Class Assignment 1: Predicting Medical Cost

Aphisit Jaemyaem st126130

Importing libraries

In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

1. Load data

In [3]:
```
df = pd.read_csv(r'C:\Users\lenovo\OneDrive\Desktop\master\AT82.03 - ML\insuranc
```

In [4]:
```
df.head()
```

Out[4]:

	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

In [5]:
```
df.shape
```

Out[5]:
```
(1338, 7)
```

In [6]:
```
df.describe()
```

Out[6]:

	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

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1338 non-null   int64  
 1   sex       1338 non-null   object 
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64  
 4   smoker    1338 non-null   object 
 5   region    1338 non-null   object 
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

In [8]:
```
df.columns
```

Out[8]:
```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges'], dtype
='object')
```

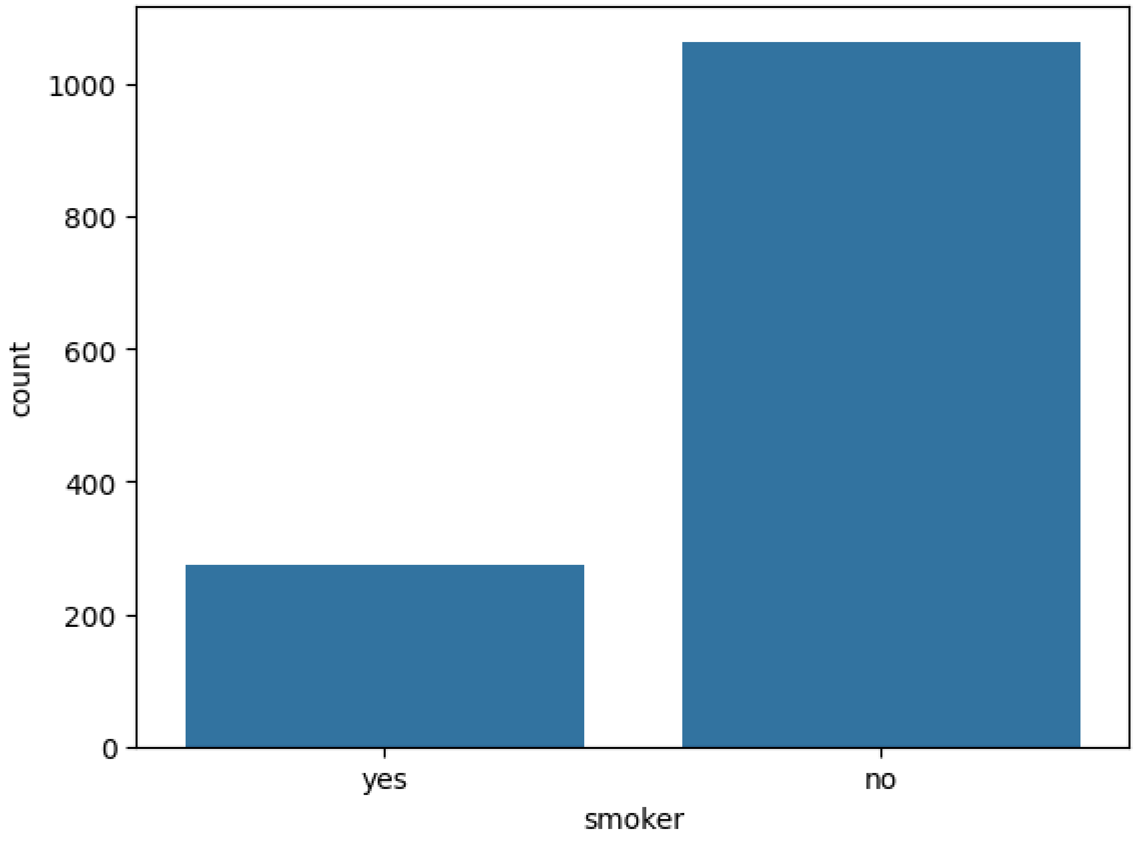
2. Exploratory Data Analysis

2.1 Univariate analyis

Countplot

In [9]:
```
sns.countplot(data = df, x = 'smoker')
```

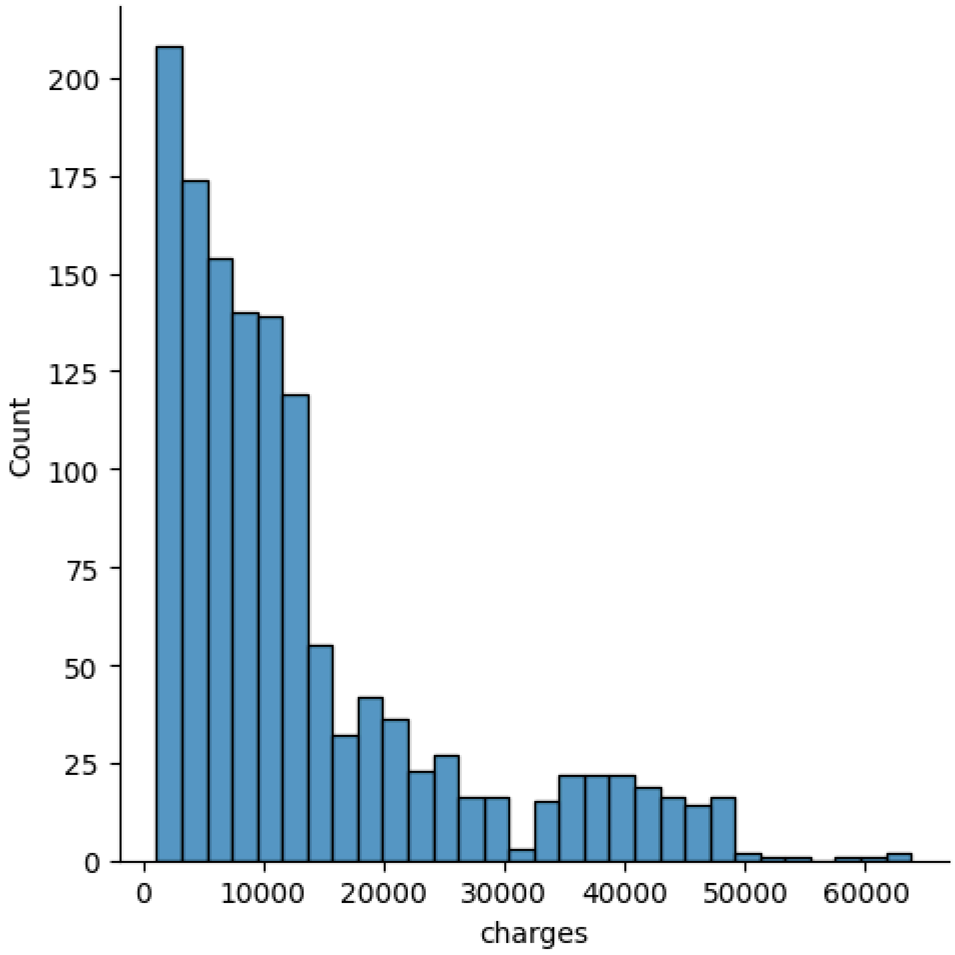
Out[9]:
```
<Axes: xlabel='smoker', ylabel='count'>
```



