For understanding the Error Evaluation & R Squared (model comparison) deeper, that will help me find a true alpha mathematically wise & not by only intuition & brute force, I will create the python algo here by hand.

In the exercise on harvardx we compared the TV Budget (x) VS the Sales of that product (y).

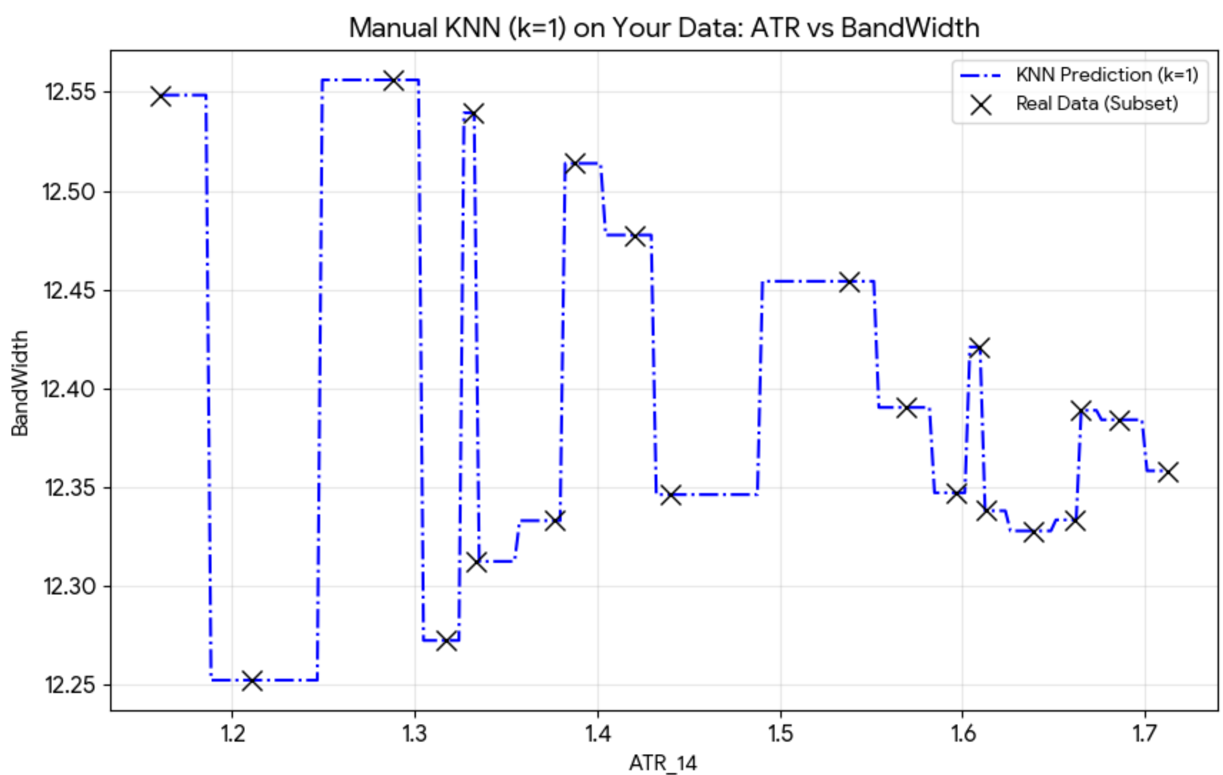
But here I will use the algorithm called WickSurfer 006, that use a polynomial regression channel to create a bandwidth that is trapping the price and wait for a pullback:

