Experiment 1: Data Preprocessing in machine learning

Aim: To prepare data set for processing.

Theory:

Data preprocessing prepares raw data for modeling by improving its quality and ensuring compatibility with algorithms. This process begins with data cleaning, where missing values are handled, outliers are removed, and inconsistencies are corrected. Next, data transformation standardizes scales and encodes categorical features, while feature engineering enhances the dataset by creating and selecting the most relevant features. Afterward, data is split into training and test sets, often with cross-validation for robust evaluation. If there's class imbalance, techniques like oversampling, undersampling, or synthetic data generation address it. Together, these steps enable accurate, efficient model training.

Procedure:

1.Load and visualize data

2.Handling missing data

3.Feature selection

4.Encode the categorical data label encodeing

5.Tarin test split

6.split data set into k-folds

7.Feature scaling

Conclusion:

With this steps we can enable to prepare the data for processing.

Experiment 2: Linear Regression in Machine Learning

Aim: Linear regression is an approach for predicting a response using a single feature.

x	0	1	2	3	4	5	6	7	8	9
y	1	3	2	5	7	8	8	9	10	12

Theory:

Linear regression is a statistical method that is used to predict a continuous dependent variable(target variable) based on one or more independent variables(predictor variables). This technique assumes a linear relationship between the dependent and independent variables, which implies that the dependent variable changes proportionally with changes in the independent variables. In other words, linear regression is used to determine the extent to which one or more variables can predict the value of the dependent variable.

Algorithm:

1. **Define the Model**

   Assume a linear relationship between the independent variable X and the dependent variable Y given by:

   $$Y = mX + b$$

   where m is the slope and b is the y-intercept.

2. **Calculate the Slope (m) and Intercept (b)**

   Use the Least Squares formulas to minimize the error:

   $$m = \frac{N\sum(XY) - \sum(X)\sum(Y)}{N\sum(X^2) - (\sum(X))^2}$$

   $$b = \frac{\sum(Y) - m\sum(X)}{N}$$

   where:

   - N is the number of data points.
   - $\sum(X)$ is the sum of all X values.
   - $\sum(Y)$ is the sum of all Y values.
   - $\sum(XY)$ is the sum of the product of X and Y values.
   - $\sum(X^2)$ is the sum of the square of X values.

3. **Make Predictions**

   Use the calculated m and b to predict Y values:

   $$Y_{\text{pred}} = mX + b$$

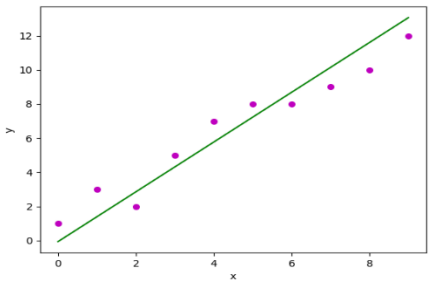
4. **Evaluate the Model (Optional)**

   Calculate metrics like Mean Squared Error (MSE) or R-squared to evaluate the model's performance.

Source Code:

```
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef(x, y):
  n = np.size(x)
  m_x = np.mean(x)
```

```
    m_y = np.mean(y)
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x
    return (b_0, b_1)
def plot_regression_line(x, y, b):
    plt.scatter(x, y, color = "m",
           marker = "o", s = 30)
    y_pred = b[0] + b[1]*x
    plt.plot(x, y_pred, color = "g")
    plt.xlabel('x')
    plt.ylabel('y')
```



Experiment 3: Polynomial Regression in Machine Learning

Aim: To implementation of the Polynomial Regression model from scratch and validation of the model on a dummy dataset.

Theory:

Polynomial Regression is an extension of linear regression used when the relationship between the independent variable X and the dependent variable Y is non-linear. Instead of fitting a straight line, polynomial regression fits a curve to capture the complexity in the data. It achieves this by introducing polynomial terms (e.g., X^2, X^3, etc.) into the model, making it possible to fit curves of different shapes.

Key Points:

- **Flexibility**: Polynomial regression allows for modeling non-linear relationships by adjusting the degree of the polynomial.
- **Careful Tuning**: Selecting the right degree is essential to balance the model's ability to capture patterns without overfitting.
- **Applications**: Polynomial regression is useful for trends that show curvature, such as growth curves, seasonal trends, and other naturally curving data patterns.

Algorithm:

1. Import necessary libraries like numpy for data manipulation, PolynomialFeatures from sklearn.preprocessing to generate polynomial terms, and LinearRegression from sklearn.linear_model for model fitting.
2. Prepare the dataset by creating or loading data for the independent variable X and the dependent variable Y. Reshape X to be a 2D array if it's a single feature for compatibility with sklearn.
3. Transform the original features to polynomial features. Use PolynomialFeatures to generate terms up to the specified degree (for example, quadratic or cubic). Fit and transform X to get these polynomial features.
4. Fit the model using LinearRegression. Fit the transformed polynomial features and the target variable Y to the linear regression model, which will learn the coefficients for each polynomial term.
5. Make predictions for Y values using the fitted model.

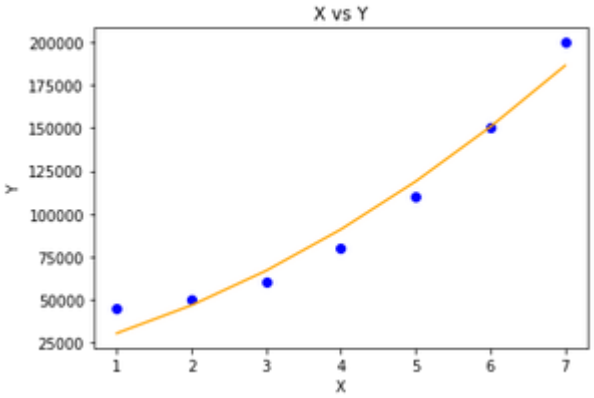
Source Code:

```
import numpy as np
import math
import matplotlib.pyplot as plt
class PolynomailRegression() :
        def __init__( self, degree, learning_rate, iterations ) :
                self.degree = degree
                self.learning_rate = learning_rate
                self.iterations = iterations
        def transform( self, X ) :
                X_transform = np.ones( ( self.m, 1 ) )
                j = 0
                for j in range( self.degree + 1 ) :
```

```
                if j != 0 :
                    x_pow = np.power( X, j )
                    X_transform = np.append( X_transform, x_pow.reshape( -1, 1 ), axis
= 1 )
        return X_transform
    def normalize( self, X ) :
        X[:, 1:] = ( X[:, 1:] - np.mean( X[:, 1:], axis = 0 ) ) / np.std( X[:, 1:], axis = 0 )
        return X
    def fit( self, X, Y ) :
        self.X = X
        self.Y = Y
        self.m, self.n = self.X.shape
        self.W = np.zeros( self.degree + 1 )
        X_transform = self.transform( self.X )
        X_normalize = self.normalize( X_transform )
        for i in range( self.iterations ) :
            h = self.predict( self.X )
            error = h - self.Y
            self.W = self.W - self.learning_rate * ( 1 / self.m ) * np.dot( X_normalize.T,
error )
        return self
    def predict( self, X ) :
        X_transform = self.transform( X )
        X_normalize = self.normalize( X_transform )
        return np.dot( X_transform, self.W )
def main() :
    X = np.array( [ [1], [2], [3], [4], [5], [6], [7] ] )
    Y = np.array( [ 45000, 50000, 60000, 80000, 110000, 150000, 200000 ] )
    model = PolynomailRegression( degree = 2, learning_rate = 0.01, iterations = 500 )
    model.fit( X, Y )
    Y_pred = model.predict( X )
    plt.scatter( X, Y, color = 'blue' )
    plt.plot( X, Y_pred, color = 'orange' )
```

```
        plt.title( 'X vs Y' )
        plt.xlabel( 'X' )
        plt.ylabel( 'Y' )
        plt.show()
if __name__ == "__main__" :
        main()
```

Output:



Experiment 4: Logistic Regression in Machine Learning

Aim: Target variable can have only 2 possible types: “0” or “1” which may represent “win” vs “loss”, “pass” vs “fail”, “dead” vs “alive”, etc., in this case, sigmoid functions are used, which is already discussed above.

Importing necessary libraries based on the requirement of model. This Python code shows how to use the breast cancer dataset to implement a Logistic Regression model for classification.

Theory:

Logistic regression is a **supervised machine learning algorithm** used for **classification tasks** where the goal is to predict the probability that an instance belongs to a given class or not. Logistic regression is a statistical algorithm which analyze the relationship between two data factors.

Key Points:

- Logistic regression predicts the output of a categorical dependent variable. Therefore, the outcome must be a categorical or discrete value.
- It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
- In Logistic regression, instead of fitting a regression line, we fit an “S” shaped logistic function, which predicts two maximum values (0 or 1).

Algorithm:

1. Import Libraries

Import necessary libraries such as sklearn for data handling, logistic regression model, and metrics, as well as numpy and matplotlib for array manipulation and visualization.

2. Load Dataset

Load a binomial dataset (e.g., make_classification from sklearn.datasets or the breast_cancer dataset from sklearn.datasets).

3. Data Preprocessing

- Split the dataset into features (X) and target variable (y).

- Normalize or standardize the features if necessary (optional for logistic regression but may

improve performance).

4. Train-Test Split

Use train_test_split to divide the dataset into training and testing sets (e.g., 80% training, 20% testing).

5. Initialize Logistic Regression Model

Initialize the LogisticRegression model from sklearn.linear_model with default or specified parameters.

6. Train the Model

Fit the model on the training data using the .fit() method.

7. Make Predictions

Use the .predict() method on the test set to get the predicted labels.

8. Evaluate the Model

- Calculate accuracy using accuracy_score.

- Optionally, calculate other metrics (e.g., precision, recall, F1-score, confusion matrix) for better evaluation.

9. Visualize Results

(Optional) Visualize the decision boundary if the data is 2D or view a confusion matrix for classification performance.

Source Code:

```
from sklearn.datasets import load_breast_cancer

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score
```

```
X, y = load_breast_cancer(return_X_y=True)
X_train, X_test,\
    y_train, y_test = train_test_split(X, y,
                                       test_size=0.20,
                                       random_state=23)
clf = Logistic Regression(random state=0)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Logistic Regression model accuracy (in %):", acc*100)
```

Output:

```
Logistic Regression model accuracy (in %): 95.6140350877193
```