# QQ Plot (VIMP)

## QQ Plot (VIMP)

## Q. How to find if a given distribution is normal or not?

### 1. Visual Methods (Quick Check)

**A. Histogram**

**B. Q-Q Plot (Quantile-Quantile Plot)**

### 2. Statistical Tests (Mathematical Check)

| Test | Null Hypothesis ($H_0$: Data is normal?) | If $p < 0.05$ | If $p > 0.05$ |
|---|---|---|---|
| Shapiro-Wilk (Best for Small Data) | Data follows a normal distribution | Reject $H_0$ (Not normal) | Fail to reject $H_0$ (Normal) |
| Kolmogorov-Smirnov | Data follows a normal distribution | Reject $H_0$ (Not normal) | Fail to reject $H_0$ (Normal) |
| Anderson-Darling | Data follows a normal distribution | Check critical values | Check critical values |
| D'Agostino's $K^2$ | Data follows a normal distribution | Reject $H_0$ (Not normal) | Fail to reject $H_0$ (Normal) |

### Conclusion (Which Method to Use?)

| **Method** | **Use When?** |
|---|---|
| Histogram | Quick visual check |
| Q-Q Plot | Best for detecting deviations |

| | |
|---|---|
| Shapiro-Wilk | Best for small datasets (<5000 samples) |
| D'Agostino's $K^2$ | Best for medium/large datasets |
| Kolmogorov-Smirnov | Best for comparing to normal distribution |
| Skewness/Kurtosis | Quick numerical check |

👉 For small data: Use Shapiro-Wilk + Q-Q Plot.

👉 For large data: Use D'Agostino's $K^2$ + Histogram/Q-Q Plot.

# Q-Q Plot

- Quantile-Quantile plot
- A **Q-Q plot** is a graphical tool to check **whether a dataset follows a normal distribution**.

💡 **Generally used to check the Normality of data.**

## What Does a Q-Q Plot Do?

- It compares **the quantiles (percentiles) of your data** with **the quantiles of a normal distribution**.
- If data is normally distributed, the points **should lie along a straight diagonal line**.

## How to Interpret a Q-Q Plot?

| Pattern | Interpretation |
|---|---|
| **Points lie along the diagonal** | Data is normally distributed |
| **Points curve upward (S-shape)** | Data is right-skewed (positive skew) |
| **Points curve downward (reverse S-shape)** | Data is left-skewed (negative skew) |
| **Extreme deviations at ends (tails higher or lower than expected)** | Heavy-tailed or light-tailed distribution |

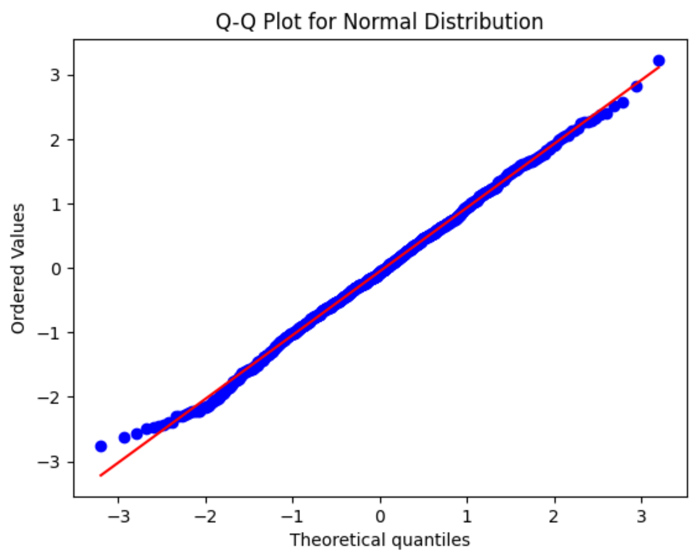
```
import numpy as np
import matplotlib.pyplot as plt
```

```
import scipy.stats as stats

# Example data (e.g., test scores or any dataset)
data = np.random.normal(size=1000)  # Normally distributed data

# Create a Q-Q plot
stats.probplot(data, dist="norm", plot=plt)

# Show the plot
plt.title('Q-Q Plot for Normal Distribution')
plt.show()
```




`stats.probplot(data, dist="norm", plot=plt)`: creates the Q-Q plot comparing your data against the normal distribution.

