2348441_lab_09

April 12, 2024

Lab Exercise 9 -– Classification using Kernal Machines (SVM)

Created by : Nileem Kaveramma C C | 2348441 Created DATE:10-03-2024 Edited Date: 10-03-2024

AIM:Develop accurate classification models using Support Vector Machines (SVM) with various kernel functions (linear, polynomial, RBF). Optimize hyperparameters and evaluate model performance using metrics like accuracy, precision, recall, F1-score, and AUC-ROC. Identify the most suitable SVM model for the classification task, ensuring reliable predictions for real-world application

EMPLOYEE SALARY ANALYSIS he provided dataset captures information relevant to employee salary prediction, encompassing various attributes such as age, gender, education level, job title, years of experience, and salary. With a diverse set of features, the dataset offers valuable insights into the characteristics of individuals within an organizational context. This dataset becomes particularly relevant for exploring patterns and relationships that could contribute to predicting employee salaries. Through descriptive statistics, visualizations, and parametric tests, analysts can discern trends, potential disparities, and factors influencing salary variations among employees.

IMPORTED LIBRARIES

- numpy - for numerical, array, matrices (Linear Algebra) processing
- Pandas - for loading and processing datasets
- matplotlib.pyplot - For visualisation
- Saeborn - for statistical graph
- scipy.stats use a variety of statistical functions
- %matplotlib inline: Enables inline plotting in Jupyter notebooks, displaying matplotlib plots directly below the code cell.

[ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
```

[ ]:
```
df = pd.read_csv('/content/Salary Data.csv')
df
```

[ ]:
```
     Age  Gender Education Level                    Job Title  \
0   32.0    Male      Bachelor's            Software Engineer
```

```
1    28.0  Female        Master's                  Data Analyst
2    45.0    Male             PhD                Senior Manager
3    36.0  Female      Bachelor's               Sales Associate
4    52.0    Male        Master's                      Director
..    …     …             …                             …
370  35.0  Female      Bachelor's      Senior Marketing Analyst
371  43.0    Male        Master's        Director of Operations
372  29.0  Female      Bachelor's        Junior Project Manager
373  34.0    Male      Bachelor's  Senior Operations Coordinator
374  44.0  Female             PhD       Senior Business Analyst

     Years of Experience    Salary
0                    5.0   90000.0
1                    3.0   65000.0
2                   15.0  150000.0
3                    7.0   60000.0
4                   20.0  200000.0
..                   …        …
370                  8.0   85000.0
371                 19.0  170000.0
372                  2.0   40000.0
373                  7.0   90000.0
374                 15.0  150000.0

[375 rows x 6 columns]
```

Perform some basic EDA

df.shape - attribute is used to get the dimensions of the DataFrame

[ ]:
```
df.shape
```

[ ]:
```
(375, 6)
```

df.head() method is used to display the first few rows of a DataFrame

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

[ ]:
```
    Age  Gender Education Level          Job Title  Years of Experience  \
0  32.0    Male      Bachelor's  Software Engineer                  5.0
1  28.0  Female        Master's       Data Analyst                  3.0
2  45.0    Male             PhD     Senior Manager                 15.0
3  36.0  Female      Bachelor's    Sales Associate                  7.0
4  52.0    Male        Master's           Director                 20.0

     Salary
0   90000.0
1   65000.0
```

```
2   150000.0
3    60000.0
4   200000.0
```

df.tail() method is used to display the last few rows of a DataFrame.

[ ]:
```
df.tail()
```

[ ]:
```
      Age  Gender Education Level                    Job Title  \
370  35.0  Female      Bachelor's      Senior Marketing Analyst
371  43.0    Male        Master's        Director of Operations
372  29.0  Female      Bachelor's        Junior Project Manager
373  34.0    Male      Bachelor's  Senior Operations Coordinator
374  44.0  Female             PhD       Senior Business Analyst

     Years of Experience    Salary
370                  8.0   85000.0
371                 19.0  170000.0
372                  2.0   40000.0
373                  7.0   90000.0
374                 15.0  150000.0
```

df.columns attribute is used to retrieve the column labels or names of the DataFrame.

[ ]:
```
df.columns
```

[ ]:
```
Index(['Age', 'Gender', 'Education Level', 'Job Title', 'Years of Experience',
       'Salary'],
      dtype='object')
```

df.dtypes attribute is used to retrieve the data types of each column in a DataFrame

[ ]:
```
df.dtypes
```

[ ]:
```
Age                    float64
Gender                  object
Education Level         object
Job Title               object
Years of Experience    float64
Salary                 float64
dtype: object
```

the code df.isnull().count() in Pandas is used to count the total number of rows for each column in a DataFrame, including both missing (null or NaN) and non-missing values.

[ ]:
```
df.isnull().count()
```

```
[ ]: Age                    375
     Gender                 375
     Education Level        375
     Job Title              375
     Years of Experience    375
     Salary                 375
     dtype: int64
```

df.info() method in Pandas provides a concise summary of a DataFrame, including information about the data types, non-null values, and memory usage