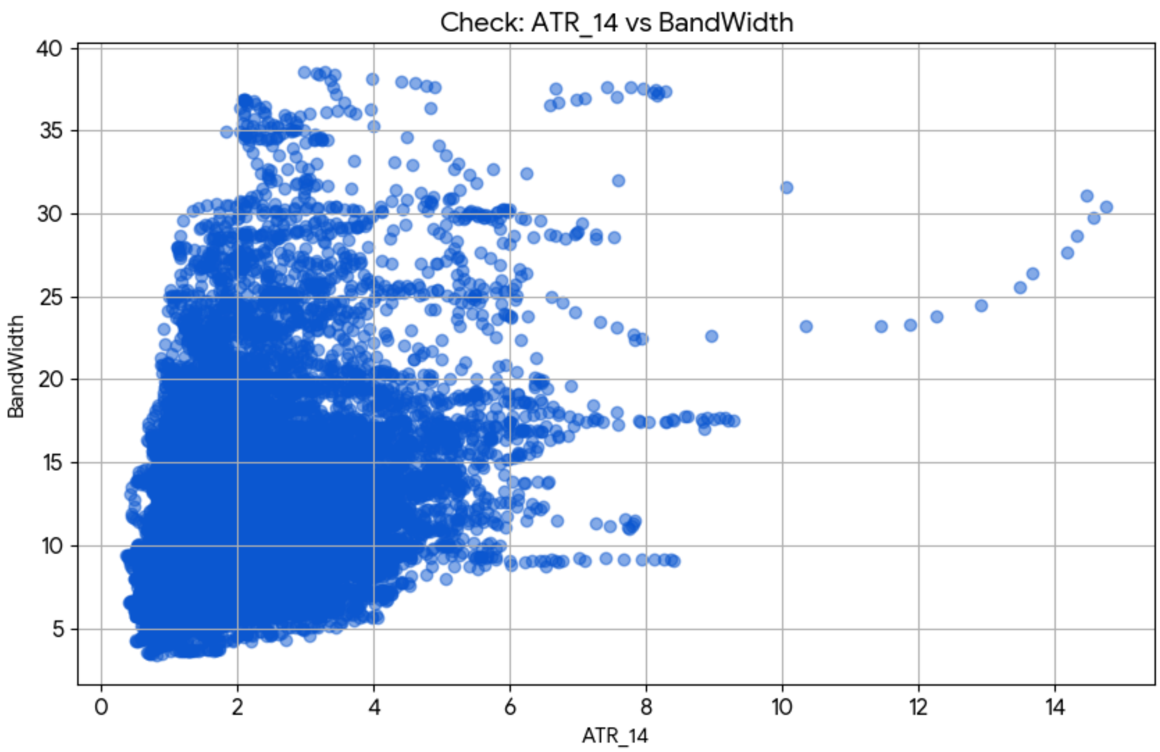
We will compare the market volatility using a Volatility measure called ATR.