Distribution plot

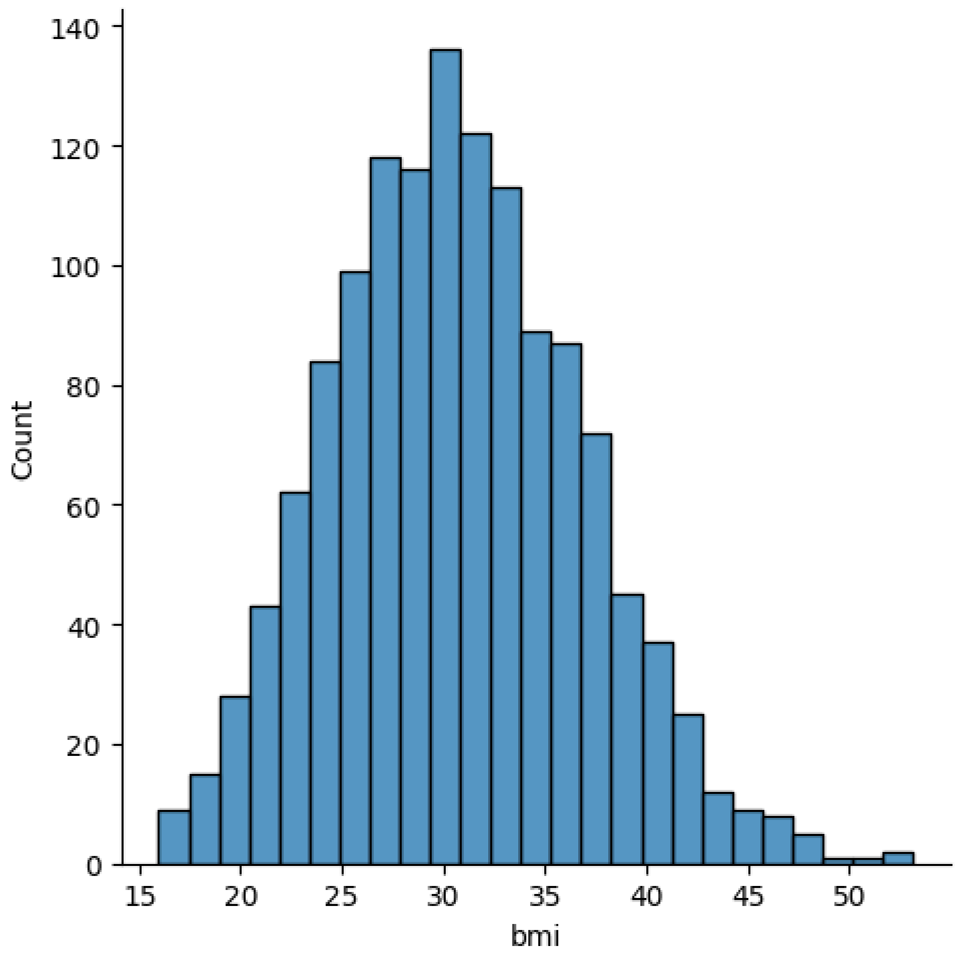
```
In [10]: sns.displot(data = df, x = 'charges')
```

```
Out[10]: <seaborn.axisgrid.FacetGrid at 0x1e4f6d59940>
```



```
In [11]: sns.displot(data = df, x = 'bmi')
```

```
Out[11]: <seaborn.axisgrid.FacetGrid at 0x1e4f9e80e10>
```

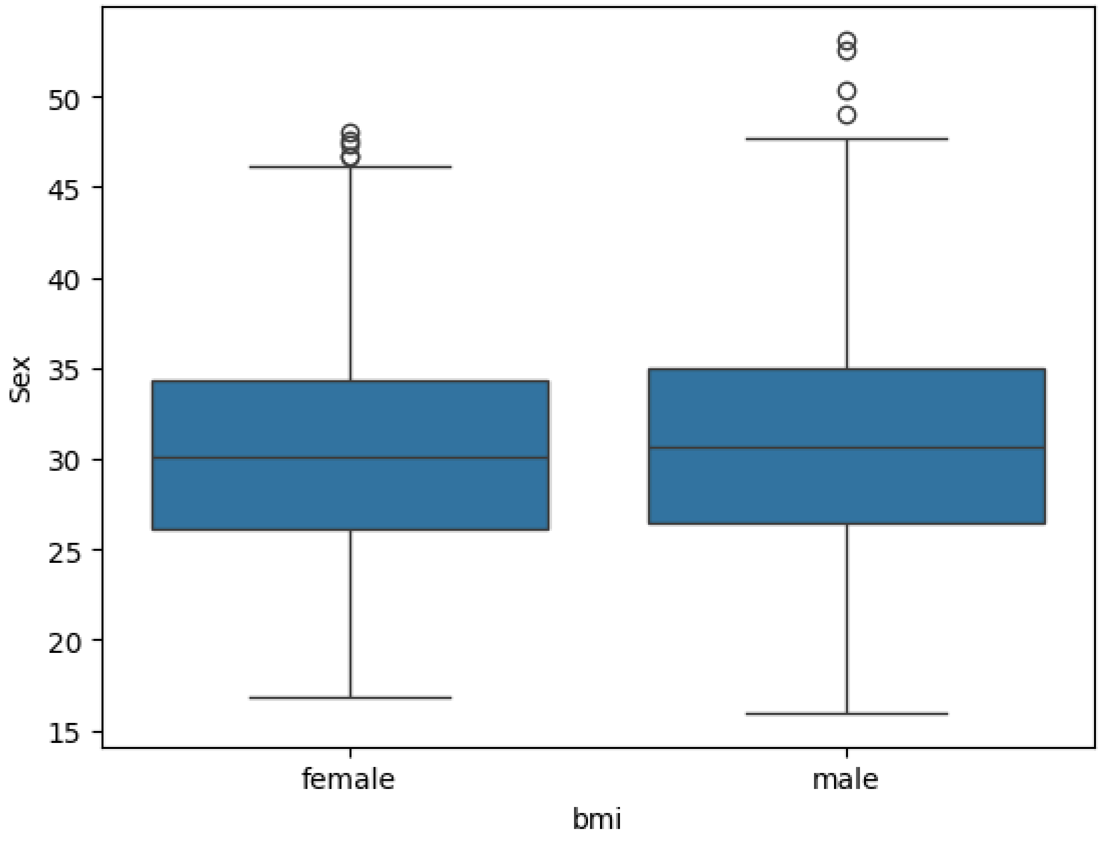


2.2 Multivariate analysis

Boxplot

In [12]:
```
sns.boxplot(x = df["sex"], y = df["bmi"]);
plt.ylabel("Sex")
plt.xlabel("bmi")
```

