In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
sns.set_theme(color_codes=True)
```

In [2]:
```
df = pd.read_csv('insurance.csv')
df.head()
```

Out[2]:

	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

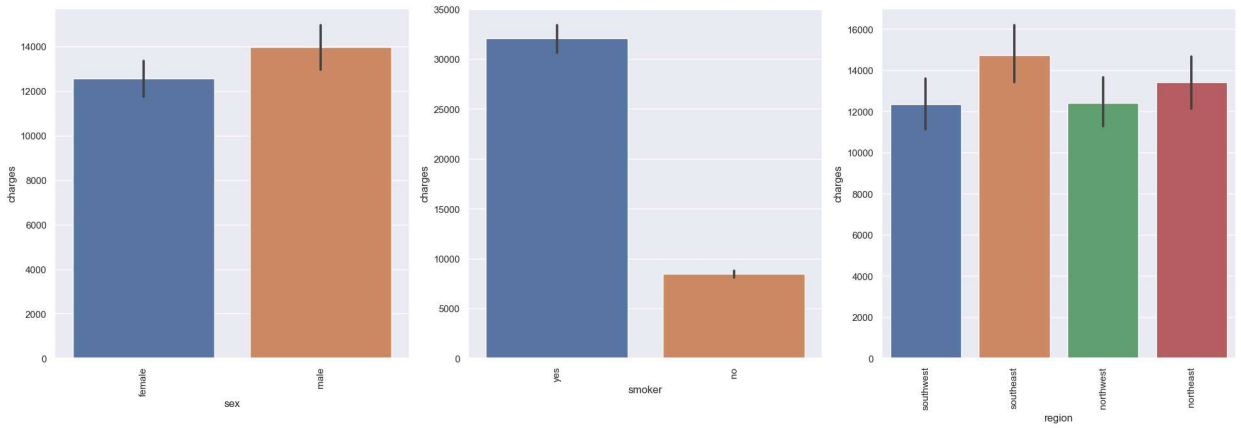
Data Preprocessing Part 1

In [3]:
```
#Cheeck Object data types unique value
df.select_dtypes(include='object').nunique()
```

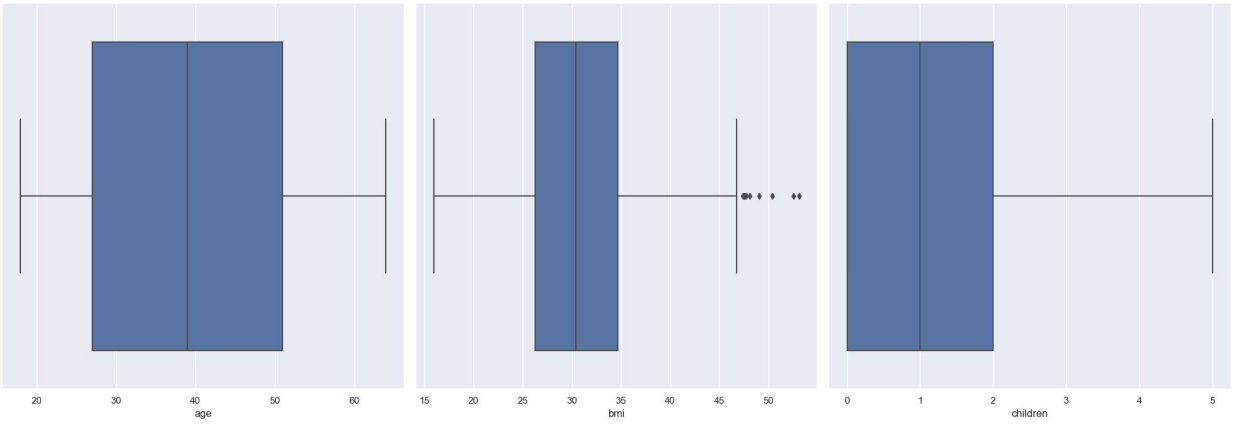
Out[3]:
```
sex       2
smoker    2
region    4
dtype: int64
```

Exploratory Data Analysis

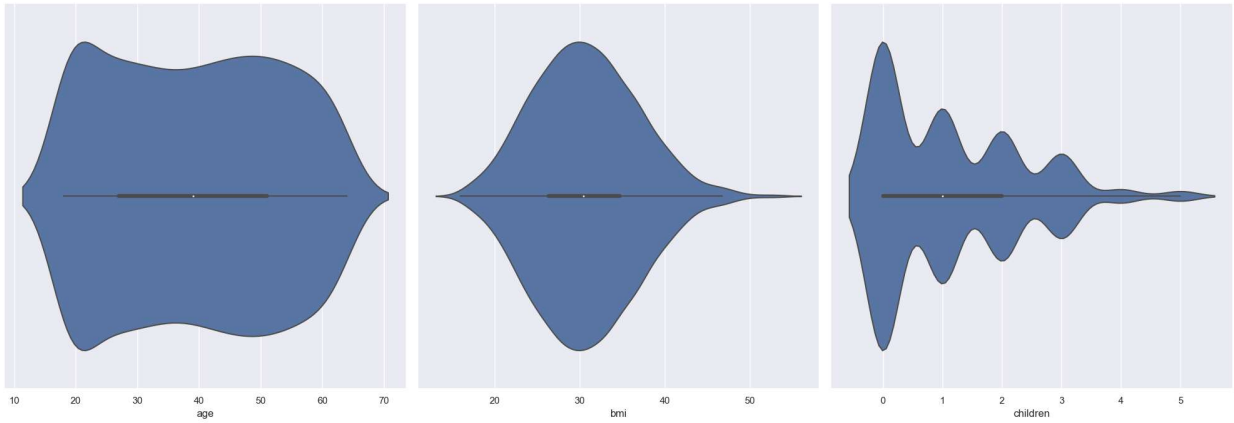
In [5]:
```
# list of categorical variables to plot
cat_vars = ['sex', 'smoker', 'region']

