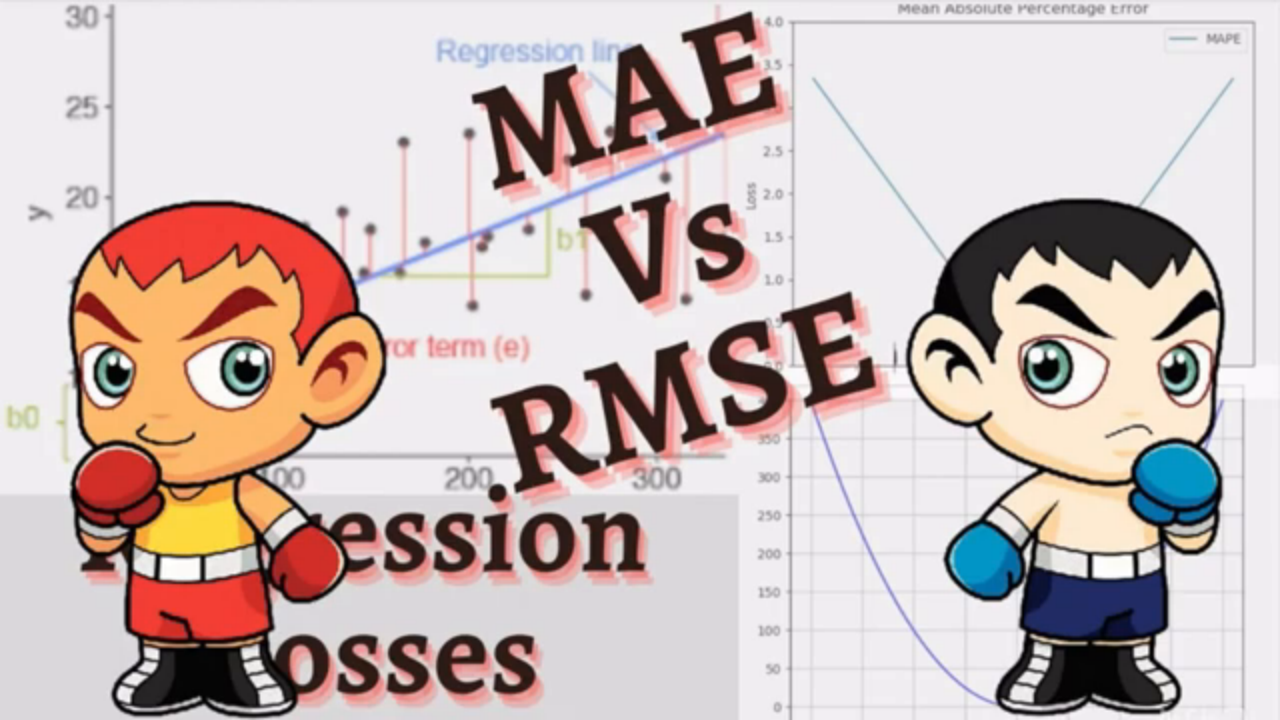
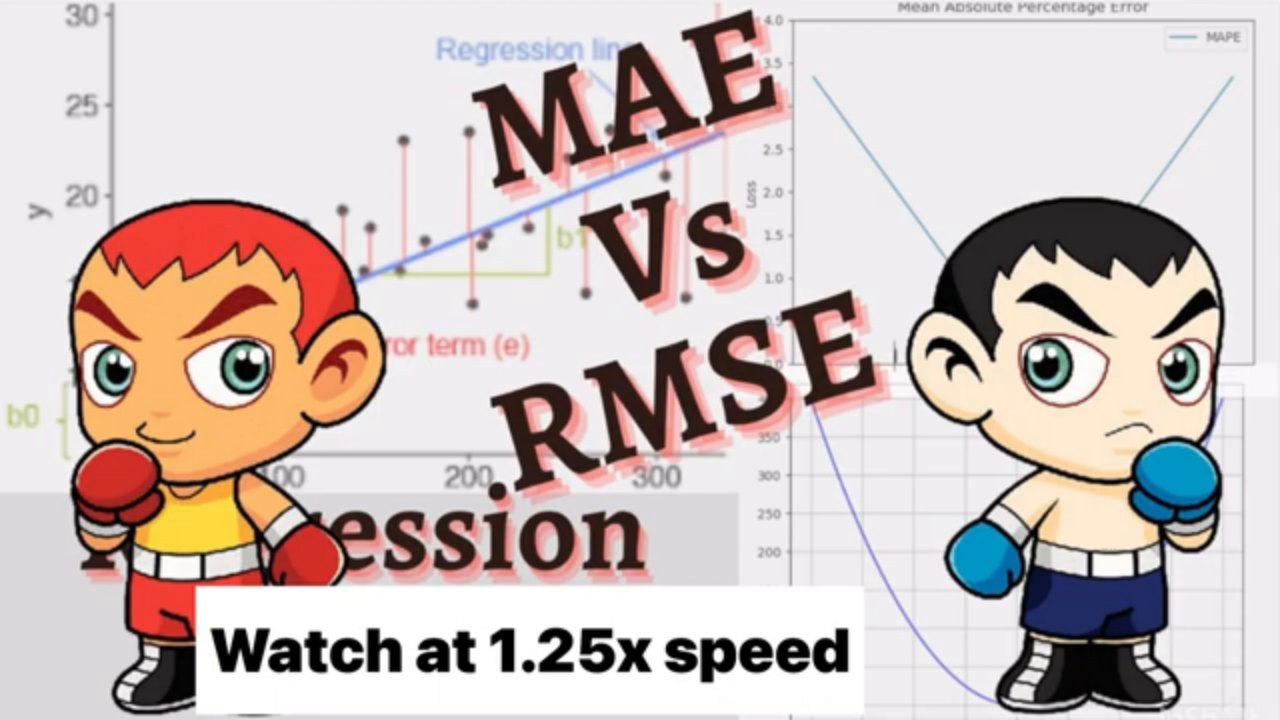
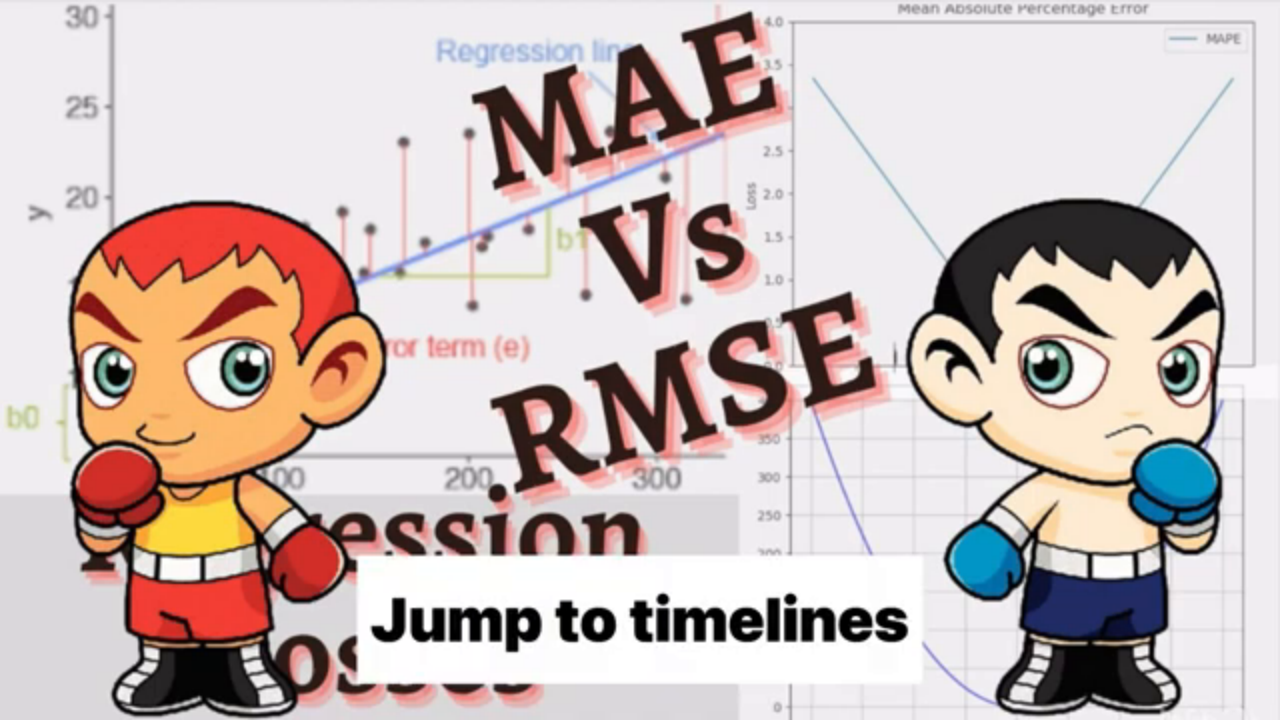
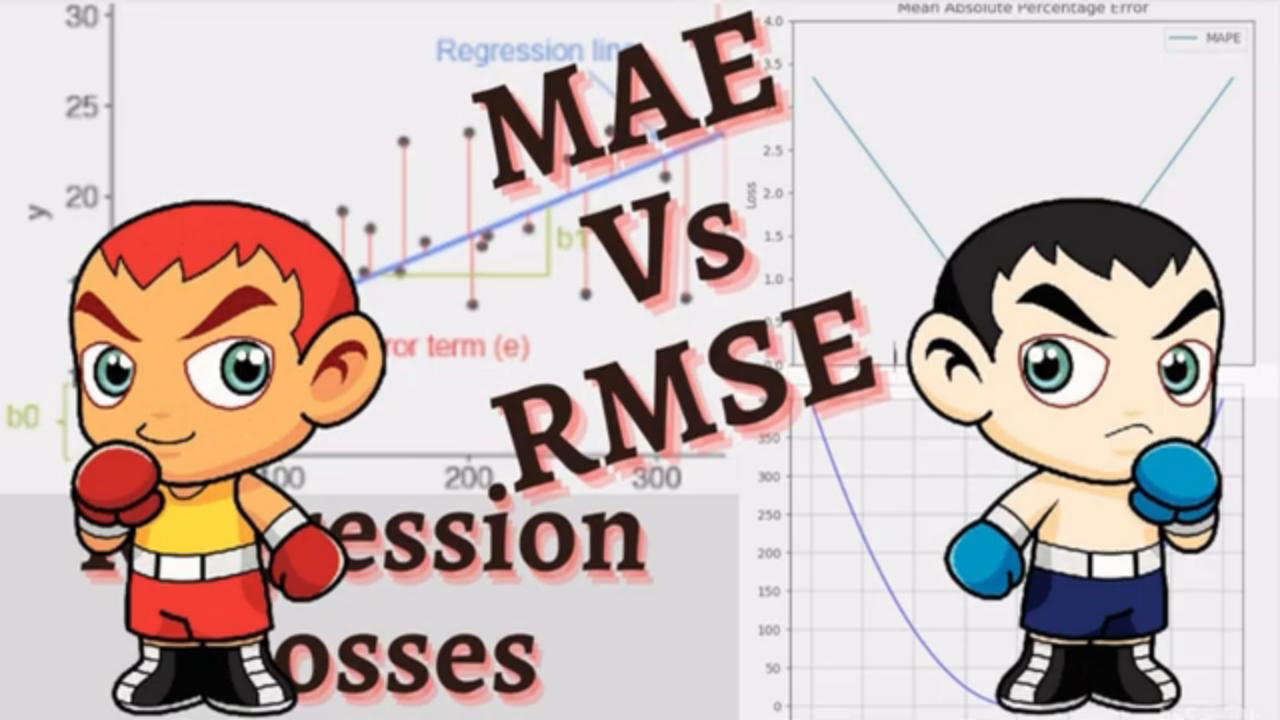
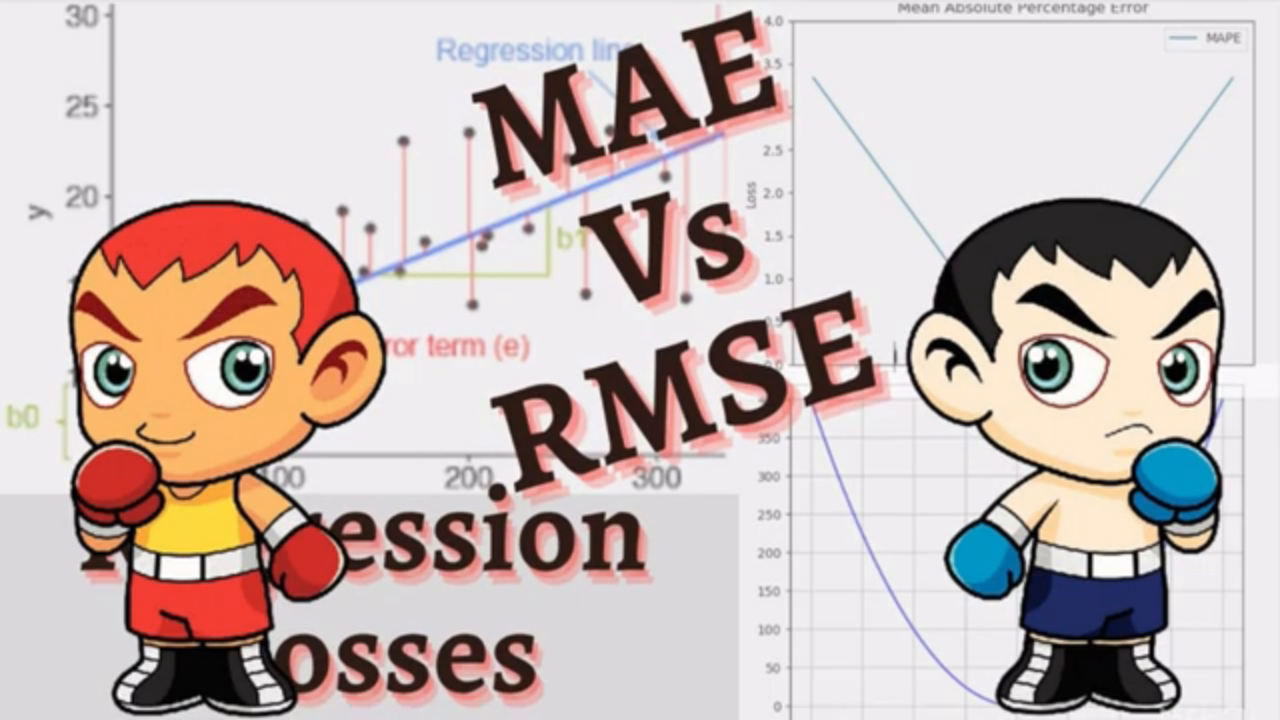