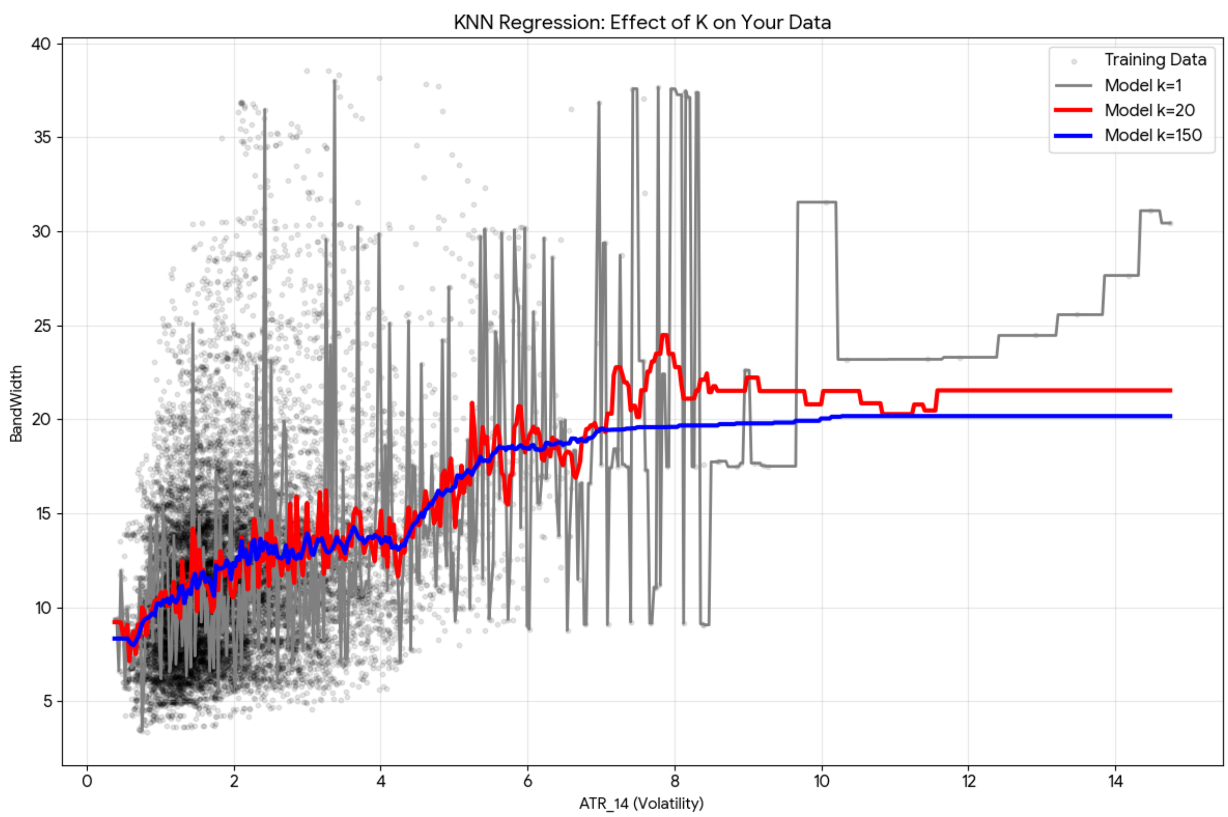
VS

The polynomial Bandwidth Distance between the Yellow lines

I used a CSV Dataset between 2024 - 2026 & used 20 random real market data as a research example:




Check: ATR_14 vs BandWidth
BandWidth
ATR_14
40
35
30
25
20
15
10
5
0
2
4
6
8
10
12
14


KNN Regression: Effect of K on Your Data
Training Data
Model k=1
Model k=20
Model k=150
BandWidth
ATR_14 (Volatility)
40
35
30
25
20
15
10
5
0
2
4
6
8
10
12
14

ATR_14 (X - Predictor)	BandWidth (y - Response)
0.8128	11.2319
0.8236	11.1215
0.8451	10.9822
0.8614	11.0541
0.8829	10.8763
0.9012	11.452
0.9255	11.6781
0.9411	11.5542
0.9632	11.8903
0.9854	12.0124
1.0125	12.4531
1.0341	12.3325
1.0528	14.4596
1.1028	14.7715
1.14	14.293
1.1514	14.5817
1.1928	14.0999
1.2541	15.6782
1.3211	16.1234
1.4522	17.8901

Step 1

We analyze how market Volatility ATR dictates the algorithmic price channel width (bandwidth).