# create figure with subplots
fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 7))
axs = axs.flatten()

# create barplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.barplot(x=var, y='charges', data=df, ax=axs[i])
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```



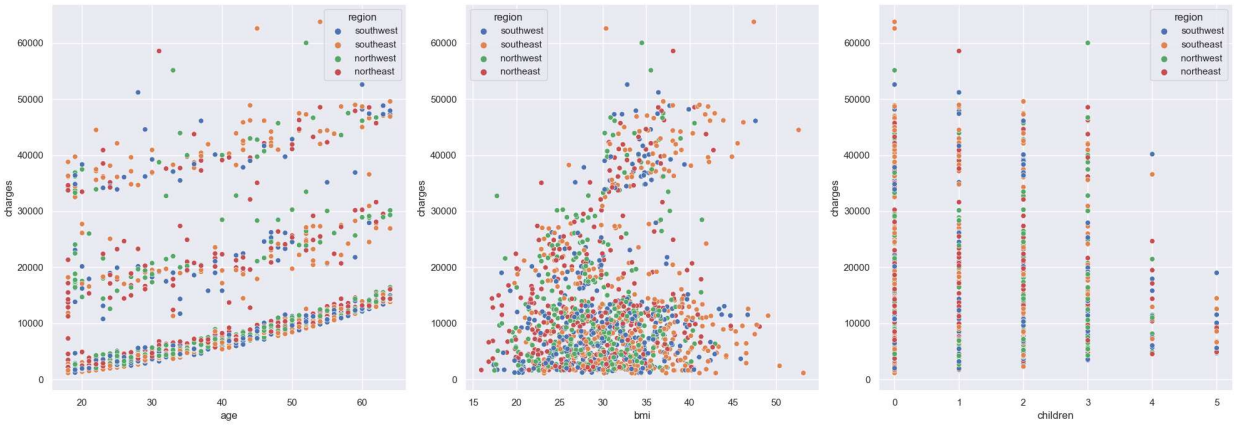
In [8]:
```
# list of numerical variables to plot
num_vars = ['age', 'bmi', 'children']

# create figure with subplots
fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 7))
axs = axs.flatten()

# create violinplot for each numerical variable
for i, var in enumerate(num_vars):
    sns.boxplot(x=var, data=df, ax=axs[i])

# adjust spacing between subplots
fig.tight_layout()

plt.show()
```



In [9]:
```
# list of numerical variables to plot
num_vars = ['age', 'bmi', 'children']

# create figure with subplots
fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 7))
axs = axs.flatten()

# create violinplot for each numerical variable
for i, var in enumerate(num_vars):
    sns.violinplot(x=var, data=df, ax=axs[i])

# adjust spacing between subplots
fig.tight_layout()

plt.show()
```

