 13.60, 26.00
    'F1': [22.60, 13.00, 30.10, 14.10, 21.00, 14.20, 19.80, 31.80, 27.00, 17
           16.60, 22.50, 14.10, 14.70, 25.70, 19.20, 23.20, 28.10, 19.00, 12
    'F2': [14.60, 16.80, 16.90, 18.10, 20.20, 15.30, 17.60, 28.10, 19.10, 19
           18.70, 21.30, 16.00, 25.30, 13.30, 24.40, 11.40, 11.10, 18.40, 19
    'F3': [26.10, 27.80, 19.70, 16.90, 23.30, 26.00, 23.90, 20.50, 21.00, 27
           23.10, 22.80, 24.90, 22.00, 23.40, 21.60, 24.00, 30.60, 29.10, 31
}

df = pd.DataFrame(data)
stacked_data = df.stack().reset_index()
stacked_data.columns = ['Index', 'Group', 'Weight Loss']

tukey = pairwise_tukeyhsd(stacked_data['Weight Loss'], stacked_data['Group']
print(tukey.summary())
tukey.plot_simultaneous()
plt.show()
```

```
Multiple Comparison of Means - Tukey HSD, FWER=0.05
===================================================
group1 group2 meandiff p-adj  lower  upper  reject
---------------------------------------------------
     C     F1     4.33 0.0326   0.26    8.4   True
     C     F2     2.34 0.4365  -1.73   6.41  False
     C     F3    8.325    0.0  4.255 12.395   True
    F1     F2    -1.99 0.5756  -6.06   2.08  False
    F1     F3    3.995 0.0563 -0.075  8.065  False
    F2     F3    5.985 0.0013  1.915 10.055   True
---------------------------------------------------
```



In [ ]:

```python
import numpy as np
from scipy.stats import dunnett

c = np.array([16.30, 11.90, 19.30, 23.60, 15.10, 16.80, 8.50, 18.70, 15.20,
f1 = np.array([22.60, 13.00, 30.10, 14.10, 21.00, 14.20, 19.80, 31.80, 27.00
f2 = np.array([14.60, 16.80, 16.90, 18.10, 20.20, 15.30, 17.60, 28.10, 19.10
f3 = np.array([26.10, 27.80, 19.70, 16.90, 23.30, 26.00, 23.90, 20.50, 21.00

res = dunnett(f1, f2, f3, control=c)
res.pvalue
```

Out[ ]:

```
array([1.80154936e-02, 3.07340568e-01, 2.45973916e-06])
```

In [ ]:
```python
import numpy as np
import scipy.stats as stats

group_means = np.array([15.965, 20.295, 18.305, 24.290])
j = 20
mse = 24.006875
alpha = 0.05
k = 5

alpha_adj = alpha / k

t_stats = []
t_stats.append((group_means[0] - group_means[1])/(np.sqrt(mse / 20 + mse / 2
t_stats.append((group_means[0] - group_means[2])/(np.sqrt(mse / 20 + mse / 2
t_stats.append((group_means[0] - group_means[3])/(np.sqrt(mse / 20 + mse / 2
t_stats.append((group_means[0] - (group_means[1] + group_means[2] + group_me
t_stats.append((group_means[1] - group_means[2])/(np.sqrt(mse / 20 + mse / 2

df = 76
t_critical = stats.t.ppf(1 - alpha_adj / 2, df)

results = []

for t_stat in t_stats:
    results.append(abs(t_stat) > t_critical)

for result in results:
    print(result)
```

```
True
False
True
True
False
```

In [ ]:
```python
means = np.array([15.965, 20.295, 18.305, 24.290])
J = 20
I = len(means)
df1 = I - 1
df2 = I * (J - 1)
alpha = 0.05
mse = 24.006875
F_obs = 10.363

grand_mean = np.mean(means)

SSTr = J * np.sum((means - grand_mean) ** 2)
lambda_ = SSTr / mse

f_crit = f.ppf(1 - alpha, df1, df2)
power = 1 - ncf.cdf(f_crit, df1, df2, lambda_)

print(f"Noncentrality parameter (lambda): {lambda_:.4f}")
print(f"Critical F value: {f_crit:.4f}")
print(f"Estimated power: {power:.4f}")
```

