Certainly! Here's the **comprehensive table** you requested. It includes:

Model	Core Ideas (Pros)	Limitations (Cons)	Technical Requirements	Time Encoding	Forecast Target	Eval Metrics	Method
Prophet	Seasonality & holiday built-in	Weak with non-linearity	Minimal	Internal	Price, seasonality	RMSE, MAPE	Additive Model
LSTM	Captures temporal non-linearity	Needs much data	No stationarity	Cyclical or embedded	Price, return, volatility	RMSE, MAPE, R^2	DL (RNN)
GRU	Efficient vs. LSTM	Misses long dependencies	Same as LSTM	Same	Price, volatility	Same as LSTM	DL (RNN)
Bi-LSTM	Considers future/past context	Future not always known	Same as LSTM	Same	Price (post-event), sentiment	Same as LSTM	DL (RNN)
CNN (Time Series)	Local pattern learning	Poor for long term	No stationarity	Treat time as channel	Price trend, short-term	MSE, Accuracy (patterns)	DL (CNN)
Transformer	Global attention, long memory	Heavy compute	None, large data needed	Sin/Cos positional	Price, return, volatility, regime	RMSE, MAE, R^2	DL (Attention)
Informer/Autoformer	Long horizon attention models	Complex	No stationarity	Learned embeddings	Price, volatility, regime	RSE, MASE	DL (Efficient Transformer)
N-HiTS	Fast + scalable forecasting	Research stage	None	Internal	Trend, price	MASE, RSE	DL (Residual Hierarchical)
DeepAR	Probabilistic & multiple series	Needs AR-like structure	None	Internal embeddings	Price, volatility	NLL, CRPS	DL (Auto-regressive RNN)
Deep State Space	Uncertainty + deep learning	Heavy compute	None	Fourier/time basis	Regime, macro-volatility	NLL, RMSE	Bayesian DL
TFT	Interpretable, deep attention	Tuning complex	None	Learned embeddings	Price, volatility, return	MAE, RMSE, R^2	DL (Attention + Gating)
XGBoost/LightGBM	Strong on structured data	Manual lags needed	Scaling + feature prep	One-hot / cyclical	Price, return, volatility	R^2, MAE, RMSE	Tree Boosting
TabNet	Deep learning for tabular data	New; needs fine-tuning	No stationarity	Embedded internally	Price, return	RMSE, R^2	DL (Attentive FCN)
DDPG (RL)	Continuous actions	Reward shaping is tricky	Agent tuning	In state vector	Return, trading actions	Sharpe, PnL, DD	RL (Policy Gradient)
TD3	Stable vs. DDPG (2 critics)	Slower training	Same	Same	Return, risk mgmt	Sharpe, PnL	RL (Value + Policy)
SAC	High exploration entropy	Hyperparameter sensitive	High compute	Same	Risk-adjusted returns	Sharpe, Entropy, Profit	RL (Entropy Optim.)
PPO	Stable & safe RL training	Reward design matters	Sample efficiency	Same	Portfolio weights	Sharpe, MDD	RL (Clipped Policy)
A3C	Fast, parallel training	Sample inefficiency	Same	Same	Return, policy learning	Sharpe, Reward curve	RL (Actor-Critic)
AlphaStock	Combines RL + DL like AlphaZero	Very compute heavy	RL + DL infra	Structured state	Portfolio strategy	Profit, Sharpe	Hybrid (Policy RL + DL)
Meta-Learning (MAML, etc.)	Fast adaptation to change	Experimental	Time-aware adaptation	Dynamic embeddings	Regime change, multi-goal	Transfer performance	Meta-Learning

📜 Project-Wide Notes and Critical Considerations

---

1. Stationarity Concerns

- Classical models (ARIMA, SARIMA) **require stationarity** (use differencing, detrending).
- DL/RL models (LSTM, Transformer, PPO, SAC, etc.) **do not require stationarity**, but **large shifts** (structural breaks) can still harm them.

✅ **Best practice**: Detrend input data (optional) and scale features.

---

2. Feature Encoding for Time

- **Cyclical Encoding** (preferred for DL/RL):
  $month\_sin=\sin(2\pi\times month12), month\_cos=\cos(2\pi\times month12) month\_sin=\sin(2\pi\times 12month), month\_cos=\cos(2\pi\times 12month)$

- **One-hot encoding** or **embedding layers** for categorical time features (week, day-of-week) when used in tree models or DL hybrid models.

---

3. Technical Indicators to Include (examples)

Type	Examples
Trend	Moving Averages (SMA, EMA), MACD
Momentum	RSI, Stochastic Oscillator, Williams %R
Volatility	ATR (Average True Range), Bollinger Bands
Volume-based	OBV (On-Balance Volume), Volume Price Trend

---

4. Fundamental Indicators

- Earnings per Share (EPS)
- Price-to-Earnings Ratio (P/E)
- Price-to-Book Ratio (P/B)
- Dividend Yield
- Debt-to-Equity Ratio (D/E)
- Revenue, Net Income Growth
- Free Cash Flow
- Insider Buying Activity

---

5. Indices and Macro Factors to Monitor

- S&P500, NASDAQ, Dow Jones (US indices)
- VIX (Volatility Index)
- Interest Rates (e.g., Fed Funds Rate)
- Inflation Data (CPI, PPI)
- Employment Data (Unemployment rate)
- Global Economic Sentiment Indices
- Oil Prices, Gold Prices, etc.

---

6. Time Series Specific Concerns

Concern	Description
Seasonality	Holiday effects, quarterly earnings
Regime Changes	Market crash, major political events
Noise	Random market movements unrelated to fundamentals
Data Snooping Bias	Overfitting due to too much data mining
Outliers	Extreme points can distort learning

✅ Take Care Notes (Golden Rules)

- Always **normalize/standardize** inputs (especially for DL).
- Watch for **data leakage** (future information contaminating the past).
- **Expand features** carefully: lags, rolling means, cumulative sums.
- Use **cross-validation specific to time series** (e.g., expanding windows).
- Regular **model retraining** is needed (market changes constantly).

📌 Common Forecasting Goals by Model

Forecast Target	Suggested Models
Price	ARIMA, LSTM, Transformer, DeepAR, TFT
Return	LSTM, GRU, RL (DDPG, PPO), TFT
Volatility	Transformer, DeepAR, SAC
Momentum	CNN, LSTM, XGBoost
Risk	PPO, SAC, TD3, TFT
Portfolio Allocation	AlphaStock, RL methods (DDPG, PPO, A3C)
Multi-horizon / Sequence Output	Transformer, TFT, N-HiTS, DeepAR, Meta-Learning

---

📚 Notes on Evaluation Metrics

Metric	Purpose
RMSE / MAE / MSE	General accuracy on price prediction
MAPE / SMAPE	Scaled error, interpretability
Sharpe Ratio	Return/risk in trading
Max Drawdown (MDD)	Risk of capital loss
CRPS / NLL	Probabilistic forecasts (DeepAR, Bayesian)
R^2 (R-squared)	Goodness of fit
PnL (Profit)	Total return for RL agents

You're asking for **clear, text-based flowcharts** that explain:

- **Data flow** and **logic steps** for **each model**.
- In a way that teaches you **how to implement** a real **stock market prediction system** with each approach.
- Focused on **Deep Learning (DL)** and **Reinforcement Learning (RL)**.

I'll break it down carefully.

📈 Stock Prediction - Model Implementation Flows

🧠 1. Deep Learning Models (DL)

➡️ LSTM / GRU for Stock Prediction

sql
CopyEdit

```
Raw Data (OHLCV + indicators + time features)
    ↓
Data Cleaning (remove NaN, outliers)
    ↓
Feature Engineering
    - Technical indicators (RSI, MACD)
    - Cyclical encoding (month/day/week sin-cos)
    - Normalize/standardize inputs
    ↓
Data Preparation
    - Create sequences (lookback windows)
    - X = past N days' features
    - y = future price / return
    ↓
Train-Test Split (time-aware split)
    ↓
Build Model
    - LSTM layers (1-3 layers)
    - Dropout (for regularization)
    - Dense output (linear or sigmoid)
    ↓
Train Model (early stopping, low LR)
    ↓
Evaluate
    - RMSE, MAE, R² on validation set
    ↓
Predict future price/return
```

➡️ Bi-LSTM (Bidirectional LSTM)

(Same as above, **BUT**)

- At model step:
  ➡️ **Use Bidirectional wrapper** around LSTM layers to allow backward+forward temporal attention.

➡️ CNN for Stock Prediction

java
CopyEdit

```
Raw Data (candlestick OHLCV, indicators)
    ↓
Data Cleaning & Feature Engineering
    ↓
Data Shaping
    - Convert into 2D shape (sequence length × features)
    ↓
Build CNN Model
    - Conv1D or Conv2D layers
    - Pooling layers (optional)
    - Flatten → Dense output
    ↓
Train and Evaluate
    ↓
Predict patterns (short-term momentum, price)
```

➡️ Transformer (Vanilla)

mathematica
CopyEdit

```
Raw Data (OHLCV, Indicators, Macro data)
    ↓
Feature Engineering
    - Technical + Fundamental data
    - Cyclic Encode time features
    ↓
Build Input Sequences
    ↓
Positional Encoding
    ↓
Build Transformer Encoder-Decoder
    - Multi-Head Attention
    - Feedforward Networks
    ↓
Train Model
    ↓
Evaluate
    - RMSE, MAE, MAPE
    ↓
Predict multiple steps ahead
```

➡️ Informer / Autoformer / Reformer

(Almost same as Transformer, but optimized)

java
CopyEdit

```
Raw Data
    ↓
Feature Engineering
    ↓
Long-Horizon Sequence Preparation
    ↓
Efficient Transformer Variant
    - Sparse attention
    - Decomposition of trends (Autoformer)
    ↓
Train and Forecast long horizons
```

➡️ DeepAR (Amazon)

java
CopyEdit

```
Raw Data (multiple stock series)
    ↓
Time Series Normalization
    ↓
Feed Past Data (Autoregressive Inputs)
    ↓
Recurrent Neural Network (RNN)
    ↓
Output Future Probabilistic Forecasts
    - Predict mean and variance
```

➡️ Temporal Fusion Transformer (TFT)

sql
CopyEdit

```
Raw Data
    ↓
Static Features + Dynamic Features Separation
    ↓
Embedding Layers
    - Time Embeddings
    - Feature Embeddings
    ↓
Gated Residual Networks (GRN)
    ↓
Attention Layers
    - Select important historical steps
    - Select important features
    ↓
Train with Quantile Loss
    ↓
Forecast future price/volatility/return
```

➡️ N-BEATS / N-HiTS

scss

CopyEdit

```
Raw Data (time series)
    ↓
Normalization
    ↓
Build Fully Connected Block Layers
    - Trend basis
    - Seasonality basis
    ↓
Predict sequences (future horizon)
    ↓
(Optionally ensemble many models)
```

🤖 2. Reinforcement Learning Models (RL)

➡️ DDPG / TD3 / SAC / PPO (RL for Trading)

sql
CopyEdit

```
Raw Data (OHLCV, indicators, macro)
    ↓
Environment Setup
    - Define State (features, indicators, past returns)
    - Define Action (buy, sell, hold, allocation %)
    - Define Reward (portfolio return, Sharpe Ratio)
    ↓
Agent Setup
    - Actor network: outputs action
    - Critic network: evaluates action
    ↓
Training Loop
    - Observe State
    - Select Action
    - Execute Action
    - Receive Reward
    - Update Policy (Actor + Critic)
    ↓
Evaluation
    - Total return
    - Sharpe Ratio
    - Max Drawdown
```

➡️ AlphaStock (Advanced RL + DL approach)

pgsql
CopyEdit

```
Raw Data (market states + historical data)
    ↓
Environment = Portfolio Simulator
    ↓
RL Agent = Deep Network Policy (like AlphaZero)
```

```
    - Actor network
    - Critic network
    ↓
Self-play / Simulated Trading
    ↓
Reward Shaping
    - Profit, Risk Control, Transaction Costs
    ↓
Training Massive Simulations
    ↓
Evaluate on Unseen Market Conditions
```

➡️ Meta-Learning (MAML for Stock Prediction)

csharp
CopyEdit

```
Raw Data (market tasks)
    ↓
Meta-Training Phase
    - Learn an initialization that adapts fast
    ↓
Meta-Testing Phase
    - Fine-tune quickly on new stocks / new market regimes
    ↓
Adapted Model predicts price, return, regime
```

📌 Overall Deep Learning/RL Stock Prediction Workflow

markdown
CopyEdit

```
1. Problem Formulation
    - Price prediction? Return? Volatility?
2. Data Collection
    - OHLCV, Technical, Macro, Fundamental
3. Data Cleaning
    - NaN removal, Outlier smoothing
4. Feature Engineering
    - Technical Indicators
    - Cyclical Date Encoding
5. Model Selection
    - DL (LSTM, Transformer) for pure forecasting
    - RL (PPO, DDPG) for action/trading strategy
6. Model Training
    - Time-aware validation (expanding window CV)
7. Model Evaluation
    - RMSE, MAPE, Sharpe Ratio
8. Backtesting
    - Simulate strategies on historical data
9. Deployment
    - Predict live or trade live
10. Monitoring
```

```
- Retrain and adapt to market regime changes
```

✅ Important Note

- **DL = Prediction** (future values)
- **RL = Decision Making** (how to act in market)

They can work **together** (first predict price, then act accordingly)!

Multicollinearity (high correlation between two or more input features) can seriously hurt many models (especially linear ones like ARIMA, SARIMAX, or even tree models if too severe).

Here are some **excellent Python libraries and methods** to **detect multicollinearity**:

📚 Libraries and Methods to Check Multicollinearity

Library	Functionality	Example Function / Method	Notes
statsmodels	Calculate Variance Inflation Factor (VIF)	`variance_inflation_factor()`	Gold standard for VIF checks
pingouin	VIF and partial correlations	`pingouin.vif()`	Very easy, returns clean DataFrame
scikit-learn	Correlation matrix + PCA analysis	`np.corrcoef(), PCA()`	Not direct VIF but useful
seaborn + matplotlib	Visual correlation heatmaps	`sns.heatmap(corr_matrix)`	Fast visual spotting of collinear pairs
pycaret	Built-in multicollinearity removal in preprocess	`setup(..., remove_multicollinearity=True)`	Full pipeline automation
feature-engine	Select/remove collinear features	`feature_engine.selection.DropCorrelatedFeatures`	Specialized for data cleaning

---

📌 Typical Code Snippets for Checking Multicollinearity:

---

1. VIF Calculation (statsmodels)

python
CopyEdit

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
import pandas as pd

# Assume `X` is your features DataFrame
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
```

```
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

print(vif_data)
```

- **VIF > 5**: Moderate multicollinearity
- **VIF > 10**: Serious multicollinearity

---

2. Correlation Matrix with Heatmap (seaborn)

python
CopyEdit

```
import seaborn as sns
import matplotlib.pyplot as plt

corr = X.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.show()
```

- Look for **correlation coefficients > 0.8 or < -0.8** between features!

---

3. Automatic VIF Check (pingouin)

python
CopyEdit

```
import pingouin as pg

vif = pg.vif(X)
print(vif)
```

- Much faster, good for big DataFrames.

✅ Best Practice for DL models:

- Deep Learning (LSTM, Transformer, etc.) are **less sensitive** to multicollinearity.
- For **classical models** (ARIMA, XGBoost without regularization, LightGBM default settings), **you must clean collinear features** first!

⚡ Recommended Quick Start

- For small projects: use **pingouin** or **statsmodels** VIF.
- For large pipelines: use **feature-engine** inside a preprocessing pipeline.