PATTERN RECOGNITION AND ANOMALY DETECTION

LAB FILE

Name: Aviral Khanna
Sap: 500108516
Roll No.: R2142221156
Batch: B-3 AI/ML(Hons.) Submitted

To: Ms. Pooja Sarin

Experiment - 1

What is Anaconda :

Anaconda is a popular open-source distribution for Python and R, widely used in data science, machine learning, and scientific computing. It simplifies package management and deployment, making it easier to work with large-scale data analysis and AI/ML projects.

Key Features of Anaconda

- **Package & Environment Management:** Uses condato manage dependencies and virtual environments.

- **Pre-installed Libraries:** Comes with over 1,500+ scientific packages like NumPy, Pandas, SciPy, and Matplotlib.

- **Jupyter Notebook & Spyder:** Includes tools for interactive coding and visualization.

- **Cross-Platform:** Available for Windows, macOS, and Linux.

- **Optimized for Machine Learning:** Supports deep learning frameworks like TensorFlow, PyTorch, and Scikit-learn.

Installing Anaconda

```
wget https://repo.anaconda.com/archive/Anaconda3-2024.02-0-Linux-x86_64.sh
bash Anaconda3-2024.02-0-Linux-x86_64.sh
```

Set up environment :

```
conda create --name myenv python=3.9
```

```
conda activate myenv
```

Install Packages :

```
conda install numpy pandas matplotlib
```

Experiment - 2

NumPy:

NumPy (Numerical Python) is a powerful library for numerical computations. It provides support for multi-dimensional arrays and matrices, along with mathematical functions to operate on these data structures efficiently. NumPy arrays are more efficient and faster than Python lists due to their fixed type, memory optimization, and vectorized operations.

Key Features of NumPy:

- Support for N-dimensional arrays (ndarray).
- Mathematical and statistical functions.
- Linear algebra operations.
- Random number generation.

Pandas:

Pandas is a widely used data manipulation and analysis library built on top of NumPy. It provides two main data structures:

1. Series – A one-dimensional labeled array capable of holding any data type.
2. DataFrame – A two-dimensional, tabular data structure with labeled axes (rows and columns), similar to a spreadsheet or SQL table.

Key Features of Pandas:

- DataFrame and Series for structured data handling.
- Efficient data selection, filtering, and transformation.

```python
# Importing Required Libraries
import numpy as np
import pandas as pd
```

```python
## Part 1: NumPy Operations
### Creating and Manipulating NumPy Arrays
"""

# Creating a NumPy array
array = np.array([[1, 2, 3], [4, 5, 6]])
print("NumPy Array:")
print(array)

# Performing mathematical operations
array_squared = np.square(array)
print("\nSquared Array:")
print(array_squared)

# Generating random numbers
random_array = np.random.rand(3, 3)
print("\nRandom Array:")
print(random_array)
```

```python
"""
## Part 2: Pandas Operations
### Creating and Manipulating DataFrames
"""

# Creating a DataFrame
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'Score': [85, 90, 95]
}
df = pd.DataFrame(data)
print("\nPandas DataFrame:")
print(df)

# Descriptive statistics
print("\nDataFrame Description:")
print(df.describe())

# Adding a new column
df['Passed'] = df['Score'] > 80
print("\nUpdated DataFrame:")
print(df)

# Filtering data
filtered_df = df[df['Age'] > 28]
print("\nFiltered DataFrame (Age > 28):")
print(filtered_df)
```

Experiment - 3

Linear Regression

Linear Regression is a fundamental statistical and machine learning technique used for modeling the relationship between a dependent variable (target) and one or more independent variables (features). It assumes a linear relationship between the variables and is widely used for prediction and analysis.

---

1. Types of Linear Regression

1. **Simple Linear Regression**:
   - Involves one independent variable (X) and one dependent variable (Y).
   - The model equation is:

$$Y = \beta_0 + \beta_1 X + \epsilon$$

2. **Multiple Linear Regression**:
   - Extends Simple Linear Regression to multiple independent variables.
   - The model equation is:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n + \epsilon$$

In [1]:
```
#linear regression in making
```

In [4]:
```
# things we can learn in making this
# how would i know it right now i just started it
```

In [ ]:

In [ ]:

In [49]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegre
from sklearn.metrics import mean_squared_error, r2_score
```

In [2]:
```
import numpy as np
import pandas as pd
```

EDA

In [30]:
```
df = pd.read_csv("possum.csv")
df1 = pd.read_csv("possum.csv")
```

In [31]:
```
df.head()
```

Out[31]:

	ase	site	Pop	sex	age	hdlngth	skullw	totlngth	taill	footlgth	earconch	eye
0	1	1	Vic	m	8.0	94.1	60.4	89.0	36.0	74.5	54.5	15.2
1	2	1	Vic	f	6.0	92.5	57.6	91.5	36.5	72.5	51.2	16.0
2	3	1	Vic	f	6.0	94.0	60.0	95.5	39.0	75.4	51.9	15.5
3	4	1	Vic	f	6.0	93.2	57.1	92.0	38.0	76.1	52.2	15.2
4	5	1	Vic	f	2.0	91.5	56.3	85.5	36.0	71.0	53.2	15.1

In [32]:
```
df.describe()
```

Out[32]:

	case	site	age	hdlngth	skullw	totlngth	
count	104.000000	104.000000	102.000000	104.000000	104.000000	104.000000	104.00
mean	52.500000	3.625000	3.833333	92.602885	56.883654	87.088462	37.0(
std	30.166206	2.349086	1.909244	3.573349	3.113426	4.310549	1.9!
min	1.000000	1.000000	1.000000	82.500000	50.000000	75.000000	32.00
25%	26.750000	1.000000	2.250000	90.675000	54.975000	84.000000	35.87
50%	52.500000	3.000000	3.000000	92.800000	56.350000	88.000000	37.00
75%	78.250000	6.000000	5.000000	94.725000	58.100000	90.000000	38.00
max	104.000000	7.000000	9.000000	103.100000	68.600000	96.500000	43.00

In [33]:
```
df.isna().sum()
```

Out[33]:
```
case        0
site        0
Pop         0
sex         0
age         2
hdlngth     0
skullw      0
totlngth    0
taill       0
footlgth    1
earconch    0
eye         0
chest       0
belly       0
dtype: int64
```

In [7]:
```
df.drop(["case"], inplace=True, axis=1) #only a index no need in the data
```

In [8]:
```
categorical_column  =  df.select_dtypes(include="object").columns
```

In [9]:
```
df = df.drop(["Pop","sex"], axis=1) # popping out categorical columns and
```

In [16]:
```
numerical_column = df.select_dtypes(exclude="object").columns
print(categorical_column, numerical_column)
```

```
Index(['Pop', 'sex'], dtype='object') Index(['site', 'age', 'hdlngth', 'skullw',
'totlngth', 'taill', 'footlgth',
       'earconch', 'eye', 'chest', 'belly'],
      dtype='object')
```

In [17]:
```
df.describe()
```

Out[17]:

	site	age	hdlngth	skullw	totlngth	taill	foc
count	104.000000	102.000000	104.000000	104.000000	104.000000	104.000000	103.00
mean	3.625000	3.833333	92.602885	56.883654	87.088462	37.009615	68.45
std	2.349086	1.909244	3.573349	3.113426	4.310549	1.959518	4.39
min	1.000000	1.000000	82.500000	50.000000	75.000000	32.000000	60.30
25%	1.000000	2.250000	90.675000	54.975000	84.000000	35.875000	64.60
50%	3.000000	3.000000	92.800000	56.350000	88.000000	37.000000	68.00
75%	6.000000	5.000000	94.725000	58.100000	90.000000	38.000000	72.50
max	7.000000	9.000000	103.100000	68.600000	96.500000	43.000000	77.90

In [18]:

```
df.dtypes
```

Out[18]:

```
site          int64
age         float64
hdlngth     float64
skullw      float64
totlngth    float64
taill       float64
footlgth    float64
earconch    float64
eye         float64
chest       float64
belly       float64
dtype: object
```

In [19]:

```
df.corr().style.background_gradient(cmap="coolwarm")
```

Out[19]:

age	-0.131423	1.000000	0.319022	0.285107	0.260280	0.118241	0.126190
hdlngth	-0.163646	0.319022	1.000000	0.710827	0.691094	0.287429	0.391605
skullw	-0.083548	0.285107	0.710827	1.000000	0.526413	0.255921	0.275059
totlngth	-0.260843	0.260280	0.691094	0.526413	1.000000	0.565646	0.444832
taill	0.380444	0.118241	0.287429	0.255921	0.565646	1.000000	-0.126277
footlgth	-0.783009	0.126190	0.391605	0.275059	0.444832	-0.126277	1.000000
earconch	-0.790716	0.053405	0.121463	-0.000537	0.154484	-0.385136	0.783050
eye	-0.036987	0.235553	0.347175	0.321991	0.247786	0.198134	0.005213
chest	-0.345494	0.334209	0.631498	0.629737	0.577890	0.174997	0.450590
belly	-0.175266	0.354298	0.562663	0.451838	0.519465	0.294493	0.302584

from the above correlation matrix findings :

1.earconch and footlength highest relation

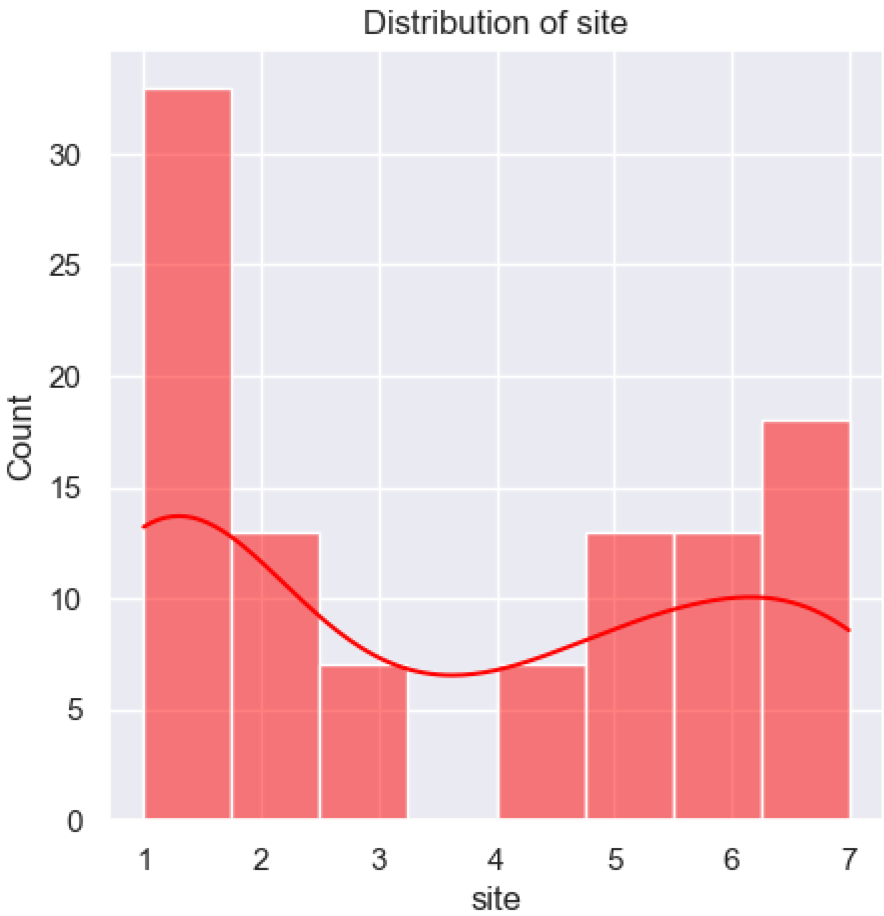
2. headlength and skullwidth

3. headlength and totlength

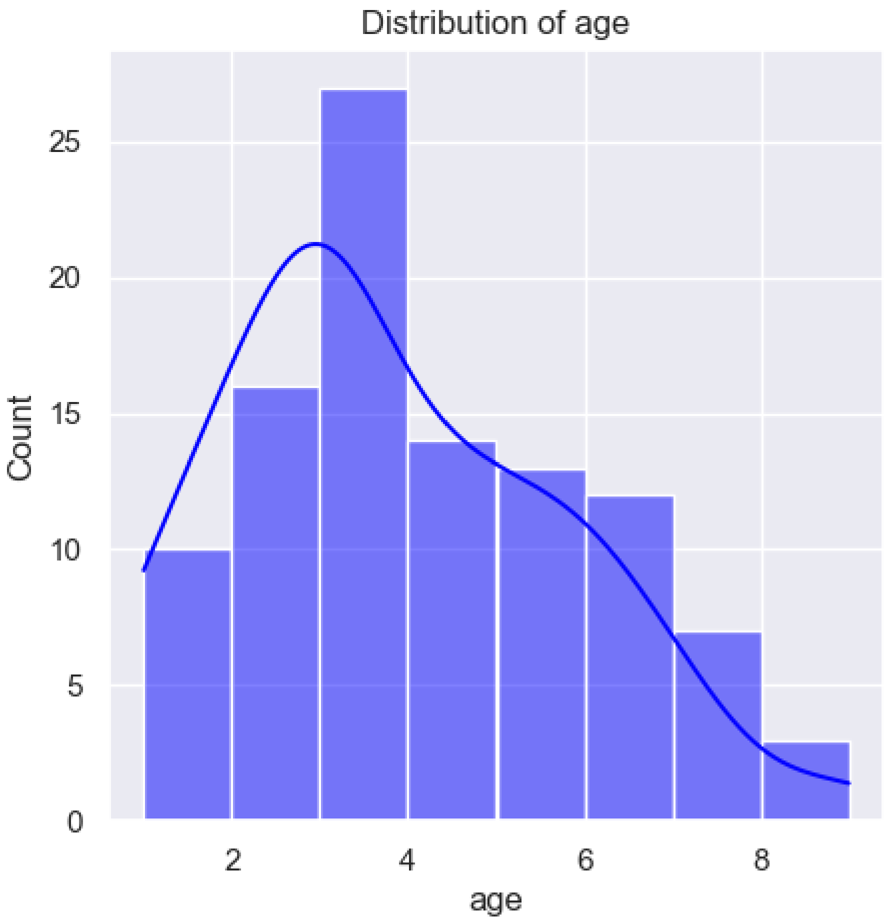
In [20]:
```
colors=["red","blue", "green","orange","black","purple", "brown","pink","
```

In [21]:
```
for i in range(11):
    plt.figure(figsize=(5,5)) sns.set(style="darkgrid")
    sns.histplot(df, x=df[numerical_column[i]], kde=True, color=colors[i]
    plt.title(f"Distribution of {numerical_column[i]}")
    plt.show()
```

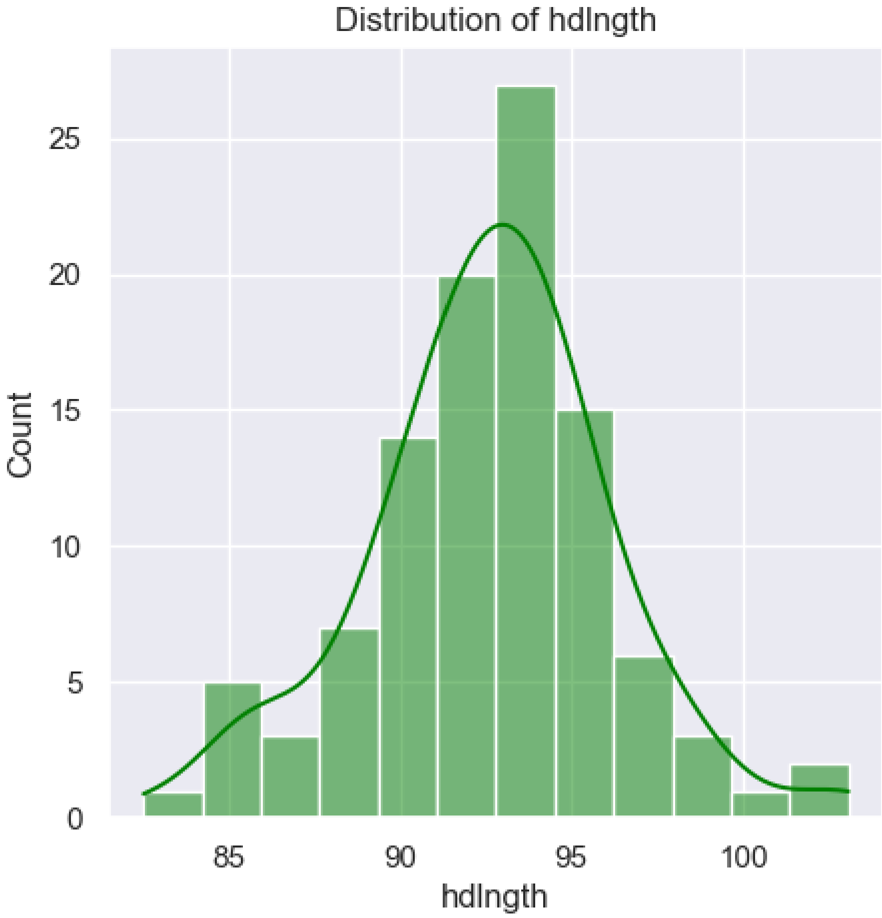
```
/Users/parz/miniforge3/envs/data-science/lib/python3.9/site-packages/sea
born/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated
and will be removed in a future version. Convert inf values to NaN befor
e operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



