Data Wrangling

1. **Setup**

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

2. **Loading Data**

- **CSV File**

```
df = pd.read_csv('file.csv')
```

- **Excel File**

```
df = pd.read_excel('file.xlsx')
```

- **SQL Database**

```
from sqlalchemy import create_engine
engine = create_engine('sqlite:///database.db')
df = pd.read_sql('SELECT * FROM table', engine)
```

3. **Basic Data Exploration**

- **View DataFrame**

```
df.head()         # First 5 rows
df.tail()         # Last 5 rows
df.info()         # Summary of DataFrame
df.describe()     # Statistical summary
```

- **Check for Missing Values**

```
df.isnull().sum() # Count of missing values per column
```

4. **Data Selection and Filtering**

- **Select Columns**

```
df['column_name']          # Single column
df[['col1', 'col2']]       # Multiple columns
```

- **Select Rows by Index**

```
df.iloc[0]                 # First row
df.iloc[1:5]               # Rows from index 1 to 4
```

- **Select Rows by Condition**

```
df[df['column_name'] > value]       # Rows where column_name > value
df[df['column_name'].str.contains('text')]  # Rows where column_name
contains 'text'
```

5. Data Cleaning

- **Drop Missing Values**

```
df.dropna()                # Drop any rows with missing values
df.dropna(subset=['col']) # Drop rows where 'col' has missing values
```

- **Fill Missing Values**

```
df.fillna(value)           # Fill missing values with a specified value
df.fillna(method='ffill')  # Forward fill
df.fillna(method='bfill')  # Backward fill
```

- **Drop Columns/Rows**

```
df.drop(columns=['col1', 'col2'])  # Drop columns
df.drop(index=[0, 1])              # Drop rows by index
```

- **Rename Columns**

```
df.rename(columns={'old_name': 'new_name'}, inplace=True)
```

6. Data Transformation

- **Add New Column**

```
df['new_col'] = df['col1'] + df['col2']
```

- **Apply Functions**

```
df['col'] = df['col'].apply(lambda x: x + 1)  # Apply function to each
element
```

- **Group By and Aggregation**

```
df.groupby('col').sum()          # Sum of groups
df.groupby('col').agg({'col1': 'mean', 'col2': 'sum'})
```

- **Pivot Table**

```
pd.pivot_table(df, values='value_col', index='index_col',
columns='columns_col')
```

7. Handling Duplicates

- **Check for Duplicates**

```
df.duplicated()                # Boolean series of duplicate rows
df[df.duplicated()]            # Duplicate rows
```

- **Drop Duplicates**

```
df.drop_duplicates()           # Drop duplicate rows
```

8. Merging and Joining

- **Concatenate DataFrames**

```
pd.concat([df1, df2])         # Concatenate along rows
pd.concat([df1, df2], axis=1) # Concatenate along columns
```

- **Merge DataFrames**

```
pd.merge(df1, df2, on='key')        # Merge on key column
pd.merge(df1, df2, how='left', on='key')  # Left join
pd.merge(df1, df2, how='right', on='key') # Right join
pd.merge(df1, df2, how='inner', on='key') # Inner join
pd.merge(df1, df2, how='outer', on='key') # Outer join
```

9. **Reshaping Data**

- **Melt DataFrame**

```
pd.melt(df, id_vars=['id_col'], value_vars=['var1', 'var2'])
```

- **Pivot DataFrame**

```
df.pivot(index='index_col', columns='columns_col', values='values_col')
```

10. **Date and Time Handling**

- **Convert to DateTime**

```
df['date_col'] = pd.to_datetime(df['date_col'])
```

- **Extract Date Components**

```
df['year'] = df['date_col'].dt.year
df['month'] = df['date_col'].dt.month
df['day'] = df['date_col'].dt.day
```

- **Date Difference**

```
df['date_diff'] = df['date_col'] - pd.Timestamp('2024-01-01')
```

11. **Saving Data**

- **To CSV**

```
df.to_csv('file.csv', index=False)
```

- **To Excel**

```
df.to_excel('file.xlsx', index=False)
```

- **To SQL**

```
df.to_sql('table_name', engine, if_exists='replace', index=False)
```

Certainly! Here's a more advanced data wrangling cheat sheet, covering additional techniques and operations for handling complex data transformations, performance optimization, and advanced manipulation.