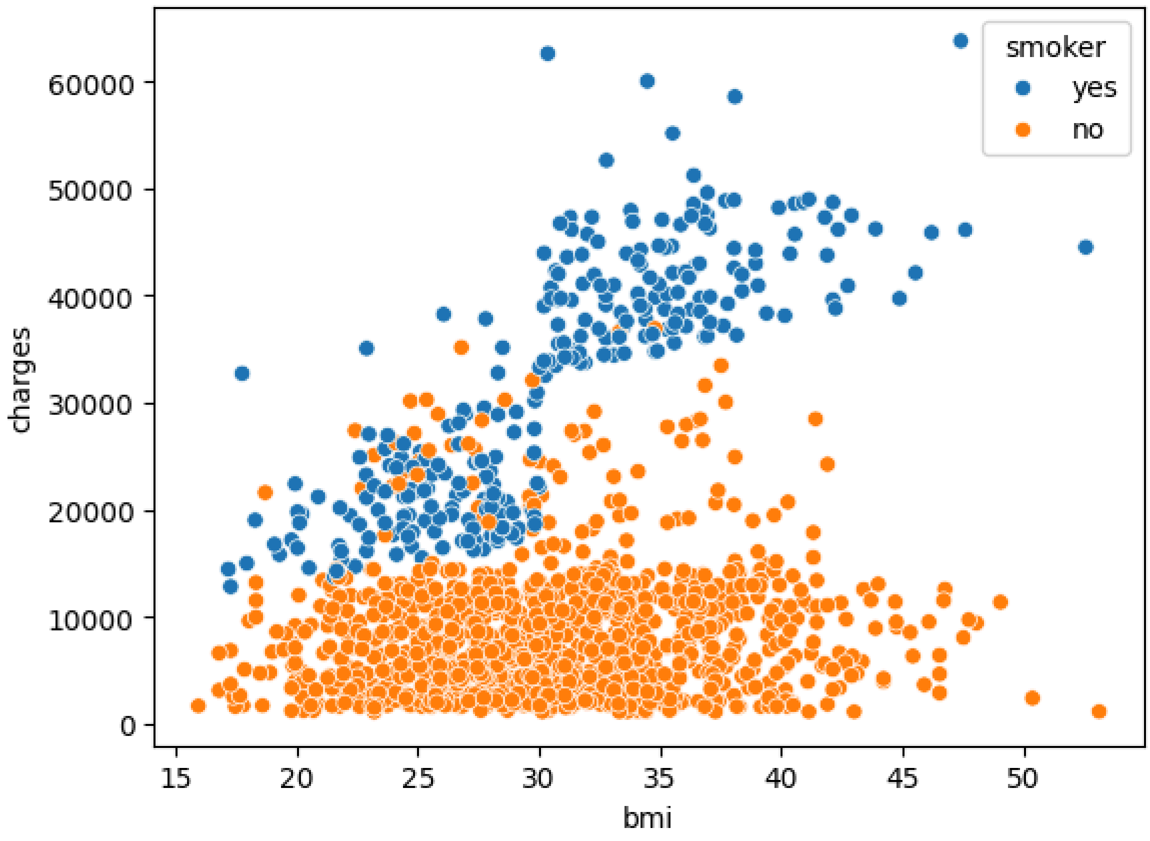
Out[12]:
```
Text(0.5, 0, 'bmi')
```



Scatterplot

```
In [13]: sns.scatterplot(x = df['bmi'], y = df['charges'], hue=df['smoker'])
```

```
Out[13]: <Axes: xlabel='bmi', ylabel='charges'>
```



Correlation Matrix

```
In [14]: df = df.drop('region', axis='columns')
```

```
In [15]: from sklearn.preprocessing import LabelEncoder

         le = LabelEncoder()
         df["sex"] = le.fit_transform(df["sex"])

         df["sex"].unique()
```

```
Out[15]: array([0, 1])
```

```
In [16]: le = LabelEncoder()
         df["smoker"] = le.fit_transform(df["smoker"])

         df["smoker"].unique()
```

```
Out[16]: array([1, 0])
```

```
In [17]: le.classes_
```

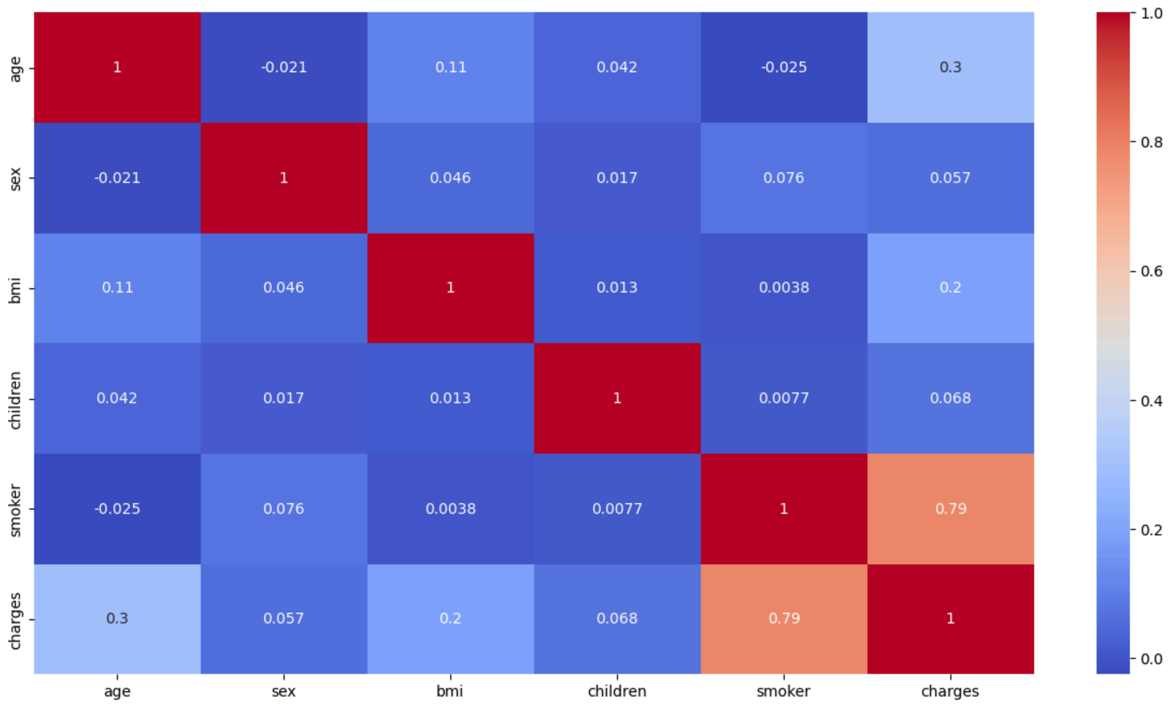
```
Out[17]: array(['no', 'yes'], dtype=object)
```

```
In [18]: le.transform(["no", "yes"])
```

```
Out[18]: array([0, 1])
```

```
In [19]: plt.figure(figsize = (15,8))
         sns.heatmap(df.corr(), annot=True, cmap="coolwarm")
```

```
Out[19]: <Axes: >
```



	age	sex	bmi	children	smoker	charges
age	1	-0.021	0.11	0.042	-0.025	0.3
sex	-0.021	1	0.046	0.017	0.076	0.057
bmi	0.11	0.046	1	0.013	0.0038	0.2
children	0.042	0.017	0.013	1	0.0077	0.068
smoker	-0.025	0.076	0.0038	0.0077	1	0.79
charges	0.3	0.057	0.2	0.068	0.79	1

3. Feature selection

```
In [20]: #x is our strong features
         X = df[  ['smoker', 'bmi','age','children']  ]
```

```
#y is simply the life expectancy col
y = df["charges"]
```

