In [1]:
```
# Project 3: Predict BAC Stock Price Using Machine Learning and Macroeco

# 1) Build ML models to predict BAC's next day price using historical st
# peer financial stocks (JPM, MS, C, WFC), and key macroeconomic var (VI
# Oil Prices, and Gold Price)

# 2) Perform feature engineering

# 3) Apply different ML algorithms such as Decision Tree, Random Forest,
# Neighbors (KNN) models

# 4) Evaluate the model based on R square, rmse, mse, mae and other metr
```

In [2]:
```
# Import neccessary data science packages
import numpy as np
import pandas as pd
import yfinance as yf
```

In [3]:
```
# 'BAC' => Bank of America Corp (peer finance stock)
# 'JPM' => JP Morgan Chase & Co. (peer finance stock)
# 'MS' => Morgan Stanley (peer finance stock)
# 'C' => Citi Group (peer finance stock)
# 'WFC' => Wells Fargo & Co (peer finance stock)
# 'SPY' => S&P500 ETF (Market Index)
# '^VIX' => CBOE Volatility Index (market fear indicator)
# '^TNX' => 10Y US Treasury Yield (Interest Rate Indicator)
# 'DX-Y.NYB' => US Dollar Index (Strength of US Dollar)
# 'CL=F' => # Crude Oil Future (Inflation/energy proxy)
# 'GC=F' => Gold Futures (safe asset)
```

In [4]:
```
# Download the data from yahoo finance

tickers = ['BAC','JPM', 'MS', 'C', 'WFC', 'SPY', '^VIX', '^TNX', 'DX-Y.N

data = yf.download(tickers, start = '2002-01-01', end = '2025-01-01')['C
data
```

```
YF.download() has changed argument auto_adjust default to True

[**********************100%************************]  11 of 11 completed
```

Out[4]:

Ticker	BAC	C	CL=F	DX-Y.NYB	GC=F	JPM	MS	
Date								
2002-01-02	17.843855	276.156769	21.010000	115.790001	278.899994	18.587128	27.954092	75.13
2002-01-03	17.824017	276.373352	20.370001	116.110001	278.200012	19.083126	28.764050	75.98
2002-01-04	18.101761	281.516418	21.620001	116.330002	278.899994	19.942179	30.191624	76.49
2002-01-07	17.886364	278.214081	21.480000	116.330002	278.600006	19.891054	29.963816	75.95
2002-01-08	17.600115	267.981903	21.250000	116.830002	278.899994	19.737631	29.336084	75.77
...	...	...	...	...	...	...	...	
2024-12-24	44.101688	69.952347	70.099998	108.260002	2620.000000	239.589233	125.215462	599.49
2024-12-26	44.270622	70.297180	69.620003	108.129997	2638.800049	240.409912	126.171379	599.53
2024-12-27	44.061939	69.952347	70.599998	108.000000	2617.199951	238.462036	124.919815	593.22
2024-12-30	43.634636	69.351349	70.989998	108.129997	2606.100098	236.632812	123.924477	586.45
2024-12-31	43.674385	69.351349	71.720001	108.489998	2629.199951	237.018433	123.894913	584.32

5810 rows × 11 columns

In [5]:
```
# Information about your data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 5810 entries, 2002-01-02 to 2024-12-31
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   BAC       5789 non-null   float64
 1   C         5789 non-null   float64
 2   CL=F      5779 non-null   float64
 3   DX-Y.NYB  5807 non-null   float64
 4   GC=F      5775 non-null   float64
 5   JPM       5789 non-null   float64
 6   MS        5789 non-null   float64
 7   SPY       5789 non-null   float64
 8   WFC       5789 non-null   float64
 9   ^TNX      5783 non-null   float64
 10  ^VIX      5789 non-null   float64
dtypes: float64(11)
memory usage: 544.7 KB
```

In [6]:
```
# Statistical Summary of Data
data.describe()
```

Out[6]:

Ticker	BAC	C	CL=F	DX-Y.NYB	GC=F	JPM	M
count	5789.000000	5789.000000	5779.000000	5807.000000	5775.000000	5789.000000	5789.00000
mean	22.095994	110.073951	66.811741	90.834779	1212.050857	60.711951	37.52045
std	10.042737	110.526008	24.071656	10.092432	555.276716	49.422008	23.83538
min	2.470939	7.663626	-37.630001	71.330002	278.100006	8.205750	6.46061
25%	12.736433	36.660908	48.520000	81.879997	732.649994	25.016428	21.58594
50%	22.863873	50.757572	66.230003	90.779999	1253.000000	37.881187	29.30907
75%	29.551432	197.121552	84.514999	97.825001	1648.750000	89.981766	42.48669
max	47.207829	383.228149	145.289993	120.239998	2788.500000	247.479630	133.03035

In [7]:
```
# How many missing values are present in my dataset
data.isnull().sum()
```

Out[7]:
```
Ticker
BAC          21
C            21
CL=F         31
DX-Y.NYB      3
GC=F         35
JPM          21
MS           21
SPY          21
WFC          21
^TNX         27
^VIX         21
dtype: int64
```

In [8]:
```
# What can we do with missing data?
# 1) Drop the values
# 2) Forward Fill => Carries forward the last known value
# 3) Backward Fill => Filling the value backward
# 4) Average of the particular stock
# 5) Interpolation => Linear Interpolation or Cubic Spline Interpolation
```

In [9]:
```
# Eg: Forward Fill

#df = [100, np.nan, np.nan, 200]
# df = df.fillna(method = 'ffill') or df = df.ffill()
# df = [100,100,100,200]
```

In [10]:
```
# Eg: Backward Fill

#df = [100, np.nan, np.nan, 200]
# df = [100,200,200,200]
```

In [11]:
```
# Average
#df = [100, np.nan, np.nan, 200]
#df = [100, 150, 150, 200]

#(200+100)/2 = 150
```

In [12]:
```
# Appplying forward fill on my dataset
data = data.ffill()
data.head()
```

Out[12]:

Ticker	BAC	C	CL=F	DX-Y.NYB	GC=F	JPM	MS	SPY
Date								
2002-01-02	17.843855	276.156769	21.010000	115.790001	278.899994	18.587128	27.954092	75.131638
2002-01-03	17.824017	276.373352	20.370001	116.110001	278.200012	19.083126	28.764050	75.983566
2002-01-04	18.101761	281.516418	21.620001	116.330002	278.899994	19.942179	30.191624	76.490845
2002-01-07	17.886364	278.214081	21.480000	116.330002	278.600006	19.891054	29.963816	75.951057
2002-01-08	17.600115	267.981903	21.250000	116.830002	278.899994	19.737631	29.336084	75.775467

In [13]:
```
# How many missing values are present in my dataset
data.isnull().sum()
```