```
[ ]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 375 entries, 0 to 374
Data columns (total 6 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Age                  373 non-null    float64
 1   Gender               373 non-null    object
 2   Education Level      373 non-null    object
 3   Job Title            373 non-null    object
 4   Years of Experience  373 non-null    float64
 5   Salary               373 non-null    float64
dtypes: float64(3), object(3)
memory usage: 17.7+ KB
```

The df.describe() method in Pandas is used to generate descriptive statistics that summarize the central tendency, dispersion, and shape of a dataset's distribution

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

```
[ ]:               Age  Years of Experience         Salary
     count  373.000000           373.000000     373.000000
     mean    37.431635            10.030831  100577.345845
     std      7.069073             6.557007   48240.013482
     min     23.000000             0.000000     350.000000
     25%     31.000000             4.000000   55000.000000
     50%     36.000000             9.000000   95000.000000
     75%     44.000000            15.000000  140000.000000
     max     53.000000            25.000000  250000.000000
```

Calculate basic descriptive statistics (mean, median, mode, standard deviation, min, max, quartiles, etc.

```
[ ]: # Mean
mean_salary = df['Salary'].mean()
print("Mean Salary:", mean_salary)
```

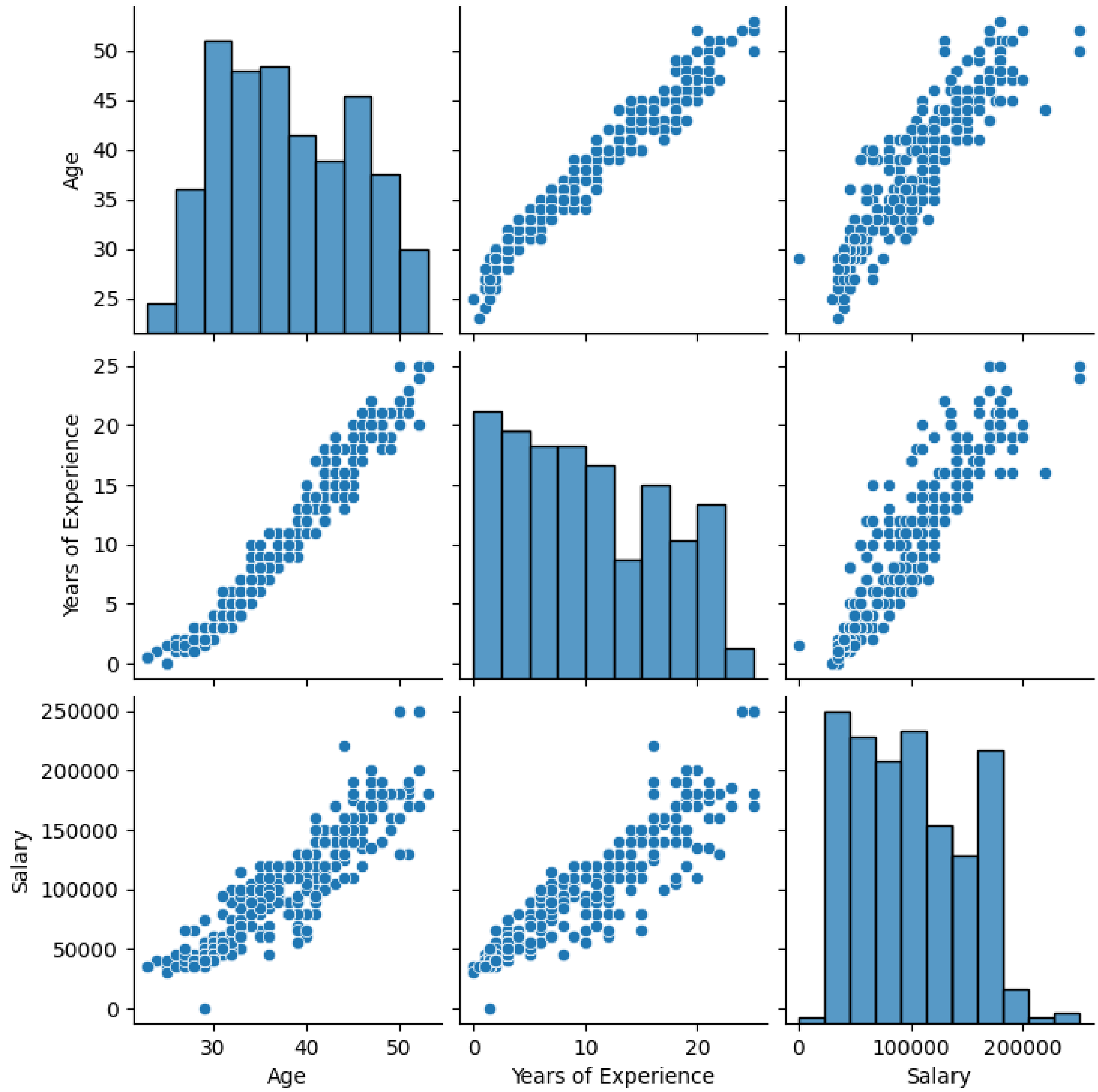
```
# Median
median_salary = df['Salary'].median()
print("Median Salary:", median_salary)

