🐍 Day 2: Python Fundamentals for Data Engineering - Complete Guide

📚 What You'll Learn Today

- **Python Environment Setup** for data engineering
- **Core Python Concepts** essential for data work
- **File Handling and Data Structures** for data processing
- **Introduction to Pandas** for data manipulation
- **Your First Data Pipeline** script

🎯 Learning Objectives

By the end of Day 2, you will:

1. Have a properly configured Python development environment
2. Master Python data structures crucial for data engineering
3. Understand file I/O operations for various data formats
4. Write your first data processing script with pandas
5. Handle common data engineering tasks in Python

🛠️ Environment Setup (30 minutes)

Step 1: Python Installation Verification

```bash
# Check Python version (should be 3.9+)
python --version
# or
python3 --version

# Check pip installation
pip --version
```

Step 2: Create Virtual Environment

```bash
# Create virtual environment
python -m venv data_engineering_env

# Activate environment
# Windows:
data_engineering_env\Scripts\activate

# macOS/Linux:
source data_engineering_env/bin/activate
```

Step 3: Install Essential Libraries

```bash
# Create requirements.txt
pip install pandas numpy matplotlib seaborn jupyter requests
pip freeze > requirements.txt
```

Step 4: Set Up Development Environment

```python
# Create project structure
data-engineering-50-days/
├── day-02/
│   ├── notebooks/
│   ├── data/
│   ├── scripts/
│   └── requirements.txt
├── datasets/
└── venv/
```

🔥 Python Fundamentals for Data Engineering

Data Types and Variables

python

```python
# Essential data types for data engineering
customer_id = 12345                     # int
revenue = 1599.99                       # float
customer_name = "John Doe"              # string
is_active = True                        # boolean
signup_date = None                      # NoneType for missing data

# Type checking (important for data validation)
print(f"Customer ID type: {type(customer_id)}")
print(f"Revenue type: {type(revenue)}")
```

Data Structures for Data Engineering

1. Lists - For Sequential Data

python

```python
# Customer transaction amounts
transactions = [99.99, 150.00, 75.50, 200.25, 89.99]

# Adding new transactions
transactions.append(125.00)
transactions.extend([300.00, 45.99])

# Data engineering operations
total_revenue = sum(transactions)
avg_transaction = sum(transactions) / len(transactions)
max_transaction = max(transactions)

print(f"Total Revenue: ${total_revenue:.2f}")
print(f"Average Transaction: ${avg_transaction:.2f}")
print(f"Highest Transaction: ${max_transaction:.2f}")
```

2. Dictionaries - For Structured Data

python

```
# Customer record (similar to database row)
customer_record = {
    'customer_id': 12345,
    'name': 'John Doe',
    'email': 'john.doe@email.com',
    'age': 32,
    'city': 'New York',
    'total_spent': 1599.99,
    'orders': ['ORD001', 'ORD002', 'ORD003']
}

# Accessing and updating data
print(f"Customer: {customer_record['name']}")
print(f"Total Spent: ${customer_record['total_spent']}")

# Adding new information
customer_record['last_login'] = '2024-06-10'
customer_record['total_spent'] += 99.99

# Data validation
required_fields = ['customer_id', 'name', 'email']
missing_fields = [field for field in required_fields if field not in customer_record]
print(f"Missing fields: {missing_fields}")
```

3. Sets - For Data Deduplication

python

```
# Remove duplicate customer IDs
customer_ids = [1001, 1002, 1003, 1001, 1004, 1002, 1005]
unique_customers = set(customer_ids)
print(f"Original count: {len(customer_ids)}")
print(f"Unique customers: {len(unique_customers)}")

# Find common customers between datasets
dataset_a_customers = {1001, 1002, 1003, 1004}
dataset_b_customers = {1003, 1004, 1005, 1006}
common_customers = dataset_a_customers.intersection(dataset_b_customers)
print(f"Common customers: {common_customers}")
```

📁 File Handling for Data Engineering

Working with CSV Files

```python
import csv

# Writing data to CSV
customers_data = [
    ['customer_id', 'name', 'email', 'city', 'total_spent'],
    [1001, 'John Doe', 'john@email.com', 'New York', 1599.99],
    [1002, 'Jane Smith', 'jane@email.com', 'Los Angeles', 2345.50],
    [1003, 'Bob Johnson', 'bob@email.com', 'Chicago', 876.25]
]

# Write CSV file
with open('day-02/data/customers.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(customers_data)

# Read CSV file
with open('day-02/data/customers.csv', 'r') as file:
    reader = csv.DictReader(file)
    for row in reader:
        print(f"Customer: {row['name']}, Spent: ${row['total_spent']}")
```

Working with JSON Files

```python
import json

# Customer data in JSON format
customer_json = {
    "customer_id": 1001,
    "personal_info": {
        "name": "John Doe",
        "email": "john@email.com",
        "age": 32
    },
    "address": {
        "street": "123 Main St",
        "city": "New York",
        "zipcode": "10001"
    },
    "orders": [
        {"order_id": "ORD001", "amount": 99.99, "date": "2024-06-01"},
        {"order_id": "ORD002", "amount": 150.00, "date": "2024-06-05"}
    ]
}

# Write JSON file
with open('day-02/data/customer.json', 'w') as file:
    json.dump(customer_json, file, indent=2)

# Read JSON file
with open('day-02/data/customer.json', 'r') as file:
    data = json.load(file)
    print(f"Customer: {data['personal_info']['name']}")
    print(f"Orders: {len(data['orders'])}")
```

🐼 Introduction to Pandas

Why Pandas for Data Engineering?

Pandas is the world's most popular Python library, used for everything from data manipulation to data analysis. The pandas library is one of the most frequently used libraries for data engineering in Python. This versatile library equips data engineers with powerful manipulation and analysis capabilities.

Creating DataFrames

python

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

# Create DataFrame from dictionary
sales_data = {
    'date': ['2024-06-01', '2024-06-02', '2024-06-03', '2024-06-04', '2024-06-05'],
    'product': ['Laptop', 'Mouse', 'Keyboard', 'Monitor', 'Laptop'],
    'category': ['Electronics', 'Accessories', 'Accessories', 'Electronics', 'Electron
    'quantity': [2, 10, 5, 1, 3],
    'price': [999.99, 25.99, 79.99, 299.99, 999.99],
    'customer_id': [1001, 1002, 1003, 1004, 1001]
}

df = pd.DataFrame(sales_data)
print("Sales DataFrame:")
print(df)
print(f"\nDataFrame shape: {df.shape}")
print(f"Data types:\n{df.dtypes}")
```

Essential DataFrame Operations

python

```python
# Basic information about the dataset
print("Dataset Info:")
print(df.info())
print("\nBasic Statistics:")
print(df.describe())

# Data exploration
print(f"Unique products: {df['product'].nunique()}")
print(f"Unique customers: {df['customer_id'].nunique()}")
print(f"Date range: {df['date'].min()} to {df['date'].max()}")

# Calculate total sales amount
df['total_amount'] = df['quantity'] * df['price']
print("\nDataFrame with calculated column:")
print(df[['product', 'quantity', 'price', 'total_amount']])
```

Data Filtering and Selection

```python
# Filter high-value transactions
high_value_sales = df[df['total_amount'] > 500]
print("High-value sales (>$500):")
print(high_value_sales)

# Filter by multiple conditions
electronics_sales = df[(df['category'] == 'Electronics') & (df['quantity'] > 1)]
print("\nElectronics sales with quantity > 1:")
print(electronics_sales)

# Select specific columns
summary = df[['product', 'quantity', 'total_amount']]
print("\nSales summary:")
print(summary)
```

Data Aggregation

```python
# Group by operations
category_summary = df.groupby('category').agg({
    'total_amount': ['sum', 'mean', 'count'],
    'quantity': 'sum'
}).round(2)

print("Sales by Category:")
print(category_summary)

# Customer analysis
customer_summary = df.groupby('customer_id').agg({
    'total_amount': 'sum',
    'quantity': 'sum',
    'product': 'count'
}).rename(columns={'product': 'order_count'})

print("\nCustomer Summary:")
print(customer_summary)
```

🚀 Your First Data Pipeline Script

Let's create a complete data processing pipeline:

python

```python
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import os

class SalesDataPipeline:
    def __init__(self, input_file, output_dir):
        self.input_file = input_file
        self.output_dir = output_dir
        self.df = None

    def extract_data(self):
        """Extract data from CSV file"""
        try:
            print(f"Extracting data from {self.input_file}")
            self.df = pd.read_csv(self.input_file)
            print(f"Successfully loaded {len(self.df)} records")
            return True
        except FileNotFoundError:
            print(f"Error: File {self.input_file} not found")
            return False
        except Exception as e:
            print(f"Error loading data: {e}")
            return False

    def transform_data(self):
        """Clean and transform the data"""
        print("Transforming data...")

        # Convert date column to datetime
        self.df['date'] = pd.to_datetime(self.df['date'])

        # Calculate total amount
        self.df['total_amount'] = self.df['quantity'] * self.df['price']

        # Add calculated fields
        self.df['year'] = self.df['date'].dt.year
        self.df['month'] = self.df['date'].dt.month
        self.df['day_of_week'] = self.df['date'].dt.day_name()

        # Data quality checks
        print(f"Checking data quality...")
        null_counts = self.df.isnull().sum()
```

```python
        if null_counts.any():
            print(f"Found null values:\n{null_counts[null_counts > 0]}")

        # Remove any duplicate records
        initial_count = len(self.df)
        self.df = self.df.drop_duplicates()
        final_count = len(self.df)
        if initial_count != final_count:
            print(f"Removed {initial_count - final_count} duplicate records")

        print("Data transformation completed")

    def load_data(self):
        """Save processed data to output files"""
        print("Loading data to output files...")

        # Create output directory if it doesn't exist
        os.makedirs(self.output_dir, exist_ok=True)

        # Save main processed dataset
        output_file = os.path.join(self.output_dir, 'processed_sales.csv')
        self.df.to_csv(output_file, index=False)
        print(f"Saved processed data to {output_file}")

        # Create summary reports
        self.create_summary_reports()

    def create_summary_reports(self):
        """Generate summary reports"""

        # Daily sales summary
        daily_summary = self.df.groupby('date').agg({
            'total_amount': 'sum',
            'quantity': 'sum',
            'customer_id': 'nunique'
        }).rename(columns={'customer_id': 'unique_customers'})

        daily_file = os.path.join(self.output_dir, 'daily_summary.csv')
        daily_summary.to_csv(daily_file)
        print(f"Saved daily summary to {daily_file}")

        # Product performance
        product_summary = self.df.groupby('product').agg({
            'total_amount': 'sum',
```

```python
            'quantity': 'sum',
            'customer_id': 'nunique'
        }).sort_values('total_amount', ascending=False)

        product_file = os.path.join(self.output_dir, 'product_performance.csv')
        product_summary.to_csv(product_file)
        print(f"Saved product performance to {product_file}")

    def run_pipeline(self):
        """Execute the complete ETL pipeline"""
        print("Starting Sales Data ETL Pipeline...")
        print("=" * 50)

        # Extract
        if not self.extract_data():
            return False

        # Transform
        self.transform_data()

        # Load
        self.load_data()

        print("=" * 50)
        print("Pipeline completed successfully!")
        return True

# Example usage
if __name__ == "__main__":
    # Run the pipeline
    pipeline = SalesDataPipeline(
        input_file='day-02/data/customers.csv',
        output_dir='day-02/output'
    )

    pipeline.run_pipeline()
```

🎯 Real-World Example: E-commerce Data Processing

Let's work with a realistic e-commerce dataset:

python