Train test split

In [21]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, randc
```

4. Preprocessing

In [22]:
```
#check for missing values
X_train[['smoker', 'bmi','age','children']].isna().sum()
```

Out[22]:
```
smoker      0
bmi         0
age         0
children    0
dtype: int64
```

In [23]:
```
X_test[['smoker', 'bmi','age','children']].isna().sum()
```

Out[23]:
```
smoker      0
bmi         0
age         0
children    0
dtype: int64
```

In [24]:
```
y_train.isna().sum()
```

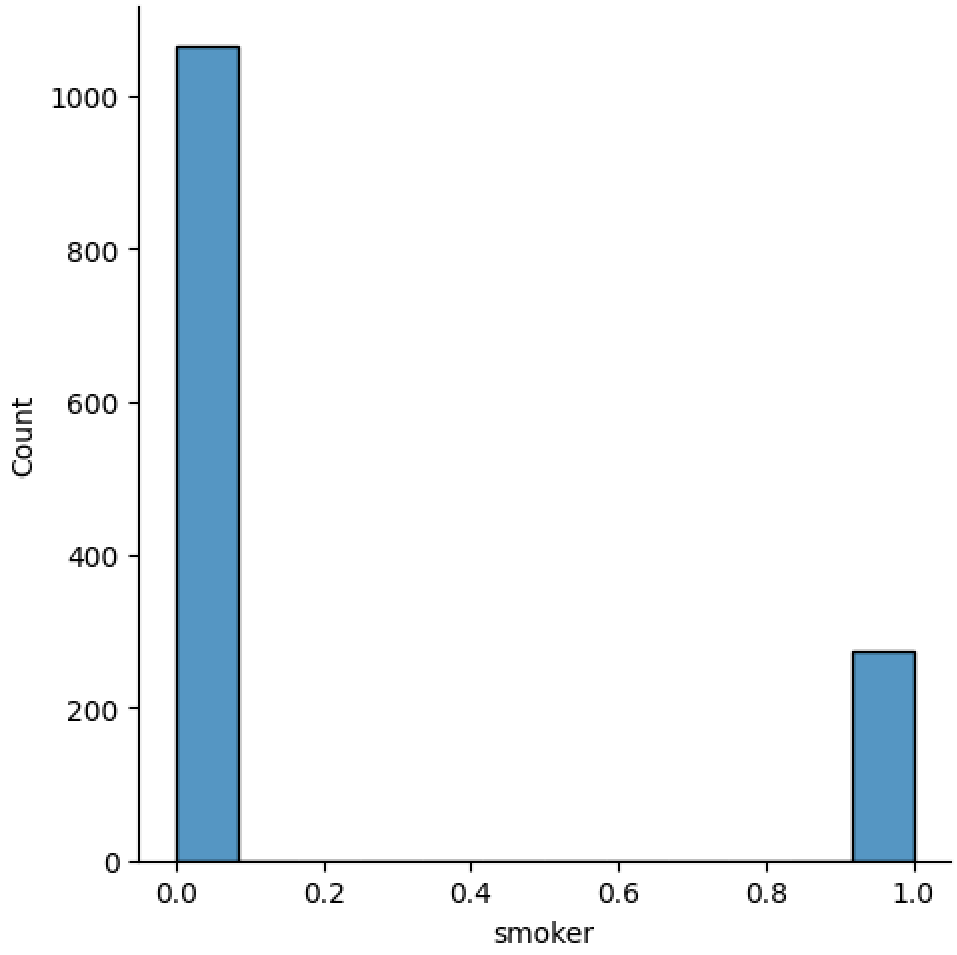
Out[24]:
```
np.int64(0)
```

In [25]:
```
y_test.isna().sum()
```

Out[25]:
```
np.int64(0)
```

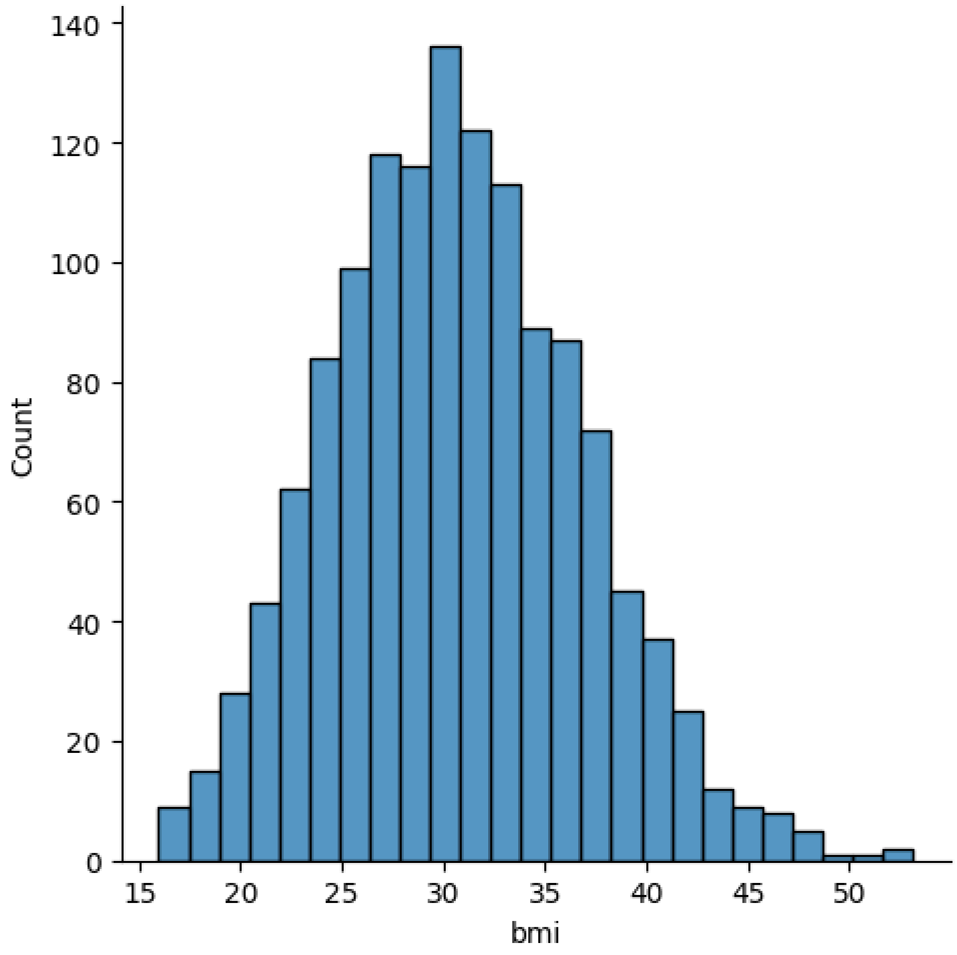
In [26]:
```
sns.displot(data=df, x='smoker')
```

Out[26]:
```
<seaborn.axisgrid.FacetGrid at 0x1e4fa49d090>
```



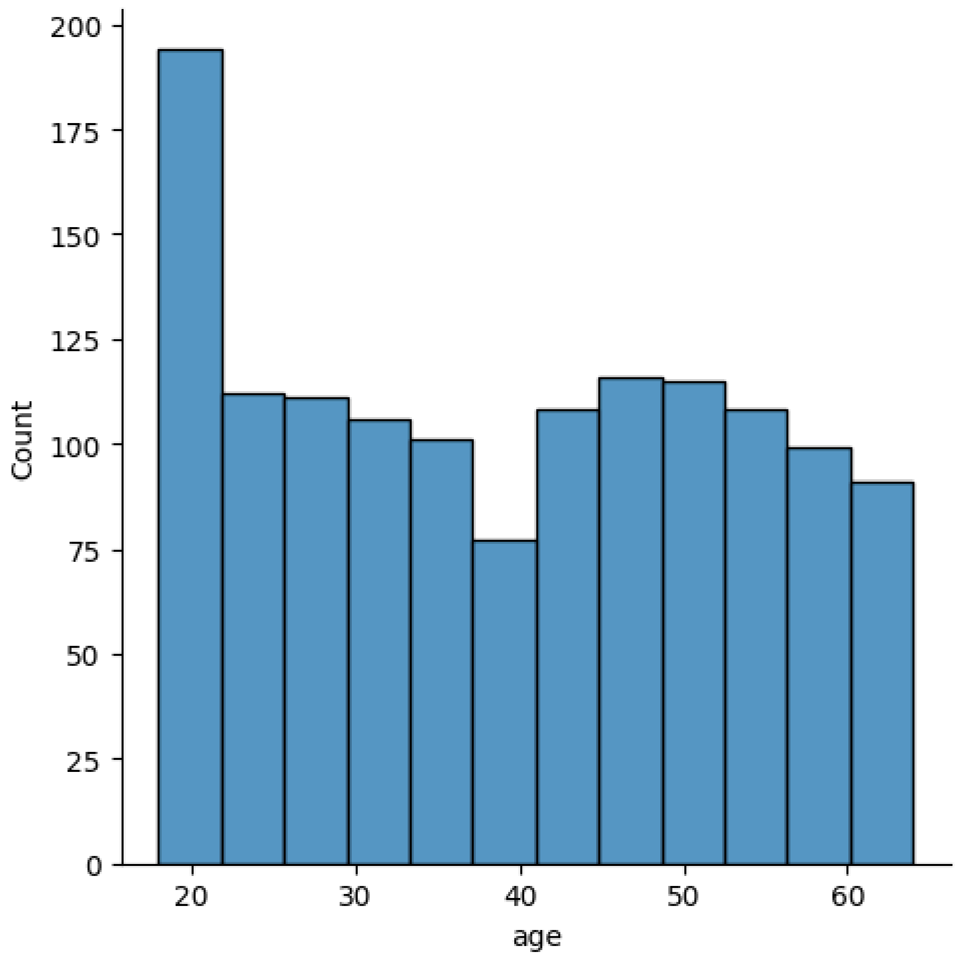
```
In [27]: sns.displot(data=df, x='bmi')
```

