Importing Libraries

```
import pandas as pd
from statsmodels.multivariate.manova import MANOVA
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.stats.multicomp import pairwise_tukeyhsd
```

Importig Dataset

```
# Step 1: Load data
file_path = "Manova data.xlsx"
df = pd.read_excel(file_path, sheet_name="Upload-sums")
```

```
df.head(2)
```

	TCP upload sum - BBR-v3	TCP upload sum - BBR-n+	TCP upload sum - Cubic	TCP upload sum - Reno	TCP upload sum - DCTCP	TCP upload sum - Vegas	TCP upload sum - Veno	TCP upload sum - Nevada	TCP upload sum - Yeah
0	163.580000	187.526857	129.728440	192.900000	170.920204	78.896850	292.767311	137.580000	185.844459
1	117.987962	161.410500	116.348551	246.044211	120.326247	55.930096	201.862007	108.819575	167.371564

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

Transformation

```
# Step 2: Clean column names
df.columns = (
    df.columns
    .str.strip()
    .str.replace(' ', '_', regex=False)
    .str.replace('-', '_', regex=False)
    .str.replace('+', 'plus', regex=False)
)


# Step 3: Add Observation column (assuming each row is a subject)
df['Observation'] = df.index

# Step 4: Pivot to long format (for visualization/inspection)
df_long = df.melt(
    id_vars=['Observation'],
    var_name='Algorithm',
    value_name='UploadThroughput'
)

# Step 5: Pivot back to wide format for MANOVA
df_wide = df_long.pivot(index='Observation', columns='Algorithm', values='UploadThroughput').reset_index()
```

```
df_long
```

	Observation	Algorithm	UploadThroughput
0	0	TCP_upload_sum___BBR_v3	163.580000
1	1	TCP_upload_sum___BBR_v3	117.987962
2	2	TCP_upload_sum___BBR_v3	90.414199
3	3	TCP_upload_sum___BBR_v3	86.638446
4	4	TCP_upload_sum___BBR_v3	94.789337
...	...	...	...
2695	295	TCP_upload_sum___Yeah	93.310664
2696	296	TCP_upload_sum___Yeah	94.189948
2697	297	TCP_upload_sum___Yeah	94.871799
2698	298	TCP_upload_sum___Yeah	98.566608
2699	299	TCP_upload_sum___Yeah	122.600000

2700 rows × 3 columns

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

Applying MANOVA

```
# Step 6: Build MANOVA formula
dependent_vars = df_wide.columns.drop('Observation').tolist()
manova_formula = ' + '.join(dependent_vars) + ' ~ 1'

# Step 7: Run MANOVA
maov = MANOVA.from_formula(manova_formula, data=df_wide)
results = maov.mv_test()
print(results)
```

```
                  Multivariate linear model
================================================================
```

```
-----------------------------------------------------------------
       Intercept         Value   Num DF  Den DF   F Value   Pr > F
-----------------------------------------------------------------
          Wilks' lambda   0.0019 9.0000 291.0000 16659.0929 0.0000
         Pillai's trace   0.9981 9.0000 291.0000 16659.0929 0.0000
 Hotelling-Lawley trace 515.2297 9.0000 291.0000 16659.0929 0.0000
    Roy's greatest root 515.2297 9.0000 291.0000 16659.0929 0.0000
=================================================================
```

Results of MANOVA

🎓 What Is MANOVA Testing?

You gave it data for **many algorithms** (like BBR-v3, Reno, Cubic, etc.), and it's testing:

> "Are these algorithms producing **different upload speeds** overall?"

MANOVA is like a **supercharged version of ANOVA**, but it compares **many outputs at once**, not just one.

📊 Now the Results You Got:

```yaml
Wilks' lambda:         0.0019    p = 0.0000
Pillai's trace:        0.9981    p = 0.0000
Hotelling-Lawley:    515.2297    p = 0.0000
Roy's greatest root: 515.2297    p = 0.0000
```

🔍 Focus on: **Pr > F (p-value)**

This is the "Is it significant?" column.

- **p = 0.0000** (actually, very very small) means:
  ✅ **YES, there are significant differences** between the algorithms.

🧠 What Each Test Means (you don't need to know all):

Test Name	Meaning in simple terms
Wilks' Lambda	Lower = better separation between groups
Pillai's Trace	Higher = stronger evidence for group differences
Hotelling-Lawley	Also shows differences, more sensitive when groups are small
Roy's Root	Looks at the biggest difference across groups

All of them say the **same thing here**:

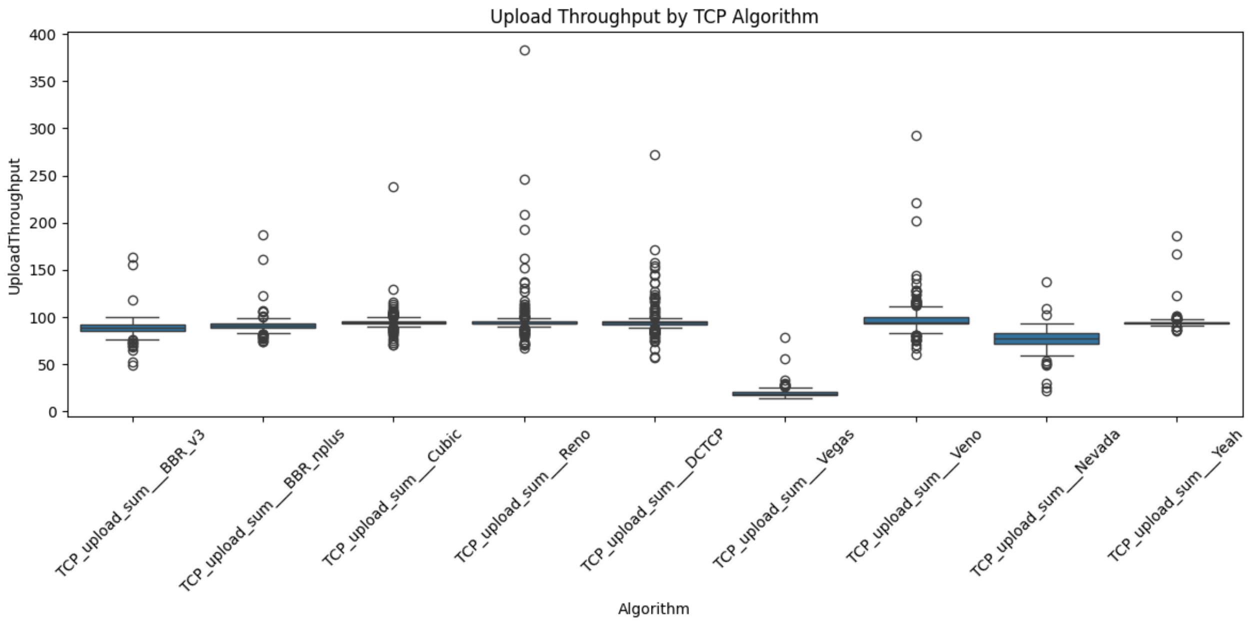
> ❗ Your TCP algorithms behave **differently** in terms of upload throughput.

✅ Final Answer (TL;DR):

> The MANOVA results say: **Yes, the different TCP algorithms give different upload performance**, and it's statistically very strong ($p < 0.0001$).

Visualization of Throughput

```
# Boxplot for visual comparison
plt.figure(figsize=(12, 6))
sns.boxplot(data=df_long, x='Algorithm', y='UploadThroughput')
plt.xticks(rotation=45)
plt.title("Upload Throughput by TCP Algorithm")
plt.tight_layout()
plt.show()
```



Tukey Analysis (Additional Work)

```
# Tukey's HSD test: pairwise comparison of all algorithms
tukey = pairwise_tukeyhsd(
    endog=df_long['UploadThroughput'],
    groups=df_long['Algorithm'],
    alpha=0.05
)

# Print the Tukey test results
print(tukey.summary())
```

```
                    Multiple Comparison of Means - Tukey HSD, FWER=0.05
=========================================================================================
         group1                    group2          meandiff p-adj   lower    upper   reject
-----------------------------------------------------------------------------------------
TCP_upload_sum___BBR_nplus TCP_upload_sum___BBR_v3  -3.0228 0.0957  -6.2888   0.2431  False
TCP_upload_sum___BBR_nplus  TCP_upload_sum___Cubic   4.0084 0.0045   0.7425   7.2744   True
TCP_upload_sum___BBR_nplus  TCP_upload_sum___DCTCP   4.5561 0.0005   1.2901    7.822   True
TCP_upload_sum___BBR_nplus TCP_upload_sum___Nevada -14.1294    0.0 -17.3954 -10.8635   True
TCP_upload_sum___BBR_nplus   TCP_upload_sum___Reno   5.5859    0.0     2.32   8.8519   True
TCP_upload_sum___BBR_nplus  TCP_upload_sum___Vegas -71.8107    0.0 -75.0767 -68.5448   True
TCP_upload_sum___BBR_nplus   TCP_upload_sum___Veno    6.803    0.0    3.537  10.0689   True
TCP_upload_sum___BBR_nplus   TCP_upload_sum___Yeah   3.5191 0.0236   0.2531    6.785   True
   TCP_upload_sum___BBR_v3  TCP_upload_sum___Cubic   7.0312    0.0   3.7653  10.2972   True
   TCP_upload_sum___BBR_v3  TCP_upload_sum___DCTCP   7.5789    0.0   4.3129  10.8448   True
   TCP_upload_sum___BBR_v3 TCP_upload_sum___Nevada -11.1066    0.0 -14.3726  -7.8407   True
   TCP_upload_sum___BBR_v3   TCP_upload_sum___Reno   8.6088    0.0   5.3428  11.8747   True
   TCP_upload_sum___BBR_v3  TCP_upload_sum___Vegas -68.7879    0.0 -72.0538  -65.522   True
   TCP_upload_sum___BBR_v3   TCP_upload_sum___Veno   9.8258    0.0   6.5598  13.0917   True
   TCP_upload_sum___BBR_v3   TCP_upload_sum___Yeah   6.5419    0.0    3.276   9.8078   True
    TCP_upload_sum___Cubic  TCP_upload_sum___DCTCP   0.5476 0.9999  -2.7183   3.8136  False
    TCP_upload_sum___Cubic TCP_upload_sum___Nevada -18.1379    0.0 -21.4038 -14.8719   True
    TCP_upload_sum___Cubic   TCP_upload_sum___Reno   1.5775 0.8562  -1.6884   4.8435  False
    TCP_upload_sum___Cubic  TCP_upload_sum___Vegas -75.8191    0.0 -79.0851 -72.5532   True
    TCP_upload_sum___Cubic   TCP_upload_sum___Veno   2.7945 0.1643  -0.4714   6.0605  False
    TCP_upload_sum___Cubic   TCP_upload_sum___Yeah  -0.4893 0.9999  -3.7553   2.7766  False
    TCP_upload_sum___DCTCP TCP_upload_sum___Nevada -18.6855    0.0 -21.9514 -15.4195   True
    TCP_upload_sum___DCTCP   TCP_upload_sum___Reno   1.0299 0.9878  -2.2361   4.2958  False
    TCP_upload_sum___DCTCP  TCP_upload_sum___Vegas -76.3668    0.0 -79.6327 -73.1008   True
    TCP_upload_sum___DCTCP   TCP_upload_sum___Veno   2.2469  0.449   -1.019   5.5128  False
    TCP_upload_sum___DCTCP   TCP_upload_sum___Yeah   -1.037 0.9872  -4.3029    2.229  False
   TCP_upload_sum___Nevada   TCP_upload_sum___Reno  19.7154    0.0  16.4494  22.9813   True
   TCP_upload_sum___Nevada  TCP_upload_sum___Vegas -57.6813    0.0 -60.9472 -54.4153   True
   TCP_upload_sum___Nevada   TCP_upload_sum___Veno  20.9324    0.0  17.6664  24.1983   True
   TCP_upload_sum___Nevada   TCP_upload_sum___Yeah  17.6485    0.0  14.3826  20.9145   True
     TCP_upload_sum___Reno  TCP_upload_sum___Vegas -77.3967    0.0 -80.6626 -74.1307   True
     TCP_upload_sum___Reno   TCP_upload_sum___Veno    1.217 0.9652  -2.0489    4.483  False
     TCP_upload_sum___Reno   TCP_upload_sum___Yeah  -2.0669 0.5685  -5.3328   1.1991  False
    TCP_upload_sum___Vegas   TCP_upload_sum___Veno  78.6137    0.0  75.3477  81.8796   True
    TCP_upload_sum___Vegas   TCP_upload_sum___Yeah  75.3298    0.0  72.0639  78.5957   True
     TCP_upload_sum___Veno   TCP_upload_sum___Yeah  -3.2839 0.0475  -6.5498  -0.0179   True
-----------------------------------------------------------------------------------------
```

Results of Tukey

🔍 What Tukey's Test Is Saying

The Tukey HSD test compares **every pair** of algorithms and asks:

> "Are their average upload speeds significantly different?"

You care most about the last column:

✅ `reject == True` → Significant difference

❌ `reject == False` → No significant difference

🧠 Summary of Key Insights

✅ Some BIG Differences (very clear and significant):

- `Vegas` is **much slower** than almost everything else
  (e.g. vs. BBR, Reno, Cubic, etc. — all `reject == True`)
- `Nevada` is **much faster** than many others like Cubic, DCTCP, Yeah, etc.

❌ Some Similar Performance (no significant difference):

- `BBR-n+` vs `BBR-v3`: ❌ `p = 0.0957`, not significant
- `Cubic` vs `DCTCP`: ❌ `p ≈ 1.0`, nearly identical
- `Cubic` vs `Reno`: ❌ no significant difference
- `Reno` vs `Veno` or `Yeah`: ❌ very similar

Start coding or generate with AI.

📊 Interpretation in Simple Terms

Here's a **child-level explanation**:

Imagine you're testing 9 race cars (algorithms). You run each one, and time how fast they go (upload speed).
Now you compare:

- 🚀 Some are **super fast** (like Nevada, BBR-v3)
- 🐢 Some are **really slow** (like Vegas)
- ▯ Some are **too close to call** — they perform almost the same (like Cubic vs DCTCP)

Start coding or generate with AI.