Regression

Data collection

```
import pandas as pd


# Read the csv file
df = pd.read_csv('/content/Health_insurance.csv')


#Display the first 5 rows
df.head()
```

Show hidden output

Remove Duplicate Records

```
df = df.drop_duplicates(subset=['pid'])
```

Remove Outliers

```
# prompt: remove outliers in all numeric columns

import numpy as np
# Assuming 'df' is your DataFrame and contains numeric columns you want to process.
def remove_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df_filtered = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
    return df_filtered

numeric_cols = df.select_dtypes(include=np.number).columns

for col in numeric_cols:
    df = remove_outliers_iqr(df, col)
```

Handle missing values

```
numeric_cols
```

```
Index(['pid', 'age', 'bmi', 'children', 'diabetes', 'heart rate', 'Creatinine',
       'glucose', 'charges'],
      dtype='object')
```

```
# Check missing values in all columns
df.isnull().sum()
```

Show hidden output

```
# Drop rows having NA values in charges column
df = df.dropna(subset=['charges'])



# Create a list of columns to fill NA values
columns_to_fill = ['bmi', 'heart rate','glucose'] # Example columns, replace with your actual columns

# Fill NA values in specified columns with the mean of each column
for col in columns_to_fill:
    df[col] = df[col].fillna(df[col].mean())
```

Clean categorical columns

```
category_cols = df.select_dtypes(include="object").columns
```

```
for col in category_cols:
    print(df[col].unique())
```

Show hidden output

```
# replace 'Female' with female in sex column

dict1 = {'Female':'female','mal':'male'}

df['sex'] = df['sex'].replace(dict1)

# Calculate the mode of the column
mode_value = df['sex'].mode()[0]

# Replace NaN values with the mode
df['sex'].fillna(mode_value, inplace=True)
```

Show hidden output

```
# One-hot encode the sex, smoker, region column data
df = pd.get_dummies(df, columns=['sex', 'smoker', 'region'])


# Display the first 5 rows
df.head()
```

Show hidden output

```
# Normalize numerical columns

# Store min-max values for later use
min_max_values = {}

cols = ['age', 'bmi','heart rate','Creatinine','glucose']
for i in cols:
  min_val = df[i].min()
  max_val = df[i].max()
  df[i] = (df[i] - df[i].min()) / (df[i].max() - df[i].min())
  min_max_values[i] = (min_val, max_val)



min_max_values
```

```
{'age': (18.0, 64.0),
 'bmi': (15.96, 47.52),
 'heart rate': (39.32142857, 129.125),
 'Creatinine': (0.266666667, 3.311111111),
 'glucose': (69.1, 252.0)}
```

```
# Save to a JSON file
import json
with open("min_max_values.json", "w") as json_file:
    json.dump(min_max_values, json_file)


# Display the first 5 rows
df.head()
```

	pid	age	bmi	children	diabetes	heart rate	Creatinine	glucose	charges	sex_female	sex_male	smoker_no	smoker_ye
0	1	0.021739	0.378327	0	1	0.328677	0.555657	0.248969	16884.92400	True	False	False	Tru
2	3	0.217391	0.539924	3	0	0.367433	0.527112	0.436851	4449.46200	False	True	True	Fals
3	4	0.326087	0.213720	0	0	0.614436	0.104797	0.323401	21984.47061	False	True	True	Fals
4	5	0.304348	0.409379	0	0	0.318457	0.552920	0.419081	3866.85520	False	True	True	Fals
5	6	0.282609	0.309886	0	0	0.388185	0.442062	0.159832	3756.62160	True	False	True	Fals

```
df.corr()
```

Show hidden output

```
# prompt: seaborn correlation matrixx

import seaborn as sns
```

```
import matplotlib.pyplot as plt

# Assuming 'df' is your DataFrame
plt.figure(figsize=(12, 10))
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

Show hidden output

Train test split

```
# Separate the features and target variable
X = df.drop(columns = ['charges','pid','sex_male', 'smoker_yes'], axis=1)
y = df['charges']
```

```
# Split the data into train and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Linear Regression

```
# Linear Regression model to predict the outcome

from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(X_train, y_train)
# Predict the target variable for the test set
y_pred = reg.predict(X_test)
```

```
# compute the rmse, r2 score
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
print('RMSE: ', rmse)
print('R2: ', r2)
```

```
RMSE:  4908.881291904217
R2:  0.5316342951865664
```

```
pd.DataFrame(X.columns,reg.coef_)
```

Show hidden output

```
x_test_sample = X_test.iloc[0,:]
x_test_sample_reshaped = np.array(x_test_sample).reshape(1, -1)
```

```
y_pred_sample = reg.predict(x_test_sample_reshaped)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but Line
  warnings.warn(
```

```
y_pred_sample
```

```
array([11494.81605439])
```

```
import joblib
joblib.dump(reg, 'linear_regression_model.pkl')
```

```
['linear_regression_model.pkl']
```

```
!pip install streamlit pyngrok
```

