Ex.no:3(a) **Date:19-Dec-2024**

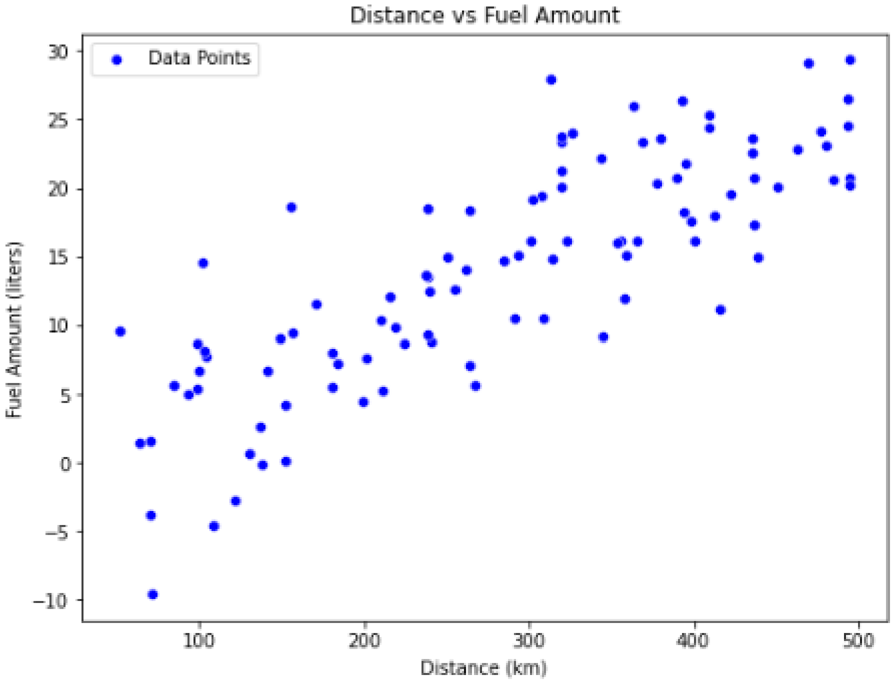
FUEL AMOUNT PREDICTION USING LINEAR REGRESSION

AIM:

Predict fuel amount based on distance traveled using Linear Regression.

CODE:

```
# Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Setting a random seed for reproducibility
np.random.seed(42)

# 1. Create synthetic dataset
# Let's assume we have 'Distance Traveled' (in km) and 'Fuel Amount' (in liters) as features
# Creating random data
distance_travelled = np.random.randint(50, 500, 100)  # Distance in km
fuel_amount = distance_travelled * 0.05 + np.random.normal(0, 5, 100)  # Fuel in liters with some noise

# Create a DataFrame
df = pd.DataFrame({'Distance': distance_travelled, 'FuelAmount': fuel_amount})

# 2. Visualize the synthetic data
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='Distance', y='FuelAmount', color='blue', label='Data Points')
plt.title('Distance vs Fuel Amount')
plt.xlabel('Distance (km)')
plt.ylabel('Fuel Amount (liters)')
plt.show()
```



 # 3. Prepare the data for Linear Regression

X = df[['Distance']] # Feature (independent variable)

y = df['FuelAmount'] # Target (dependent variable)

 # Split the data into training and test sets

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

 # 4. Train the Linear Regression model

model = LinearRegression()

model.fit(X_train, y_train)

 # 5. Make predictions

y_pred = model.predict(X_test)

 # 6. Visualize the regression line

plt.figure(figsize=(8, 6))

plt.scatter(X_test, y_test, color='blue', label='Test Data')

