Created by Daniel Zaldaña @ZaldanaDaniel

Time Series Analysis

Cheat Sheet

1. Basic Concepts

Time Series Components

- **Trend** (T_t): Long-term direction
- **Seasonal** (S_t): Regular patterns
- **Cyclical** (C_t): Irregular fluctuations
- **Random** (R_t): Unexplained variation

Decomposition Models

Additive: $Y_t = T_t + S_t + C_t + R_t$

Multiplicative: $Y_t = T_t \times S_t \times C_t \times R_t$

2. Statistical Measures

Core Statistics

Mean:

$$\bar{X} = \frac{1}{n}\sum_{t=1}^{n} X_t$$

Moving Average:

$$MA(k) = \frac{1}{k}\sum_{i=0}^{k-1} X_{t-i}$$

Variance:

$$\sigma^2 = \frac{1}{n-1}\sum_{t=1}^{n}(X_t - \bar{X})^2$$

3. Autocorrelation

Correlation Measures

ACF:

$$\rho_k = \frac{\sum_{t=k+1}^{n}(X_t - \bar{X})(X_{t-k} - \bar{X})}{\sum_{t=1}^{n}(X_t - \bar{X})^2}$$

PACF:

$$\phi_{kk} = \text{Corr}(X_t, X_{t-k} | X_{t-1}, ..., X_{t-k+1})$$

4. Time Series Models

Model Specifications

AR(p):

$$X_t = c + \sum_{i=1}^{p} \phi_i X_{t-i} + \varepsilon_t$$

MA(q):

$$X_t = \mu + \varepsilon_t + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}$$

ARMA(p,q):

$$X_t = c + \sum_{i=1}^{p} \phi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + \varepsilon_t$$

ARIMA(p,d,q):

$$(1-B)^d (X_t - \mu) = \sum_{i=1}^{p} \phi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + \varepsilon_t$$

5. Forecasting Metrics

Error Measures

MAE:

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|Y_t - \hat{Y}_t|$$

RMSE:

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(Y_t - \hat{Y}_t)^2}$$

MAPE:

$$MAPE = \frac{100\%}{n}\sum_{t=1}^{n}|\frac{Y_t - \hat{Y}_t}{Y_t}|$$

6. Stationarity Tests

Stationarity Conditions

Weak Stationarity:

- $E[X_t] = \mu$ (constant mean)
- $Var(X_t) = \sigma^2$ (constant variance)
- $Cov(X_t, X_{t+k}) = \gamma_k$ (covariance depends only on k)

Augmented Dickey-Fuller: $\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \sum_{i=1}^{p}\delta_i \Delta y_{t-i} + \varepsilon_t$

KPSS Test: $\eta_c = \frac{\sum_{t=1}^{T} S_t^2}{T^2\hat{\sigma}^2}$ where $S_t = \sum_{i=1}^{t} e_i$
Unit Root Condition: $|1 - \sum_{i=1}^{p}\phi_i z^i| \neq 0$ for $|z| \leq 1$

7. Advanced Statistical Measures

Additional Metrics

Cross-Correlation Function (CCF): $\rho_{xy}(k) = \frac{\sum_{t=k+1}^{n}(X_t-\bar{X})(Y_{t-k}-\bar{Y})}{\sqrt{\sum_{t=1}^{n}(X_t-\bar{X})^2}\sqrt{\sum_{t=1}^{n}(Y_t-\bar{Y})^2}}$
Ljung-Box Q-statistic: $Q(m) = n(n+2)\sum_{k=1}^{m}\frac{\hat{\rho}_k^2}{n-k}$
Information Criteria: $AIC = -2\ln(L) + 2k$ $BIC = -2\ln(L) + k\ln(n)$

8. Time Series Patterns

Pattern Analysis

Exponential Growth: $Y_t = Y_0(1+r)^t$
Logistic Growth: $Y_t = \frac{K}{1+(\frac{K-Y_0}{Y_0})e^{-rt}}$
Holt-Winters Multiplicative:

$$\text{Level: } L_t = \alpha\frac{Y_t}{S_{t-s}} + (1-\alpha)(L_{t-1} + T_{t-1})$$
$$\text{Trend: } T_t = \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1}$$
$$\text{Seasonal: } S_t = \gamma\frac{Y_t}{L_t} + (1-\gamma)S_{t-s}$$

9. Spectral Analysis

Frequency Domain

Periodogram: $I(\omega) = \frac{1}{2\pi n}|\sum_{t=1}^{n} X_t e^{-i\omega t}|^2$
Spectral Density: $f(\omega) = \frac{1}{2\pi}\sum_{k=-\infty}^{\infty}\gamma_k e^{-ik\omega}$
Coherence: $C_{xy}(\omega) = \frac{|f_{xy}(\omega)|^2}{f_{xx}(\omega)f_{yy}(\omega)}$

10. Model Diagnostics

Diagnostic Tests

Durbin-Watson: $DW = \frac{\sum_{t=2}^{n}(e_t - e_{t-1})^2}{\sum_{t=1}^{n} e_t^2}$
Breusch-Godfrey: $LM = (n-p)R^2$
Forecast Error Decomposition:

$$MSE = \frac{1}{n}\sum(Y_t - \hat{Y}_t)^2$$
$$= (\bar{Y} - \bar{\hat{Y}})^2 + (s_Y - r s_{\hat{Y}})^2 + 2(1-r)s_Y s_{\hat{Y}}$$

11. Seasonal Analysis

Seasonal Metrics

Seasonal Index:

$$SI_i = \frac{\text{Average for season } i}{\text{Overall average}} \times 100$$

SARIMA Model:

$$\phi_p(B)\Phi_P(B^s)(1-B)^d(1-B^s)^D y_t = \theta_q(B)\Theta_Q(B^s)\varepsilon_t$$

1. Basic Statistics

Statistical Components

```python
import numpy as np
import pandas as pd

def calculate_basic_stats(data):
    """Calculate basic statistics"""
    return {
        'mean': np.mean(data),
        'variance': np.var(data, ddof=1),
        'std': np.std(data, ddof=1)
    }

def moving_average(data, window):
    """Calculate moving average"""
    return pd.Series(data).rolling(
        window=window
    ).mean().values
```

2. Correlation Analysis

ACF and PACF

```python
from statsmodels.tsa.stattools import (
    acf, pacf
)

def correlation_analysis(data, lags):
    """Calculate ACF and PACF"""
    acf_vals = acf(
        data,
        nlags=lags,
        fft=True
    )
    pacf_vals = pacf(
        data,
        nlags=lags,
        method='yw'
    )
    return {
        'acf': acf_vals,
        'pacf': pacf_vals
    }
```

3. Time Series Models

Model Implementations

```python
from statsmodels.tsa.arima.model import (
    ARIMA
)

def fit_arima_models(data):
    """Fit ARIMA and variants"""
    models = {}

    # AR(1) model
    models['ar'] = ARIMA(
        data,
        order=(1, 0, 0)
    ).fit()

    # MA(1) model
    models['ma'] = ARIMA(
        data,
        order=(0, 0, 1)
    ).fit()

    # ARMA(1,1)
    models['arma'] = ARIMA(
        data,
        order=(1, 0, 1)
    ).fit()

    # ARIMA(1,1,1)
    models['arima'] = ARIMA(
        data,
        order=(1, 1, 1)
    ).fit()

    return models
```

4. Forecasting Metrics

Error Measures

```
def calculate_metrics(actual, pred):
    """Calculate forecast metrics"""

    # Mean Absolute Error
    mae = np.mean(np.abs(
        actual - pred
    ))

    # Root Mean Square Error
    rmse = np.sqrt(np.mean(
        (actual - pred) ** 2
    ))

    # Mean Absolute Percentage Error
    mape = np.mean(np.abs(
        (actual - pred) / actual
    )) * 100

    return {
        'mae': mae,
        'rmse': rmse,
        'mape': mape
    }
```

5. Stationarity Tests

Stationarity Analysis

```python
from statsmodels.tsa.stattools import (
    adfuller, kpss
)

def check_stationarity(data):
    """Perform stationarity tests"""

    # ADF Test
    adf_result = adfuller(data)

    # KPSS Test
    kpss_result = kpss(data)

    # Rolling statistics
    roll_mean = pd.Series(data
        ).rolling(window=12).mean()
    roll_std = pd.Series(data
        ).rolling(window=12).std()

    return {
        'adf_stat': adf_result[0],
        'adf_pval': adf_result[1],
        'kpss_stat': kpss_result[0],
        'kpss_pval': kpss_result[1],
        'roll_mean': roll_mean,
        'roll_std': roll_std
    }
```

6. Advanced Measures

Advanced Statistics

```python
from statsmodels.stats.diagnostic import (
    acorr_ljungbox
)

def advanced_stats(data):
    """Calculate advanced metrics"""

    # Ljung-Box Test
    lb_test = acorr_ljungbox(
        data,
        lags=10
    )

    # Information Criteria
    model = ARIMA(
        data,
        order=(1,0,0)
    ).fit()

    return {
        'lb_stat': lb_test.lb_stat,
        'lb_pval': lb_test.lb_pvalue,
        'aic': model.aic,
        'bic': model.bic
    }
```

7. Seasonal Components

Seasonal Analysis

```
from statsmodels.tsa.seasonal import (
    seasonal_decompose
)

def analyze_seasonality(data, period):
    """Analyze seasonal patterns"""

    # Decomposition
    decomp = seasonal_decompose(
        data,
        period=period,
        model='multiplicative'
    )

    # Seasonal Indices
    indices = pd.Series(
        decomp.seasonal
    ).unique()

    return {
        'trend': decomp.trend,
        'seasonal': decomp.seasonal,
        'residual': decomp.resid,
        'indices': indices
    }
```

8. Growth Models

Growth Patterns

```
from scipy.optimize import curve_fit

def fit_growth_models(data, time):
    """Fit growth models"""

    def exp_growth(t, y0, r):
        return y0 * (1 + r) ** t

    def logistic(t, K, y0, r):
        return K / (1 + (
            (K - y0) / y0
        ) * np.exp(-r * t))

    # Fit models
    exp_params = curve_fit(
        exp_growth,
        time,
        data
    )[0]

    log_params = curve_fit(
        logistic,
        time,
        data
    )[0]

    return {
        'exp': exp_params,
        'logistic': log_params
    }
```

9. Spectral Analysis

Frequency Domain

```
from scipy import signal

def spectral_analysis(data, fs=1.0):
    """Perform spectral analysis"""

    # Periodogram
    freqs, psd = signal.periodogram(
        data,
        fs=fs
    )

    # Spectral Density
    f_welch, psd_welch = signal.welch(
        data,
        fs=fs
    )

    return {
        'freq': freqs,
        'psd': psd,
        'welch_freq': f_welch,
        'welch_psd': psd_welch
    }
```

10. Model Diagnostics

Diagnostic Tests

```
from statsmodels.stats.diagnostic import (
    durbin_watson,
    acorr_breusch_godfrey
)

def model_diagnostics(model, resid):
    """Perform model diagnostics"""

    # Durbin-Watson
    dw = durbin_watson(resid)

    # Breusch-Godfrey
    bg = acorr_breusch_godfrey(
        model,
        nlags=5
    )

    return {
        'dw_stat': dw,
        'bg_stat': bg[0],
        'bg_pval': bg[1]
    }
```