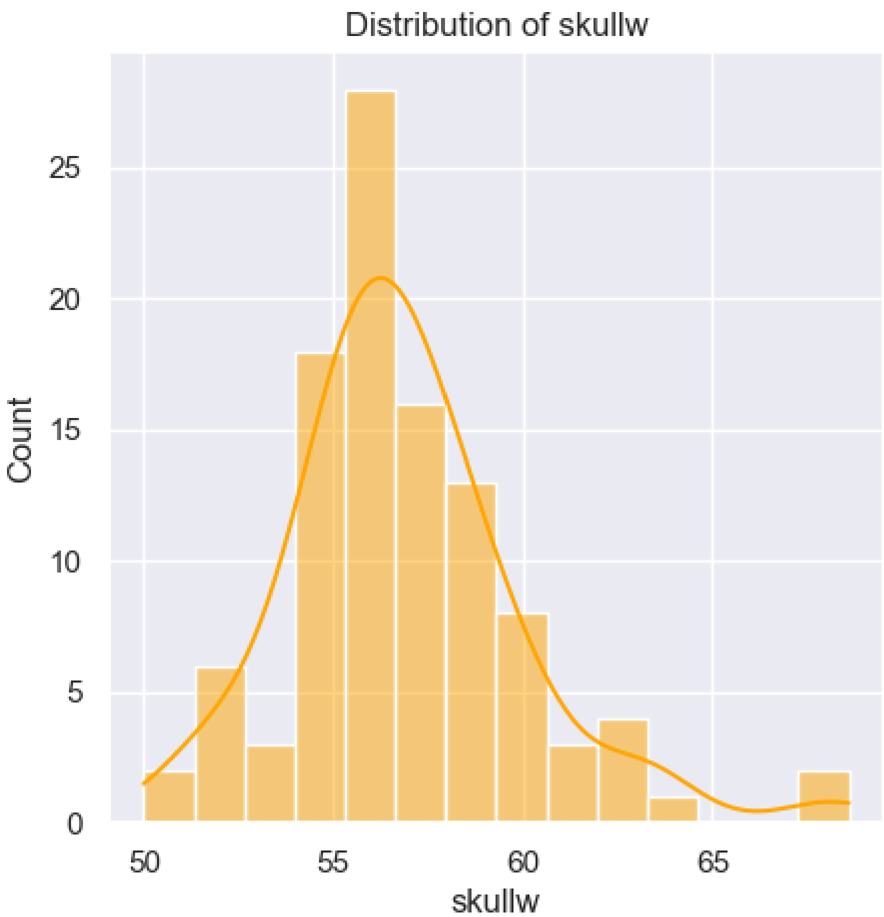

```
/Users/parz/miniforge3/envs/data-science/lib/python3.9/site-packages/sea
born/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated
and will be removed in a future version. Convert inf values to NaN befor
e operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



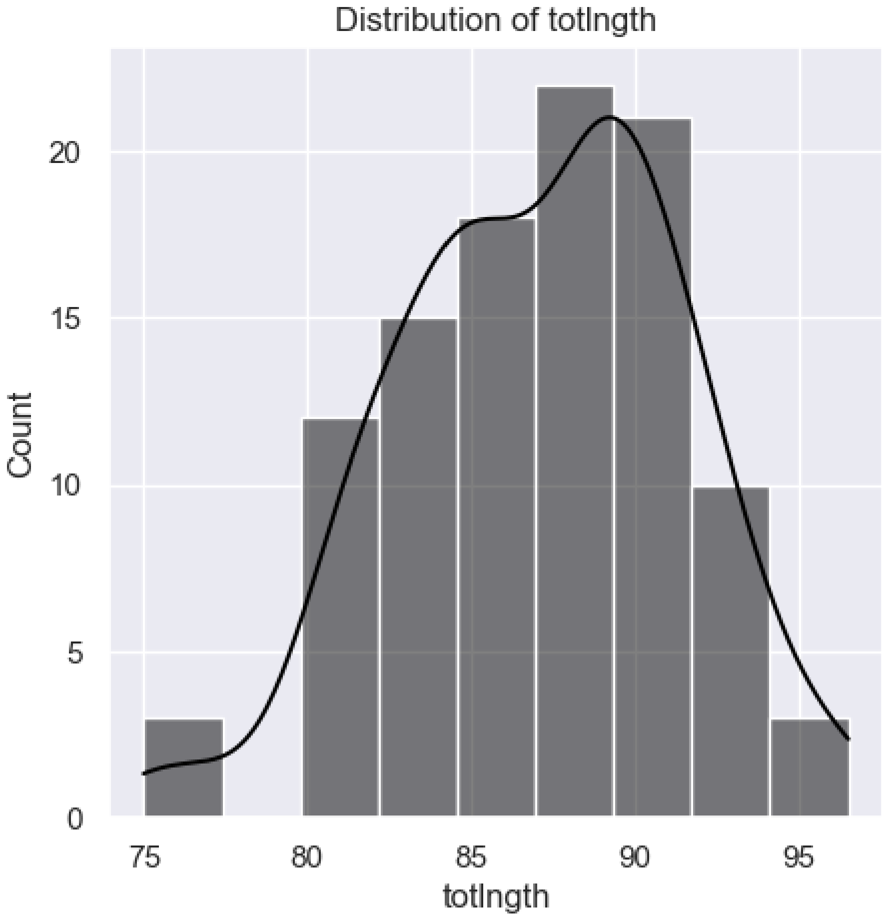

```
/Users/parz/miniforge3/envs/data-science/lib/python3.9/site-packages/sea
born/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated
and will be removed in a future version. Convert inf values to NaN befor
e operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



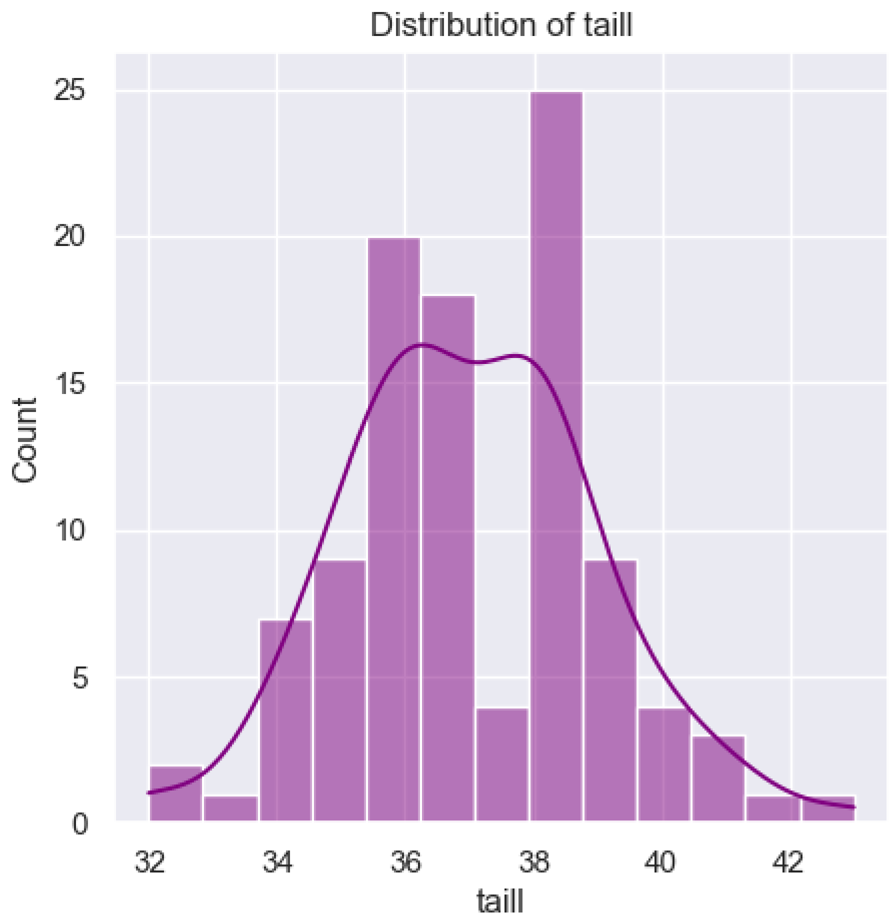

```
/Users/parz/miniforge3/envs/data-science/lib/python3.9/site-packages/sea
born/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated
and will be removed in a future version. Convert inf values to NaN befor
e operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



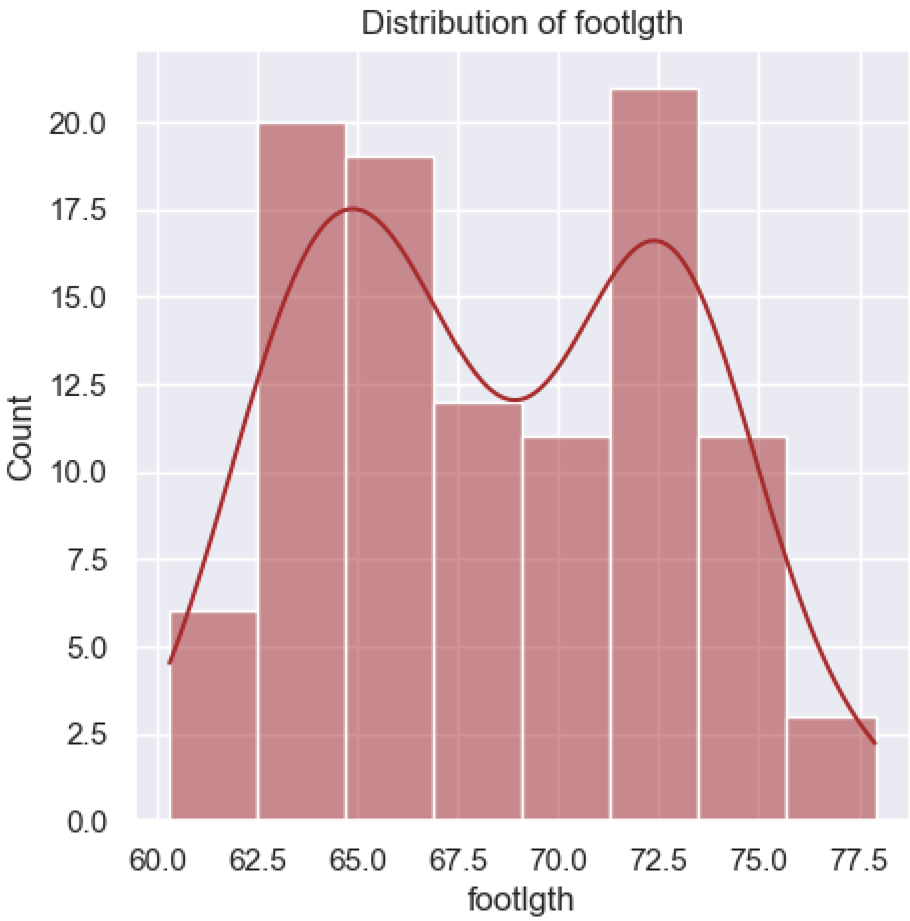