```python
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

# Generate sample e-commerce data
np.random.seed(42)

def generate_ecommerce_data(num_records=1000):
    """Generate realistic e-commerce transaction data"""

    # Product catalog
    products = {
        'Electronics': ['iPhone 14', 'Samsung Galaxy', 'MacBook Pro', 'iPad', 'AirPods
        'Clothing': ['T-Shirt', 'Jeans', 'Sneakers', 'Hoodie', 'Dress'],
        'Home': ['Coffee Maker', 'Vacuum Cleaner', 'Bed Sheets', 'Lamp', 'Chair']
    }

    # Generate data
    data = []
    start_date = datetime(2024, 1, 1)

    for i in range(num_records):
        # Random date in 2024
        random_days = np.random.randint(0, 160)  # First 160 days of 2024
        transaction_date = start_date + timedelta(days=random_days)

        # Random category and product
        category = np.random.choice(list(products.keys()))
        product = np.random.choice(products[category])

        # Generate realistic prices based on category
        if category == 'Electronics':
            base_price = np.random.uniform(99, 1999)
        elif category == 'Clothing':
            base_price = np.random.uniform(19, 199)
        else:  # Home
            base_price = np.random.uniform(29, 599)

        record = {
            'transaction_id': f'TXN{i+1:06d}',
            'date': transaction_date.strftime('%Y-%m-%d'),
            'customer_id': np.random.randint(1001, 1500),
            'product': product,
```

```
            'category': category,
            'quantity': np.random.randint(1, 5),
            'unit_price': round(base_price, 2),
            'discount_percent': np.random.choice([0, 5, 10, 15, 20], p=[0.6, 0.15, 0.1
        }

        data.append(record)

    return pd.DataFrame(data)

# Generate and save sample data
ecommerce_df = generate_ecommerce_data(1000)

# Calculate derived fields
ecommerce_df['discount_amount'] = (ecommerce_df['unit_price'] *
                                  ecommerce_df['quantity'] *
                                  ecommerce_df['discount_percent'] / 100)

ecommerce_df['total_amount'] = (ecommerce_df['unit_price'] *
                               ecommerce_df['quantity'] -
                               ecommerce_df['discount_amount'])

# Save to CSV
ecommerce_df.to_csv('day-02/data/ecommerce_transactions.csv', index=False)
print("Generated e-commerce sample data")
print(f"Dataset shape: {ecommerce_df.shape}")
print("\nSample records:")
print(ecommerce_df.head())
```

Analyzing the E-commerce Data

python

```python
# Load and analyze the generated data
df = pd.read_csv('day-02/data/ecommerce_transactions.csv')

# Basic analysis
print("E-commerce Data Analysis")
print("=" * 40)
print(f"Total transactions: {len(df):,}")
print(f"Date range: {df['date'].min()} to {df['date'].max()}")
print(f"Total revenue: ${df['total_amount'].sum():,.2f}")
print(f"Average order value: ${df['total_amount'].mean():.2f}")

# Category performance
category_stats = df.groupby('category').agg({
    'total_amount': ['sum', 'mean', 'count'],
    'quantity': 'sum',
    'customer_id': 'nunique'
}).round(2)

print("\nCategory Performance:")
print(category_stats)

# Top customers
top_customers = df.groupby('customer_id').agg({
    'total_amount': 'sum',
    'transaction_id': 'count'
}).sort_values('total_amount', ascending=False).head(10)

print("\nTop 10 Customers by Revenue:")
print(top_customers)

# Monthly trends
df['date'] = pd.to_datetime(df['date'])
df['month'] = df['date'].dt.to_period('M')

monthly_revenue = df.groupby('month')['total_amount'].sum()
print("\nMonthly Revenue Trends:")
print(monthly_revenue)
```

✅ Day 2 Practical Tasks

Task 1: Environment Setup (30 minutes)

- [ ] Set up Python virtual environment
- [ ] Install required packages (pandas, numpy, matplotlib)
- [ ] Create project folder structure
- [ ] Test installation with simple script

Task 2: Python Fundamentals Practice (45 minutes)

- [ ] Complete data structures exercises
- [ ] Practice file I/O operations
- [ ] Work with different data formats (CSV, JSON)
- [ ] Handle errors and exceptions

Task 3: Pandas Basics (60 minutes)

- [ ] Create DataFrames from different sources
- [ ] Practice data filtering and selection
- [ ] Perform basic aggregations
- [ ] Generate summary statistics

Task 4: Build Your First Pipeline (45 minutes)

- [ ] Run the complete ETL pipeline script
- [ ] Modify the pipeline for different data sources
- [ ] Add error handling and logging
- [ ] Create custom transformation functions

📚 Essential Resources for Day 2

📖 Documentation and Tutorials

1. **Python Official Documentation**
   - Source: python.org
   - Focus: Built-in functions and standard library
2. **Pandas Documentation**
   - Source: pandas.pydata.org
   - Pandas offers functions for data transformation, aggregation and visualization, which are important for analysis
3. **GeeksforGeeks Pandas Tutorial**
   - Source: geeksforgeeks.org/pandas-tutorial

- Comprehensive tutorial covering fundamentals to advanced operations

🎥 Video Resources

1. **"Python and Pandas for Data Engineering" - Duke University**
    - Platform: Coursera
    - Learn how to set up a version-controlled Python working environment which can utilize third party libraries
2. **"Data Manipulation with pandas" - DataCamp**
    - Platform: DataCamp
    - Hands-on experience manipulating real-world datasets, such as Walmart sales figures and global temperature time series

📊 Practice Datasets

1. **Kaggle Sample Sales Data**
    - Source: kaggle.com/kyanyoga/sample-sales-data
    - Use case: Practice data loading and basic analysis
2. **Retail Dataset**
    - Source: UCI Machine Learning Repository
    - Use case: Real-world e-commerce data analysis
3. **Generated Sample Data**
    - Source: Your Day 2 scripts
    - Use case: Controlled environment for learning

🛠️ Development Tools

1. **Jupyter Notebook**
    - Installation: `pip install jupyter`
    - Usage: Interactive development and documentation
2. **VS Code with Python Extension**
    - Source: code.visualstudio.com
    - Extensions: Python, Jupyter, GitLens
3. **Anaconda Distribution**
    - Source: anaconda.com
    - Benefits: Pre-configured data science environment

🎯 Common Challenges and Solutions

Challenge 1: Memory Issues with Large Files

python

```python
# Solution: Read data in chunks
def process_large_csv(filename, chunk_size=10000):
    """Process large CSV files in chunks"""
    for chunk in pd.read_csv(filename, chunksize=chunk_size):
        # Process each chunk
        processed_chunk = chunk.groupby('category')['amount'].sum()
        yield processed_chunk

# Usage
results = []
for chunk_result in process_large_csv('large_dataset.csv'):
    results.append(chunk_result)

final_result = pd.concat(results).groupby(level=0).sum()
```

Challenge 2: Data Type Issues

python

```python
# Solution: Explicit data type conversion
def clean_data_types(df):
    """Clean and convert data types"""
    # Convert dates
    date_columns = ['created_date', 'updated_date']
    for col in date_columns:
        if col in df.columns:
            df[col] = pd.to_datetime(df[col], errors='coerce')

    # Convert numeric columns
    numeric_columns = ['price', 'quantity', 'amount']
    for col in numeric_columns:
        if col in df.columns:
            df[col] = pd.to_numeric(df[col], errors='coerce')

    return df
```

Challenge 3: Missing Data Handling

python

```python
# Solution: Comprehensive missing data strategy
def handle_missing_data(df):
    """Handle missing data based on business rules"""

    # Fill missing numerical values with median
    numeric_columns = df.select_dtypes(include=[np.number]).columns
    for col in numeric_columns:
        df[col].fillna(df[col].median(), inplace=True)