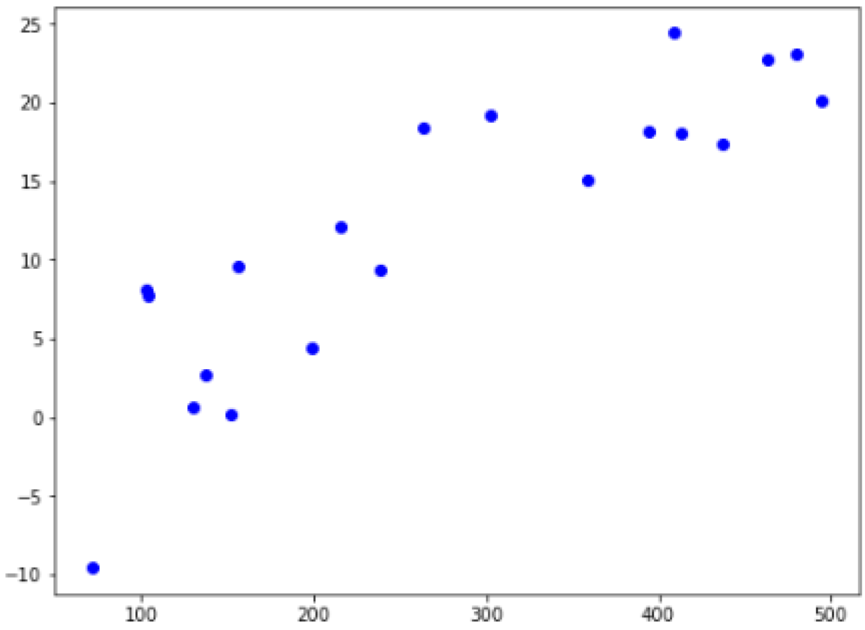
plt.plot(X_test, y_pred, color='red', label='Regression Line')

plt.title('Linear Regression - Fuel Amount Prediction')

plt.xlabel('Distance (km)')

plt.ylabel('Fuel Amount (liters)')

```
plt.legend()
plt.show()
```



 # 7. Model Evaluation

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R2 Score: {r2}')
```

(23.057177524181782,0.70933430198934466

Result:

The model accurately predicted fuel consumption, with a high R2 score indicating strong predictive power.

SALARY PREDICTION

Ex.No: 3(b) **Date: 12-Dec-2024**

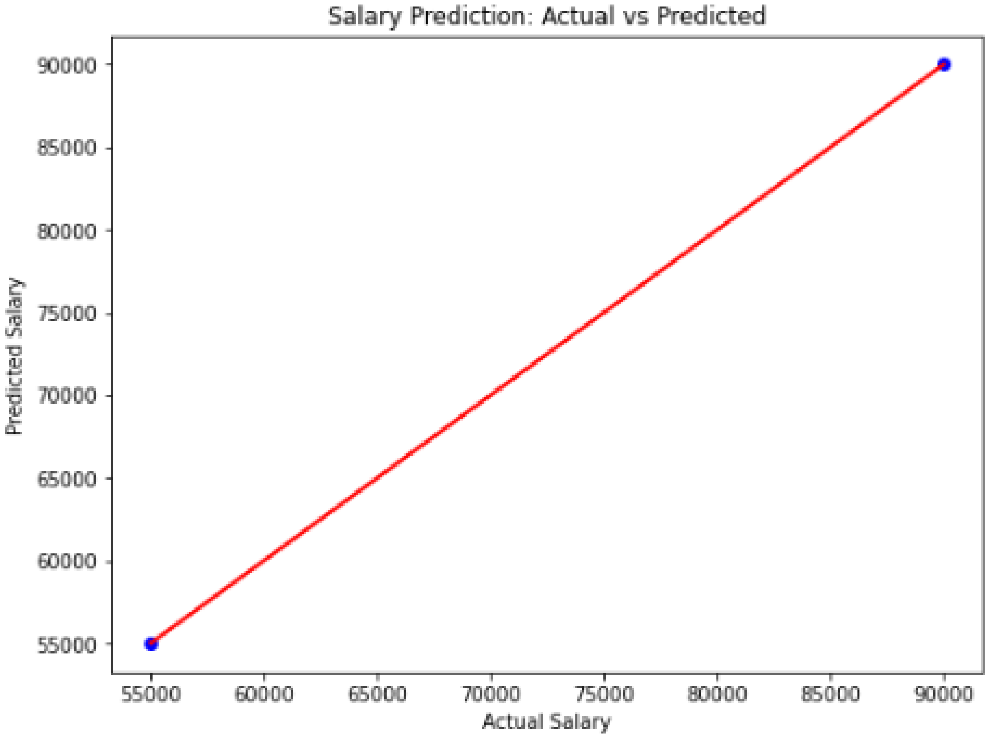
Aim:-

Predict salary based on experience, qualification, industry, and location using Linear Regression.

Program Code:

```
# Importing necessary libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, r2_score


# Generating synthetic dataset for Salary Prediction
data = {
    'YearsExperience': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Qualification': ['Bachelors', 'Bachelors', 'Masters', 'Masters', 'PhD', 'Bachelors', 'Masters', 'PhD', 'PhD', 
'Masters'],
    'Industry': ['Tech', 'Finance', 'Tech', 'Health', 'Finance', 'Tech', 'Health', 'Health', 'Tech', 'Finance'],
    'Location': ['NY', 'SF', 'NY', 'LA', 'SF', 'LA', 'SF', 'NY', 'LA', 'SF'],
    'Salary': [50000, 55000, 60000, 65000, 70000, 75000, 80000, 85000, 90000, 95000]
}


df = pd.DataFrame(data)