Advanced Data Wrangling Cheat Sheet

1. Advanced Data Manipulation

- **Transform Columns Using Lambda Functions**

```
df['new_col'] = df['col'].transform(lambda x: x**2)  # Apply function to
each group
```

- **Apply Functions Across Rows**

```
df.apply(lambda row: row['col1'] + row['col2'], axis=1)  # Row-wise
operation
```

- **Vectorized Operations with NumPy**

```
df['log_col'] = np.log(df['numeric_col'] + 1)  # Vectorized log
transformation
```

2. Advanced Aggregations and Grouping

- **Custom Aggregation Functions**

```
def custom_agg(series):
    return series.sum() - series.mean()

df.groupby('group_col').agg(custom_agg)
```

- **Multiple Aggregations**

```
df.groupby('group_col').agg({'col1': ['mean', 'std'], 'col2': 'sum'})
```

- **Expanding and Rolling Windows**

```
df['rolling_mean'] = df['col'].rolling(window=3).mean()  # Rolling mean with
window size of 3
df['expanding_sum'] = df['col'].expanding().sum()  # Expanding sum
```

3. Handling Time Series Data

- **Resampling Time Series Data**

```
df.set_index('date_col', inplace=True)
df.resample('M').sum()  # Resample by month and sum values
```

- **Time Shifts**

```
df['shifted_col'] = df['col'].shift(periods=1)  # Shift values by one period
```

- **Time-Based Indexing**

```
df.loc['2024-01-01':'2024-01-31']  # Select data for January 2024
```

4. Advanced Merging and Joining

- **Merging on Multiple Keys**

```
df1.merge(df2, left_on=['key1', 'key2'], right_on=['key1', 'key2'],
how='outer')
```

- **Joining with Index**

```
df1.join(df2, how='left')  # Join on index
```

- **Concatenating with Hierarchical Indexing**

```
pd.concat([df1, df2], keys=['df1', 'df2'])
```

5. Efficient Data Handling

- **Using Dask for Large Datasets**

```
import dask.dataframe as dd
df = dd.read_csv('large_file.csv')
df.compute()  # Convert Dask DataFrame to Pandas DataFrame
```

- **Memory Optimization**

```
df['col'] = df['col'].astype('category')  # Convert to categorical to save
memory
```

- **Chunk Processing**

```
chunks = pd.read_csv('large_file.csv', chunksize=10000)
for chunk in chunks:
    process(chunk)
```

6. Data Transformation and Feature Engineering

- **Polynomial Features**

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2)
df_poly = poly.fit_transform(df[['col1', 'col2']])
```

- **One-Hot Encoding**

```
pd.get_dummies(df, columns=['categorical_col'])
```

- **Label Encoding**

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['encoded_col'] = le.fit_transform(df['categorical_col'])
```

7. Advanced String Operations

- **Regular Expressions**

```
df['extracted'] = df['text_col'].str.extract(r'(\d+)')  # Extract numbers
from text
```

- **String Operations with Regex**

```
df['cleaned'] = df['text_col'].str.replace(r'\W', '')  # Remove non-
alphanumeric characters
```

8. Data Validation and Quality

- **Validation Functions**

```
def validate_row(row):
    return row['col'] > 0  # Custom validation logic

df['is_valid'] = df.apply(validate_row, axis=1)
```

- **Cross-Validation of Data**

```
df['duplicated'] = df.duplicated(subset=['col1', 'col2'], keep=False)  #
Flag duplicates
```

9. Data Export and Visualization

- **Export to JSON**

```
df.to_json('file.json')
```

- **Basic Plotting with Pandas**

```
df.plot(kind='line', x='x_col', y='y_col')  # Simple line plot
```

- **Interactive Visualization with Plotly**

```
import plotly.express as px
fig = px.scatter(df, x='x_col', y='y_col')
fig.show()
```

This advanced cheat sheet covers a range of techniques to handle more complex data wrangling tasks, optimize performance, and enhance your data analysis capabilities. Feel free to expand on these techniques based on the specific requirements of your data and analysis.

Certainly! Here's an even more advanced cheat sheet focusing on specialized techniques and libraries for data wrangling, integration with big data ecosystems, and handling complex data structures.

Specialized Advanced Data Wrangling Cheat Sheet

1. Advanced Data Manipulation

- **Efficient String Operations with `str` Methods**

```
df['clean_text'] = df['text_col'].str.lower().str.strip()  # Clean and
preprocess text data
```

- **Handling Nested JSON Data**

```
df = pd.json_normalize(data, sep='_')  # Flatten JSON data into DataFrame
```

- **Transformations with `applymap`**

```
df.applymap(lambda x: x**2 if isinstance(x, (int, float)) else x)  # Apply
function to each element
```

2. Handling Big Data

- **Using `Vaex` for Large Datasets**

```
import vaex
df = vaex.open('large_file.hdf5')  # Open large file
df.describe()  # Fast, out-of-core operations
```