    # Fill missing categorical values with mode
    categorical_columns = df.select_dtypes(include=['object']).columns
    for col in categorical_columns:
        df[col].fillna(df[col].mode()[0], inplace=True)

    return df
```

📝 Day 2 Deliverables

1. Working Python Environment ✅

- Virtual environment with required packages
- Project structure for remaining 48 days
- Development tools configured

2. Python Skills Assessment

Rate yourself after today (1-10):

- [ ] Python basics: ___/10
- [ ] Data structures: ___/10
- [ ] File handling: ___/10
- [ ] Pandas basics: ___/10
- [ ] Error handling: ___/10

3. Code Repository Update

```bash
# Commit your Day 2 work
git add .
git commit -m "Day 2: Python fundamentals and pandas basics"
git push origin main
```

4. Learning Journal Entry

Create `day-02/learning-notes.md`:

```markdown
# Day 2: Python Fundamentals - Learning Notes

## Key Concepts Mastered
- Python environment setup and virtual environments
- Essential data structures for data engineering
- File I/O operations with CSV and JSON
- Pandas DataFrame operations and data manipulation
- Basic ETL pipeline implementation

## Practical Skills Gained
- Created and managed virtual environments
- Built first data processing pipeline
- Performed data analysis with pandas
- Handled different file formats
- Implemented error handling in data scripts

## Challenges Faced
- [Document any difficulties and how you solved them]

## Real-World Applications
- E-commerce transaction processing
- Sales data analysis and reporting
- Data quality checks and validation
- Automated data transformation workflows

## Tomorrow's Preparation
- Review SQL basics
- Set up PostgreSQL environment
- Download sample databases for practice
```

🔗 Bonus: Python Best Practices for Data Engineering

1. Code Organization

```python
# Good structure for data engineering scripts
import pandas as pd
import numpy as np
from datetime import datetime
import logging

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def extract_data(source_path):
    """Extract data from source"""
    pass

def transform_data(df):
    """Apply transformations"""
    pass

def validate_data(df):
    """Validate data quality"""
    pass

def load_data(df, destination_path):
    """Load data to destination"""
    pass

if __name__ == "__main__":
    # Main execution
    pass
```

2. Error Handling

```python
def safe_data_operation(func):
    """Decorator for safe data operations"""
    def wrapper(*args, **kwargs):
        try:
            return func(*args, **kwargs)
        except Exception as e:
            logger.error(f"Error in {func.__name__}: {e}")
            raise
    return wrapper

@safe_data_operation
def process_customer_data(df):
    # Your data processing logic
    return df
```

3. Configuration Management

```python
# config.py
import os

class Config:
    DATA_DIR = os.getenv('DATA_DIR', 'data/')
    OUTPUT_DIR = os.getenv('OUTPUT_DIR', 'output/')
    CHUNK_SIZE = int(os.getenv('CHUNK_SIZE', 10000))
    LOG_LEVEL = os.getenv('LOG_LEVEL', 'INFO')
```

✅ Day 2 Checklist

- [ ] ✅ Set up Python virtual environment
- [ ] ✅ Install and test essential libraries
- [ ] ✅ Master Python data structures
- [ ] ✅ Practice file I/O operations
- [ ] ✅ Learn pandas basics and DataFrame operations
- [ ] ✅ Build your first ETL pipeline
- [ ] ✅ Work with real-world e-commerce data
- [ ] ✅ Update GitHub repository
- [ ] ✅ Document learning progress

- [ ] ✅ Prepare for tomorrow's SQL session

🎯 Tomorrow's Preview: Day 3 - SQL Fundamentals

What to expect:

- PostgreSQL installation and setup
- Essential SQL operations for data engineering
- Working with sample databases
- Data modeling concepts
- Your first database-driven data pipeline

Preparation:

- Download PostgreSQL installer
- Review basic database concepts
- Prepare sample datasets for database loading

🎉 *Congratulations on completing Day 2! You now have a solid Python foundation for data engineering. Tomorrow, we'll add SQL to your toolkit - the backbone of data engineering.*

Progress: 4% (2/50 days) | **Next**: Day 3 - SQL Fundamentals