Assignment # 7

[Find errors and complete the steps]

[Note]

Use the PyCharm software if google colab is not working properly

1. House Price Prediction System

📌 Concepts Used:

- Linear Regression Model
- Cost Function (MSE)
- Ordinary Least Squares (OLS)
- R-squared & Adjusted R-squared

📌 Project Overview:

- Build a model that predicts house prices based on features like square footage, number of bedrooms, location, and age of the house.

```
import pandas as pd
import numpy as np
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.metrics import mean_squared_error, r2_score

# Load dataset
df = pd.read_csv('house_data.csv')
```

```
# Check for non-numeric columns (excluding the target variable 'price')
non_numeric_columns = df.select_dtypes(include=['object']).columns.tolist()
print(f"Non-numeric columns: {non_numeric_columns}")

# Convert categorical columns to numerical using one-hot encoding
if non_numeric_columns:
    df = pd.get_dummies(df, columns=non_numeric_columns, drop_first=True)

# Ensure 'price' column exists
if 'price' not in df.columns:
    raise KeyError("The dataset does not contain a 'price' column. Please
check the CSV file.")

# Define features (X) and target (y)
X = df.drop(columns=['price'])  # Features (all except target)
y = df['price']  # Target variable

# Ensure all feature columns are numeric
if not np.issubdtype(X.dtypes.values[0], np.number):
    raise ValueError("Some features are still non-numeric. Check the dataset
preprocessing.")

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

# Train model
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluate model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Model Evaluation:\nMSE: {mse:.2f}, R-squared: {r2:.2f}')
```

[Your Task]

1. Install the dependencies
2. Collect house pricing data (from Kaggle or real estate websites or use already given).
3. Preprocess data (handle missing values, categorical encoding).
4. Train a linear regression model using OLS or Gradient Descent.
5. Evaluate using R-squared and Adjusted R-squared.
6. Run the above code
7. Show the output

2. Salary Prediction System

📌 Concepts Used:

- Dependent & Independent Variables
- Regression Line & Coefficients
- Gradient Descent Optimization
- Bias-Variance Tradeoff

📌 Project Overview:

- Develop a system that predicts the salary of an employee based on their experience, education, job role, and location.

```
import pandas as pd
import numpy as np
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.metrics import mean_squared_error, r2_score

# Load dataset (Replace 'salary_data.csv' with an actual dataset)
df = pd.read_csv('salary_data.csv')

# Preprocess data
df.dropna(inplace=True)

# Identify categorical columns
```

```
categorical_columns = ['degree', 'job_role', 'location']
existing_categorical_columns = [col for col in categorical_columns if col in
df.columns]

# Apply one-hot encoding only if columns exist
if existing_categorical_columns:
    df = pd.get_dummies(df, columns=existing_categorical_columns,
drop_first=True)

# Define features and target variable
if 'Salary' in df.columns:
    X = df.drop(columns=['Salary'])
    y = df['Salary']

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

    # Train model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Predictions
    y_pred = model.predict(X_test)

    # Evaluate model
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f'MSE: {mse}, R-squared: {r2}')

    # Example prediction
    sample_input = X_test.iloc[[0]]  # Keep feature names
    predicted_salary = model.predict(sample_input)
    print(f'Predicted Salary for sample input: {predicted_salary[0]}')

else:
    print("Error: The 'Salary' column is missing from the dataset.")
```

[Your Task]

```
1. Install the dependencies
2. Collect salary dataset from sources like Glassdoor or use given
   data.
3. Define dependent (salary) and independent variables (experience,
   degree, etc.).
4. Apply linear regression and tune the model.
5. Visualize the regression line.
6. Create a simple web app for input and predictions.
7. Compile and run the code
8. Print and show the output
```

Stock Price Trend Prediction

📌 Concepts Used:

- Equation of a Straight Line
- Ordinary Least Squares (OLS)
- R-squared Evaluation

📌 Project Overview:

- Predict the future closing price of a stock based on past stock prices, volume, and market trends.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import yfinance as yf
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import streamlit as st

# Fetch stock data
def get_stock_data(ticker):
```

```
    stock = yf.Ticker(ticker)
    df = stock.history(period='5y')
    df = df[['Close', 'Volume']].dropna()
    df['Day'] = np.arange(len(df))
    return df

# Load dataset
ticker = 'AAPL'  # Example stock symbol
df = get_stock_data(ticker)

# Define features and target variable
X = df[['Day', 'Volume']]
y = df['Close']

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

# Train model
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluate model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'MSE: {mse}, R-squared: {r2}')

# Streamlit app for visualization
st.title('Stock Price Trend Prediction')
st.write(f'Stock: {ticker}')

fig, ax = plt.subplots()
ax.plot(df['Day'], df['Close'], label='Actual Prices', color='blue')
ax.scatter(X_test['Day'], y_pred, label='Predicted Prices', color='red')
ax.set_xlabel('Day')
ax.set_ylabel('Stock Price')
ax.legend()
st.pyplot(fig)
```

```
# Prediction function
def predict_stock_price(day, volume):
    input_data = np.array([day, volume]).reshape(1, -1)
    return model.predict(input_data)[0]