MAE
Vs
RMSE
Regression losses
Regression line
Mean Absolute Percentage Error
MAPE
Loss


MAE
Vs
RMSE
Watch at 1.25x speed

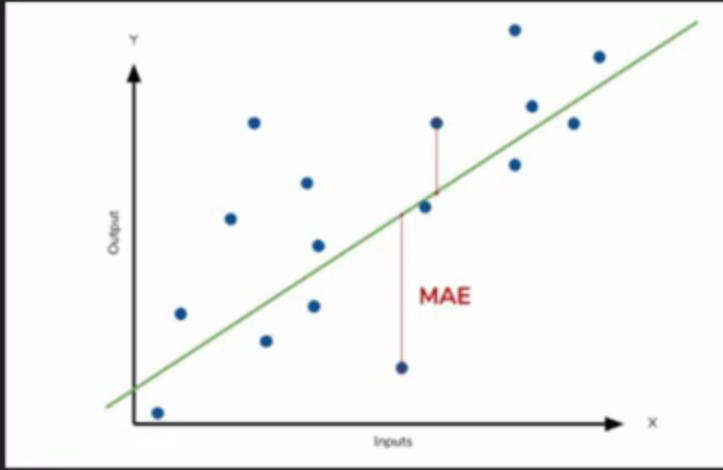

MAE
Vs
RMSE
Jump to timelines


MAE
Vs
RMSE
Regression lin
Mean Absolute Percentage Error
MAPE
Loss
y
b0
b1
or term (e)
30
25
20
200
300
4.0
3.5
2.5
2.0
1.5
1.0
350
300
250
200
150
100
50
0


MAE
Vs
RMSE
Regression lin
Mean Absolute Percentage Error
MAPE
Loss
y
b0
b1
or term (e)
200
300
esion
osses

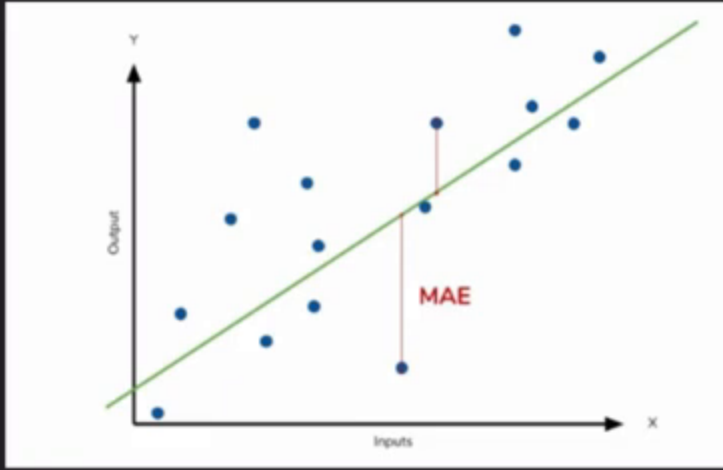
MAE (Mean Absolute Error)

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$



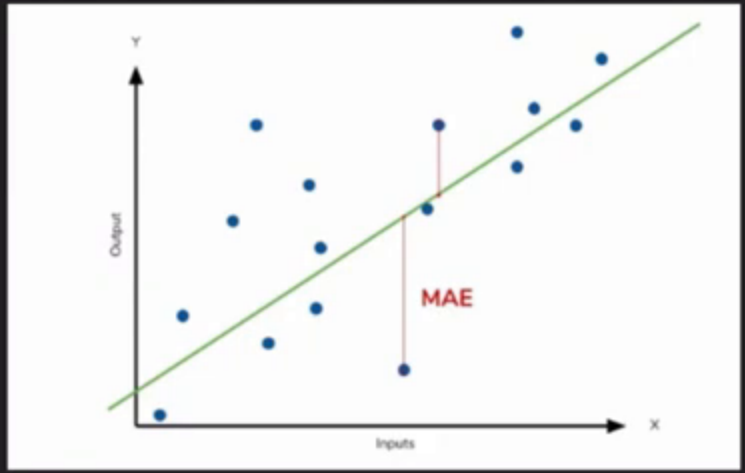
MAE (Mean Absolute Error)

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$



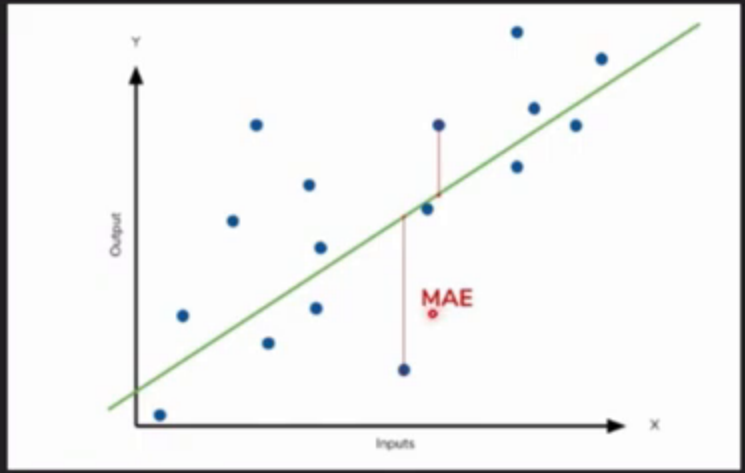
MAE (Mean Absolute Error)

$$\text{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$



MAE (Mean Absolute Error)

