1. Create a CSV file named `house_prices.csv`:

```
Area,Bedrooms,Age,Distance,Price
1500,3,10,5,910
1800,3,15,6,850
1200,2,8,2.5,420
2000,4,12,2,800
1700,3,15,2.5,690
1500,2,12,7,500
1100,2,7,4.5,400
1600,3,17,1.5,610
1400,3,10,5,550
2100,5,5,2,450
```

2. Python program to read and display the dataset:

```
import pandas as pd

# Read the CSV file
df = pd.read_csv('house_prices.csv')

# Display the entire dataset
print(df)
```

(b) Prepare the data [2.5 Marks]

```
from sklearn.model_selection import train_test_split

# Independent and dependent variables
X = df[['Area', 'Bedrooms', 'Age', 'Distance']]
y = df['Price']

# Split into training and testing sets (80%-20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

(c) Multiple Linear Regression [10 Marks]

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# 1. Build the model
model = LinearRegression()
model.fit(X_train, y_train)

# Display coefficients and intercept
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)

# 2. Evaluate the model
y_pred = model.predict(X_test)

print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))

# Plot Actual vs Predicted
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Prices")
```

```
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted Prices")
plt.grid(True)
plt.show()

# Predict price of new house
new_house = [[1750, 3, 10, 4.0]]
predicted_price = model.predict(new_house)
print("Predicted price for new house:", predicted_price[0])
```

BankAccount Class Design [5 Marks]

```
class BankAccount:
    def __init__(self, account_number, name, balance):
        self.account_number = account_number
        self.name = name
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount

    def withdrawal(self, amount):
        if amount <= self.balance:
            self.balance -= amount
        else:
            print("Insufficient balance!")

    def balance_enquiry(self):
        print(f"Account Balance: {self.balance}")

# Inheritance: SavingsAccount
class SavingsAccount(BankAccount):
    def __init__(self, account_number, name, balance, interest_rate):
        super().__init__(account_number, name, balance)
        self.interest_rate = interest_rate

    def add_interest(self):
        interest = self.balance * self.interest_rate / 100
        self.balance += interest
        print(f"Interest added: {interest}, New Balance: {self.balance}")

# Example usage
account = SavingsAccount("123456", "Ratikanta", 1000, 5)
account.deposit(500)
account.withdrawal(200)
account.add_interest()
account.balance_enquiry()
```