```
/Users/parz/miniforge3/envs/data-science/lib/python3.9/site-packages/sea
born/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated
and will be removed in a future version. Convert inf values to NaN befor
e operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



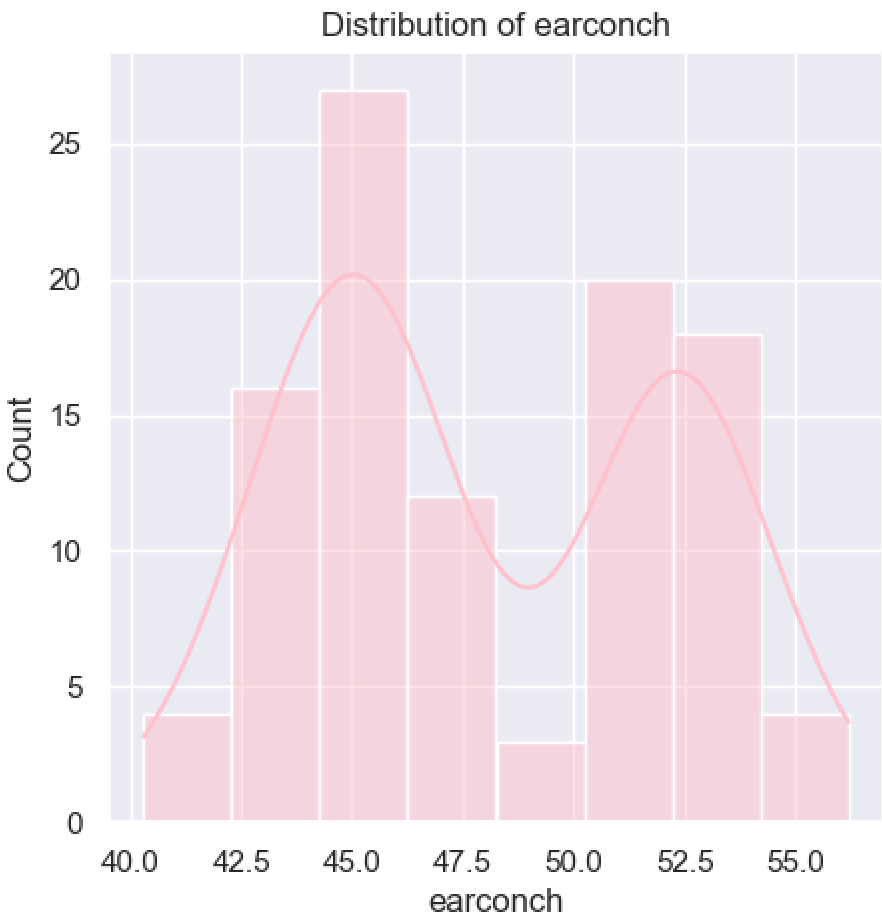

```
/Users/parz/miniforge3/envs/data-science/lib/python3.9/site-packages/sea
born/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated
and will be removed in a future version. Convert inf values to NaN befor
e operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



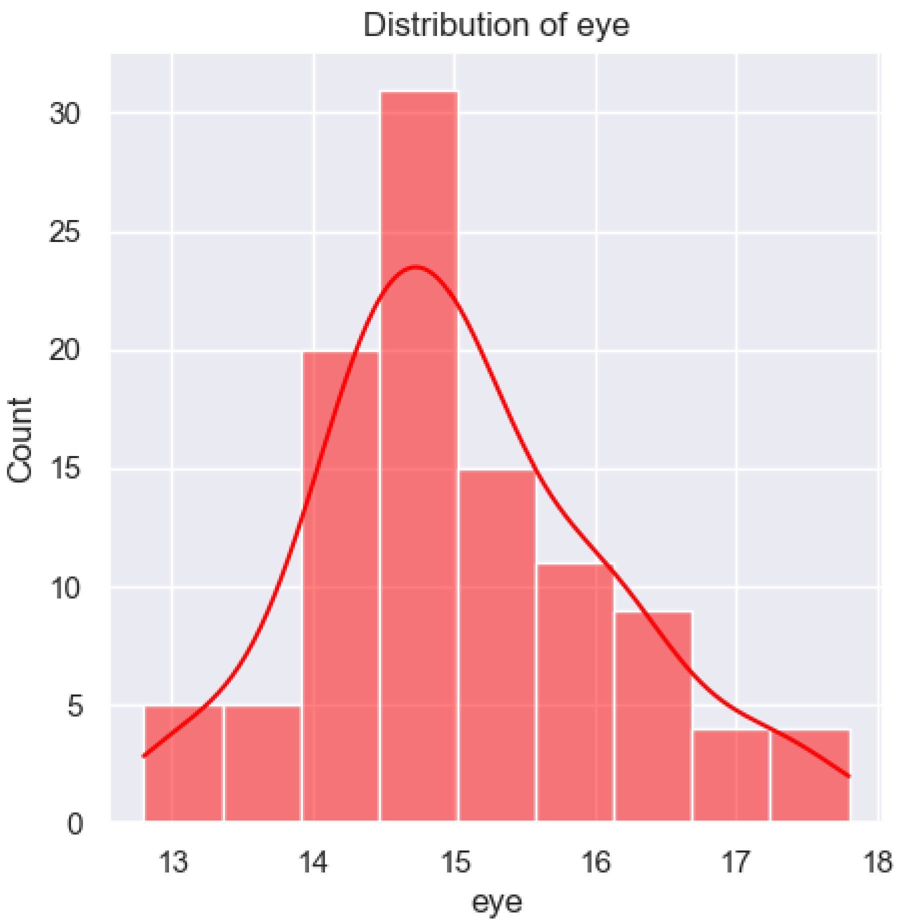

```
/Users/parz/miniforge3/envs/data-science/lib/python3.9/site-packages/sea
born/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated
and will be removed in a future version. Convert inf values to NaN befor
e operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



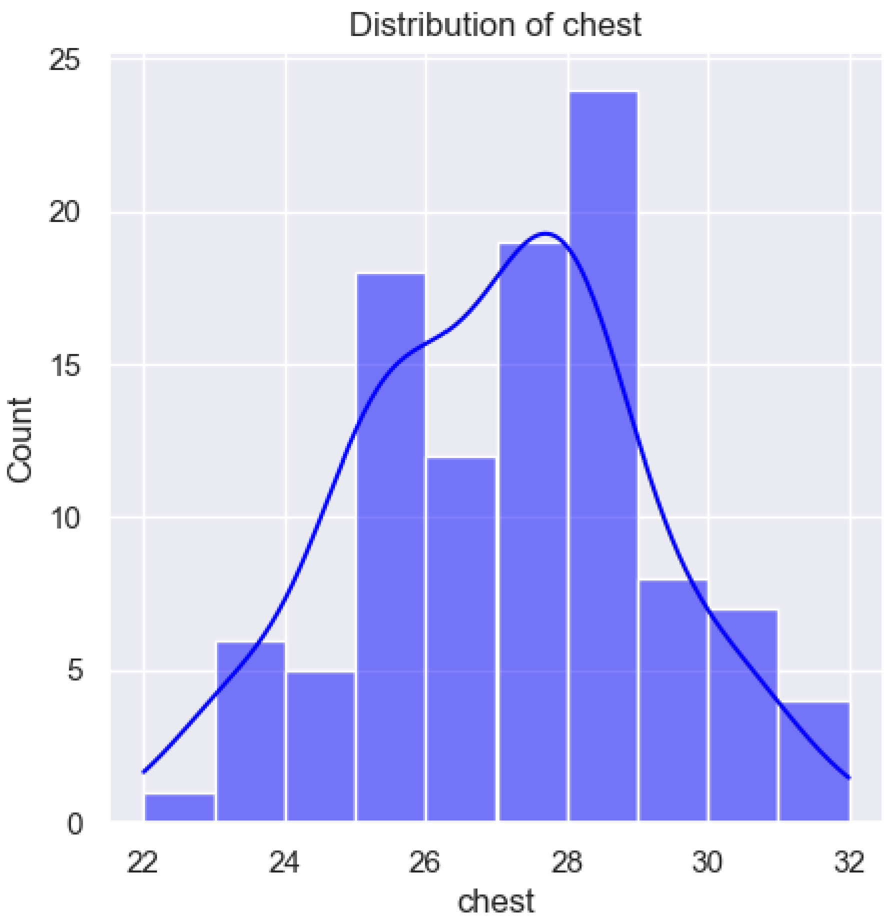

```
/Users/parz/miniforge3/envs/data-science/lib/python3.9/site-packages/sea
born/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated
and will be removed in a future version. Convert inf values to NaN befor
e operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



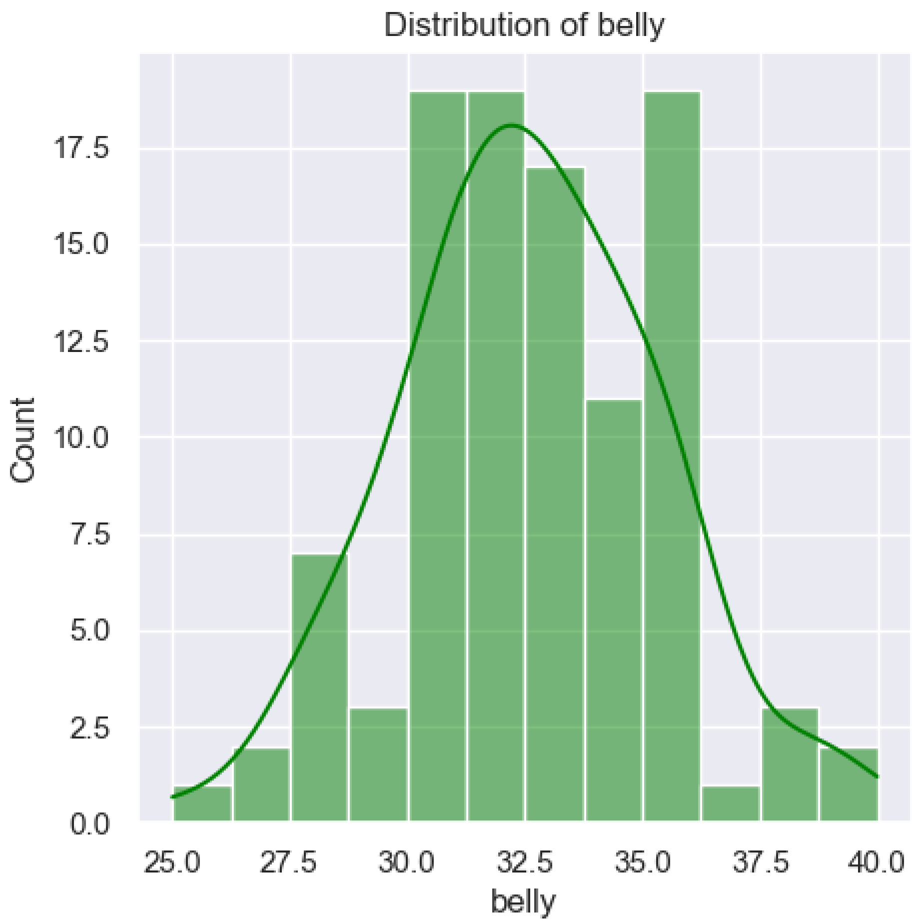

```
/Users/parz/miniforge3/envs/data-science/lib/python3.9/site-packages/sea
born/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated
and will be removed in a future version. Convert inf values to NaN befor
e operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```




```
/Users/parz/miniforge3/envs/data-science/lib/python3.9/site-packages/sea
born/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated
and will be removed in a future version. Convert inf values to NaN befor
e operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```




```
/Users/parz/miniforge3/envs/data-science/lib/python3.9/site-packages/sea
born/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated
and will be removed in a future version. Convert inf values to NaN befor
e operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [23]:
```
def  cat_num_feature_selector(dataframe):
    cat_features  =  [feature  for  feature  in  dataframe.columns  if  df[featur
    num_features  =  [feature  for  feature  in  dataframe.columns  if  df[featur
    return  cat_features,  num_features
```