```
Noncentrality parameter (lambda): 31.0891
Critical F value: 2.7249
Estimated power: 0.9981
```

In [ ]:
```python
import pandas as pd
import statsmodels.api as sm
import statsmodels.formula.api as smf

data = [100.7, 102.4, 104.3, 109.2, 101.7, 99.4, 103.5, 104.0, 107.7, 105.9,
block = ['B1', 'B1', 'B2', 'B2', 'B3', 'B3'] * 4
dose = ['Control'] * 6 + ['Dose 1'] * 6 + ['Dose 2'] * 6 + ['Dose 3'] * 6

df = pd.DataFrame({
    'Response': data,
    'Block': block,
    'Dose': dose
})

df['Block'] = df['Block'].astype('category')
df['Dose'] = df['Dose'].astype('category')

model = smf.ols('Response ~ C(Dose) + C(Block)', data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=1)

print(anova_table)
```

In [ ]:
```python
SSTr = 0
for drug in ['Control', 'Dose 1', 'Dose 2', 'Dose 3']:
    SSTr += 6 * ((df[df['Drug'] == drug].sum()[0] / 6) - 102.429166667)**2
SSTr
```

In [ ]:
```
SSB = 0
for block in ['B1', 'B2', 'B3']:
    SSB += 8 * ((df[df['Block'] == block].sum()[0] / 8) - 102.429166667)**2
SSB
```

Out[ ]: 161.68083333333323

In [ ]:
```
((df[df['Block'] == 'B1'].sum()[0]))
```

Out[ ]: 807.8

In [ ]:
```
for i in range(4):
    print(i)
df.loc[0].Response
```

```
0
1
2
3
```

Out[ ]: 100.7

In [ ]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder

drug_levels = ['Control'] * 6 + ['Dose 1'] * 6 + ['Dose 2'] * 6 + ['Dose 3']
block_levels = ['B1', 'B1', 'B2', 'B2', 'B3', 'B3'] * 4

df = pd.DataFrame({'Drug': drug_levels, 'Block': block_levels})

X = pd.get_dummies(df, columns=['Drug', 'Block'], drop_first=False)
X.insert(0, 'Intercept', 1)

X = X[['Intercept', 'Drug_Control', 'Drug_Dose 1', 'Drug_Dose 2', 'Drug_Dose
       'Block_B1', 'Block_B2', 'Block_B3']]
X
```

Out[ ]:

	Intercept	Drug_Control	Drug_Dose 1	Drug_Dose 2	Drug_Dose 3	Block_B1	Block_B2	Bloc
0	1	1	0	0	0	1	0	
1	1	1	0	0	0	1	0	
2	1	1	0	0	0	0	1	
3	1	1	0	0	0	0	1	
4	1	1	0	0	0	0	0	
5	1	1	0	0	0	0	0	
6	1	0	1	0	0	1	0	
7	1	0	1	0	0	1	0	
8	1	0	1	0	0	0	1	
9	1	0	1	0	0	0	1	
10	1	0	1	0	0	0	0	
11	1	0	1	0	0	0	0	
12	1	0	0	1	0	1	0	
13	1	0	0	1	0	1	0	
14	1	0	0	1	0	0	1	
15	1	0	0	1	0	0	1	
16	1	0	0	1	0	0	0	
17	1	0	0	1	0	0	0	
18	1	0	0	0	1	1	0	
19	1	0	0	0	1	1	0	
20	1	0	0	0	1	0	1	
21	1	0	0	0	1	0	1	
22	1	0	0	0	1	0	0	
23	1	0	0	0	1	0	0	

```
In [ ]: import pandas as pd
import numpy as np

response = [
    100.7, 102.4, 104.3, 109.2, 101.7, 99.4,   # Control
    103.5, 104.0, 107.7, 105.9, 104.5, 102.3,   # Dose 1
    97.8, 94.6, 105.2, 102.9, 98.0, 99.9,       # Dose 2
    102.6, 102.2, 106.8, 106.6, 100.1, 96.0     # Dose 3
]

drug = ['Control'] * 6 + ['Dose 1'] * 6 + ['Dose 2'] * 6 + ['Dose 3'] * 6
block = (['B1', 'B1', 'B2', 'B2', 'B3', 'B3']) * 4

df = pd.DataFrame({
    'Response': response,
    'Drug': drug,
    'Block': block
})

X = pd.get_dummies(df[['Drug', 'Block']], drop_first=False)
X.insert(0, 'Intercept', 1)

X_np = X.values
y_np = df['Response'].values

# least squares solution using pseudoinverse
beta_hat = np.linalg.pinv(X_np) @ y_np

coeff_names = X.columns
coefficients = pd.Series(beta_hat, index=coeff_names)
print(coefficients)
```

```
Intercept       64.692105
Drug_Control    16.693860
Drug_Dose 1     18.393860
Drug_Dose 2     13.477193
Drug_Dose 3     16.127193
Block_B1        20.109868
Block_B2        25.209868
Block_B3        19.372368
dtype: float64
```

```
In [ ]: import numpy as np
import pandas as pd
from numpy.linalg import inv

data = [100.7, 102.4, 104.3, 109.2, 101.7, 99.4,  # Control
        103.5, 104.0, 107.7, 105.9, 104.5, 102.3,  # Dose 1
        97.8, 94.6, 105.2, 102.9, 98.0, 99.9,     # Dose 2
        102.6, 102.2, 106.8, 106.6, 100.1, 96.0]  # Dose 3
block = ['B1', 'B1', 'B2', 'B2', 'B3', 'B3'] * 4
drug = ['Control'] * 6 + ['Dose 1'] * 6 + ['Dose 2'] * 6 + ['Dose 3'] * 6


df = pd.DataFrame({
    'Response': data,
    'Block': block,
    'Drug': drug
})

# Columns: Intercept, Drug_Control, Drug_Dose1, Drug_Dose2, Block_B1, Block_
X = np.array([[1, 1, 0, 0, 1, 0],  # Control, Block B1
              [1, 1, 0, 0, 1, 0],  # Control, Block B1
              [1, 1, 0, 0, 0, 1],  # Control, Block B2
              [1, 1, 0, 0, 0, 1],  # Control, Block B2
              [1, 1, 0, 0, -1, -1],  # Control, Block B3
              [1, 1, 0, 0, -1, -1],  # Control, Block B3
              [1, 0, 1, 0, 1, 0],  # Dose 1, Block B1
              [1, 0, 1, 0, 1, 0],  # Dose 1, Block B1
              [1, 0, 1, 0, 0, 1],  # Dose 1, Block B1
              [1, 0, 1, 0, 0, 1],  # Dose 1, Block B1
              [1, 0, 1, 0, -1, -1],  # Dose 1, Block B1
              [1, 0, 1, 0, -1, -1],  # Dose 1, Block B1
              [1, 0, 0, 1, 1, 0],  # Dose 2, Block B1
              [1, 0, 0, 1, 1, 0],  # Dose 2, Block B1
              [1, 0, 0, 1, 0, 1],  # Dose 2, Block B1
              [1, 0, 0, 1, 0, 1],  # Dose 2, Block B1
              [1, 0, 0, 1, -1, -1],  # Dose 2, Block B1
              [1, 0, 0, 1, -1, -1],  # Dose 2, Block B1
              [1, -1, -1, -1, 1, 0],  # Dose 3, Block B1
              [1, -1, -1, -1, 1, 0],  # Dose 3, Block B1
              [1, -1, -1, -1, 0, 1],  # Dose 3, Block B1
              [1, -1, -1, -1, 0, 1],  # Dose 3, Block B1
              [1, -1, -1, -1, -1, -1],  # Dose 3, Block B1
              [1, -1, -1, -1, -1, -1],  # Dose 3, Block B1
])

X_df = pd.DataFrame(X, columns=["Intercept", "Drug_Control", "Drug_Dose1", "

Y = np.array(df['Response'])
beta_hat = np.linalg.inv(X.T @ X) @ X.T @ Y

beta_hat_df = pd.Series(beta_hat, index=["mu", "alpha1", "alpha2", "alpha3",
print("Full-Rank Least Squares Estimate (beta_hat):")
print(beta_hat_df)
print("alpha 4 calculated = - (alpha1 + alpha2 + alpha3) = " + str(-1 * ( be
print("beta 3 calculated = - (b1 + b2) = " + str(-1 * ( beta_hat_df['beta1']
```

```
Full-Rank Least Squares Estimate (beta_hat):
mu        102.429167
alpha1      0.520833
alpha2      2.220833
alpha3     -2.695833
beta1      -1.454167
beta2       3.645833
dtype: float64
alpha 4 calculated = - (alpha1 + alpha2 + alpha3) = -0.045833333333317405
beta 3 calculated = - (b1 + b2) = -2.191666666666665
```