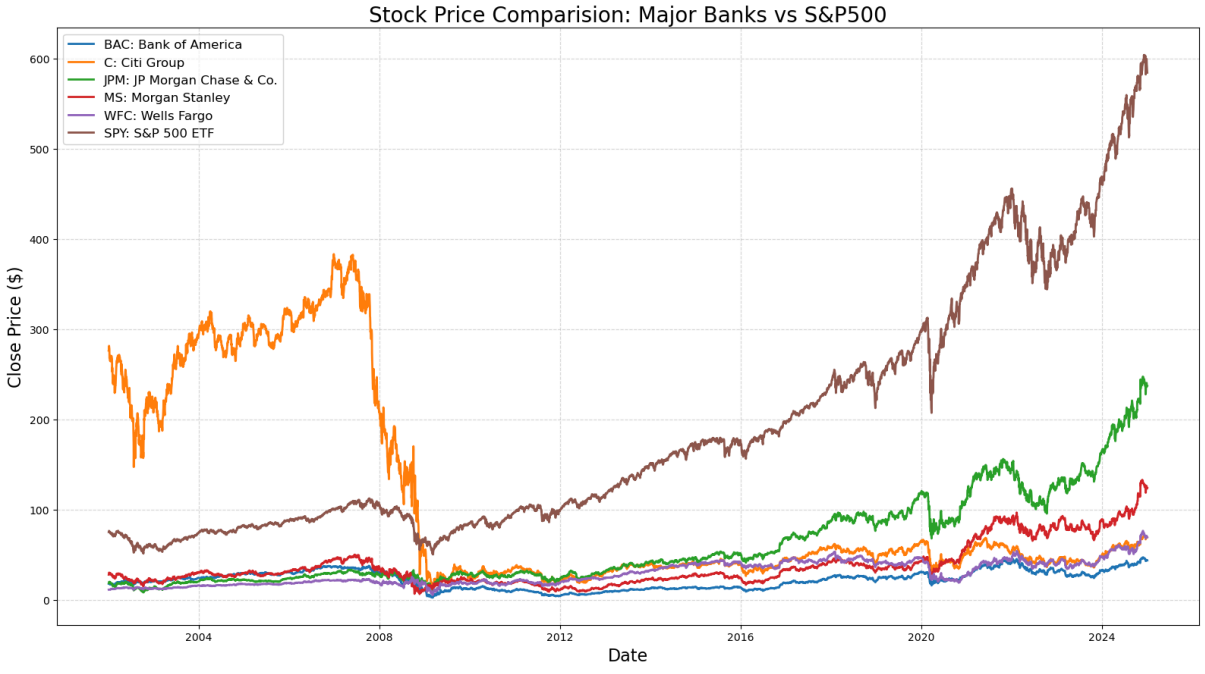
Out[13]:
```
Ticker
BAC         0
C           0
CL=F        0
DX-Y.NYB    0
GC=F        0
JPM         0
MS          0
SPY         0
WFC         0
^TNX        0
^VIX        0
dtype: int64
```

In [14]:
```
# Correlation in our data
data.corr()
```

Out[14]:

Ticker	BAC	C	CL=F	DX-Y.NYB	GC=F	JPM	MS	SPY	
Ticker									
BAC	1.000000	0.461779	-0.070637	0.360871	0.071502	0.569557	0.786216	0.544844	0.3
C	0.461779	1.000000	-0.328865	-0.056293	-0.753355	-0.404546	-0.078854	-0.429433	-0.5
CL=F	-0.070637	-0.328865	1.000000	-0.513374	0.436498	0.117551	0.110137	0.140270	0.1
DX-Y.NYB	0.360871	-0.056293	-0.513374	1.000000	0.165210	0.527508	0.509171	0.542913	0.4
GC=F	0.071502	-0.753355	0.436498	0.165210	1.000000	0.787185	0.581200	0.810607	0.6
JPM	0.569557	-0.404546	0.117551	0.527508	0.787185	1.000000	0.902370	0.987253	0.8
MS	0.786216	-0.078854	0.110137	0.509171	0.581200	0.902370	1.000000	0.901737	0.6
SPY	0.544844	-0.429433	0.140270	0.542913	0.810607	0.987253	0.901737	1.000000	0.8
WFC	0.342323	-0.509659	0.149889	0.449883	0.694652	0.834764	0.660625	0.824806	1.0
^TNX	0.381009	0.716295	-0.045080	0.055238	-0.526051	-0.196493	0.098139	-0.243903	-0.3
^VIX	-0.214983	-0.165584	-0.111997	0.003703	-0.022830	-0.136455	-0.153446	-0.126401	-0.3

