Assignment 8: Machine learning basics (REGRESSION MODELS IN ML)

Objectives

1. Understand the concept and application of regression analysis.
2. Implement different regression models.
3. Evaluate the performance of the models using various metrics.

Link - https://www.kaggle.com/datasets/simranjain17/insurance?select=insurance.csv

```
Task 1: Data Exploration and Preprocessing
1. Load the dataset and display the first few rows.
2. Perform basic statistical analysis to understand the distribution of the features.
3. Check for missing values and handle them appropriately.
4. Check for categorical features and convert them to numerical features.
5. Perform feature engineering, including the creation of new features and scaling of numerical features.
6. Split the data into training and testing sets.
```

```
#1. Load the dataset and display the first few rows.
import pandas as pd
import numpy as np
df = pd.read_csv('insurance.csv')
df.head()
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

```
#2. Perform basic statistical analysis to understand the distribution of the features.
df.describe()
```

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

```
#Analyze categorical features using df['column_name'].value_counts().
print('value count at Sex',df['sex'].value_counts())
print('value count at Smoker',df['smoker'].value_counts())
print('value count at Region',df['region'].value_counts())
```

```
value count at Sex sex
male      676
female    662
Name: count, dtype: int64
value count at Smoker smoker
no     1064
yes     274
Name: count, dtype: int64
value count at Region region
southeast    364
southwest    325
northwest    325
northeast    324
Name: count, dtype: int64
```

```
#3. Check for missing values and handle them appropriately.
df.isnull().sum()
# No missing Values
```

	0
age	0
sex	0
bmi	0
children	0
smoker	0
region	0
charges	0

dtype: int64

```
#4. Check for categorical features and convert them to numerical features.
##### ------ DO NOT RUN AS WE ARE WORKING ON LINEAR REGRESSION --------#####
df = pd.get_dummies(df,columns=['sex','smoker','region'])
df.head()
```

	age	bmi	children	charges	sex_female	sex_male	smoker_no	smoker_yes	region_northeast	region_northwest	region_southe
0	19	27.900	0	16884.92400	True	False	False	True	False	False	Fa
1	18	33.770	1	1725.55230	False	True	True	False	False	False	T
2	28	33.000	3	4449.46200	False	True	True	False	False	False	T
3	33	22.705	0	21984.47061	False	True	True	False	False	True	Fa
4	32	28.880	0	3866.85520	False	True	True	False	False	True	Fa

