**Random Forest model with 0.85 R^2 score**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split, cross\_val\_score, GridSearchCV

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

import joblib

import streamlit as st

import os

# Title for the Streamlit app

st.title('Medical Cost Prediction 💰 using Random Forest 🌳🔥')

# File upload widget

uploaded\_file = st.file\_uploader("Upload a CSV file", type='csv')

if uploaded\_file is not None:

    try:

        data = pd.read\_csv(uploaded\_file)

        st.write("Dataset Preview:")

        st.write(data.head())

    except Exception as e:

        st.error(f"Error loading file: {e}")

        st.stop()

    # Detect Outliers using IQR 🔍

    Q1 = data['charges'].quantile(0.25)

    Q3 = data['charges'].quantile(0.75)

    IQR = Q3 - Q1

    outliers = data[(data['charges'] < Q1 - 1.5 \* IQR) | (data['charges'] > Q3 + 1.5 \* IQR)]

    st.write(f"Number of Outliers Detected: {len(outliers)}")

    st.write("Before Capping Outliers:")

    st.write(outliers)

    if len(outliers) > 0:

        st.warning("Outliers detected 🚨❗ Capping the outliers to make the model more accurate 🔥")

        upper\_limit = Q3 + 1.5 \* IQR

        data['charges'] = data['charges'].apply(lambda x: upper\_limit if x > upper\_limit else x)

        st.success("Outliers have been capped ✂️📌")

    else:

        st.write("No Outliers Found ✅")

    st.write("After Capping Outliers:")

    st.write(data.head())

    # Encode categorical variables

    data = pd.get\_dummies(data, columns=['sex', 'smoker', 'region'], drop\_first=True)

    # Scale numerical features

    scaler = StandardScaler()

    numerical\_cols = ['age', 'bmi', 'children']

    data[numerical\_cols] = scaler.fit\_transform(data[numerical\_cols])

    # Features (X) and target variable (y)

    X = data.drop(columns=['charges'])

    y = data['charges']

    # Train-Test Split

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

    # Hyperparameter Tuning using GridSearchCV

    param\_grid = {

        'n\_estimators': [100, 200, 300],

        'max\_depth': [None, 10, 20],

        'min\_samples\_split': [2, 5, 10],

        'min\_samples\_leaf': [1, 2, 4]

    }

    rf\_model = RandomForestRegressor(random\_state=42)

    grid\_search = GridSearchCV(estimator=rf\_model, param\_grid=param\_grid, cv=5, scoring='r2')

    grid\_search.fit(X\_train, y\_train)

    best\_model = grid\_search.best\_estimator\_

    st.subheader("🔥 Best Parameters Found Using GridSearchCV")

    st.write(grid\_search.best\_params\_)

    # Cross Validation Scoring

    cross\_val\_scores = cross\_val\_score(best\_model, X\_train, y\_train, cv=5, scoring='r2')

    st.subheader("Cross-Validation R² Scores 📌")

    st.write(cross\_val\_scores)

    st.write(f"Average Cross-Validation R² Score: {cross\_val\_scores.mean():.2f}")

    # Save model

    joblib.dump(best\_model, "random\_forest\_model.pkl")

    # User Inputs for Prediction

    st.subheader('Enter Your Details for Prediction 📊')

    age = st.slider("Age", 18, 100, 30)

    bmi = st.slider("BMI", 10, 50, 25)

    children = st.slider("Children", 0, 10, 1)

    sex = st.selectbox("Sex", ["Male", "Female"])

    smoker = st.selectbox("Smoker", ["Yes", "No"])

    region = st.selectbox("Region", ["Northwest", "Northeast", "Southeast", "Southwest"])

    # Prepare input for model

    user\_input = pd.DataFrame({

        'age': [age],

        'bmi': [bmi],

        'children': [children],

        'smoker\_yes': [1 if smoker == "Yes" else 0],

        'sex\_male': [1 if sex == "Male" else 0],

        'region\_northwest': [1 if region == "Northwest" else 0],

        'region\_southeast': [1 if region == "Southeast" else 0],

        'region\_southwest': [1 if region == "Southwest" else 0]

    })

    # Ensure column order matches training data

    user\_input = user\_input.reindex(columns=X\_train.columns, fill\_value=0)

    # Scale input features

    user\_input[numerical\_cols] = scaler.transform(user\_input[numerical\_cols])

    # Prediction

    if st.button("Predict Cost 💵"):

        try:

            prediction = best\_model.predict(user\_input)

            st.success(f"Predicted Medical Cost: ${prediction[0]:,.2f}")

        except Exception as e:

            st.error(f"Error in prediction: {e}")