Show hidden output

```
streamlit_code = """
import streamlit as st
import joblib
```

```
import numpy as np
import json

# Load the trained model
model = joblib.load('linear_regression_model.pkl')

# Streamlit App
st.title("Insurance Charges Prediction")
st.write("Enter the input values for prediction.")

# Input fields for each feature

age = st.number_input("Age", min_value=18, max_value=65, step=1, value=30)
bmi = st.number_input("BMI", min_value=15.0, max_value=50.0, step=0.1, value=25.0)
children = st.number_input("Number of Children", min_value=0, max_value=10, step=1, value=0)
diabetes = st.selectbox("Has Diabetes?", options=["No", "Yes"])
heart_rate = st.number_input("Heart Rate", min_value=50, max_value=120, step=1, value=75)
creatinine = st.number_input("Creatinine", min_value=0.0, max_value=2.0, step=0.1, value=1.0)
glucose = st.number_input("Glucose", min_value=50.0, max_value=300.0, step=1.0, value=100.0)

# Categorical features
sex_female = st.selectbox("Sex", options=["Male", "Female"]) == "Female"
smoker_no = st.selectbox("Smoker?", options=["Yes", "No"]) == "No"

# Region selection
region = st.selectbox("Region", options=["Northeast", "Northwest", "Southeast", "Southwest"])
region_northeast = region == "Northeast"
region_northwest = region == "Northwest"
region_southeast = region == "Southeast"
region_southwest = region == "Southwest"

# Normalization function
def min_max_scale(value, min_val, max_val):
    return (value - min_val) / (max_val - min_val)

# Load Min-Max values from JSON
with open("min_max_values.json", "r") as json_file:
    min_max_values = json.load(json_file)

# Normalize inputs
age_norm = min_max_scale(age, *min_max_values['age'])
bmi_norm = min_max_scale(bmi, *min_max_values['bmi'])
heart_rate_norm = min_max_scale(heart_rate, *min_max_values['heart rate'])
creatinine_norm = min_max_scale(creatinine, *min_max_values['Creatinine'])
glucose_norm = min_max_scale(glucose, *min_max_values['glucose'])

# Prepare input data for prediction
input_data = np.array([
    age_norm, bmi_norm, children, int(diabetes == "Yes"), heart_rate_norm, creatinine_norm, glucose_norm,
    int(sex_female), int(smoker_no), int(region_northeast), int(region_northwest),
    int(region_southeast), int(region_southwest)
]).reshape(1, -1)

# Predict button
if st.button("Predict Charges"):
    prediction = model.predict(input_data)
    st.success(f"Predicted Insurance Charges: ${prediction[0]:,.2f}")

"""
# Save the code to a file
with open('app.py', 'w') as f:
    f.write(streamlit_code)
```

Start coding or generate with AI.

```
!streamlit run app.py &>/content/logs.txt &
```

```
from pyngrok import ngrok
ngrok.set_auth_token("2VhfMAwGeQgS75KRWpcXKDgOAKY_2FKRsWFNpPVv6dpUeEUxv")
```

```
from pyngrok import ngrok

# Expose the Streamlit server running on port 8501
public_url = ngrok.connect(8501)
print(f"Streamlit App is live at: {public_url}")
```

```
Streamlit App is live at: NgrokTunnel: "https://87b3-34-125-39-242.ngrok-free.app" -> "http://localhost:8501"
```

```
# Kill all existing ngrok processes
!pkill -f ngrok
```

```
WARNING:pyngrok.process.ngrok:t=2025-03-04T04:20:00+0000 lvl=warn msg="Stopping forwarder" name=http-8501-b0352b71-0ec5-47ee-9dc9-11
WARNING:pyngrok.process.ngrok:t=2025-03-04T04:20:00+0000 lvl=warn msg="Error restarting forwarder" name=http-8501-b0352b71-0ec5-47ee
WARNING:pyngrok.process.ngrok:t=2025-03-04T04:20:00+0000 lvl=warn msg="Stopping forwarder" name=http-8501-d09ece5c-c354-4564-ac09-da
WARNING:pyngrok.process.ngrok:t=2025-03-04T04:20:00+0000 lvl=warn msg="Error restarting forwarder" name=http-8501-d09ece5c-c354-4564
```

Start coding or generate with AI.

Show hidden output

```
# Polynomial regression model to predict the outcome
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=1)
X_poly = poly.fit_transform(X_train)
model = LinearRegression()
model.fit(X_poly, y_train)

# Predict the output for new data

X_new_poly = poly.transform(X_test)
y_pred = model.predict(X_new_poly)

# Calculate the mean squared error
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
print('RMSE:', rmse)
print('R2:', r2)
```

```
RMSE: 6969.8499953743
R2: 0.617955748832657
```

Start coding or generate with AI.