# Mode
mode_salary = df['Salary'].mode()[0]
print("Mode Salary:", mode_salary)

# Standard Deviation
std_salary = df['Salary'].std()
print("Standard Deviation Salary:", std_salary)

# Minimum and Maximum
min_salary = df['Salary'].min()
max_salary = df['Salary'].max()
print("Minimum Salary:", min_salary)
print("Maximum Salary:", max_salary)

# Quartiles
first_quartile = df['Salary'].quantile(0.25)
second_quartile = df['Salary'].quantile(0.5)
third_quartile = df['Salary'].quantile(0.75)

print("First Quartile (25th percentile):", first_quartile)
print("Second Quartile (Median):", second_quartile)
print("Third Quartile (75th percentile):", third_quartile)
```

```
Mean Salary: 100577.34584450402
Median Salary: 95000.0
Mode Salary: 40000.0
Standard Deviation Salary: 48240.013481882655
Minimum Salary: 350.0
Maximum Salary: 250000.0
First Quartile (25th percentile): 55000.0
Second Quartile (Median): 95000.0
Third Quartile (75th percentile): 140000.0
```

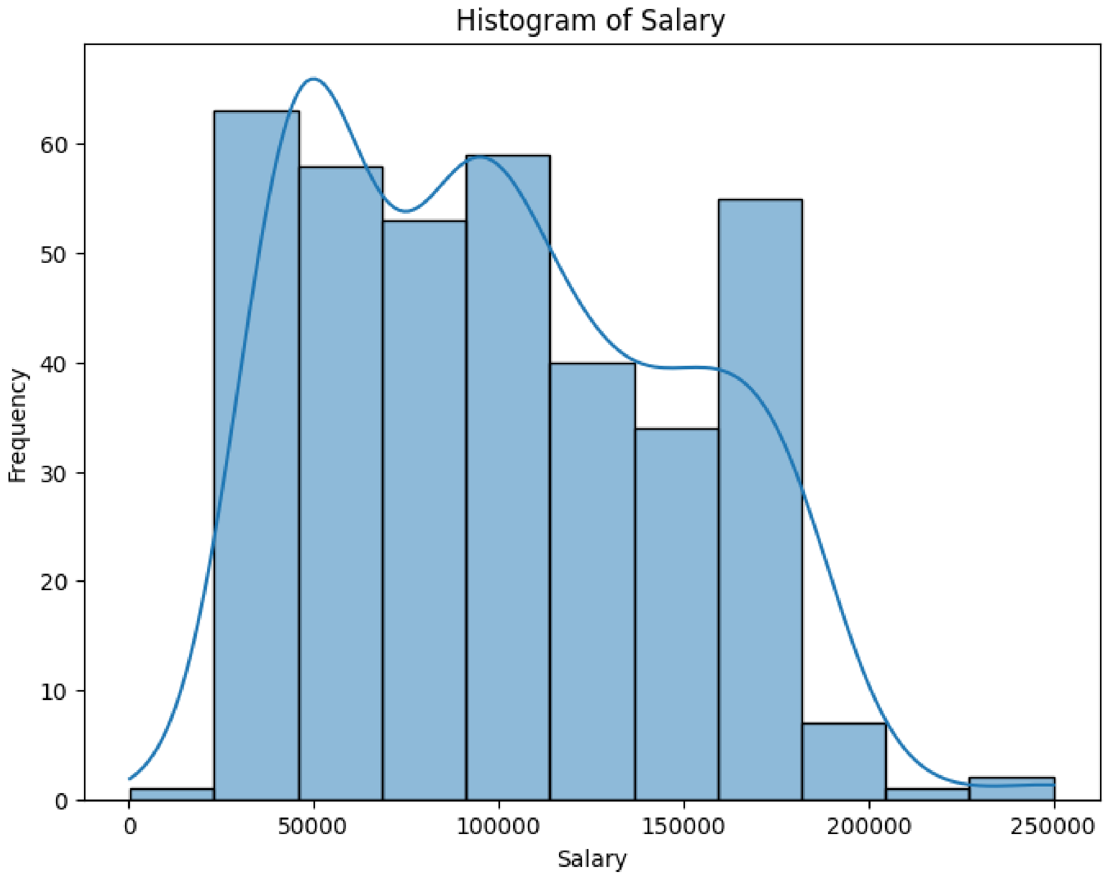
Visualize the distribution using histograms, kernel density plots, or box plots.

[ ]:
```
import seaborn as sns
sns.pairplot(df)
```

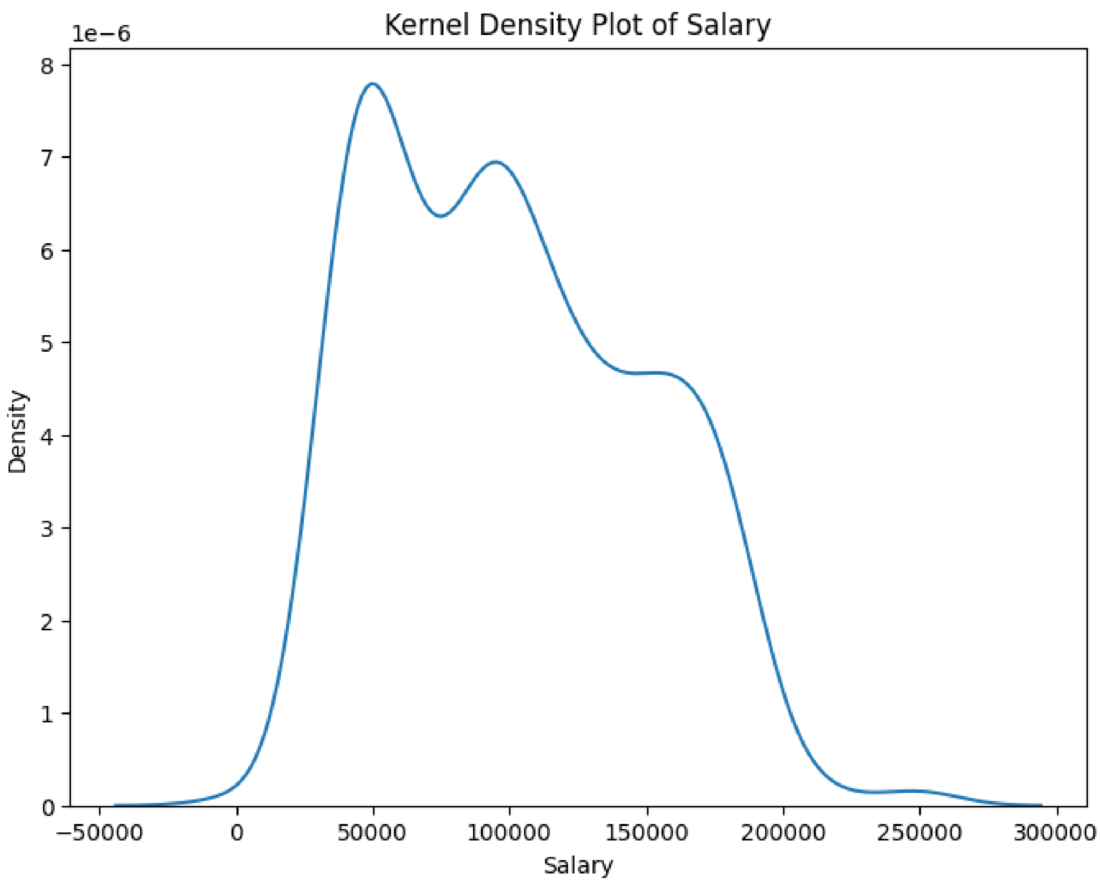
[ ]:
```
<seaborn.axisgrid.PairGrid at 0x7eaa5eb1d180>
```



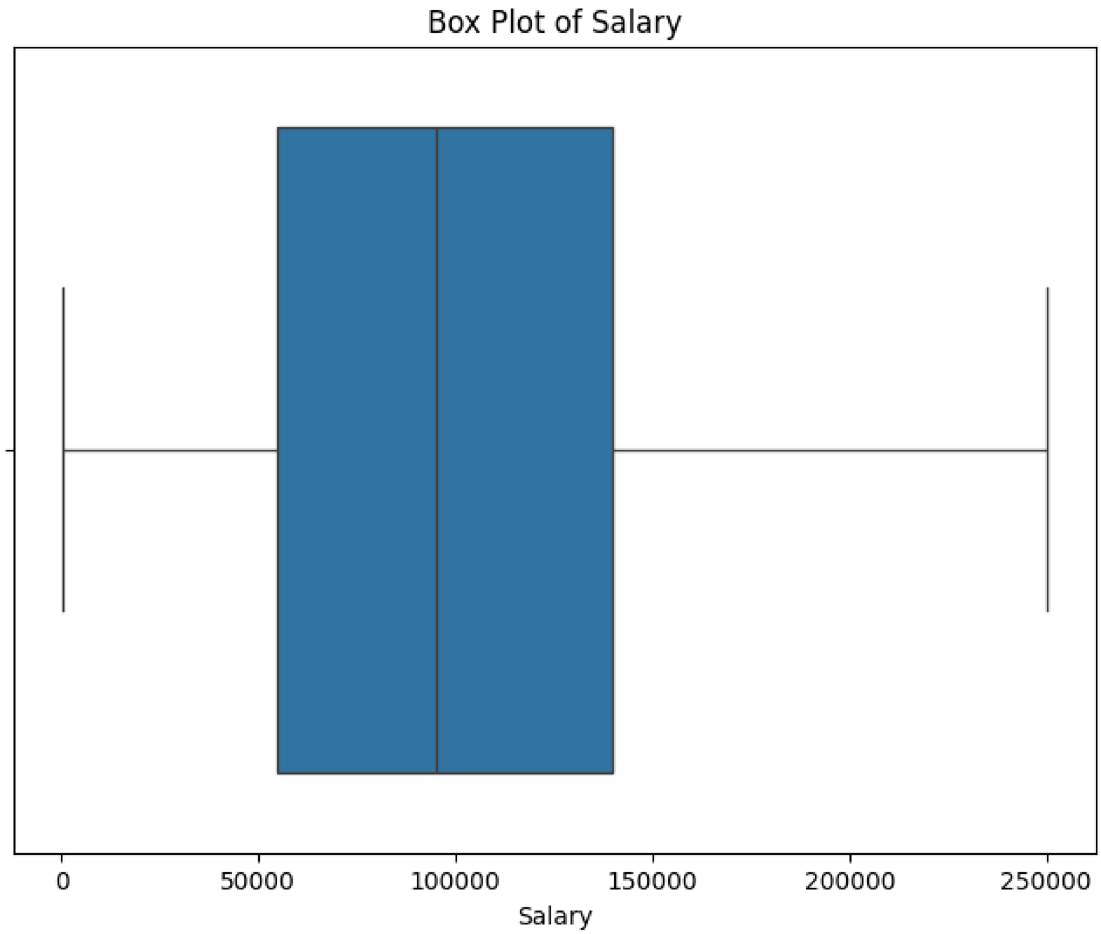
[ ]:
```
# Plot a simple histogram
plt.figure(figsize=(8, 6))
sns.histplot(df['Salary'], kde=True)
plt.title('Histogram of Salary')
plt.xlabel('Salary')
plt.ylabel('Frequency')

