1: Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_sco
re

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tcn import TCN


import math
import sys
print(sys.executable)
```

```
WARNING:tensorflow:From b:\Dublin City University\Practicum\Proj\venv_311
\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_sof
tmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_so
ftmax_cross_entropy instead.

b:\Dublin City University\Practicum\Proj\venv_311\Scripts\python.exe
```

2: Load and Explore Dataset

In [2]:
```
multimodal = pd.read_csv("B:/Dublin City University/Practicum/Proj/Dataset/main/processed/multimodal_dataset_final_v4_1.csv", parse_dates=["Date"])
multimodal.head()
```

Out[2]:

	Date	Open	High	Low	Close	Volume	Adj Close
0	2016-07-01	17924.240234	18002.380859	17916.910156	17949.369141	82160000	17949.369141
1	2016-06-30	17712.759766	17930.609375	17711.800781	17929.990234	133030000	17929.990234
2	2016-06-29	17456.019531	17704.509766	17456.019531	17694.679688	106380000	17694.679688
3	2016-06-28	17190.509766	17409.720703	17190.509766	17409.720703	112190000	17409.720703
4	2016-06-27	17355.210938	17355.210938	17063.080078	17140.240234	138740000	17140.240234

5 rows × 56 columns

In [3]:
```
missing_values = multimodal.isnull().sum()
print("\nMissing values per column:")
missing_values[missing_values > 0]
```

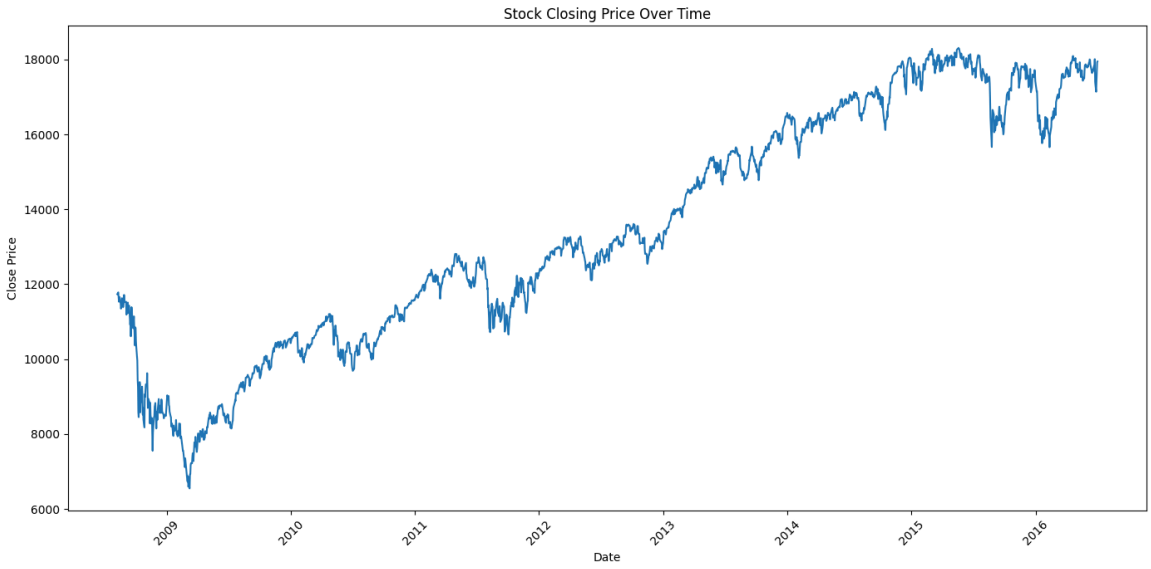
```
Missing values per column:
```

Out[3]:
```
Log_Returns           1
Volatility_Log_10    10
pct_change            1
Next_3_Close          3
Next_7_Close          7
Next_Close            1
dtype: int64
```

In [4]:
```
multimodal.describe()
multimodal.dtypes
```

Out[4]:
```
Date                          datetime64[ns]
Open                                 float64
High                                 float64
Low                                  float64
Close                                float64
Volume                                 int64
Adj Close                            float64
Log_Returns                          float64
Volatility_Log_10                    float64
cl-op                                float64
hi-lo                                float64
Label                                  int64
vader_news_sentiment                 float64
FinBERT_news_sentiment               float64
Smart_news_sentiment                 float64
news_buying_intent                   float64
news_selling_intent                  float64
news_uncertainty_intent              float64
news_urgency_intent                  float64
news_prediction_intent               float64
news_fear_intent                     float64
news_greed_intent                    float64
news_question_intent                 float64
news_action_intent                   float64
vader_reddit_sentiment               float64
FinBERT_reddit_sentiment             float64
Smart_reddit_sentiment               float64
reddit_buying_intent                 float64
reddit_selling_intent                float64
reddit_uncertainty_intent            float64
reddit_urgency_intent                float64
reddit_prediction_intent             float64
reddit_fear_intent                   float64
reddit_greed_intent                  float64
reddit_question_intent               float64
reddit_action_intent                 float64
Target                                 int64
pct_change                           float64
finbert_final_sentiment              float64
total_buying_intent                  float64
total_selling_intent                 float64
total_uncertainty_intent             float64
total_urgency_intent                 float64
total_prediction_intent              float64
total_fear_intent                    float64
total_greed_intent                   float64
total_question_intent                float64
total_action_intent                  float64
sentiment_minus_uncertainty          float64
sentiment_minus_fear                 float64
sentiment_minus_action               float64
sentiment_minus_urgency              float64
sentiment_minus_prediction           float64
Next_3_Close                         float64
Next_7_Close                         float64
Next_Close                           float64
dtype: object
```

In [5]:
```
plt.figure(figsize=(14, 7))
plt.plot(pd.to_datetime(multimodal['Date']), multimodal['Close'])
plt.title('Stock Closing Price Over Time')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