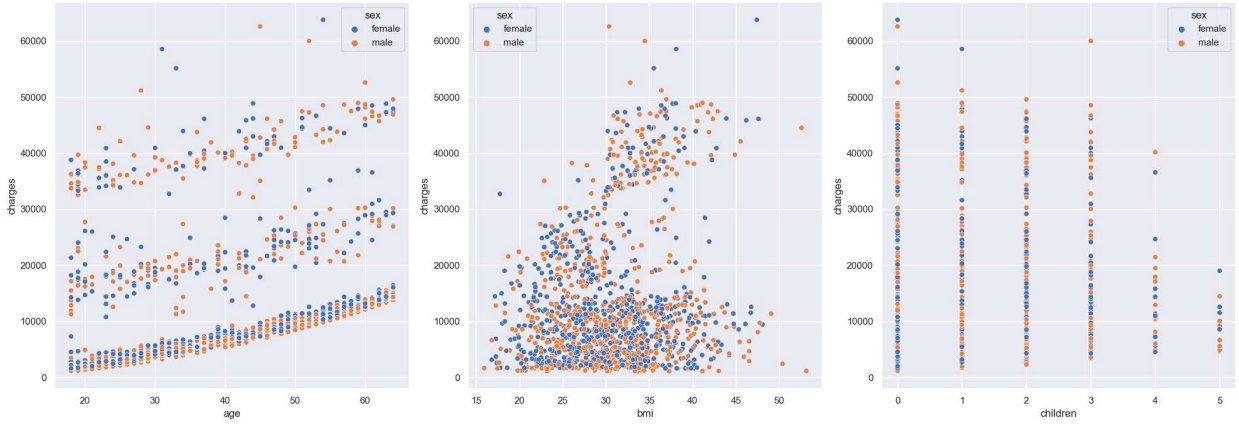


In [10]:
```
num_vars = ['age', 'bmi', 'children']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 7))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.scatterplot(x=var, y='charges', hue='region', data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



In [11]:
```
num_vars = ['age', 'bmi', 'children']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 7))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.scatterplot(x=var, y='charges', hue='smoker', data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```

In [12]:
```
num_vars = ['age', 'bmi', 'children']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 7))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.scatterplot(x=var, y='charges', hue='sex', data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



Data Preprocessing Part 2

In [13]:
```
df.head()
```

Out[13]:

	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 [14]:
```
df.shape
```

Out[14]:
```
(1338, 7)
```

In [15]:
```
#Check missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[15]:
```
Series([], dtype: float64)
```

Label Encoding each Object datatypes

In [16]:
```
# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Print the column name and the unique values
    print(f"{col}: {df[col].unique()}")
```

```
sex: ['female' 'male']
smoker: ['yes' 'no']
region: ['southwest' 'southeast' 'northwest' 'northeast']
```

In [17]:
```
from sklearn import preprocessing

# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Initialize a LabelEncoder object
    label_encoder = preprocessing.LabelEncoder()

    # Fit the encoder to the unique values in the column
    label_encoder.fit(df[col].unique())

    # Transform the column using the encoder
    df[col] = label_encoder.transform(df[col])

    # Print the column name and the unique encoded values
    print(f"{col}: {df[col].unique()}")
```

```
sex: [0 1]
smoker: [1 0]
region: [3 2 1 0]
```

In [18]:
```
df.dtypes
```

Out[18]:
```
age           int64
sex           int32
bmi         float64
children      int64
smoker        int32
region        int32
charges     float64
dtype: object
```

Remove Outliers using Z-Score

In [19]:
```
from scipy import stats

# define a function to remove outliers using z-score for only selected numerical colu
def remove_outliers(df, cols, threshold=3):
    # loop over each selected column
    for col in cols:
        # calculate z-score for each data point in selected column
        z = np.abs(stats.zscore(df[col]))
        # remove rows with z-score greater than threshold in selected column
        df = df[(z < threshold) | (df[col].isnull())]
    return df
```

In [20]:
```
selected_cols = ['bmi']
df_clean = remove_outliers(df, selected_cols)
df_clean.shape
```