In [24]:
```
cat,  num  =  cat_num_feature_selector(df)
```

In [25]:
```
corr_metrix  =  df[num].corr()
corr_metrix
```

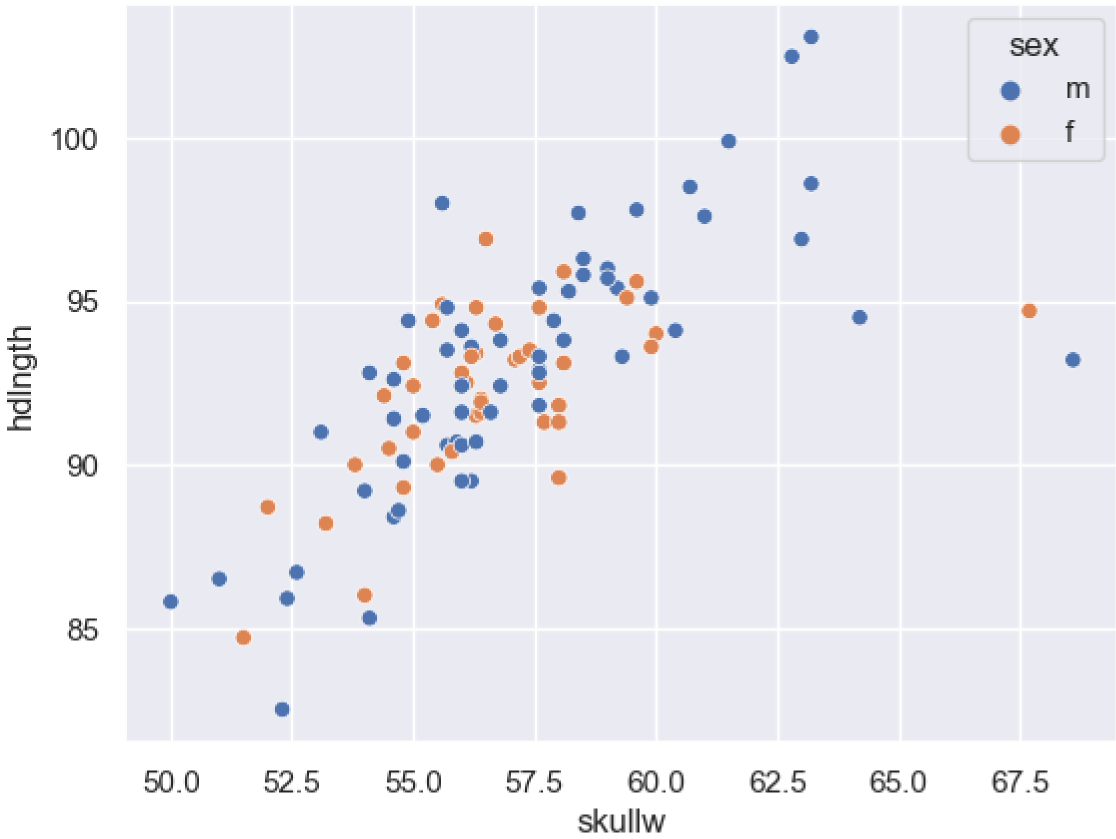
Out[25]:

	site	age	hdlngth	skullw	totlngth	taill	footlgth
site	1.000000	-0.131423	-0.163646	-0.083548	-0.260843	0.380444	-0.783009
age	-0.131423	1.000000	0.319022	0.285107	0.260280	0.118241	0.126190
hdlngth	-0.163646	0.319022	1.000000	0.710827	0.691094	0.287429	0.391605
skullw	-0.083548	0.285107	0.710827	1.000000	0.526413	0.255921	0.275059
totlngth	-0.260843	0.260280	0.691094	0.526413	1.000000	0.565646	0.444832
taill	0.380444	0.118241	0.287429	0.255921	0.565646	1.000000	-0.126277
footlgth	-0.783009	0.126190	0.391605	0.275059	0.444832	-0.126277	1.000000
earconch	-0.790716	0.053405	0.121463	-0.000537	0.154484	-0.385136	0.783050
eye	-0.036987	0.235553	0.347175	0.321991	0.247786	0.198134	0.005213
chest	-0.345494	0.334209	0.631498	0.629737	0.577890	0.174997	0.450590
belly	-0.175266	0.354298	0.562663	0.451838	0.519465	0.294493	0.302584

In [28]:
```
df.columns
```

Out[28]:
```
Index(['site', 'age', 'hdlngth', 'skullw', 'totlngth', 'taill', 'footlgt h',
       'earconch', 'eye', 'chest', 'belly'],
      dtype='object')
```

In [43]:
```
sns.scatterplot(data=df1, x='skullw', y='hdlngth', hue='sex')
plt.show()
```



In [44]:
```
# split the data into training and for test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df1.drop(columns='hdl
                                                    df1['hdlngth'],
                                                    test_size=0.25,
                                                    random_state=42)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[44]:
```
((75, 13), (26, 13), (75,), (26,))
```

In [45]:
```
cat, num = cat_num_feature_selector(X_train)
```

In [46]:
```
transformer = ColumnTransformer([
    ('encoder', OneHotEncoder(), cat),
    ('scaler', StandardScaler(), num)
], remainder='passthrough')

X_train_transformed = transformer.fit_transform(X_train)
X_test_transformed = transformer.transform((X_test))
```

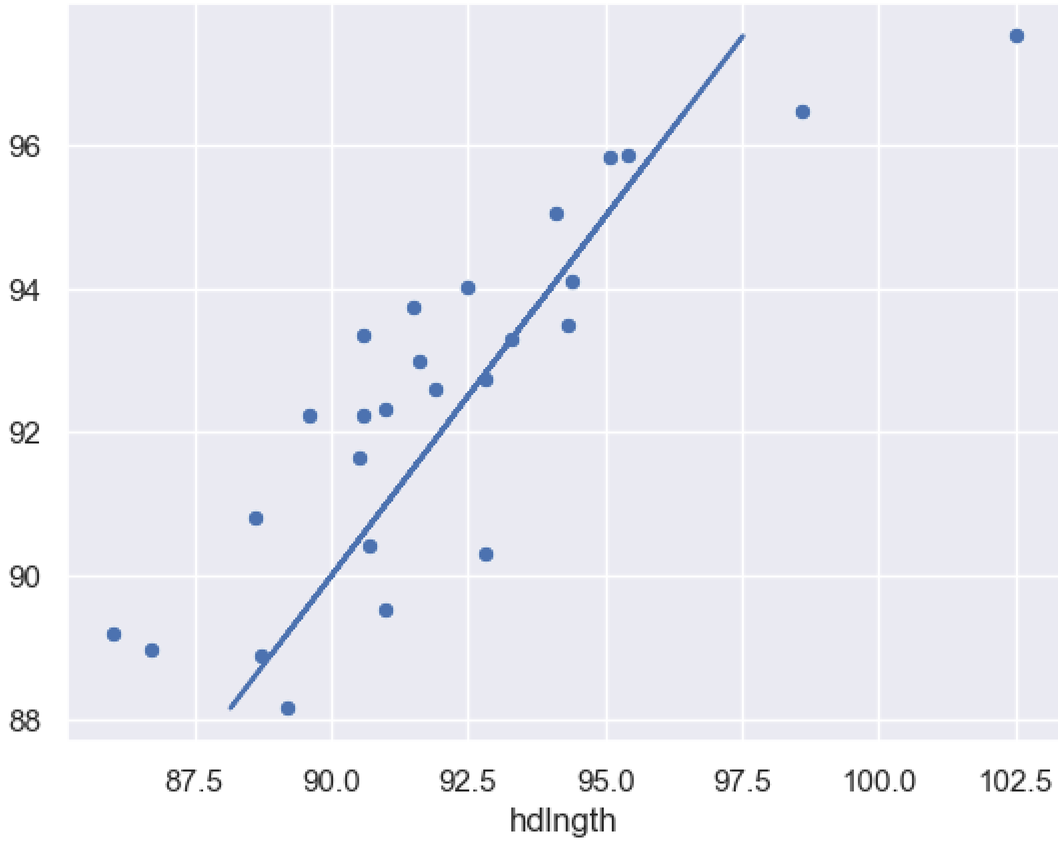
In [47]:
```
df1 = df1.dropna()
```

In [48]:
```
model = RandomForestRegressor()
model.fit(X_train_transformed, y_train)
y_pred = model.predict(X_test_transformed)
```

In [50]:
```
mse = mean_squared_error(y_test, y_pred)
rmse = (mse)**0.5
r2 = r2_score(y_test, y_pred)
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
print(f"R2 score: {r2}")
```

```
MSE: 3.5157507307692266
RMSE: 1.8750335279053616
R2 score: 0.6992377331031833
```

In [51]:
```
sns.scatterplot(x=y_test, y=y_pred)
plt.plot(y_pred, model.predict(X_test_transformed))
plt.show()
```

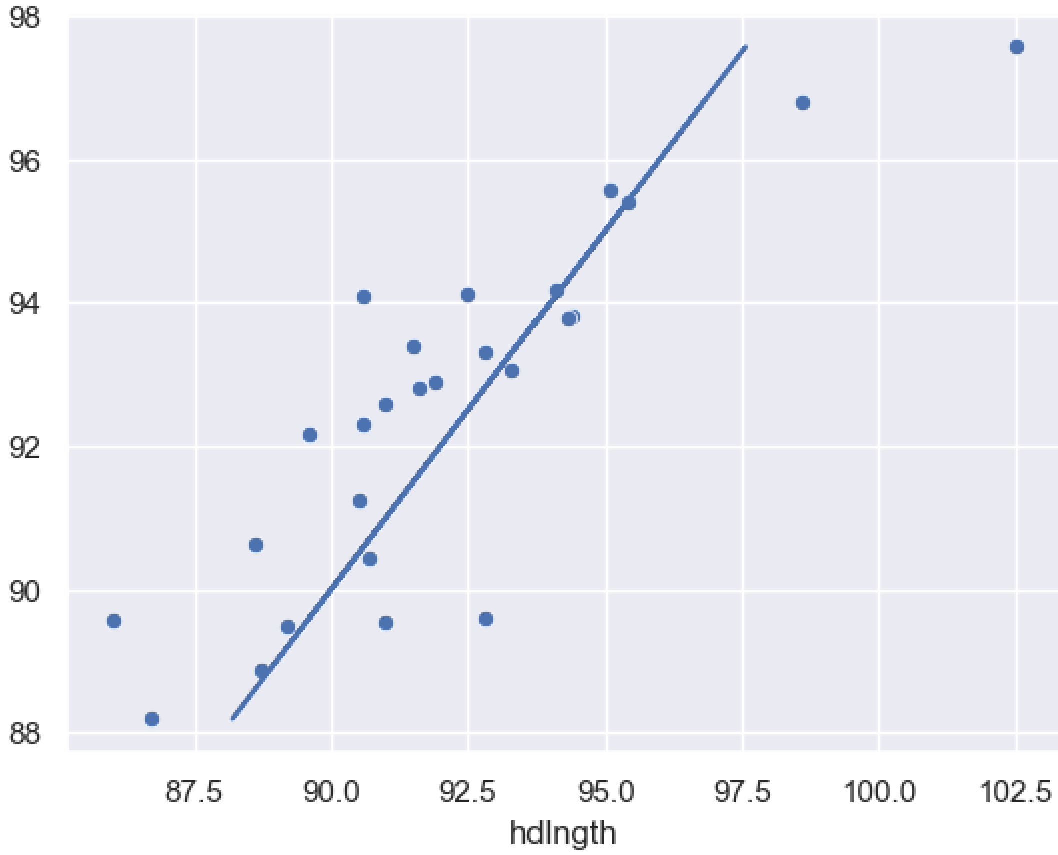


In [52]:
```
model = GradientBoostingRegressor()
model.fit(X_train_transformed, y_train)
y_pred = model.predict(X_test_transformed)
```

In [53]:
```
mse = mean_squared_error(y_test, y_pred)
rmse = (mse)**0.5
r2 = r2_score(y_test, y_pred)
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
print(f"R2 score: {r2}")
```

MSE: 3.605053581475904
RMSE: 1.8986978647156856
R2 score: 0.6915981335195656

In [54]:
```
sns.scatterplot(x=y_test, y=y_pred)
plt.plot(y_pred, model.predict(X_test_transformed))
plt.show()
```



In [ ]:

Experiment - 4

1. Logistic Regression

Logistic Regression is a supervised learning algorithm used for binary and multi-class classification problems. Unlike linear regression, which predicts continuous values, logistic regression predicts probabilities and maps them to discrete classes.

1.1. Sigmoid Function

Logistic regression uses the sigmoid (logistic) function to map input values to a probability range of (0,1):

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

1.2. Variants

1. **Binary Logistic Regression**: Used for two-class classification (e.g., spam vs. not spam).

2. **Multinomial Logistic Regression**: Used for multi-class classification.

3. **Ordinal Logistic Regression**: Used for ordered categories.

1.3. Applications

- Email spam classification.
- Credit risk assessment (loan default prediction).

In [12]:
```
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import pprint
import pickle
import itertools
```

In [2]:
```
df = pd.read_csv('breast-cancer.csv')
```

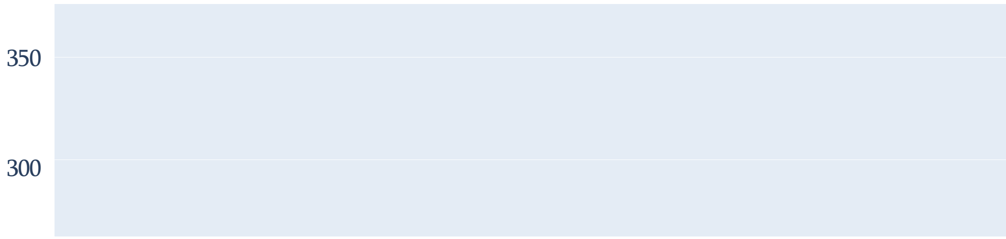
In [3]:
```
df.head()
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoo
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	

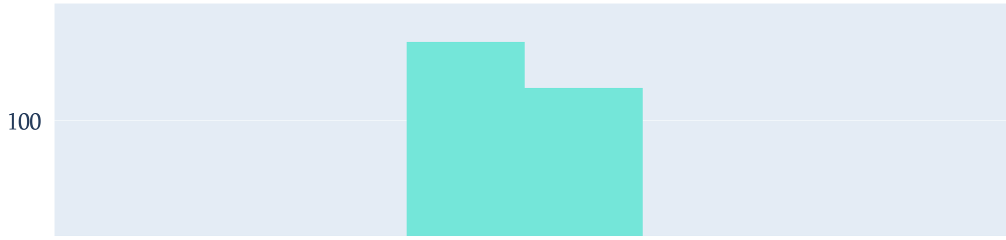
5 rows × 32 columns

In [4]:
```
px.histogram(data_frame=df,  x='diagnosis',  color='diagnosis',color_discre
```

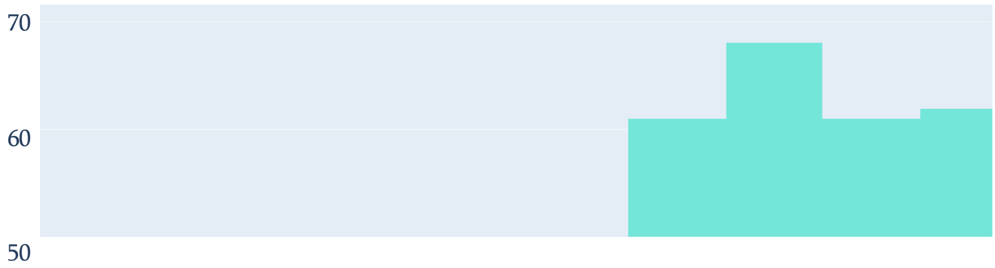


In [5]:

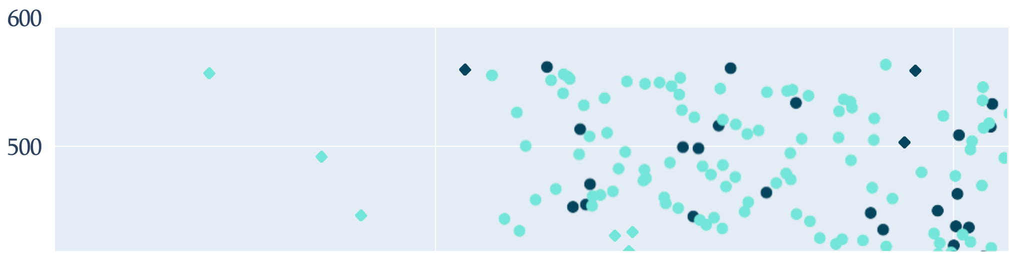
```
px.histogram(data_frame=df,x='area_mean',color='diagnosis',color_discrete
```



In [6]:
```
px.histogram(data_frame=df,x='perimeter_mean',color='diagnosis',color_dis
```



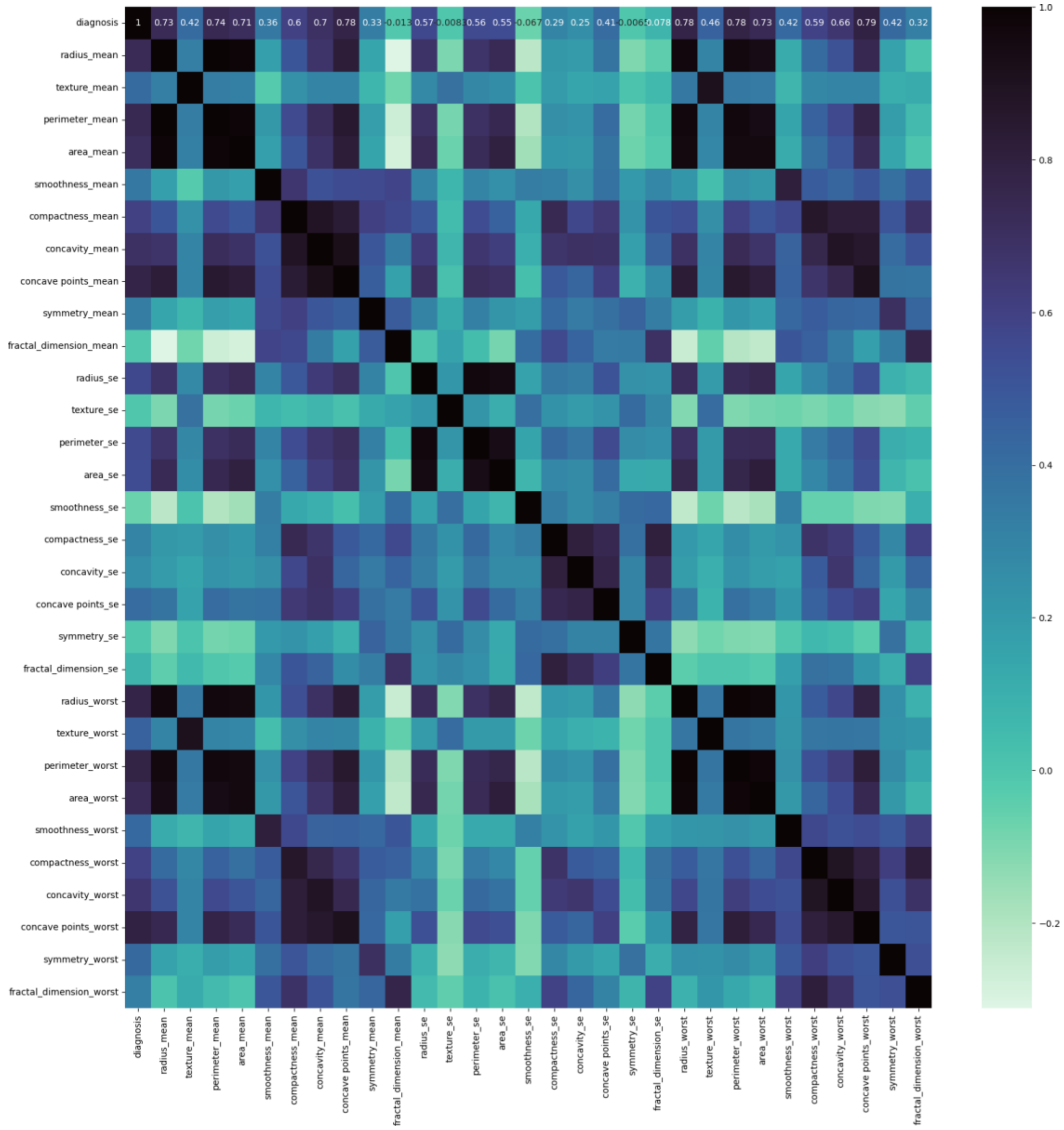
In [7]:
```
px.scatter(data_frame=df,x='symmetry_worst',color='diagnosis',color_discr
```



In [8]:
```
df.drop('id', axis=1, inplace=True) #redundant columns
```

In [9]:
```
df['diagnosis'] = (df['diagnosis'] == 'M').astype(int)
```

In [10]:
```
corr  =  df.corr()
plt.figure(figsize=(20,20))
sns.heatmap(corr, cmap='mako_r',annot=True)
plt.show()
```



In [14]:

```
# Get the absolute value of the correlation
cor_target = abs(corr["diagnosis"])

# Select highly correlated features (thresold = 0.2)
relevant_features = cor_target[cor_target>0.2]

# Collect the names of the features
names = [index for index, value in relevant_features.items()]

# Drop the target variable from the results
names.remove('diagnosis')

# Display the results
pprint.pprint(names)
```

```
['radius_mean',
 'texture_mean',
 'perimeter_mean',
 'area_mean',
 'smoothness_mean',
 'compactness_mean',
 'concavity_mean', 'concave
 points_mean',
 'symmetry_mean',
 'radius_se', 'perimeter_se',
 'area_se', 'compactness_se',
 'concavity_se', 'concave
 points_se', 'radius_worst',
 'texture_worst',
 'perimeter_worst',
 'area_worst',
 'smoothness_worst',
 'compactness_worst',
 'concavity_worst', 'concave
 points_worst',
 'symmetry_worst',
 'fractal_dimension_worst']
```

In [15]:

```
X = df[names].values
y = df['diagnosis'].values
```

In [16]:

```
deftrain_test_split(X, y, random_state=42, test_size=0.2): """
    Splits the data into training and testing sets.

    Parameters:
        X (numpy.ndarray): Features array of shape (n_samples, n_features y
        (numpy.ndarray): Target array of shape (n_samples,). random_state (int):
        Seed for the random number generator. Default test_size (float): Proportion
        of samples to include in the test s

    Returns:
        Tuple[numpy.ndarray]: A tuple containing X_train, X_test, y_train
    """
    # Get number of samples
    n_samples = X.shape[0]

    # Set the seed for the random number generator
    np.random.seed(random_state)

    # Shuffle the indices
    shuffled_indices = np.random.permutation(np.arange(n_samples))
    # Determine the size of the test set
    test_size = int(n_samples * test_size)

    # Split the indices into test and train test_indices =
    shuffled_indices[:test_size] train_indices =
    shuffled_indices[test_size:]

    # Split the features and target arrays into test and train X_train, X_test =
    X[train_indices], X[test_indices] y_train, y_test =
    y[train_indices], y[test_indices]
```

```
    return X_train, X_test, y_train, y_test
```

In [17]:

```
X_train, X_test, y_train, y_test = train_test_split(X,y)
```

In [18]:

```
def standardize_data(X_train, X_test):
    """
    Standardizes the input data using mean and standard deviation.

    Parameters:
        X_train (numpy.ndarray): Training data.
        X_test (numpy.ndarray): Testing data.

    Returns:
        Tuple of standardized training and testing data.
    """
    # Calculate the mean and standard deviation using the training data
    mean = np.mean(X_train, axis=0)
    std = np.std(X_train, axis=0)

    # Standardize the data
    X_train = (X_train - mean) / std
    X_test = (X_test - mean) / std

    return X_train, X_test
```

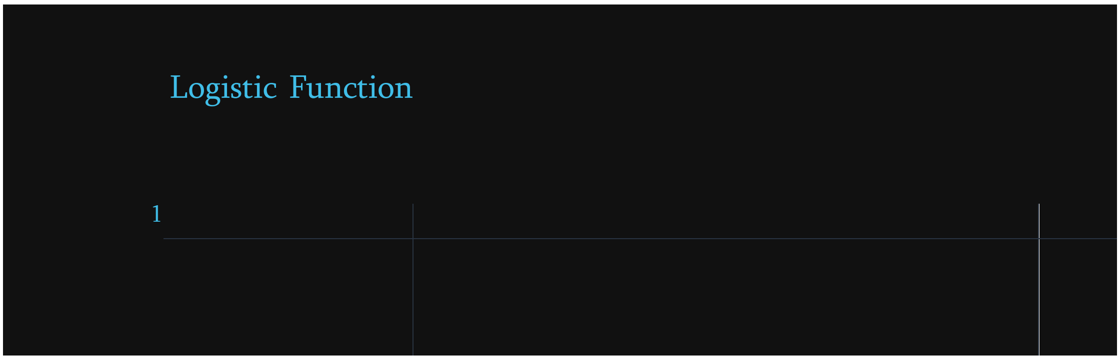
In [19]:

```
X_train, X_test = standardize_data(X_train, X_test)
    Compute the sigmoid function for a given input.

    The sigmoid function is a mathematical function used in logistic regr
    to map any real-valued number to a value between 0 and 1.

    Parameters:
        z (float or numpy.ndarray): The input value(s) for which to compu

    Returns:
        float or numpy.ndarray: The sigmoid of the input value(s).

    Example:
        >>> sigmoid(0)
        0.5
    """
    # Compute the sigmoid function using the formula: 1 / (1 + e^(-z)).
    sigmoid_result = 1 / (1 + np.exp(-z))
```

In [20]:

```
    # Return the computed sigmoid value.
    return sigmoid_result
fig = px.line(x=z, y=sigmoid(z),title='Logistic Function',template='plotl
fig.update_layout(
    title_font_color="#41BEE9",
    xaxis=dict(color="#41BEE9"),
    yaxis=dict(color="#41BEE9")
```

```
)
fig.show()
```



In [21]:

```
class LogisticRegression:
    """
    Logistic Regression model.

    Parameters:
        learning_rate (float): Learning rate for the model.

    Methods:
        initialize_parameter(): Initializes the parameters of the model.
        sigmoid(z): Computes the sigmoid activation function for given in
        forward(X): Computes forward propagation for given input X.
        compute_cost(predictions): Computes the cost function for given p
        compute_gradient(predictions): Computes the gradients for the mod
        fit(X, y, iterations, plot_cost): Trains the model on given input
        predict(X): Predicts the labels for given input X.
    """

    def __init__(self, learning_rate=0.0001):
        np.random.seed(1)
        self.learning_rate = learning_rate

    def initialize_parameter(self):
        """
        Initializes the parameters of the model.
        """
```

```
        self.W = np.zeros(self.X.shape[1]) self.b = 0.0


    def forward(self, X): """
        Computes forward propagation for given input X.

        Parameters:
            X (numpy.ndarray): Input array.

        Returns:
            numpy.ndarray: Output array.
        """
        #print(X.shape, self.W.shape)


        Z = np.matmul(X, self.W) + self.b A =
        sigmoid(Z)
        return A

    def compute_cost(self, predictions): """
        Computes the cost function for given predictions.

        Parameters:
            predictions (numpy.ndarray): Predictions of the model.

        Returns:
            float: Cost of the model.
        """
        m = self.X.shape[0]            # number of training examples
        # compute the cost
        cost = np.sum((-np.log(predictions + 1e-8) * self.y) + (-np.log(1 1 - self.y))      # we
                  are adding small value epsilon to avoi
        cost = cost / m
        return cost

    def compute_gradient(self, predictions): """
        Computes the gradients for the model using given predictions.

        Parameters:
            predictions (numpy.ndarray): Predictions of the model.
        """
        # get training shape
        m = self.X.shape[0]


                  # compute gradients
        self.dW = np.matmul(self.X.T, (predictions - self.y)) self.dW =
        np.array([np.mean(grad) for grad in self.dW])

        self.db = np.sum(np.subtract(predictions, self.y))

        # scale gradients
        self.dW = self.dW * 1 / m self.db
        = self.db * 1 / m


    def fit(self, X, y, iterations, plot_cost=True): """
```

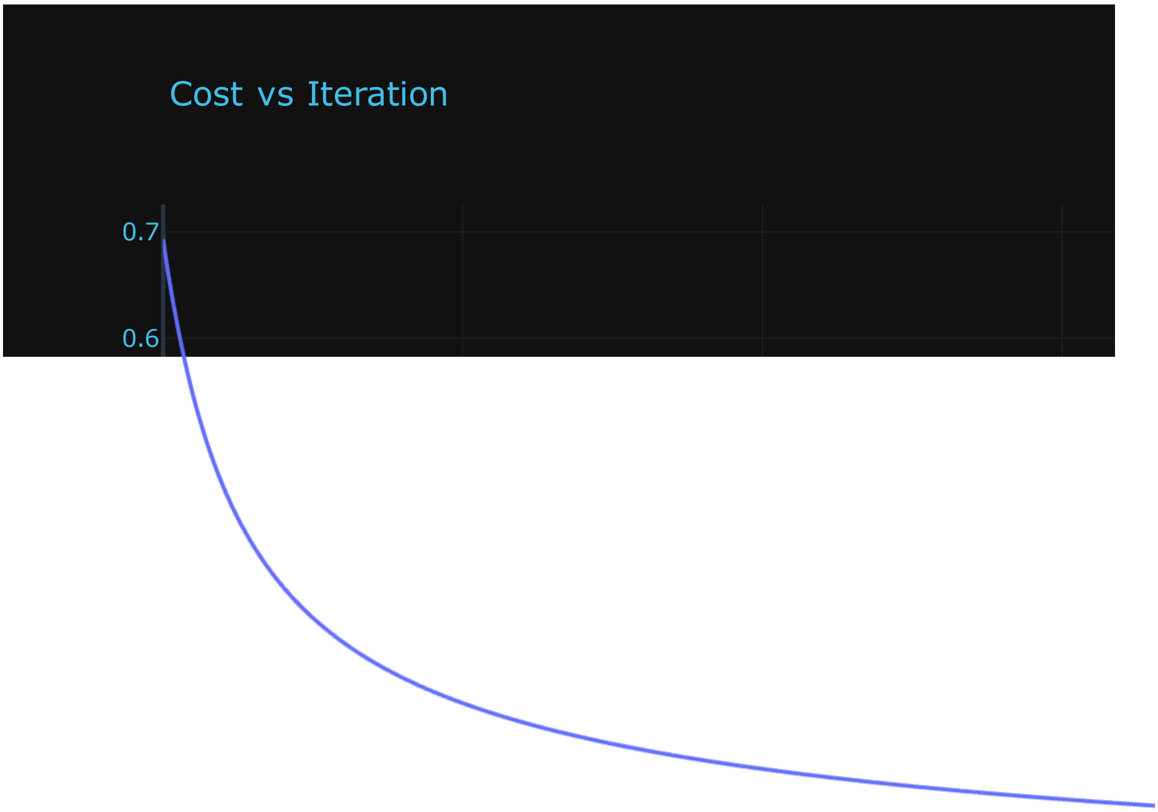
```
        Trains the model on given input X and labels y for specified iter

        Parameters:
            X (numpy.ndarray): Input features array of shape (n_samples, y
            (numpy.ndarray): Labels array of shape (n_samples, 1) iterations (int):
            Number of iterations for training. plot_cost (bool): Whether to plot
            cost over iterations or not

        Returns:
            None.
        """
        self.X = X
        self.y = y

        self.initialize_parameter() costs

        = []
        for i in range(iterations):
            # forward propagation
            predictions = self.forward(self.X)

            # compute cost
            cost = self.compute_cost(predictions) costs.append(cost)

            # compute gradients
            self.compute_gradient(predictions)

            # update parameters
            self.W = self.W - self.learning_rate * self.dW self.b = self.b
            - self.learning_rate * self.db

            # print cost every 100 iterations
            if i % 10000 == 0:
                print("Cost after iteration {}: {}".format(i, cost))

        if plot_cost:
            fig = px.line(y=costs,title="Cost vs Iteration",template="plo
            fig.update_layout(
                title_font_color="#41BEE9",
                xaxis=dict(color="#41BEE9",title="Iterations"),
                yaxis=dict(color="#41BEE9",title="cost")
            )
            fig.show()
    def predict(self, X): """
        Predicts the labels for given input X.

        Parameters:
            X (numpy.ndarray): Input features array.

        Returns:
            numpy.ndarray: Predicted labels.
        """
        predictions = self.forward(X)
        return np.round(predictions)


    def save_model(self, filename=None): """
```

```
        Save the trained model to a file using pickle.

        Parameters:
        filename (str): The name of the file to save the model to.
        """
        model_data = {
            'learning_rate': self.learning_rate,
            'W': self.W,
            'b':  self.b
        }

        with open(filename, 'wb') as file:
            pickle.dump(model_data, file)

    @classmethod
    def load_model(cls, filename):
        """
        Load a trained model from a file using pickle.

        Parameters:
            filename (str): The name of the file to load the model from.

        Returns:
            LogisticRegression: An instance of the LogisticRegression cla
        """
        with open(filename, 'rb') as file:
            model_data = pickle.load(file)

        # Create a new instance of the class and initialize it with the l
        loaded_model = cls(model_data['learning_rate'])
        loaded_model.W = model_data['W']
        loaded_model.b = model_data['b']

        return loaded_model
```

In [22]:

```
lg = LogisticRegression()
lg.fit(X_train, y_train,100000)
```

```
Cost after iteration 0: 0.6931471605599454
Cost after iteration 10000: 0.25707783705582454
Cost after iteration 20000: 0.19529178673689726
Cost after iteration 30000: 0.16685820756163852
Cost after iteration 40000: 0.149789395486765
Cost after iteration 50000: 0.13818761340315544
Cost after iteration 60000: 0.1296814121248933
Cost after iteration 70000: 0.1231144039988139
Cost after iteration 80000: 0.11785163708790082
Cost after iteration 90000: 0.11351377138600201
```



In [ ]:

Experiment - 5

Polynomial Regression

Polynomial Regression extends linear regression by introducing polynomial terms to capture non-linearity.

2.1. Model Equation

Polynomial regression models a higher-degree relationship:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + ... + \beta_n X^n + \epsilon$$

where n is the degree of the polynomial.

2.2. Key Concepts

- **Overfitting**: A high-degree polynomial may fit the training data too closely and generalize poorly.
- **Underfitting**: A low-degree polynomial may not capture the complexity of the data.

2.3. Selection of Polynomial Degree

To avoid overfitting or underfitting, techniques like **Cross-Validation** and **Regularization (Ridge/Lasso)** can be used.

2.4. Applications

- Predicting non-linear trends in financial markets.
- Estimating complex physical phenomena (e.g., projectile motion).

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
dataset = pd.read_csv('Position_Salaries.csv')
```

In [3]:
```
dataset.head()
```

Out[3]:

	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000

In [4]:
```
#Dependent feature
y  =  np.asarray(dataset['Salary'].values.tolist())

# Independent Feature
X  =  np.asarray(dataset['Level'].values.tolist())
```

In [5]:
```
X = X.reshape(-1,1)
```

In [6]:
```
y = y.reshape(len(y),1) # Changing the shape from (50,) to (50,1)
```

In [7]:
```
def poly_features(features, X):
  data = pd.DataFrame(np.zeros((X.shape[0],features)))
  for i in range(1,features+1):
    data.iloc[:,i-1] = (X**i).reshape(-1,1)
  X_poly  =  np.array(data.values.tolist())
  return X_poly
```

In [8]:
```
def split_data(X,y,test_size=0.2,random_state=0):
    np.random.seed(random_state)                  #set the seed for repro
    indices = np.random.permutation(len(X))       #shuffling the indices
    data_test_size = int(X.shape[0] * test_size)  #Get the test size

    #Separating the Independent and Dependent features into the Train and
    train_indices = indices[data_test_size:]
    test_indices = indices[:data_test_size]
    X_train = X[train_indices]
    y_train = y[train_indices]
    X_test = X[test_indices]
    y_test = y[test_indices]
    return X_train, y_train, X_test, y_test
```

In [9]:
```
  prediction_values = list()
  for i in range(X.shape[0]):
    value = regressor.predict(W_trained,X[i])
    prediction_values.append(value)
  return prediction_values
```

In [10]:
```
class  polynomialRegression():

   def __init__(self):
      #No instance Variables required
      pass

   def forward(self,X,y,W): """
      Parameters:
      X (array) : Independent Features
      y (array) : Dependent Features/ Target Variable W
      (array) : Weights

      Returns:
      loss (float) : Calculated Sqaured Error Loss for y and y_pred y_pred (array) :
      Predicted Target Variable
      """
      y_pred = sum(W * X)
                        loss = ((y_pred-y)**2)/2            #Loss = Squared Error, we introduce 1/2 f
      return loss, y_pred

   def updateWeights(self,X,y_pred,y_true,W,alpha,index): """
      Parameters:
      V (array) : Independent Features
      y_pred (array) : Predicted Target Variable
      y_true (array) : Dependent Features/ Target Variable W
      (array) : Weights
      alpha (float) : learning rate
      index (int) : Index to fetch the corresponding values of W, X and y

      Returns:
      W (array) : Update Values of Weight """
      for i in range(X.shape[1]):
                              #alpha = learning rate, rest of the RHS is derivative of loss funct
         W[i] -= (alpha * (y_pred-y_true[index])*X[index][i])
      return W

   def train(self, X, y, epochs=10, alpha=0.001, random_state=0): """
      Parameters:
      X (array) : Independent Feature
      y (array) : Dependent Features/ Target Variable
      epochs (int) : Number of epochs for training, default value is 10 alpha (float)
      : learning rate, default value is 0.001

      Returns:
      y_pred (array) : Predicted Target Variable
      loss (float) : Calculated Sqaured Error Loss for y and y_pred """

      num_rows = X.shape[0] #Number of Rows
      num_cols = X.shape[1] #Number of Columns
      W = np.random.randn(1,num_cols) / np.sqrt(num_rows) #Weight Initializ

      #Calculating Loss and Updating Weights
      train_loss = []
      num_epochs = []
```

```
        train_indices = [i for i in range(X.shape[0])]
        for j in range(epochs): cost=0
            np.random.seed(random_state)
            np.random.shuffle(train_indices) for i in
            train_indices:
                loss, y_pred = self.forward(X[i],y[i],W[0]) cost+=loss
                W[0] = self.updateWeights(X,y_pred,y,W[0],alpha,i) train_loss.append(cost)
            num_epochs.append(j)
        return W[0], train_loss, num_epochs

    def test(self, X_test, y_test, W_trained): """
        Parameters:
        X_test (array) : Independent Features from the Test Set
        y_test (array) : Dependent Features/ Target Variable from the Test Se W_trained
        (array) : Trained Weights
        test_indices (list) : Index to fetch the corresponding values of W_tr
                              X_test and y_test

        Returns:
        test_pred (list) : Predicted Target Variable
        test_loss (list) : Calculated Sqaured Error Loss for y and y_pred """
        test_pred = []
        test_loss = []
        test_indices = [i for i in range(X_test.shape[0])]
        for i in test_indices:
                loss, y_test_pred = self.forward(X_test[i], W_trained, y_test[i])
                test_pred.append(y_test_pred)
                test_loss.append(loss)
        return test_pred, test_loss


    def predict(self, W_trained, X_sample): prediction =
        sum(W_trained * X_sample) return prediction

    def plotLoss(self, loss, epochs): """
        Parameters:
        loss (list) : Calculated Sqaured Error Loss for y and y_pred epochs
        (list): Number of Epochs

        Returns: None
        Plots a graph of Loss vs Epochs """
        plt.plot(epochs, loss)
        plt.xlabel('Number of Epochs')
        plt.ylabel('Loss') plt.title('Plot
        Loss') plt.show()
```

In [11]:

```
X = np.asarray(dataset['Level'].values.tolist())
```

In [12]:

```
X = X.reshape(-1,1)
```

In [13]:
```
X
```

Out[13]:
```
array([[ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8],
       [ 9],
       [10]])
```

In [14]:
```
X = poly_features(2,X)
```