3: Data Preprocessing

In [6]:
```
# Drop top 10 (rolling NaNs) and bottom 7 (from shift(-7))
multimodal_modelling = multimodal.iloc[10:-7].copy()

# Optional: reset index
# multimodal_modelling.reset_index(drop=True, inplace=True)

# Sanity check
print(f"Shape: {multimodal_modelling.shape}")
print(multimodal_modelling.isnull().sum())
```

```
Shape: (1972, 56)
Date                           0
Open                           0
High                           0
Low                            0
Close                          0
Volume                         0
Adj Close                      0
Log_Returns                    0
Volatility_Log_10              0
cl-op                          0
hi-lo                          0
Label                          0
vader_news_sentiment           0
FinBERT_news_sentiment         0
Smart_news_sentiment           0
news_buying_intent             0
news_selling_intent            0
news_uncertainty_intent        0
news_urgency_intent            0
news_prediction_intent         0
news_fear_intent               0
news_greed_intent              0
news_question_intent           0
news_action_intent             0
vader_reddit_sentiment         0
FinBERT_reddit_sentiment       0
Smart_reddit_sentiment         0
reddit_buying_intent           0
reddit_selling_intent          0
reddit_uncertainty_intent      0
reddit_urgency_intent          0
reddit_prediction_intent       0
reddit_fear_intent             0
reddit_greed_intent            0
reddit_question_intent         0
reddit_action_intent           0
Target                         0
pct_change                     0
finbert_final_sentiment        0
total_buying_intent            0
total_selling_intent           0
total_uncertainty_intent       0
total_urgency_intent           0
total_prediction_intent        0
total_fear_intent              0
total_greed_intent             0
total_question_intent          0
total_action_intent            0
sentiment_minus_uncertainty    0
sentiment_minus_fear           0
sentiment_minus_action         0
sentiment_minus_urgency        0
sentiment_minus_prediction     0
Next_3_Close                   0
Next_7_Close                   0
Next_Close                     0
dtype: int64
```

In [7]:
```
df_targets = multimodal_modelling[["Date", "Target", "Label", "Next_Close",
"Next_3_Close", "Next_7_Close"]].copy()

df_arima = multimodal_modelling[["Date", "Close", "Next_Close"]].copy()
df_arima.set_index("Date", inplace=True)

drop_cols_lstm = ["Date", "Label", "Target", "Next_Close", "Next_3_Close",
"Next_7_Close"]
df_lstm = multimodal_modelling.drop(columns=drop_cols_lstm).copy()

drop_cols_tcn = ["Date", "Label", "Target", "Next_Close", "Next_3_Close",
"Next_7_Close"]
df_tcn = multimodal_modelling.drop(columns=drop_cols_tcn).copy()
```

