PYTHON PANDAS CHEAT SHEET

CREATING DATAFRAMES

Creating DataFrames is the foundation of using Pandas. Here's how to create a simple DataFrame and display its content.

```
import pandas as pd
import numpy as np

# Create a simple DataFrame
df = pd.DataFrame(
    [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]],
    columns=["A", "B", "C"],
    index=["x", "y", "z", "zz"]
)

# Display the first few rows
df.head()

# Display the last two rows
df.tail(2)
```

LOADING DATA

Loading data into DataFrames from various file formats is crucial for real-world data analysis.

```
# Load data from CSV
coffee = pd.read_csv('./warmup-data/coffee.csv')

# Load data from Parquet
results = pd.read_parquet('./data/results.parquet')

# Load data from Excel
olympics_data = pd.read_excel('./data/olympics-
data.xlsx', sheet_name="results")
```

ACCESSING DATA

Accessing different parts of the DataFrame allows for flexible data manipulation and inspection.

```
# Access columns
df.columns

# Access index
df.index.tolist()

# General info about the DataFrame
df.info()

# Statistical summary
df.describe()

# Number of unique values in each column
df.nunique()

# Access unique values in a column
df['A'].unique()

# Shape and size of DataFrame
df.shape
df.size
```

FILTERING DATA

Filtering data is essential for extracting relevant subsets based on conditions.

```
# Filter rows based on conditions
bios.loc[bios["height_cm"] > 215]

# Multiple conditions
bios[(bios['height_cm'] > 215) &
(bios['born_country']=='USA')]

# Filter by string conditions
bios[bios['name'].str.contains("keith", case=False)]

# Regex filters
bios[bios['name'].str.contains(r'^[AEIOUaeiou]',
na=False)]
```

ADDING/REMOVING COLUMNS

Adding and removing columns is important for maintaining and analyzing relevant data.

```
# Add a new column
coffee['price'] = 4.99

# Conditional column
coffee['new_price'] = np.where(coffee['Coffee
Type']=='Espresso', 3.99, 5.99)

# Remove a column
coffee.drop(columns=['price'], inplace=True)

# Rename columns
coffee.rename(columns={'new_price': 'price'},
inplace=True)

# Create new columns from existing ones
coffee['revenue'] = coffee['Units Sold'] *
coffee['price']
```

MERGING AND CONCATENATING DATA

Merging and concatenating DataFrames is useful for combining different datasets for comprehensive analysis.

```
# Merge DataFrames
nocs = pd.read_csv('./data/noc_regions.csv')
bios_new = pd.merge(bios, nocs,
left_on='born_country', right_on='NOC', how='left')

# Concatenate DataFrames
usa = bios[bios['born_country']=='USA'].copy()
gbr = bios[bios['born_country']=='GBR'].copy()
new_df = pd.concat([usa, gbr])
```

HANDLING NULL VALUES

Handling null values is essential to ensure the integrity of data analysis.

```
# Fill NaNs with a specific value
coffee['Units Sold'].fillna(0, inplace=True)

# Interpolate missing values
coffee['Units Sold'].interpolate(inplace=True)

# Drop rows with NaNs
coffee.dropna(subset=['Units Sold'], inplace=True)
```

AGGREGATING DATA

Aggregation functions like value counts and group by help in summarizing data efficiently.

```
# Value counts
bios['born_city'].value_counts()

# Group by and aggregation
coffee.groupby(['Coffee Type'])['Units Sold'].sum()
coffee.groupby(['Coffee Type'])['Units Sold'].mean()

# Pivot table
pivot = coffee.pivot(columns='Coffee Type',
index='Day', values='revenue')
```

ADVANCED FUNCTIONALITY

Advanced functionalities such as rolling calculations, rankings, and shifts can provide deeper insights.

```
# Cumulative sum
coffee['cumsum'] = coffee['Units Sold'].cumsum()

# Rolling window
latte = coffee[coffee['Coffee Type']=="Latte"].copy()
latte['3day'] = latte['Units Sold'].rolling(3).sum()

# Rank
bios['height_rank'] =
bios['height_cm'].rank(ascending=False)

# Shift
coffee['yesterday_revenue'] =
coffee['revenue'].shift(1)
```

NEW FUNCTIONALITY

The PyArrow backend offers optimized performance for certain operations, particularly string operations.

```
# PyArrow backend
results_arrow = pd.read_csv('./data/results.csv',
engine='pyarrow', dtype_backend='pyarrow')
results_arrow.info()
```

DATA CLEANING

IMPORTING PANDAS

import pandas as pd

- This line imports the pandas library, which is used for data manipulation and analysis. The alias pd is commonly used to refer to it.

LOADING THE DATASET

df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/refs/heads/master/titanic.csv')

- This line loads the Titanic dataset from a URL and stores it in a variable called df. The dataset is in CSV (Comma-Separated Values) format.

DISPLAYING THE DATASET

display(df)

- This line shows the contents of the dataset in a tabular form.