    # Model Evaluation

    y\_pred = best\_model.predict(X\_test)

    st.subheader('Model Performance 📈')

    st.write(f"Mean Absolute Error: {mean\_absolute\_error(y\_test, y\_pred):,.2f}")

    st.write(f"Mean Squared Error: {mean\_squared\_error(y\_test, y\_pred):,.2f}")

    st.write(f"R² Score: {r2\_score(y\_test, y\_pred):.2f}")

    # Feature Importance 🔥

    st.subheader("Feature Importance 🌟")

    importance = pd.Series(best\_model.feature\_importances\_, index=X.columns)

    importance = importance.sort\_values(ascending=False)

    st.write(importance)

    # Visualizations

    st.subheader('Visualizations 📊')

    plot\_option = st.selectbox('Choose a visualization:',

                               ['Distribution of Charges', 'Charges by Smoking Status',

                                'BMI vs Charges', 'Age Distribution',

                                'Residuals Distribution', 'Actual vs Predicted'])

    if plot\_option == 'Distribution of Charges':

        fig, ax = plt.subplots()

        sns.histplot(data['charges'], kde=True, ax=ax)

        ax.set\_title('Distribution of Charges')

        st.pyplot(fig)

    elif plot\_option == 'BMI vs Charges':

        fig, ax = plt.subplots()

        sns.scatterplot(data=data, x='bmi', y='charges', hue='smoker\_yes', ax=ax)

        ax.set\_title('BMI vs Charges')

        st.pyplot(fig)

    elif plot\_option == 'Charges by Smoking Status':

        fig, ax = plt.subplots()

        sns.boxplot(x='smoker\_yes', y='charges', data=data, ax=ax)

        ax.set\_title('Charges by Smoking Status')

        st.pyplot(fig)

    elif plot\_option == 'Age Distribution':

        fig, ax = plt.subplots()

        sns.histplot(data['age'], kde=True, ax=ax)

        ax.set\_title('Age Distribution')

        st.pyplot(fig)

    elif plot\_option == 'Residuals Distribution':

        residuals = y\_test - y\_pred

        fig, ax = plt.subplots()

        sns.histplot(residuals, kde=True, color='purple', ax=ax)

        ax.set\_title('Residuals Distribution')

        st.pyplot(fig)

    elif plot\_option == 'Actual vs Predicted':

        fig, ax = plt.subplots()

        ax.scatter(y\_test, y\_pred, color='blue', alpha=0.6)

        ax.plot([min(y\_test), max(y\_test)], [min(y\_test), max(y\_test)], color='red', linestyle='--')

        ax.set\_title('Actual vs Predicted Charges')

        st.pyplot(fig)

**Random Forest model with 0.88 R^2 score**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split, cross\_val\_score, RandomizedSearchCV

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

import joblib

import streamlit as st

import os

st.title('Medical Cost Prediction 💰 using Random Forest 🌳🔥')

uploaded\_file = st.file\_uploader("Upload a CSV file", type='csv')

if uploaded\_file is not None:

    try:

        data = pd.read\_csv(uploaded\_file)

        st.write("Dataset Preview:")

        st.write(data.head())

    except Exception as e:

        st.error(f"Error loading file: {e}")

        st.stop()

    if 'charges' not in data.columns:

        st.error("Dataset must contain a 'charges' column for prediction.")

        st.stop()

    # Save original for UI sliders/dropdowns

    original\_data = data.copy()

    # Drop missing values from target

    data.dropna(subset=['charges'], inplace=True)

    # Outlier Detection & Capping

    Q1 = data['charges'].quantile(0.25)

    Q3 = data['charges'].quantile(0.75)

    IQR = Q3 - Q1

    upper\_limit = Q3 + 1.5 \* IQR

    data['charges'] = data['charges'].apply(lambda x: upper\_limit if x > upper\_limit else x)

    # Feature separation

    target\_col = 'charges'

    feature\_cols = [col for col in data.columns if col != target\_col]

    categorical\_cols = data[feature\_cols].select\_dtypes(include=['object', 'category']).columns.tolist()

    numerical\_cols = data[feature\_cols].select\_dtypes(include=['int64', 'float64']).columns.tolist()

    # Encoding & Scaling

    data = pd.get\_dummies(data, columns=categorical\_cols, drop\_first=True)

    scaler = StandardScaler()

    data[numerical\_cols] = scaler.fit\_transform(data[numerical\_cols])

    # Split features and target

    X = data.drop(columns=[target\_col])

    y = data[target\_col]

