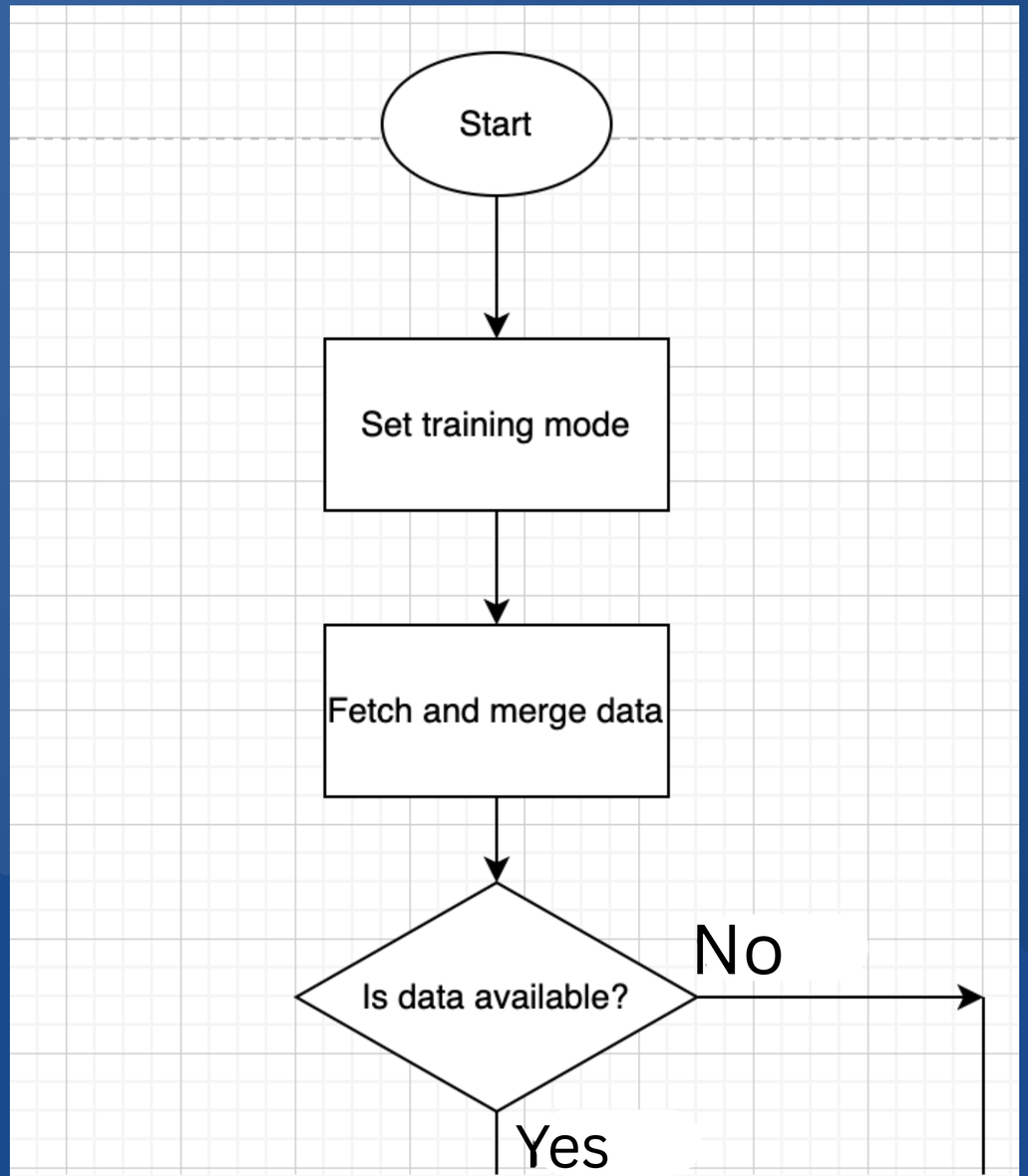
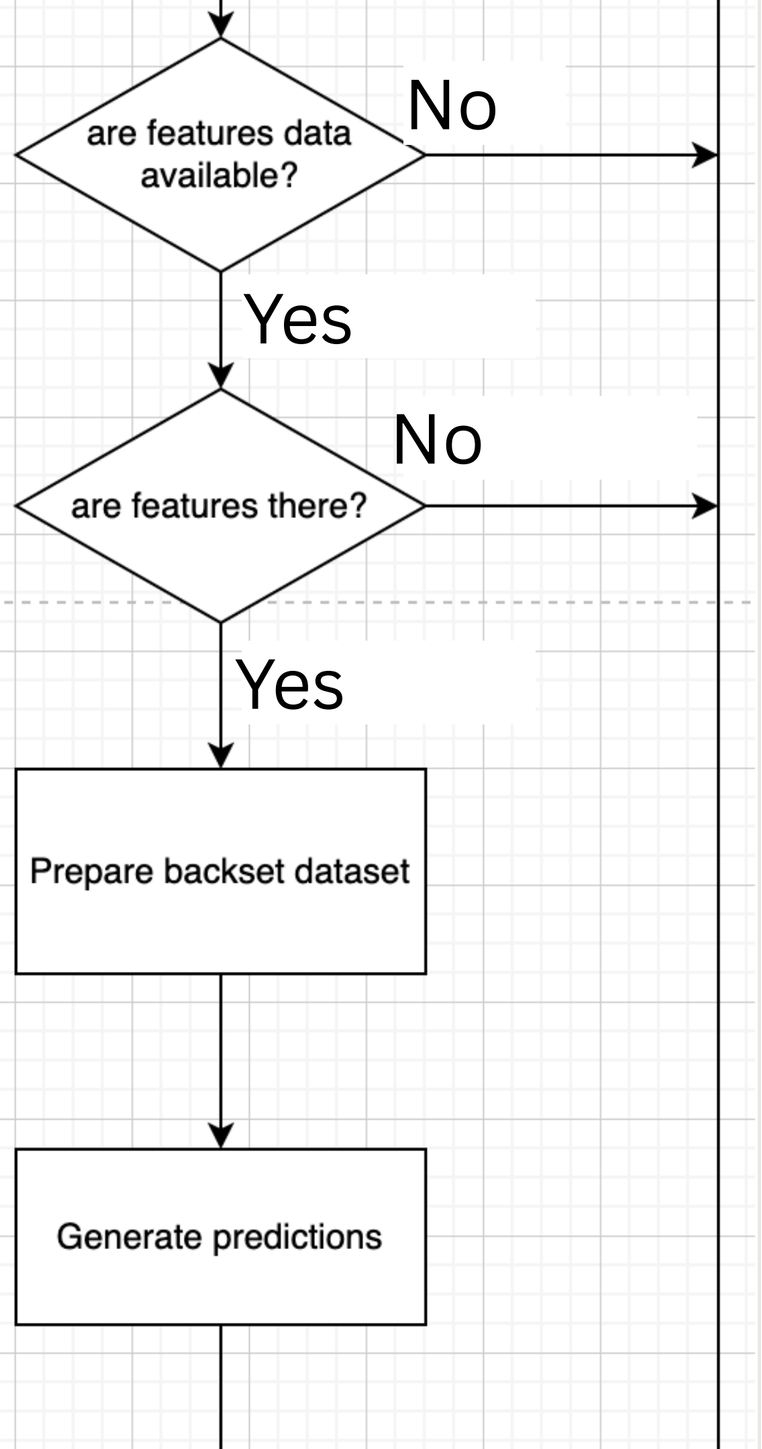
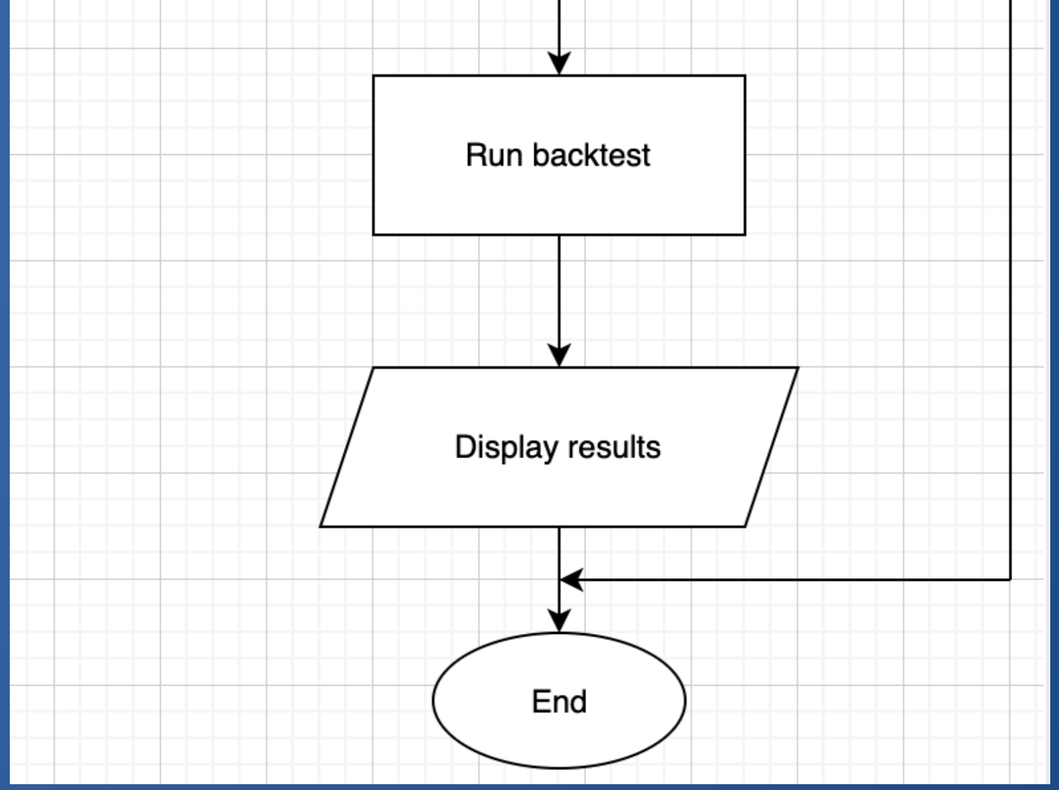
QUANTITATIVE TRADING

CONCEPTUAL DIAGRAM




are features data available?
No
Yes
are features there?
No
Yes
Prepare backset dataset
Generate predictions


Run backtest
Display results
End

THE PROGRAM

```
# 1. Data Collection
merged_data = await DataProcessor.merge_data_sources()
if merged_data.empty:
    print("Failed to fetch data")
    return False
```

```
# 2. Feature Engineering
feature_data = DataProcessor.calculate_features(merged_data)
if feature_data.empty:
    print("Feature calculation failed")
    return False
```

```python
# 3. Prepare Data
X = feature_data.drop(columns=['target'])
y = feature_data['target']

scaler = config.SCALER_TYPE()
X_scaled = scaler.fit_transform(X)
X_scaled_df = pd.DataFrame(X_scaled, index=X.index, columns=X.columns)

X_seq, y_seq, seq_indices = LSTMModel.create_sequences(
    X_scaled_df, y, config.LOOKBACK_WINDOW)

# Train/Val/Test Split
train_idx = int(len(X_seq) * (1 - config.TEST_SIZE - config.VALIDATION_SIZE))
val_idx = int(len(X_seq) * (1 - config.TEST_SIZE))

X_train, y_train = X_seq[:train_idx], y_seq[:train_idx]
X_val, y_val = X_seq[train_idx:val_idx], y_seq[train_idx:val_idx]
X_test, y_test = X_seq[val_idx:], y_seq[val_idx:]
test_dates = seq_indices[val_idx:]
```

```python
# 4. Train Model
model = LSTMModel.build((config.LOOKBACK_WINDOW, X_train.shape[2]))
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=config.NN_EPOCHS,
    batch_size=config.NN_BATCH_SIZE,
    callbacks=LSTMModel.create_callbacks(),
    verbose=1
)

# 5. Save Artifacts
model.save(config.model_path)
joblib.dump(scaler, config.scaler_path)
joblib.dump({
    'test_dates': test_dates,
    'feature_names': X.columns.tolist()
}, config.metadata_path)
```

WHAT HAPPEN IN TRAIN MODE?

1. THE PIPELINE STARTS BY PRINTING A HEADER, COLLECTING DATA FROM MULTIPLE SOURCES, MERGING THEM, AND PERFORMING FEATURE ENGINEERING TO CREATE USEFUL INDICATORS.
2. IT SEPARATES THE FEATURES (X) AND TARGET VARIABLE (Y), SCALES THE DATA, RESHAPES IT INTO SEQUENCES FOR THE LSTM MODEL, AND SPLITS IT INTO TRAINING, VALIDATION, AND TEST SETS.
3. THE LSTM MODEL IS BUILT AND TRAINED OVER SEVERAL EPOCHS, THEN THE TRAINED MODEL, SCALER, AND METADATA (SUCH AS FEATURE NAMES AND TEST DATES) ARE SAVED FOR FUTURE USE.

THE PROGRAM

```
# 1. Load Artifacts
try:
    model = load_model(config.model_path)
    scaler = joblib.load(config.scaler_path)
    metadata = joblib.load(config.metadata_path)
    test_dates = metadata['test_dates']
    feature_names = metadata['feature_names']
except Exception as e:
    print(f"Error loading artifacts: {e}")
    return None
```

```python
# 2. Prepare Backtest Data
merged_data = await DataProcessor.merge_data_sources()
if merged_data.empty:
    print("Failed to fetch backtest data")
    return None

feature_data = DataProcessor.calculate_features(merged_data)
if feature_data.empty:
    print("Feature calculation failed")
    return None

# Get data for test period + lookback
start_date = test_dates[0] - timedelta(hours=config.LOOKBACK_WINDOW)
feature_data = feature_data.loc[start_date:]

# Scale features
X_test = feature_data[feature_names]
X_scaled = scaler.transform(X_test)
X_scaled_df = pd.DataFrame(X_scaled, index=X_test.index, columns=feature_names)

# Create sequences aligned with test dates
X_test_seq = []
aligned_dates = []

for i in range(len(X_scaled_df) - config.LOOKBACK_WINDOW):
    target_date = X_scaled_df.index[i + config.LOOKBACK_WINDOW]
    if target_date in test_dates:
        X_test_seq.append(X_scaled_df.iloc[i:i+config.LOOKBACK_WINDOW].values)
        aligned_dates.append(target_date)
```

```python
X_test_seq = np.array(X_test_seq)
aligned_dates = pd.DatetimeIndex(aligned_dates)
```

```python
# 3. Generate Predictions
predictions = model.predict(X_test_seq).flatten()

# Generate signals
signals = np.zeros(len(predictions))
signals[predictions > config.ENTRY_THRESHOLD] = 1
signals[predictions < -config.ENTRY_THRESHOLD] = -1

signals_df = pd.DataFrame({
    'signal': signals,
    'prediction': predictions
}, index=aligned_dates)
```

```python
# 4. Run Backtest
price_data = feature_data.loc[aligned_dates, ['close', 'volatility']]
backtester = Backtester()
results = backtester.run_backtest(price_data, signals_df['signal'])

# 5. Display Results
print("\n=== Backtest Results ===")
print(f"Final Portfolio Value: ${results['final_value']:,.2f}")
print(f"Total Return: {(results['final_value']/config.INITIAL_CAPITAL-1)*100:.2f}%")
print(f"Sharpe Ratio: {results['sharpe_ratio']:.2f}")
print(f"Max Drawdown: {results['max_drawdown']*100:.2f}%")
print(f"Number of Trades: {len(results['trades'])}")

if results['trades_csv_path']:
    print(f"\nDetailed trades saved to: {results['trades_csv_path']}")

return results
```

WHAT HAPPEN IN BACKTEST MODE?

1. THE PIPELINE BEGINS BY LOADING THE SAVED MODEL, SCALER, AND METADATA, THEN GATHERS AND PROCESSES NEW DATA, APPLIES THE SAME FEATURE ENGINEERING, SCALES IT, AND ALIGNS SEQUENCES WITH TEST DATES.
2. THE MODEL MAKES PREDICTIONS ON THE TEST SEQUENCES, AND TRADING SIGNALS (BUY, SELL, HOLD) ARE GENERATED BASED ON THRESHOLD RULES.
3. THESE SIGNALS ARE USED IN A SIMULATED TRADING STRATEGY OVER HISTORICAL PRICE DATA, AND PERFORMANCE METRICS LIKE PORTFOLIO VALUE, RETURN, SHARPE RATIO, DRAWDOWN, AND TRADE COUNT ARE PRINTED FOR EVALUATION.

Tung Tung Tung Sahur

Thank you!