```
Out[27]: <seaborn.axisgrid.FacetGrid at 0x1e4fa545090>
```



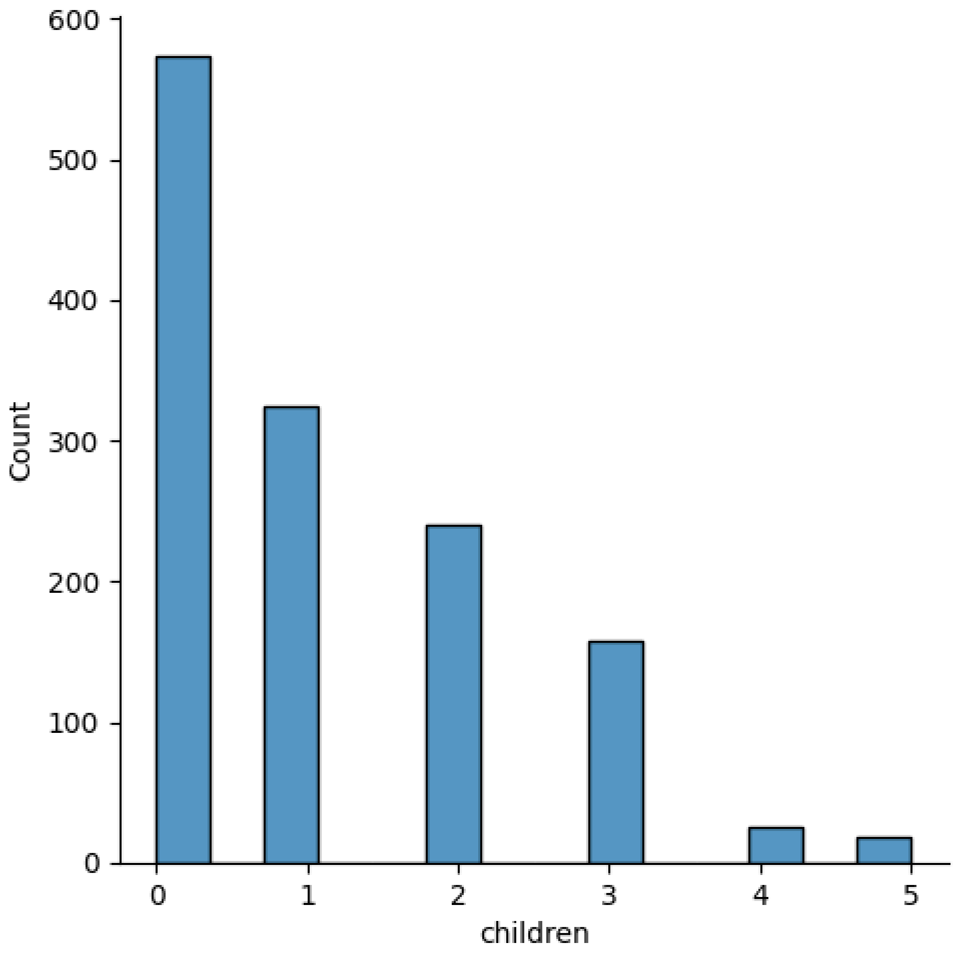
```
In [28]: sns.displot(data=df, x='age')
```

```
Out[28]: <seaborn.axisgrid.FacetGrid at 0x1e4fa49cf50>
```



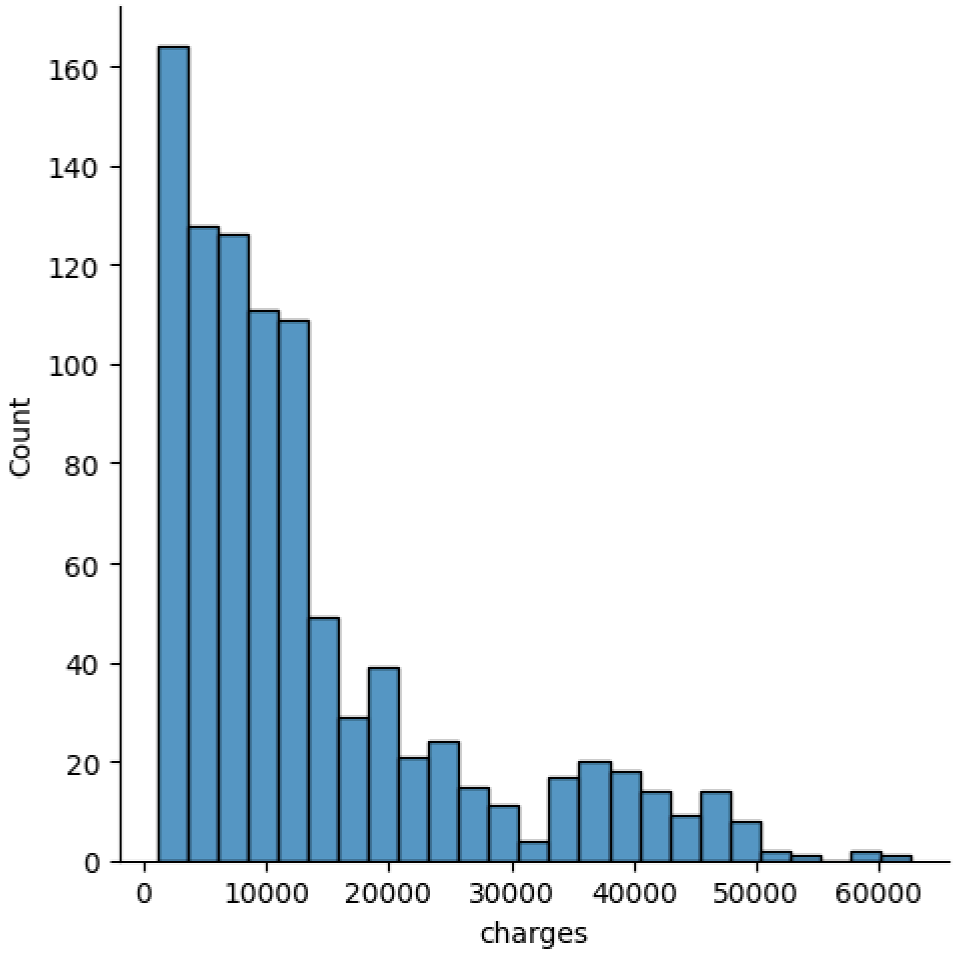
```
In [29]: sns.displot(data=df, x='children')
```

