**PROJECT CODE:**  
  
**EXPLORATORY DATA ANALYSIS PART:**

import cudf

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import LabelEncoder

df = cudf.read\_csv('insurance.csv')

print(df.head())

print(df.shape)

print(df.dtypes)

print(df.isnull().sum())

print(df.describe())

df['charges'].to\_pandas().hist()

plt.xlabel('Charges')

plt.ylabel('Frequency')

plt.title('Distribution of Charges')

plt.show()

df['age'].to\_pandas().hist()

plt.xlabel('Age')

plt.ylabel('Frequency')

plt.title('Distribution of Age')

plt.show()

df['sex'].value\_counts()

df['sex'].to\_pandas().hist()

plt.xlabel('Sex')

plt.ylabel('Frequency')

plt.title('Distribution of Sex')

plt.show()

df['bmi'].to\_pandas().hist()

plt.xlabel('BMI')

plt.ylabel('Frequency')

plt.title('Distribution of BMI')

plt.show()

df['children'].value\_counts()

df['children'].to\_pandas().hist()

plt.xlabel('Children')

plt.ylabel('Frequency')

plt.title('Distribution of Children')

plt.show()

df['smoker'].value\_counts()

df['smoker'].to\_pandas().hist()

plt.xlabel('Smoker')

plt.ylabel('Frequency')

plt.title('Distribution of Smoker')

plt.show()

df['region'].value\_counts()

df['region'].to\_pandas().hist()

plt.xlabel('Region')

plt.ylabel('Frequency')

plt.title('Distribution of Region')

plt.show()

df\_pandas = df.to\_pandas()

le = LabelEncoder()

df\_pandas['sex'] = le.fit\_transform(df\_pandas['sex'])

df\_pandas['smoker'] = le.fit\_transform(df\_pandas['smoker'])

df\_pandas['region'] = le.fit\_transform(df\_pandas['region'])

df = cudf.from\_pandas(df\_pandas)

corr = df.corr()

plt.figure(figsize=(12,10))

sns.heatmap(corr.to\_pandas(), annot=True, cmap='coolwarm')

plt.title('Correlation Matrix')

plt.show()

sns.pairplot(df\_pandas, x\_vars=['age', 'bmi', 'children'], y\_vars='charges', hue='sex', kind='reg')

plt.show()

sns.pairplot(df\_pandas, x\_vars=['age', 'bmi', 'children'], y\_vars='charges', hue='sex', kind='reg')

plt.show()

sns.pairplot(df\_pandas, x\_vars=['age', 'bmi', 'children'], y\_vars='charges', hue='region', kind='reg')

plt.show()

sns.boxplot(x='sex', y='charges', data=df\_pandas)

plt.show()

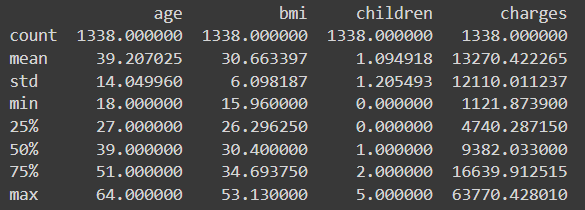
sns.boxplot(x='smoker', y='charges', data=df\_pandas)

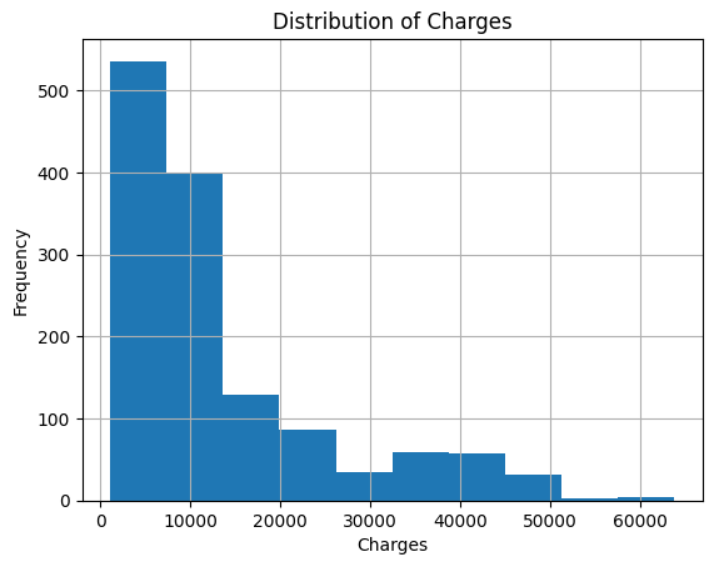
plt.show()