    # Train-test split

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

    # Model tuning

    param\_dist = {

        'n\_estimators': [100, 200, 300, 400, 500],

        'max\_depth': [None, 10, 20, 30, 40],

        'min\_samples\_split': [2, 5, 10, 15],

        'min\_samples\_leaf': [1, 2, 4, 6],

        'max\_features': ['auto', 'sqrt', 'log2']

    }

    rf\_model = RandomForestRegressor(random\_state=42)

    random\_search = RandomizedSearchCV(rf\_model, param\_distributions=param\_dist, n\_iter=50, cv=5, scoring='r2', random\_state=42, n\_jobs=-1)

    random\_search.fit(X\_train, y\_train)

    best\_model = random\_search.best\_estimator\_

    st.subheader("🔥 Best Parameters Found Using RandomizedSearchCV")

    st.write(random\_search.best\_params\_)

    cross\_val\_scores = cross\_val\_score(best\_model, X\_train, y\_train, cv=5, scoring='r2')

    st.subheader("Cross-Validation R² Scores 📌")

    st.write(cross\_val\_scores)

    st.write(f"Average Cross-Validation R² Score: {cross\_val\_scores.mean():.2f}")

    joblib.dump(best\_model, "random\_forest\_model.pkl")

    st.subheader('Enter Your Details for Prediction 📊')

    user\_inputs = {}

    for col in numerical\_cols:

        if col.lower() == 'age':

            user\_inputs[col] = st.slider("Age", min\_value=17, max\_value=90, value=30)

        elif col.lower() == 'bmi':

            user\_inputs[col] = st.slider("BMI", min\_value=14.0, max\_value=40.0, value=25.0)

        elif col.lower() == 'children':

            user\_inputs[col] = st.slider("Children", min\_value=0, max\_value=20, value=1)

        else:

            col\_min = float(original\_data[col].min())

            col\_max = float(original\_data[col].max())

            default\_val = float(original\_data[col].median())

            user\_inputs[col] = st.slider(f"{col.capitalize()}", min\_value=round(col\_min, 2), max\_value=round(col\_max, 2), value=round(default\_val, 2))

    for col in categorical\_cols:

        options = original\_data[col].dropna().unique().tolist()

        selected = st.selectbox(f"{col.capitalize()}", options)

        for opt in options:

            user\_inputs[f"{col}\_{opt}"] = 1 if selected == opt else 0

    user\_input\_df = pd.DataFrame([user\_inputs])

    user\_input\_df = user\_input\_df.reindex(columns=X\_train.columns, fill\_value=0)

    user\_input\_df[numerical\_cols] = scaler.transform(user\_input\_df[numerical\_cols])

    if st.button("Predict Cost 💵"):

        try:

            prediction = best\_model.predict(user\_input\_df)

            st.success(f"Predicted Medical Cost: Rs.{prediction[0]:,.2f}")

        except Exception as e:

            st.error(f"Error in prediction: {e}")

    y\_pred = best\_model.predict(X\_test)

    st.subheader('Model Performance 📈')

    st.write(f"Mean Absolute Error: {mean\_absolute\_error(y\_test, y\_pred):,.2f}")

    st.write(f"Mean Squared Error: {mean\_squared\_error(y\_test, y\_pred):,.2f}")

    st.write(f"R² Score: {r2\_score(y\_test, y\_pred):.2f}")

    st.subheader("Feature Importance 🌟")

    importance = pd.Series(best\_model.feature\_importances\_, index=X.columns)

    importance = importance.sort\_values(ascending=False)

    st.write(importance)

    st.subheader('Visualizations 📊')

    plot\_option = st.selectbox('Choose a visualization:',

                               ['Distribution of Charges', 'Residuals Distribution', 'Actual vs Predicted'])

    if plot\_option == 'Distribution of Charges':

        fig, ax = plt.subplots()

        sns.histplot(y, kde=True, ax=ax)

        ax.set\_title('Distribution of Charges')

        st.pyplot(fig)

    elif plot\_option == 'Residuals Distribution':

        residuals = y\_test - y\_pred

        fig, ax = plt.subplots()

        sns.histplot(residuals, kde=True, color='purple', ax=ax)

        ax.set\_title('Residuals Distribution')

        st.pyplot(fig)

    elif plot\_option == 'Actual vs Predicted':

        fig, ax = plt.subplots()

        ax.scatter(y\_test, y\_pred, color='blue', alpha=0.6)

        ax.plot([min(y\_test), max(y\_test)], [min(y\_test), max(y\_test)], color='red', linestyle='--')

        ax.set\_title('Actual vs Predicted Charges')

        st.pyplot(fig)