DATA SCIENCE BRAIN
@datasciencebrain

DATA SCIENCE

All Process Cheat Sheet

Save for later reference

DATA LOADING

CSV Files:
import pandas as pd

 # Basic CSV read
```
df = pd.read_csv('filename.csv')
```

 # Specify column names
```
df = pd.read_csv('filename.csv',
header=None, names=['col1', 'col2'])
```

 # Specify index column
```
df = pd.read_csv('filename.csv',
index_col='column_name')
```

 # Handling Date columns
```
df = pd.read_csv('filename.csv',
parse_dates=['date_column'])
```

Excel Files:

 # Basic Excel read
```
df = pd.read_excel('filename.xlsx')
```

 # Specify sheet name
```
df = pd.read_excel('filename.xlsx',
sheet_name='Sheet1')
```

 # Specify column names and index
```
df = pd.read_excel('filename.xlsx', header=0,
index_col='column_name')
```

Other Formats (JSON, SQL, etc.):
 # JSON
```
df = pd.read_json('filename.json')
```

 # SQL Databasefrom sqlalchemy import create_engine
```
engine = create_engine('sqlite:///:memory:')
df = pd.read_sql('SELECT * FROM
table_name', engine)
```

DATA EXPLORATION & ANALYSIS

General Info:
 # Display first n rows
```
df.head()
```

 # Display last n rows
```
df.tail()
```

 # Data types and non-null counts
```
df.info()
```

 # Summary statistics
```
df.describe()
```

Descriptive Statistics:
 # Mean of each column
```
df.mean()
```

 # Median of each column
```
df.median()
```

 # Correlation matrix
```
df.corr()
```

Null Values:
 # Count of null values in each column
```
df.isnull().sum()
```

 # Drop rows with any null values
```
df.dropna()
```

 # Fill null values with a specific value
```
df.fillna(value)
```

More Exploratory Data Analysis (EDA):
 # Value counts for a categorical variable
```
df['column'].value_counts()
```

 # Unique values in a column
```
df['column'].unique()
```

 # Cross-tabulation between two columns
```
pd.crosstab(df['column1'], df['column2'])
```

Visualization:
**** LOOK A PART 4 ****

DATA CLEANING

Handling Missing Values:

 # Interpolate missing values
```
df.interpolate()
```

 # Fill missing values with a specific value
```
df.fillna(value)
```

 # Drop rows with any missing values
```
df.dropna()
```

 # Drop columns with any missing values
```
df.dropna(axis=1)
```

Dropping Columns:

 # Drop columns by name
```
df.drop(['col1', 'col2'], axis=1, inplace=True)
```

 # Drop columns containing a specific pattern
```
df.drop(df.columns[df.columns.str.contains('pattern')], axis=1, inplace=True)
```

Data Transformation:

 # Convert categorical variable to dummy/indicator variables
```
pd.get_dummies(df['categorical_column'])
```

 # Apply a function to each element in a column
```
df['column'] = df['column'].apply(lambda x: function(x))
```

 # Replace values in a column
```
df['column'].replace({'old_value': 'new_value'}, inplace=True)
```

Outliers:
```
# Detect and handle outliers (e.g., using Z-score)

from scipy import stats

z_scores = stats.zscore(df['column'])
df_no_outliers = df[(z_scores < 3) & (z_scores > -3)]
```

DATA VISUALIZATION

Matplotlib for Basic Plots:
```
import matplotlib.pyplot as plt
```

 # Bar chart
```
plt.bar(df['category'], df['value'], color='blue')
```

 # Boxplot
```
plt.boxplot(df['value'])
```

 # Line chart
```
plt.plot(df['x'], df['y'])
```

Seaborn for EDA:
```
import seaborn as sns
```

 # Count plot for categorical variable
```
sns.countplot(x='category', data=df)
```

 # Scatter plot matrix
```
sns.pairplot(df)
```

 # Heatmap for correlation
```
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
```

Advanced Plots:

 # Violin plot
```
sns.violinplot(x='category', y='value', data=df)
```

 # Joint plot for bivariate analysis
```
sns.jointplot(x='x', y='y', data=df, kind='scatter')
```

 # FacetGrid for multi-plot grids
```
g = sns.FacetGrid(df, col='category', margin_titles=True)
g.map(plt.scatter, 'x', 'y', color='blue')
```

MACHINE LEARNING

Train-Test Split:
from sklearn.model_selection import train_test_split

 # Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

Choose a Model:
Linear Models:
- **Linear Regression:** from sklearn.linear_model import LinearRegression
- **Logistic Regression:** from sklearn.linear_model import LogisticRegression

Tree-based Models:
- **Decision Trees:** from sklearn.tree import DecisionTreeClassifier
- **Random Forest:** from sklearn.ensemble import RandomForestClassifier
- **Gradient Boosting:** from sklearn.ensemble import GradientBoostingClassifier

Support Vector Machines:
- **Support Vector Classifier (SVC):** from sklearn.svm import SVC

Nearest Neighbors:
- **K-Nearest Neighbors (KNN)**: from sklearn.neighbors import KNeighborsClassifier

Naive Bayes:
- **Gaussian Naive Bayes:** from sklearn.naive_bayes import GaussianNB

Clustering:
- **K-Means:** from sklearn.cluster import KMeans

Neural Networks:
- **Multi-layer Perceptron (MLP):** from sklearn.neural_network import MLPClassifier

Ensemble Methods:
- **AdaBoost:** from sklearn.ensemble import AdaBoostClassifier
- Bagging: from sklearn.ensemble import BaggingClassifier

Dimensionality Reduction:
- **Principal Component Analysis (PCA):** from sklearn.decomposition import PCA

Text Processing (for Natural Language Processing):
- **TF-IDF Vectorizer:** from sklearn.feature_extraction.text import TfidfVectorizer
- **Count Vectorizer**: from sklearn.feature_extraction.text import CountVectorizer

MACHINE LEARNING

Train the Model:
 # Initialize the model
model = RandomForestClassifier()
 # Fit the model to the training data
model.fit(X_train, y_train)

Evaluate the Model:
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

 # Accuracy score
print(accuracy_score(y_test, predictions))

 # Classification report
print(classification_report(y_test, predictions))

 # Confusion matrix
print(confusion_matrix(y_test, predictions))

Regression Evaluation Metrics:
Mean Absolute Error (MAE):
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_true, y_pred)

Root Mean Squared Error (RMSE):
from sklearn.metrics import mean_squared_error
rmse = mean_squared_error(y_true, y_pred, squared=False)

R-squared (R2 Score):
from sklearn.metrics import r2_score
r2 = r2_score(y_true, y_pred)

Mean Squared Logarithmic Error (MSLE):
from sklearn.metrics import mean_squared_log_error
msle = mean_squared_log_error(y_true, y_pred)

Explained Variance Score:
from sklearn.metrics import explained_variance_score
evs = explained_variance_score(y_true, y_pred)

Median Absolute Error:
from sklearn.metrics import median_absolute_error
medae = median_absolute_error(y_true, y_pred)

MACHINE LEARNING

Cross-Validation:

```
from sklearn.model_selection import cross_val_score

# Perform cross-validation
scores = cross_val_score(model, X, y, cv=5)
```

Hyperparameter Tuning:

```
from sklearn.model_selection import GridSearchCV

# Define a parameter grid
param_grid = {'n_estimators': [50, 100, 200], 'max_depth': [None, 10, 20]}

# GridSearchCV for hyperparameter tuning
grid = GridSearchCV(model, param_grid, cv=5)
grid.fit(X_train, y_train)

# Get the best parameters
best_params = grid.best_params_
```

8. Model Saving and Loading:

```
import joblib

# Save the model
joblib.dump(model, 'model.pkl')

# Load the model
loaded_model = joblib.load('model.pkl')
```

DEEP LEARNING

Install TensorFlow:

```
pip install tensorflow
```

Import TensorFlow:

```
import tensorflow as tf
```

Build a Neural Network:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation

# Define a Sequential model
model = Sequential()

# Add layers to the model
model.add(Dense(units=128, activation='relu', input_shape=(input_dim,)))
model.add(Dropout(0.5))
model.add(Dense(units=64, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(units=output_dim, activation='softmax'))
```

Compile the Model:

```
# Compile the model with an optimizer, loss function, and metrics
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

DEEP LEARNING

Train the Model:

```
# Train the model with training data
model.fit(X_train, y_train, epochs=10,
batch_size=32, validation_split=0.2)
```

Evaluate the Model:

```
# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(X_test,
y_test)
print(f'Test Accuracy: {test_acc}')
```

Convolutional Neural Network (CNN):

```
from tensorflow.keras.layers import Conv2D,
MaxPooling2D, Flatten

# Example CNN architecture
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu',
input_shape=(img_height, img_width,
channels)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(units=128, activation='relu'))
model.add(Dense(units=output_dim,
activation='softmax'))
```

Recurrent Neural Network (RNN):

```
from tensorflow.keras.layers import
Embedding, LSTM

# Example RNN architecture for sequence
data
model = Sequential()
model.add(Embedding(input_dim=vocab_size,
output_dim=embedding_dim,
input_length=max_seq_length))
model.add(LSTM(units=50,
return_sequences=True))
model.add(LSTM(units=50))
model.add(Dense(units=output_dim,
activation='softmax'))
```

MODEL SAVING AND LOADING

Model Saving and Loading in scikit-learn:

```
import joblib

# Save the scikit-learn model
joblib.dump(model, 'model.pkl')

# Load the scikit-learn model
loaded_model =
joblib.load('model.pkl')
```

Model Saving and Loading in TensorFlow:

```
# Save the entire TensorFlow
model (including architecture,
optimizer, and learned weights)
model.save('tensorflow_model')
```

Load a Model:

```
loaded_model =
tf.keras.models.load_model('tensorfl
ow_model')
```

DEPLOYMENT

scikit-learn Model Deployment:
Once you've trained and saved your scikit-learn model (model.pkl), you can deploy it using various methods depending on your deployment environment.

Flask API:
You can create a simple Flask API to serve your scikit-learn model. Use the flask library to set up an API that receives input data, passes it through the model, and returns predictions.

```
from flask import Flask, request, jsonify
import joblib

app = Flask(_name_)
model = joblib.load('model.pkl')

@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(port=5000)
```

Docker Container:
You can package your Flask API into a Docker container for easy deployment and scalability.

DEPLOYMENT

TensorFlow Model Deployment:
For TensorFlow models, you can use TensorFlow Serving or deploy them as part of a web application.

TensorFlow Serving:
TensorFlow Serving is a system for serving machine learning models in production environments. You can export your TensorFlow model in the SavedModel format and use TensorFlow Serving to deploy it.

```
# Save the TensorFlow model in the SavedModel format
model.save('path_to_saved_model',
save_format='tf')
```

Follow the TensorFlow Serving documentation for setting up a server and making predictions.

Flask API for TensorFlow Model:
Similar to scikit-learn, you can use Flask to create an API for serving TensorFlow models.

```
from flask import Flask, request, jsonify
import tensorflow as tf

app = Flask(_name_)
model =
tf.keras.models.load_model('tensorflow_model')

@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(port=5000)
```