4: Time Series Stationarity Analysis for ARIMA

In [8]:
```
df_arima.head()
```

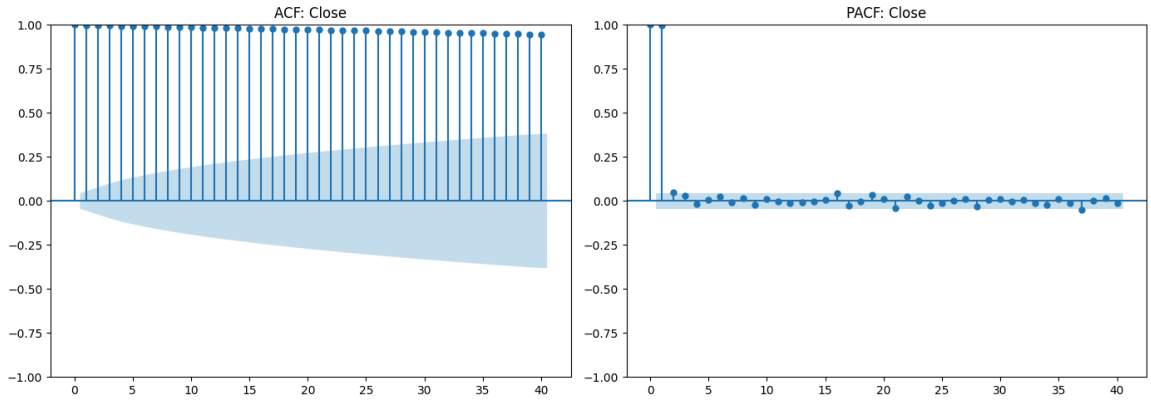
Out[8]:

Date	Close	Next_Close
2016-06-17	17675.160156	17733.099609
2016-06-16	17733.099609	17640.169922
2016-06-15	17640.169922	17674.820312
2016-06-14	17674.820312	17732.480469
2016-06-13	17732.480469	17865.339844

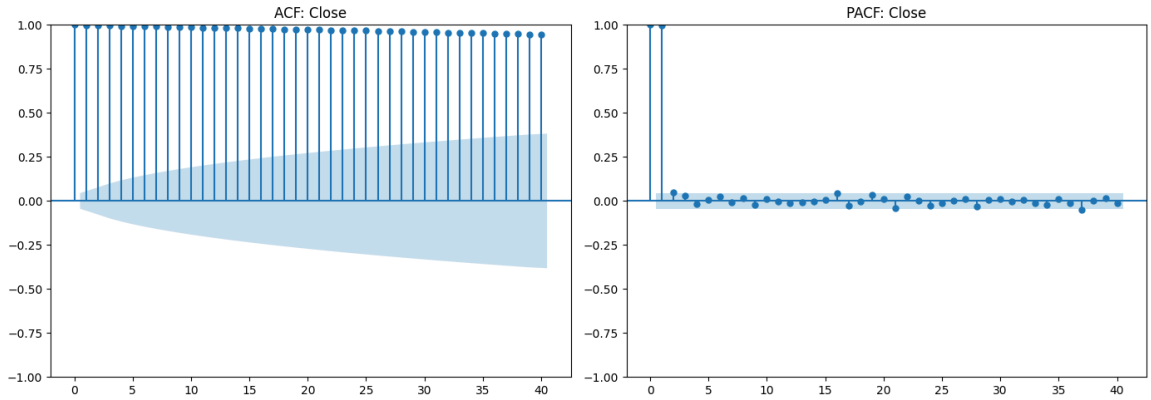
In [9]:
```
result = adfuller(df_arima["Close"])
print(f"ADF Statistic: {result[0]}")
print(f"p-value: {result[1]}")
```

```
ADF Statistic: -1.210171226984291
p-value: 0.6692050737972017
```