Out[20]:
```
(1334, 7)
```

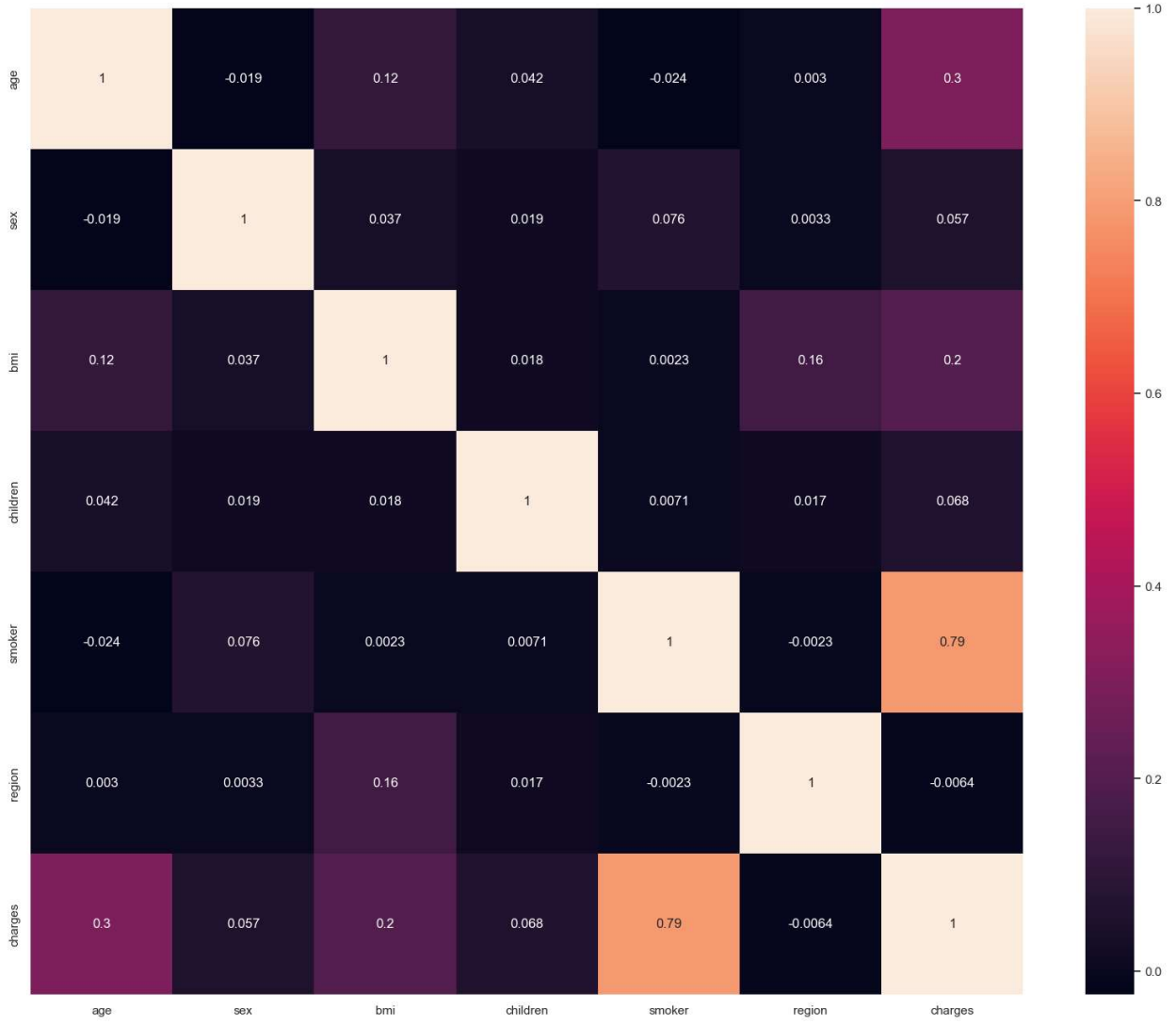
In [21]:
```
#dataframe before the outlier removed
df.shape
```

Out[21]:
```
(1338, 7)
```

Correlation Heatmap

In [22]:
```
#Correlation Heatmap
plt.figure(figsize=(20, 16))
sns.heatmap(df_clean.corr(), fmt='.2g', annot=True)
```