```
#By using Label Encoder
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['sex'] = le.fit_transform(df['sex'])
#df.head()
df['smoker'] = le.fit_transform(df['smoker'])
#df.head()
df['region'] = le.fit_transform(df['region'])
df.head()
```

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	1	3	16884.92400
1	18	1	33.770	1	0	2	1725.55230
2	28	3	33.000	3	0	2	4449.46200
3	33	0	22.705	0	0	1	21984.47061
4	32	0	28.880	0	0	1	3866.85520

Scale numerical features using StandardScaler or MinMaxScaler. Check for correlations among features (df.corr()) and remove highly correlated ones if necessary.

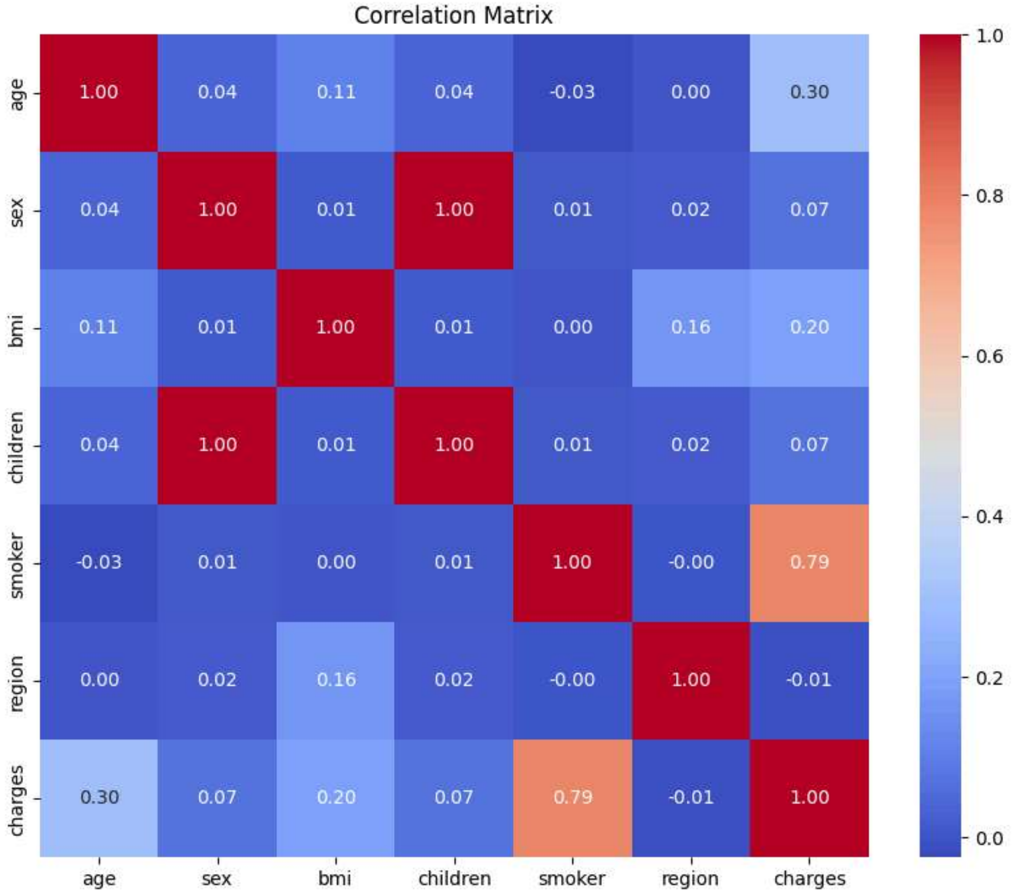
```
#5. Perform feature engineering, including the creation of new features and scaling of numerical features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df[['age','bmi','children']] = scaler.fit_transform(df[['age','bmi','children']])
df.head()
```

	age	sex	bmi	children	smoker	region	charges
0	-1.438764	0	-0.453320	-0.908614	1	3	16884.92400
1	-1.509965	1	0.509621	-0.078767	0	2	1725.55230
2	-0.797954	3	0.383307	1.580926	0	2	4449.46200
3	-0.441948	0	-1.305531	-0.908614	0	1	21984.47061
4	-0.513149	0	-0.292556	-0.908614	0	1	3866.85520

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Compute correlation matrix
correlation_matrix = df.corr()

# Visualize the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```




```
###--- DO NOT RUN ---###
#As correation map is all less than 0.85.
# Threshold for high correlation
threshold = 0.85

# Find correlated features
correlated_features = set()
for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > threshold:
            colname = correlation_matrix.columns[i]
            correlated_features.add(colname)

# Drop correlated features
df.drop(columns=correlated_features, inplace=True)
print("Dropped correlated features:", correlated_features)
```

```
df.corr()
```

	age	sex	bmi	children	smoker	region	charges
age	1.000000	0.042469	0.109272	0.042469	-0.025019	0.002127	0.299008
sex	0.042469	1.000000	0.012759	1.000000	0.007673	0.016569	0.067998
bmi	0.109272	0.012759	1.000000	0.012759	0.003750	0.157566	0.198341
children	0.042469	1.000000	0.012759	1.000000	0.007673	0.016569	0.067998
smoker	-0.025019	0.007673	0.003750	0.007673	1.000000	-0.002181	0.787251
region	0.002127	0.016569	0.157566	0.016569	-0.002181	1.000000	-0.006208
charges	0.299008	0.067998	0.198341	0.067998	0.787251	-0.006208	1.000000

```
#6. Split the data into training and testing sets.
from sklearn.model_selection import train_test_split
X = df.drop('charges', axis=1)
y = df['charges']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1070, 6)
(268, 6)
(1070,)
(268,)
```

Task 2 -> Linear Regression

Task 2: Implement Regression Models

1. Train the following regression models:

o Linear Regression <-- (Let us Focus on This First)

o Decision Tree Regression

o Random Forest Regression

o Gradient Boosting Regression

o Support Vector Regression (SVR)

2. For each model, train it using the training set and predict on the testing set.