In [10]:
```
plt.figure(figsize=(14, 5))
plt.subplot(1, 2, 1)
plot_acf(multimodal_modelling["Close"], lags=40, ax=plt.gca())
plt.title("ACF: Close")

plt.subplot(1, 2, 2)
plot_pacf(multimodal_modelling["Close"], lags=40, ax=plt.gca(), method='yw
m')
plt.title("PACF: Close")

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



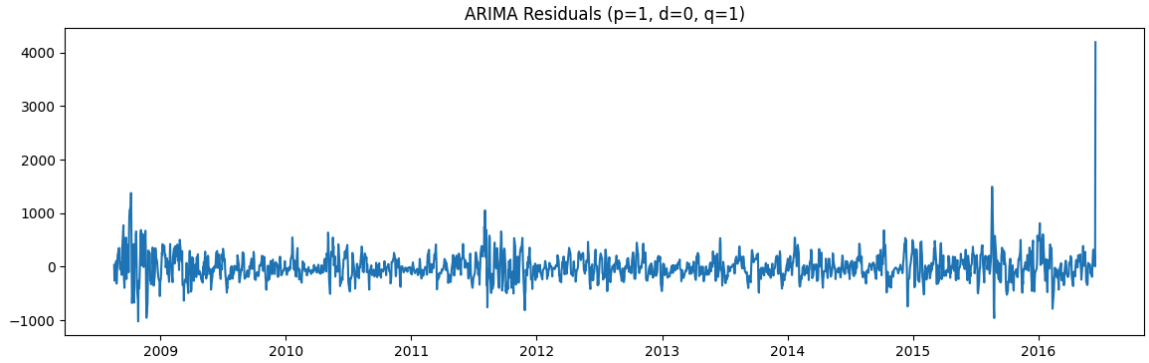
In [11]:
```
plt.figure(figsize=(14, 5))
plt.subplot(1, 2, 1)
plot_acf(df_arima["Close"], lags=40, ax=plt.gca())
plt.title("ACF: Close")

plt.subplot(1, 2, 2)
plot_pacf(df_arima["Close"], lags=40, ax=plt.gca(), method='ywm')
plt.title("PACF: Close")

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



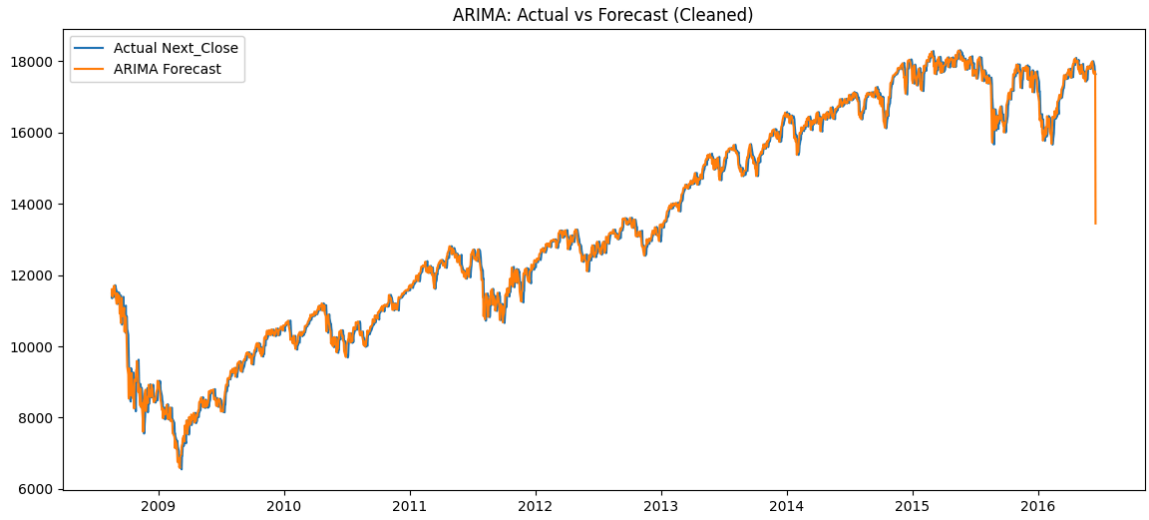
In [12]:
```
# Fit ARIMA model (you can ADF test and gridsearch later)
model = ARIMA(df_arima["Close"], order=(1,0,1))
model_fit = model.fit()

# Forecast next day
forecast = model_fit.predict(start=0, end=len(df_arima)-1)
true = df_arima["Next_Close"]

# Shift Close forward to align with Next_Close
forecast = forecast.shift(1)  # now forecast[i] ≈ Close[i+1]
forecast = forecast[:len(true)]

# Align forecast and true by dropping NaNs introduced by shift
mask = forecast.notna()
forecast_clean = forecast[mask]
true_clean = true[mask]

# Evaluation
print("R²:", r2_score(true_clean, forecast_clean))
print("MSE:", mean_squared_error(true_clean, forecast_clean))
```

```
R²: 0.9937197053841297
MSE: 61862.3376472596

b:\Dublin City University\Practicum\Proj\venv_311\Lib\site-packages\statsm
odels\tsa\base\tsa_model.py:473: ValueWarning: A date index has been provi
ded, but it has no associated frequency information and so will be ignored
when e.g. forecasting.
  self._init_dates(dates, freq)
b:\Dublin City University\Practicum\Proj\venv_311\Lib\site-packages\statsm
odels\tsa\base\tsa_model.py:473: ValueWarning: A date index has been provi
ded, but it is not monotonic and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
b:\Dublin City University\Practicum\Proj\venv_311\Lib\site-packages\statsm
odels\tsa\base\tsa_model.py:473: ValueWarning: A date index has been provi
ded, but it has no associated frequency information and so will be ignored
when e.g. forecasting.
  self._init_dates(dates, freq)
b:\Dublin City University\Practicum\Proj\venv_311\Lib\site-packages\statsm
odels\tsa\base\tsa_model.py:473: ValueWarning: A date index has been provi
ded, but it is not monotonic and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
b:\Dublin City University\Practicum\Proj\venv_311\Lib\site-packages\statsm
odels\tsa\base\tsa_model.py:473: ValueWarning: A date index has been provi
ded, but it has no associated frequency information and so will be ignored
when e.g. forecasting.
  self._init_dates(dates, freq)
b:\Dublin City University\Practicum\Proj\venv_311\Lib\site-packages\statsm
odels\tsa\base\tsa_model.py:473: ValueWarning: A date index has been provi
ded, but it is not monotonic and so will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
```