In [ ]:
```
print("alphas in reference to Mean:")
for drug in ['Control', 'Dose 1', 'Dose 2', 'Dose 3']:
    print((df[df['Drug'] == drug].sum()[0] / 6) - 102.429166667)
```

```
alphas in reference to Mean:
0.5208333329999988
2.2208333329999874
-2.6958333336666698
-0.04583333366666409
```

In [ ]:
```
print("betas in reference to mean:")
for block in ['B1', 'B2', 'B3']:
    print((df[df['Block'] == block].sum()[0] / 8) - 102.429166667)
```

```
betas in reference to mean:
-1.4541666670000097
3.645833332999999
-2.1916666669999927
```

In [ ]:
```
coefficients # Rank Deficient
beta_hat_df # Full rank

# C1
print("C1")
c1_rd = np.array([0, 1, -1, 0, 0, 0, 0, 0])  # Contrast between Dose 1 and D
c1_fr = np.array([0, 1, -1, 0, 0, 0])
print("RD", np.dot(c1_rd, coefficients))  # First contrast
print("FR", np.dot(c1_fr, beta_hat_df))  # First contrast

# C2
print("C2")
c2_rd = np.array([0, 1, -1/3, -1/3, -1/3, 0, 0, 0])  # Contrast between Dose
c2_fr = np.array([0, 4/3, 0, 0, 0, 0])
print("RD", np.dot(c2_rd, coefficients))  # First contrast
print("FR", np.dot(c2_fr, beta_hat_df))  # First contrast

# C1
c3_rd = np.array([0, 1, 1, 0, 0, -1, -1, 0])  # Contrast between Dose 1 and
c3_fr = np.array([0, 1, 1, 0, -1, -1])
print("RD", np.dot(c3_rd, coefficients))  # First contrast
print("FR", np.dot(c3_fr, beta_hat_df))  # First contrast
```

```
C1
RD -1.6999999999999957
FR -1.7000000000000066
C2
RD 0.6944444444444535
FR 0.6944444444444283
RD -10.232017543859627
FR 0.5499999999999843
```

In [ ]:
```
from scipy.stats import kruskal

group1 = [8, 10, 12, 10, 13, 12, 12, 15, 13, 9]
group2 = [8, 14, 16, 14, 10, 11, 10, 9, 9, 12]
group3 = [11, 12, 11, 23, 19, 11, 17, 17, 16, 16]
group4 = [13, 17, 20, 15, 11, 17, 16, 5, 11, 20]

statistic, p_value = kruskal(group1, group2, group3, group4)

print("Kruskal-Wallis Statistic:", statistic)
print("P-value:", p_value)
```

```
Kruskal-Wallis Statistic: 9.27796080658906
P-value: 0.025814669132235304
```