```
/var/folders/lb/_lc8jsw57gg97jlqvd8bn5x80000gn/T/ipykernel_28117/8636191
3.py:4: FutureWarning: Setting an item of incompatible dtype is deprecat
ed and will raise in a future error of pandas. Value '[[ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]]' has dtype incompatible with float64, please explicitly cast to a
compatible dtype first.
  data.iloc[:,i-1] = (X**i).reshape(-1,1)
```

In [15]:
```
#Adding the feature X0 = 1, so we have the equation: y =  W0 + (W1 * X1)
X = np.concatenate((X,np.ones((10,1))), axis = 1)
```

In [16]:
```
X
```

Out[16]:
```
array([[  1.,   1.,   1.],
       [  2.,   4.,   1.],
       [  3.,   9.,   1.],
       [  4.,  16.,   1.],
       [  5.,  25.,   1.],
       [  6.,  36.,   1.],
       [  7.,  49.,   1.],
       [  8.,  64.,   1.],
       [  9.,  81.,   1.],
       [ 10., 100.,   1.]])
```

In [17]:
```
y
```

Out[17]:
```
       [  50000],
       [  60000],
       [  80000],
       [ 110000],
       [ 150000],
       [ 200000],
       [ 300000],
       [ 500000],
       [1000000]])
```

In [24]:
```
X_train, y_train, X_test, y_test = split_data(X,y)
```

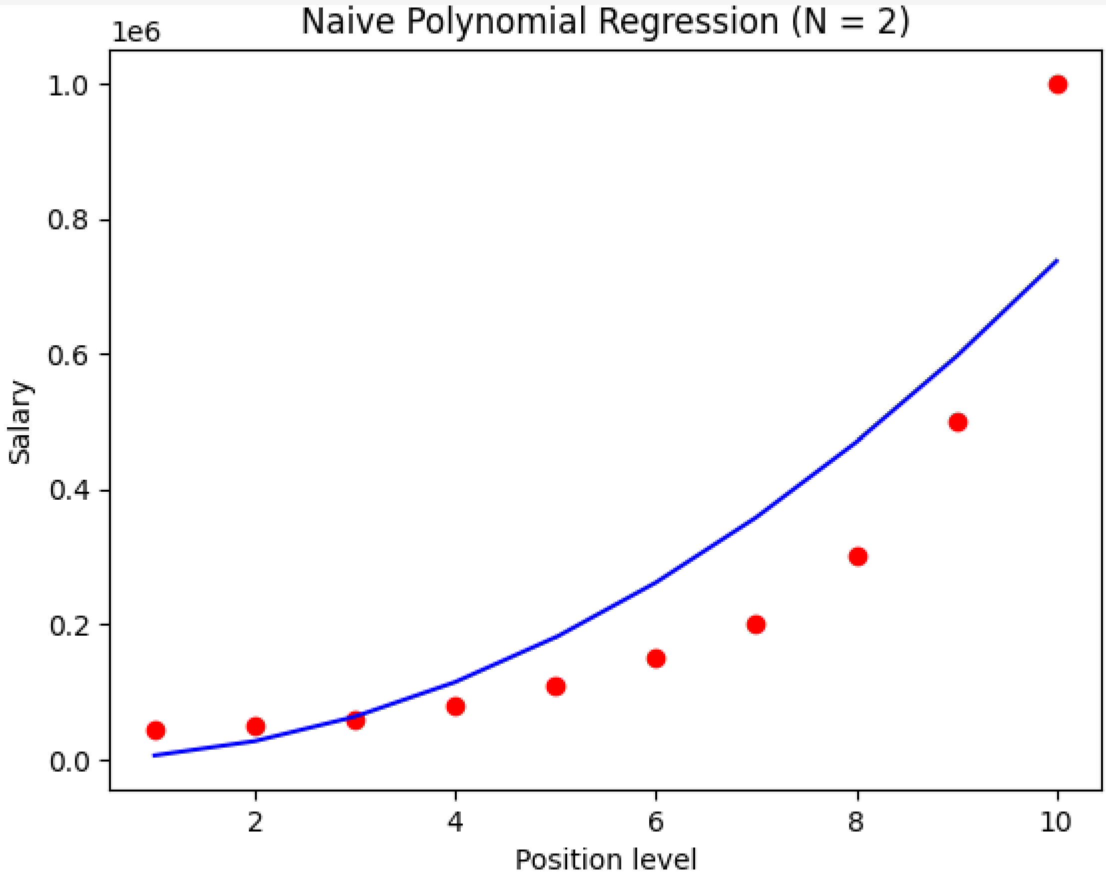
In [25]:
```
regressor = polynomialRegression()
```

In [28]:
```
W_trained, train_loss, num_epochs = regressor.train(X_train, y_train, epo
```

In [29]:
```
test_pred, test_loss = regressor.test(X_test, y_test, W_trained)
```

In [30]:
```
pred_plot = pred_to_plot(W_trained,X)
```

In [31]:
```
plt.scatter(X[:,0], y, color = 'red')
plt.plot(X[:,0], pred_plot, color = 'blue')
plt.title('Naive Polynomial Regression (N = 2)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```



In [32]:
```
X = np.asarray(dataset['Level'].values.tolist())
```

In [33]:
```
X = X.reshape(-1,1)
```

In [34]:
```
X_poly = poly_features(4,X)
```

```
/var/folders/lb/_lc8jsw57gg97jlqvd8bn5x80000gn/T/ipykernel_28117/8636191
3.py:4: FutureWarning: Setting an item of incompatible dtype is deprecat
ed and will raise in a future error of pandas. Value '[[ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]]' has dtype incompatible with float64, please explicitly cast to a
compatible dtype first.
  data.iloc[:,i-1] = (X**i).reshape(-1,1)
```

In [35]:
```
X_poly = np.concatenate((X_poly,np.ones((10,1))), axis = 1)
```

In [36]:
```
X_poly
```

Out[36]:
```
array([[1.000e+00, 1.000e+00, 1.000e+00, 1.000e+00, 1.000e+00],
       [2.000e+00, 4.000e+00, 8.000e+00, 1.600e+01, 1.000e+00],
       [3.000e+00, 9.000e+00, 2.700e+01, 8.100e+01, 1.000e+00],
       [4.000e+00, 1.600e+01, 6.400e+01, 2.560e+02, 1.000e+00],
       [5.000e+00, 2.500e+01, 1.250e+02, 6.250e+02, 1.000e+00],
       [6.000e+00, 3.600e+01, 2.160e+02, 1.296e+03, 1.000e+00],
       [7.000e+00, 4.900e+01, 3.430e+02, 2.401e+03, 1.000e+00],
       [8.000e+00, 6.400e+01, 5.120e+02, 4.096e+03, 1.000e+00],
       [9.000e+00, 8.100e+01, 7.290e+02, 6.561e+03, 1.000e+00],
       [1.000e+01, 1.000e+02, 1.000e+03, 1.000e+04, 1.000e+00]])
```

In [37]:
```
y
```

Out[37]:
```
array([[  45000],
       [  50000],
       [  60000],
       [  80000],
       [ 110000],
       [ 150000],
       [ 200000],
       [ 300000],
       [ 500000],
       [1000000]])
```

In [38]:
```
X_train, y_train, X_test, y_test =split_data(X_poly,y)
```

In [39]:
```
regressor = polynomialRegression()
```

In [40]:
```
W_trained, train_loss, num_epochs = regressor.train(X_train, y_train, epo
```

In [41]:
```
test_pred, test_loss = regressor.test(X_test, y_test, W_trained)
```

In [42]:
```
X_poly
```

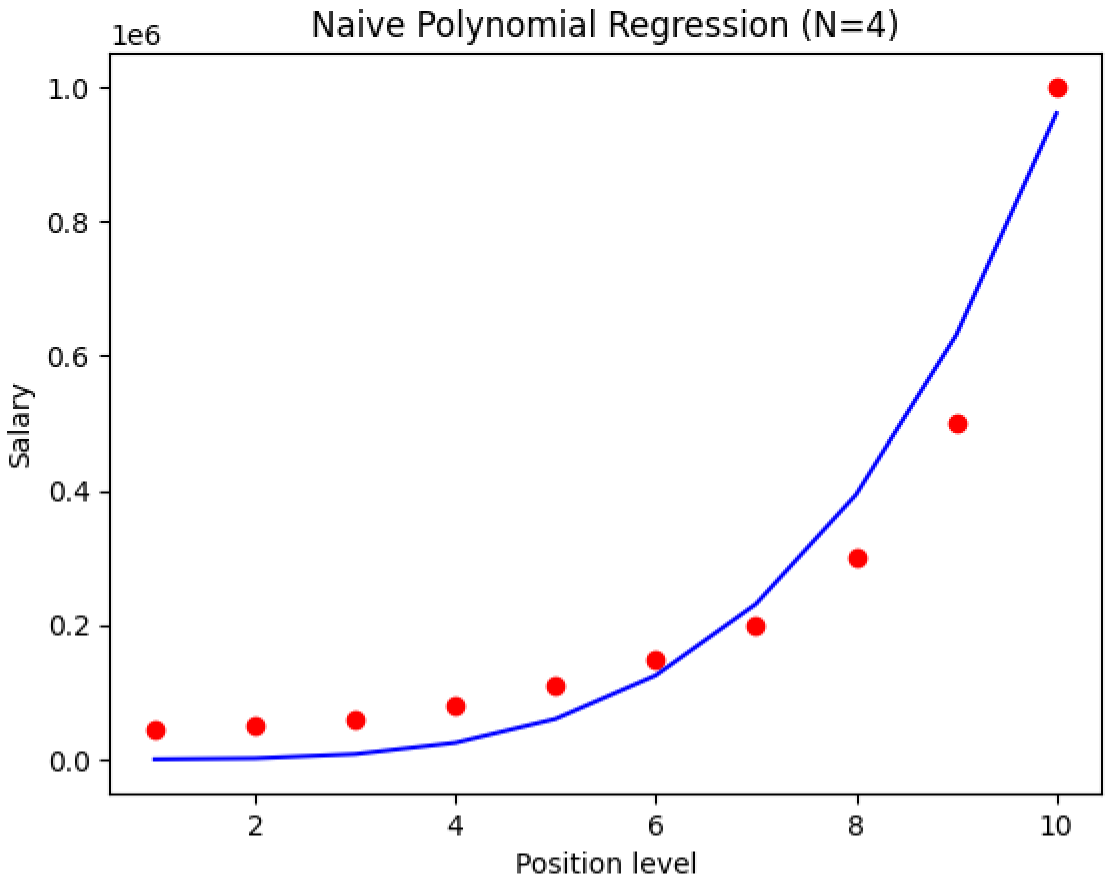
Out[42]:
```
array([[1.000e+00, 1.000e+00, 1.000e+00, 1.000e+00, 1.000e+00],
       [2.000e+00, 4.000e+00, 8.000e+00, 1.600e+01, 1.000e+00],
       [3.000e+00, 9.000e+00, 2.700e+01, 8.100e+01, 1.000e+00],
       [4.000e+00, 1.600e+01, 6.400e+01, 2.560e+02, 1.000e+00],
       [5.000e+00, 2.500e+01, 1.250e+02, 6.250e+02, 1.000e+00],
       [6.000e+00, 3.600e+01, 2.160e+02, 1.296e+03, 1.000e+00],
       [7.000e+00, 4.900e+01, 3.430e+02, 2.401e+03, 1.000e+00],
       [8.000e+00, 6.400e+01, 5.120e+02, 4.096e+03, 1.000e+00],
       [9.000e+00, 8.100e+01, 7.290e+02, 6.561e+03, 1.000e+00],
       [1.000e+01, 1.000e+02, 1.000e+03, 1.000e+04, 1.000e+00]])
```

In [43]:
```
pred_plot = pred_to_plot(W_trained,X_poly)
```

In [44]:
```
pred_plot
```

Out[44]:
```
[101.56686988148302,
 1570.332736753576,
 7876.252690310476,
 24783.645533405484,
 60351.39230382515,
 124932.93627428928,
 231176.28295245094,
 394024.0000808964,
 630713.2176371454,
 960775.6278336504]
```

In [45]:
```
plt.scatter(X_poly[:,0], y, color = 'red')
plt.plot(X_poly[:,0], pred_plot, color = 'blue')
plt.title('Naive Polynomial Regression (N=4)')
plt.xlabel('Position  level')
plt.ylabel('Salary')
plt.show()
```



In [46]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
```

In [47]:
```
X_sk = dataset.iloc[:, 1].values
y_sk = dataset.iloc[:, -1].values
```

In [48]:
```
X_sk = X_sk.reshape(-1,1)
y_sk = y_sk.reshape(-1,1)
```

In [49]:
```
# Constructing the polynomials of our Independent features
poly_reg = PolynomialFeatures(degree = 4)
X_poly_sk = poly_reg.fit_transform(X_sk)
```

In [50]:
```
#Get the shapes of X and y
print("The shape of the independent fatures are ",X_poly_sk.shape)
print("The shape of the dependent fatures are ",y_sk.shape)
```