In [13]:
```
residuals = true_clean - forecast_clean
plt.figure(figsize=(14,4))
plt.plot(residuals)
plt.title("ARIMA Residuals (p=1, d=0, q=1)")
plt.show()
```



In [14]:
```
plt.figure(figsize=(14,6))
plt.plot(forecast_clean.index, true_clean, label="Actual Next_Close")
plt.plot(forecast_clean.index, forecast_clean, label="ARIMA Forecast")
plt.title("ARIMA: Actual vs Forecast (Cleaned)")
plt.legend()
plt.show()
```



LSTM Model

In [15]:
```
df_lstm.columns
```

Out[15]:
```
Index(['Open', 'High', 'Low', 'Close', 'Volume', 'Adj Close', 'Log_Returns',
       'Volatility_Log_10', 'cl-op', 'hi-lo', 'vader_news_sentiment',
       'FinBERT_news_sentiment', 'Smart_news_sentiment', 'news_buying_intent',
       'news_selling_intent', 'news_uncertainty_intent', 'news_urgency_intent',
       'news_prediction_intent', 'news_fear_intent', 'news_greed_intent',
       'news_question_intent', 'news_action_intent', 'vader_reddit_sentiment',
       'FinBERT_reddit_sentiment', 'Smart_reddit_sentiment',
       'reddit_buying_intent', 'reddit_selling_intent',
       'reddit_uncertainty_intent', 'reddit_urgency_intent',
       'reddit_prediction_intent', 'reddit_fear_intent', 'reddit_greed_intent',
       'reddit_question_intent', 'reddit_action_intent', 'pct_change',
       'finbert_final_sentiment', 'total_buying_intent',
       'total_selling_intent', 'total_uncertainty_intent',
       'total_urgency_intent', 'total_prediction_intent', 'total_fear_intent',
       'total_greed_intent', 'total_question_intent', 'total_action_intent',
       'sentiment_minus_uncertainty', 'sentiment_minus_fear',
       'sentiment_minus_action', 'sentiment_minus_urgency',
       'sentiment_minus_prediction'],
      dtype='object')
```

In [16]:
```
# Separate features and target
X = df_lstm
y = df_targets[['Next_Close', 'Next_3_Close', 'Next_7_Close']].values

# Scale features and target
X_scaler = MinMaxScaler()
y_scaler = MinMaxScaler()

X_scaled = X_scaler.fit_transform(X)
y_scaled = y_scaler.fit_transform(y)

# Create sequences
def create_sequences(X, y, window_size=60):
    Xs, ys = [], []
    for i in range(window_size, len(X)):
        Xs.append(X[i-window_size:i])
        ys.append(y[i])
    return np.array(Xs), np.array(ys)

X_seq, y_seq = create_sequences(X_scaled, y_scaled)

# Return final sequence shapes
X_seq.shape, y_seq.shape
```

Out[16]:
```
((1912, 60, 50), (1912, 3))
```

In [19]:
```
# simple lstm model
X_train, X_test, y_train, y_test = train_test_split(X_seq, y_seq, test_size
=0.2, shuffle=False)

model = Sequential()
model.add(LSTM(64, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(3, activation='linear'))

model.compile(optimizer='adam', loss='mse')

history = model.fit(
    X_train, y_train,
    epochs=20,
    batch_size=32,
    validation_split=0.1,
    verbose=1
)

y_pred_scaled = model.predict(X_test)

y_pred = y_scaler.inverse_transform(y_pred_scaled)
y_true = y_scaler.inverse_transform(y_test)

y_pred[:5], y_true[:5]

rmse_s = np.sqrt(mean_squared_error(y_true, y_pred))
r2_s = r2_score(y_true, y_pred)

print(f"RMSE: {rmse_s:.2f}")
print(f"R²: {r2_s:.4f}")

# Evaluate individual targets
target_names = ['Next_Close', 'Next_3_Close', 'Next_7_Close']

for i, name in enumerate(target_names):
    rmse_is = np.sqrt(mean_squared_error(y_true[:, i], y_pred[:, i]))
    r2_is = r2_score(y_true[:, i], y_pred[:, i])
    print(f"{name} — RMSE: {rmse_is:.2f}, R²: {r2_is:.4f}")
```

```
WARNING:tensorflow:From b:\Dublin City University\Practicum\Proj\venv_311
\Lib\site-packages\keras\src\optimizers\__init__.py:309: The name tf.trai
n.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instea
d.

Epoch 1/20
WARNING:tensorflow:From b:\Dublin City University\Practicum\Proj\venv_311
\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.Rag
gedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorV
alue instead.

43/43 [==============================] - 2s 22ms/step - loss: 0.1208 - val
_loss: 0.0317
Epoch 2/20
43/43 [==============================] - 1s 12ms/step - loss: 0.0176 - val
_loss: 0.0037
Epoch 3/20
43/43 [==============================] - 1s 12ms/step - loss: 0.0121 - val
_loss: 0.0019
Epoch 4/20
43/43 [==============================] - 1s 12ms/step - loss: 0.0105 - val
_loss: 0.0026
Epoch 5/20
43/43 [==============================] - 1s 12ms/step - loss: 0.0090 - val
_loss: 0.0025
Epoch 6/20
43/43 [==============================] - 1s 13ms/step - loss: 0.0074 - val
_loss: 0.0016
Epoch 7/20
43/43 [==============================] - 1s 13ms/step - loss: 0.0075 - val
_loss: 0.0010
Epoch 8/20
43/43 [==============================] - 1s 15ms/step - loss: 0.0074 - val
_loss: 0.0012
Epoch 9/20
43/43 [==============================] - 1s 15ms/step - loss: 0.0071 - val
_loss: 0.0011
Epoch 10/20
43/43 [==============================] - 1s 14ms/step - loss: 0.0060 - val
_loss: 8.7106e-04
Epoch 11/20
43/43 [==============================] - 1s 16ms/step - loss: 0.0062 - val
_loss: 0.0023
Epoch 12/20
43/43 [==============================] - 1s 13ms/step - loss: 0.0059 - val
_loss: 0.0011
Epoch 13/20
43/43 [==============================] - 1s 13ms/step - loss: 0.0062 - val
_loss: 9.1592e-04
Epoch 14/20
43/43 [==============================] - 1s 13ms/step - loss: 0.0052 - val
_loss: 7.6712e-04
Epoch 15/20
43/43 [==============================] - 1s 13ms/step - loss: 0.0055 - val
_loss: 0.0017
Epoch 16/20
43/43 [==============================] - 1s 13ms/step - loss: 0.0053 - val
_loss: 8.1409e-04
Epoch 17/20
43/43 [==============================] - 1s 12ms/step - loss: 0.0050 - val
_loss: 6.3779e-04
```

```
Epoch 18/20
43/43 [==============================] - 1s 13ms/step - loss: 0.0047 - val
_loss: 0.0012
Epoch 19/20
43/43 [==============================] - 1s 15ms/step - loss: 0.0046 - val
_loss: 6.8214e-04
Epoch 20/20
43/43 [==============================] - 1s 13ms/step - loss: 0.0045 - val
_loss: 0.0012
12/12 [==============================] - 0s 5ms/step
RMSE: 1032.29
R²: 0.1996
Next_Close — RMSE: 878.52, R²: 0.4006
Next_3_Close — RMSE: 858.13, R²: 0.4387
Next_7_Close — RMSE: 1299.48, R²: -0.2405
```

In [20]:
```
# two Stacked layers LSTM Model
model = Sequential()
model.add(LSTM(64, return_sequences=True, input_shape=(X_train.shape[1], X_
train.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(32))
model.add(Dropout(0.2))
model.add(Dense(3, activation='linear'))

model.compile(optimizer='adam', loss='mse')

early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_wei
ghts=True)

history = model.fit(
    X_train, y_train,
    epochs=100,
    batch_size=32,
    validation_split=0.1,
    callbacks=[early_stop],
    verbose=1
)

y_pred_scaled = model.predict(X_test)

y_pred = y_scaler.inverse_transform(y_pred_scaled)
y_true = y_scaler.inverse_transform(y_test)

rmse = np.sqrt(mean_squared_error(y_true, y_pred))
r2 = r2_score(y_true, y_pred)

print(f"RMSE: {rmse:.2f}")
print(f"R²: {r2:.4f}")

# Evaluate individual targets
target_names = ['Next_Close', 'Next_3_Close', 'Next_7_Close']

for i, name in enumerate(target_names):
    rmse_i = np.sqrt(mean_squared_error(y_true[:, i], y_pred[:, i]))
    r2_i = r2_score(y_true[:, i], y_pred[:, i])
    print(f"{name} — RMSE: {rmse_i:.2f}, R²: {r2_i:.4f}")
```

```
Epoch 1/100
43/43 [==============================] - 4s 34ms/step - loss: 0.0734 - val
_loss: 0.0043
Epoch 2/100
43/43 [==============================] - 1s 21ms/step - loss: 0.0188 - val
_loss: 0.0032
Epoch 3/100
43/43 [==============================] - 1s 21ms/step - loss: 0.0151 - val
_loss: 0.0012
Epoch 4/100
43/43 [==============================] - 1s 21ms/step - loss: 0.0138 - val
_loss: 6.7413e-04
Epoch 5/100
43/43 [==============================] - 1s 21ms/step - loss: 0.0125 - val
_loss: 0.0011
Epoch 6/100
43/43 [==============================] - 1s 21ms/step - loss: 0.0122 - val
_loss: 7.3642e-04
Epoch 7/100
43/43 [==============================] - 1s 21ms/step - loss: 0.0116 - val
_loss: 6.7150e-04
Epoch 8/100
43/43 [==============================] - 1s 22ms/step - loss: 0.0100 - val
_loss: 5.6399e-04
Epoch 9/100
43/43 [==============================] - 1s 22ms/step - loss: 0.0100 - val
_loss: 6.0739e-04
Epoch 10/100
43/43 [==============================] - 1s 22ms/step - loss: 0.0088 - val
_loss: 5.4255e-04
Epoch 11/100
43/43 [==============================] - 1s 22ms/step - loss: 0.0089 - val
_loss: 5.4264e-04
Epoch 12/100
43/43 [==============================] - 1s 22ms/step - loss: 0.0084 - val
_loss: 5.8334e-04
Epoch 13/100
43/43 [==============================] - 1s 23ms/step - loss: 0.0073 - val
_loss: 0.0011
Epoch 14/100
43/43 [==============================] - 1s 22ms/step - loss: 0.0075 - val
_loss: 7.8224e-04
Epoch 15/100
43/43 [==============================] - 1s 22ms/step - loss: 0.0071 - val
_loss: 0.0012
12/12 [==============================] - 1s 8ms/step
RMSE: 544.46
R²: 0.7769
Next_Close — RMSE: 478.60, R²: 0.8221
Next_3_Close — RMSE: 484.93, R²: 0.8208
Next_7_Close — RMSE: 651.99, R²: 0.6877
```

Temporal Convolutional Networks

In [23]:
```
X = df_tcn.values
y = df_targets[['Next_Close', 'Next_3_Close', 'Next_7_Close']].values

# Scale features and target
X_scaler = MinMaxScaler()
y_scaler = MinMaxScaler()

X_scaled = X_scaler.fit_transform(X)
y_scaled = y_scaler.fit_transform(y)
```

In [24]:
```
def create_sequences(X, y, window_size=60):
    X_seq, y_seq = [], []
    for i in range(window_size, len(X)):
        X_seq.append(X[i-window_size:i])
        y_seq.append(y[i])
    return np.array(X_seq), np.array(y_seq)

X_seq, y_seq = create_sequences(X_scaled, y_scaled)
print(f"X_seq shape: {X_seq.shape}, y_seq shape: {y_seq.shape}")
```

```
X_seq shape: (1912, 60, 50), y_seq shape: (1912, 3)
```

In [25]:
```
split = int(0.8 * len(X_seq))
X_train, X_test = X_seq[:split], X_seq[split:]
y_train, y_test = y_seq[:split], y_seq[split:]
```

In [32]:
```
model = Sequential([
    TCN(
        input_shape=(X_train.shape[1], X_train.shape[2]),
        nb_filters=64,
        kernel_size=3,
        dilations=[1, 2, 4, 8],
        return_sequences=False,
        activation='relu',
        dropout_rate=0.2
    ),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(3)  # Output layer for Next_Close, Next_3_Close, Next_7_Close
])
model.compile(optimizer='adam', loss='mse')
```

In [33]:
```
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)

history = model.fit(
    X_train, y_train,
    validation_split=0.1,
    epochs=100,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/100
43/43 [==============================] - 3s 21ms/step - loss: 0.9192 - val
_loss: 0.0099
Epoch 2/100
43/43 [==============================] - 1s 15ms/step - loss: 0.0985 - val
_loss: 0.0076
Epoch 3/100
43/43 [==============================] - 1s 15ms/step - loss: 0.0686 - val
_loss: 0.0043
Epoch 4/100
43/43 [==============================] - 1s 14ms/step - loss: 0.0539 - val
_loss: 0.0034
Epoch 5/100
43/43 [==============================] - 1s 14ms/step - loss: 0.0456 - val
_loss: 0.0028
Epoch 6/100
43/43 [==============================] - 1s 14ms/step - loss: 0.0387 - val
_loss: 0.0025
Epoch 7/100
43/43 [==============================] - 1s 14ms/step - loss: 0.0361 - val
_loss: 0.0024
Epoch 8/100
43/43 [==============================] - 1s 14ms/step - loss: 0.0333 - val
_loss: 0.0021
Epoch 9/100
43/43 [==============================] - 1s 14ms/step - loss: 0.0290 - val
_loss: 0.0021
Epoch 10/100
43/43 [==============================] - 1s 14ms/step - loss: 0.0278 - val
_loss: 0.0017
Epoch 11/100
43/43 [==============================] - 1s 15ms/step - loss: 0.0245 - val
_loss: 0.0018
Epoch 12/100
43/43 [==============================] - 1s 14ms/step - loss: 0.0229 - val
_loss: 0.0023
Epoch 13/100
43/43 [==============================] - 1s 14ms/step - loss: 0.0214 - val
_loss: 0.0022
Epoch 14/100
43/43 [==============================] - 1s 14ms/step - loss: 0.0212 - val
_loss: 0.0025
Epoch 15/100
43/43 [==============================] - 1s 14ms/step - loss: 0.0216 - val
_loss: 0.0030
```

In [34]:
```
y_pred_scaled = model.predict(X_test)
y_pred = y_scaler.inverse_transform(y_pred_scaled)
y_true = y_scaler.inverse_transform(y_test)

mse_t = mean_squared_error(y_true, y_pred)
r2_t = r2_score(y_true, y_pred)

print(f"✅ MSE: {mse_t:.4f}")
print(f"✅ R²: {r2_t:.4f}")

# Evaluate individual targets
target_names = ['Next_Close', 'Next_3_Close', 'Next_7_Close']

for i, name in enumerate(target_names):
    rmse_it = np.sqrt(mean_squared_error(y_true[:, i], y_pred[:, i]))
    r2_it = r2_score(y_true[:, i], y_pred[:, i])
    print(f"{name} — RMSE: {rmse_it:.2f}, R²: {r2_it:.4f}")
```

```
12/12 [==============================] - 0s 5ms/step
✅ MSE: 3001186.1816
✅ R²: -1.2714
Next_Close — RMSE: 1685.84, R²: -1.2071
Next_3_Close — RMSE: 1712.76, R²: -1.2359
Next_7_Close — RMSE: 1796.65, R²: -1.3713
```

In [36]:
```
from tensorflow.keras.utils import plot_model

plot_model(model, to_file='tcn_model_architecture.png', show_shapes=True, s
how_layer_names=True)
```

```
You must install pydot (`pip install pydot`) and install graphviz (see ins
tructions at https://graphviz.gitlab.io/download/) for plot_model to work.
```

In [35]:
```
plt.figure(figsize=(14,6))
plt.plot(y_true, label='Actual')
plt.plot(y_pred, label='TCN Prediction')
plt.title("TCN: Actual vs Predicted Next_Close")
plt.legend()
plt.show()

# Residuals
residuals = y_true - y_pred
plt.figure(figsize=(14,4))
plt.plot(residuals)
plt.axhline(0, color='gray', linestyle='--')
plt.title("Residuals")
plt.show()
```