**`probplot()`** is a function from the **`scipy.stats`** module that generates a **Q-Q plot** (Quantile-Quantile plot) for comparing your data against a theoretical distribution.

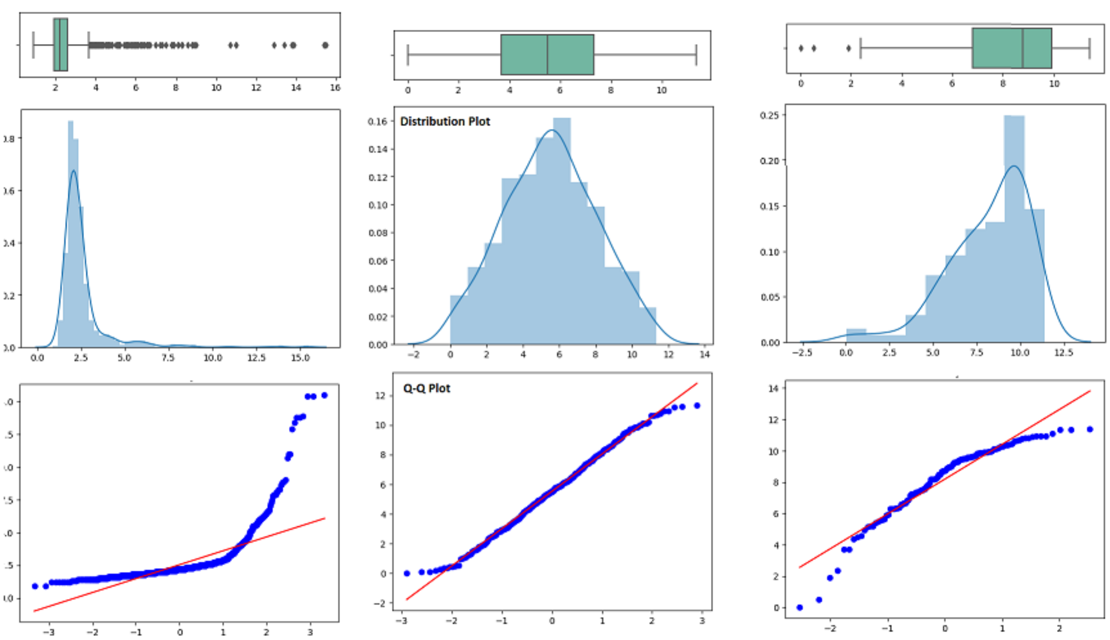
- `dist="norm"` specifies that we are comparing the data to a **normal distribution**.
- `plot=plt` tells it to use **matplotlib** for plotting.
    - `dist="expon"` for the **exponential distribution**.
    - `dist="gamma"` for the **gamma distribution**.

## How to Interpret:

- If the data follows a normal distribution, the points on the Q-Q plot should form a straight line, indicating that the **quantiles** of the data match the **theoretical quantiles** of a normal distribution.
- If the points deviate from the line, particularly at the ends, it suggests that the data is **not normal**.

# Q-Q Plot

- It is particularly useful for determining whether a set of data follows a normal distribution.




- In Q-Q- Plot, you take X & Y.
- You already know the distribution of Y (eg. Normally Distributed)
- You compare your data with the normal distribution

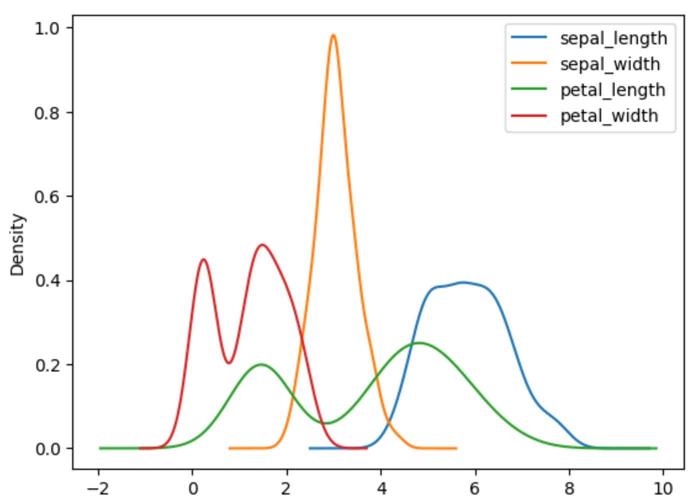
## Steps:

1. Generate theoretical data (Normal Distribution)

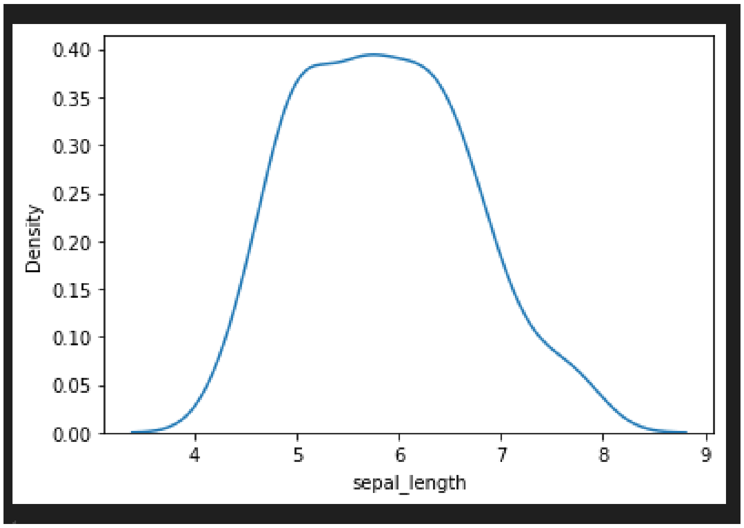
2. Sort your own data
3. Calculate quantiles of your data
4. Do the same with theoretical data
5. Now that you have both the quantiles, you plot them on a graph
    - Scatter plot
6. Compare

**We will use the `iris` df:**

```
df = sns.load_dataset('iris')
df.plot(kind='kde')
```




```
sns.kdeplot(df['sepal_length'])
```




```
temp = sorted(df['sepal_length'].tolist())
```

1. `df['sepal_length']`: This accesses the column named `'sepal_length'` from the DataFrame `df`.
2. `.tolist()`: This converts the `'sepal_length'` column into a list.
3. `sorted(...)`: This sorts the list in ascending order.
4. `temp = ...`: This assigns the sorted list to the variable `temp`.

- Now, calculate percentiles of this and store in a variable.

```
np.percentile(temp, 100)

Output: 7.9
```

- Calculates 100th percentile
- We have to calculate 1 to 100
  - So, we need to run a loop and store these in a list

```
y_quant=[]

for i in range(1,101):
    y_quant.append(np.percentile(temp, i))
```

```
print(y_quant)

Output: [4.3, 4.4, 4.4, 4.547, 4.6, 4.6, 4.694, 4.743, 4.8, 4.8, 4.
8, 4.9, 4.9, 4.9, 4.9, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.029, 5.1, 5.
1, 5.1, 5.1, 5.1, 5.123, 5.2, 5.2, 5.27, 5.4, 5.4, 5.4, 5.4, 5.5, 5.
5, 5.5, 5.5, 5.511, 5.6, 5.6, 5.6, 5.606999999999999, 5.7, 5.7, 5.7,
5.7, 5.7, 5.8, 5.8, 5.8, 5.8, 5.8, 5.9, 5.9, 6.0, 6.0, 6.0, 6.0, 6.
1, 6.1, 6.1, 6.1, 6.2, 6.2, 6.234, 6.3, 6.3, 6.3, 6.3, 6.3, 6.328,
6.4, 6.4, 6.4, 6.4, 6.473000000000001, 6.5, 6.5, 6.5200000000000005,
6.6, 6.7, 6.7, 6.7, 6.7, 6.7, 6.763, 6.8, 6.861000000000015, 6.9,
6.9, 7.008000000000001, 7.156999999999999, 7.2, 7.254999999999998,
7.407999999999999, 7.6530000000000005, 7.7, 7.7, 7.9]
```

- Now we got percentiles of our data
- Now we need to generate a normal data & its percentiles

```
samples = np.random.normal(loc=0,scale=1,size=100)
```

- `loc=0,scale=1` is optional
- We got 100 normally distributed samples

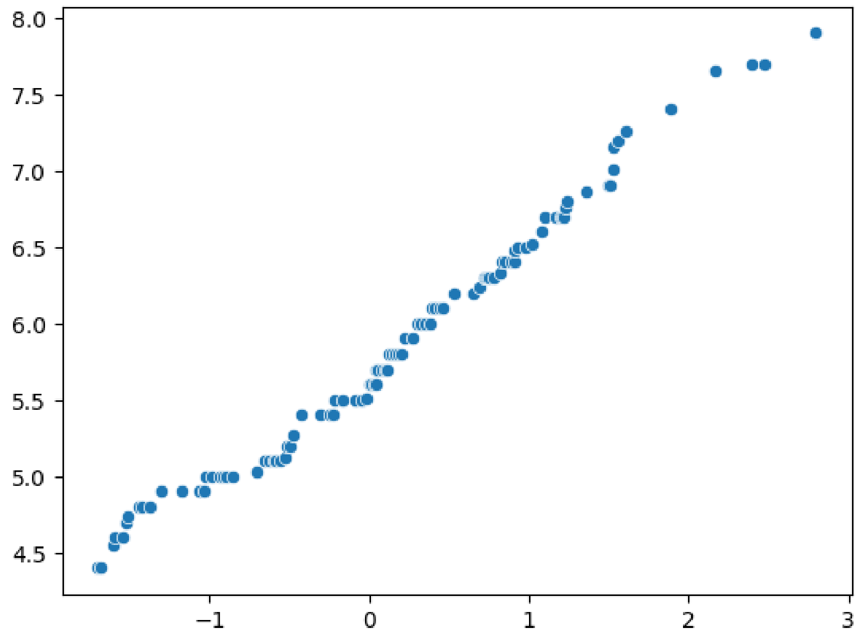
Calculate the quartile of the above data & store it in a variable

```
x_quant = []

for i in range(1,101):
  x_quant.append(np.percentile(samples,i))
```

Now plot a scatterplot:

```
sns.scatterplot(x=x_quant,y=y_quant)
```




## Using statsmodel

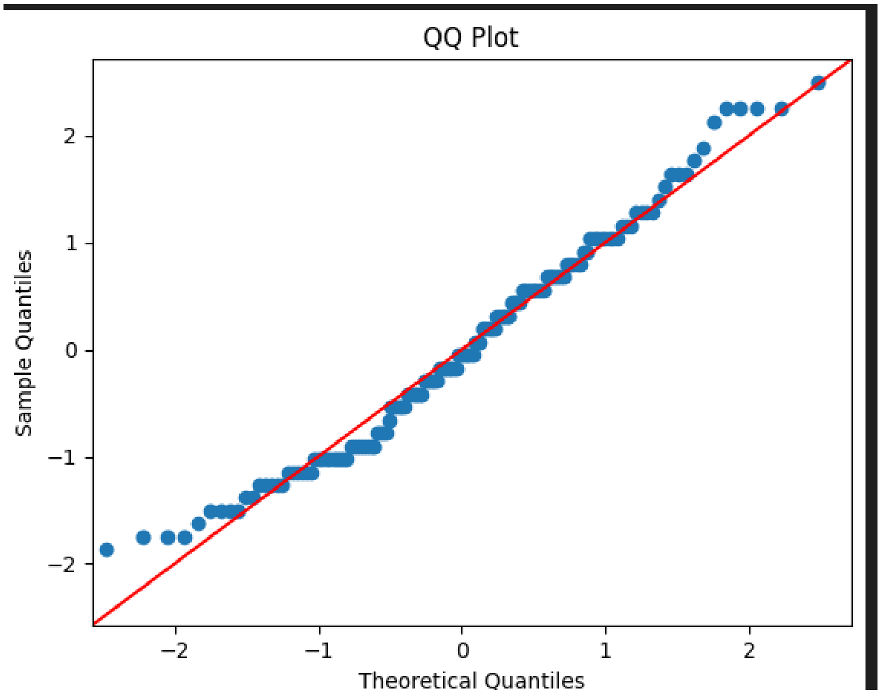
- You can also create a **Q-Q plot** using the `statsmodels` library, which provides more customization options than `scipy.stats.probplot`
- Documentation → https://www.statsmodels.org/dev/generated/statsmodels.graphics.gofplots.qqplot.html

```
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Create a QQ plot of the two sets of data
fig = sm.qqplot(df['sepal_length'], line='45', fit=True)

# Add a title and labels to the plot
plt.title('QQ Plot')
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Sample Quantiles')

# Show the plot
plt.show()
```

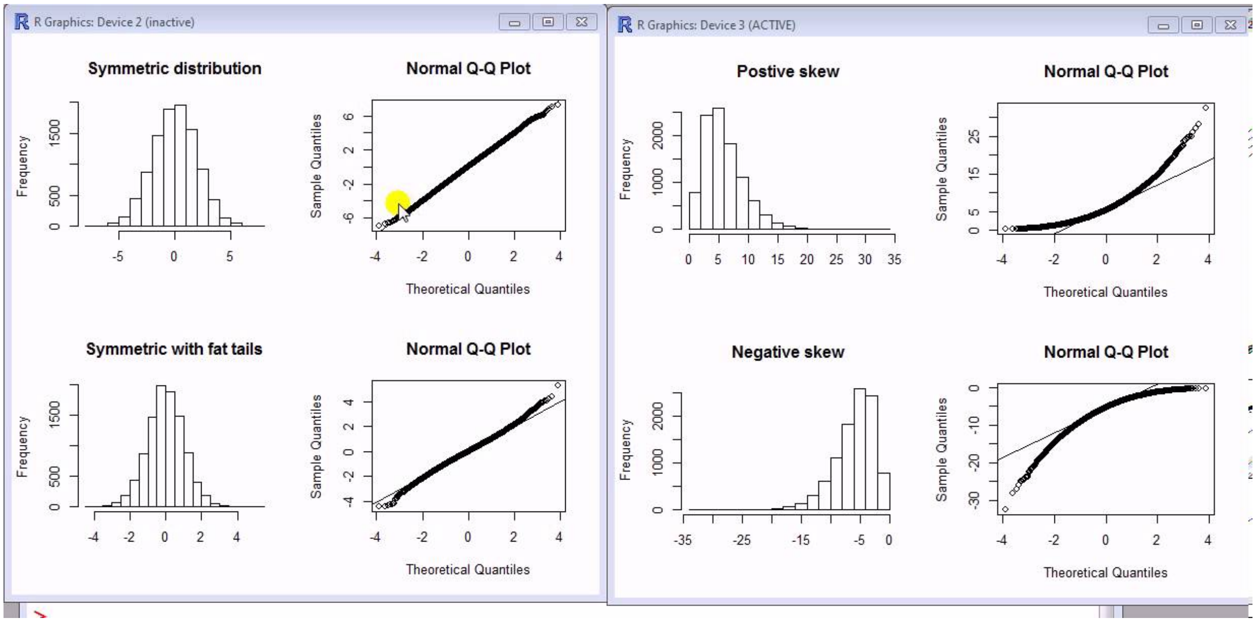


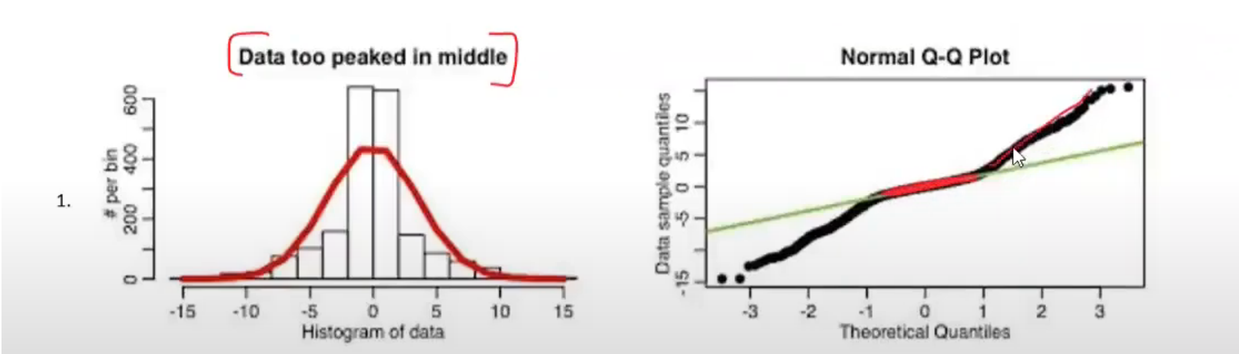

`line='45'` is an option that adds a reference line (the line `y=x`, which represents the ideal result if the data were normally distributed).

