Time Series Analysis [CheatSheet]

1. Basic Time Series Operations with Pandas

- **Convert to DateTime**: `pd.to_datetime(df['date_column'])`
- **Set DateTime as Index**: `df.set_index('datetime_column', inplace=True)`
- **Resample Time Series Data**: `df.resample('D').mean()`
- **Time-Based Indexing**: `df['2020-01-01':'2020-01-31']`
- **Rolling Window Statistics**: `df.rolling(window=5).mean()`
- **Expanding Window Statistics**: `df.expanding(min_periods=1).mean()`
- **Shifting and Lagging**: `df.shift(1)`
- **Differencing (for Stationarity)**: `df.diff(periods=1)`

2. Visualization of Time Series Data

- **Line Plot**: `df.plot()`
- **Rolling Mean Plot**: `df.rolling(window=12).mean().plot()`
- **Histogram and Density Plot**: `df.hist()`, `df.plot(kind='kde')`
- **Box Plot of Yearly/Monthly Data**: `df.boxplot(groupby=df.index.year)`

3. Seasonal Decomposition and Analysis

- **Seasonal Decomposition with statsmodels**: `from statsmodels.tsa.seasonal import seasonal_decompose; seasonal_decompose(df, model='additive')`
- **Autocorrelation and Partial Autocorrelation Plots**: `from statsmodels.graphics.tsaplots import plot_acf, plot_pacf; plot_acf(df)`

4. Time Series Forecasting Models

- **ARIMA Model with statsmodels**: `from statsmodels.tsa.arima.model import ARIMA; ARIMA(df, order=(5,1,0)).fit()`
- **Seasonal ARIMA (SARIMA)**: `from statsmodels.tsa.statespace.sarimax import SARIMAX; SARIMAX(df, order=(1, 1, 1), seasonal_order=(1, 1, 1, 12)).fit()`

By: Waleed Mousa

- **Simple Exponential Smoothing**: `from statsmodels.tsa.holtwinters import SimpleExpSmoothing; SimpleExpSmoothing(df).fit()`
- **Holt-Winters' Seasonal Method**: `from statsmodels.tsa.holtwinters import ExponentialSmoothing; ExponentialSmoothing(df, seasonal='add', seasonal_periods=12).fit()`

5. Stationarity Tests and Transformation

- **ADF Test (Augmented Dickey-Fuller Test)**: `from statsmodels.tsa.stattools import adfuller; adfuller(df)`
- **KPSS Test (Kwiatkowski-Phillips-Schmidt-Shin)**: `from statsmodels.tsa.stattools import kpss; kpss(df)`
- **Box-Cox Transformation**: `from scipy.stats import boxcox; df_transformed, lam = boxcox(df)`

6. Time Series Cross-Validation

- **Time Series Split with sklearn**: `from sklearn.model_selection import TimeSeriesSplit; TimeSeriesSplit(n_splits=5)`

7. Error Metrics for Forecasting

- **Mean Absolute Error (MAE)**: `from sklearn.metrics import mean_absolute_error; mean_absolute_error(true_values, predictions)`
- **Mean Squared Error (MSE)**: `from sklearn.metrics import mean_squared_error; mean_squared_error(true_values, predictions)`
- **Root Mean Squared Error (RMSE)**: `np.sqrt(mean_squared_error(true_values, predictions))`

8. Handling Multivariate Time Series

- **Vector Autoregression (VAR)**: `from statsmodels.tsa.api import VAR; model = VAR(multivariate_df).fit()`
- **Granger Causality Tests**: `from statsmodels.tsa.stattools import grangercausalitytests; grangercausalitytests(df, maxlag=3)`

9. Time Series Clustering

- **K-Means Clustering on Time Series**: `from tslearn.clustering import TimeSeriesKMeans; TimeSeriesKMeans(n_clusters=3).fit(df)`

10. Time Series Anomaly Detection

- **Anomaly Detection with Facebook's Prophet**: `from fbprophet import Prophet; m = Prophet(); m.fit(df); forecast = m.predict(future); m.plot(forecast)`
- **Isolation Forest for Anomaly Detection**: `from sklearn.ensemble import IsolationForest; IsolationForest().fit(df)`