```
The shape of the independent fatures are  (10, 5)
The shape of the dependent fatures are  (10, 1)
```

In [51]:
```
X_poly_sk
```

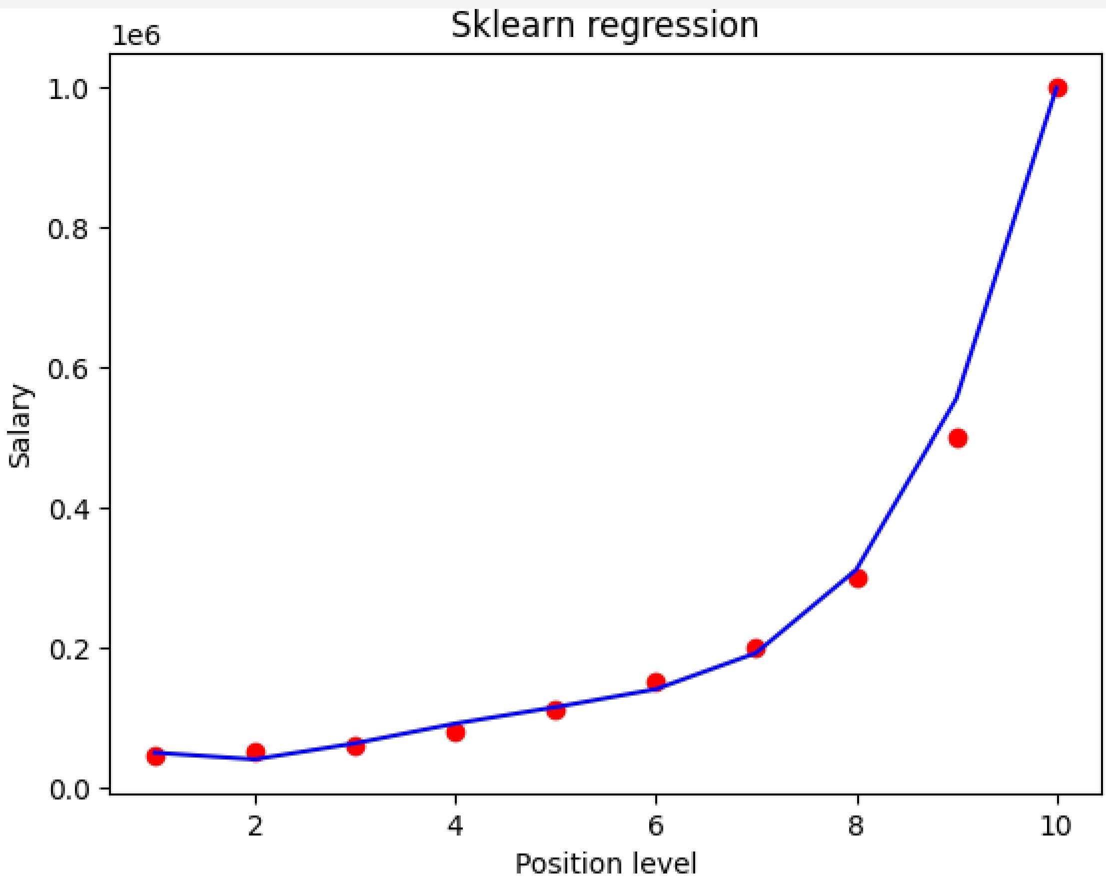
Out[51]:
```
array([[1.000e+00, 1.000e+00, 1.000e+00, 1.000e+00, 1.000e+00],
       [1.000e+00, 2.000e+00, 4.000e+00, 8.000e+00, 1.600e+01],
       [1.000e+00, 3.000e+00, 9.000e+00, 2.700e+01, 8.100e+01],
       [1.000e+00, 4.000e+00, 1.600e+01, 6.400e+01, 2.560e+02],
       [1.000e+00, 5.000e+00, 2.500e+01, 1.250e+02, 6.250e+02],
       [1.000e+00, 6.000e+00, 3.600e+01, 2.160e+02, 1.296e+03],
       [1.000e+00, 7.000e+00, 4.900e+01, 3.430e+02, 2.401e+03],
       [1.000e+00, 8.000e+00, 6.400e+01, 5.120e+02, 4.096e+03],
       [1.000e+00, 9.000e+00, 8.100e+01, 7.290e+02, 6.561e+03],
       [1.000e+00, 1.000e+01, 1.000e+02, 1.000e+03, 1.000e+04]])
```

In [52]:
```
X_train_sk, X_test_sk, y_train_sk, y_test_sk = train_test_split(X_poly_sk
```

In [53]:
```
regressor_sk = LinearRegression()
regressor_sk.fit(X_train_sk, y_train_sk)
```

Out[53]:
```
▾ LinearRegression
LinearRegression()
```

In [54]:
```
plt.scatter(X_poly_sk[:,1], y, color = 'red')
plt.plot(X_poly_sk[:,1], regressor_sk.predict(X_poly_sk), color = 'blue')
plt.title('Sklearn regression')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```



In [ ]:

Experiment - 6

Support Vector Machine (SVM)

SVM is a powerful supervised learning algorithm used for classification and regression.

1. Objective of SVM

SVM aims to find a **hyperplane** that best separates data into different classes by maximizing the **margin** between the nearest points of different classes.

2. Key Components

- **Hyperplane**: The decision boundary that separates classes.
- **Support Vectors**: Data points closest to the hyperplane that influence its position.
- **Margin**: The distance between the hyperplane and the nearest support vectors.

3. Types of SVM

Linear SVM:

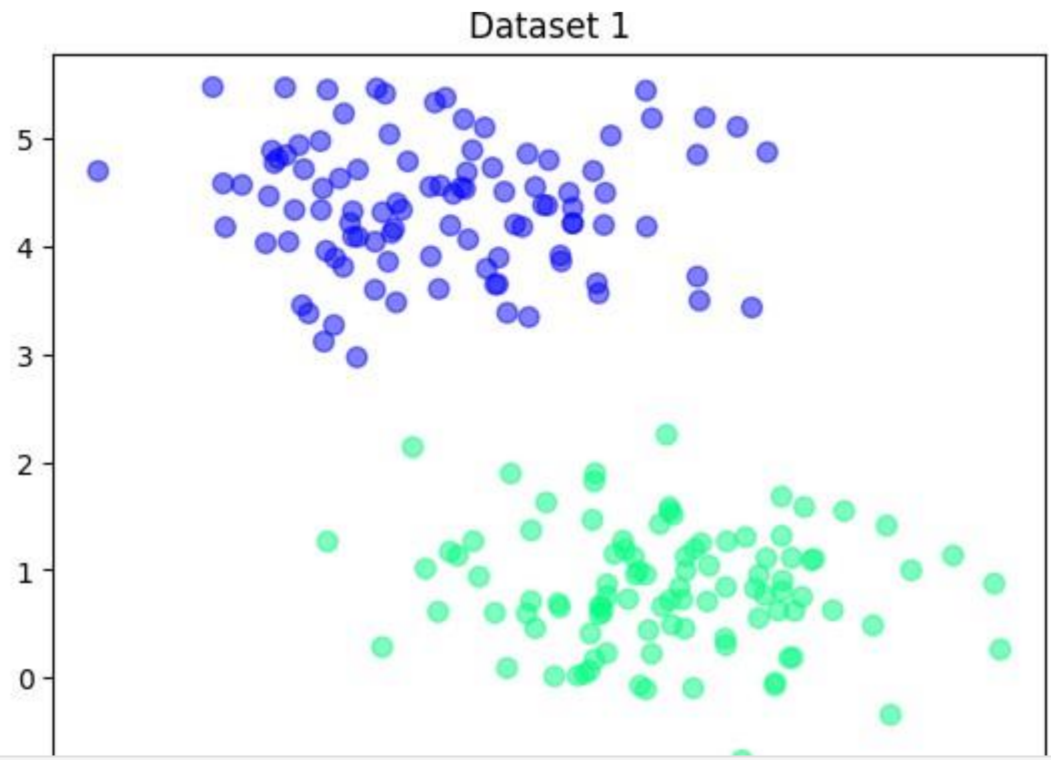
- Used when data is linearly separable.
- The decision function is:

$$f(X) = w^T X + b$$

In [1]:
```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs, make_moons, make_circles
from sklearn.metrics import mean_squared_error
from sklearn.svm import SVC
from sklearn.model_selection  import  train_test_split
from sklearn.metrics import accuracy_score
```

In [2]:
```
X1, y1 = make_blobs(n_samples=200, centers=2,random_state=0, cluster_std=
y1 = np.where(y1 <= 0, -1, 1)
print("First five rows and col values \nX1 : \n",X1[:5], " \n y1 :\n",y1[
plt.scatter(X1[:, 0], X1[:, 1], c=y1, s=50, cmap='winter', alpha=.5)
plt.title("Dataset 1")
plt.show()
```

```
First five rows and col values X1 :
 [[2.51526543  1.11143935]
 [1.8155981   1.11969719]
 [2.69637316  0.62563218]
 [1.67280531  0.65930057]
 [1.89593761  5.18540259]]
 y1 :
 [ 1  1  1  1 -1]
```



In [3]:
```
class SVM_soft_margin:

    def __init__(self, alpha = 0.001, lambda_ = 0.01, n_iterations = 1000
        self.alpha = alpha # learning rate
        self.lambda_ = lambda_ # tradeoff
        self.n_iterations = n_iterations # number of iterations
```

```
        self.w = None # weights or slopes
        self.b = None # intercept


    def fit(self, X, y):

        n_samples, n_features = X.shape
        self.w = np.zeros(n_features) # initalizing with 0
        self.b = 0 # initializewith 0

        for iteration in range(self.n_iterations):
            for i, Xi in enumerate(X):
                # yixiw-b≥1
                if y[i] * (np.dot(Xi, self.w) - self.b) >= 1 :
                    self.w -= self.alpha * (2 * self.lambda_ * self.w) #
                else:
                    self.w -= self.alpha * (2 * self.lambda_ * self.w - n
                    self.b -= self.alpha * y[i] # b = b − α* (yi)
        return self.w, self.b

    def predict(self, X):
        pred = np.dot(X, self.w) - self.b
        result = [1 if val > 0 else -1 for val in pred] # returning in th
        return result
```

In [4]:

```
def get_hyperplane(x, w, b, offset):
        return (-w[0] * x + b + offset) / w[1]
```

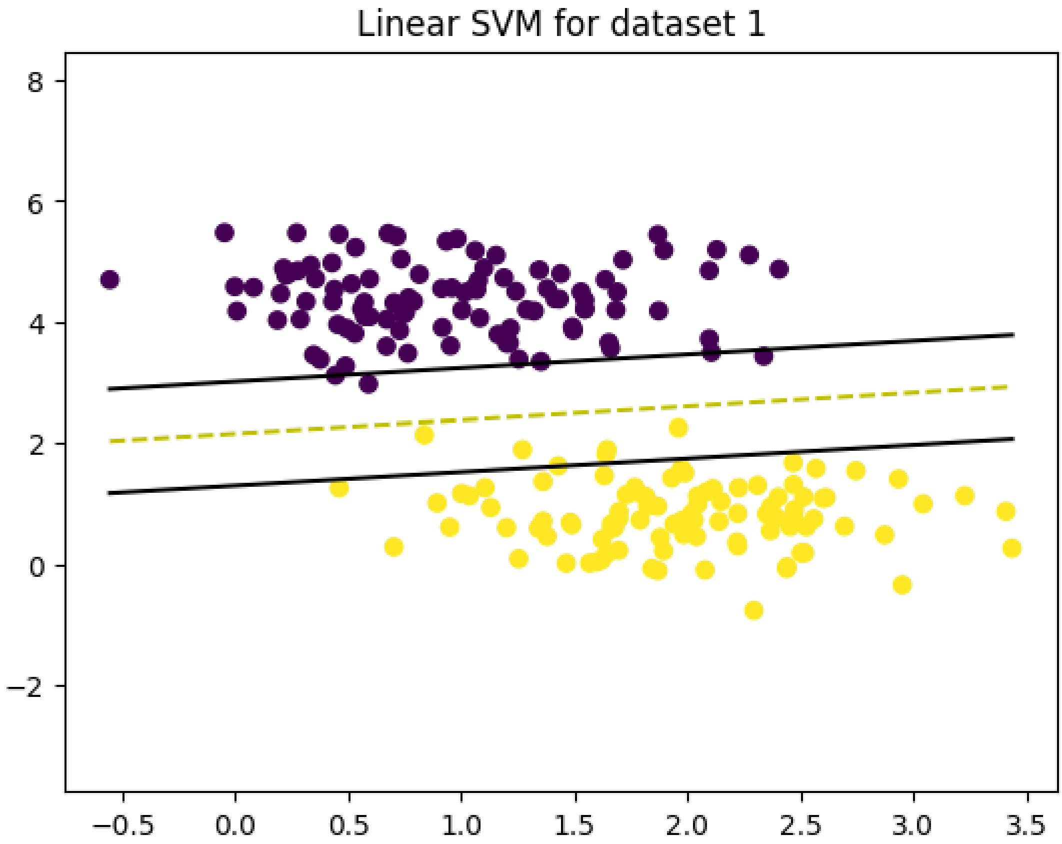
In [5]:

```
def plot_svm(X, y, w, b, title ='Plot for linear SVM'):

    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    plt.scatter(X[:,0], X[:,1], marker='o',c=y)

    x0_1 = np.amin(X[:,0])
    x0_2 = np.amax(X[:,0])

    x1_1 = get_hyperplane(x0_1, w, b, 0)
    x1_2 = get_hyperplane(x0_2, w, b, 0)

    x1_1_m = get_hyperplane(x0_1, w, b, -1)
    x1_2_m = get_hyperplane(x0_2, w, b, -1)

    x1_1_p = get_hyperplane(x0_1, w, b, 1)
    x1_2_p = get_hyperplane(x0_2, w, b, 1)

    ax.plot([x0_1, x0_2],[x1_1, x1_2], 'y--')
    ax.plot([x0_1, x0_2],[x1_1_m, x1_2_m], 'k')
    ax.plot([x0_1, x0_2],[x1_1_p, x1_2_p], 'k')

    x1_min = np.amin(X[:,1])
    x1_max = np.amax(X[:,1])
    ax.set_ylim([x1_min-3,x1_max+3])

    plt.title(title)
    plt.show()
```

In [6]:

```
svm1 = SVM_soft_margin()
w1,b1 = svm1.fit(X1,y1)
print("For dataset 1, score:" ,accuracy_score(svm1.predict(X1),y1))
plot_svm(X1, y1, w1, b1, title= 'Linear SVM for dataset 1')
```

```
For dataset 1, score: 1.0
```



In [ ]: