Time Series Analysis in R

Comprehensive Cheat Sheet

1 Data Handling and Exploration

Basic TS Functions

`ts()`	Create time series object
`c(ts_object)`	Convert TS to vector
`frequency(ts)`	Get observations per unit time
`cycle(ts)`	Get position in seasonal cycle
`time(ts)`	Extract time points
`window(ts)`	Extract subset of series
`log(ts)`	Log transformation
`as.ts()`	Convert object to time series

Example

```
# Create time series of monthly data
my_ts <- ts(data, start=c(2020,1), frequency=12)

# Extract subset
subset_ts <- window(my_ts, start=c(2020,7), end=c(2021,6))
```

2 Visualization

Plotting Functions

`plot(ts)`	Basic time series plot
`lines(lowess())`	Add lowess smoother
`matplot()`	Plot matrix columns
`boxplot(ts~cycle)`	Boxplot by season
`layout(matrix())`	Arrange multiple plots
`ts.plot()`	Plot multiple series
`par(mfrow=c(r,c))`	Create plot grid
`pairs()`	Scatter plot matrix for multiple variables

Example

```
# Multiple plots in one figure
par(mfrow=c(2,2))
plot(ts_data)
acf(ts_data)
pacf(ts_data)
cpgram(ts_data)

# Add smoother to plot
plot(ts_data)
lines(lowess(time(ts_data), ts_data, f=0.1), col="red", lwd=2)

# Seasonal plots
# Convert time series to matrix with one column per year
temp_matrix <- matrix(ts_data, nrow=12, byrow=FALSE)
matplot(temp_matrix, type="l")

# Boxplot by month/season
boxplot(ts_data ~ cycle(ts_data))
```

3 Decomposition and Smoothing

Decomposition Functions

`decompose(ts)`	Classical decomposition using moving averages
`stl(ts, s.window)`	STL decomposition (Seasonal-Trend decomposition using Loess), more robust with irregular seasonality
`filter(ts, sides, filter)`	Moving average smoothing to remove noise and highlight trends
`lowess(x, y, f)`	LOWESS smoother for nonparametric trend fitting

Example

```
# Classic decomposition
dcmp <- decompose(ts_data)
plot(dcmp)

# STL decomposition
stl_dcmp <- stl(ts_data, s.window="periodic")
plot(stl_dcmp)

# Moving average (centered, 3-point)
ma3 <- filter(ts_data, sides=2, rep(1,3)/3)

# Weighted moving average
wma <- filter(ts_data, sides=2, c(1,2,1)/4)
```

Moving Average Smoothers (Lab 3)

```
# Simple centered moving average
ma3 = filter(ts_data, sides=2, rep(1,3)/3)

# Weighted moving average
ma2.2 = filter(ts_data, sides=2, c(1,2,1)/4)

# Compare different smoothers
plot(ts_data, type="p")
lines(ma3, col="red")
lines(ma2.2, col="blue")
```

COSINOR Regression (Lab 3)

```
# Time is centered around mean time
wk = time(ts_data) - mean(time(ts_data))
wk2 = wk^2 # Quadratic term
wk3 = wk^3 # Cubic term

# Create Fourier terms
cs = cos(2*pi*wk)
sn = sin(2*pi*wk)

# Polynomial trend model
reg1 = lm(ts_data ~ wk + wk2 + wk3)

# Polynomial trend + seasonal components
reg2 = lm(ts_data ~ wk + wk2 + wk3 + cs + sn)

# Compare models
plot(ts_data)
lines(fitted(reg1), col="blue")
lines(fitted(reg2), col="red")
```

4 Correlation Analysis

Correlation Functions

`lag(ts, k)`	Lag time series by k periods to examine relationships between current and past values
`acf(ts, lag.max)`	Autocorrelation function to identify overall correlation structure and seasonality
`pacf(ts, lag.max)`	Partial autocorrelation function to identify direct correlations (helps determine AR order)
`ccf(ts1, ts2)`	Cross-correlation function to analyze relationship between two time series
`ARMAacf(ar, ma, lag.max)`	Theoretical ACF for ARMA models to compare with empirical patterns
`ARMAacf(ar, ma, lag.max, pacf=TRUE)`	Theoretical PACF for ARMA models

Example

```r
# Plot autocorrelation function
acf(ts_data, lag.max=24)

# Compare empirical vs theoretical ACF
ar_model <- ar(ts_data)
theoretical_acf <- ARMAacf(ar=ar_model$ar, lag.max=20)

# Scatter plots of lagged values
plot(x=lag(ts_data, k=1), y=ts_data)
title(main=paste("r␣=", round(cor(ts_data[-1], ts_data[-length(ts_data)]), digits=3)))
```