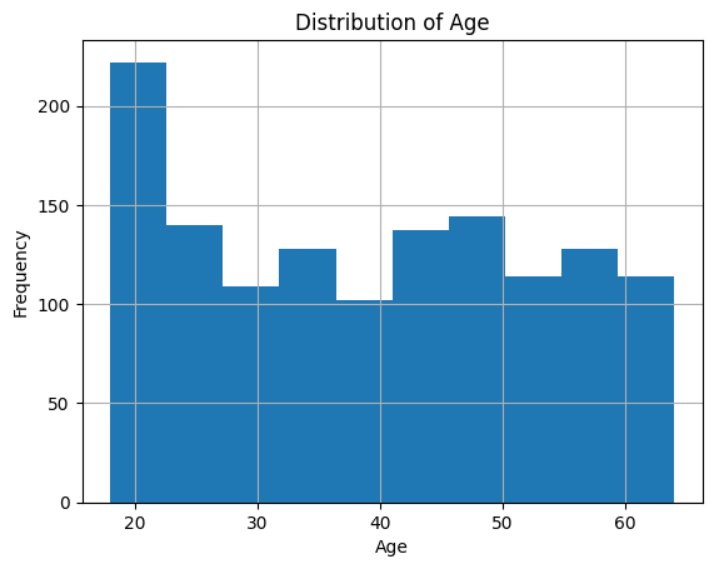
sns.boxplot(x='region', y='charges', data=df\_pandas)

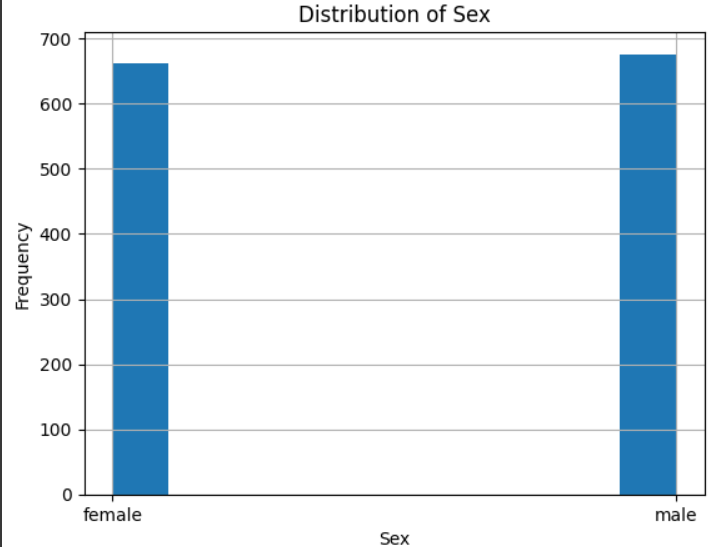
plt.show()

