Data Science Cheatsheet

Essential workflow, techniques, and tools for comprehensive data science projects

This cheatsheet provides a quick reference to fundamental data science concepts, methodologies, and Python libraries, ideal for both beginners starting their data science journey and experienced professionals seeking efficient workflow optimization.

Data Collection
Gather and import data from sources

Data Cleaning
Prepare and transform raw data

Data Analysis
Explore patterns and statistics

Machine Learning
Build predictive models

Data Visualization
Communicate insights effectively

Essential Python Libraries

Core Data Science Stack

Key libraries like NumPy, Pandas, Matplotlib, Seaborn, and scikit-learn form the foundation of data science workflows.

```
# Essential imports for data science
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score,
classification_report
```

NumPy: `import numpy as np`

Fundamental package for numerical computing with Python.

```
# Create arrays
arr = np.array([1, 2, 3, 4, 5])
matrix = np.array([[1, 2], [3, 4]])
# Basic operations
np.mean(arr)      # Average
np.std(arr)       # Standard deviation
np.reshape(arr, (5, 1))  # Reshape array
# Generate data
np.random.normal(0, 1, 100)  # Random normal
distribution
```

Pandas: `import pandas as pd`

Data manipulation and analysis library.

```
# Create DataFrame
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
# Read data
df = pd.read_csv('data.csv')
# Basic exploration
df.head()        # First 5 rows
df.info()        # Data types and missing values
df.describe()    # Summary statistics
# Data manipulation
df.groupby('column').mean()
df.fillna(df.mean())  # Handle missing values
```

Matplotlib & Seaborn: Visualization

Create statistical visualizations and plots.

```
# Matplotlib basics
plt.plot(x, y)
plt.hist(data, bins=20)
plt.scatter(x, y)
plt.show()

# Seaborn for statistical plots
sns.boxplot(data=df, x='category', y='value')
sns.heatmap(df.corr(), annot=True)
sns.pairplot(df)
```

Data Science Workflow

The workflow is prominently presented showing where various Python packages are used throughout the data science process.

01

1. Problem Definition

Data science is a multi-disciplinary field, combining mathematics, statistics, programming, and business intelligence. Define objectives and success metrics.

```
# Define business problem
# - What question are we answering?
# - What metrics will measure
success?
# - What data do we need?
```

02

2. Data Collection & Import

Gather data from various sources and formats.

```
# Multiple data sources
df_csv = pd.read_csv('data.csv')
df_json = pd.read_json('data.json')
df_sql = pd.read_sql('SELECT * FROM
table', connection)
# APIs and web scraping
import requests
response =
requests.get('https://api.example.co
m/data')
```

03

3. Data Exploration (EDA)

Understand data structure, patterns, and quality.

```
# Exploratory Data Analysis
df.shape             # Dimensions
df.dtypes            # Data types
df.isnull().sum()    # Missing values
df['column'].value_counts()  #
Frequency counts
df.corr()            # Correlation matrix

# Visualizations for EDA
sns.histplot(df['numeric_column'])
sns.boxplot(data=df,
y='numeric_column')
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True)
```

Data Cleaning & Preprocessing

Handling Missing Data

Before analyzing data, it must be cleaned and prepared. This includes handling missing data, removing duplicates, and normalizing variables. Data cleaning is often the most time-consuming yet critical aspect of the data science process.

```
# Identify missing values
df.isnull().sum()
df.isnull().sum() / len(df) * 100  # Percentage missing

# Handle missing values
df.dropna()                  # Remove rows with NaN
df.fillna(df.mean())         # Fill with mean
df.fillna(method='forward')  # Forward fill
df.fillna(method='backward') # Backward fill

# Advanced imputation
from sklearn.impute import SimpleImputer, KNNImputer
imputer = SimpleImputer(strategy='median')
df_filled = pd.DataFrame(imputer.fit_transform(df))
```