5 Spectral Analysis

Spectral Functions

`spectrum(ts)`	Spectral density estimation to identify cyclical components
`cpgram(ts)`	Cumulative periodogram to check for white noise and hidden periodicities
`spec.pgram(ts)`	More control over periodogram calculation with options for smoothing

Example

```r
# Identify frequency components
spec <- spectrum(ts_data)

# Find dominant frequency
omega <- spec$freq[which.max(spec$spec)]
period <- 1/omega

# From Lab 7: Working with frequency components
# Calculate periodogram
spec_pgram <- spec.pgram(ts_data, spans=c(3,5),
                       taper=0.1, log="no")

# Frequency to period conversion
freqs <- spec_pgram$freq
periods <- 1/freqs[freqs > 0]
```

6 Stationarity

Stationarity Functions

`diff(ts)`	First difference to remove trends and achieve stationarity
`diff(ts, lag=s)`	Seasonal difference to remove seasonal components
`Box.test(ts, lag, type)`	Box-Pierce or Ljung-Box test to test for autocorrelation in residuals
`adf.test()`	Augmented Dickey-Fuller test for stationarity (requires `tseries` package)
`kpss.test()`	KPSS test for trend stationarity (requires `tseries` package)

Example

```r
# Remove trend with differencing
ts_diff <- diff(ts_data)
plot(ts_diff)

# Remove seasonality (monthly data)
ts_sdiff <- diff(ts_data, lag=12)

# Seasonal differencing followed by regular differencing
ts_both_diff <- diff(diff(ts_data, lag=12))

# Test for stationarity
library(tseries)
adf_result <- adf.test(ts_data)
kpss_result <- kpss.test(ts_data)
```

7 AR, MA, and ARIMA Modeling

Model Fitting Functions

`ar(ts, method, order.max)`	Fits autoregressive model (often uses Yule-Walker method)
`arima.sim(model, n)`	Simulates ARIMA process for generating data with known properties
`arima0(ts, order=c(p,d,q))`	Fits ARIMA model (faster version) for exploratory model fitting
`arima(ts, order=c(p,d,q))`	Fits ARIMA model with ML estimation for final model selection and inference
`arima(ts, order, seasonal)`	Fits seasonal ARIMA for data with seasonal patterns
`tsdiag(model)`	Diagnostic plots for time series models to check model adequacy
`residuals(model)`	Extracts residuals from a model for diagnostic checking
`AIC(model), BIC(model)`	Information criteria for model comparison and selection
`auto.arima()`	Automatic ARIMA model selection (requires `forecast` package)

Example

```r
# Fit ARIMA(1,1,1) model
model <- arima(ts_data, order=c(1,1,1))
summary(model)

# Seasonal ARIMA
sarima <- arima(ts_data, order=c(1,0,1),
              seasonal=list(order=c(0,1,1), period=12))
```

Simulating ARMA Processes (Lab 4)

```
# Simulate AR(1) process
ar1_sim <- arima.sim(model=list(ar=0.7), n=100)

# Simulate AR(2) process
ar2_sim <- arima.sim(model=list(ar=c(0.7, -0.4)), n=100)

# Simulate ARMA(2,2) process
arma_sim <- arima.sim(model=list(ar=c(0.7, -0.4),
                             ma=c(-0.2, 0.25)),
                    n=100,
                    sd=sqrt(2))

# Simulate with different error distribution
arma_t_sim <- arima.sim(model=list(ar=c(0.7, -0.4),
                             ma=c(-0.2, 0.25)),
                     n=100,
                     rand.gen=function(n,...) {
                       return(sqrt(0.2)*rt(n, df=5))
                     })
```

AR(2) Stationarity Conditions (Lab 4)

For an AR(2) process to be stationary, the parameters ϕ_1 and ϕ_2 must satisfy:

- $\phi_2 > -1$
- $\phi_2 + \phi_1 < 1$
- $\phi_2 - \phi_1 < 1$

```
# Check stationarity by analyzing roots
# For AR(2): (1-B-B)Y = e
# Roots must be outside unit circle
roots <- polyroot(c(1, -0.7, 0.4))
abs(roots) # Should be > 1 for stationarity
```

8 Forecasting

Forecasting Functions

`predict(model, n.ahead)`	Basic forecasting from ARIMA models
`predict(model, newdata)`	Forecasting from regression models with predictors
`forecast()`	Enhanced forecasting with visualizations (requires `forecast` package)

Example

```
# Forecast next 12 periods
fore <- predict(arima_model, n.ahead=12)

# Plot with prediction intervals
ts.plot(ts_data, fore$pred,
      fore$pred + 2*fore$se,
      fore$pred - 2*fore$se,
      gpars=list(col=c(1,2,4,4)))

# Using forecast package
library(forecast)
model_fc <- forecast(arima_model, h=12)
plot(model_fc)
```