GETTING INFORMATION ABOUT THE DATASET

df.info()

- This command provides a summary of the dataset, including the number of rows, columns, and the data types of each column. It also shows if there are any missing values.

CLEANING THE DATASET

dfClean = df.drop(['PassengerId','Name','Cabin','Ticket'], axis=1)

- This line removes (drops) the columns PassengerId, Name, Cabin, and Ticket from the dataset, as they may not be useful for the analysis.

HANDLING MISSING VALUES

dfClean['Age'].fillna(dfClean['Age'].mean(), inplace=True)
dfClean['Embarked'].fillna(dfClean['Embarked'].mode()[0], inplace=True)

- The missing values in the Age column are replaced with the average (mean) age.
- The missing values in the Embarked column are filled with the most common value (mode).

ENCODING CATEGORICAL DATA

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
dfClean['Sex'] = le.fit_transform(dfClean['Sex'])
dfClean['Embarked'] = le.fit_transform(dfClean['Embarked'])

- Categorical data (like Sex and Embarked) is converted into numbers using LabelEncoder. For example, the Sex column values (male and female) are turned into 0 and 1.

CHECKING CORRELATIONS

dfClean.corr()

- This line computes the correlation between different columns in the dataset. Correlation measures how changes in one feature relate to changes in another.

SEPARATING FEATURES AND TARGET VARIABLE

x = dfClean.drop('Survived', axis=1)
y = dfClean['Survived']

- The features (x) are all the columns except Survived, which is the target variable (y) that indicates whether a passenger survived or not.

SPLITTING THE DATA INTO TRAINING AND TEST SETS

```
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)
```

- This line splits the dataset into training and testing sets. 80% of the data will be used for training, and 20% will be used for testing. The random_state=42 ensures that the split is reproducible.

FILTERING THE DATA

```
dfClean[(dfClean['Age'] <= 5) | (dfClean['Sex'] == 1)]
```

- This line filters the dataset to show only the rows where the passenger's age is 5 or younger, or where the passenger's sex is represented as 1 (after encoding).

DATA ANALYSIS

LOADING AND EXPLORING THE DATA

```
import pandas as pd
mpg_df = pd.read_csv('autompg1.csv')
mpg_df.info()
```

- We import the pandas library, load a CSV file (autompg1.csv) into a dataframe (mpg_df), and check the data's structure using mpg_df.info().

CREATING A BASIC PLOT

```
import matplotlib.pyplot as plt
plt.scatter(x='mpg', y='cylinder', data=mpg_data)
plt.show()
```

- This creates a scatter plot with 'mpg' (miles per gallon) on the x-axis and 'cylinder' on the y-axis to visualize their relationship.

INSTALLING AND USING SEABORN FOR VISUALIZATION

```
pip install seaborn
import seaborn as sns
sns.pairplot(mpg_df)
```

- We install and use the seaborn library to create a pairplot, which shows relationships between different numerical columns in mpg_df.

CALCULATING CORRELATIONS

```
mpg_df.corr()
```

- This calculates and shows the correlation values (relationships) between different columns in the data.

ENCODING TEXT DATA

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
mpg_df['car name'] = le.fit_transform(mpg_df['car_name'])
```

- LabelEncoder turns text data in 'car_name' into numbers so the model can use it. Here, we replace 'car name' with numbers to help with calculations.

HANDLING MISSING VALUES

```
mpg_df[mpg_df['horsepower']=='?']
mpg_df['horsepower'].replace('?', np.nan, inplace=True)
mpg_df['horsepower'].fillna(mpg_df['horsepower'].mean(), inplace=True)
mpg_df['horsepower'] = mpg_df['horsepower'].astype('float')
```

- Here, ? in the 'horsepower' column is treated as a missing value and replaced with the average horsepower value for accuracy.

PREPARING DATA FOR MODEL TRAINING

```
x = mpg_df[['origin', 'model year', 'cylinder', 'displacement', 'weight', 'horsepower']]
y = mpg_df['mpg']
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_std = sc.fit_transform(x)
```

- We set x as the input features and y as the target to predict ('mpg'). We then scale x with StandardScaler to standardize values, which improves model performance.

SPLITTING DATA FOR TRAINING AND TESTING

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_std, y, test_size=0.2, random_state=42)
```

- We split the data into training (80%) and testing (20%) sets. The training set is used to teach the model, while the test set is for evaluation.

CREATING AND TRAINING A MODEL

```
from sklearn.linear_model import LinearRegression
lr_model = LinearRegression()
lr_model.fit(x_train, y_train)
```

- A LinearRegression model is created and trained using the x_train and y_train data.

MAKING PREDICTIONS

```
y_pred = lr_model.predict(x_test)
```

- We use the trained model to predict mpg values based on x_test.

EVALUATING MODEL ACCURACY

```
from sklearn.metrics import r2_score
print('Accuracy score:', r2_score(y_test, y_pred) * 100, '%')
```

- We calculate the model’s accuracy by checking how close the predictions (y_pred) are to the actual test values (y_test) using r2_score. An r2_score closer to 1 means better accuracy.