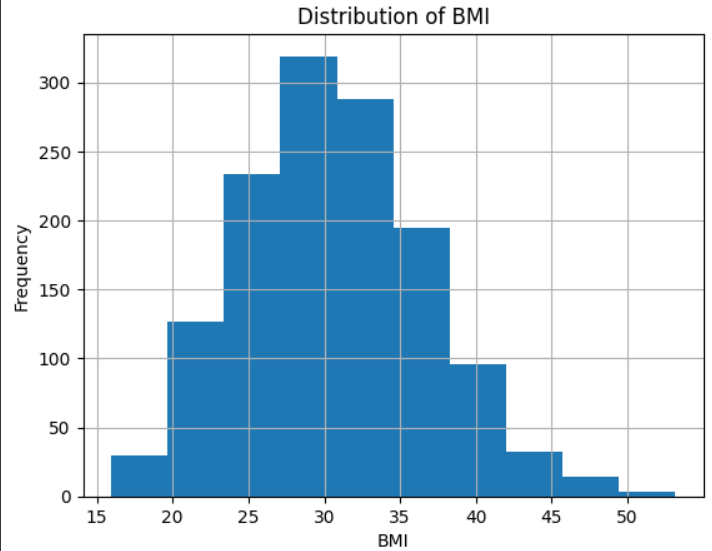
**DESCRIBE():**

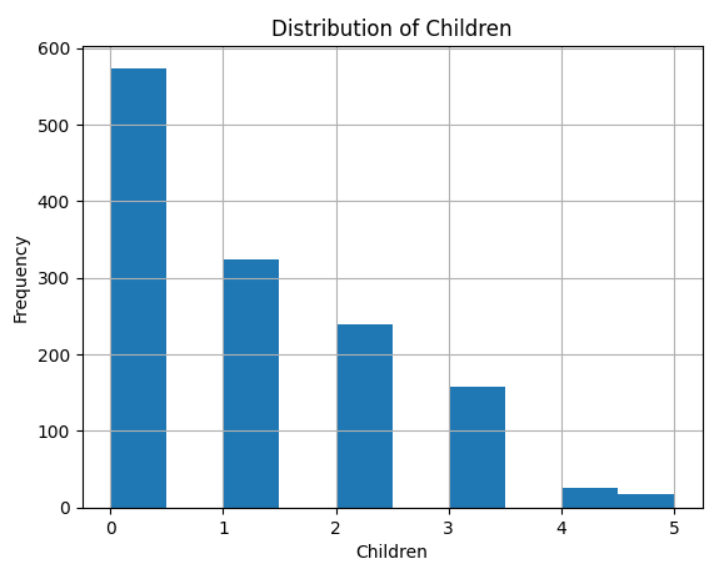
****

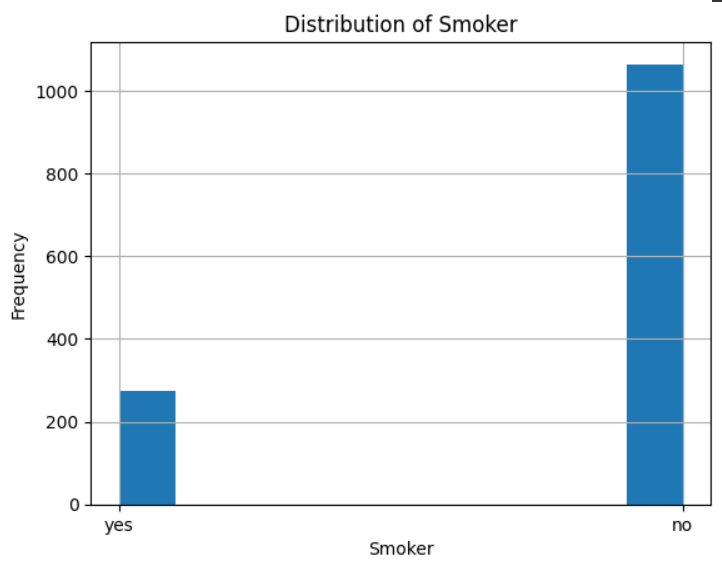
**HISTOGRAMS:**

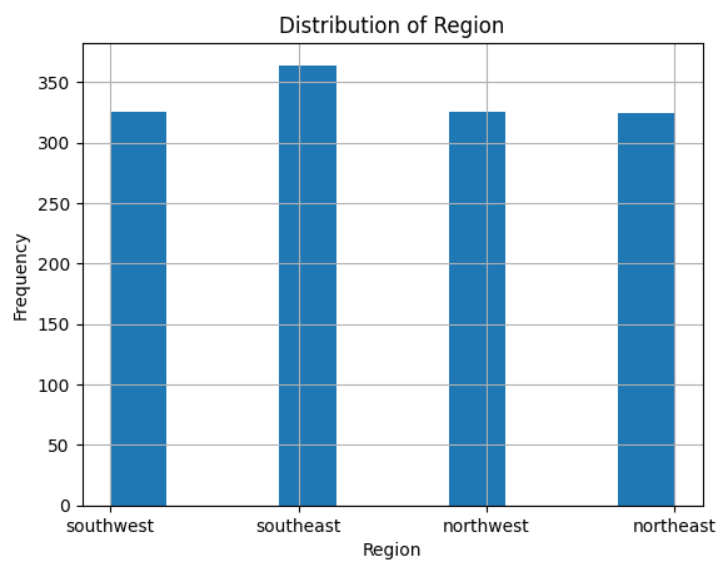
****

****

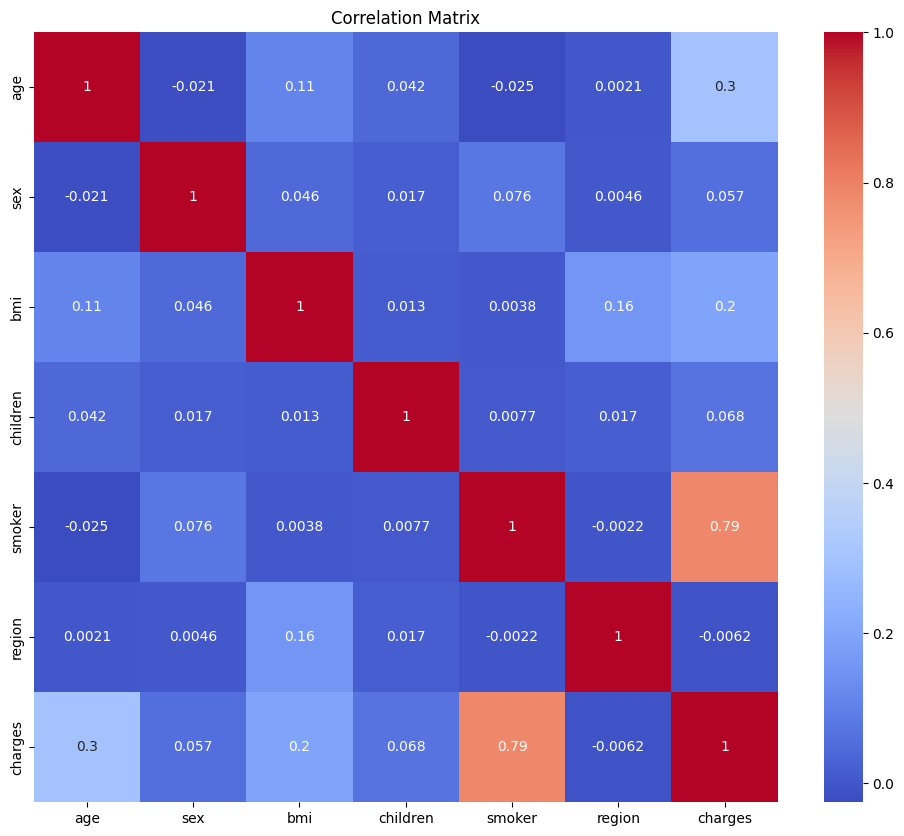
****

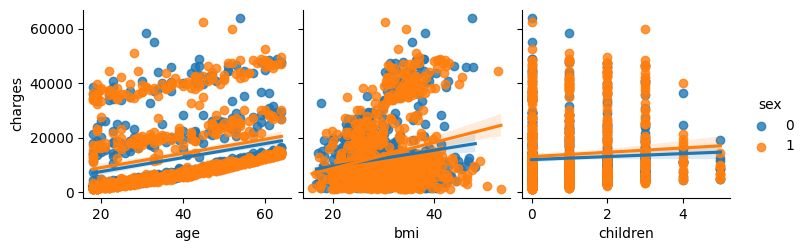
****

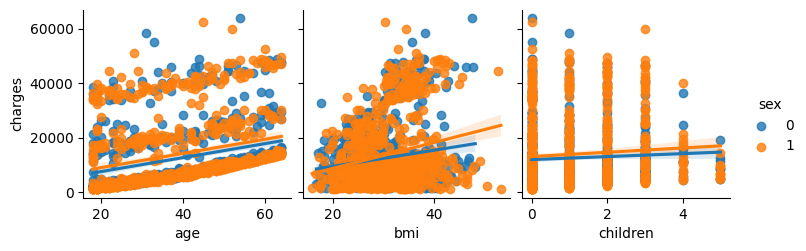