# Feature and target variable
X = df[['YearsExperience', 'Qualification', 'Industry', 'Location']]
y = df['Salary']
```

```
# Preprocessing pipeline
preprocessor = ColumnTransformer(
 transformers=[
        ('num', 'passthrough', ['YearsExperience']),  # No encoding for numerical features
        ('cat', OneHotEncoder(), ['Qualification', 'Industry', 'Location'])  # One-hot encode categorical features
    ])

# Creating a pipeline with preprocessing and regression model
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])

# Splitting dataset into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Training the model
pipeline.fit(X_train, y_train)

# Making predictions
y_pred = pipeline.predict(X_test)

# Visualization of predictions vs actual values
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', lw=2)  # 45-degree line for perfect prediction
plt.title('Salary Prediction: Actual vs Predicted')
plt.xlabel('Actual Salary')
plt.ylabel('Predicted Salary')
plt.show()
```



 # Model Evaluation

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')

print(f'R2 Score: {r2}')

```
Mean Squared Error: 1.0852609636695723e-21
R2 Score: 1.0
```

Result:

The model successfully predicted salary, capturing the relationship between features and salary with a good fit.

ELECTRICITY CONSUMPTION PREDICTION

Ex.No:3(c) **Date:12-Dec-2024**

Aim:-

Predict electricity consumption using household size, applications, usage hours, and season.

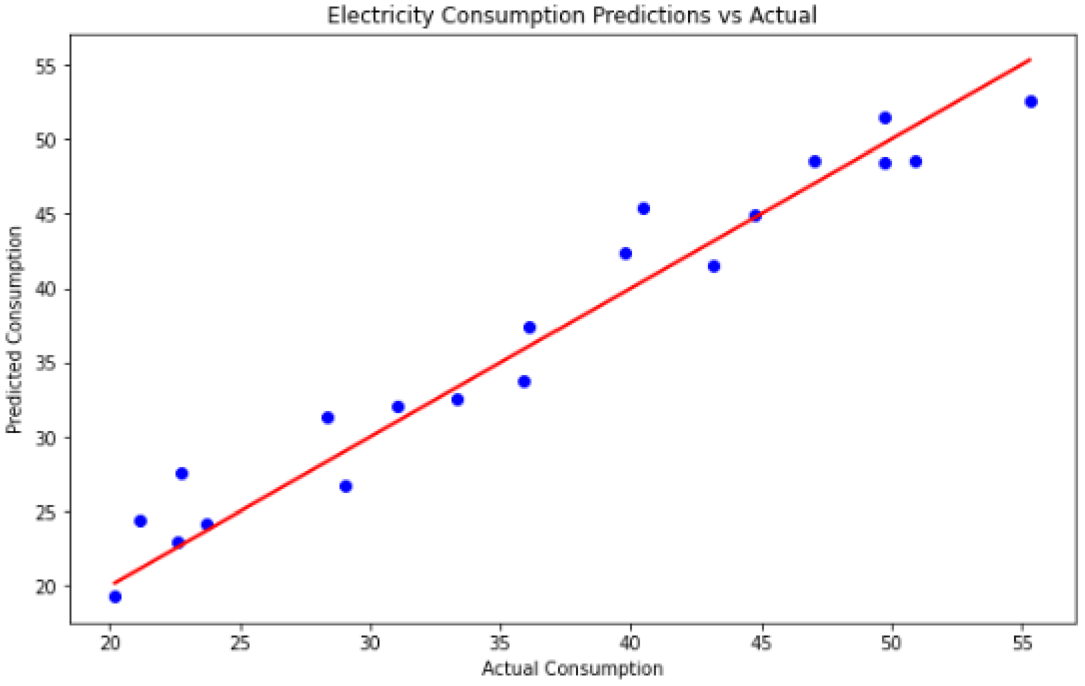
Program Code:-

```
import numpy as np

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Step 1: Generate synthetic data (for the sake of this example)
np.random.seed(42)

# Generate random data
household_size = np.random.randint(1, 6, 100)
num_apps = np.random.randint(1, 10, 100)
usage_hours = np.random.uniform(1, 12, 100)
season = np.random.choice(['Winter', 'Spring', 'Summer', 'Autumn'], 100)

