Cheat Sheet for comprehensive IBM Data Science Professional Certificate

Data Science Overview

- **Definition**: Data science is the interdisciplinary field that uses scientific methods, processes, algorithms, and systems to extract knowledge and insights from structured and unstructured data.

- **Key Components**:

  - **Data Collection**

  - **Data Cleaning**

  - **Data Analysis**

  - **Data Visualization**

  - **Machine Learning**

  - **Model Deployment**

Python for Data Science

- **Basic Syntax**:

  - **Variables**: `x = 10`, `name = "John"`

  - **Data Types**: `int`, `float`, `str`, `list`, `tuple`, `dict`, `set`

  - **Control Structures**:

    - **If-Else**:

```
if condition:
    # code
elif another_condition:
    # code
else:
    # code
```

    - **Loops**:

```
for i in range(5):
    # code
while condition:
    # code
```

- **Libraries**:

  - **NumPy**: Numerical operations, arrays

```
import numpy as np
arr = np.array([1, 2, 3])
```

  - **Pandas**: Data manipulation, DataFrames

```
import pandas as pd
df = pd.DataFrame({
    'A': [1, 2, 3],
    'B': [4, 5, 6]
})
```

  - **Matplotlib**: Data visualization

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3], [4, 5, 6])
plt.show()
```

  - **Seaborn**: Statistical data visualization

```
import seaborn as sns
sns.scatterplot(x='A', y='B', data=df)
```

Data Collection

- **Web Scraping**:

  - **BeautifulSoup**: HTML parsing

```
from bs4 import BeautifulSoup
import requests
url = 'http://example.com'
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')
```

  - **Selenium**: Automated browser interactions

```
from selenium import webdriver
driver = webdriver.Chrome()
driver.get('http://example.com')
```

- **APIs**:

  - **Requests Library**: HTTP requests

```
import requests
response = requests.get('http://api.example.com/data')
data = response.json()
```

Data Cleaning

- **Handling Missing Data**:

  - **Drop Missing Values**: `df.dropna()`

  - **Fill Missing Values**: `df.fillna(value)`

- **Data Transformation**:

  - **Normalization**: `(df - df.min()) / (df.max() - df.min())`

  - **Standardization**: `(df - df.mean()) / df.std()`

- **Data Encoding**:

  - **One-Hot Encoding**: `pd.get_dummies(df['column'])`

  - **Label Encoding**: `df['column'].map({'A': 1, 'B': 2})`

Data Analysis

- **Descriptive Statistics**:

  - **Mean**: `df.mean()`

  - **Median**: `df.median()`

  - **Mode**: `df.mode()`

  - **Standard Deviation**: `df.std()`

- **Correlation**:

  - **Pearson Correlation**: `df.corr()`

- **Spearman Correlation**: `df.corr(method='spearman')`

- **Grouping and Aggregation**:

  - **GroupBy**: `df.groupby('column').agg({'column': 'mean'})`

  - **Pivot Tables**: `pd.pivot_table(df, index='column', values='value')`

Data Visualization

- **Matplotlib**:

  - **Line Plot**: `plt.plot(x, y)`

  - **Bar Plot**: `plt.bar(x, y)`

  - **Scatter Plot**: `plt.scatter(x, y)`

- **Seaborn**:

  - **Heatmap**: `sns.heatmap(df.corr())`

  - **Pair Plot**: `sns.pairplot(df)`

  - **Box Plot**: `sns.boxplot(x='column', y='value', data=df)`

Machine Learning

- **Supervised Learning**:

  - **Linear Regression**:

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

  - **Decision Trees**:

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
```

- **Unsupervised Learning**:

  - **K-Means Clustering**:

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3)
model.fit(X)
```

- **PCA**:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
```

- **Model Evaluation**:
  - **Cross-Validation**: `from sklearn.model_selection import cross_val_score`
  - **Confusion Matrix**: `from sklearn.metrics import confusion_matrix`
  - **ROC Curve**: `from sklearn.metrics import roc_curve`

Model Deployment

- **Flask**:
  - **Basic Flask App**:

```
from flask import Flask, request, jsonify
app = Flask(__name__)

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json(force=True)
    prediction = model.predict([data['input']])
    return jsonify({'prediction': prediction.tolist()})

if __name__ == '__main__':
    app.run(debug=True)
```

- **Docker**:
  - **Dockerfile**:

```
FROM python:3.8
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

- **Build and Run**:

```
docker build -t myapp .
docker run -p 5000:5000 myapp
```

Tools and Platforms

- **Jupyter Notebook**:

- **Magic Commands**:

  - **Run Shell Commands**: `!ls`

  - **Time Execution**: `%timeit function()`

  - **Load Extensions**: `%load_ext extension_name`

- **IBM Watson Studio**:

- **Create Projects**: Manage data, notebooks, and models

- **Collaborate**: Share projects with team members

- **Deploy Models**: Use Watson Machine Learning for deployment

Best Practices

- **Version Control**: Use Git for tracking changes

- **Documentation**: Write clear and concise documentation

- **Testing**: Regularly test code and models

- **Ethics**: Ensure data privacy and ethical use of data

Resources

- **Books**:

- "Python for Data Analysis" by Wes McKinney
- "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" by Aurélien Géron

- **Online Courses**:

- Coursera: IBM Data Science Professional Certificate
- edX: Data Science MicroMasters

- **Communities**:

- Stack Overflow
- Kaggle Forums
- Reddit: r/datascience

By Ahmed Baheeg Khorshid

ver 1.0