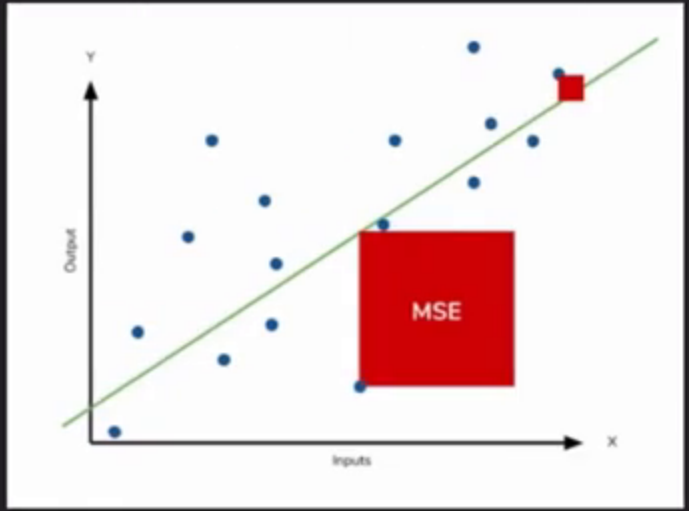
$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n} |y_j - \hat{y}_j|$$



MSE (Mean Squared Error)

$$MSE = \frac{1}{n} \Sigma \underbrace{\left(y - \hat{y} \right)}^{2}$$

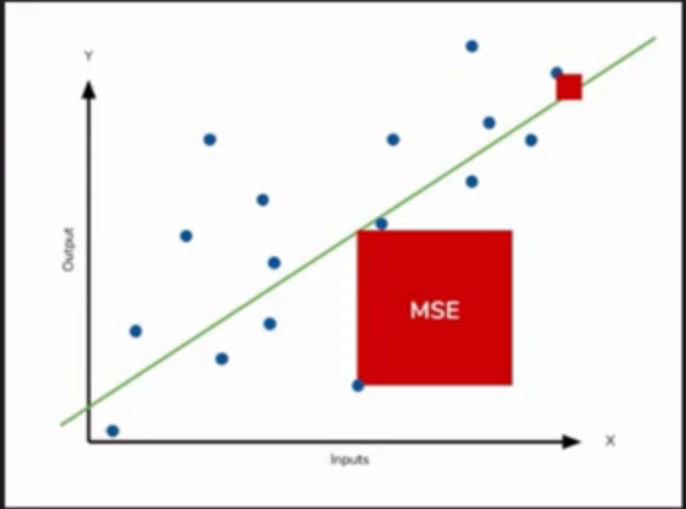
The square of the difference between actual and predicted



MSE (Mean Squared Error)

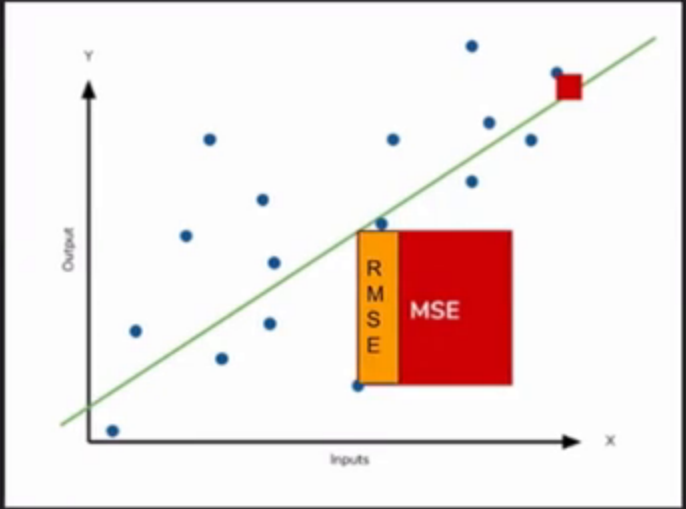
$$MSE = \frac{1}{n} \Sigma \underbrace{\left(y - \hat{y} \right)}^{2}$$

The square of the difference between actual and predicted



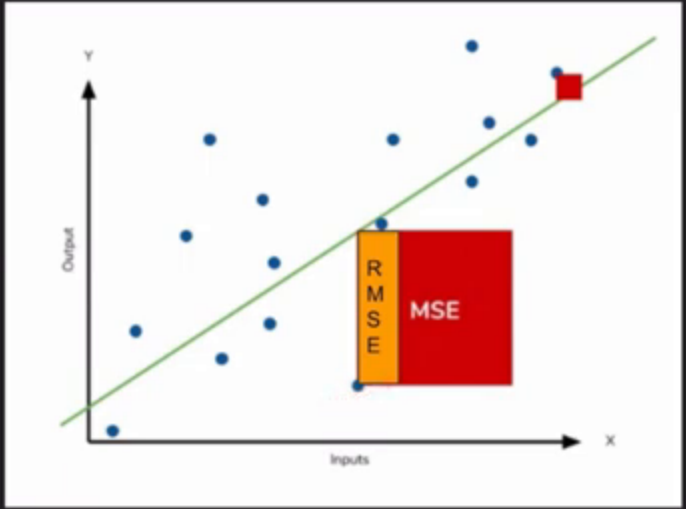
RMSE (Root Mean Squared Error)

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$$



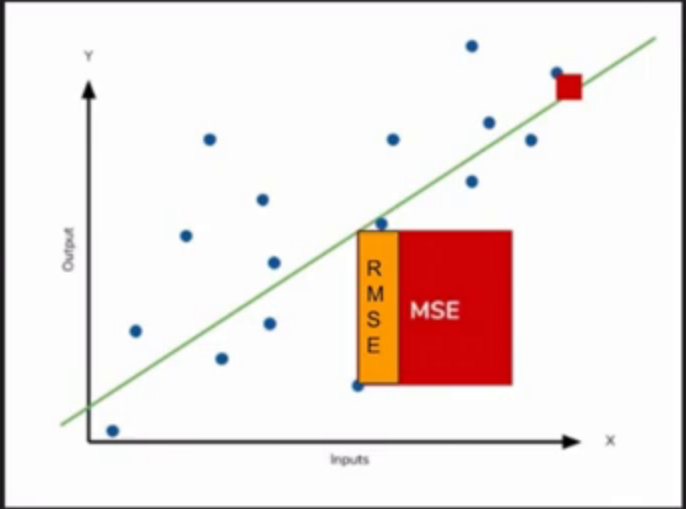
RMSE (Root Mean Squared Error)

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$$



RMSE (Root Mean Squared Error)

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$$



Similarities

- Distance based metrics
- Negatively oriented scores - lower values are better
- Range - 0 to ∞
- Mean or Average prediction across whole dataset

Similarities

- Distance based metrics
- Negatively oriented scores - lower values are better
- Range - 0 to ∞
- Mean or Average prediction across whole dataset

Similarities

- Distance based metrics
- Negatively oriented scores - lower values are better
- Range - 0 to ∞
- Mean or Average prediction across whole dataset

Differences

- MSE or RMSE gives high weightage to large errors
- MAE treats all errors the same way
- MAE <= RMSE
- MAE is easily interpretable
- RMSE is more sensitive to noise or outliers
- MSE/RMSE are smooth functions and easy to differentiate

Differences

- MSE or RMSE gives high weightage to large errors
- MAE treats all errors the same way
- MAE <= RMSE
- MAE is easily interpretable
- RMSE is more sensitive to noise or outliers
- MSE/RMSE are smooth functions and easy to differentiate

Differences

- MSE or RMSE gives high weightage to large errors
- MAE treats all errors the same way
- MAE <= RMSE
- MAE is easily interpretable
- RMSE is more sensitive to noise or outliers
- MSE/RMSE are smooth functions and easy to differentiate

Differences

- MSE or RMSE gives high weightage to large errors
- MAE treats all errors the same way
- MAE <= RMSE
- MAE is easily interpretable
- RMSE is more sensitive to noise or outliers
- MSE/RMSE are smooth functions and easy to differentiate

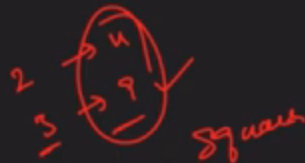
Differences

- MSE or RMSE gives high weightage to large errors
- MAE treats all errors the same way
- MAE <= RMSE
- MAE is easily interpretable
- RMSE is more sensitive to noise or outliers
- MSE/RMSE are smooth functions and easy to differentiate

Differences

- MSE or RMSE gives high weightage to large errors
- MAE treats all errors the same way
- MAE <= RMSE
- MAE is easily interpretable
- RMSE is more sensitive to noise or outliers
- MSE/RMSE are smooth functions and easy to differentiate

Differences



- MSE or RMSE gives high weightage to large errors
- MAE treats all errors the same way
- MAE <= RMSE
- MAE is easily interpretable
- RMSE is more sensitive to noise or outliers
- MSE/RMSE are smooth functions and easy to differentiate

Differences

- MSE or RMSE gives high weightage to large errors
- MAE treats all errors the same way
- MAE <= RMSE
- MAE is easily interpretable
- RMSE is more sensitive to noise or outliers
- MSE/RMSE are smooth functions and easy to differentiate

Differences

- MSE or RMSE gives high weightage to large errors
- MAE treats all errors the same way
- MAE <= RMSE
- MAE is easily interpretable
- RMSE is more sensitive to noise or outliers
- MSE/RMSE are smooth functions and easy to differentiate