By using kNN regression, we find the optimal balance between

* reacting to market changes
* Ignoring random noise

```
# STEP 1: Library Imports and Data Initialization
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split

# Load your specific algorithm data
# Filename from your research: WickSurfer_Data_XAUUSD_2023-2024.r.csv
df = pd.read_csv('WickSurfer_Data_XAUUSD_2023-2024.r.csv')

# Clean column names to remove any leading/trailing spaces
df.columns = df.columns.str.strip()

# --- Feature Selection (The "TV vs Sales" equivalent) ---
# Predictor (x): ATR_14 (Market Volatility)
# Response (y): BandWidth (Polynomial Channel Width)

# As mentioned in your research image: "used 20 random real market data"
# We take a small subset for the manual part of the research
df_subset = df.iloc[100:120].copy()

x_true = df_subset['ATR_14'].values
y_true = df_subset['BandWidth'].values

# --- Data Sorting ---
# We sort the data by X values to ensure the plots look like a continuous
# line rather than a "spider web" of connected points.
idx = np.argsort(x_true)
x_true = x_true[idx]
y_true = y_true[idx]

# Preview the data to confirm we are ready
print("X (ATR_14) Values:", x_true[:5])
print("Y (BandWidth) Values:", y_true[:5])
```

1. We import pandas for data handling, numpy for mathematical operations and matplotlib for visuals (graphs) and sklearn to later compare our kNN (K) results & for model selection, test data split)

2. Import the dataset CSV

3. ATR (market volatility) will be the x predictor. Bandwidth of the polynomial channel will be the y response

4. We choose 20 random market data between 2023 - 2024

5. We sort the data so that the kNN graph will make sense

```python
# STEP 2: Manual KNN Logic (The "find_nearest" Engine)

# 1. Define the function that finds the index of the nearest neighbor
def find_nearest(array, value):
    """
    This function calculates the absolute distance between a 'new' point
    and all known historical points in our data.
    """
    # Calculate absolute differences and find the index of the minimum value
    idx = pd.Series(np.abs(array - value)).idxmin()

    # Return the index and the actual value from the array
    return idx, array[idx]

# 2. Create synthetic X-values for smooth plotting
# We create 200 points between the min and max ATR of your subset
x_synth = np.linspace(np.min(x_true), np.max(x_true), 200)

# 3. Initialize the Y-values (predictions) to zero
y_synth = np.zeros(len(x_synth))

# 4. Run the manual KNN (k=1) prediction loop
# For every synthetic ATR value, find the BandWidth of its closest neighbor
for i, xi in enumerate(x_synth):
    # We take the Sales (BandWidth) value from the closest index found
    closest_idx, closest_val = find_nearest(x_true, xi)
    y_synth[i] = y_true[closest_idx]

# 5. Visualization of the manual research (Subset)
plt.figure(figsize=(10, 6))
plt.plot(x_synth, y_synth, '-.', label='KNN Prediction (k=1)', color='blue')
plt.plot(x_true, y_true, 'kx', markersize=10, label='Real Market Data (Subset)')

plt.title('Manual KNN (k=1) on WickSurfer Data: ATR vs BandWidth')
plt.xlabel('ATR_14 (Volatility)')
plt.ylabel('BandWidth (Channel Width)')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

Step 2

Now we are creating the heart of the algorithm -> The kNN :

Taking a k number of data neighbors in the graph into account :

① The find nearest function :

* Takes the data array & a new point
* Subtract one from the other.
* Uses Idxmin() func to the neigbor that is the closest to it.

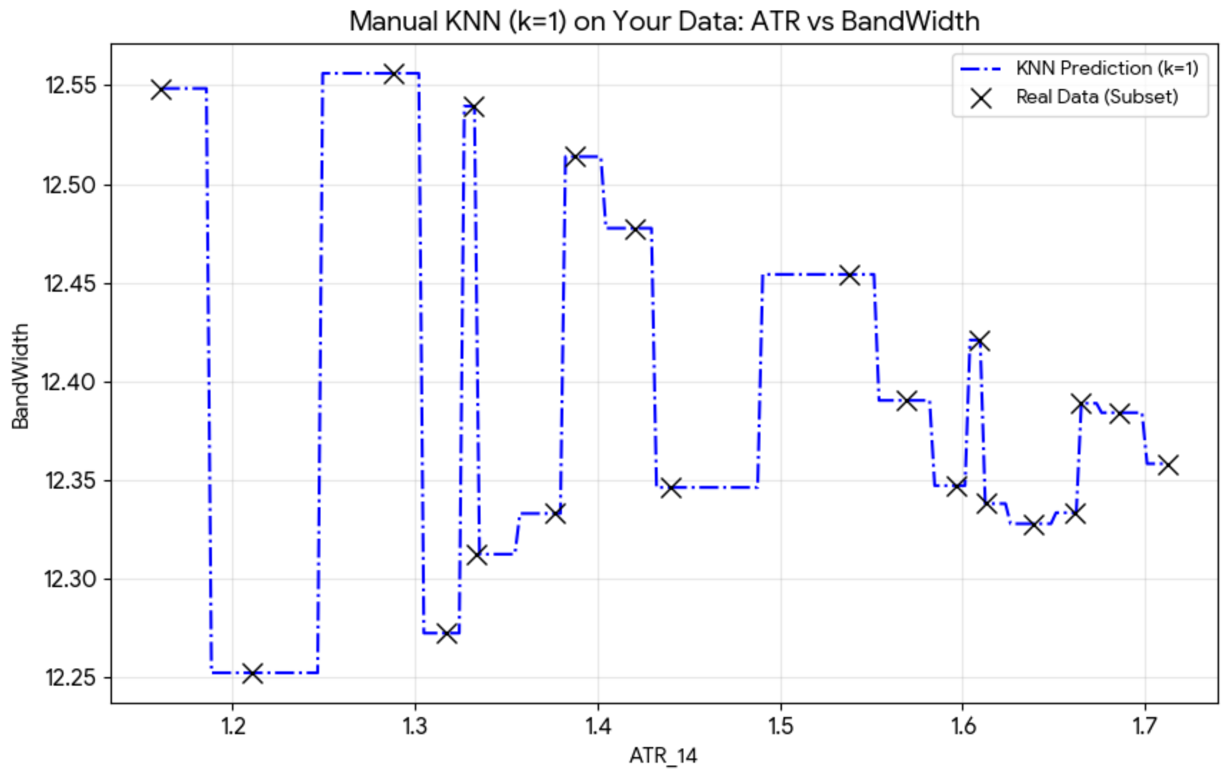
② In real market ATR don't move in a perfect line

So for using that volatility measurement we :

* We use 200 equal points on the ATR (x) in the graph.
* We use the minimal value of the ATR from the real data we randomly chose as the first value and the max valus as the last.

* the 200 points is between them

* When we run the model we basically ask him 200 times :
What would you if the ATR will be 1 , 1.01 , 1.13 ... ? 200 times.



③ **How the steps above work? :**

1. The loop is getting from the "x_synth" that the current ATR is 1.15
2. The find_nearest func check which of the following real data is the closest to it

3. the model understand that the data when the ATR was 1.10 is the closest (example)

4. The model will continue to give the same bandwidth result until there will be a closer value nearby.

Step 3

* Now we use All the real market data we have
* using a % of the data as a training data & a % as testing data