In [15]:
```
import matplotlib.pyplot as plt

# Create a larger picture
plt.figure(figsize = (16,9))

# Plot each stock
plt.plot(data.index, data['BAC'], label = 'BAC: Bank of America', linewi
plt.plot(data.index, data['C'], label = 'C: Citi Group', linewidth = 2)
plt.plot(data.index, data['JPM'], label = 'JPM: JP Morgan Chase & Co.',
plt.plot(data.index, data['MS'], label = 'MS: Morgan Stanley', linewidth
plt.plot(data.index, data['WFC'], label = 'WFC: Wells Fargo', linewidth
plt.plot(data.index, data['SPY'], label = 'SPY: S&P 500 ETF', linewidth

# Title, label for X & Y Axis
plt.title('Stock Price Comparision: Major Banks vs S&P500', fontsize = 2
plt.xlabel('Date', fontsize = 16)
plt.ylabel('Close Price ($)', fontsize = 16)

# Add grid lines for readability
plt.grid(True, linestyle = '--', alpha = 0.5)

# Customize legend
plt.legend(fontsize = 12, loc = 'upper left')

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



In [16]:
```
data.head()
```

Out[16]:

Ticker	BAC	C	CL=F	DX-Y.NYB	GC=F	JPM	MS	SPY
Date								
2002-01-02	17.843855	276.156769	21.010000	115.790001	278.899994	18.587128	27.954092	75.131638
2002-01-03	17.824017	276.373352	20.370001	116.110001	278.200012	19.083126	28.764050	75.983566
2002-01-04	18.101761	281.516418	21.620001	116.330002	278.899994	19.942179	30.191624	76.490845
2002-01-07	17.886364	278.214081	21.480000	116.330002	278.600006	19.891054	29.963816	75.951057
2002-01-08	17.600115	267.981903	21.250000	116.830002	278.899994	19.737631	29.336084	75.775467

In [17]:
```
data.columns
```

Out[17]:
```
Index(['BAC', 'C', 'CL=F', 'DX-Y.NYB', 'GC=F', 'JPM', 'MS', 'SPY', 'WFC',
       '^TNX', '^VIX'],
      dtype='object', name='Ticker')
```

In [18]:
```
# Feature Engineering
df = pd.DataFrame(index = data.index)

# Create lag features - Stock Data
df['JPM(t-1)'] = data['JPM'].shift(1)
df['BAC(t-1)'] = data['BAC'].shift(1)
df['MS(t-1)'] = data['MS'].shift(1)
df['C(t-1)'] = data['C'].shift(1)
df['WFC(t-1)'] = data['WFC'].shift(1)
df['SPY(t-1)'] = data['SPY'].shift(1)

# Create lag features - Macroeconmic Data
df['VIX(t-1)'] = data['^VIX'].shift(1)
df['10Y_Yield(t-1)'] = data['^TNX'].shift(1)
df['Gold_Futures(t-1)'] = data['GC=F'].shift(1)
df['US_Dollar_Index(t-1)'] = data['DX-Y.NYB'].shift(1)
df['Crude_Oil_Futures(t-1)'] = data['CL=F'].shift(1)

# Technical Indicators = Moving Average and Rolling Volatility
df['BAC_MA5'] = data['BAC'].rolling(window = 5).mean().shift(1)
df['BAC_MA10'] = data['BAC'].rolling(window = 10).mean().shift(1)
df['BAC_Volatility5'] = data['BAC'].pct_change(5).shift(1)