11. Time Series Data Preparation

- **Filling Missing Values**: `df.fillna(method='ffill')`
- **Time Series Feature Engineering**: `df['month'] = df.index.month`

12. Frequency and Period Conversion

- **Change Frequency of Time Series**: `df.asfreq('M')`
- **Converting to Periods**: `df.to_period('M')`

13. Multivariate Time Series Forecasting

- **Vector Autoregressive Moving-Average (VARMA)**: `from statsmodels.tsa.statespace.varmax import VARMAX; VARMAX(df, order=(1, 1)).fit()`
- **Cointegration Test**: `from statsmodels.tsa.vector_ar.vecm import coint_johansen; coint_johansen(df, det_order=0, k_ar_diff=1)`

14. Time Series Simulation

- **Simulate AR Process**: `from statsmodels.tsa.arima_process import ArmaProcess; ar = np.array([1, -0.9]); ma = np.array([1]); ArmaProcess(ar, ma).generate_sample(nsample=100)`
- **Simulate MA Process**: `from statsmodels.tsa.arima_process import ArmaProcess; ar = np.array([1]); ma = np.array([1, 0.9]); ArmaProcess(ar, ma).generate_sample(nsample=100)`

15. Advanced Time Series Analysis Techniques

- **Wavelet Transform for Time Series**: `import pywt; coeffs = pywt.wavedec(df, 'haar')`

- **Dynamic Time Warping (DTW)**: `from fastdtw import fastdtw; distance, path = fastdtw(ts1, ts2)`

16. Time Series Decomposition and Prediction

- **LOESS Smoothing (STL Decomposition)**: `from statsmodels.tsa.seasonal import STL; STL(df).fit()`
- **Recurrent Neural Network (RNN) with TensorFlow/Keras**: `from tensorflow.keras.models import Sequential; from tensorflow.keras.layers import SimpleRNN; model = Sequential([SimpleRNN(50, return_sequences=True, input_shape=(n_input, n_features)), ...])`

17. Fourier Transform for Periodicity

- **Fourier Analysis to Identify Cyclical Patterns**: `np.fft.fft(df)`

18. Time Series Data Mining

- **Time Series Segmentation**: `from tslearn.preprocessing import TimeSeriesScalerMeanVariance; TimeSeriesScalerMeanVariance().fit_transform(df)`

19. Integration with Machine Learning

- **Random Forest for Time Series Prediction**: `from sklearn.ensemble import RandomForestRegressor; RandomForestRegressor().fit(X_train, y_train)`

20. Advanced Forecasting Techniques

- **Long Short-Term Memory (LSTM) Network**: `from tensorflow.keras.models import Sequential; from tensorflow.keras.layers import LSTM; model = Sequential([LSTM(50, return_sequences=True, input_shape=(n_input, n_features)), ...])`
- **Prophet for Univariate Time Series**: `from fbprophet import Prophet; m = Prophet(); m.fit(df); future = m.make_future_dataframe(periods=365); forecast = m.predict(future)`

21. Time Series Regression Models

- **Time Series Linear Regression**: `from sklearn.linear_model import LinearRegression; LinearRegression().fit(X_train, y_train)`
- **Lasso and Ridge Regression for Time Series**: `from sklearn.linear_model import Lasso, Ridge; Lasso(alpha=0.1).fit(X_train, y_train)`

22. Integrating Time Series with Panel Data

- **Fixed and Random Effects Models with Panel Data**: `import statsmodels.api as sm; from statsmodels.regression.mixed_linear_model import MixedLM; MixedLM.from_formula('y ~ x', groups='group', data=panel_data).fit()`

23. Advanced Statistical Analysis for Time Series

- **Vector Autoregressive Fractionally Integrated Moving Average (VARFIMA)**: `from statsmodels.tsa.statespace.varmax import VARMAX; VARMAX(df, order=(1, 1), trend='c').fit(disp=False)`
- **Dynamic Factor Models**: `from statsmodels.tsa.statespace.dynamic_factor import DynamicFactor; DynamicFactor(df, k_factors=1, factor_order=2).fit()`