```python
# STEP 3: Professional KNN Implementation with Scikit-Learn

# 1. Prepare the FULL dataset (Cleaning and Reshaping)
# We drop any missing values to ensure the model runs smoothly
df_clean = df[['ATR_14', 'BandWidth']].dropna()

# Important: Scikit-learn requires X to be a 2D matrix (reshape)
X = df_clean[['ATR_14']].values
y = df_clean['BandWidth'].values

# 2. Split the dataset into Training (60%) and Testing (40%)
# This allows us to train the model on one set and validate it on another
x_train, x_test, y_train, y_test = train_test_split(X, y, train_size=0.6,
random_state=42)

# 3. Create a smooth grid for plotting predictions
# We generate 500 points covering the full range of ATR in the dataset
x_grid = np.linspace(X.min(), X.max(), 500).reshape(-1, 1)

# 4. Initialize the plot
fig, ax = plt.subplots(figsize=(12, 8))

# Plot the Training Data in the background (faded black dots)
ax.scatter(x_train, y_train, color='black', alpha=0.1, s=10, label='Training Data')

# 5. Loop over different K values to compare model complexity
# We use the values from your research: 1 (Noisy), 20 (Optimal), 150 (Flat)
k_values = [1, 20, 150]
colors = ['grey', 'red', 'blue']
linewidths = [1, 3, 3]

for i, k in enumerate(k_values):
    # Initialize the Scikit-learn KNN Regressor
    model = KNeighborsRegressor(n_neighbors=k)

    # Fit (Train) the model on the training set
    model.fit(x_train, y_train)

    # Predict over our smooth grid for visualization
    y_pred_grid = model.predict(x_grid)

    # Plot the regression line
    ax.plot(x_grid, y_pred_grid, color=colors[i],
            linewidth=linewidths[i], label=f'Model k={k}')