Out[22]: <AxesSubplot:>



Train Test Split

In [23]:
```
X = df_clean.drop('charges', axis=1)
y = df_clean['charges']
```

In [24]:
```
#test size 20% and train size 80%
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,random_state=0
```

Decision Tree Regressor

In [25]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_boston


# Create a DecisionTreeRegressor object
dtree = DecisionTreeRegressor()

# Define the hyperparameters to tune and their values
param_grid = {
    'max_depth': [2, 4, 6, 8],
    'min_samples_split': [2, 4, 6, 8],
    'min_samples_leaf': [1, 2, 3, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

# Create a GridSearchCV object
grid_search = GridSearchCV(dtree, param_grid, cv=5, scoring='neg_mean_squared_error')

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'max_depth': 4, 'max_features': 'auto', 'min_samples_leaf': 4, 'min_samples_split':
2}
```

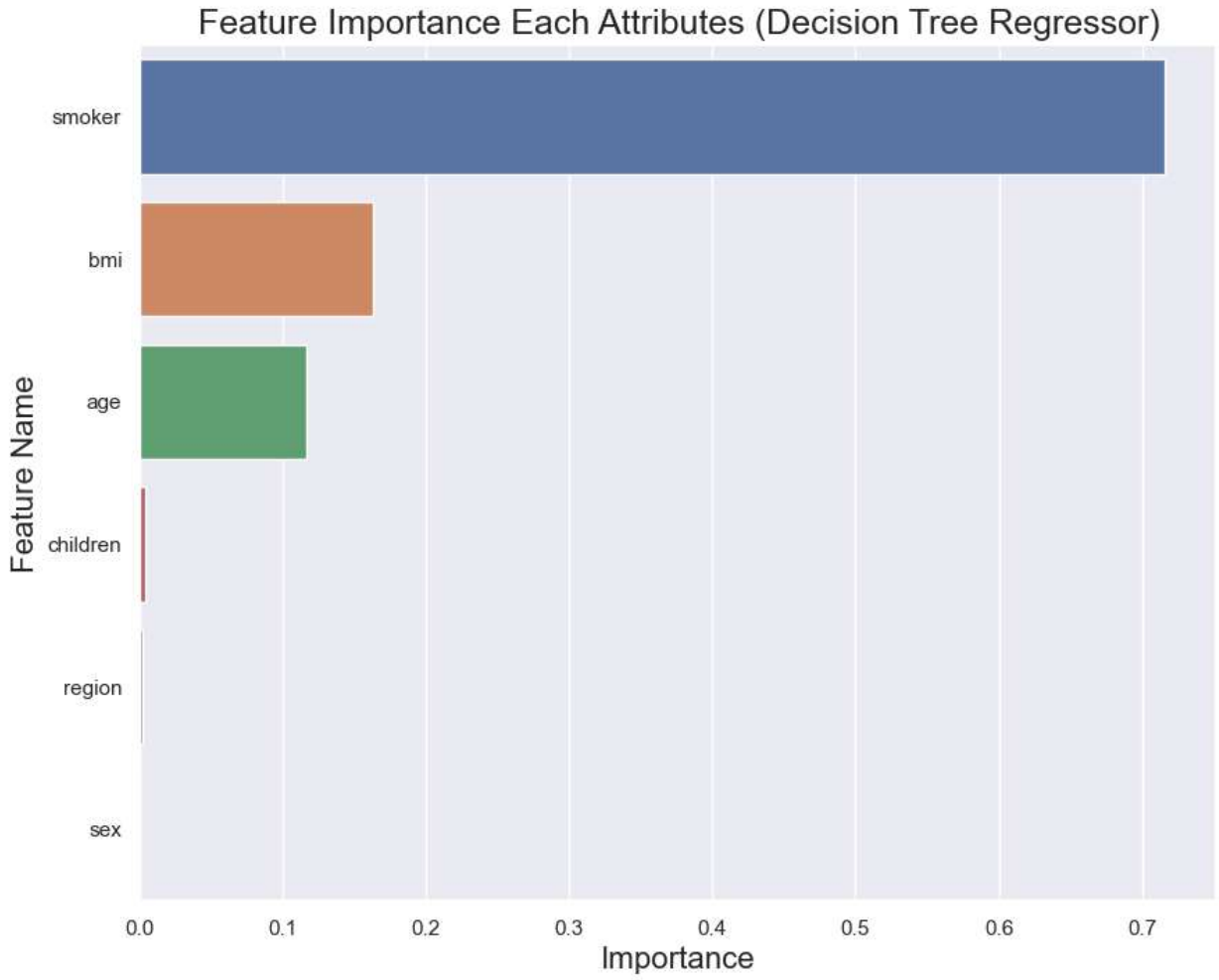
In [26]:
```
from sklearn.tree import DecisionTreeRegressor
dtree = DecisionTreeRegressor(random_state=0, max_depth=4, max_features='auto', min_s
dtree.fit(X_train, y_train)
```

Out[26]:
```
DecisionTreeRegressor(max_depth=4, max_features='auto', min_samples_leaf=4,
                      random_state=0)
```

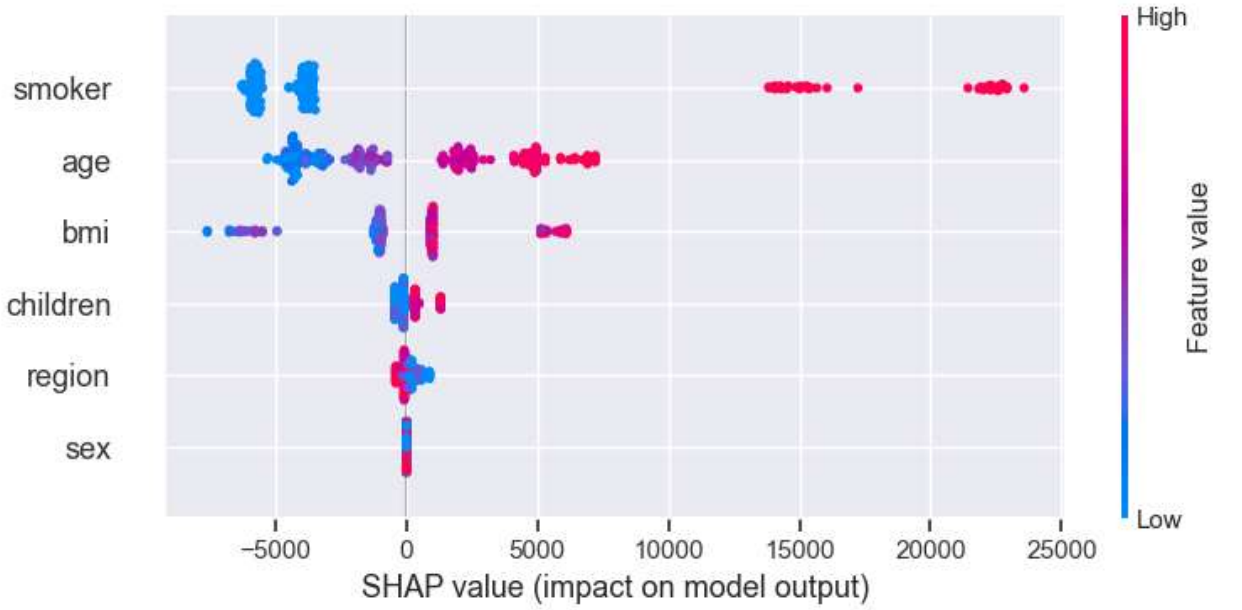
In [27]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = dtree.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, y_pred)
rmse = math.sqrt(mse)

print('MAE is {}'.format(mae))
print('MAPE is {}'.format(mape))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))
print('RMSE score is {}'.format(rmse))
```

```
MAE is 2627.77861129252
MAPE is 0.31327836581317675
MSE is 21611660.168950517
R2 score is 0.8510144440017449
RMSE score is 4648.834280650421
```

In [28]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Feature Importance Each Attributes (Decision Tree Regressor)', fontsize=18
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



In [29]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [30]:
```
explainer = shap.Explainer(dtree, X_test)
shap_values = explainer(X_test)
shap.plots.waterfall(shap_values[0])
```

Random Forest Regressor

In [31]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Create a Random Forest Regressor object
rf = RandomForestRegressor()

# Define the hyperparameter grid
param_grid = {
    'max_depth': [3, 5, 7, 9],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt']
}

# Create a GridSearchCV object
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='r2')

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best hyperparameters: ", grid_search.best_params_)
```