Data Transformation

Data normalization (scaling data to a standard range like [0, 1]) helps avoid biases due to differences in feature magnitude.

```
# Scaling and normalization
from sklearn.preprocessing import StandardScaler,
MinMaxScaler
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df[numeric_columns])

# Min-Max scaling to [0,1]
minmax = MinMaxScaler()
df_normalized =
minmax.fit_transform(df[numeric_columns])

# Encoding categorical variables
pd.get_dummies(df, columns=['category_column'])
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['encoded'] = le.fit_transform(df['category'])
```

Outlier Detection & Treatment

Identify and handle extreme values that may skew analysis.

```
# Statistical outlier detection
Q1 = df['column'].quantile(0.25)
Q3 = df['column'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers
df_clean = df[(df['column'] >= lower_bound) &
        (df['column'] <= upper_bound)]

# Z-score method
from scipy import stats
z_scores = np.abs(stats.zscore(df['column']))
df_no_outliers = df[z_scores < 3]
```

Feature Engineering

Create new variables to improve model performance.

```
# Create new features
df['feature_ratio'] = df['feature1'] / df['feature2']
df['feature_sum'] = df['feature1'] + df['feature2']

# Date/time features
df['date'] = pd.to_datetime(df['date'])
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month
df['day_of_week'] = df['date'].dt.day_name()

# Binning continuous variables
df['age_group'] = pd.cut(df['age'], bins=[0, 18, 35, 50, 100],
               labels=['Child', 'Young Adult', 'Adult',
'Senior'])
```

Statistical Analysis

Descriptive Statistics

These measures of central tendency summarize data and provide insight into its distribution. They are foundational for understanding any dataset. Mean is the average of all values in a dataset. It's highly sensitive to outliers.

```
# Central tendency
mean = df['column'].mean()
median = df['column'].median()
mode = df['column'].mode()[0]

# Variability measures
std_dev = df['column'].std()
variance = df['column'].var()
range_val = df['column'].max() - df['column'].min()

# Distribution shape
skewness = df['column'].skew()
kurtosis = df['column'].kurtosis()

# Percentiles
percentiles = df['column'].quantile([0.25, 0.5, 0.75, 0.95])
```

Hypothesis Testing

Test statistical hypotheses and validate assumptions.

```
# T-test for comparing means
from scipy.stats import ttest_ind, ttest_1samp
# One-sample t-test
t_stat, p_value = ttest_1samp(data, population_mean)

# Two-sample t-test
group1 = df[df['group'] == 'A']['value']
group2 = df[df['group'] == 'B']['value']
t_stat, p_value = ttest_ind(group1, group2)

# Chi-square test for independence
from scipy.stats import chi2_contingency
chi2, p_value, dof, expected =
chi2_contingency(contingency_table)
```

Correlation Analysis

Understand relationships between variables.

```
# Correlation matrix
correlation_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True,
cmap='coolwarm')

# Specific correlations
pearson_corr = df['var1'].corr(df['var2'])
spearman_corr = df['var1'].corr(df['var2'],
method='spearman')

# Statistical significance of correlation
from scipy.stats import pearsonr
correlation, p_value = pearsonr(df['var1'], df['var2'])
```

ANOVA & Regression

Analyze variance and relationships between variables.

```
# One-way ANOVA
from scipy.stats import f_oneway
group_data = [df[df['group'] == g]['value'] for g in
df['group'].unique()]
f_stat, p_value = f_oneway(*group_data)

# Linear regression analysis
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
X = df[['feature1', 'feature2']]
y = df['target']
model = LinearRegression().fit(X, y)
y_pred = model.predict(X)
r2 = r2_score(y, y_pred)
```

Machine Learning Models

No data science cheatsheet is complete without an understanding of machine learning models and their applications.