```
#Model Definition. --
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
```

Task 3 --> Linear Regression

Model Evaluation

1. Evaluate each model using the following metrics:

o Mean Absolute Error (MAE)

o Mean Squared Error (MSE)

o Root Mean Squared Error (RMSE)

o Mean Absolute Percentage Error (MAPE)

o R-squared (R2)

2. Compare the performance of the models based on these metrics and find out which model performs the best

```
# Mean Absolute Error (MAE)
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_test, y_pred_lr)
print(mae)
```

```
4187.322474715386
```

```
# Mean Squared Error (MSE)
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred_lr)
print(mse)
```

```
33640657.13645164
```

```
# Root Mean Squared Error (RMSE)
import numpy as np
rmse = np.sqrt(mse)
print(rmse)
```

```
5800.056649417455
```

```
# Mean Absolute Percentage Error (MAPE)
from sklearn.metrics import mean_absolute_percentage_error
mape_1 = mean_absolute_percentage_error(y_test, y_pred_lr)
mape_2 = np.mean(np.abs((y_test - y_pred_lr) / y_test)) * 100
print(mape_1)
print(mape_2)
```

```
0.4713245248823263
47.13245248823263
```

```
# R-squared (R2)
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred_lr)
print(r2)
```

```
0.7833112270019789
```

Start coding or generate with AI.

Task 2 -- Decision Tree

```
Task 2: Implement Regression Models
1. Train the following regression models:
o Linear Regression
o Decision Tree Regression <-- (Let us Focus on This Now)
o Random Forest Regression
o Gradient Boosting Regression
o Support Vector Regression (SVR)
2. For each model, train it using the training set and predict on the testing set.
```

```
#Model Definition. --
from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor(random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
```

Task 3 -> Decision Tree

```
Model Evaluation
1. Evaluate each model using the following metrics:
o Mean Absolute Error (MAE)
o Mean Squared Error (MSE)
o Root Mean Squared Error (RMSE)
o Mean Absolute Percentage Error (MAPE)
o R-squared (R2)
2. Compare the performance of the models based on these metrics and find out which model performs
the best
```

```
# Mean Absolute Error (MAE)
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_test, y_pred_dt)
print(mae)
```

```
2825.7985160037315
```

```
# Mean Squared Error (MSE)
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred_dt)
print(mse)
```

```
36954941.25856166
```

```
# Root Mean Squared Error (RMSE)
import numpy as np
rmse = np.sqrt(mse)
print(rmse)
```

```
6079.057596253029
```

```
# Mean Absolute Percentage Error (MAPE)
from sklearn.metrics import mean_absolute_percentage_error
mape_1 = mean_absolute_percentage_error(y_test, y_pred_dt)
mape_2 = np.mean(np.abs((y_test - y_pred_dt) / y_test)) * 100
print(mape_1)
print(mape_2)
```

```
0.3079511465933207
30.795114659332068
```

```
# R-squared (R2)
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred_dt)
print(r2)
```

```
0.7619630066960008
```

Start coding or generate with AI.

```
# This is formatted as code
```

Task 2 -- Random Forest

```
Task 2: Implement Regression Models
1. Train the following regression models:
o Linear Regression
o Decision Tree Regression
o Random Forest Regression <-- (Let us Focus on This Now)
o Gradient Boosting Regression
o Support Vector Regression (SVR)
2. For each model, train it using the training set and predict on the testing set.
```

```
#Model Definition. --
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
```

Task 3 -> Random Forest

```
Model Evaluation
1. Evaluate each model using the following metrics:
o Mean Absolute Error (MAE)
o Mean Squared Error (MSE)
o Root Mean Squared Error (RMSE)
o Mean Absolute Percentage Error (MAPE)
o R-squared (R2)
2. Compare the performance of the models based on these metrics and find out which model performs
the best
```

```
# Mean Absolute Error (MAE)
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_test, y_pred_rf)
print(mae)
```

```
2443.8447090707723
```

```
# Mean Squared Error (MSE)
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred_rf)
print(mse)
```

```
20226798.292691052
```

```
# Root Mean Squared Error (RMSE)
import numpy as np
rmse = np.sqrt(mse)
print(rmse)
```