****

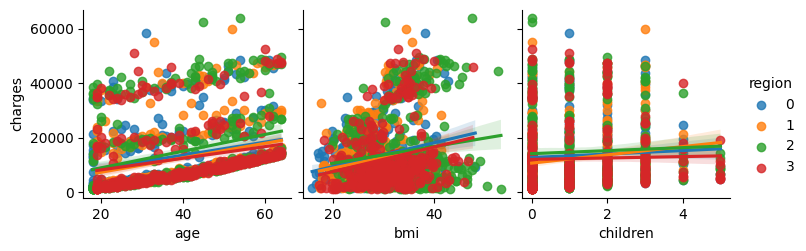
****

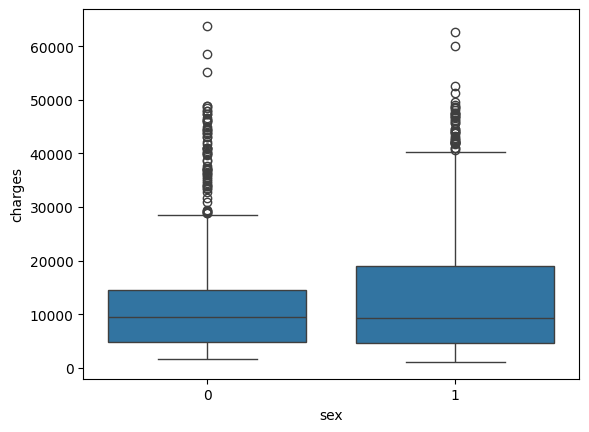
**CORRELATION MATRIX:**

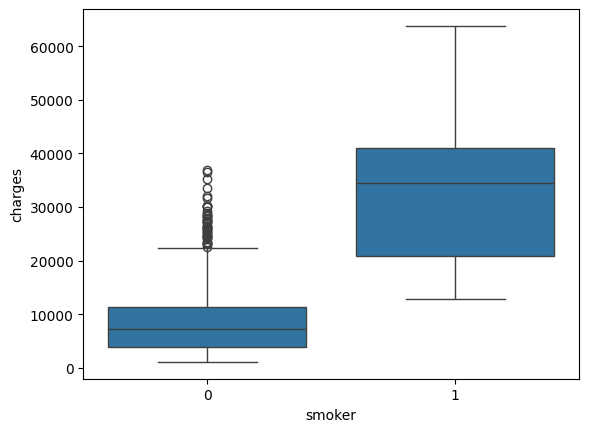


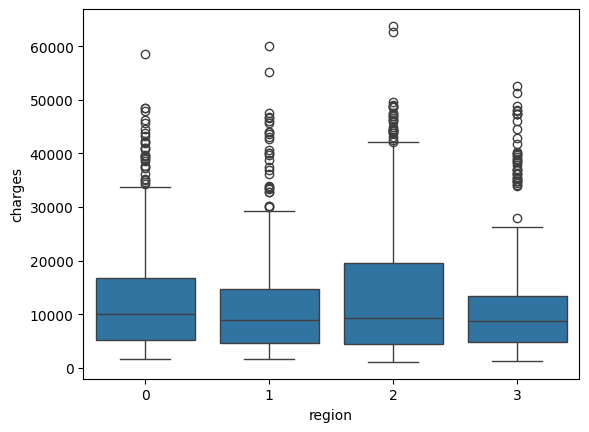
**PAIR PLOTS:** 





**BOX PLOTS:**





**MODEL BUILDING FOR PREDICTION PART:**

import cudf

import cupy as cp

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.metrics import mean\_absolute\_error, accuracy\_score

import torch

import torch.nn as nn

import torch.optim as optim

from sklearn.neighbors import KNeighborsRegressor

from sklearn.linear\_model import LinearRegression

from sklearn.svm import SVR

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

print('Using device:', device)

df = cudf.read\_csv('insurance.csv')

df = df.dropna()