Supervised Learning - Classification

Decision Trees: A tree-like model of decisions and their possible consequences. Each node represents a test on an attribute, and each branch represents the outcome. It's commonly used for classification tasks.

```
# Train-test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Logistic Regression
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
y_pred = log_reg.predict(X_test)

# Decision Tree
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth=5)
dt.fit(X_train, y_train)

# Random Forest
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train, y_train)
```

Supervised Learning - Regression

Predict continuous target variables.

```
# Linear Regression
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)

# Polynomial Regression
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)

# Ridge & Lasso Regression
from sklearn.linear_model import Ridge, Lasso
ridge = Ridge(alpha=1.0)
lasso = Lasso(alpha=0.1)
ridge.fit(X_train, y_train)
lasso.fit(X_train, y_train)
```

Unsupervised Learning

Discover patterns in data without labeled outcomes.

```
# K-Means Clustering
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(X)
df['cluster'] = clusters

# Principal Component Analysis (PCA)
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Hierarchical Clustering
from scipy.cluster.hierarchy import dendrogram, linkage
linkage_matrix = linkage(X_scaled, method='ward')
dendrogram(linkage_matrix)
```

Model Evaluation

Assess model performance using appropriate metrics.

```
# Classification metrics
from sklearn.metrics import accuracy_score,
precision_score, recall_score, f1_score, confusion_matrix
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred,
average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d')

# Regression metrics
from sklearn.metrics import mean_squared_error,
mean_absolute_error
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
```

Data Visualization

Exploratory Visualizations

Understand data distributions and relationships.

```
# Distribution plots
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 1)
plt.hist(df['numeric_col'], bins=20, edgecolor='black')
plt.subplot(1, 3, 2)
sns.boxplot(y=df['numeric_col'])
plt.subplot(1, 3, 3)
sns.violinplot(y=df['numeric_col'])

# Relationship plots
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='feature1', y='feature2',
hue='category')
sns.regplot(data=df, x='feature1', y='target')

# Categorical data
sns.countplot(data=df, x='category')
sns.barplot(data=df, x='category', y='value')
```

Advanced Visualizations

Create comprehensive dashboards and reports.

```
# Subplots for multiple views
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
axes[0,0].hist(df['col1'])
axes[0,1].scatter(df['col1'], df['col2'])
axes[1,0].boxplot(df['col1'])
sns.heatmap(df.corr(), ax=axes[1,1])

# Interactive plots with Plotly
import plotly.express as px
fig = px.scatter(df, x='feature1', y='feature2',
          color='category', size='value',
          hover_data=['additional_info'])
fig.show()
```

Statistical Plots

Visualize statistical relationships and model results.

```
# Pair plots for correlation
sns.pairplot(df, hue='target_category')

# Residual plots for regression
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.scatter(y_pred, y_test - y_pred)
plt.xlabel('Predicted')
plt.ylabel('Residuals')
plt.subplot(1, 2, 2)
plt.scatter(y_test, y_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(),
y_test.max()], 'r--')

# ROC Curve for classification
from sklearn.metrics import roc_curve, auc
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')
```

Customization & Styling

Professional visualization formatting.

```
# Set style and colors
plt.style.use('seaborn-v0_8')
sns.set_palette("husl")

# Custom figure settings
plt.figure(figsize=(12, 8))
plt.title('Professional Chart Title', fontsize=16,
fontweight='bold')
plt.xlabel('X-axis Label', fontsize=14)
plt.ylabel('Y-axis Label', fontsize=14)
plt.legend(loc='best')
plt.grid(True, alpha=0.3)
plt.tight_layout()

# Save high-quality plots
plt.savefig('analysis_plot.png', dpi=300,
bbox_inches='tight')
```

Model Deployment & MLOps

Model Persistence

Save and load trained models for production use.

```
# Save models with pickle
import pickle
with open('model.pkl', 'wb') as f:
  pickle.dump(trained_model, f)

# Load saved model
with open('model.pkl', 'rb') as f:
  loaded_model = pickle.load(f)

# Using joblib for sklearn models
import joblib
joblib.dump(trained_model, 'model.joblib')
loaded_model = joblib.load('model.joblib')

# Model versioning with timestamps
import datetime
timestamp =
datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
model_name = f'model_{timestamp}.pkl'
```