Differences

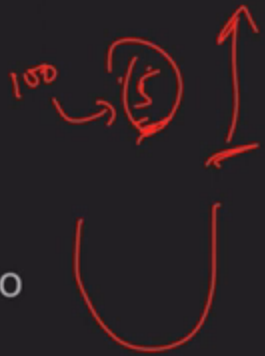
- MSE or RMSE gives high weightage to large errors
- MAE treats all errors the same way
- MAE <= RMSE
- MAE is easily interpretable
- RMSE is more sensitive to noise or outliers
- MSE/RMSE are smooth functions and easy to differentiate

Differences

- MSE or RMSE gives high weightage to large errors
- MAE treats all errors the same way
- MAE <= RMSE
- MAE is easily interpretable
- RMSE is more sensitive to noise or outliers
- MSE/RMSE are smooth functions and easy to differentiate

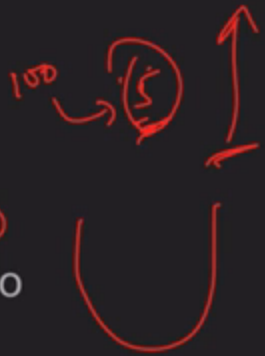
Differences

- MSE or RMSE gives high weightage to large errors
- MAE treats all errors the same way
- MAE <= RMSE
- MAE is easily interpretable
- RMSE is more sensitive to noise or outliers
- MSE/RMSE are smooth functions and easy to differentiate



Differences

- MSE or RMSE gives high weightage to large errors
- MAE treats all errors the same way
- MAE <= RMSE
- MAE is easily interpretable
- RMSE is more sensitive to noise or outliers
- MSE/RMSE are smooth functions and easy to differentiate



```
[ ] import numpy as np
    import seaborn as sns
    import matplotlib.pyplot as plt
```

Definitions of MAE, MSE, RMSE

[ ] 1 cell hidden

Create random sample data

[ ] 1 cell hidden

Helper function for creating noisy data and plotting the charts

1 cell hidden

```
[ ] import numpy as np
    import seaborn as sns
    import matplotlib.pyplot as plt
```

Definitions of MAE, MSE, RMSE

[ ] 1 cell hidden

Create random sample data

[ ] 1 cell hidden

Helper function for creating noisy data and plotting the charts

1 cell hidden

+ Code + Text

Connect Editing

```
[ ] import numpy as np
    import seaborn as sns
    import matplotlib.pyplot as plt
```

Definitions of MAE, MSE, RMSE

[ ] 1 cell hidden

Create random sample data

[ ] 1 cell hidden

Helper function for creating noisy data and plotting the charts

1 cell hidden

Connected to
Python 3 Google Compute Engine backend

RAM: 0.81 GB/12.68 GB Disk: 22.54 GB/107.72 GB

```
[1] import numpy as np
    import seaborn as sns
    import matplotlib.pyplot as plt
```

Definitions of MAE, MSE, RMSE

1 cell hidden

Create random sample data

[ ] 1 cell hidden

Helper function for creating noisy data and plotting the charts

[ ] 1 cell hidden

```
[1] import numpy as np
    import seaborn as sns
    import matplotlib.pyplot as plt
```

Definitions of MAE, MSE, RMSE

```
[ ] def mae(data, ref):
        mae = 0
        c = 0
        for i in data:
          mae += abs(i-ref)
          c += 1
        return mae / c

    def mse(data, ref):
        mse = 0
        c = 0
        for i in data:
          mse += (i-ref)**2
          c += 1
        return mse / c

    def rmse(data, ref):
        return (mse(data, ref))**(1/2)
```

Create random sample data

```
[ ]     mae = 0
        c = 0
        for i in data:
          mae += abs(i-ref)
          c += 1
        return mae / c

    def mse(data, ref):
        mse = 0
        c = 0
        for i in data:
          mse += (i-ref)**2
          c += 1
        return mse / c

    def rmse(data, ref):
        return (mse(data, ref))**(1/2)
```

Create random sample data

```
[ ] num_sets = 1000
    num_points = 1000

    # Generates x sets with y data points each
    data = np.random.normal(100, 20, (num_sets, num_points))
```

Helper function for creating noisy data and plotting the charts

```
[2]         c += 1
        return mse / c

    def rmse(data, ref):
        return (mse(data, ref))**(1/2)
```

Create random sample data

```
[3] num_sets = 1000
    num_points = 1000

    # Generates x sets with y data points each
    data = np.random.normal(100, 20, (num_sets, num_points))
```

Helper function for creating noisy data and plotting the charts

```
[ ] def evaluate_metrics(data, num_outliers, amplitude_outliers):

        # Sample random "num_ouliers" points for each set
        noise = np.random.uniform(0, num_points, (num_sets, num_outliers))

        # Lists to keep the metrics for all original sets
        mae_original = []
        mse_original = []
        rmse_original = []
        # Lists to keep the metrics for all noisy sets
```

```
[2]         c += 1
        return mse / c

    def rmse(data, ref):
        return (mse(data, ref))**(1/2)
```

Create random sample data

```
[3] num_sets = 1000
    num_points = 1000

    # Generates x sets with y data points each
    data = np.random.normal(100, 20, (num_sets, num_points))
```

Helper function for creating noisy data and plotting the charts

```
[ ] def evaluate_metrics(data, num_outliers, amplitude_outliers):

        # Sample random "num_ouliers" points for each set
        noise = np.random.uniform(0, num_points, (num_sets, num_outliers))

        # Lists to keep the metrics for all original sets
        mae_original = []
        mse_original = []
        rmse_original = []
        # Lists to keep the metrics for all noisy sets
```

+ Code + Text

RAM Disk

Editing

Helper function for creating noisy data and plotting the charts

```
[ ] def evaluate_metrics(data, num_outliers, amplitude_outliers):

        # Sample random "num_ouliers" points for each set
        noise = np.random.uniform(0, num_points, (num_sets, num_outliers))

        # Lists to keep the metrics for all original sets
        mae_original = []
        mse_original = []
        rmse_original = []
        # Lists to keep the metrics for all noisy sets
        mae_noisy = []
        mse_noisy = []
        rmse_noisy = []

        # For each observation set, evaluates all three distances to the mean of the set
        for i in range(data.shape[0]):
            # Recovers the observations
            observations = data[i]
            # Create a noisy verion of the data with the randomly-chosen outliers
            outliers = [int(i) for i in noise[i]]
            observations_noisy = []
            c  = 0
            for x in observations:
                if c in outliers:
                    observations_noisy.append(x * amplitude_outliers)
                else:
                    observations_noisy.append(x)
```

+ Code + Text

RAM Disk

Editing

Helper function for creating noisy data and plotting the charts

```
[ ] def evaluate_metrics(data, num_outliers, amplitude_outliers):

        # Sample random "num_ouliers" points for each set
        noise = np.random.uniform(0, num_points, (num_sets, num_outliers))

        # Lists to keep the metrics for all original sets
        mae_original = []
        mse_original = []
        rmse_original = []
        # Lists to keep the metrics for all noisy sets
        mae_noisy = []
        mse_noisy = []
        rmse_noisy = []

        # For each observation set, evaluates all three distances to the mean of the set
        for i in range(data.shape[0]):
            # Recovers the observations
            observations = data[i]
            # Create a noisy verion of the data with the randomly-chosen outliers
            outliers = [int(i) for i in noise[i]]
            observations_noisy = []
            c  = 0
            for x in observations:
                if c in outliers:
                    observations_noisy.append(x * amplitude_outliers)
                else:
                    observations_noisy.append(x)
```

+ Code + Text

RAM Disk

Editing

Helper function for creating noisy data and plotting the charts

```
[ ] def evaluate_metrics(data, num_outliers, amplitude_outliers):

        # Sample random "num_ouliers" points for each set
        noise = np.random.uniform(0, num_points, (num_sets, num_outliers))

        # Lists to keep the metrics for all original sets
        mae_original = []
        mse_original = []
        rmse_original = []
        # Lists to keep the metrics for all noisy sets
        mae_noisy = []
        mse_noisy = []
        rmse_noisy = []

        # For each observation set, evaluates all three distances to the mean of the set
        for i in range(data.shape[0]):
            # Recovers the observations
            observations = data[i]
            # Create a noisy verion of the data with the randomly-chosen outliers
            outliers = [int(i) for i in noise[i]]
            observations_noisy = []
            c  = 0
            for x in observations:
                if c in outliers:
                    observations_noisy.append(x * amplitude_outliers)
                else:
                    observations_noisy.append(x)
```