24. Non-Linear Time Series Models

- **Generalized Autoregressive Conditional Heteroskedasticity (GARCH)**: `from arch import arch_model; arch_model(df, vol='Garch', p=1, o=0, q=1).fit()`
- **Nonlinear Autoregressive (NAR) Model**: `from statsmodels.tsa.nonlinear.tsa import NAR; NAR(df, lag=1).fit()`

25. Incorporating External Variables in Time Series

- **VAR with Exogenous Variables (VARX)**: `from statsmodels.tsa.api import VARMAX; VARMAX(df, exog=exog_data, order=(1,1)).fit()`
- **SARIMAX Model with Exogenous Regressors**: `from statsmodels.tsa.statespace.sarimax import SARIMAX; SARIMAX(df, order=(1, 1, 1), seasonal_order=(1, 1, 1, 12), exog=exog_data).fit()`

26. Real-Time Time Series Analysis

- **Streaming Data Analysis**: `from streamz import Stream; stream = Stream(); stream.map(func)`
- **Online Learning for Time Series**: `from sklearn.linear_model import SGDRegressor; SGDRegressor().partial_fit(X_train, y_train)`

27. Time Series in Finance

- **Efficient Frontier Analysis**: `from pypfopt.efficient_frontier import EfficientFrontier; EfficientFrontier(mean_returns, cov_matrix)`
- **Value at Risk (VaR) Calculation**: `from scipy.stats import norm; VaR = norm.ppf(1-confidence_level, mean, std)`

28. Hybrid Models for Time Series

- **Combining ARIMA with Machine Learning**: `from sklearn.ensemble import RandomForestRegressor; RandomForestRegressor().fit(ARIMA_residuals, y_train)`
- **Wavelet-Based Hybrid Forecasting Models**: `import pywt; coeffs = pywt.wavedec(df, 'haar'); model.fit(coeffs)`

29. Scalable Time Series Analysis

- **Distributed Time Series with Dask**: `import dask.dataframe as dd; ddf = dd.from_pandas(df, npartitions=10)`
- **Using Spark for Large Scale Time Series**: `from pyspark.sql.functions import window; df.groupBy(window('timestamp', '1 day')).mean()`

30. Advanced Visualization Techniques

- **Interactive Time Series Plotting with Plotly**: `import plotly.express as px; px.line(df, x='date', y='value')`
- **Heatmap of Time Series Correlations**: `sns.heatmap(df.corr())`

31. Deep Learning for Time Series

- **Convolutional Neural Networks (CNN) for Time Series**: `from tensorflow.keras.layers import Conv1D; Conv1D(filters=64, kernel_size=2, activation='relu')`
- **Attention Mechanisms in Time Series (e.g., Transformer models)**: `from tensorflow.keras.layers import MultiHeadAttention; MultiHeadAttention(num_heads=2, key_dim=2)`

32. Time Series Benchmarking and Evaluation

- **Model Cross-Validation**: `from sklearn.model_selection import cross_val_score; cross_val_score(model, X, y, cv=TimeSeriesSplit())`
- **Time Series Model Selection**: `from sktime.forecasting.model_selection import ForecastingGridSearchCV; ForecastingGridSearchCV(model, param_grid)`

33. Custom Time Series Model Development

- **Developing Custom Forecasting Models**: `class CustomModel(): def fit(self, X, y): ...; def predict(self, X): ...`
- **Implementing Custom Loss Functions**: `def custom_loss(y_true, y_pred): return keras.backend.mean(keras.backend.square(y_pred - y_true))`

34. Time Series Anomaly/Change Point Detection

- **Changepoint Detection with Ruptures**: `import ruptures as rpt; rpt.Pelt().fit_predict(df, pen=10)`
- **Anomaly Detection with LSTM Autoencoders**: `from tensorflow.keras.layers import LSTM; model = Sequential([LSTM(32, activation='relu', input_shape=(timesteps, n_features), return_sequences=True), ...])`