Cross-Validation & Hyperparameter Tuning

Optimize model performance and prevent overfitting.

```
# Cross-validation
from sklearn.model_selection import cross_val_score,
StratifiedKFold
cv_scores = cross_val_score(model, X, y, cv=5,
scoring='accuracy')
print(f"CV Accuracy: {cv_scores.mean():.3f} (+/-
{cv_scores.std() * 2:.3f})")

# Grid Search for hyperparameter tuning
from sklearn.model_selection import GridSearchCV
param_grid = {
  'n_estimators': [100, 200, 300],
  'max_depth': [3, 5, 7],
  'min_samples_split': [2, 5, 10]
}
grid_search = GridSearchCV(RandomForestClassifier(),
param_grid, cv=5)
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_
```

Performance Monitoring

Having quick access to essential concepts and commands can make all the difference in your workflow. Whether you're a beginner finding your footing or an experienced practitioner looking for a reliable reference, cheat sheets serve as invaluable companions.

```
# Model performance tracking
import time
start_time = time.time()
predictions = model.predict(X_test)
inference_time = time.time() - start_time
print(f"Inference time: {inference_time:.4f} seconds")

# Memory usage monitoring
import psutil
process = psutil.Process()
memory_usage = process.memory_info().rss / 1024 /
1024  # MB
print(f"Memory usage: {memory_usage:.2f} MB")

# Feature importance analysis
feature_importance = model.feature_importances_
importance_df = pd.DataFrame({
  'feature': X.columns,
  'importance': feature_importance
}).sort_values('importance', ascending=False)
```

Model Documentation

Document model assumptions, performance, and usage.

```
# Create model report
model_report = {
  'model_type': type(model).__name__,
  'training_data_shape': X_train.shape,
  'features_used': list(X.columns),
  'performance_metrics': {
    'accuracy': accuracy_score(y_test, y_pred),
    'precision': precision_score(y_test, y_pred,
average='weighted'),
    'recall': recall_score(y_test, y_pred,
average='weighted')
  },
  'training_date': datetime.datetime.now().isoformat(),
  'model_version': '1.0'
}

# Save model metadata
import json
with open('model_metadata.json', 'w') as f:
  json.dump(model_report, f, indent=2)
```

Best Practices & Tips

Professional data science workflow recommendations.

Code Organization

Structure projects for reproducibility and collaboration.

```
# Project structure
project/
 ├── data/
 │   ├── raw/
 │   └── processed/
 ├── notebooks/
 ├── src/
 │   ├── data_processing.py
 │   ├── modeling.py
 │   └── visualization.py
 ├── models/
 ├── reports/
 └── requirements.txt

# Version control with git
git init
git add .
git commit -m "Initial data
science project setup"
```

Environment Management

Ensure reproducible environments across systems.

```
# Create virtual environment
python -m venv ds_env
source ds_env/bin/activate  #
Linux/Mac
# ds_env\Scripts\activate   #
Windows

# Requirements file
pip freeze > requirements.txt

# Conda environment
conda create -n ds_project
python=3.9
conda activate ds_project
conda install pandas numpy
scikit-learn matplotlib seaborn
jupyter
```

Data Quality Checks

Validate data integrity throughout the pipeline.

```
# Data validation functions
def validate_data(df):
  checks = {
    'shape': df.shape,
    'missing_values':
df.isnull().sum().sum(),
    'duplicates':
df.duplicated().sum(),
    'data_types':
df.dtypes.to_dict()
  }
  return checks

# Automated data quality report
def data_quality_report(df):
  print(f"Dataset shape:
{df.shape}")
  print(f"Missing values:
{df.isnull().sum().sum()}")
  print(f"Duplicate rows:
{df.duplicated().sum()}")
  print("\nColumn data types:")
  print(df.dtypes)
```

Reference: This cheatsheet covers essential data science concepts, workflows, and Python implementations for comprehensive data analysis and machine learning projects. The reader should have at least a basic understanding of statistics and linear algebra, though beginners may find this resource helpful as well.