Helper function for creating noisy data and plotting the charts

```
[ ] def evaluate_metrics(data, num_outliers, amplitude_outliers):

        # Sample random "num_ouliers" points for each set
        noise = np.random.uniform(0, num_points, (num_sets, num_outliers))

        # Lists to keep the metrics for all original sets
        mae_original = []
        mse_original = []
        rmse_original = []
        # Lists to keep the metrics for all noisy sets
        mae_noisy = []
        mse_noisy = []
        rmse_noisy = []

        # For each observation set, evaluates all three distances to the mean of the set
        for i in range(data.shape[0]):
            # Recovers the observations
            observations = data[i]
            # Create a noisy verion of the data with the randomly-chosen outliers
            outliers = [int(i) for i in noise[i]]
            observations_noisy = []
            c  = 0
            for x in observations:
                if c in outliers:
                    observations_noisy.append(x * amplitude_outliers)
                else:
                    observations_noisy.append(x)
```

```
[ ]
    mse_original = []
    rmse_original = []
    # Lists to keep the metrics for all noisy sets
    mae_noisy = []
    mse_noisy = []
    rmse_noisy = []

    # For each observation set, evaluates all three distances to the mean of the set
    for i in range(data.shape[0]):
        # Recovers the observations
        observations = data[i]
        # Create a noisy verion of the data with the randomly-chosen outliers
        outliers = [int(i) for i in noise[i]]
        observations_noisy = []
        c  = 0
        for x in observations:
            if c in outliers:
                observations_noisy.append(x * amplitude_outliers)
            else:
                observations_noisy.append(x)
            c += 1

        # Calculates the mean value of the set.
        # It is important to use the mean of the original set in both cases, to prevent it from shifting and altering the resu
        set_mean = np.mean(observations)

        # Evaluate the metrics
        mae_original.append(mae(observations, set_mean))
        mse_original.append(mse(observations, set_mean))
        rmse_original.append(rmse(observations, set_mean))
        mae_noisy.append(mae(observations_noisy, set_mean))
        mse_noisy.append(mse(observations_noisy, set_mean))
```

```
[ ]
mse_original = []
rmse_original = []
# Lists to keep the metrics for all noisy sets
mae_noisy = []
mse_noisy = []
rmse_noisy = []

# For each observation set, evaluates all three distances to the mean of the set
for i in range(data.shape[0]):
    # Recovers the observations
    observations = data[i]
    # Create a noisy verion of the data with the randomly-chosen outliers
    outliers = [int(i) for i in noise[i]]
    observations_noisy = []
    c  = 0
    for x in observations:
        if c in outliers:
            observations_noisy.append(x * amplitude_outliers)
        else:
            observations_noisy.append(x)
        c += 1

    # Calculates the mean value of the set.
    # It is important to use the mean of the original set in both cases, to prevent it from shifting and altering the resu
    set_mean = np.mean(observations)

    # Evaluate the metrics
    mae_original.append(mae(observations, set_mean))
    mse_original.append(mse(observations, set_mean))
    rmse_original.append(rmse(observations, set_mean))
    mae_noisy.append(mae(observations_noisy, set_mean))
    mse_noisy.append(mse(observations_noisy, set_mean))
```

```
[ ]
mse_original = []
rmse_original = []
# Lists to keep the metrics for all noisy sets
mae_noisy = []
mse_noisy = []
rmse_noisy = []

# For each observation set, evaluates all three distances to the mean of the set
for i in range(data.shape[0]):
    # Recovers the observations
    observations = data[i]
    # Create a noisy verion of the data with the randomly-chosen outliers
    outliers = [int(i) for i in noise[i]]
    observations_noisy = []
    c  = 0
    for x in observations:
        if c in outliers:
            observations_noisy.append(x * amplitude_outliers)
        else:
            observations_noisy.append(x)
        c += 1

    # Calculates the mean value of the set.
    # It is important to use the mean of the original set in both cases, to prevent it from shifting and altering the resu
    set_mean = np.mean(observations)

    # Evaluate the metrics
    mae_original.append(mae(observations, set_mean))
    mse_original.append(mse(observations, set_mean))
    rmse_original.append(rmse(observations, set_mean))
    mae_noisy.append(mae(observations_noisy, set_mean))
    mse_noisy.append(mse(observations_noisy, set_mean))
```

```
[ ]  mse_original = []
     rmse_original = []
     # Lists to keep the metrics for all noisy sets
     mae_noisy = []
     mse_noisy = []
     rmse_noisy = []

     # For each observation set, evaluates all three distances to the mean of the set
     for i in range(data.shape[0]):
         # Recovers the observations
         observations = data[i]
         # Create a noisy verion of the data with the randomly-chosen outliers
         outliers = [int(i) for i in noise[i]]
         observations_noisy = []
         c  = 0
         for x in observations:
             if c in outliers:
                 observations_noisy.append(x * amplitude_outliers)
             else:
                 observations_noisy.append(x)
             c += 1

         # Calculates the mean value of the set.
         # It is important to use the mean of the original set in both cases, to prevent it from shifting and altering the resu
         set_mean = np.mean(observations)

         # Evaluate the metrics
         mae_original.append(mae(observations, set_mean))
         mse_original.append(mse(observations, set_mean))
         rmse_original.append(rmse(observations, set_mean))
         mae_noisy.append(mae(observations_noisy, set_mean))
         mse_noisy.append(mse(observations_noisy, set_mean))
```

```
[ ]                observations_noisy.append(x)
            c += 1

        # Calculates the mean value of the set.
        # It is important to use the mean of the original set in both cases, to prevent it from shifting and altering the resu
        set_mean = np.mean(observations)

        # Evaluate the metrics
        mae_original.append(mae(observations, set_mean))
        mse_original.append(mse(observations, set_mean))
        rmse_original.append(rmse(observations, set_mean))
        mae_noisy.append(mae(observations_noisy, set_mean))
        mse_noisy.append(mse(observations_noisy, set_mean))
        rmse_noisy.append(rmse(observations_noisy, set_mean))

    # Plots all
    plt.figure(figsize=(12, 4))
    ax1 = plt.subplot(1, 3, 1)
    ax2 = plt.subplot(1, 3, 2)
    ax3 = plt.subplot(1, 3, 3)
    sns.histplot(mae_original, ax = ax1, color = 'maroon', kde = True, fill = False)
    sns.histplot(mae_noisy, ax = ax1, color = 'r', kde = True)
    sns.histplot(rmse_original, ax = ax2, color = 'darkgreen', kde = True, fill = False)
    sns.histplot(rmse_noisy, ax = ax2, color = 'g', kde = True)
    sns.histplot(mse_original, ax = ax3, color = 'darkblue', kde = True, fill = False)
    sns.histplot(mse_noisy, ax = ax3, color = 'b', kde = True)
    ax1.legend(["Original", "Noisy"], loc = 'lower right')
    ax2.legend(["Original", "Noisy"], loc = 'lower right')
    ax3.legend(["Original", "Noisy"], loc = 'lower right')
    ax1.set_title("MAE")
    ax2.set_title("RMSE")
```

+ Code + Text

RAM Disk

Editing

```
[ ]                observations_noisy.append(x)
            c += 1

        # Calculates the mean value of the set.
        # It is important to use the mean of the original set in both cases, to prevent it from shifting and altering the resu
        set_mean = np.mean(observations)

        # Evaluate the metrics
        mae_original.append(mae(observations, set_mean))
        mse_original.append(mse(observations, set_mean))
        rmse_original.append(rmse(observations, set_mean))
        mae_noisy.append(mae(observations_noisy, set_mean))
        mse_noisy.append(mse(observations_noisy, set_mean))
        rmse_noisy.append(rmse(observations_noisy, set_mean))

    # Plots all
    plt.figure(figsize=(12, 4))
    ax1 = plt.subplot(1, 3, 1)
    ax2 = plt.subplot(1, 3, 2)
    ax3 = plt.subplot(1, 3, 3)
    sns.histplot(mae_original, ax = ax1, color = 'maroon', kde = True, fill = False)
    sns.histplot(mae_noisy, ax = ax1, color = 'r', kde = True)
    sns.histplot(rmse_original, ax = ax2, color = 'darkgreen', kde = True, fill = False)
    sns.histplot(rmse_noisy, ax = ax2, color = 'g', kde = True)
    sns.histplot(mse_original, ax = ax3, color = 'darkblue', kde = True, fill = False)
    sns.histplot(mse_noisy, ax = ax3, color = 'b', kde = True)
    ax1.legend(["Original", "Noisy"], loc = 'lower right')
    ax2.legend(["Original", "Noisy"], loc = 'lower right')
    ax3.legend(["Original", "Noisy"], loc = 'lower right')
    ax1.set_title("MAE")
    ax2.set_title("RMSE")
```

```
[ ]         # Evaluate the metrics
            mae_original.append(mae(observations, set_mean))
            mse_original.append(mse(observations, set_mean))
            rmse_original.append(rmse(observations, set_mean))
            mae_noisy.append(mae(observations_noisy, set_mean))
            mse_noisy.append(mse(observations_noisy, set_mean))
            rmse_noisy.append(rmse(observations_noisy, set_mean))

        # Plots all
        plt.figure(figsize=(12, 4))
        ax1 = plt.subplot(1, 3, 1)
        ax2 = plt.subplot(1, 3, 2)
        ax3 = plt.subplot(1, 3, 3)
        sns.histplot(mae_original, ax = ax1, color = 'maroon', kde = True, fill = False)
        sns.histplot(mae_noisy, ax = ax1, color = 'r', kde = True)
        sns.histplot(rmse_original, ax = ax2, color = 'darkgreen', kde = True, fill = False)
        sns.histplot(rmse_noisy, ax = ax2, color = 'g', kde = True)
        sns.histplot(mse_original, ax = ax3, color = 'darkblue', kde = True, fill = False)
        sns.histplot(mse_noisy, ax = ax3, color = 'b', kde = True)
        ax1.legend(["Original", "Noisy"], loc = 'lower right')
        ax2.legend(["Original", "Noisy"], loc = 'lower right')
        ax3.legend(["Original", "Noisy"], loc = 'lower right')
        ax1.set_title("MAE")
        ax2.set_title("RMSE")
        ax3.set_title("MSE")
        plt.tight_layout()
```

+ Code + Text

RAM Disk

Editing

```
[4]  ax3.legend(["Original", "Noisy"], loc = "lower right")
     ax1.set_title("MAE")
     ax2.set_title("RMSE")
     ax3.set_title("MSE")
     plt.tight_layout()
```

evaluate

+ Code + Text

RAM Disk

Editing

```
[4]     ax3.legend(["Original", "Noisy"], loc = "lower right")
        ax1.set_title("MAE")
        ax2.set_title("RMSE")
        ax3.set_title("MSE")
        plt.tight_layout()
```

```
evaluate_metrics(data, num_outliers=0)
```

s here

+ Code + Text

RAM Disk

Editing

```
[4]     ax3.legend(["Original", "Noisy"], loc = "lower right")
        ax1.set_title("MAE")
        ax2.set_title("RMSE")
        ax3.set_title("MSE")
        plt.tight_layout()
```

```
evaluate_metrics(data, num_outliers=0, amplitude_outliers=0)
```