9 Regression Models for Time Series

Regression Functions

`lm(y ~trend + seasonal)`	Linear regression with trend and seasonality to model deterministic patterns
`lm(y ~cbind(x1, x2))`	Regression with multiple predictors for models with covariates
`cos(2*pi*time()), sin()`	Fourier terms for modeling seasonality in regression
`summary(aov(model))`	ANOVA table for regression model

Example

```
# Trend with harmonic seasonality
t <- time(ts_data)
s <- sin(2*pi*t)
c <- cos(2*pi*t)
reg <- lm(ts_data ~ t + s + c)

# Polynomial trend terms
t2 <- t^2
t3 <- t^3
reg_poly <- lm(ts_data ~ t + t2 + t3 + s + c)

# Multiple harmonics (from Lab 3)
s1 <- sin(2*pi*t)
c1 <- cos(2*pi*t)
s2 <- sin(4*pi*t) # Second harmonic (twice the frequency)
c2 <- cos(4*pi*t)
reg_harm <- lm(ts_data ~ t + s1 + c1 + s2 + c2)

# ANOVA analysis
summary(aov(reg_harm))
```

10 Multivariate Time Series

Multivariate Functions

`VAR(ts_data, p)`	Vector Autoregression for modeling relationships between multiple time series
`VARselect(ts_data)`	Helps select VAR order to determine optimal lag order in VAR models
`irf(var_model)`	Impulse response function analysis
`fevd(var_model)`	Forecast error variance decomposition
`causality(var_model)`	Granger causality tests

VAR Modeling Example (Lab 7)

```
library(vars)

# Create multivariate time series
mts_data <- ts(cbind(series1, series2), frequency=12)

# Select optimal lag order
lag_selection <- VARselect(mts_data, lag.max=10,
                           type="const")
optimal_p <- lag_selection$selection[["SC"]] # Using Schwarz criterion

# Fit VAR model
var_model <- VAR(mts_data, p=optimal_p, type="const")
summary(var_model)

# Analyze impulse response
irf_result <- irf(var_model, n.ahead=12)
plot(irf_result)

# Forecast error variance decomposition
fevd_result <- fevd(var_model, n.ahead=10)
plot(fevd_result)

# Granger causality
causality(var_model, cause="series1")
```

11 ACF/PACF Pattern Recognition

Model Identification Patterns

Process	ACF	PACF
AR(p)	Tails off gradually	Cuts off after lag p
MA(q)	Cuts off after lag q	Tails off gradually
ARMA(p,q)	Tails off gradually	Tails off gradually
White Noise	No significant spikes	No significant spikes
Seasonal	Spikes at lags s, 2s, 3s	Spikes at lags s, 2s, 3s

12 Model Diagnostic Steps

1. **Plot residuals** - should resemble white noise
2. **ACF of residuals** - no significant autocorrelation
3. **Ljung-Box test** - p-values should exceed significance level
4. **QQ-plot** - check normality assumption

5. **Cumulative periodogram** - should follow diagonal line

Diagnostic Checks (Lab 5)

```
# Fit model
model <- arima(ts_data, order=c(1,0,1))

# Run diagnostics
tsdiag(model)

# Manual diagnostics
par(mfrow=c(2,2))
# Plot residuals
plot(residuals(model), main="Residuals")
# ACF of residuals
acf(residuals(model), main="ACF␣of␣Residuals")
# Ljung-Box test p-values
lb_pvalues <- sapply(1:20, function(i)
  Box.test(residuals(model), lag=i, type="Ljung-Box")$p.value)
plot(lb_pvalues, main="Ljung-Box␣p-values",
     ylab="p-value", xlab="lag")
abline(h=0.05, col="red", lty=2)
# QQ plot
qqnorm(residuals(model))
qqline(residuals(model), col="red")

# Cumulative periodogram
cpgram(residuals(model))
```

13 ARIMA Modeling Workflow

1. **Plot data** - identify patterns and anomalies
2. **Transform if needed** - typically log transformation
3. **Difference until stationary** - regular/seasonal
4. **Examine ACF/PACF** - identify potential orders
5. **Fit candidate models** - try several specifications
6. **Compare using AIC/BIC** - select best model
7. **Check diagnostics** - validate residual behavior
8. **Forecast** - generate predictions with intervals