```
4497.421293662742
```

```
# Mean Absolute Percentage Error (MAPE)
from sklearn.metrics import mean_absolute_percentage_error
mape_1 = mean_absolute_percentage_error(y_test, y_pred_rf)
mape_2 = np.mean(np.abs((y_test - y_pred_rf) / y_test)) * 100
print(mape_1)
print(mape_2)
```

```
0.2848107120782547
28.48107120782547
```

```
# R-squared (R2)
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred_rf)
print(r2)
```

```
0.8697136002443739
```

Start coding or generate with AI.

Task 2 -- Gradient Boosting Regression

```
Task 2: Implement Regression Models
1. Train the following regression models:
o Linear Regression
o Decision Tree Regression
o Random Forest Regression
o Gradient Boosting Regression <-- (Let us Focus on This Now)
o Support Vector Regression (SVR)
2. For each model, train it using the training set and predict on the testing set.
```

```
#Model Definition. --
from sklearn.ensemble import GradientBoostingRegressor
gb = GradientBoostingRegressor(random_state=42)
gb.fit(X_train, y_train)
y_pred_gb = gb.predict(X_test)
```

Task 3 -> Gradient Boosting Regression

```
Model Evaluation
1. Evaluate each model using the following metrics:
o Mean Absolute Error (MAE)
o Mean Squared Error (MSE)
o Root Mean Squared Error (RMSE)
o Mean Absolute Percentage Error (MAPE)
o R-squared (R2)
2. Compare the performance of the models based on these metrics and find out which model performs
the best
```

```
# Mean Absolute Error (MAE)
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_test, y_pred_gb)
print(mae)
```

```
2391.8150556402825
```

```
# Mean Squared Error (MSE)
from sklearn.metrics import mean_squared_error
```

```
mse = mean_squared_error(y_test, y_pred_gb)
print(mse)
```

```
18790601.38835806
```

```
# Root Mean Squared Error (RMSE)
import numpy as np
rmse = np.sqrt(mse)
print(rmse)
```

```
4334.812728176162
```

```
# Mean Absolute Percentage Error (MAPE)
from sklearn.metrics import mean_absolute_percentage_error
mape_1 = mean_absolute_percentage_error(y_test, y_pred_gb)
mape_2 = np.mean(np.abs((y_test - y_pred_gb) / y_test)) * 100
print(mape_1)
print(mape_2)
```

```
0.2841914754921504
28.41914754921504
```

```
# R-squared (R2)
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred_gb)
print(r2)
```

```
0.8789645415598533
```

Start coding or generate with AI.

Task 2 -- Support Vector Regression (SVR)

```
Task 2: Implement Regression Models
1. Train the following regression models:
o Linear Regression
o Decision Tree Regression
o Random Forest Regression
o Gradient Boosting Regression
o Support Vector Regression (SVR) <-- (Let us Focus on This Now)
2. For each model, train it using the training set and predict on the testing set.
```

```
#Model Definition. --
from sklearn.svm import SVR
svr = SVR()
svr.fit(X_train, y_train)
y_pred_svr = svr.predict(X_test)
```

Task 3 -> Support Vector Regression (SVR)

```
Model Evaluation
1. Evaluate each model using the following metrics:
o Mean Absolute Error (MAE)
o Mean Squared Error (MSE)
o Root Mean Squared Error (RMSE)
o Mean Absolute Percentage Error (MAPE)
o R-squared (R2)
2. Compare the performance of the models based on these metrics and find out which model performs
the best
```

```
# Mean Absolute Error (MAE)
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_test, y_pred_svr)
print(mae)
```

```
8611.850539715408
```

```
# Mean Squared Error (MSE)
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred_svr)
print(mse)
```

```
166379686.4153107
```

```
# Root Mean Squared Error (RMSE)
import numpy as np
rmse = np.sqrt(mse)
print(rmse)
```

```
12898.82500134453
```

```
# Mean Absolute Percentage Error (MAPE)
from sklearn.metrics import mean_absolute_percentage_error
mape_1 = mean_absolute_percentage_error(y_test, y_pred_svr)
mape_2 = np.mean(np.abs((y_test - y_pred_dt) / y_test)) * 100
print(mape_1)
print(mape_2)
```

```
1.1282866638644895
30.795114659332068
```

```
# R-squared (R2)
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred_svr)
print(r2)
```

```
-0.07169755795477806
```

Start coding or generate with AI.