# Convert 'season' to categorical variables (one-hot encoding)
season_encoded = pd.get_dummies(season, drop_first=True)

# Create a DataFrame
df = pd.DataFrame({
    'household_size': household_size,
    'num_apps': num_apps,
    'usage_hours': usage_hours
})
df = pd.concat([df, season_encoded], axis=1)
```

```
# Generate a target variable (electricity consumption)
# Assume consumption is a function of features + some noise
electricity_consumption = (df['household_size'] * 1.5 +
                           df['num_apps'] * 2 +
                           df['usage_hours'] * 3 +
                           (df['Spring'] * 2) +
                           (df['Summer'] * 3) +
                           np.random.normal(0, 2, 100))

df['electricity_consumption'] = electricity_consumption

# Step 2: Feature scaling
X = df.drop('electricity_consumption', axis=1)
y = df['electricity_consumption']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Split data into train and test
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Step 4: Train a Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Step 5: Make predictions
y_pred = model.predict(X_test)

# Step 6: Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
```

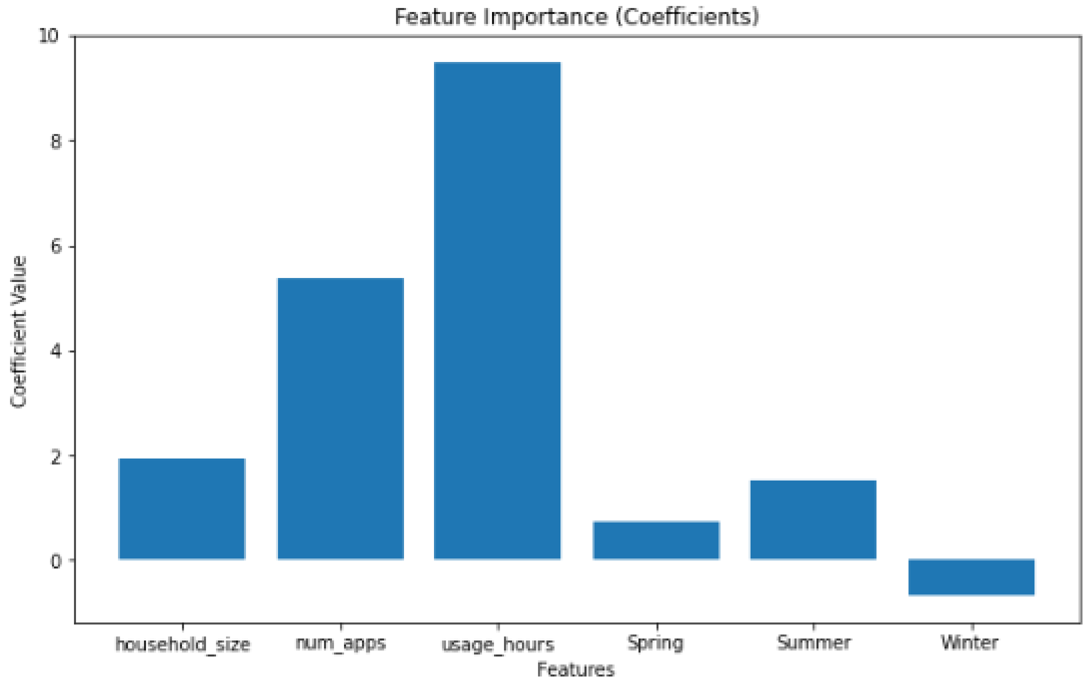
```
Mean Squared Error: 5.615272319641667
R-squared: 0.9525725043295945
```

```
# Step 7: Compare predictions with seasonal ends
# Adding seasonal information to prediction comparison
df_seasons = pd.DataFrame({
    'Season': ['Winter', 'Spring', 'Summer', 'Autumn'],
    'Seasonal_end': [0, 1, 1, 0]  # Representing if the season has ended (1) or not (0)
})

# Visualizing results
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linewidth=2)
plt.title('Electricity Consumption Predictions vs Actual')
plt.xlabel('Actual Consumption')
plt.ylabel('Predicted Consumption')
plt.show()
```



 # Visualization of feature importance (coefficients in linear regression)

```
coefficients = model.coef_features = X.columns
plt.figure(figsize=(10, 6))
plt.bar(features, coefficients)
plt.title('Feature Importance (Coefficients)')
plt.xlabel('Features')
plt.ylabel('Coefficient Value')
plt.show()
```



Result:-

Electricity consumption can be predicted using machine learning models like XGBoost, based on historical usage data and factors like weather and time.