Key Exercises and Insights

Week 3: Python Basics

1. Guessing Game:
    - A program prompts users to guess a secret number between 1 and 100 until they guess correctly.
    - Example:
    - `python`

```
secret_number = 22
while True:
    guess = int(input("Guess a number between 1 and 100: "))
    if guess == secret_number:
        print("Congratulations! You guessed the correct
number.")
        break
    else:
        print("Incorrect guess. Try again.")
```

    -
    -
2.
3. Sum of Numbers:
    - A `for` loop calculates the sum of all numbers from 1 to a user-provided positive integer
    - n
    - n.
    - Example: Input
    - n=5
    - n=5, output is
    - 15
    - 15 (sum of
    - 1+2+3+4+5
    - 1+2+3+4+5).
4.
5. Using `zip` for Iterables:

- Combines lists to calculate revenues or display paired information.
- Example: Calculate revenue for products using prices and quantities sold.

6.
7. Even Numbers in a List:
    - A function iterates through a list to print even numbers.
8.
9. Sales Analysis:
    - Calculates total weekly sales and each day's percentage contribution to the total.
10.
11. Student Grades:
    - Assigns letter grades based on numerical scores using specified criteria (A:
    - ≥ 90
    - ≥ 90, B:
    - $80-89$
    - $80-89$, etc.).
12.

Week 4: NumPy and Probability

1. Array Manipulation:
    - Tasks include accessing specific rows/columns, slicing arrays, and counting zeros in an array.
2.
3. Exponential Distribution:
    - Explores customer arrival probabilities at a coffee shop using exponential distribution.
    - Formula for CDF:
    - $F(x;\lambda)=P(X<x)=1-e-\lambda x, x\geq 0$
    - $F(x;\lambda)=P(X<x)=1-e$
    - $-\lambda x$
    - $, x\geq 0$
    -
    - Example questions include calculating probabilities for specific time intervals (e.g., within 2 minutes, between 1 and 3 minutes).
4.

Week 5: Titanic Dataset Analysis

1. General Dataset Description:
    - Shape:
    - 891
    - 891 rows,
    - 12
    - 12 columns.
    - Data types: Numerical (e.g., Age, Fare), Categorical (e.g., Sex, Embarked).
    - Missing values in columns like `Age`, `Cabin`, and `Embarked`.
2.
3. Basic Statistics:
    - Mean age of passengers: ~29.7 years.
    - Survival rate: ~38% survived.
4.
5. Key Observations:
    - More females survived compared to males.
    - Passengers in higher classes (Pclass = 1) had better survival rates.
    - Younger passengers had higher survival rates.
6.
7. Suggestions for Improvement:
    - Drop columns like `Cabin` due to excessive missing data.
    - Group or sort data by features like `Pclass` or `Embarked` for better analysis.
8.
9. Visualizations:
    - Survival count by gender.
    - Age distribution by survival status.
    - Fare distribution across classes and survival status.
    - Correlation heatmap for numerical features.
10.

Week 6: Data Types and Titanic EDA

1. Data Type Classification:
    - Examples include numerical-discrete (shoe size), numerical-continuous (weight), categorical-ordinal (education level), etc.
2.
3. Titanic EDA Tasks:
    - Handling missing values: Fill `Age` with median, `Embarked` with mode, drop `Cabin`.
    - Detecting outliers using IQR for numerical features like Age and Fare.

- Feature selection: Drop irrelevant columns like `PassengerId` and `Ticket`.

4. Visualizations:
    - Survival rate comparisons across categories like gender and embarkation port.
    - Boxplots showing fare differences by class and survival status.

Summary

These exercises cover foundational Python programming concepts, data manipulation with libraries like NumPy, probability concepts, and exploratory data analysis (EDA) using real-world datasets such as Titanic. The tasks emphasize practical skills such as writing loops, handling missing data, visualizing distributions, and deriving insights from datasets.

Key Exercises and Insights

Week 7: Regression Modeling

1. Data Exploration:
    - Explored an advertising dataset containing features like TV, Radio, Newspaper budgets, and Sales.
    - Key steps included detecting missing values, identifying outliers using boxplots, and visualizing distributions.
2.
3. Simple Linear Regression:
    - Built models to predict Sales based on individual features (e.g., Radio or Newspaper budgets).
    - Regression formulas derived::
    - Comments: Radio has a stronger relationship with sales compared to Newspaper.
4.
5. Multiple Linear Regression:
    - Modeled Sales using all three features (TV, Radio, Newspaper).
    - Formula:
    - $Sales = 4.7438 + 0.0536 \times TV + 0.1027 \times Radio + 0.0079 \times Newspaper$
    - $Sales = 4.7438 + 0.0536 \times TV + 0.1027 \times Radio + 0.0079 \times Newspaper$
    - Insights:TV and Radio have significant positive impacts on sales.
6.
7. Comparison:

- Simple regression evaluates variables in isolation, while multiple regression accounts for interactions among predictors.

8.

Week 8: Classification and Clustering

1. Wine Quality Classification:
    - Dataset: Chemical properties of wines with a categorical target (`quality_category`).
    - Steps included:Data exploration (shape, data types, missing values, outliers).
    - Explanation of the decision tree:The root node splits based on alcohol levels.
2.
3. K-Means Clustering:
    - Performed clustering on scaled data using three clusters.
    - Plots compared clustering effectiveness for different variable pairs (e.g., alcohol vs volatile acidity).
    - Comments: Some features separate clusters better than others (e.g., alcohol vs sulphates).
4.

Week 9: Titanic Dataset Classification

1. Preprocessing:
    - Filled missing `Age` values with the mean.
    - Dropped irrelevant columns like `Cabin` and converted `Sex` to binary values (male = 1, female = 0).
2.
3. Decision Tree Classification:
    - Target variable: `Survived`.
    - Split dataset into training (70%) and testing (30%).
    - Evaluated the model using confusion matrix, classification report, ROC curve, and AUC score.
4.
5. Regression Task Suggestion:
    - Suggested independent variables: `Pclass`, `Sex`, `Age`, `SibSp`, `Parch`, `Fare`.
    - Dependent variable: `Survived`.
6.

Week 10: Object-Oriented Programming (OOP)

1. Book Class:
    - Created a class with attributes (`title`, `author`) and a method (`display_info`) to display book details.
    - Example objects: `book1` ("1984"), `book2` ("To Kill a Mockingbird").
2.
3. Rectangle Class:
    - Attributes: `length`, `width`.
    - Methods:: Computes area as
4.
5. BankAccount Class:
    - Attributes: `principal`, `rate`, `time`.
    - Methods:: Computes interest as
6.

Key Exercises and Insights

Object-Oriented Programming (OOP)

Question 1: Class Attributes and Methods

- Class Attributes:
    - Shared across all instances of the class.
    - Example: `total_cars` in the `Car` class tracks the total number of cars created.
-
- Instance Attributes:
    - Unique to each instance of the class.
    - Example: `make` and `year` store specific details about a car.
-
- Methods:
    - Instance Methods: Operate on instance attributes (e.g., `display_info`).
    - Class Methods: Operate on class attributes using the `@classmethod` decorator (e.g., `set_total_cars`, `display_total_cars`).
-

Code Implementation:

```
python
class Car:
    # Class attribute
    total_cars = 0
```

```
    def __init__(self, make, year):
        self.make = make  # Instance attribute
        self.year = year  # Instance attribute

    # Instance method
    def display_info(self):
        print(f"This car is a {self.year} {self.make}.")

    # Class method to set total cars
    @classmethod
    def set_total_cars(cls, count):
        cls.total_cars = count

    # Class method to display total cars
    @classmethod
    def display_total_cars(cls):
        print(f"Total cars set to: {cls.total_cars}")

# Example usage
car1 = Car("Toyota", 2021)
car1.display_info()  # Output: This car is a 2021 Toyota.
car2 = Car("Honda", 2020)
car2.display_info()  # Output: This car is a 2020 Honda.

Car.set_total_cars(100)
Car.display_total_cars()  # Output: Total cars set to: 100
```

Question 2: Inheritance with ElectricCar

- Created a subclass `ElectricCar` that inherits from `Car`.
- Added a new attribute `battery_capacity` to store battery size in kWh.
- Used the parent class's `__init__` method to initialize inherited attributes.

Code Implementation:

```
python
class ElectricCar(Car):
```

```
    def __init__(self, make, year, battery_capacity):
        super().__init__(make, year)  # Call parent class's
__init__ method
        self.battery_capacity = battery_capacity

    # Method specific to ElectricCar
    def display_battery_info(self):
        print(f"This electric car has a
{self.battery_capacity} kWh battery.")

# Example usage
my_electric_car = ElectricCar("Tesla", 2022, 75)
my_electric_car.display_info()  # Output: This car is a 2022
Tesla.
my_electric_car.display_battery_info()  # Output: This electric
car has a 75 kWh battery.

ElectricCar.set_total_cars(150)
ElectricCar.display_total_cars()  # Output: Total cars set to:
150
```

Machine Learning Applications

Question 3: Classification Task for Titanic Dataset

- Goal: Predict survival (`Survived`) using features like `Pclass`, `Sex`, `Age`, etc.

Steps:

1. Data Preprocessing:
    - Filled missing values in `Age` with the mean.
    - Dropped irrelevant columns (`Cabin`, `PassengerId`, etc.).
    - Converted categorical variables (e.g., `Sex`) into numerical values.
2.
3. Modeling and Validation:
    - Used a decision tree classifier.
    - Evaluated the model using confusion matrix, classification report, ROC curve, and AUC score.

4.

Code Implementation:

```
python
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import classification_report,
confusion_matrix, roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# Preprocessing
data['Age'] = data['Age'].fillna(data['Age'].mean())
data.drop(['Cabin', 'PassengerId', 'Name', 'Ticket'], axis=1,
inplace=True)
data['Sex'] = data['Sex'].map({'male': 1, 'female': 0})

X = data.drop('Survived', axis=1)
y = data['Survived']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Train Decision Tree model
clf = DecisionTreeClassifier(max_depth=3)
clf.fit(X_train, y_train)

# Plot decision tree
plt.figure(figsize=(20,10))
plot_tree(clf, feature_names=X.columns, class_names=['Not
Survived', 'Survived'], filled=True)
plt.show()

# Evaluate model
y_pred = clf.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
# ROC Curve and AUC Score
y_pred_prob = clf.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
auc_score = roc_auc_score(y_test, y_pred_prob)

plt.plot(fpr, tpr, label=f"AUC = {auc_score:.2f}")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```

Summary

This upload reinforces key OOP concepts like class attributes/methods and inheritance while introducing machine learning tasks. The exercises demonstrate:

- How to structure reusable code with classes and inheritance.
- Practical applications of decision trees for classification tasks.
- Comprehensive evaluation of machine learning models using metrics like confusion matrix and ROC curves.

These exercises provide an excellent foundation for both programming principles and applied data science.