X = df.drop('charges', axis=1)

y = df['charges']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)

scaler = StandardScaler()

X\_train\_num = X\_train.drop(['sex', 'smoker', 'region'], axis=1)

X\_train\_num = cp.asarray(scaler.fit\_transform(X\_train\_num.values.get()))

encoder = OneHotEncoder(sparse=False)

X\_train\_cat = X\_train[['sex', 'smoker', 'region']]

X\_train\_cat = encoder.fit\_transform(X\_train\_cat.to\_pandas())

X\_train\_cat = cp.asarray(X\_train\_cat)

X\_train = cp.concatenate((X\_train\_num, X\_train\_cat), axis=1)

X\_test\_num = X\_test.drop(['sex', 'smoker', 'region'], axis=1)

X\_test\_num = cp.asarray(scaler.transform(X\_test\_num.values.get()))

X\_test\_cat = X\_test[['sex', 'smoker', 'region']]

X\_test\_cat = encoder.transform(X\_test\_cat.to\_pandas())

X\_test\_cat = cp.asarray(X\_test\_cat)

X\_test = cp.concatenate((X\_test\_num, X\_test\_cat), axis=1)

X\_train\_tensor = torch.tensor(X\_train, dtype=torch.float32, device=device)

y\_train\_tensor = torch.tensor(y\_train.values, dtype=torch.float32, device=device)

X\_test\_tensor = torch.tensor(X\_test, dtype=torch.float32, device=device)

y\_test\_tensor = torch.tensor(y\_test.values, dtype=torch.float32, device=device)

class NeuralNet(nn.Module):

    def \_\_init\_\_(self, input\_size):

        super(NeuralNet, self).\_\_init\_\_()

        self.fc1 = nn.Linear(input\_size, 64)

        self.fc2 = nn.Linear(64, 1)

    def forward(self, x):

        x = torch.relu(self.fc1(x))

        x = self.fc2(x)

        return x

class SklearnModel:

    def \_\_init\_\_(self, model):

        self.model = model

    def fit(self, X, y):

        self.model.fit(X, y)

    def predict(self, X):

        return self.model.predict(X)

models = [

    SklearnModel(KNeighborsRegressor()),

    SklearnModel(LinearRegression()),

    SklearnModel(SVR()),

    SklearnModel(DecisionTreeRegressor()),

    SklearnModel(RandomForestRegressor())

]

model\_names = ['KNN', 'Linear Regression', 'Support Vector Regression', 'Decision Trees', 'Random Forest']

errors = []

for model in models:

    if isinstance(model, NeuralNet):

        model.to(device)

        criterion = nn.L1Loss()

        optimizer = optim.Adam(model.parameters())

        for epoch in range(50):

            model.train()

            optimizer.zero\_grad()

            outputs = model(X\_train\_tensor)

            loss = criterion(outputs.squeeze(), y\_train\_tensor)

            loss.backward()

            optimizer.step()

        model.eval()

        with torch.no\_grad():

            y\_pred = model(X\_test\_tensor)

            error = mean\_absolute\_error(y\_test\_tensor.cpu().numpy(), y\_pred.cpu().numpy())

            errors.append(error)

    else:

        X\_train\_np = cp.asnumpy(X\_train)

        y\_train\_np = cp.asnumpy(y\_train)

        X\_test\_np = cp.asnumpy(X\_test)

        model.fit(X\_train\_np, y\_train\_np)

        y\_pred = model.predict(X\_test\_np)

        error = mean\_absolute\_error(cp.asnumpy(y\_test), y\_pred)

        errors.append(error)

results = dict(zip(model\_names, errors))

for name, error in results.items():

    print(f'{name}: {error:.2f}')

plt.figure(figsize=(10,8))

plt.bar(results.keys(), results.values(), color=['blue', 'orange', 'green', 'red', 'purple'])

plt.xlabel('Model')

plt.ylabel('Mean Absolute Error')

plt.title('Comparison of Model Performance')

plt.show()

**FINAL OUTPUT :**

KNN: 3532.65

Linear Regression: 4243.65

Support Vector Regression: 8478.46

Decision Trees: 2868.22

Random Forest: 2576.46