| Parameter | What It Does |
| --- | --- |
| `'45'` | Draws a **45-degree diagonal line** through the origin (not adjusted for mean & standard deviation). |
| `'s'` | **Standardized line** → Passes through the **first and third quartiles** (Q1 and Q3). |
| `'r'` | **Regression line** → Best-fit line based on **least squares regression**. |
| `'q'` | **Quartile line** → Passes through **theoretical and sample quartiles** (Q1 and Q3). |
| `None` | **No reference line** (just plots the data points). |

🔷 If data is **normal** → Points will **closely follow the red diagonal line.**
🔷 If data is
**right-skewed (positive skew)** → Points **curve upwards**.
🔷 If data is
**left-skewed (negative skew)** → Points **curve downwards**.
🔷 If tails are
**heavy (extreme values)** → Points **deviate at the ends**.

## How to interpret QQ plots:




## Does QQ plot only detect normal distribution?

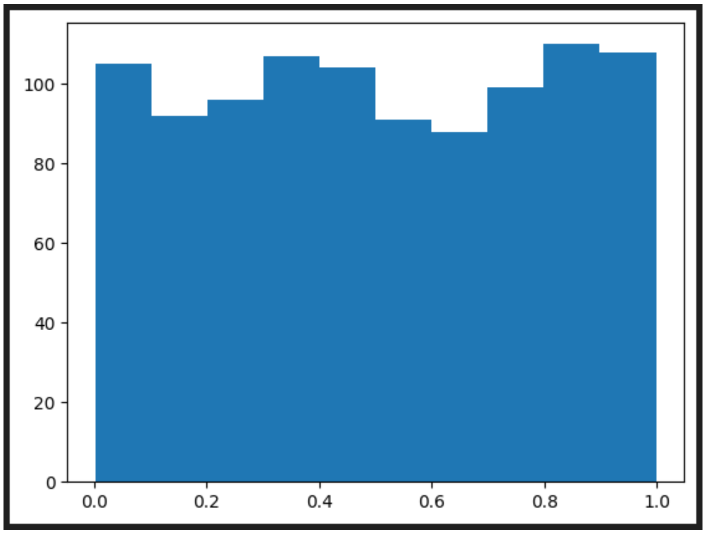
- **NO.**

Let’s generate a uniform distribution

```
x = np.random.uniform(low=0, high=1, size=1000)
```

we generated points in the range of **0 to 1**

```
plt.hist(x)
```




- Now a uniform distribution to the data

```
params = stats.uniform.fit(x)
dist = stats.uniform(loc=params[0], scale=params[1])
```

`params = stats.uniform.fit(x)`

- **How it works:**
  - The `fit()` method in **SciPy's** `stats.uniform` finds the parameters for a uniform distribution by using:
    - `loc = min(x)` → The smallest value in `x` (the starting point of the distribution).
    - `scale = max(x) - min(x)` → The range (difference between max and min).
- **Output:**
  - `params` will be a tuple `(loc, scale)`, where:
    - `loc` → Minimum value in `x`
    - `scale` → (Maximum - Minimum)

```
params

Output: (0.0014296235821444903, 0.9971388444141985)
```

- Gives us min & Maximum - Minimum

`stats.uniform(loc=params[0], scale=params[1])`

  - Create a Uniform Distribution object using the fitted parameters.
    - Min & Maximum - Minimum

The `stats.uniform` function represents a **continuous uniform distribution**

where:

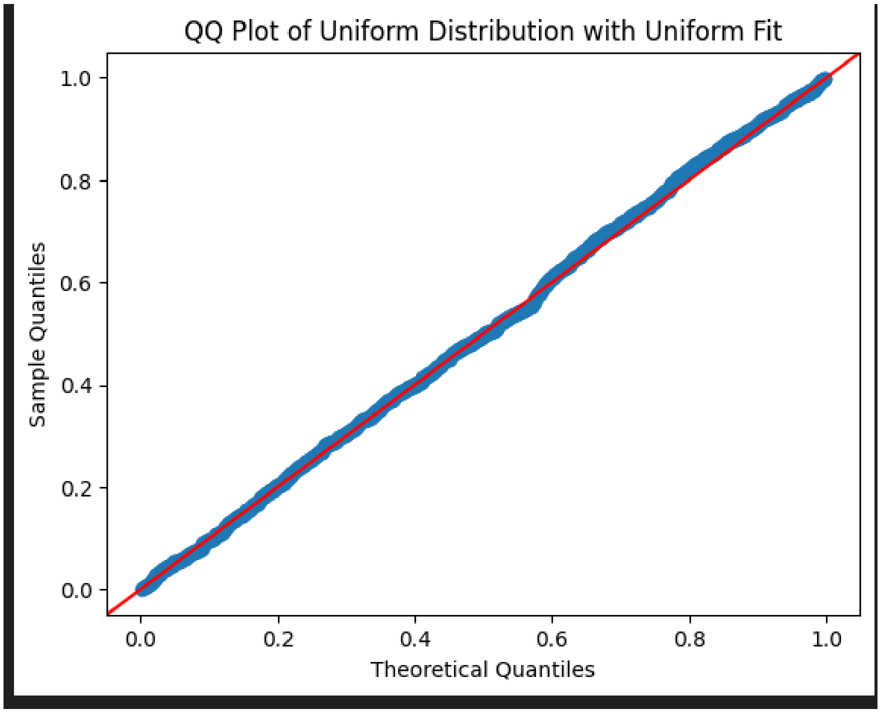
  - `loc` = **Start point** (minimum value)
  - `scale` = **Width** (max-min range)

**Create a Q-Q Plot:**

```
# Create a QQ plot of the data using the uniform distribution
fig = sm.qqplot(x, dist=dist, line='45')

# Add a title and labels to the plot
plt.title('QQ Plot of Uniform Distribution with Uniform Fit')
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Sample Quantiles')

# Show the plot
plt.show()
```




- `x`: The dataset you're testing for uniformity. This could be any set of values, and it will be compared to the uniform distribution.
- `dist=dist`: This argument specifies the distribution you're comparing your data to. Here, `dist` is the uniform distribution object you created earlier using the fitted parameters (`loc` and `scale`).
    - In previous example of normal distribution, we used `dist="norm"` cuz it is defined already
        - Even if you don't explicitly mention `dist="norm"`, it assumes it's a normal distribution
        - By default, `sm.qqplot()` assumes that you want to compare the data against the **standard normal distribution** (with mean 0 and standard deviation 1) unless you specify otherwise.
    - But here, for uniform distribution, we have to create an object

## `scipy.stats.probplot()` vs `statsmodels.api.qqplot()`

| Feature | `scipy.stats.probplot()` | `statsmodels.api.qqplot()` |
| --- | --- | --- |
| Basic Syntax | `stats.probplot(x, dist="norm", plot=plt)` | `sm.qqplot(x, dist=stats.norm, line='45')` |

| **Feature** | `scipy.stats.probplot()` | `statsmodels.api.qqplot()` |
| --- | --- | --- |
| Distribution Specification | "norm", "uniform", "expon" (string input) | stats.norm, stats.uniform, etc. (**must use function reference**) 🔴 |
| Fitted Distributions | ❌ No | ✅ Yes ( `dist=stats.norm(loc, scale` )) |
| Line Options | ❌ Only regression line | ✅ `'45', 's', 'r', 'q'` |
| Regression Line Included | ✅ Yes, by default | ✅ Optional (line='r') |
| Customization (e.g., Labels, Titles) | ✅ Works with matplotlib | ✅ Works with matplotlib |
| Matplotlib Subplot Support | ✅ `(plot=ax)` | ✅ `(fig=sm.qqplot(..., ax=ax))` |
| Ease of Use | ✅ Simpler | ❌ Slightly more complex |
| Best for Normality Testing | ✅ Yes | ✅ Yes |
| Best for Comparing Against Fitted Distributions | ❌ No | ✅ Yes |