Latex Report for lab task 7

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.metrics import f1_score,accuracy_score,precision_score,
    ↪ recall_score
from sklearn.model_selection import train_test_split
# from sklearn.preprocessing import MinMaxScaler,Normalizer,
    ↪ MaxAbsScaler,RobustScaler
import numpy as np
from sklearn.linear_model import Perceptron
```

```
iris =load_iris()
print(iris)
```

'data': array([[5.1, 3.5, 1.4, 0.2], [4.9, 3. , 1.4, 0.2], [4.7, 3.2, 1.3, 0.2], [4.6, 3.1, 1.5, 0.2], [5. , 3.6, 1.4, 0.2], [5.4, 3.9, 1.7, 0.4], [4.6, 3.4, 1.4, 0.3], [5. , 3.4, 1.5, 0.2], [4.4, 2.9, 1.4, 0.2], [4.9, 3.1, 1.5, 0.1], [5.4, 3.7, 1.5, 0.2], [4.8, 3.4, 1.6, 0.2], [4.8, 3. , 1.4, 0.1], [4.3, 3. , 1.1, 0.1], [5.8, 4. , 1.2, 0.2], [5.7, 4.4, 1.5, 0.4], [5.4, 3.9, 1.3, 0.4], [5.1, 3.5, 1.4, 0.3], [5.7, 3.8, 1.7, 0.3], [5.1, 3.8, 1.5, 0.3], [5.4, 3.4, 1.7, 0.2], [5.1, 3.7, 1.5, 0.4], [4.6, 3.6, 1. , 0.2], [5.1, 3.3, 1.7, 0.5], [4.8, 3.4, 1.9, 0.2], [5. , 3. , 1.6, 0.2], [5. , 3.4, 1.6, 0.4], [5.2, 3.5, 1.5, 0.2], [5.2, 3.4, 1.4, 0.2], [4.7, 3.2, 1.6, 0.2], [4.8, 3.1, 1.6, 0.2], [5.4, 3.4, 1.5, 0.4], [5.2, 4.1, 1.5, 0.1], [5.5, 4.2, 1.4, 0.2], [4.9, 3.1, 1.5, 0.2], [5. , 3.2, 1.2, 0.2], [5.5, 3.5, 1.3, 0.2], [4.9, 3.6, 1.4, 0.1], [4.4, 3. , 1.3, 0.2], [5.1, 3.4, 1.5, 0.2], [5. , 3.5, 1.3, 0.3], [4.5, 2.3, 1.3, 0.3], [4.4, 3.2, 1.3, 0.2], [5. , 3.5, 1.6, 0.6], [5.1, 3.8, 1.9, 0.4], [4.8, 3. , 1.4, 0.3], [5.1, 3.8, 1.6, 0.2], [4.6, 3.2, 1.4, 0.2], [5.3, 3.7, 1.5, 0.2], [5. , 3.3, 1.4, 0.2], [7. , 3.2, 4.7, 1.4], [6.4, 3.2, 4.5, 1.5], [6.9, 3.1, 4.9, 1.5], [5.5, 2.3, 4. , 1.3], [6.5, 2.8, 4.6, 1.5], [5.7, 2.8, 4.5, 1.3], [6.3, 3.3, 4.7, 1.6], [4.9, 2.4, 3.3, 1.], [6.6, 2.9, 4.6, 1.3], [5.2, 2.7, 3.9, 1.4], [5. , 2. , 3.5, 1.], [5.9, 3. , 4.2, 1.5], [6. , 2.2, 4. , 1.], [6.1, 2.9, 4.7, 1.4], [5.6, 2.9, 3.6, 1.3], [6.7, 3.1, 4.4, 1.4], [5.6, 3. , 4.5, 1.5], [5.8, 2.7, 4.1, 1.], [6.2, 2.2, 4.5, 1.5], [5.6, 2.5, 3.9, 1.1], [5.9, 3.2, 4.8, 1.8], [6.1, 2.8, 4. , 1.3], [6.3, 2.5, 4.9, 1.5], [6.1, 2.8, 4.7, 1.2], [6.4, 2.9, 4.3, 1.3], [6.6, 3. , 4.4, 1.4], [6.8, 2.8, 4.8, 1.4], [6.7, 3. , 5. , 1.7], [6. , 2.9, 4.5, 1.5], [5.7, 2.6, 3.5, 1.], [5.5, 2.4, 3.8, 1.1], [5.5, 2.4, 3.7, 1.], [5.8, 2.7, 3.9, 