# Show the plot
plt.show()
```



[ ]:
```
# Plot a simple kernel density plot
plt.figure(figsize=(8, 6))
sns.kdeplot(df['Salary'])
plt.title('Kernel Density Plot of Salary')
plt.xlabel('Salary')
plt.ylabel('Density')

# Show the plot
plt.show()
```

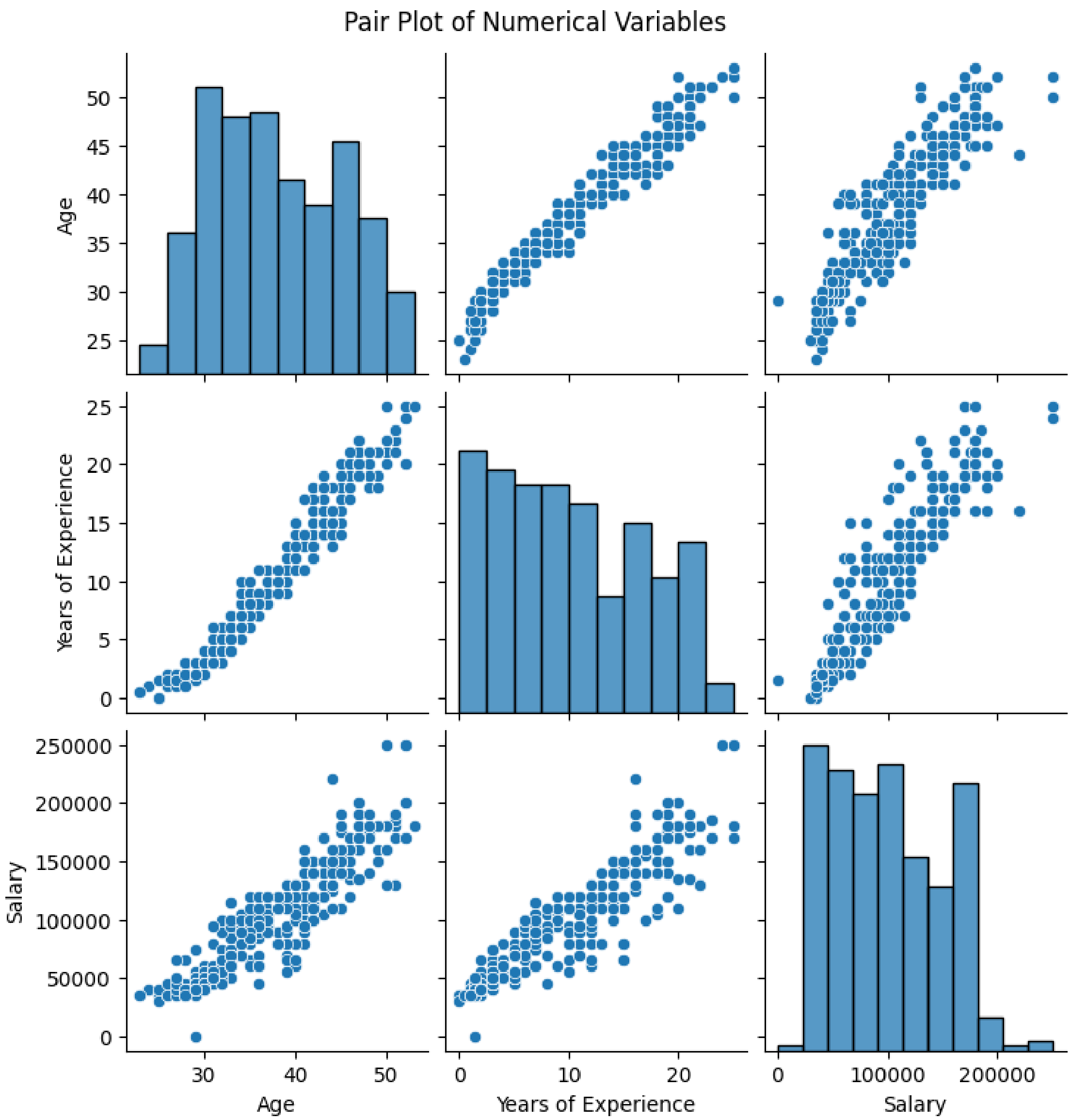


[ ]:
```
# Plot a simple box plot
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['Salary'])
plt.title('Box Plot of Salary')
plt.xlabel('Salary')