- **Using `Dask` for Parallel Computing**

```
import dask.dataframe as dd
ddf = dd.read_csv('large_file.csv')
```

```
ddf = ddf.groupby('col').agg({'value': 'mean'})
result = ddf.compute()  # Execute computation
```

3. Advanced Data Integration

- **Connecting with APIs**

```
import requests
response = requests.get('https://api.example.com/data')
data = response.json()
df = pd.json_normalize(data)
```

- **Streaming Data with `Kafka`**

```
from kafka import KafkaConsumer
consumer = KafkaConsumer('topic_name', bootstrap_servers='localhost:9092')
for message in consumer:
    process(message.value)
```

- **Integrating with `Spark`**

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('app_name').getOrCreate()
df_spark = spark.read.csv('large_file.csv', header=True, inferSchema=True)
df_spark.createOrReplaceTempView('table_name')
```

4. Advanced Feature Engineering

- **Using `FeatureTools` for Automated Feature Engineering**

```
import featuretools as ft
es = ft.EntitySet(id='data')
es = es.add_dataframe(dataframe_name='df', dataframe=df, index='index')
feature_matrix, feature_defs = ft.dfs(entityset=es,
target_dataframe_name='df')
```

- **Generating Polynomial and Interaction Features with `sklearn`**

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=3, interaction_only=True,
include_bias=False)
X_poly = poly.fit_transform(df[['feature1', 'feature2']])
```

5. Data Quality and Validation

- **Schema Validation with `pandera`**

```
import pandera as pa

class Schema(pa.SchemaModel):
    col1: pa.Field(gt=0)
    col2: pa.Field(str)

schema = Schema.validate(df)
```

- **Validation with Custom Functions**

```
def validate_date_format(date_str):
    try:
        pd.to_datetime(date_str, format='%Y-%m-%d')
        return True
    except ValueError:
        return False

df['valid_date'] = df['date_col'].apply(validate_date_format)
```

6. Time Series Analysis

- **Time Series Forecasting with `Prophet`**

```
from fbprophet import Prophet

df_prophet = df.rename(columns={'date_col': 'ds', 'value_col': 'y'})
model = Prophet()
model.fit(df_prophet)
future = model.make_future_dataframe(periods=365)
forecast = model.predict(future)
```

- **Seasonal Decomposition with `statsmodels`**

```
from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(df['value'], model='multiplicative', period=365)
result.plot()
```

7. Data Cleaning and Imputation

- **Advanced Imputation with `KNN`**

```
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=5)
df_imputed = imputer.fit_transform(df)
```

- **Imputation with Multiple Imputation**

```
from sklearn.impute import IterativeImputer
imputer = IterativeImputer()
df_imputed = imputer.fit_transform(df)
```

- **Handling Outliers with `IQR`**

```
Q1 = df['col'].quantile(0.25)
Q3 = df['col'].quantile(0.75)
IQR = Q3 - Q1
df_no_outliers = df[~((df['col'] < (Q1 - 1.5 * IQR)) | (df['col'] > (Q3 +
1.5 * IQR)))]
```

8. Data Export and Sharing

- **Export to Parquet for Efficiency**

```
df.to_parquet('data.parquet', compression='gzip')
```

- **Exporting to SQL Databases**

```
from sqlalchemy import create_engine
engine = create_engine('sqlite:///database.db')
df.to_sql('table_name', con=engine, if_exists='replace', index=False)
```

- **Exporting to Excel with Multiple Sheets**

```
with pd.ExcelWriter('data.xlsx') as writer:
    df1.to_excel(writer, sheet_name='Sheet1')
    df2.to_excel(writer, sheet_name='Sheet2')
```

9. Advanced String Manipulation

- **Using `regex` for Complex Pattern Matching**

```
import re
df['phone'] = df['text_col'].str.extract(r'\(?(\d{3})\)?[-.\s]?(\d{3})
[-.\s]?(\d{4})')
```

- **Text Vectorization with `TfidfVectorizer`**

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(max_features=1000)
X = vectorizer.fit_transform(df['text_col'])
```

10. Advanced Aggregation and Group Operations

- **Hierarchical Aggregation**

```
df.groupby(['level1', 'level2']).agg({'value': ['mean', 'std', 'max']})
```

- **Custom Group-wise Operations**

```
def custom_function(group):
    return group.apply(lambda x: x['value'] * 2 if x['condition'] else
x['value'])

df.groupby('group_col').apply(custom_function)
```

These advanced techniques and tools will help you handle complex data wrangling scenarios, improve efficiency, and integrate with various data ecosystems. Whether working with large datasets, integrating data from different sources, or performing sophisticated analyses, these methods should provide powerful solutions.