CS 210: Data Management for Data Science

Final Practice Exam

Fall 2023

Name: _____ NetID: _____

This is a closed book, closed notes exam, only 7 pages of HAND WRITTEN NOTES allowed.

No electronic devices are permitted.

Name: _____

1. (10 points) Write a Python function `inverse_dictionary` that takes a dictionary as input and returns a new dictionary where the keys and values are swapped. For example, `inverse_dictionary({'a': 1, 'b': 2, 'c': 3})` should return `{1: 'a', 2: 'b', 3: 'c'}`.

```
def inverse_dictionary(input_dict):
    #Empty dictionary to store the inverted key value pairs
    inverted_dict = {}

    #Iterate through the key value pairs in dictionary
    for key, value in input_dict.items():
        #Swap key and value and add the pair to the dictionary
        inverted_dict[value] = key

    #Return the inverted dictionary as the result
    return inverted_dict

#Taking the example for testing
original_dict = {'a': 1, 'b': 2, 'c': 3}

#Calling the inverse_dictionary function
result_dict = inverse_dictionary(original_dict)

#Printing the result dictionary
print(result_dict)
```

Name: _____

2. (10 points) Write a Python function **odd_even_sums** that takes a list of numbers as input and returns a tuple containing the sum of odd numbers and the sum of even numbers in the list. For example, `odd_even_sums([1, 2, 3, 4, 5, 6])` should return (9, 12).

```
def odd_even_sums(numbers):
    #Initialize sum variables for odd and even nums
    odd_sum = 0
    even_sum = 0

    # Iterate through the numbers in the list
    for num in numbers:
        # Check if the number is odd or even
        if num % 2 == 0:
            even_sum += num
        else:
            odd_sum += num

    # Return a tuple with the sum of odd numbers and even numbers
    return (odd_sum, even_sum)

#Testing the code:
numbers_list = [1, 2, 3, 4, 5, 6]
result_tuple = odd_even_sums(numbers_list)

print(result_tuple)
```

Name: _____

3. (10 points) Given a DataFrame `df` with columns `['employee', 'experience', 'department']`

   a. Write a Pandas script to double the experience for employees in the `IT` department.

```
df.loc[df['department'] == 'IT', 'experience'] *= 2
```

Department	**Average Experience**
IT	X.XX
Finance	Y.YY
Marketing	Z.ZZ

```
avg_experience = df.groupby('department')['experience'].mean()
```

Name: _____

4. (10 points) Given a Pandas DataFrame `df` with columns `['product', 'quantity', 'price']`.

   a. Write a Pandas command to remove rows with missing values in the `price` column.

```
#inplace=True modifies the original df instead of creating a new one
df.dropna(subset=['price'], inplace=True)
```

   b. Create a new column **TotalCost**, by multiplying `quantity` and `price` for each product.

```
df['TotalCost'] = df['quantity'] * df['price']
```

Name: _____

5. (10 points) Given a dataset with columns `timestamp` and `sales_amount`

   a. Using Matplotlib, write code to create a scatter plot showing the sales amount over time.

```
#Extract timestamp and sales_amount columns
timestamps = pd.to_datetime(data["timestamp"])
sales_amounts = data["sales_amount"]

#Create the scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(timestamps, sales_amounts)

#Set labels and title
plt.xlabel("Timestamp")
plt.ylabel("Sales Amount")
plt.title("Sales Amount over Time")

plt.show
```

   b. Enhance the scatter plot with a title, axis labels, and color the data points differently based on whether the sales amount is above or below the median.

```
#Extract timestamp and sales_amount columns
timestamps = pd.to_datetime(data["timestamp"])
sales_amount = data["sales_amount"]

#Calculate the median sales amount
median_sales = sales_amounts.median()

#New col 'color' on whether sales amt is above or below the median
df['color'] = df['sales_amount']
.apply(lambda x: 'green' if x > median_sales else 'red')

#Plot the scatter plot with colored data points
plt.figure(figsize=(10, 6))
plt.scatter(df['timestamp'], df['sales_amount'],
color=df['color'], marker='o')

#Adding labels and title
plt.title('Sales Amount Over Time with Color')
plt.xlabel('Timestamp')
plt.ylabel('Sales Amount')

#Display the plot
plt.show()
```

Name: _____

6. (10 points) You are given a Pandas DataFrame customer_df with columns 'Date (in DD-MM-YYYY format),' and 'NewCustomers'

   a. Write code to create a line plot using Matplotlib to show the total NewCustomers acquired by the company for each month. Aggregate the customer data by month, and ensure the x-axis represents the month and year (e.g., Jan 2023).

```
#Convert the date to datetime with month and year
customer_df["MonthYear"] = customer_df["Date"].dt.strftime("%b %Y")

#Aggregate the data by month and sum the NewCustomers
monthly_custs = customer_df.groupby("MonthYear")["NewCustomers"].sum()

#Create the line plot
plt.figure(figsize=(10, 6))
plt.plot(monthly_custs.index, monthly_custs.values)

#labels and title
plt.xlabel("Month Year")
plt.ylabel("Total New Customers")
plt.title("Total New Customers Acquired by Month")

# Show the plot
plt.tight_layout()
plt.show()
```

   b. Using Matplotlib, create a pie chart that illustrates the distribution of NewCustomers among the top 6 dates with highest NewCustomers acquired.

```
#Sort by NewCustomers
sorted_df = customer_df.sort_values(by="NewCustomers", ascending=False)

#Top 6 dates and their NewCustomers
top_dates = sorted_df["Date"].head(6)
top_customers = sorted_df["NewCustomers"].head(6)

#Create the pie chart
plt.figure(figsize=(8, 8))
plt.pie(top_customers, labels=top_dates.dt.strftime("%b %d"),
autopct="%1.1f%%")
plt.title("Distribution of New Customers (Top 6 Dates)")

#Show the pie chart
plt.show()
```

Name: _____

7. (15 points) You are working with a database containing a single table named **Projects**.

   - **Projects** table:
     - `ProjectID` (integer, primary key)
     - `ProjectName` (string)
     - `Department` (string)
     - `Budget` (integer)

   a. Write an SQL query to find the departments with budget greater than $20,000.

```sql
SELECT DISTINCT Department
FROM Projects
WHERE Budget > 20000;
```

   b. Write an SQL query to retrieve the `ProjectName`, `Department`, and `Budget` for all projects. Exclude projects with a budget less than $5,000. Order the results by `Budget` in descending order.

```sql
SELECT ProjectName, Department, Budget
FROM Projects
WHERE Budget >= 5000
ORDER BY Budget DESC;
```

Name: _____

8. (15 points) You are given a database with two tables: **Inventory** and **Purchases**.

   - **Inventory** table:
     - `ProductID` (integer, primary key)
     - `ProductName` (string)
     - `Category` (string)
     - `StockQuantity` (decimal)
   - **Purchases** table:
     - `PurchaseID` (integer, primary key)
     - `ProductID` (integer, foreign key to Inventory)
     - `PurchaseDate` (date)
     - `Quantity` (integer)

   a. Monthly Purchase Analysis: Write an SQL query to find the total quantity purchased for each product for each month. Display the `ProductName`, month of the `PurchaseDate`, and the total quantity. Order the results by `ProductName` and then by month.

```
SELECT
    i.ProductName,
    MONTH(p.PurchaseDate) AS PurchaseMonth,
    SUM(p.Quantity) AS TotalQuantity
FROM
    Inventory AS i
INNER JOIN
    Purchases AS p ON i.ProductID = p.ProductID
GROUP BY
    i.ProductName, MONTH(p.PurchaseDate)
ORDER BY
    i.ProductName ASC, PurchaseMonth ASC;
```

   b. Top Stocked Categories: Write an SQL query to identify the top 3 stocked categories based on the highest average stock quantity.

```
SELECT
    Category,
    AVG(StockQuantity) AS AverageStockQuantity
FROM
    Inventory
GROUP BY
    Category
ORDER BY
    AverageStockQuantity DESC
LIMIT 3;
```

Name: _____

9. (10 points) Write a regular expression to identify valid phone numbers in the format (XXX) XXX-XXXX. The area code should be in the range from 100 to 999, and the rest of the numbers should be in the range from 000-0000 to 999-9999. Valid numbers include (123) 456-7890, (555) 123-4567, but not (999) 000-00000 or (123) 456-789.

```
r"^\(\d{3}\) \d{3}-\d{4}$"
```

Name: _____

10. (Extra Credit: 10 points) Numpy Question

    a. Given a 2D Numpy array `mat` and a 1D Numpy array `vec`, write a function to add the vector to each row of the matrix. For instance, if `mat` is an array of shape `(3, 4)` and `vec` is an array with value `[1, 2, 3, 4]`, the function should return a modified matrix with each row incremented by the corresponding elements of the vector. import numpy as np

```
def add_vector_to_matrix(mat, vec):
    result = mat + vec
    return result

#Exampe matrix
matrix = np.array([[1, 2, 3, 4],
                   [5, 6, 7, 8],
                   [9, 10, 11, 12]])

vector = np.array([1, 2, 3, 4])

result_matrix = add_vector_to_matrix(matrix, vector)
print(result_matrix)
```

    b. Modify the function to subtract the mean of each column from that column.

```
def subtract_column_mean(mat):
    #Calculating the mean of each column
    column_means = np.mean(mat, axis=0)

    #Subtract the mean of each column from that column
    result = mat - column_means

    return result

# Example matrix:
matrix = np.array([[1, 2, 3, 4],
                   [5, 6, 7, 8],
                   [9, 10, 11, 12]])

result_matrix = subtract_column_mean(matrix)
print(result_matrix)
```

This page is intentionally left blank