Your Name: Hafiz Muhammad Arslan Ashraf

Course: HCCDA-Artificial Intelligence

Python Labs

Lab 1: Lab Name Python Functions,list,dictionary, loops, condition

Code:

```
# Q1: Function that does NOT return a value

def print_receipt(customer_name, purchased_item):
    print("===== RECEIPT =====")
    print(f"Customer Name : {customer_name}")
    print(f"Purchased Item: {purchased_item}")
    print("====================")

print_receipt("Arslan", "Laptop")
# Q2: Function that RETURNS a value

def calculate_zakat(total_savings):
    zakat = total_savings * 0.025
    return zakat

# Driver code
savings = float(input("Enter your total savings: "))
zakat_amount = calculate_zakat(savings)
print(f"Your calculated Zakat is: {zakat_amount}")

# Q3: Sentence processing

import string

sentence = input("Enter a sentence: ")

# Words in reverse order
words = sentence.split()
reversed_words = ' '.join(reversed(words))

# Capitalize first letter of each word
capitalized = sentence.title()

# Remove punctuation
cleaned = sentence.translate(str.maketrans('', '', string.punctuation))

print("Reversed Words: ", reversed_words)
print("Capitalized Words: ", capitalized)
print("Without Punctuation: ", cleaned)
```

```
# Q4: Update product price in dictionary

products = {
    'Laptop': 70000,
    'Mouse': 1500,
    'Keyboard': 2500,
    'Monitor': 20000
}

product_name = input("Enter the product name to update: ")

if product_name in products:
    new_price = float(input(f"Enter new price for {product_name}: "))
    products[product_name] = new_price
    print("Updated product list:", products)
else:
    print("Error: Product not found in the list.")
# Q5: Remove key-value pairs if value is duplicated under another key

data = {'a': 100, 'b': 200, 'c': 100, 'd': 300, 'e': 200}
seen_values = set()
cleaned = {}

for key, value in data.items():
    if value not in seen_values:
        cleaned[key] = value
        seen_values.add(value)

print("Dictionary after removing duplicate values:", cleaned)
```

Lab 2: Lab Name Numpy OOP

Code:
```
def withdraw_amount():
    try:
        # Take input from the user
        user_input = input("Enter the amount you want to withdraw: ")

        # Try to convert input to float
        amount = float(user_input)

        # Check for negative amount
        if amount <= 0:
            raise ValueError("Withdrawal amount must be greater than zero.")

        print(f"Transaction successful! You have withdrawn ${amount:.2f}")

    except ValueError as ve:
        print(f"Invalid input: {ve}")

    except TypeError:
        print("Type error occurred! Please enter a valid number.")

# Run the function
withdraw_amount()
def calculate_average(marks_list):
    valid_marks = []

    for mark in marks_list:
        try:
            # Attempt to convert mark to float
            valid_mark = float(mark)

            # Optionally check for realistic mark range (e.g. 0 to 100)
            if 0 <= valid_mark <= 100:
                valid_marks.append(valid_mark)
            else:
                print(f"Skipped invalid mark (out of range): {mark}")

        except (ValueError, TypeError):
            print(f"Skipped invalid mark (not a number): {mark}")

    if valid_marks:
        average = sum(valid_marks) / len(valid_marks)
        print(f"\nValid Marks: {valid_marks}")
        print(f"Average of valid marks: {average:.2f}")
    else:
        print("No valid marks available to calculate average.")

# Example list of student marks (some valid, some invalid)
student_marks = [95, '85', '', 78, 'NaN', 101, 'A+', None, 65.5, '88.5']

# Run the function
```

```
calculate_average(student_marks)
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author
        self.available = True  # Book is available by default

    def borrow_book(self):
        if self.available:
            self.available = False
            print(f"You have successfully borrowed '{self.title}' by {self.author}.")
        else:
            print(f"Sorry, '{self.title}' is currently not available.")

    def return_book(self):
        if not self.available:
            self.available = True
            print(f"You have successfully returned '{self.title}'.")
        else:
            print(f"'{self.title}' was not borrowed.")

    def book_status(self):
        status = "Available" if self.available else "Not Available"
        print(f"Book: '{self.title}' | Author: {self.author} | Status: {status}")


# Example usage
book1 = Book("Python Programming", "John Zelle")
book2 = Book("Data Structures", "Mark Allen Weiss")

# Display status
book1.book_status()
book2.book_status()

# Try borrowing books
book1.borrow_book()
book1.borrow_book()  # Try borrowing again

# Return book
book1.return_book()
book1.return_book()  # Try returning again

# Final status
book1.book_status()
import numpy as np

# Sample sales data for 12 months
monthly_sales = np.array([1200, 1500, 1100, 1700, 1300, 1600, 900, 1800, 1400, 1250, 1000, 1550])

# Step 1: Calculate average sales
average_sales = np.mean(monthly_sales)
print("Average monthly sales:", average_sales)
```

```
# Step 2: Find months with sales below average
below_avg_condition = monthly_sales < average_sales

# Step 3: Apply 10% discount to those months
monthly_sales[below_avg_condition] *= 0.9

# Step 4: Display updated sales
print("Updated monthly sales after 10% discount on below-average months:")
print(np.round(monthly_sales, 2))
```

Lab 3: Lab Name Pandas

Code:

```
 # Handling Missing Values

import pandas as pd
import numpy as np

# Sample DataFrame with missing values
df = pd.DataFrame({'A': [None, 3 , np.NAN, 7],
                   'B': [ np.nan,5, 8, np.NaN],
                   'C': [9, 10, 11, 12]})
print(df)
print()
# Check for missing values
print(df.isnull())
print(df.isna())
print("Check for missing values and Sum it ")
print(df.isnull().sum())
print(df.isna().sum())

# Fill missing values with a specific value
df_filled = df.fillna(0)
print("Fill missing values with a specific value = 0")
print(df_filled)

# Fill missing values using forward fill
df_ffill = df.fillna(method='ffill')
print("Fill missing values using forward fill")
print(df_ffill)

# New method for ffill df.ffill()
print("df.ffill()")
print(df.ffill())

# Fill missing values using backward fill
df_bfill = df.fillna(method='bfill')
print("Fill missing values using backward fill")
print(df_bfill)

# New method for bfill df.bfill()
print("df.bfill()")
print(df.bfill())
```

Q2

```
import pandas as pd
file_path = '/content/drive/MyDrive/testdataset.csv' # 'My Drive' refers to the root directory of Google Drive.
df = pd.read_csv(file_path)

print(df.head())
# Handling Missing Values
# Dropping rows with missing values
print("Dropping rows with missing values")
df_dropped = df.dropna()
print(df_dropped)
print()
# Unique values in a column
print("Unique Valus in Column")
print(df['Occupation'].unique())
print()
# Getting column names
print(df.columns)
print()

# Getting index names
print(df.index)
print()
# Ordering by a column (descending)
print("Sort by GPA Descending")
print(df.sort_values('Education_Level', ascending=False))
print()

# Checking for null values
print("Checking for null values")
print(df.isnull().sum())
print()
# Fill missing values using forward fill
df_ffill = df.fillna(method='ffill')
print("Fill missing values using forward fill")
print(df_ffill)
df_bfill = df.fillna(method='bfill')
print("Fill missing values using backward fill")
print(df_bfill)

# New method for bfill df.bfill()
print("df.bfill()")
print(df.bfill())
```

Lab 4: Lab Name Matplotlib

Code :

```
from google.colab import drive # Import 'drive' module from the 'google.colab' library
drive.mount('/content/drive') # This mounts your Google Drive into the Colab virtual environment.

import pandas as pd

file_path = '/content/drive/MyDrive/Titanicdataset.csv' # 'My Drive' refers to the root directory of Google Drive.
df = pd.read_csv(file_path)

print(df.head())
import matplotlib.pyplot as plt

# Count frequency of each category in 'Sex' column
sex_counts = df['Sex'].value_counts()

# Create bar plot using Matplotlib
plt.figure(figsize=(6, 4))
plt.bar(sex_counts.index, sex_counts.values, color=['skyblue', 'lightpink'])
plt.title('Count of Passengers by Gender')
plt.xlabel('Gender')
plt.ylabel('Number of Passengers')
plt.tight_layout()
plt.show()
# Create a scatter plot between 'Age' and 'Fare'
plt.figure(figsize=(7, 5))
plt.scatter(df['Age'], df['Fare'], alpha=0.6, color='teal')
plt.title('Scatter Plot: Age vs Fare')
plt.xlabel('Age')
plt.ylabel('Fare')
plt.grid(True)
plt.tight_layout()
plt.show()
# Plot histogram
plt.figure(figsize=(6, 4))
plt.hist(df['Age'].dropna(), bins=20, color='skyblue', edgecolor='black')
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
#line plot
# Drop missing values in Age for clean plotting
age_data = df['Age'].dropna()

# Create a simple line plot
plt.figure(figsize=(6, 4))
plt.plot(age_data.values, color='orange')
plt.title('Line Plot of Passenger Age')
plt.xlabel('Passenger Index')
plt.ylabel('Age')
plt.grid(True)
plt.tight_layout()
```

```
plt.show()
pclass_counts = df['Pclass'].value_counts()
plt.figure(figsize=(6, 6))
plt.pie(pclass_counts, labels=pclass_counts.index, autopct='%1.1f%%', startangle=90, colors=['lightblue', 'lightgreen',
'lightcoral'])
plt.title('Passenger Class Distribution')
plt.axis('equal')
plt.show()
# Remove missing values from 'Age' column
age_data = df['Age'].dropna()

# Create a box plot
plt.figure(figsize=(6, 4))
plt.boxplot(age_data)
plt.title('Box Plot of Age')
plt.ylabel('Age')
plt.xticks([1], ['Age'])  # Label the x-axis
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Q2 : import pandas as pd
import matplotlib.pyplot as plt

# Step 1: Load the dataset
file_path = '/content/drive/MyDrive/IRIS.csv' # Use the correct file path from Google Drive
df = pd.read_csv(file_path)

# Step 2: Create a 2x3 subplot layout
plt.figure(figsize=(15, 10))

# 1Histogram of Sepal Length
plt.subplot(2, 3, 1)
plt.hist(df['sepal_length'], bins=10, color='skyblue', edgecolor='black')
plt.title('Histogram: Sepal Length')
plt.xlabel('Sepal Length')
plt.ylabel('Frequency')

# 2Line Plot of Sepal Width
plt.subplot(2, 3, 2)
plt.plot(df['sepal_width'], color='green', marker='o', linestyle='-')
plt.title('Line Plot: Sepal Width')
plt.xlabel('Index')
plt.ylabel('Sepal Width')

# 3Pie Chart of Species Distribution
plt.subplot(2, 3, 3)
species_counts = df['species'].value_counts()
plt.pie(species_counts.values, labels=species_counts.index, autopct='%1.1f%%', startangle=90, colors=['gold', 'lightcoral',
'lightskyblue'])
plt.title('Pie Chart: Species Distribution')
plt.axis('equal')  # to make the pie chart round
```

```
# 4Box Plot of Petal Length
plt.subplot(2, 3, 4)
plt.boxplot(df['petal_length'], patch_artist=True, boxprops=dict(facecolor='lightgreen'))
plt.title('Box Plot: Petal Length')
plt.ylabel('Petal Length')

# 5Bar Chart: Average Sepal Width per Species
plt.subplot(2, 3, 5)
avg_sepal_width = df.groupby('species')['sepal_width'].mean()
plt.bar(avg_sepal_width.index, avg_sepal_width.values, color='mediumpurple')
plt.title('Bar Chart: Avg Sepal Width per Species')
plt.ylabel('Average Sepal Width')

# 6Scatter Plot: Sepal Length vs Petal Length
plt.subplot(2, 3, 6)
plt.scatter(df['sepal_length'], df['petal_length'], c='tomato')
plt.title('Scatter Plot: Sepal vs Petal Length')
plt.xlabel('Sepal Length')
plt.ylabel('Petal Length')

# Final adjustments
plt.tight_layout()
plt.show()
```

Lab 5: Lab Name ML Linear Regression

```
# Import required libraries
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt
import numpy as np

# Step 1: Create a sample dataset using pandas
data = {
   'Experience': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
   'Salary': [35000, 39000, 43000, 48000, 52000, 57000, 61000, 65000, 69000, 73000]
}
df = pd.DataFrame(data)

# Step 2: Define features (X) and target (y)
X = df[['Experience']]  # Independent variable(s)
y = df['Salary']        # Dependent variable

# Step 3: Split into training and testing sets (80/20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Step 4: Create and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Step 5: Make predictions
y_pred = model.predict(X_test)

# Step 6: Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)


# Step 7: Display performance metrics
print("Model Performance on Test Data:")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")


# Step 8: Optional - Plot actual vs predicted salaries
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Predicted')
plt.title('Linear Regression: Experience vs Salary')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend()
plt.grid(True)
plt.show()
```

Lab 6: Lab Name ML Logistic Regression

```
# Logistic Regression
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay,
classification_report
)

# Sample classification dataset --> Customer will purchase or not given Age and Salary
data = {
    'age': [22, 25, 47, 52, 46, 56, 44, 34, 48, 39, 40, 45, 44],
    'salary': [20000, 25000, 47000, 52000, 46000, 56000, 44000, 34000, 48000, 39000,45000,50000,51000],
    'purchased': [0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1]     ### 0 = No, 1 = Yes
}

df = pd.DataFrame(data)

# Features and target
X = df[['age', 'salary']]
y = df['purchased']

# Split into train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Model training
model = LogisticRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluation metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Confusion matrix visualization
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()
```

```
Q2 import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
from sklearn.preprocessing import LabelEncoder
import seaborn as sns

# Load Titanic dataset
titanic = sns.load_dataset('titanic')

# Drop rows with missing target values
titanic = titanic.dropna(subset=['age', 'embarked', 'sex'])

# Select features and target
X = titanic[['pclass', 'sex', 'age', 'sibsp', 'parch', 'fare', 'embarked']]
y = titanic['survived']

# Encode categorical features
X['sex'] = LabelEncoder().fit_transform(X['sex'])            # male=1, female=0
X['embarked'] = LabelEncoder().fit_transform(X['embarked'])  # C=0, Q=1, S=2

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Model training
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

# Prediction
y_pred = model.predict(X_test)

# Evaluation
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("\nAccuracy Score:", accuracy_score(y_test, y_pred))
```

Lab 7: Lab Name ML Decision Tree

Code:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Convert to DataFrame for better visualization
df = pd.DataFrame(data=np.c_[iris['data'], iris['target']],
                  columns=iris['feature_names'] + ['target'])

#print(df.head())

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Initialize the Decision Tree Classifier
# by default use 'gini'
#clf = DecisionTreeClassifier(random_state=42)
clf = DecisionTreeClassifier(criterion='entropy',max_depth=3)

# Train the model
clf.fit(X_train, y_train)
plt.figure(figsize=(20,15))
plot_tree(clf, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
plt.show()
# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))

print("\nAccuracy Score:")
print(accuracy_score(y_test, y_pred))
```

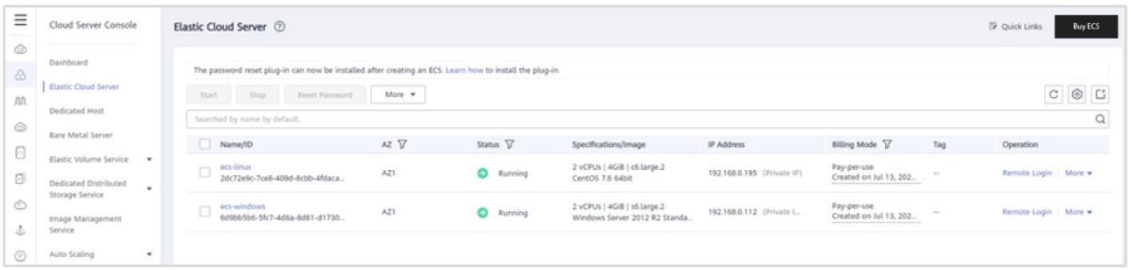
Lab 8: Lab Name ML Random Forest

Code:
```
# Importing necessary libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
data = load_iris()
X = data.data
y = data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)

# Make predictions
y_pred = rf_model.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

print(f'Random Forest Model Accuracy: {accuracy:.2f}')

# Displaying feature importance
importances = rf_model.feature_importances_
feature_names = data.feature_names
for feature, importance in zip(feature_names, importances):
    print(f'Feature: {feature}, Importance: {importance:.4f}')
```

KooLabs

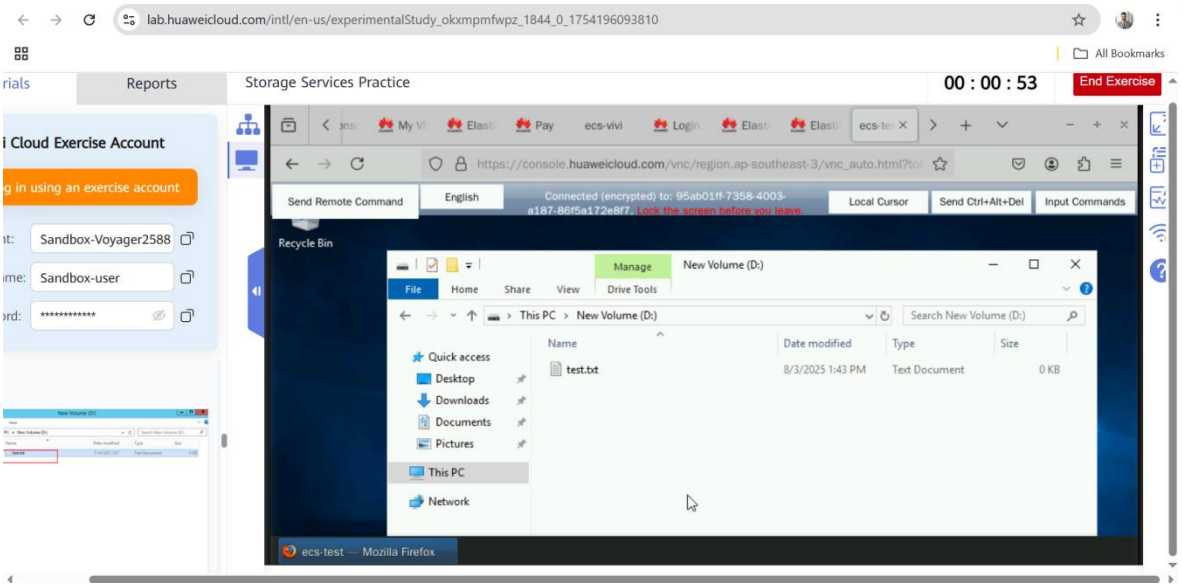
Lab 1: Lab Name : Compute Services Practice

Brief Summary: In the **Huawei Cloud Compute Services Practice (Windows ECS)**, you first create a **Virtual Private Cloud (VPC)** with a subnet and security group to define the network environment. Then, you create a **Windows-based ECS (Elastic Cloud Server)**, assign a **public IP**, configure **security rules like RDP port 3389)**, and finally **access the server remotely via Remote Desktop Protocol (RDP)**. This teaches you how to set up and manage a secure cloud-based virtual machine.



Lab 2: Lab Name: Storage Services Practice

Brief Summary: In the **Huawei Cloud Storage Services Practice**, you first create a **VPC**, then purchase an **ECS instance**, and create an **EVS (Elastic Volume Service) disk**. You **attach the disk** to the ECS, **log in remotely**, initialize and format the disk. Later, you **detach the disk**, create another ECS named **ECSTest**, attach the disk to it, log in, and create a test file (test.txt) to verify the storage. This practice demonstrates how to manage block storage across multiple ECS instances.



Lab 3: Lab Name: Authentication of IamUser

Brief Summary: In this exercise, you log into your **Huawei Cloud root account**, create new **users**, assign them permissions, and download their **access credentials (Access Key & Secret Key)**. Then, using **Postman**, you perform authentication by sending a **POST request** with an authentication JSON body and appropriate **headers** Key(like `Content-Type` and `Accept and Value Application/json`). This allows you to generate an authentication **token** for the IAM user, which can be used to securely call Huawei Cloud APIs.