+ Code + Text

RAM Disk

Editing

```
[4]     ax3.legend(["Original", "Noisy"], loc = "lower right")
        ax1.set_title("MAE")
        ax2.set_title("RMSE")
        ax3.set_title("MSE")
        plt.tight_layout()
```

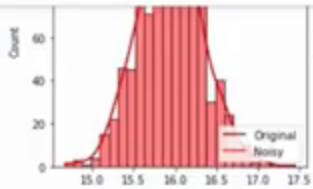
```
evaluate_metrics(data, num_outliers=0, amplitude_outliers=1)
```

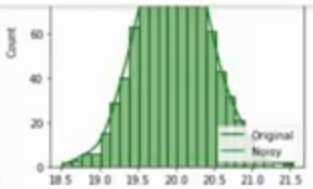
+ Code + Text

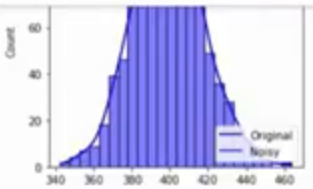
RAM Disk

Editing

[5]







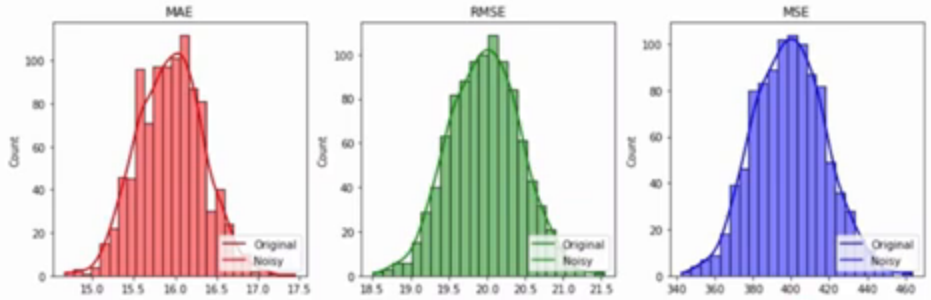
+ Code + Text

RAM Disk

Editing

```
[4]     ax3.set_title("MSE")
        plt.tight_layout()
```

```
[5] evaluate_metrics(data, num_outliers=0, amplitude_outliers=1)
```



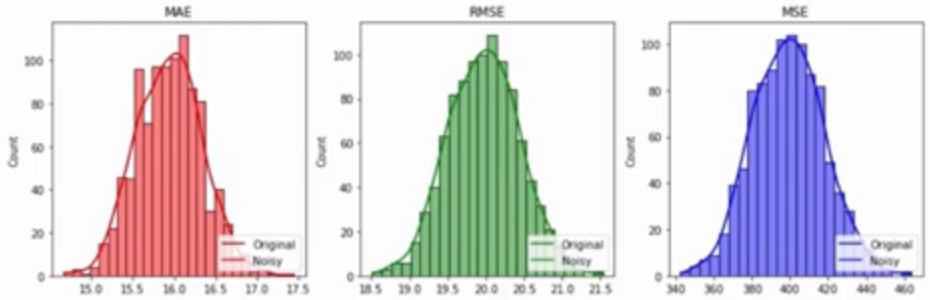
+ Code + Text

RAM Disk

Editing

```
[4]     ax3.set_title("MSE")
        plt.tight_layout()
```

```
[5] evaluate_metrics(data, num_outliers=0, amplitude_outliers=1)
```



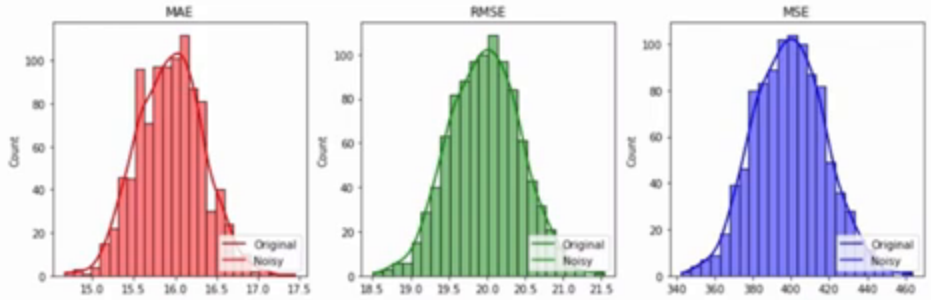
+ Code + Text

RAM Disk

Editing

```
[4]     ax3.set_title("MSE")
        plt.tight_layout()
```

```
[5] evaluate_metrics(data, num_outliers=0, amplitude_outliers=1)
```



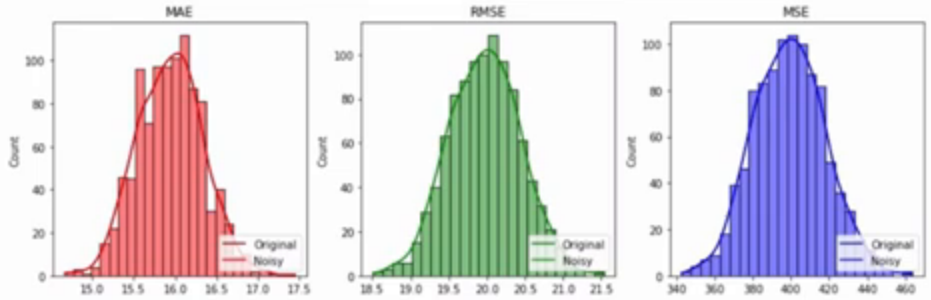
+ Code + Text

RAM Disk

Editing

```
[4]     ax3.set_title("MSE")
        plt.tight_layout()
```

```
[5] evaluate_metrics(data, num_outliers=0, amplitude_outliers=1)
```



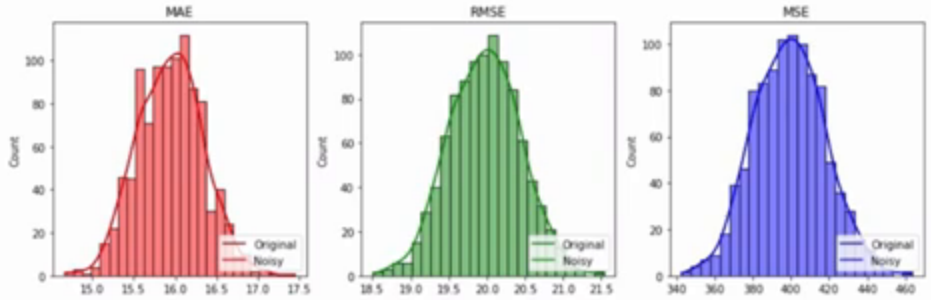
+ Code + Text

RAM Disk

Editing

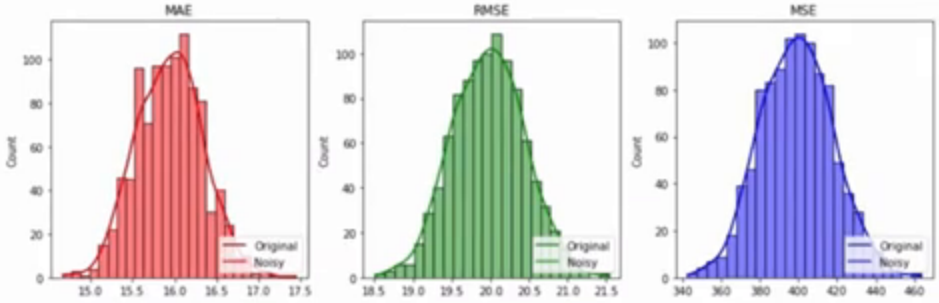
```
[4]     ax3.set_title("MSE")
        plt.tight_layout()
```

```
[5] evaluate_metrics(data, num_outliers=0, amplitude_outliers=1)
```



```
[4]  ax3.set_title("MSE")
     plt.tight_layout()
```

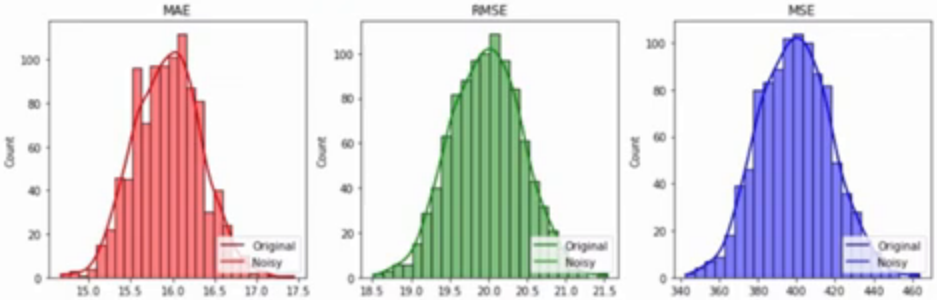
```
[5]  evaluate_metrics(data, num_outliers=0, amplitude_outliers=1)
```




```
evaluate_metrics(data, num_outliers=0, amplitude_outliers=1)
```

```
    ax3.set_title("MSE")
    plt.tight_layout()
```

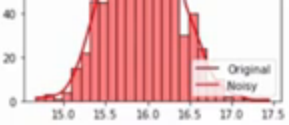
```
[5] evaluate_metrics(data, num_outliers=0, amplitude_outliers=1)
```

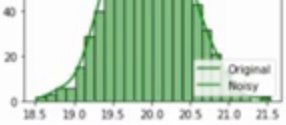


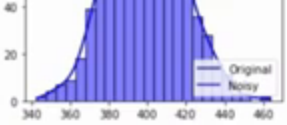

```
evaluate_metrics(data, num_outliers=2, amplitude_outliers=2)
```

4s completed at 10:49 AM

[5]



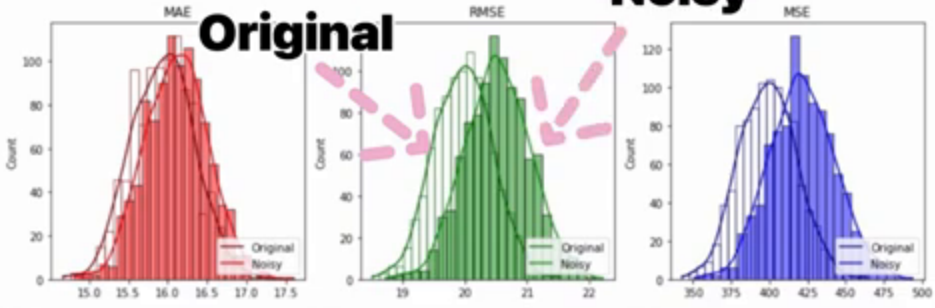





```
evaluate_metrics(data, num_outliers=2, amplitude_outliers=2)
```

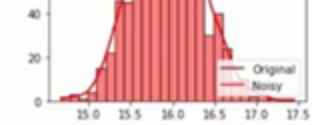
Original

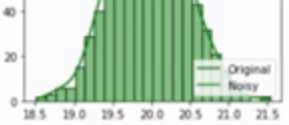
Noisy

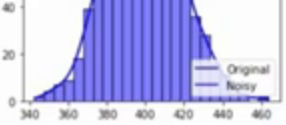


[ ]

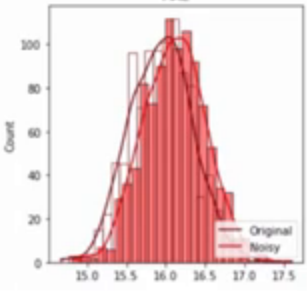
4s completed at 10:53 AM

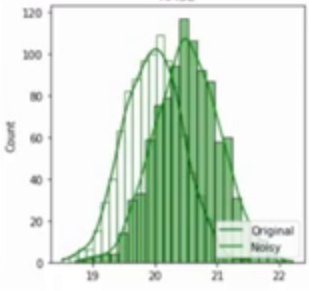


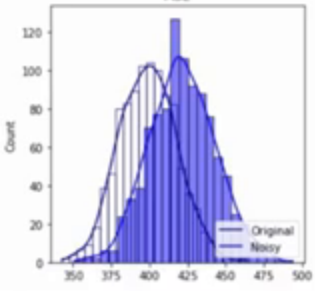




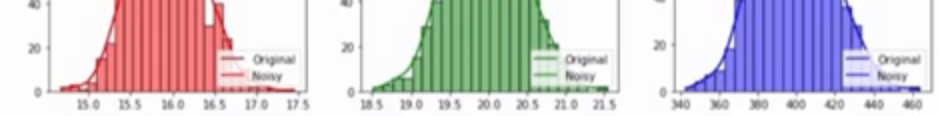

```
evaluate_metrics(data, num_outliers=2, amplitude_outliers=2)
```



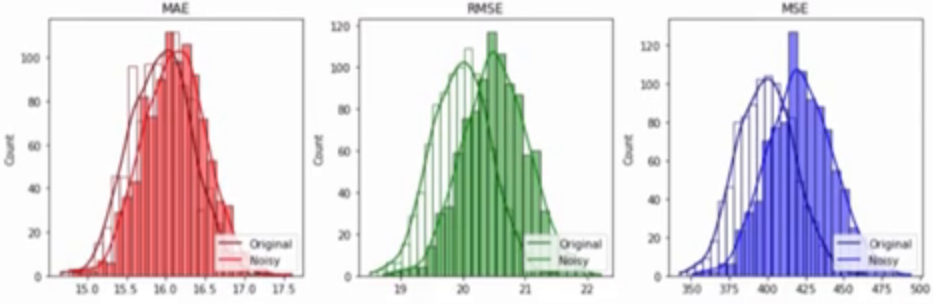





```
[ ]
```

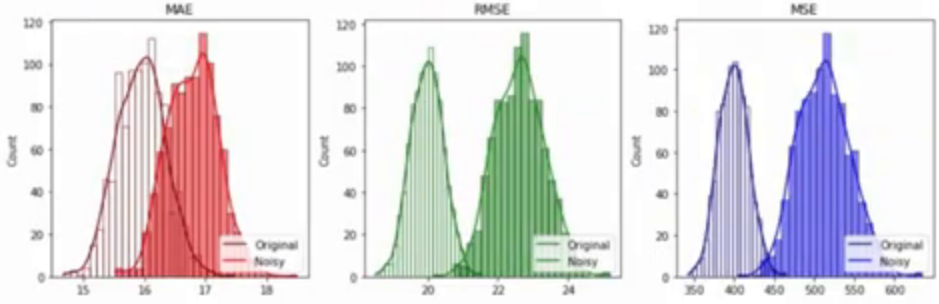



```
evaluate_metrics(data, num_outliers=2, amplitude_outliers=2)
```

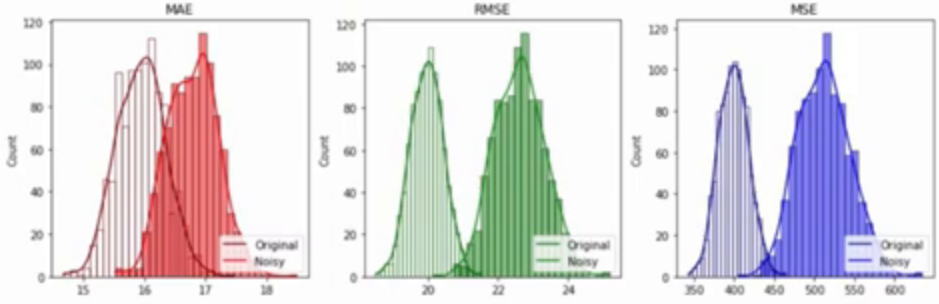



```
[ ]
```

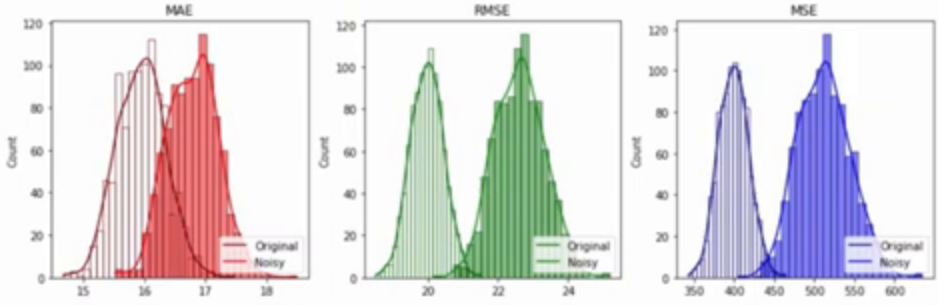
```
[7] evaluate_metrics(data, num_outliers=10, amplitude_outliers=2)
```



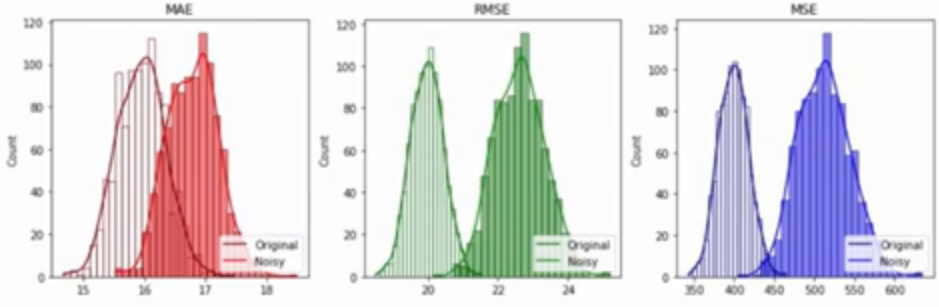
```
[7] evaluate_metrics(data, num_outliers=10, amplitude_outliers=2)
```



```
[7] evaluate_metrics(data, num_outliers=10, amplitude_outliers=2)
```

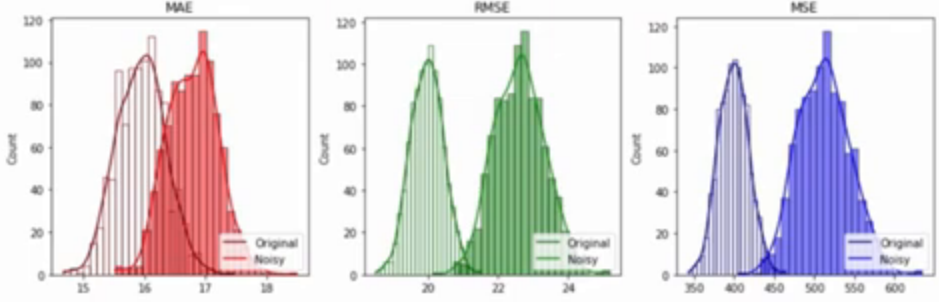


```
evaluate_metrics(data, num_outliers=10, amplitude_outliers=2)
```