# 6. Set titles and labels
ax.set_title('Professional KNN: Comparing K-Values on WickSurfer006 Data',
fontsize=16)
ax.set_xlabel('ATR_14 (Market Volatility)', fontsize=14)
ax.set_ylabel('BandWidth (Channel Width)', fontsize=14)
ax.legend(fontsize=12)
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
-
```

1. Data Reshaping : Unlike our manual function Scikit-learn is expecting the predictor X to be A 2 Dimensional Array.

A matrix :

So we need to use df[['ATR_14']] or .reshape(-1, 1) to make the data fit the requirement while being 1D.

2. Splitting the Data to train & test data by X (We reserve 40% of the data for testing). That is how we can be sure that our strategy is not overfiting/under fitting & and still profit in real tick X wise.

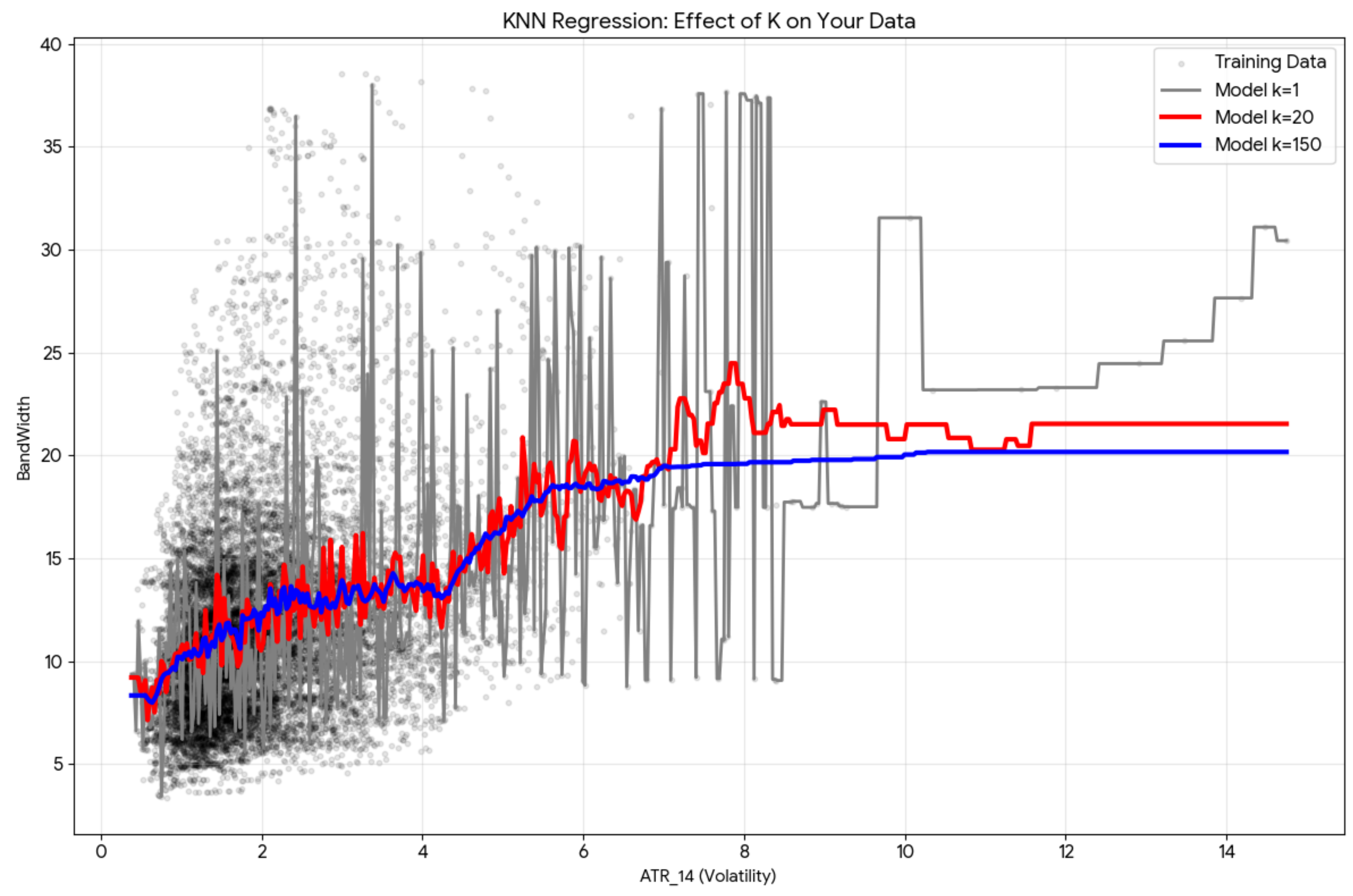
3. THE HYPERPARAMETER K:

THIS IS WHERE WE SEE THE BIAS VARIANCE TRADE OFF!

k = 1 : Jagged & overfitting (grey line)

k = 150 : Extremely underfitting, (blue line)

k = 20 : The sweet spot of the algorithm



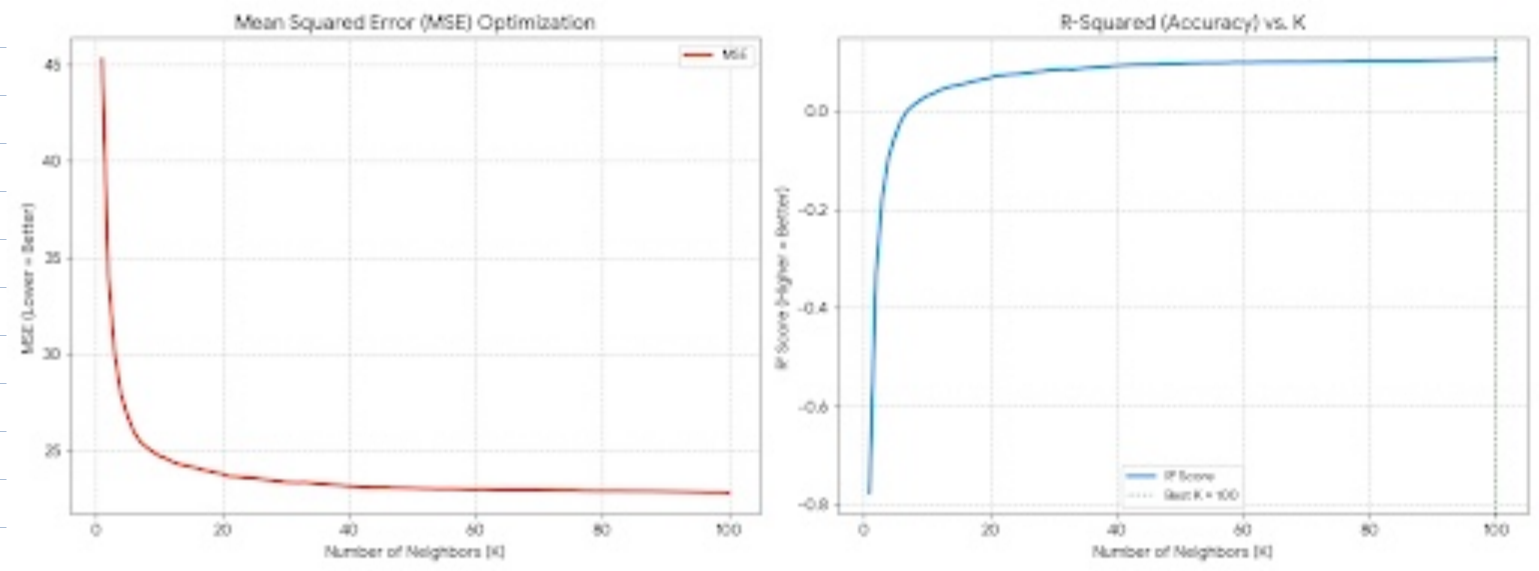
Step 4

Now we use Error Evaluation & R square
to full proof the sweet-spot K
R^2 & Mean Square Error