# Create Target Variable
df['Target'] = data['BAC']

# Drop nan values
df = df.dropna()
df
```

Out[18]:

Date	JPM(t-1)	BAC(t-1)	MS(t-1)	C(t-1)	WFC(t-1)	SPY(t-1)	VIX(t-1)	10Y_Yield
2002-01-16	19.364363	17.415894	29.402170	269.173035	11.796279	74.884499	22.700001	4.8
2002-01-17	18.668945	17.231676	28.395838	264.517151	11.804087	73.369247	23.450001	4.8
2002-01-18	18.842793	17.322374	29.295429	270.147522	11.871764	73.922081	22.250000	4.9
2002-01-21	18.362141	17.231676	28.777010	270.472290	11.811899	73.583878	22.520000	4.8
2002-01-22	18.362141	17.231676	28.777010	270.472290	11.811899	73.583878	22.520000	4.8
...	...	...	...	...	...	...	...	
2024-12-24	235.713257	43.614761	122.643349	68.740494	69.810455	592.906433	16.780001	4.5
2024-12-26	239.589233	44.101688	125.215462	69.952347	70.849594	599.496582	14.270000	4.5
2024-12-27	240.409912	44.270622	126.171379	70.297180	71.017845	599.536499	14.730000	4.5
2024-12-30	238.462036	44.061939	124.919815	69.952347	70.374565	593.225464	15.950000	4.6
2024-12-31	236.632812	43.634636	123.924477	69.351349	69.681808	586.455811	17.400000	4.5

5800 rows × 15 columns

In [19]:
```
# Train our ML Algo
# a) Tell what is X variables and my Y Variable - Supervised ML Algo
# b) Split our data into training and testing (90:10)
# c) Apply ML Algo
# d) Do Prediction
# e) Evaluate the model based on R2, rmse and mse
# f) Visualtion => Actual vs Forecasted
```

In [20]:
```
# a) Tell what is X variables and my Y Variable
X = df.drop('Target', axis = 1)
Y = df['Target']
```

In [21]:
```
# b) Split our data into training and testing (90:10)
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, shuffle = Fals
```

In [22]:
```
# c) Apply ML Algo: Decision Tree

from sklearn.tree import DecisionTreeRegressor
dt_model = DecisionTreeRegressor(max_depth = 4) # Calling the DT Model
dt_model.fit(X_train, Y_train) # Train my DT Model

from sklearn.ensemble import RandomForestRegressor
rf_model = RandomForestRegressor(n_estimators = 100)
rf_model.fit(X_train, Y_train)

from sklearn.neighbors import KNeighborsRegressor
knn_model = KNeighborsRegressor(n_neighbors = 5)
knn_model.fit(X_train, Y_train)