# User input for prediction
st.sidebar.header('Predict Future Stock Price')
day = st.sidebar.number_input('Enter Future Day:',
min_value=int(df['Day'].min()), max_value=int(df['Day'].max())+30)
volume = st.sidebar.number_input('Enter Expected Volume:',
min_value=int(df['Volume'].min()), max_value=int(df['Volume'].max()))

if st.sidebar.button('Predict'):
    prediction = predict_stock_price(day, volume)
    st.sidebar.write(f'Predicted Stock Price: ${prediction:.2f}')
```

[Your Task]

```
1. Install the dependencies
2. Fetch stock market data from Yahoo Finance API.
3. Perform feature engineering on historical stock prices.
4. Apply a linear regression model for short-term predictions.
5. Evaluate performance using R-squared and Adjusted R-squared.
6. Display trend predictions on a web dashboard (Streamlit).
7. Compile and run the code
8. Print and show the output
```

Customer Churn Prediction for a Subscription Service

📌 Concepts Used:

- Cost Function (MSE)
- Bias-Variance Tradeoff
- Regression Coefficients

📌 Project Overview:

- Predict whether a customer will continue their subscription or churn based on factors like engagement, billing issues, and customer support interactions.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import streamlit as st

# Load dataset
df = pd.read_csv('customer_churn_data.csv')  # Replace with actual dataset

# Preprocess data
df.dropna(inplace=True)  # Remove missing values

# Convert 'churn' column to integers
df['churn'] = df['churn'].astype(str).str.strip().map({'False': 0, 'True':
1})

# Convert categorical columns to numerical
df = pd.get_dummies(df, columns=['international_plan', 'voice_mail_plan'],
drop_first=True)

# Drop unnecessary columns
df.drop(columns=['Id', 'state', 'phone_number'], inplace=True)

# Define features and target variable
X = df.drop(columns=['churn'])
y = df['churn']

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

# Train model
```

```
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}\n')
print(report)

# Streamlit app for visualization
st.title('Customer Churn Prediction')
st.write(f'Accuracy: {accuracy:.2f}')

# Confusion Matrix Visualization
fig, ax = plt.subplots()
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d',
cmap='Blues', xticklabels=['No Churn', 'Churn'], yticklabels=['No Churn',
'Churn'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
st.pyplot(fig)

# User input for prediction
st.sidebar.header('Predict Customer Churn')
features = {col: st.sidebar.number_input(f'Enter {col}:',
float(X[col].min()), float(X[col].max())) for col in X.columns}

if st.sidebar.button('Predict'):
    input_data = np.array([features[col] for col in X.columns]).reshape(1, -
1)
    prediction = model.predict(input_data)[0]
    st.sidebar.write(f'Predicted Churn: {"Yes" if prediction == 1 else
"No"}')
```

[Your Task]

```
1. Install the dependencies
2. Collect customer interaction and subscription data.
3. Define independent (features like usage, complaints) and dependent
   variables (churn or not).
4. Apply linear regression and analyze coefficients.
5. Optimize model using Gradient Descent.
6. Create an alert system for potential churners.
7. Compile and run the code
8. Print and show the output
```

Energy Consumption Prediction

📌 Concepts Used:

- Mean Squared Error (MSE)
- Gradient Descent Optimization
- Bias-Variance Tradeoff

📌 Project Overview:

- Predict energy consumption of households based on temperature, appliance usage, and seasonal variations.

```python
import pandas as pd
import numpy as np
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.metrics import mean_squared_error, r2_score
import streamlit as st

# Load dataset
file_path = "owid-energy-data.csv"
df = pd.read_csv(file_path)
```

```
# Display column names to check the correct column name for energy
consumption
st.write("Dataset Columns:", df.columns.tolist())

# Identify the correct column for energy consumption
energy_columns = [col for col in df.columns if "consumption" in col.lower()]
if not energy_columns:
    raise KeyError("No column related to energy consumption found in the
dataset.")

# Use the first identified energy consumption column
energy_column = energy_columns[0]
st.write(f"Using '{energy_column}' as the target variable.")

# Preprocess data
df.dropna(inplace=True)

# One-hot encode categorical variables
categorical_cols = df.select_dtypes(include=['object']).columns.tolist()
if categorical_cols:
    df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)

# Define features and target variable
X = df.drop(columns=[energy_column])
y = df[energy_column]

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

# Train model
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluate model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
# Streamlit app
st.title('Energy Consumption Prediction')
st.write(f'MSE: {mse:.2f}, R-squared: {r2:.2f}')

# Visualization
fig, ax = plt.subplots()
ax.scatter(y_test, y_pred, alpha=0.5, color='blue')
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r',
lw=2)
ax.set_xlabel('Actual Energy Consumption')
ax.set_ylabel('Predicted Energy Consumption')
st.pyplot(fig)

# User input for prediction
st.sidebar.header('Predict Energy Consumption')
features = {col: st.sidebar.number_input(f'Enter {col}:',
float(df[col].min()), float(df[col].max())) for col in X.columns}

if st.sidebar.button('Predict'):
    input_data = np.array([features[col] for col in X.columns]).reshape(1, -
1)
    prediction = model.predict(input_data)[0]
    st.sidebar.write(f'Predicted Energy Consumption: {prediction:.2f} kWh')
```

[Your Task]

```
1. Install the dependencies
2. Collect energy usage dataset from Open Energy Data sources.
3. Perform exploratory data analysis (EDA) to understand trends.
4. Train a linear regression model with temperature and time-based
   features.
5. Optimize the model with Gradient Descent.
6. Deploy a simple dashboard to display energy usage trends.
7. Compile and run the code
8. Print and show the output
```