```
# STEP 4: Quantitative Evaluation - MSE, R-Squared, and Optimization

from sklearn.metrics import mean_squared_error, r2_score

# 1. Initialize lists to store our results
k_range = range(1, 101) # Testing K values from 1 to 100
mse_list = []
r2_list = []

# 2. Optimization Loop
# We iterate through each K to find where the error is smallest
for k in k_range:
    model = KNeighborsRegressor(n_neighbors=k)
    model.fit(x_train, y_train)

    # Predict on the TEST set (data the model hasn't seen)
    y_pred = model.predict(x_test)

    # Calculate Metrics
    mse_list.append(mean_squared_error(y_test, y_pred))
    r2_list.append(r2_score(y_test, y_pred))

# 3. Finding the Mathematically Optimal K
best_k_r2 = k_range[np.argmax(r2_list)]
best_r2_score = max(r2_list)

print(f"Research Result: The optimal K value is {best_k_r2}")
print(f"R-Squared Score at best K: {best_r2_score:.4f}")

# 4. Visualization of Error Evaluation
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

# Plot 1: Mean Squared Error (Lower is better)
ax1.plot(k_range, mse_list, color='red', linewidth=2)
ax1.set_title('Mean Squared Error (MSE) vs. K', fontsize=14)
ax1.set_xlabel('Number of Neighbors (K)')
ax1.set_ylabel('MSE (Error)')
ax1.grid(True, alpha=0.3)

# Plot 2: R-Squared Score (Higher is better)
ax2.plot(k_range, r2_list, color='blue', linewidth=2)
ax2.axvline(best_k_r2, color='green', linestyle='--', label=f'Best K =
{best_k_r2}')
ax2.set_title('R-Squared Score (Accuracy) vs. K', fontsize=14)
ax2.set_xlabel('Number of Neighbors (K)')
ax2.set_ylabel('R2 Score')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

(1) Error Metrics :

* MSE : Represent the "penalty" for wrong predictions and we want to minimize that result as much as we can.



* R^2 square : Tells us what % of the movement that can be explained by the ATR or in other words : The more the R^2 is closer to 1 $\rightarrow$ The stronger the relationship between the bandwidth & the ATR

Research Findings

The optimal κ & R^2 analysis :

1. The optimal hyperparameter (κ) : 100 (The highest Value in our tested range)

2. The R^2 the % that can be explained only by the ATR : 0.105 → 10.5%

In the financial world this is a significant finding! There is a LOT of noise in the capital market & Especially on XAUUSD

And the fact that the ATR <-> Bandwidth can explain 10.5 % of it showing that there is a statistical proof between the two.

That means that widenning the average gives us more reliable bandwidth that will less likely be breaked by a false breakout / mean reversion.