1.2], [6. , 2.7, 5.1, 1.6], [5.4, 3. , 4.5, 1.5], [6. , 3.4, 4.5, 1.6], [6.7, 3.1, 4.7, 1.5], [6.3, 2.3, 4.4, 1.3], [5.6, 3. , 4.1, 1.3], [5.5, 2.5, 4. , 1.3], [5.5, 2.6, 4.4, 1.2], [6.1, 3. , 4.6, 1.4], [5.8, 2.6, 4. , 1.2], [5. , 2.3, 3.3, 1.], [5.6, 2.7, 4.2, 1.3], [5.7, 3. , 4.2, 1.2], [5.7, 2.9, 4.2, 1.3], [6.2, 2.9, 4.3, 1.3], [5.1, 2.5, 3. , 1.1], [5.7, 2.8, 4.1, 1.3], [6.3, 3.3, 6. , 2.5], [5.8, 2.7, 5.1, 1.9], [7.1, 3. , 5.9, 2.1], [6.3, 2.9, 5.6, 1.8], [6.5, 3. , 5.8, 2.2], [7.6, 3. , 6.6, 2.1], [4.9, 2.5, 4.5, 1.7], [7.3, 2.9, 6.3, 1.8], [6.7, 2.5, 5.8, 1.8], [7.2, 3.6, 6.1, 2.5], [6.5, 3.2, 5.1, 2.], [6.4, 2.7, 5.3, 1.9], [6.8, 3. , 5.5, 2.1], [5.7, 2.5, 5. , 2.], [5.8, 2.8, 5.1, 2.4], [6.4, 3.2, 5.3, 2.3], [6.5, 3. , 5.5, 1.8], [7.7, 3.8, 6.7, 2.2], [7.7, 2.6, 6.9, 2.3], [6. , 2.2, 5. , 1.5], [6.9, 3.2, 5.7, 2.3], [5.6, 2.8, 4.9, 2.], [7.7, 2.8, 6.7, 2.], [6.3, 2.7, 4.9, 1.8], [6.7, 3.3, 5.7, 2.1], [7.2, 3.2, 6. , 1.8], [6.2, 2.8, 4.8, 1.8], [6.1, 3. , 4.9, 1.8], [6.4, 2.8, 5.6, 2.1], [7.2, 3. , 5.8, 1.6], [7.4, 2.8, 6.1, 1.9], [7.9, 3.8, 6.4, 2.], [6.4, 2.8, 5.6, 2.2], [6.3, 2.8, 5.1, 1.5], [6.1, 2.6, 5.6, 1.4], [7.7, 3. , 6.1, 2.3], [6.3, 3.4, 5.6, 2.4], [6.4, 3.1, 5.5, 1.8], [6. , 3. , 4.8, 1.8], [6.9, 3.1, 5.4, 2.1], [6.7, 3.1, 5.6, 2.4], [6.9, 3.1, 5.1, 2.3], [5.8, 2.7, 5.1, 1.9], [6.8, 3.2, 5.9, 2.3], [6.7, 3.3, 5.7, 2.5], [6.7, 3. , 5.2, 2.3], [6.3, 2.5, 5. , 1.9], [6.5, 3. , 5.2, 2.], [6.2, 3.4, 5.4, 2.3], [5.9, 3. , 5.1, 1.8]]), 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]), 'frame': None, 'target$_n$ames′ : array(['setosa′,′ versicolor′,′ virginica′], dtype =′< U10′),′ DESCR′ :′ ..$_i$ris$_d$ataset : plantsdataset − ∗∗DataSetCharacteristics : ∗∗ : NumberofInstances : 150(50ineachofthreeclasses) : NumberofAttributes : 4numeric, predictiveattributesandtheclass : AttributeInformation : −sepallengthincm − sepalwidthincm − petallengthincm − petalwidthincm − class : −Iris − Setosa − Iris −

Versicolour−Iris−Virginica : *SummaryStatistics* :===
MinMaxMeanSDClassCorrelation ===
length : 4.37.95.840.830.7826*width* : 2.04.43.050.43 − 0.4194*length* : 1.06.93.761.760.9490(*high*!)*width* :
0.12.51.200.760.9565(*high*!) ==
MissingAttributeValues : *None* : *ClassDistribution* : 33.3

```
df = pd.DataFrame(data=iris.data,columns=iris.feature_names)
display(df)
```

sepal length (cm) sepal width (cm) petal length (cm) petal width (cm) 0 5.1 3.5 1.4 0.2 1 4.9 3.0 1.4 0.2 2 4.7 3.2 1.3 0.2 3 4.6 3.1 1.5 0.2 4 5.0 3.6 1.4 0.2 .. .. 145 6.7 3.0 5.2 2.3 146 6.3 2.5 5.0 1.9 147 6.5 3.0 5.2 2.0 148 6.2 3.4 5.4 2.3 149 5.9 3.0 5.1 1.8

[150 rows x 4 columns]

```
df['target'] = iris.target #making an extra column for class with
    ↪ numerics
df['target_names'] = df['target'].apply(lambda x: iris.target_names[x
    ↪ ]) #making another with class names

#now we will exclude virginica
df_binary = df[df['target_names'].isin(['setosa','versicolor'])] #
    ↪ keeping only the first two classes
df_binary['binary_target']= df_binary['target_names'].map({'setosa
    ↪ ':0,'versicolor':1})
display(df_binary.head(10)) #checking

#scatter plot for sepal width and length
plt.scatter(df_binary.iloc[:,0], df_binary.iloc[:,1], c=df_binary['
    ↪ target'], cmap='bwr', edgecolors='k')
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.title('Binary Classification: Setosa vs Versicolor')
plt.grid(True)
plt.show()

#scatter plot for petal length and width
plt.scatter(df_binary.iloc[:,2], df_binary.iloc[:,3], c=df_binary['
    ↪ target'], cmap='bwr', edgecolors='k')
plt.xlabel(iris.feature_names[2])
plt.ylabel(iris.feature_names[3])
plt.title('Binary Classification: Setosa vs Versicolor')
plt.grid(True)
plt.show()
```

C:PC$_1$614867668840*.py* : 6 : *SettingWithCopyWarning* : *AvalueistryingtobesetonacopyofaslicefromaDataFrame.Try*
valueinstead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user$_g$*uide/indexing.htmlreturning*
a−view−versus−a−copydf$_b$*inary*['*binary*$_t$*arget*'] = *df*$_b$*inary*['*target*$_n$*ames*'].*map*('*setosa*' : 0,' *versicolor*' : 1)*sepallength*(*cm*

target target$_n$*amesbinary*$_t$*arget*00*setosa*010*setosa*020*setosa*030*setosa*040*setosa*050*setosa*060*setosa*070*setosa*080*setosa*09
*Figuresize*640*x*480*with*1*Axes* >< *Figuresize*640*x*480*with*1*Axes* >

```
x = df_binary.drop([’target’,’target_names’],axis=1) # extracting
    ↪ features
y = df_binary[’binary_target’] # target value
```

```
#spliting the data into train and test sets
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2,
    ↪ random_state=3)
```

```
percept = Perceptron(max_iter= 1000,tol=0.001,random_state=3) #the
    ↪ model will stop training after the fault tolerance exceeds
    ↪ 0.001
percept.fit(x_train,y_train) #since its supervised
y_predict=percept.predict(x_test)
```

```
print(f"Accuracy after using perceptron : {accuracy_score(y_test,
    ↪ y_predict)}")
print(f"Precision after using perceptron : {precision_score(y_test,
    ↪ y_predict)}")
print(f"Recall score after using perceptron : {recall_score(y_test,
    ↪ y_predict)}")
print(f"F1 score after using perceptron : {f1_score(y_test,y_predict)
    ↪ }")
```

Accuracy after using perceptron : 1.0 Precision after using perceptron : 1.0 Recall score after using perceptron : 1.0 F1 score after using perceptron : 1.0

```
#perceptron is an supervised learning algorithm for binary
   ↪ classifiers
#now we will build a perceptron from scratch

def unit_step_func(x):
    return np.where(x>0,1,0)

class perceptron:
    def __init__(self,learning_rate=0.00001,iterations=1000):
        self.lr = learning_rate
        self.iters = iterations
        self.activation_function = unit_step_func
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        # Ensure X and y are numpy arrays of type float
        X = np.asarray(X).astype(float)
        y = np.asarray(y)
        n_samples, n_features = X.shape

        self.weights = np.zeros(n_features)
        self.bias = 0

        y_ = np.where(y > 0, 1, 0)

        for _ in range(self.iters):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                y_predicted = self.activation_function(linear_output)

                update = self.lr * (y_[idx] - y_predicted)
                self.weights += update * x_i
                self.bias += update

    def predict(self, X):
        X = np.asarray(X).astype(float)
        linear_output = np.dot(X, self.weights) + self.bias
        y_predicted = self.activation_function(linear_output)
        return y_predicted

#now we will train the data on this perceptron
per = perceptron(learning_rate=0.01,iterations = 1500)
per.fit(x_train,y_train)
y_prediction = per.predict(x_test)
```

```
print(f"Accuracy after using new perceptron : {accuracy_score(y_test,
   ↪ y_prediction)}")
print(f"Precision after using new perceptron : {precision_score(
   ↪ y_test,y_prediction)}")
print(f"Recall score after using new perceptron : {recall_score(
   ↪ y_test,y_prediction)}")
print(f"F1 score after using new perceptron : {f1_score(y_test,
   ↪ y_prediction)}")
```

Accuracy after using perceptron : 1.0 Precision after using perceptron : 1.0 Recall score after using perceptron : 1.0 F1 score after using perceptron : 1.0