from sklearn.svm import SVR
svr_model = SVR()
svr_model.fit(X_train, Y_train)
```

Out[22]:
```
SVR
SVR()
```

In [23]:
```
# d) Do Prediction: X_test and Y_test are my actual values
dt_pred = dt_model.predict(X_test)
rf_pred = rf_model.predict(X_test)
knn_pred = knn_model.predict(X_test)
svr_pred = svr_model.predict(X_test)
```

In [24]:
```
# Actual vs Predicted = Decision Tree
result = pd.DataFrame(Y_test.index)
result['Actual'] = Y_test.values
result['DT Prediction'] = dt_pred
result['RF Prediction'] = rf_pred
result['KNN Prediction'] = knn_pred
result['SVR Prediction'] = svr_pred
result
```

Out[24]:

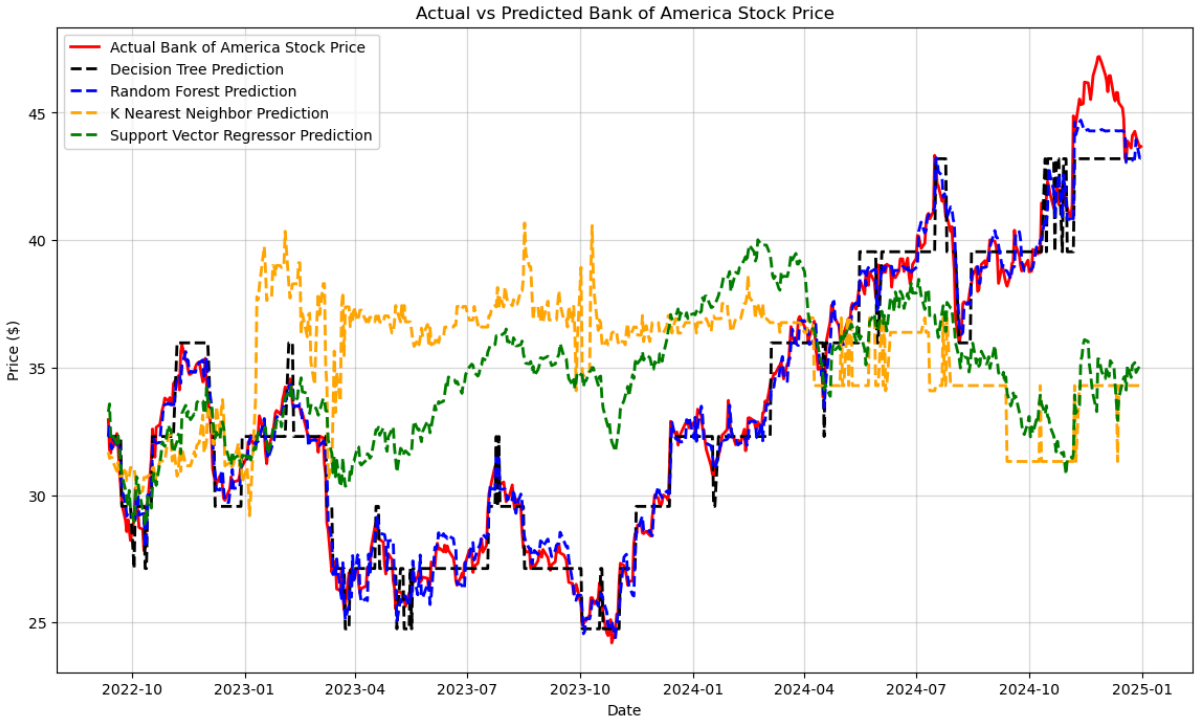
	Date	Actual	DT Prediction	RF Prediction	KNN Prediction	SVR Prediction
0	2022-09-12	32.954315	32.296849	32.291239	31.683958	33.265019
1	2022-09-13	31.767696	32.296849	32.776533	31.435372	33.574653
2	2022-09-14	31.646227	32.296849	31.895538	31.481785	32.224309
3	2022-09-15	32.244205	32.296849	31.812615	31.474262	32.349457
4	2022-09-16	31.879822	32.296849	32.189596	31.285307	32.301870
...	...	...	...	...	...	...
575	2024-12-24	44.101688	43.194882	43.130004	34.293399	35.082096
576	2024-12-26	44.270622	43.194882	43.255614	34.293399	35.195891
577	2024-12-27	44.061939	43.194882	43.936081	34.293399	34.808955
578	2024-12-30	43.634636	43.194882	43.227344	34.293399	35.070863
579	2024-12-31	43.674385	43.194882	43.248914	34.293399	35.078503

580 rows × 6 columns

In [25]:
```
# e) Evaluate the model based on R2, rmse and mse
from sklearn.metrics import r2_score, mean_squared_error

def evaluate_model(y_true, y_pred, model_name):
    r2 = r2_score(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred) # mse: mean squared error
    rmse = np.sqrt(mse) # rmse: root means sqaured error
    print("Model Name:", model_name)
    print("R2 Value", r2)
    print("MSE", mse)
    print("RMSE", rmse)
    print("\n")

# Y_test = Actual Value
# dt_pred, rf_pred = Predicted Value
evaluate_model(Y_test, dt_pred, "Decision Tree")
evaluate_model(Y_test, rf_pred, "Random Forest")
evaluate_model(Y_test, knn_pred, "K Nearest Neighbor")
evaluate_model(Y_test, svr_pred, "Support Vector Regressor")
```

```
Model Name: Decision Tree
R2 Value 0.9582605359398104
MSE 1.338860276907111
RMSE 1.1570913001604977


Model Name: Random Forest
R2 Value 0.9838263249812169
MSE 0.5187965754190673
RMSE 0.7202753469466155


Model Name: K Nearest Neighbor
R2 Value -0.5904608252834846
MSE 51.01658272081046
RMSE 7.14258935686565


Model Name: Support Vector Regressor
R2 Value 0.012732827952213976
MSE 31.66817852388099
RMSE 5.627448669146702
```

In [26]:
```
# f) Visualtion => Actual vs Forecasted