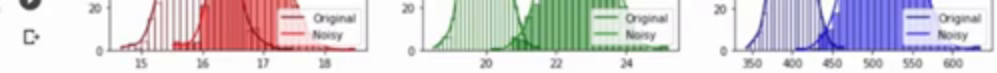



```
[ ]
```

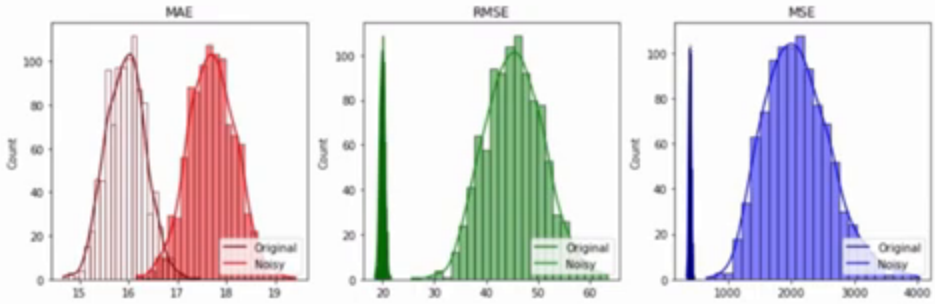
```
evaluate_metrics(data, num_outliers=10, amplitude_outliers=2)
```

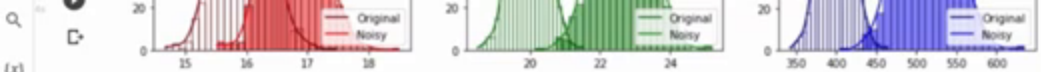



```
[ ] evaluate_metrics(data, num_outliers=0, amplitude_outliers=10)
```

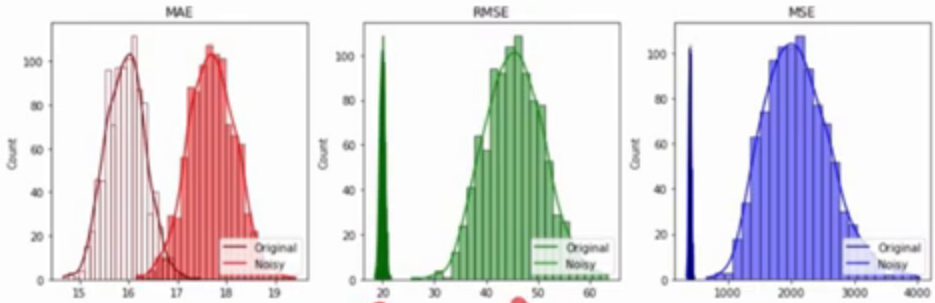


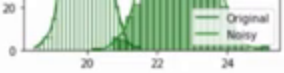

```
[8] evaluate_metrics(data, num_outliers=2, amplitude_outliers=10)
```

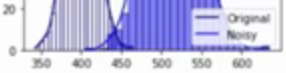




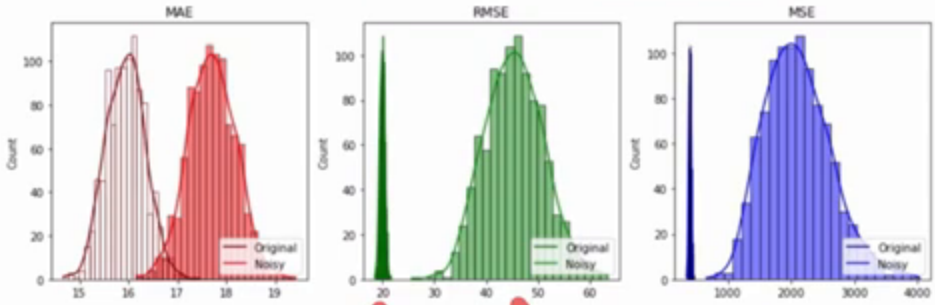

```
[8]  evaluate_metrics(data, num_outliers=2, amplitude_outliers=10)
```



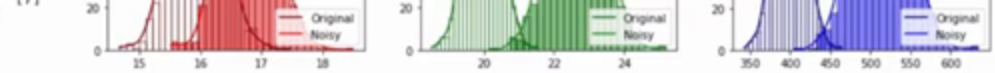




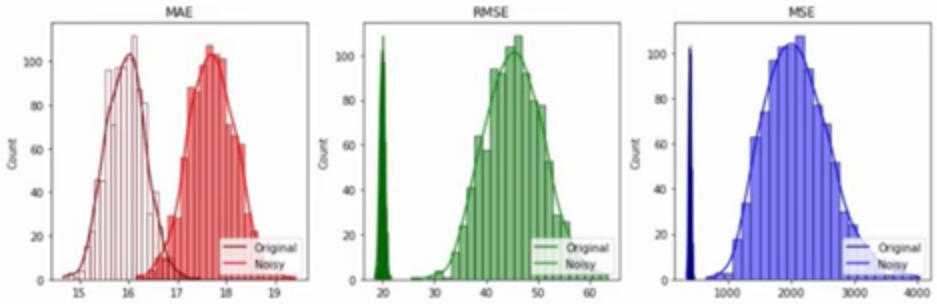

```
evaluate_metrics(data, num_outliers=2, amplitude_outliers=10)
```



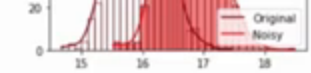

```
[ ]
```

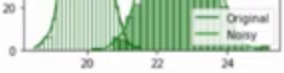


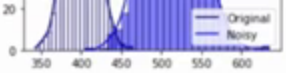

```
evaluate_metrics(data, num_outliers=2, amplitude_outliers=10)
```



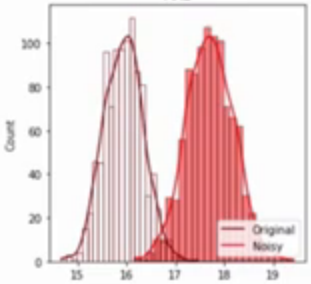

```
[ ]
```

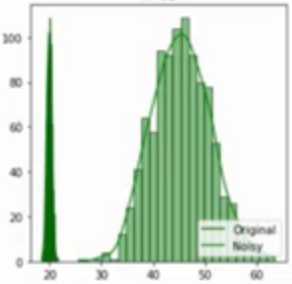


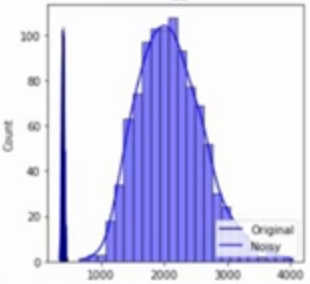




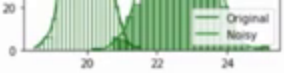

```
evaluate_metrics(data, num_outliers=2, amplitude_outliers=10)
```

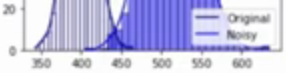




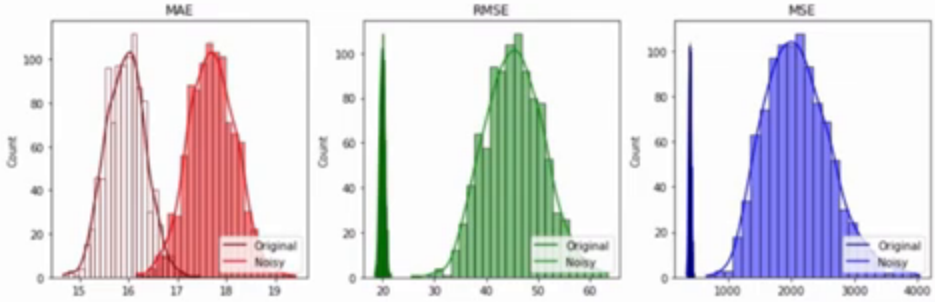



```
[ ]
```

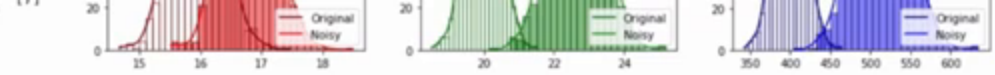




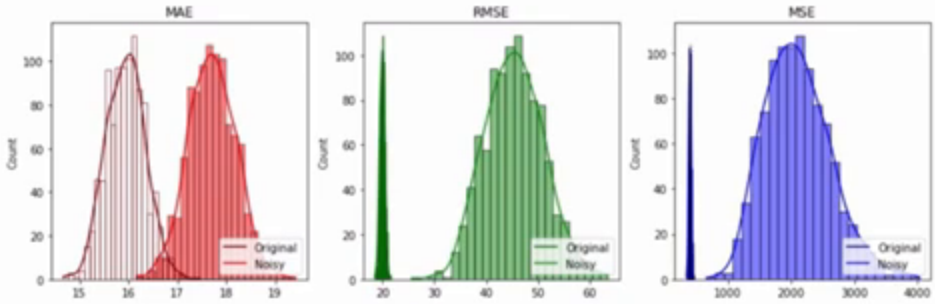

```
evaluate_metrics(data, num_outliers=2, amplitude_outliers=10)
```



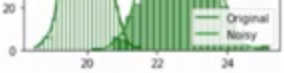

```
[ ]
```

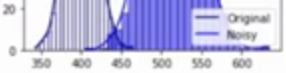



```
evaluate_metrics(data, num_outliers=2, amplitude_outliers=10)
```

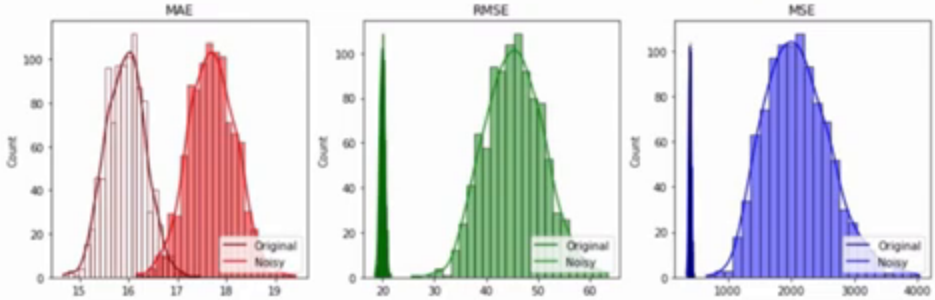



```
[ ]
```






```
evaluate_metrics(data, num_outliers=2, amplitude_outliers=10)
```




```
[ ]
```