```
Out[29]: <seaborn.axisgrid.FacetGrid at 0x1e4fa693d90>
```



```
In [30]: sns.displot(y_train)

Out[30]: <seaborn.axisgrid.FacetGrid at 0x1e4fa73ad50>
```



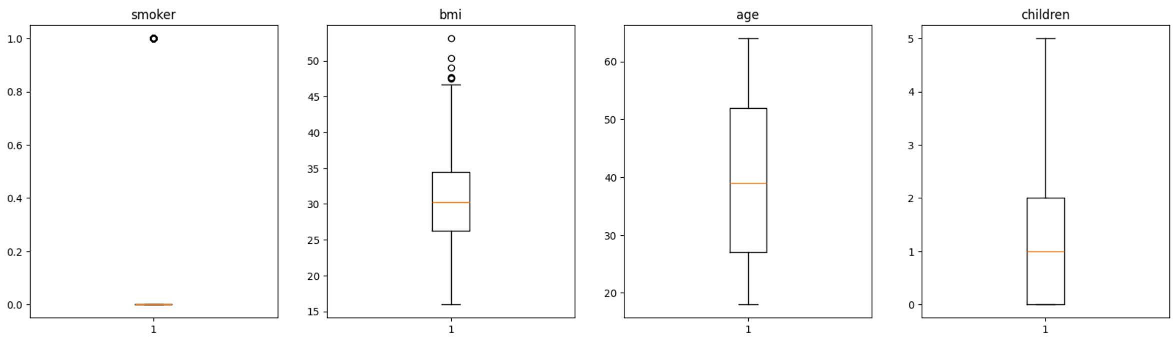
Checking Outliers

In [31]:
```
# Create a dictionary of columns.
col_dict = {'smoker':1,'bmi':2,'age':3,'children':4}

# Detect outliers in each variable using box plots.
plt.figure(figsize=(20,30))

for variable,i in col_dict.items():
                     plt.subplot(5,4,i)
                     plt.boxplot(X_train[variable])
                     plt.title(variable)

plt.show()
```



In [32]:
```
def outlier_count(col, data = X_train):

    # calculate your 25% quatile and 75% quatile
    q75, q25 = np.percentile(data[col], [75, 25])
    # calculate your inter quatile
    iqr = q75 - q25

    # min_val and max_val
    min_val = q25 - (iqr*1.5)
    max_val = q75 + (iqr*1.5)

    # count number of outliers, which are the data that are less than min_val or
    outlier_count = len(np.where((data[col] > max_val) | (data[col] < min_val))[

    # calculate the percentage of the outliers
    outlier_percent = round(outlier_count/len(data[col])*100, 2)

    if(outlier_count > 0):
        print("\n"+15*'-' + col + 15*'-'+"\n")
        print('Number of outliers: {}'.format(outlier_count))
        print('Percent of data that is outlier: {}%'.format(outlier_percent))
```

In [33]:
```
#check number of outliers for each features.
for col in X_train.columns:
    outlier_count(col)
```

```
---------------smoker---------------

Number of outliers: 195
Percent of data that is outlier: 20.83%

---------------bmi---------------

Number of outliers: 6
Percent of data that is outlier: 0.64%
```

Scaling

In [34]:
```
from sklearn.preprocessing import StandardScaler

# feature scaling helps improve reach convergence faster
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test  = scaler.transform(X_test)

#x = (x - mean) / std
#why do we want to scale our data before data analysis / machine learning

#allows your machine learning model to catch the pattern/relationship faster
#faster convergence

#how many ways to scale
#standardization <====current way
# (x - mean) / std
#--> when your data follows normal distribution

#normalization <---another way
# (x - x_min) / (x_max - x_min)
#---> when your data DOES NOT follow normal distribution (e.g., audio, signal, i
```

In [35]:
```
# Let's check shapes of all X_train, X_test, y_train, y_test
print("Shape of X_train: ", X_train.shape)
print("Shape of X_test: ", X_test.shape)
print("Shape of y_train: ", y_train.shape)
print("Shape of y_test: ", y_test.shape)
```

```
Shape of X_train:  (936, 4)
Shape of X_test:  (402, 4)
Shape of y_train:  (936,)
Shape of y_test:  (402,)
```

5. Modeling

In [36]:
```
from sklearn.linear_model import LinearRegression  #we are using regression mode
from sklearn.metrics import mean_squared_error, r2_score

lr = LinearRegression()
lr.fit(X_train, y_train)
yhat = lr.predict(X_test)

print("MSE: ", mean_squared_error(y_test, yhat))
print("r2: ", r2_score(y_test, yhat))
```

```
MSE:  33948860.84184331
r2:  0.7684636242828374
```

Much better: Cross validation + Grid search

In [37]:
```
from sklearn.linear_model import LinearRegression  #we are using regression mode
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

# Libraries for model evaluation

# models that we will be using, put them in a list
algorithms = [LinearRegression(), SVR(), KNeighborsRegressor(), DecisionTreeRegr
              RandomForestRegressor(n_estimators = 100, random_state = 0)]

# The names of the models
algorithm_names = ["Linear Regression", "SVR", "KNeighbors Regressor", "Decision
```

In [38]:
```
y_train.isna().sum()
```

Out[38]:
```
np.int64(0)
```

In [39]:
```
from sklearn.model_selection import KFold, cross_val_score

#lists for keeping mse
train_mse = []
test_mse = []

#defining splits
kfold = KFold(n_splits=5, shuffle=True)

for i, model in enumerate(algorithms):
    scores = cross_val_score(model, X_train, y_train, cv=kfold, scoring='neg_mea
    print(f"{algorithm_names[i]} - Score: {scores}; Mean: {scores.mean()}")
```

```
Linear Regression - Score: [-42562095.49557848 -37777606.57504902 -30876577.12514
288
 -41299793.69332787 -38745269.49680174]; Mean: -38252268.477180004
SVR - Score: [-1.78610093e+08 -1.63733343e+08 -1.78667328e+08 -1.37297308e+08
 -1.45426636e+08]; Mean: -160746941.685659
KNeighbors Regressor - Score: [-27135101.69467305 -33234526.96577911 -23152402.54
630787
 -27855159.4747529  -30917209.12547133]; Mean: -28458879.96139685
Decision-Tree Regressor - Score: [-40751815.48906638 -39789248.7939286  -5001633
5.84024458
 -37379659.10241125 -47255803.14104906]; Mean: -43038572.473339975
Random-Forest Regressor - Score: [-29968848.10708449 -23822962.29035486 -3951493
7.72014673
 -23061431.36408127 -21161835.42355897]; Mean: -27506002.981045265
```

Grid Search

In [40]:
```python
# use Random-Forest Regressor after i run many model for check MSE and r2.
from sklearn.model_selection import GridSearchCV

param_grid = {'bootstrap': [True], 'max_depth': [5, 10, None],
              'n_estimators': [5, 6, 7, 8, 9, 10, 11, 12, 13, 15]}

rf = RandomForestRegressor(random_state = 1)

grid = GridSearchCV(estimator = rf,
                    param_grid = param_grid,
                    cv = kfold,
                    n_jobs = -1,
                    return_train_score=True,
                    refit=True,
                    scoring='neg_mean_squared_error')

# Fit your grid_search
grid.fit(X_train, y_train);
```

6. Testing

In [41]:
```python
yhat=grid.predict(X_test)
print("MSE: ", mean_squared_error(y_test, yhat))
print("r2: ", r2_score(y_test, yhat))
```

```
MSE:  19418867.64283543
r2:  0.8675603798460394
```

7. Analysis: Feature Importance

Algorithm way

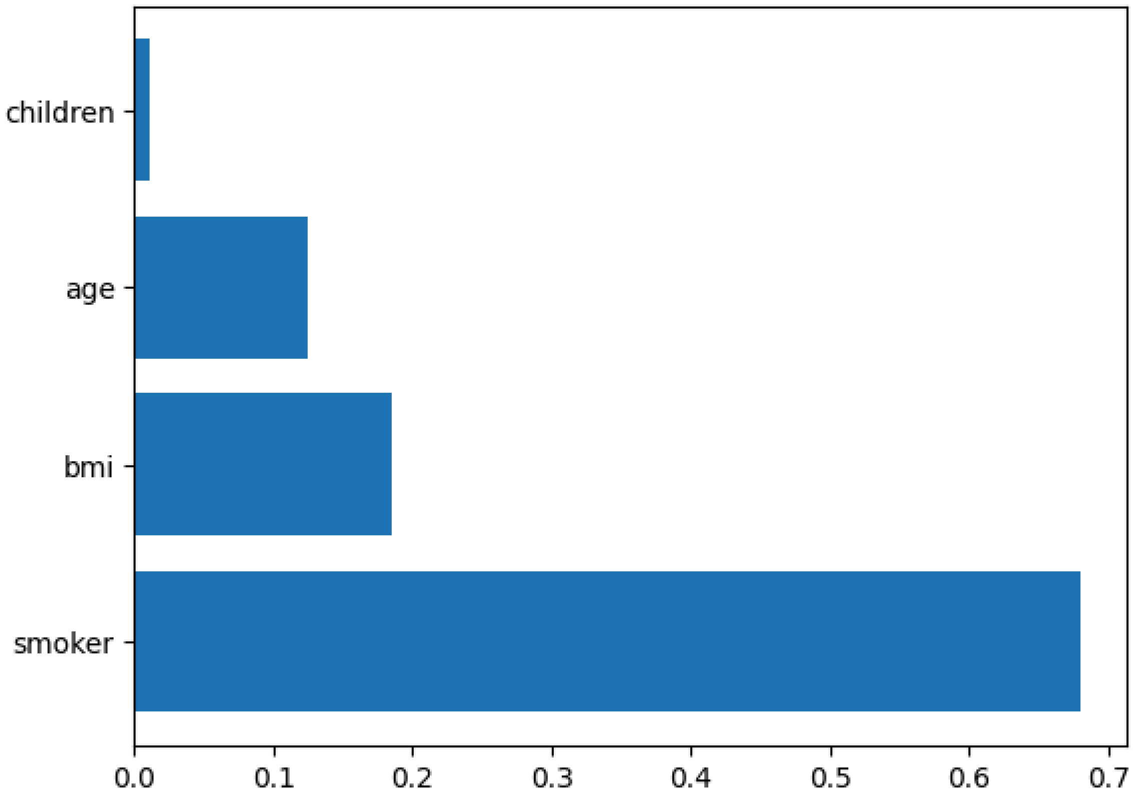
In [42]:
```python
rf = grid.best_estimator_

rf.feature_importances_
```

Out[42]:
```
array([0.68016986, 0.18478469, 0.12420335, 0.01084211])
```

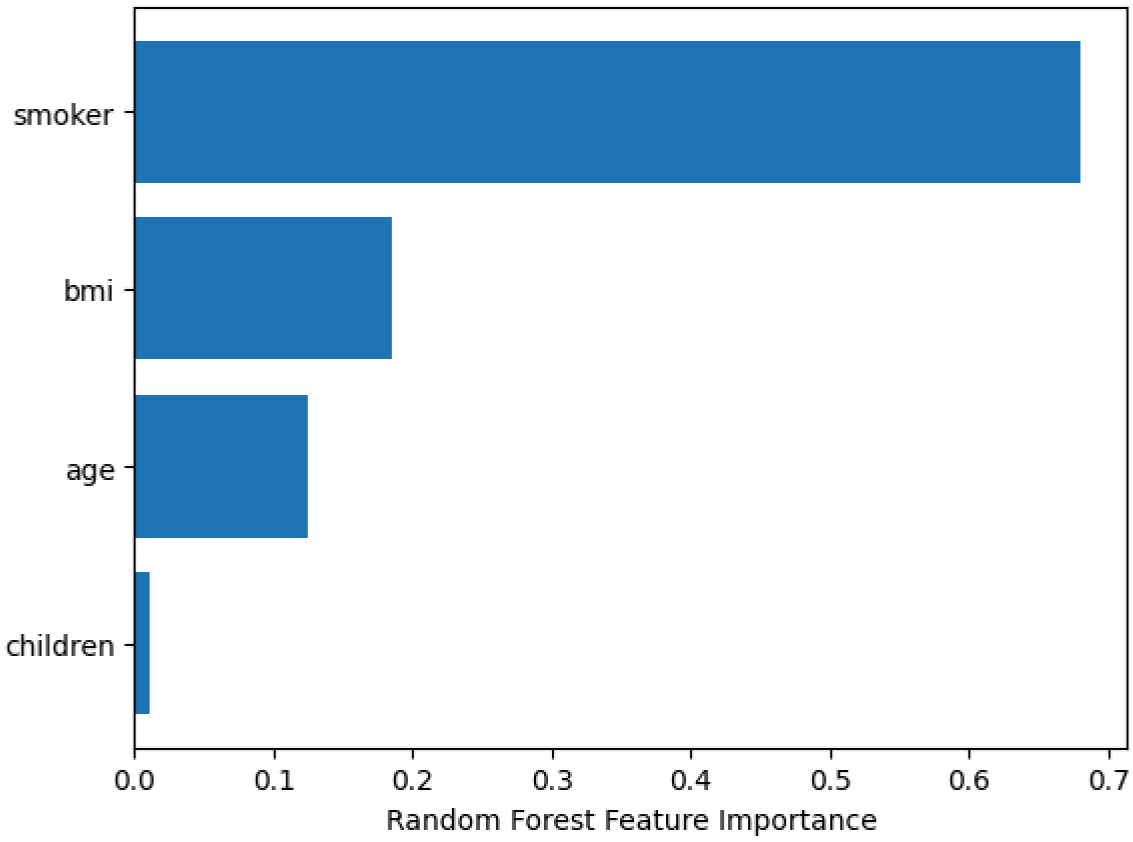
In [43]:
```python
#let's plot
plt.barh(X.columns, rf.feature_importances_)
```