Model Selection (Lab 5)

```
# Systematic model comparison
models <- list()
aic_values <- matrix(NA, nrow=3, ncol=3)
bic_values <- matrix(NA, nrow=3, ncol=3)

for (p in 0:2) {
  for (q in 0:2) {
    model_name <- paste("ARIMA(", p, ",0,", q, ")", sep="")
    models[[model_name]] <- arima(ts_data, order=c(p,0,q))
    aic_values[p+1, q+1] <- AIC(models[[model_name]])
    bic_values[p+1, q+1] <- BIC(models[[model_name]])
  }
}

# Find best model by AIC
min_aic <- which(aic_values == min(aic_values), arr.ind=TRUE)
best_p_aic <- min_aic[1] - 1
best_q_aic <- min_aic[2] - 1
cat("Best␣model␣by␣AIC:␣ARIMA(", best_p_aic, ",0,", best_q_aic, ")\n", sep="")

# Find best model by BIC
min_bic <- which(bic_values == min(bic_values), arr.ind=TRUE)
best_p_bic <- min_bic[1] - 1
best_q_bic <- min_bic[2] - 1
cat("Best␣model␣by␣BIC:␣ARIMA(", best_p_bic, ",0,", best_q_bic, ")\n", sep="")
```

14 Key Exam Insights

- **Stationarity** is fundamental - constant mean, variance, and autocorrelation
- **Transformation sequence**: log → seasonal differencing → regular differencing
- **Model parsimony**: Simpler models often forecast better
- **Residual analysis**: The true test of any model
- **Seasonal patterns**: Use seasonal ARIMA or Fourier terms
- **Combined approaches**: Regression for deterministic components, ARIMA for residuals
- **Spectral analysis**: Identify hidden periodicities
- **Smoothing techniques**: Extract trends flexibly
- **Multivariate analysis**: Use VAR for interdependent series

15 Practical Tips for Bike Share Analysis

Data Preparation and Model Selection

```
# Step 1: Load data
load("sharedbikes.RData")
load("sharedbikes_forecast.RData")

# Step 2: Linear model for trend/seasonality
mod <- lm(casual ~ t + s1 + c1 + s2 + c2 + s7_1 + c7_1 + s7_2 + c7_2, data=dat)

# Step 3: Examine residuals
dat$res <- ts(residuals(mod), start=2018, frequency=365)
plot(dat$res)
acf(dat$res)
pacf(dat$res)

# Step 4: Test for stationarity with differencing
diff_res <- diff(dat$res)
plot(diff_res)
acf(diff_res)
cpgram(diff_res)

# Step 5: Find best ARIMA model for residuals
# Initialize matrix for AIC values
p_values <- 0:6
q_values <- 0:6
aic_matrix <- matrix(NA, nrow=length(p_values), ncol=length(q_values))
rownames(aic_matrix) <- paste("p =", p_values)
colnames(aic_matrix) <- paste("q =", q_values)

# Fit ARIMA(p,1,q) models and store AIC values
for (i in 1:length(p_values)) {
  for (j in 1:length(q_values)) {
    p <- p_values[i]
    q <- q_values[j]

    tryCatch({
      model <- arima0(dat$res, order=c(p, 1, q))
      aic_matrix[i, j] <- model$aic
    }, error = function(e) {
      aic_matrix[i, j] <- NA
    })
  }
}

# Find the model with the lowest AIC
min_aic <- min(aic_matrix, na.rm=TRUE)
min_indices <- which(aic_matrix == min_aic, arr.ind=TRUE)
best_p <- p_values[min_indices[1]]
best_q <- q_values[min_indices[2]]

# Step 6: Fit best model and check diagnostics
best_model <- arima(dat$res, order=c(best_p, 1, best_q))
tsdiag(best_model)
```

Forecasting and Visualization

```
# Step 7: Generate forecasts
# Predict from linear model
linear_forecast <- predict(mod, newdata=newdat)

# Forecast residuals with ARIMA
arima_forecast <- predict(best_model, n.ahead=60)

# Combine forecasts
total_forecast <- linear_forecast + arima_forecast$pred

# Create confidence intervals
upper_ci <- total_forecast + 1.96 * arima_forecast$se
lower_ci <- total_forecast - 1.96 * arima_forecast$se

# Step 8: Plot forecasts
# Create date sequence for forecast period
forecast_dates <- seq.Date(from=max(dat$dteday) + 1,
                        by="day", length.out=60)

# Plot original data and forecasts
plot(dat$dteday, dat$casual,
     type="l",
     xlim=c(min(dat$dteday), max(forecast_dates)),
     ylim=c(min(c(dat$casual, lower_ci)), max(c(dat$casual, upper_ci))),
     xlab="Date", ylab="Number of Casual Bike Hires",
     main="Casual Bike Rentals with 60-day Forecast")

# Add forecast and prediction intervals
lines(forecast_dates, total_forecast, col="blue", lwd=2)
lines(forecast_dates, upper_ci, col="red", lty=2)
lines(forecast_dates, lower_ci, col="red", lty=2)

# Add legend
legend("topleft",
       legend=c("Observed", "Forecast", "95% Prediction Interval"),
       col=c("black", "blue", "red"),
       lty=c(1, 1, 2),
       lwd=c(1, 2, 1))
```