In [ ]:
```python
import numpy as np
import pandas as pd
from scipy.stats import rankdata, chi2

Group1 = [8, 10, 12, 10, 13, 12, 12, 15, 13, 9]
Group2 = [8, 14, 16, 14, 10, 11, 10, 9, 9, 12]
Group3 = [11, 12, 11, 23, 19, 11, 17, 17, 16, 16]
Group4 = [13, 17, 20, 15, 11, 17, 16, 5, 11, 20]

data = Group1 + Group2 + Group3 + Group4
groups = (['Group1'] * len(Group1) +
          ['Group2'] * len(Group2) +
          ['Group3'] * len(Group3) +
          ['Group4'] * len(Group4))

N = len(data)
k = 4

ranks = rankdata(data)

df = pd.DataFrame({'Data': data, 'Group': groups, 'Rank': ranks})
print(df)

group_stats = df.groupby('Group').agg(n=('Rank', 'size'),
                                      R=('Rank', 'sum'))
print("\nGroup Stats:")
print(group_stats)

H = (12 / (N * (N + 1))) * np.sum(group_stats['R']**2 / group_stats['n']) -
print("\nKruskal-Wallis H statistic (uncorrected):", H)

tie_counts = np.array([np.sum(np.array(data)==val) for val in np.unique(data
tie_term = np.sum(tie_counts**3 - tie_counts)
T = 1 - tie_term / (N**3 - N)
print("Tie correction factor T:", T)

H_corrected = H / T
print("Kruskal-Wallis H statistic (corrected):", H_corrected)

df_chi2 = k - 1  # 3

p_value = 1 - chi2.cdf(H_corrected, df_chi2)
print("\np-value:", p_value)
```

```
    Data   Group  Rank
0      8  Group1   2.5
1     10  Group1   8.5
2     12  Group1  19.0
3     10  Group1   8.5
4     13  Group1  23.0
5     12  Group1  19.0
6     12  Group1  19.0
7     15  Group1  27.5
8     13  Group1  23.0
9      9  Group1   5.0
10     8  Group2   2.5
11    14  Group2  25.5
12    16  Group2  30.5
13    14  Group2  25.5
14    10  Group2   8.5
15    11  Group2  13.5
16    10  Group2   8.5
17     9  Group2   5.0
18     9  Group2   5.0
19    12  Group2  19.0
20    11  Group3  13.5
21    12  Group3  19.0
22    11  Group3  13.5
23    23  Group3  40.0
24    19  Group3  37.0
25    11  Group3  13.5
26    17  Group3  34.5
27    17  Group3  34.5
28    16  Group3  30.5
29    16  Group3  30.5
30    13  Group4  23.0
31    17  Group4  34.5
32    20  Group4  38.5
33    15  Group4  27.5
34    11  Group4  13.5
35    17  Group4  34.5
36    16  Group4  30.5
37     5  Group4   1.0
38    11  Group4  13.5
39    20  Group4  38.5

Group Stats:
         n      R
Group
Group1  10  155.0
Group2  10  143.5
Group3  10  266.5
Group4  10  255.0

Kruskal-Wallis H statistic (uncorrected): 9.193536585365877
Tie correction factor T: 0.9909005628517824
Kruskal-Wallis H statistic (corrected): 9.27796080658906

p-value: 0.02581466913223529
```

In [ ]:
```python
# Total ranks
avg_ranks = []
for group in ['Group1','Group2','Group3','Group4']:
    avg_ranks.append(df[df['Group'] == group].sum()['Rank'] / 10)

total_ranks_mean = sum(avg_ranks) / 4
for group in ['Group1','Group2','Group3','Group4']:
    avg_ranks.append(df[df['Group'] == group].sum()['Rank'])

H = 0
num = 0
denom = 0
for group in ['Group1','Group2','Group3','Group4']:
    num += (10 * (df[df['Group'] == group].sum()['Rank']/10 - total_ranks_me
for rank in df['Rank']:
    denom += (rank - total_ranks_mean)**2
H = (40 - 1) * num / denom
H
```

Out[ ]:
```
9.277960806589036
```

In [ ]:
```python
for group in ['Group1','Group2','Group3','Group4']:
    print(10 * (df[df['Group'] == group].sum()['Rank']/10 - total_ranks_mear
```

```
250.0
378.225
378.22499999999985
250.0
```

In [ ]:
```python
import numpy as np

Group1 = np.array([8, 10, 12, 10, 13, 12, 12, 15, 13, 9])
Group2 = np.array([8, 14, 16, 14, 10, 11, 10, 9, 9, 12])
Group3 = np.array([11, 12, 11, 23, 19, 11, 17, 17, 16, 16])
Group4 = np.array([13, 17, 20, 15, 11, 17, 16, 5, 11, 20])

def calc_stat(groups):
    """Calculates difference in means (test statistic) between the group mea
    means = [np.mean(group) for group in groups]
    return np.max(means) - np.min(means)

observed_stat = calc_stat([Group1, Group2, Group3, Group4])

def permutation_test(groups, n_permutations=10000):
    """Perform permutation test"""
    combined_data = np.concatenate(groups)
    stat_list = []
    for _ in range(n_permutations):
        np.random.shuffle(combined_data)
        new_groups = np.split(combined_data, 4)
        stat_list.append(calc_stat(new_groups))
    return stat_list

perm_stats = permutation_test([Group1, Group2, Group3, Group4])

p_value = np.mean(np.array(perm_stats) >= observed_stat)

print('observed_stat = ', observed_stat)
print('perm_stats example = ', perm_stats[1:4])
print('p_value = ', p_value)
```

```
observed_stat =  4.0
perm_stats example =  [3.4000000000000004, 3.5999999999999996, 1.0]
p_value =  0.0912
```

In [ ]:
```python
print(np.mean(Group1))
print(np.mean(Group2))
print(np.mean(Group3))
print(np.mean(Group4))
```

```
11.4
11.3
15.3
14.5
```