```
Out[43]:  <BarContainer object of 4 artists>
```



```
In [44]:  sorted_idx = rf.feature_importances_.argsort()
          plt.barh(X.columns[sorted_idx], rf.feature_importances_[sorted_idx])
          plt.xlabel("Random Forest Feature Importance")
```

```
Out[44]:  Text(0.5, 0, 'Random Forest Feature Importance')
```



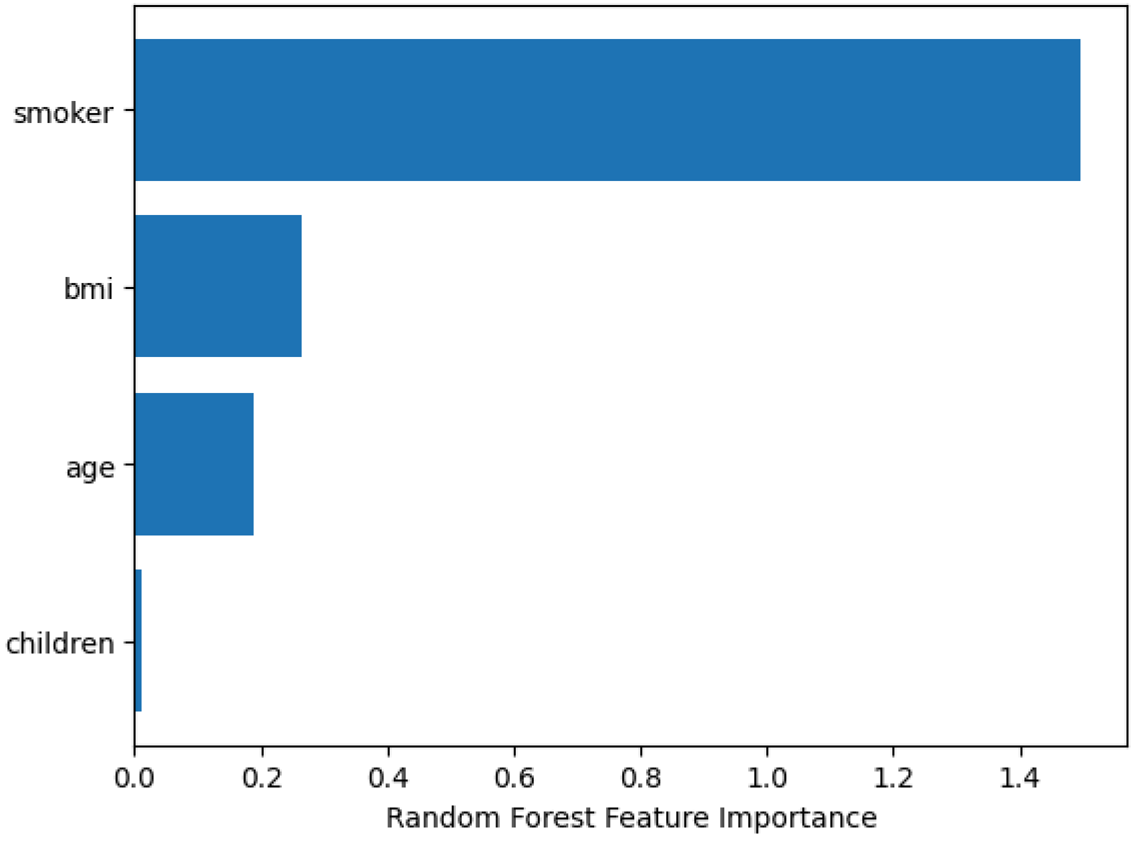
Permutation way

```
In [45]: from sklearn.inspection import permutation_importance

         perm_importance = permutation_importance(rf, X_test, y_test)

         #let's plot
         sorted_idx = perm_importance.importances_mean.argsort()
         plt.barh(X.columns[sorted_idx], perm_importance.importances_mean[sorted_idx])
         plt.xlabel("Random Forest Feature Importance")
```

```
Out[45]: Text(0.5, 0, 'Random Forest Feature Importance')
```

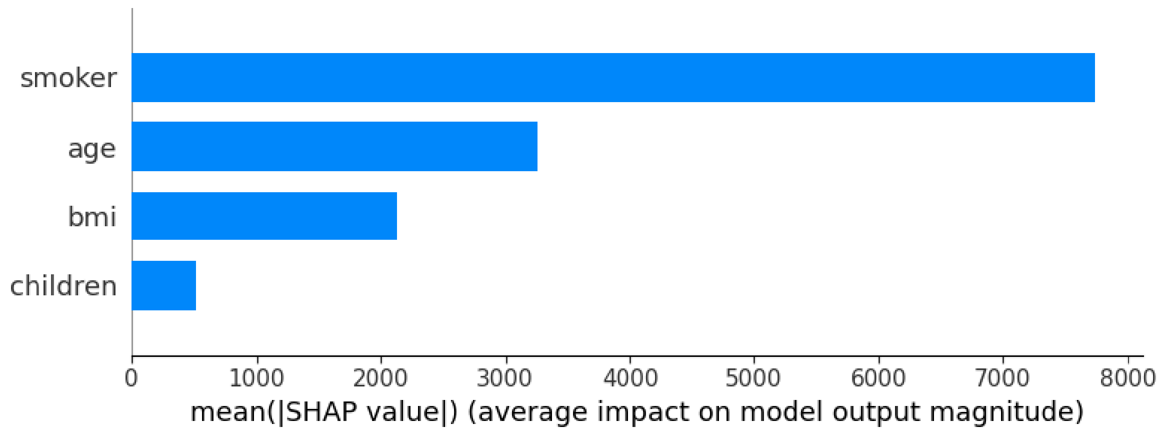


Shap way

```
In [50]: import shap

         explainer = shap.TreeExplainer(rf)
         shap_values = explainer.shap_values(X_test)
```

```
In [51]: shap.summary_plot(shap_values, X_test, plot_type="bar", feature_names = X.column
```



8. Inference

In [54]:
```
import pickle

# save the model to disk
filename = 'model/Medical Cost.model'
pickle.dump(grid, open(filename, 'wb'))
```

In [55]:
```
# load the model from disk
loaded_model = pickle.load(open(filename, 'rb'))
```

In [56]:
```
df[['smoker', 'age', 'bmi','children','charges']].loc[1]
```

Out[56]:
```
smoker         0.0000
age           18.0000
bmi           33.7700
children       1.0000
charges     1725.5523
Name: 1, dtype: float64
```

In [ ]:
```
#create unseen value
sample = np.array([[0,18.0000,33.7700,1.0000]])
```

In [ ]:
```
#Use this model to predict unseen data set
predicted_life_exp = loaded_model.predict(sample)
predicted_life_exp
```

Out[ ]:
```
array([17868.03385212])
```