35. Time Series Data Augmentation

- **Jittering for Data Augmentation**: `df_augmented = df + np.random.normal(0, 0.1, size=df.shape)`
- **Time Warping for Augmentation**: `TimeSeriesScalerMeanVariance(mu=0., std=0.1).fit_transform(df)`

36. Domain-Specific Time Series Analysis

- **Time Series for IoT Sensor Data**: `IoT_data.resample('15T').mean()`
- **Time Series in Retail and Sales Forecasting**: `sales_data.groupby('Product').resample('W').sum()`

37. Time Series Data Preprocessing

- **Normalization/Standardization of Time Series**: `from sklearn.preprocessing import StandardScaler; StandardScaler().fit_transform(df)`
- **Handling Missing Values in Time Series**: `df.interpolate(method='time')`

38. Time Series Feature Extraction

- **Time Series Feature Engineering with tsfresh**: `from tsfresh import extract_features; features = extract_features(df, column_id='id', column_sort='time')`
- **Lag Features for Time Series**: `df['lag_1'] = df['value'].shift(1)`

39. Advanced Statistical Methods in Time Series

- **Copula Models in Time Series**: `from copulas.multivariate import GaussianMultivariate; GaussianMultivariate().fit(df)`
- **Survival Analysis for Time Series**: `from lifelines import CoxPHFitter; CoxPHFitter().fit(df, 'duration', event_col='event')`

40. Time Series in Healthcare and Biostatistics

- **ECG Signal Analysis**: `import neurokit2 as nk; nk.ecg_process(ecg_signal, sampling_rate=1000)`
- **Time Series Analysis in Genomics Data**: `genomic_data.resample('1D').apply(custom_genomic_processing)`

41. Multidimensional Time Series Analysis

- **Multivariate Time Series Classification with tslearn**: `from tslearn.shapelets import ShapeletModel; ShapeletModel(n_shapelets_per_size={2: 4}).fit(X_train, y_train)`

- **Dynamic Time Warping for Multidimensional Data**: `from tslearn.metrics import dtw; dtw(ts1, ts2, global_constraint='sakoe_chiba')`

42. Forecasting with External Regressors

- **Incorporating External Factors in Forecasting**: `model = SARIMAX(endog=df['target'], exog=df[['exog1', 'exog2']], order=(1,1,1)).fit()`
- **Weather Data Integration in Energy Forecasting**: `energy_data.join(weather_data).fillna(method='ffill')`

43. Time Series Simulation and Synthetic Data

- **Simulating Synthetic Time Series Data**: `from statsmodels.tsa.arima_process import arma_generate_sample; arma_generate_sample(ar=[1, -0.5], ma=[1, 0.5], nsample=100)`
- **Monte Carlo Simulation for Forecasting**: `mc_forecasts = [model.simulate(params, steps=10) for _ in range(1000)]`

44. Time Series in Natural Language Processing

- **Sentiment Analysis Over Time**: `sentiment_df.resample('W').mean()`
- **Time Series Analysis of Document Frequencies**: `document_frequency_data.groupby('topic').resample('M').count()`

45. Integrating with Other Data Types

- **Combining Time Series with Spatial Data**: `spatial_ts_data.groupby(['region', pd.Grouper(key='time', freq='M')]).mean()`
- **Time Series and Graph Data Integration**: `graph_data.apply(lambda x: ts_model.predict(x.time_series))`

46. Time Series in Environmental and Earth Sciences

- **Climate Data Time Series Analysis**: `climate_data.groupby(pd.Grouper(freq='Y')).mean()`
- **Seismology and Earthquake Trend Analysis**: `earthquake_data.resample('1D').count()`

47. Advanced Time Series Analysis in Python

- **PyFlux for Bayesian Time Series**: `import pyflux as pf; model = pf.ARIMA(data=df, ar=1, ma=1, family=pf.Normal()).fit()`
- **GPy for Gaussian Processes in Time Series**: `import GPy; kernel = GPy.kern.RBF(input_dim=1); GPy.models.GPRegression(X, y, kernel)`