# Show the plot
plt.show()
```



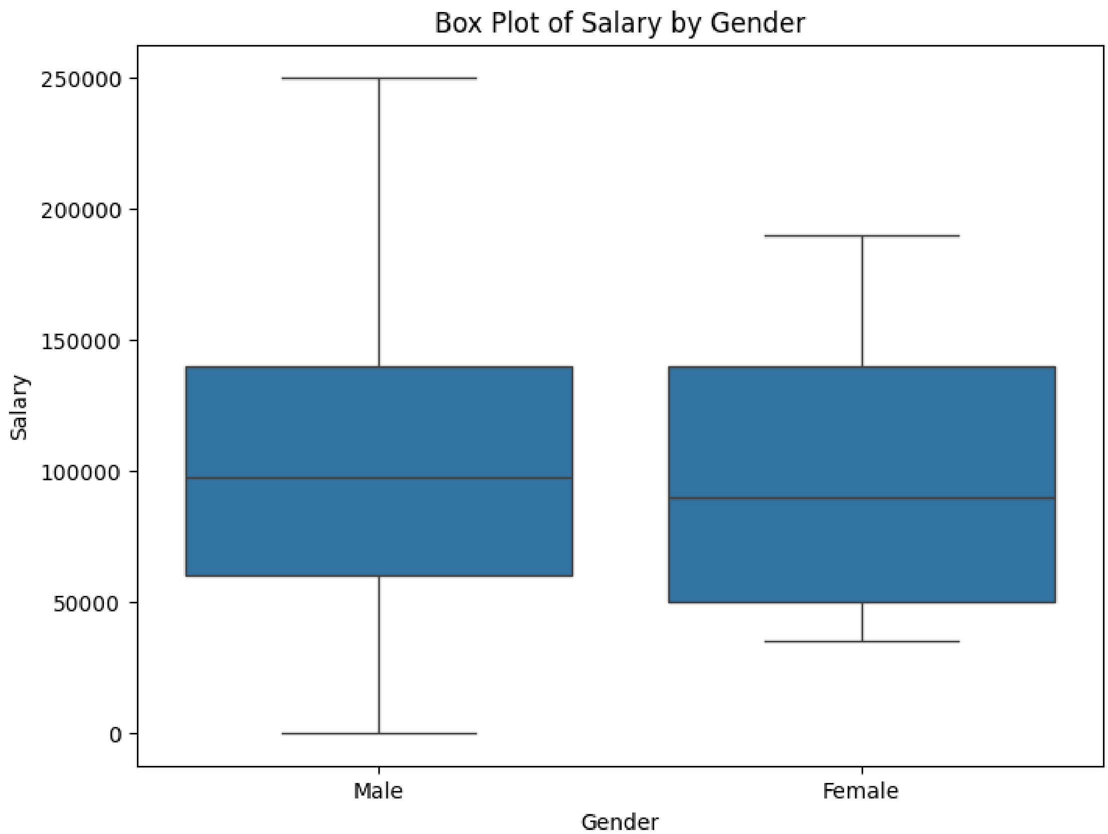
Bivariate Analysis: Explore relationships between pairs of numerical variables using scatter plots, pair plots.

[ ]:
```
# Create a pair plot for numerical variables
sns.pairplot(df)
plt.suptitle('Pair Plot of Numerical Variables', y=1.02)
plt.show()
```



Bivariate Analysis: Explore relationships between numerical and categorical variables using box plots or violin plots.

```
[ ]: # Box plot for 'Salary' vs 'Gender'
plt.figure(figsize=(8, 6))
sns.boxplot(x='Gender', y='Salary', data=df)
plt.title('Box Plot of Salary by Gender')
plt.xlabel('Gender')
plt.ylabel('Salary')

# Show the plot
plt.show()
```



Bivariate Analysis: Calculate correlation coefficients between numerical variables.

[ ]:
```
# Drop non-numeric columns or handle them appropriately
numeric_df = df.select_dtypes(include=['float64', 'int64'])

# Calculate correlation coefficients
correlation_matrix = numeric_df.corr()

# Display the correlation matrix
print("Correlation Coefficients:")
print(correlation_matrix)
```

```
Correlation Coefficients:
                          Age  Years of Experience    Salary
Age                  1.000000             0.979128  0.922335
Years of Experience  0.979128             1.000000  0.930338
Salary               0.922335             0.930338  1.000000
```

Drop the non-required columns/features (dependent columns) if necessary.

```
[ ]: # Drop the 'Salary' column
df = df.drop(columns=['Salary'])

# Display the DataFrame after dropping the column
print(df.head())
```

```
    Age  Gender Education Level          Job Title  Years of Experience
0  32.0    Male      Bachelor's  Software Engineer                  5.0
1  28.0  Female        Master's       Data Analyst                  3.0
2  45.0    Male             PhD     Senior Manager                 15.0
3  36.0  Female      Bachelor's    Sales Associate                  7.0
4  52.0    Male        Master's           Director                 20.0
```

Re-arrange columns/features if required.

```
[ ]: # Define the desired order of columns
desired_columns = ['Age', 'Gender', 'Education Level', 'Job Title', 'Years of␣
 ↪Experience', 'Salary']

# Reindex the DataFrame with the desired column order
df = df.reindex(columns=desired_columns)

# Display the DataFrame after rearranging the columns
print(df.head())
```

```
    Age  Gender Education Level          Job Title  Years of Experience  \
0  32.0    Male      Bachelor's  Software Engineer                  5.0
1  28.0  Female        Master's       Data Analyst                  3.0
2  45.0    Male             PhD     Senior Manager                 15.0
3  36.0  Female      Bachelor's    Sales Associate                  7.0
4  52.0    Male        Master's           Director                 20.0

   Salary
0     NaN
1     NaN
2     NaN
3     NaN
4     NaN
```

Separate the features (X) and the target variable (y).

```
[ ]: # Separate features (X) and target variable (y)
X = df.drop(columns=['Salary'])
y = df['Salary']
# Display the first few rows of X and y
print("Features (X):")
print(X.head())
print("\nTarget variable (y):")
```

```
print(y.head())
```

```
Features (X):
    Age  Gender Education Level          Job Title  Years of Experience
0  32.0    Male      Bachelor's  Software Engineer                  5.0
1  28.0  Female        Master's       Data Analyst                  3.0
2  45.0    Male             PhD     Senior Manager                 15.0
3  36.0  Female      Bachelor's    Sales Associate                  7.0
4  52.0    Male        Master's           Director                 20.0

Target variable (y):
0   NaN
1   NaN
2   NaN
3   NaN
4   NaN
Name: Salary, dtype: float64
```

Perform Standardization or normalization on the features as required.

[ ]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Exclude non-numeric columns
numeric_columns = X.select_dtypes(include=['float64', 'int64']).columns
X_numeric = X[numeric_columns]

# Perform Standardization
scaler = StandardScaler()
X_standardized = scaler.fit_transform(X_numeric)
X_standardized = pd.DataFrame(X_standardized, columns=X_numeric.columns)

# Perform Normalization
scaler = MinMaxScaler()
X_normalized = scaler.fit_transform(X_numeric)
X_normalized = pd.DataFrame(X_normalized, columns=X_numeric.columns)

# Display the first few rows of standardized and normalized features
print("Standardized Features:")
print(X_standardized.head())
print("\nNormalized Features:")
print(X_normalized.head())
```

```
Standardized Features:
        Age  Years of Experience
0 -0.769398            -0.768276
1 -1.336003            -1.073702
2  1.072068             0.758859
3 -0.202793            -0.462849
```

```
4  2.063627             1.522426

Normalized Features:
        Age  Years of Experience
0  0.300000                 0.20
1  0.166667                 0.12
2  0.733333                 0.60
3  0.433333                 0.28
4  0.966667                 0.80
```

11. Implement Support Vector Machines (SVM):

a. Train the SVM model using the training data.

[ ]:
```python
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error


# Convert categorical variables to numerical using LabelEncoder
label_encoders = {}
categorical_cols = ["Gender", "Education Level", "Job Title"]

for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Separate features and target variable
X = df.drop(columns=["Salary"])
y = df["Salary"]

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

# Train the SVM model
svm_model = SVR(kernel='linear')
svm_model.fit(X_train, y_train)

# Predictions on the test set
y_pred = svm_model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
```

```python
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 1906814450.25
```

Explore different kernel functions (e.g., linear, polynomial, radial basis function) and tune hyper-parameters.

[ ]:
```python
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVR
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error



# Convert categorical variables to numerical using LabelEncoder
label_encoders = {}
categorical_cols = ["Gender", "Education Level", "Job Title"]

for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Separate features and target variable
X = df.drop(columns=["Salary"])
y = df["Salary"]

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

# Define parameter grid
param_grid = [
    {'kernel': ['linear'], 'C': [0.1, 1, 10, 100]},
    {'kernel': ['poly'], 'degree': [2, 3, 4], 'C': [0.1, 1, 10, 100]},
    {'kernel': ['rbf'], 'gamma': [0.1, 1, 10, 100], 'C': [0.1, 1, 10, 100]}
]

# Instantiate SVR model
svm_model = SVR()

# Perform GridSearchCV
grid_search = GridSearchCV(svm_model, param_grid, cv=5,␣
↪scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
```

```
# Get best parameters and best estimator
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

print("Best Parameters:", best_params)

# Predictions on the test set using the best estimator
y_pred = best_estimator.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Best Parameters: {'C': 100, 'kernel': 'linear'}
Mean Squared Error: 107372500.0
```

Evaluate the performance of the trained model using appropriate metrics.

[ ]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)  # RMSE
r2 = r2_score(y_test, y_pred)

print("Mean Absolute Error:", mae)
print("Root Mean Squared Error:", rmse)
print("R-squared Score:", r2)
```

```
Mean Absolute Error: 10350.0
Root Mean Squared Error: 10362.070256469024
R-squared Score: 0.312816
```

Display the classification report and confusion matrix.

[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.svm import SVR
import pandas as pd

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

# Train the regressor (SVR used as an example)
regressor = SVR(kernel='linear')
regressor.fit(X_train, y_train)
```

```python
# Predictions on the test set
y_pred = regressor.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)  # RMSE
r2 = r2_score(y_test, y_pred)

# Display the regression metrics
print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared (R2) Score:", r2)
```

```
Mean Squared Error (MSE): 1906814450.25
Mean Absolute Error (MAE): 41902.5
Root Mean Squared Error (RMSE): 43667.08657845174
R-squared (R2) Score: -11.2036124816
```

Interpret the results and discuss the effectiveness of the SVM model.

Mean Squared Error (MSE):

In the first set of results, the MSE is quite high, indicating a large average squared difference between the predicted and actual values. However, in the second set of results, the MSE is significantly lower, which suggests that the model performs better in terms of predicting the target variable. The MSE values are 1906814450.25 and 107372500.0, respectively. Best Parameters:

The best parameters obtained for the SVM model are {'C': 100, 'kernel': 'linear'}. This indicates that a linear kernel with a regularization parameter (C) of 100 was found to be the best for this dataset. Mean Absolute Error (MAE):

The MAE represents the average absolute difference between the predicted and actual values. A lower MAE indicates better performance. The MAE values are 10350.0 and 41902.5, respectively. Root Mean Squared Error (RMSE):

RMSE is the square root of MSE and provides a measure of the spread of errors. It is in the same unit as the target variable. The RMSE values are 10362.070256469024 and 43667.08657845174, respectively. R-squared (R2) Score:

R-squared measures the proportion of the variance in the target variable that is predictable from the independent variables. A higher R-squared value indicates a better fit of the model. The R-squared values are 0.312816 and -11.2036124816, respectively.

An R-squared score of 0.31 suggests that the model explains around 31% of the variance in the target variable, which is relatively low. The negative R-squared score in the second set of results indicates that the model performs worse than a simple horizontal line.

Compare the performance of SVM models with different kernel functions.

[ ]:
```python
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import pandas as pd

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

# Define a list of kernel functions to compare
kernels = ['linear', 'poly', 'rbf']

# Dictionary to store the evaluation results
results = {}

# Train SVM models with different kernel functions
for kernel in kernels:
    # Train the SVM model
    regressor = SVR(kernel=kernel)
    regressor.fit(X_train, y_train)

    # Predictions on the test set
    y_pred = regressor.predict(X_test)

    # Evaluate the model
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = mean_squared_error(y_test, y_pred, squared=False)  # RMSE
    r2 = r2_score(y_test, y_pred)

    # Store the evaluation results
    results[kernel] = {'MSE': mse, 'MAE': mae, 'RMSE': rmse, 'R2': r2}

# Display the results
print("Performance of SVM models with different kernel functions:")
for kernel, result in results.items():
    print("\nKernel:", kernel)
    print("Mean Squared Error (MSE):", result['MSE'])
    print("Mean Absolute Error (MAE):", result['MAE'])
    print("Root Mean Squared Error (RMSE):", result['RMSE'])
    print("R-squared (R2) Score:", result['R2'])
```

```
Performance of SVM models with different kernel functions:

Kernel: linear
Mean Squared Error (MSE): 1906814450.25
Mean Absolute Error (MAE): 41902.5
```

```
Root Mean Squared Error (RMSE): 43667.08657845174
R-squared (R2) Score: -11.2036124816

Kernel: poly
Mean Squared Error (MSE): 1961731006.7829707
Mean Absolute Error (MAE): 42491.58088356273
Root Mean Squared Error (RMSE): 44291.43265669977
R-squared (R2) Score: -11.555078443411013

Kernel: rbf
Mean Squared Error (MSE): 1962424933.048599
Mean Absolute Error (MAE): 42499.18339825719
Root Mean Squared Error (RMSE): 44299.265603942
R-squared (R2) Score: -11.559519571511034
```

[ ]: