The first question.
The purpose of this exercise is to use the power of linear regression, a statistical algorithm, to predict and quantify the profit of start-up businesses based on the mentioned factors.
As we know, linear regression is a supervised learning algorithm that seeks to establish a linear relationship between independent and dependent variables. In this exercise, the independent variables are various business parameters. The dependent variable or the target variable is the profit from the start-up.
The linear regression model is trained using part of the data set and its performance is evaluated on another set to ensure its ability to generalize to unseen data. Evaluation includes evaluation of metrics such as accuracy and mean squared error (MSE), which quantify the predictive capabilities of the model.
The importance of each independent variable is measured through model coefficients, providing insights into the relative impact of R&D.
Libraries required to implement this exercise:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

The data of this exercise are as follows.

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

We separate the required columns for training and label data.

```python
# Extract features and target variable
X = data[['R&D Spend', 'Administration', 'Marketing Spend', 'State']]
y = data['Profit']
```

The above code is used to select specific columns from the DataFrame.
Y represents the target variable.

```
# One-hot encode the 'State' column
X_encoded = pd.get_dummies(X, columns=['State'], drop_first=True)
```

In the code above, we encode the "state" column.
`pd.get_dummies()`: This pandas function is used to encode one-hot encoding categorical variables. .
X_encoded is a variable that stores the encoding result.

The training data is as follows.

	R&D Spend	Administration	Marketing Spend	State_Florida	State_New York
0	165349.20	136897.80	471784.10	0	1
1	162597.70	151377.59	443898.53	0	0
2	153441.51	101145.55	407934.54	1	0
3	144372.41	118671.85	383199.62	0	1
4	142107.34	91391.77	366168.42	1	0

```
# Split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)
```

In the code above, the goal is to divide the data set into training and test sets.

```
# Train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

:`model = LinearRegression()'
This line creates an instance of the ``LinearRegression" class from the scikit-learn library, which represents a linear regression model, a type of supervised learning algorithm, for predicting a continuous target variable (in this case, "profit"). It is used based on one or more features.
During training, the model learns coefficients (weights) for each feature with the aim of minimizing the difference between the predicted values and the actual target values, and the coefficients are best adapted to the training data. This trained model can then be used to predict new or unseen data.

The accuracy of the model on the training data is equal to:

```
Train Set Accuracy Score: 0.9537019995248526
```

We also calculate the accuracy of the model on the test data.

```
# Predict on the test set
y_pred = model.predict(X_test)
```

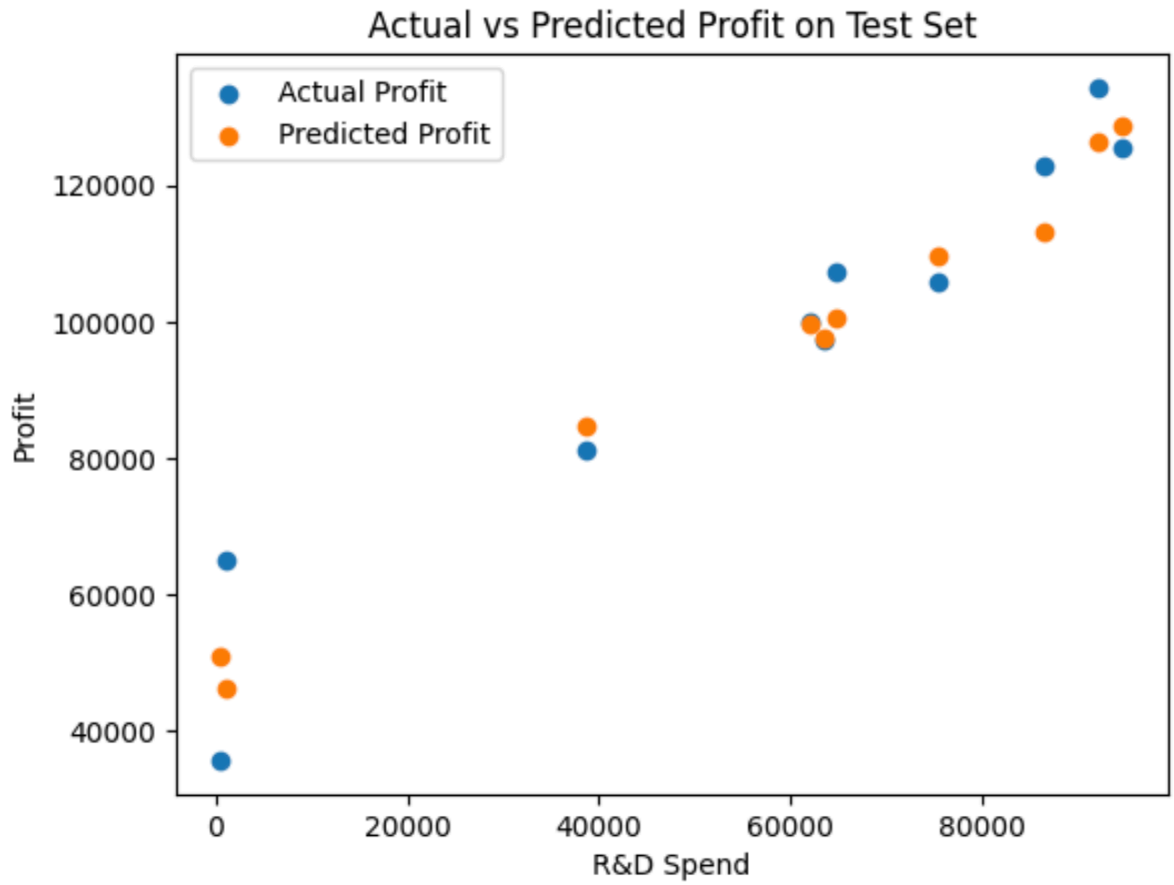
This line is used to predict the test set using a trained linear regression model.

```
Test Set Accuracy: 0.8987266414328637
```

The accuracy obtained on the test data is equal to 89%.
The MSE calculated for the test data is equal to:

```
Scaled Mean Squared Error: 0.008429774769805979
```

We plot the test data and predicted data and display them as below.
It should be noted that only one feature of the input data is selected for drawing.



The second question.
In this exercise, we are given a dataset containing information about users, namely user ID, gender, age, estimated salary, and a binary number for purchase (purchased or not).
The objective is to use logistic regression, a powerful algorithm in binary classification tasks, to detect patterns and relationships within the data. Logistic regression is particularly suitable for predicting the probability of an event occurring.
Libraries used in this exercise:

```
import pandas as pd
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
```

```
X = data[['User ID', 'Gender', 'Age', 'EstimatedSalary']]
y = data['Purchased']
```

	User ID	Gender	Age	EstimatedSalary
0	15624510	Male	19	19000
1	15810944	Male	35	20000
2	15668575	Female	26	43000
3	15603246	Female	27	57000
4	15804002	Male	19	76000

The above table shows part of a dataset with information about users. Each row corresponds to a different user and the columns show different characteristics related to these users.

```
X_encoded = pd.get_dummies(X, columns=['Gender'], drop_first=True)
```

The code above is used to one-hot encode the categorical variable "gender" into the dataset. X_encoded stores the result.
And then the desired output will be as below and gender will be binary.

	User ID	Age	EstimatedSalary	Gender_Male
0	15624510	19	19000	1
1	15810944	35	20000	1
2	15668575	26	43000	0
3	15603246	27	57000	0
4	15804002	19	76000	1

As in the previous exercise, we divide the data into two parts, training and testing.

```
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)
```

```
model = LogisticRegression()
model.fit(X_train, y_train)
```

Here, you create an example logistic regression model using the LogisticRegression class provided by scikit-learn. This class implements logistic regression for binary and multiclass classification. and adjusts the model parameters (weights and biases) based on the provided training data and tries to learn the relationship between the input features and the target variable.

The accuracy obtained for the training data is as follows.

```
Train Set Accuracy: 0.7875
```

Finally, with the help of the following code, we predict the output of the test data with the trained model.

```
y_pred = model.predict(X_test)
```

Finally, the accuracy obtained for the test data is equal to

```
Test Set Accuracy: 0.7375
```

Cross-entropy is commonly used as a loss function in logistic regression and other classification models. The main reason for using Cross-entropy as a loss function and next to it Maximum Likelihood (MLE) are useful for optimizing model parameters.

The calculated value of Cross-entropy for training and test data is equal to

```
Train Set Cross-Entropy Loss: 0.5786587838659834
Test Set Cross-Entropy Loss: 0.6089368260462589
```

The third question.

In this exercise, we examine a dataset that describes customer behavior and other information, including factors such as credit score, geography, gender, age, tenure, inventory, number of products, credit card usage, active membership status, and salary. estimated The main goal is to predict customer exit or retention based on these indicators using the perceptron algorithm.

As we know, the perceptron algorithm, the basic building block of neural networks, serves as a computational model for binary classification tasks and is suitable for predicting whether a customer is likely to leave or stay with a business. Using this algorithm, we aim to develop a predictive model that can effectively identify patterns within the dataset to identify factors contributing to customer churn. This analysis provides insights into the key drivers of customer churn and empowers businesses with the knowledge to implement targeted customer retention strategies.

In fact, by using the perceptron algorithm to predict customer churn, we seek to help understand customer behavior in common business environments.

The data of this exercise are as follows.

CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

```python
X = data[['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary']]
X = pd.concat([X, pd.get_dummies(data['Gender'], prefix='Gender')], axis=1)
X = pd.concat([X, pd.get_dummies(data['Geography'], prefix='Geography')], axis=1)

y = data['Exited'].values
```

In the initial line, the features used for prediction are selected from the dataset. Then, one-hot encoding is applied to the variables ('gender' and 'geography') to convert them into a suitable format for machine learning algorithms, creating one-hot encoding for each category.
The target variable 'y' is also set to the 'Exited' column of the dataset. This indicates whether the client has logged out or not.

```python
class Perceptron:
    def __init__(self, max_iters=100):
        self.max_iters = max_iters
        self.accuracies = []
        self.losses_train = []
        self.losses_val = []
        self.val_accuracies = []

    def fit(self, X, y, X_val=None, y_val=None):
        # Bookkeeping.
        X, y = np.asarray(X), np.asarray(y)
        iters = 0

        # Insert a bias column.
        X = np.concatenate((X, np.asarray([[1] * X.shape[0]]).T), axis=1)

        # Initialize random weights.
        self.ω = np.random.random(X.shape[1])

        # Train as many rounds as allotted, or until fully converged.
        for _ in range(self.max_iters):
            y_pred_all = []
            for idx in range(X.shape[0]):
                x_sample, y_sample = X[idx], y[idx]
                y_pred = int(np.sum(self.ω * x_sample) >= 0.5)
                if y_pred == y_sample:
                    pass
                elif y_pred == 0 and y_sample == 1:
                    self.ω = self.ω + x_sample
                elif y_pred == 1 and y_sample == 0:
                    self.ω = self.ω - x_sample

                y_pred_all.append(y_pred)

            iters += 1
            accuracy = np.mean(np.array(y_pred_all) == y)
            self.accuracies.append(accuracy)

            # Calculate MSE for training set
            mse_train = np.mean((np.array(y_pred_all) - y) ** 2)
            self.losses_train.append(mse_train)

            if X_val is not None and y_val is not None:
                val_accuracy = self.validation(X_val, y_val)
                self.val_accuracies.append(val_accuracy)
                # Calculate MSE for validation set
                y_pred_val = self.predict(X_val)
                mse_val = np.mean((y_pred_val - y_val) ** 2)
                self.losses_val.append(mse_val)

                print(f"Iteration {iters}/{self.max_iters} - Accuracy: {accuracy:.4f} - Validation Accuracy: {val_accuracy:.4f} - MSE
Train: {mse_train:.4f} - MSE Val: {mse_val:.4f}")
```

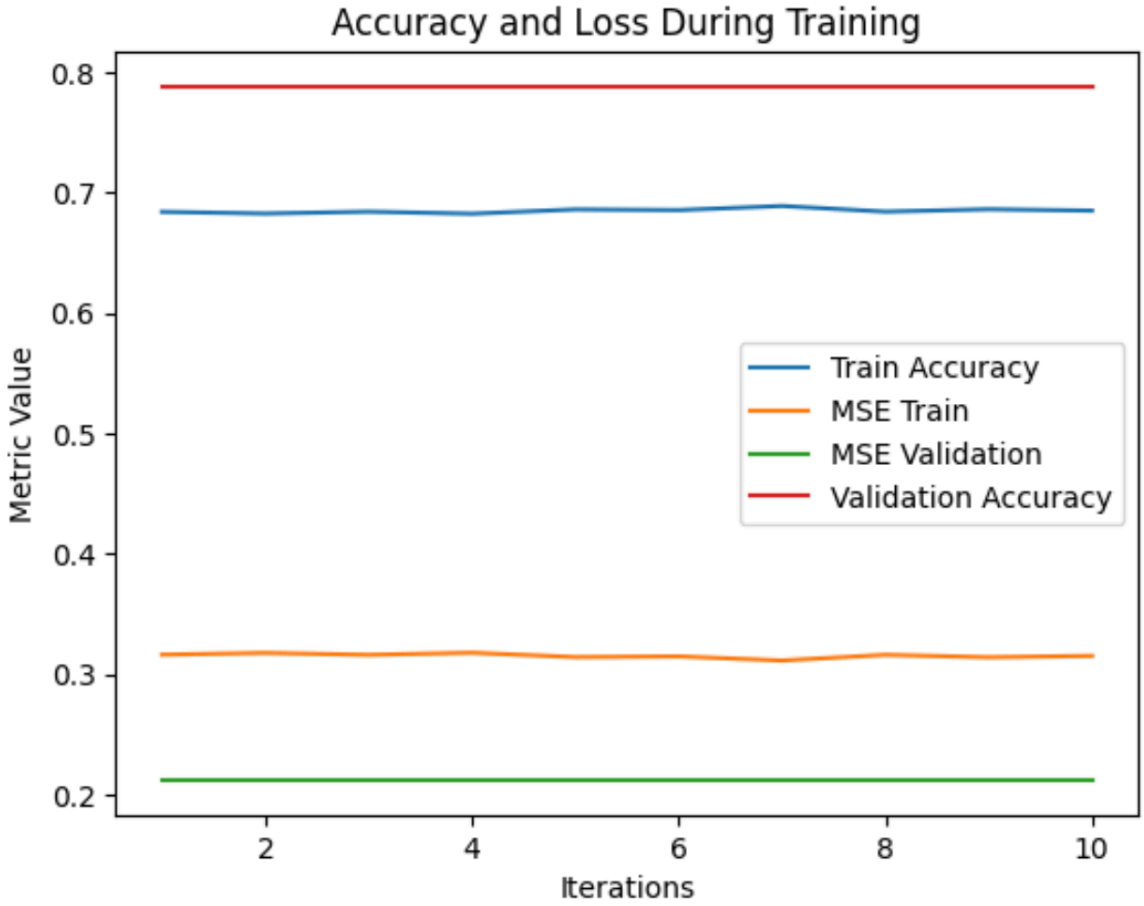
```
                print(f"Iteration {iters}/{self.max_iters} - Accuracy: {accuracy:.4f} - Validation Accuracy: {val_accuracy:.4f} - MSE
Train: {mse_train:.4f} - MSE Val: {mse_val:.4f}")
            else:
                print(f"Iteration {iters}/{self.max_iters} - Accuracy: {accuracy:.4f} - MSE Train: {mse_train:.4f}")

            if np.equal(np.array(y_pred_all), y).all():
                break

        self.iters = iters

        # Plot accuracy and loss during training
        self.plot_results()

    def validation(self, X_val, y_val):
        X_val = np.concatenate((X_val, np.asarray([[1] * X_val.shape[0]]).T), axis=1)
        y_pred_val = (X_val @ self.ω > 0.5).astype(int)
        val_accuracy = np.mean(y_pred_val == y_val)
        return val_accuracy

    def predict(self, X):
        # Inject the bias column.
        X = np.asarray(X)
        X = np.concatenate((X, np.asarray([[1] * X.shape[0]]).T), axis=1)

        return (X @ self.ω > 0.5).astype(int)

    def plot_results(self):
        # Plot accuracy and loss during training
        plt.plot(range(1, self.iters + 1), self.accuracies, label="Train Accuracy")
        if len(self.losses_train) > 0:
            plt.plot(range(1, self.iters + 1), self.losses_train, label="MSE Train")
        if len(self.losses_val) > 0:
            plt.plot(range(1, self.iters + 1), self.losses_val, label="MSE Validation")
        if len(self.val_accuracies) > 0:
            plt.plot(range(1, self.iters + 1), self.val_accuracies, label="Validation Accuracy")

        plt.xlabel("Iterations")
        plt.ylabel("Metric Value")
        plt.title("Accuracy and Loss During Training")
        plt.legend()
        plt.show()
```

A perceptron is one of the simplest forms of neural networks specifically designed for binary classification tasks. It takes a set of input features, applies weights to these features, and produces an output. If the output is above a certain threshold, it predicts a class. Otherwise, it predicts another class.
The "fit" method is responsible for training the perceptron. Takes the input features "x" and the label "y" for the training set. Optionally, it can have a validation set.
The input data are converted to NumPy arrays and a bias column is inserted into the feature matrix.
Random weights (`self.ω`) are initialized.
For each iteration, the perceptron is updated based on the training data. If the predicted output (`y_pred`) is false, the weights are adjusted.
The accuracy and mean squared error (MSE) for the training set are calculated and stored in the respective lists.

During training, the accuracy and MSE value for training and validation data is equal to

```
Iteration 1/10 - Accuracy: 0.6839 - Validation Accuracy: 0.7879 - MSE Train: 0.3161 - MSE Val: 0.2121
Iteration 2/10 - Accuracy: 0.6824 - Validation Accuracy: 0.7879 - MSE Train: 0.3176 - MSE Val: 0.2121
Iteration 3/10 - Accuracy: 0.6840 - Validation Accuracy: 0.7879 - MSE Train: 0.3160 - MSE Val: 0.2121
Iteration 4/10 - Accuracy: 0.6823 - Validation Accuracy: 0.7879 - MSE Train: 0.3177 - MSE Val: 0.2121
Iteration 5/10 - Accuracy: 0.6859 - Validation Accuracy: 0.7879 - MSE Train: 0.3141 - MSE Val: 0.2121
Iteration 6/10 - Accuracy: 0.6853 - Validation Accuracy: 0.7879 - MSE Train: 0.3147 - MSE Val: 0.2121
Iteration 7/10 - Accuracy: 0.6887 - Validation Accuracy: 0.7879 - MSE Train: 0.3113 - MSE Val: 0.2121
Iteration 8/10 - Accuracy: 0.6840 - Validation Accuracy: 0.7879 - MSE Train: 0.3160 - MSE Val: 0.2121
Iteration 9/10 - Accuracy: 0.6860 - Validation Accuracy: 0.7879 - MSE Train: 0.3140 - MSE Val: 0.2121
Iteration 10/10 - Accuracy: 0.6849 - Validation Accuracy: 0.7879 - MSE Train: 0.3151 - MSE Val: 0.2121
```

After completing the training, the value of accuracy and mse during the training process is as follows.



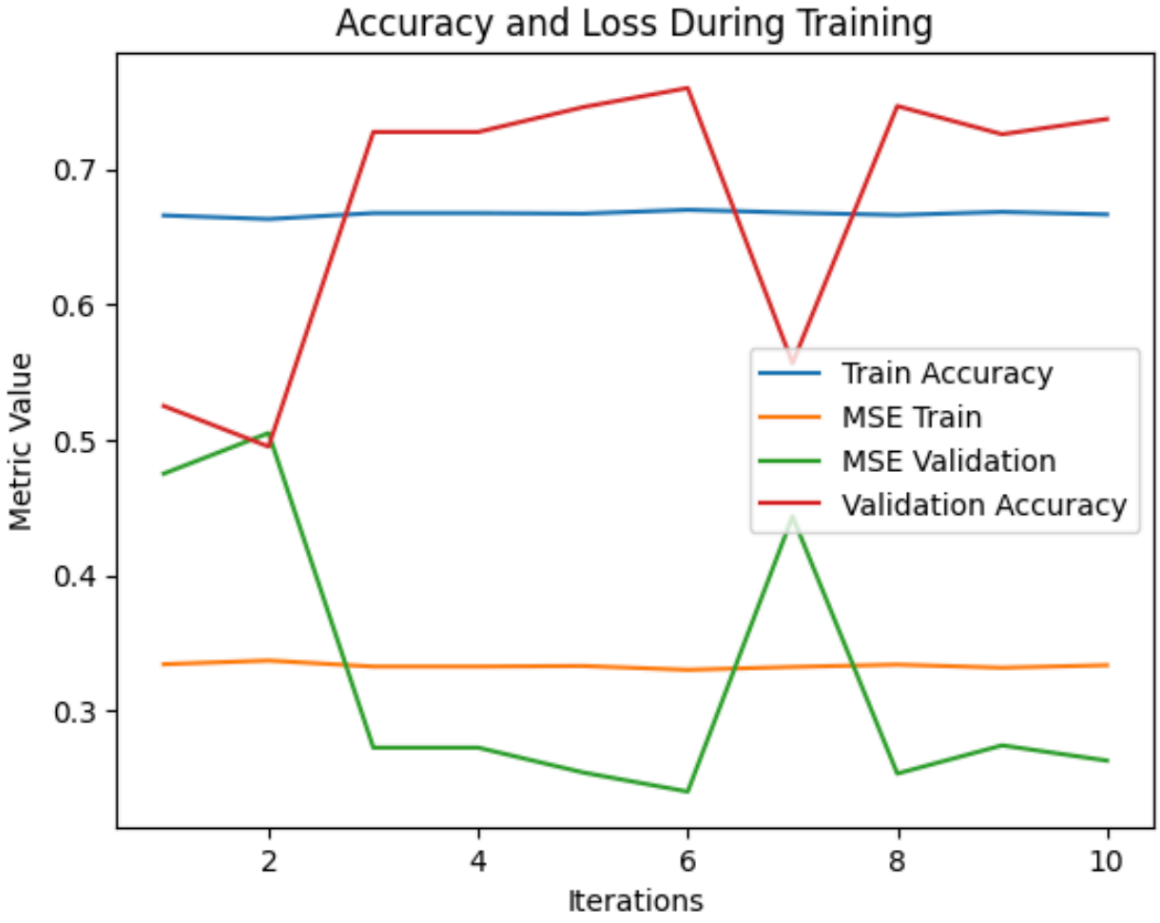
Finally, the accuracy obtained on the test data is equal to:

```
Test Set Accuracy: 0.7716666666666666
```

In this exercise, by performing multiple tests on the data, the accuracy of the test data is always higher than the accuracy of the training data. One of the reasons can be that the

number of test data is less than the training data and therefore its accuracy increases, or the test data is simpler data than the training data.
To try to improve this problem, we shuffle the data and run the algorithm again.
The results are changed as follows.

```
Iteration 1/10 - Accuracy: 0.6659 - Validation Accuracy: 0.5250 - MSE Train: 0.3341 - MSE Val: 0.4750
Iteration 2/10 - Accuracy: 0.6631 - Validation Accuracy: 0.4950 - MSE Train: 0.3369 - MSE Val: 0.5050
Iteration 3/10 - Accuracy: 0.6676 - Validation Accuracy: 0.7275 - MSE Train: 0.3324 - MSE Val: 0.2725
Iteration 4/10 - Accuracy: 0.6676 - Validation Accuracy: 0.7275 - MSE Train: 0.3324 - MSE Val: 0.2725
Iteration 5/10 - Accuracy: 0.6671 - Validation Accuracy: 0.7458 - MSE Train: 0.3329 - MSE Val: 0.2542
Iteration 6/10 - Accuracy: 0.6700 - Validation Accuracy: 0.7600 - MSE Train: 0.3300 - MSE Val: 0.2402
Iteration 7/10 - Accuracy: 0.6679 - Validation Accuracy: 0.5567 - MSE Train: 0.3321 - MSE Val: 0.4433
Iteration 8/10 - Accuracy: 0.6661 - Validation Accuracy: 0.7467 - MSE Train: 0.3339 - MSE Val: 0.2533
Iteration 9/10 - Accuracy: 0.6684 - Validation Accuracy: 0.7258 - MSE Train: 0.3316 - MSE Val: 0.2742
Iteration 10/10 - Accuracy: 0.6666 - Validation Accuracy: 0.7371 - MSE Train: 0.3334 - MSE Val: 0.2629
```




```
Test Set Accuracy: 0.7566666666666667
```