# Plot figure => tell figure size
plt.figure(figsize = (14,8))

#Plot Actual Value and Predicted Value (X = Date, Y = Stock Price)
plt.plot(Y_test.index, Y_test, label = 'Actual Bank of America Stock Pri
plt.plot(Y_test.index, dt_pred, label = 'Decision Tree Prediction', line
plt.plot(Y_test.index, rf_pred, label = "Random Forest Prediction", line
plt.plot(Y_test.index, knn_pred, label = "K Nearest Neighbor Prediction"
plt.plot(Y_test.index, svr_pred, label = "Support Vector Regressor Predi

# Highlight title, xlabel, and ylabel
plt.title("Actual vs Predicted Bank of America Stock Price")
plt.xlabel('Date')
plt.ylabel('Price ($)')
plt.grid(alpha = 0.5)
plt.legend()
plt.show()
```

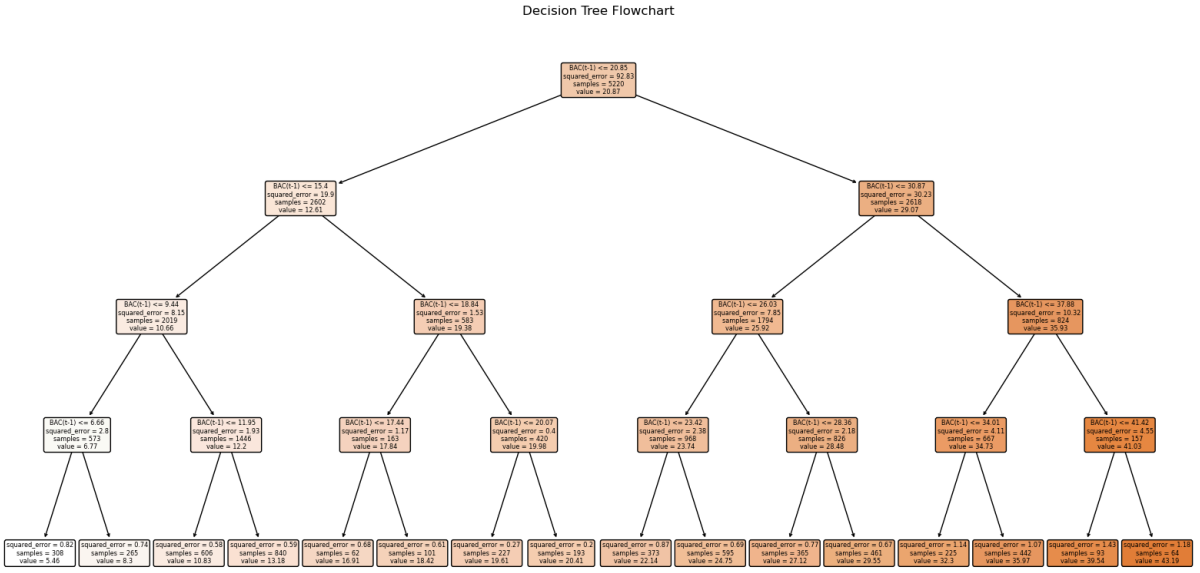


In [27]:
```
# What all features are important?
importance = dt_model.feature_importances_
features_name = X_train.columns

df_features = pd.DataFrame({'Feature':features_name, 'Importance': impor
df_features = df_features.sort_values(by = 'Importance', ascending = Fal
df_features
```

Out[27]:

	Feature	Importance
1	BAC(t-1)	1.0
0	JPM(t-1)	0.0
2	MS(t-1)	0.0
3	C(t-1)	0.0
4	WFC(t-1)	0.0
5	SPY(t-1)	0.0
6	VIX(t-1)	0.0
7	10Y_Yield(t-1)	0.0
8	Gold_Futures(t-1)	0.0
9	US_Dollar_Index(t-1)	0.0
10	Crude_Oil_Futures(t-1)	0.0
11	BAC_MA5	0.0
12	BAC_MA10	0.0
13	BAC_Volatility5	0.0

In [28]:
```
from sklearn.tree import plot_tree

plt.figure(figsize = (20,10))
plot_tree(dt_model, feature_names = X.columns, filled = True, rounded =
plt.title("Decision Tree Flowchart")
plt.show()
```



Thank You!!