```
Best hyperparameters:  {'max_depth': 5, 'max_features': 'auto', 'min_samples_leaf':
4, 'min_samples_split': 10}
```

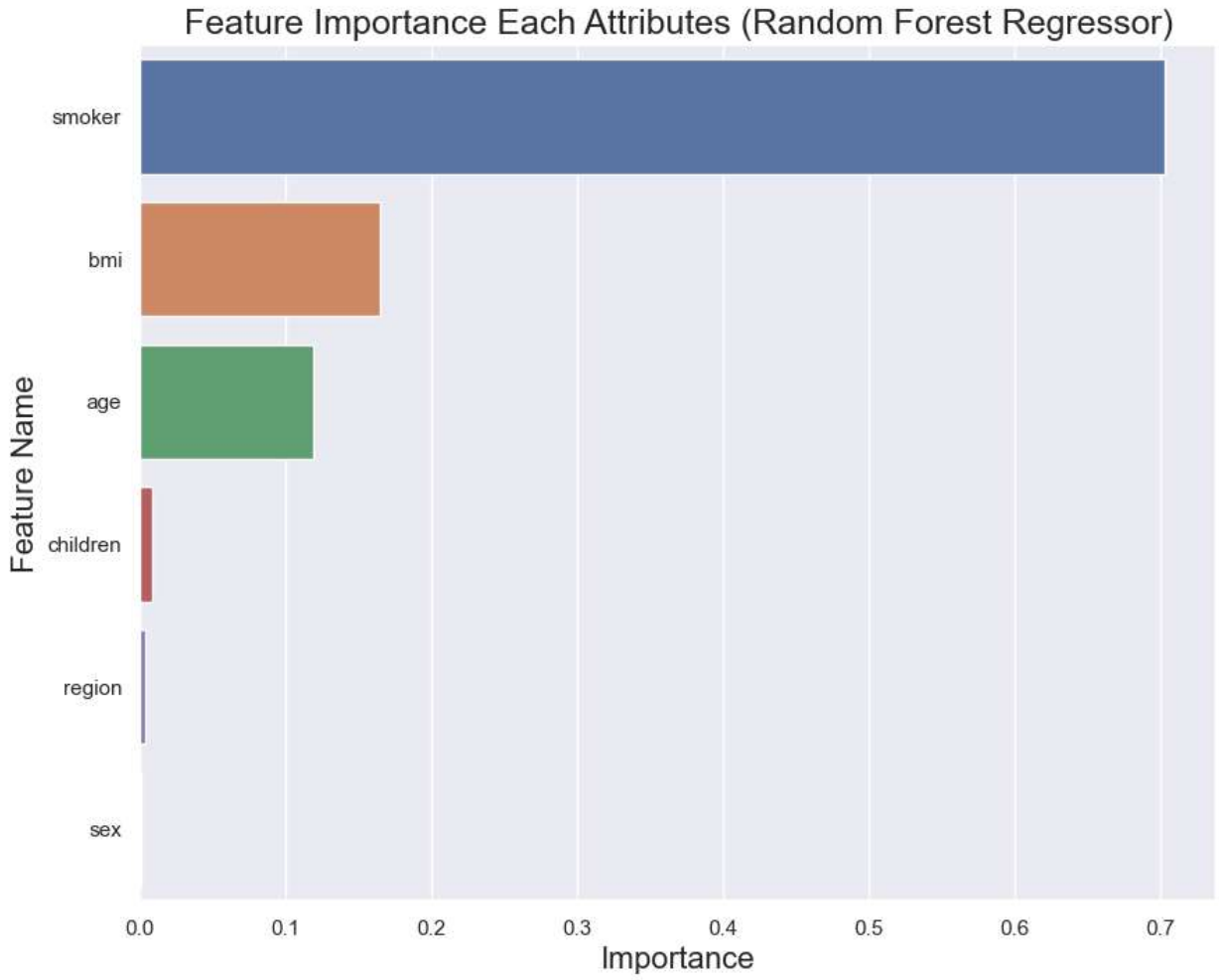
In [32]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state=0, max_depth=5, min_samples_split=10, min_sam
                           max_features='auto')
rf.fit(X_train, y_train)
```

Out[32]:
```
RandomForestRegressor(max_depth=5, min_samples_leaf=4, min_samples_split=10,
                      random_state=0)
```

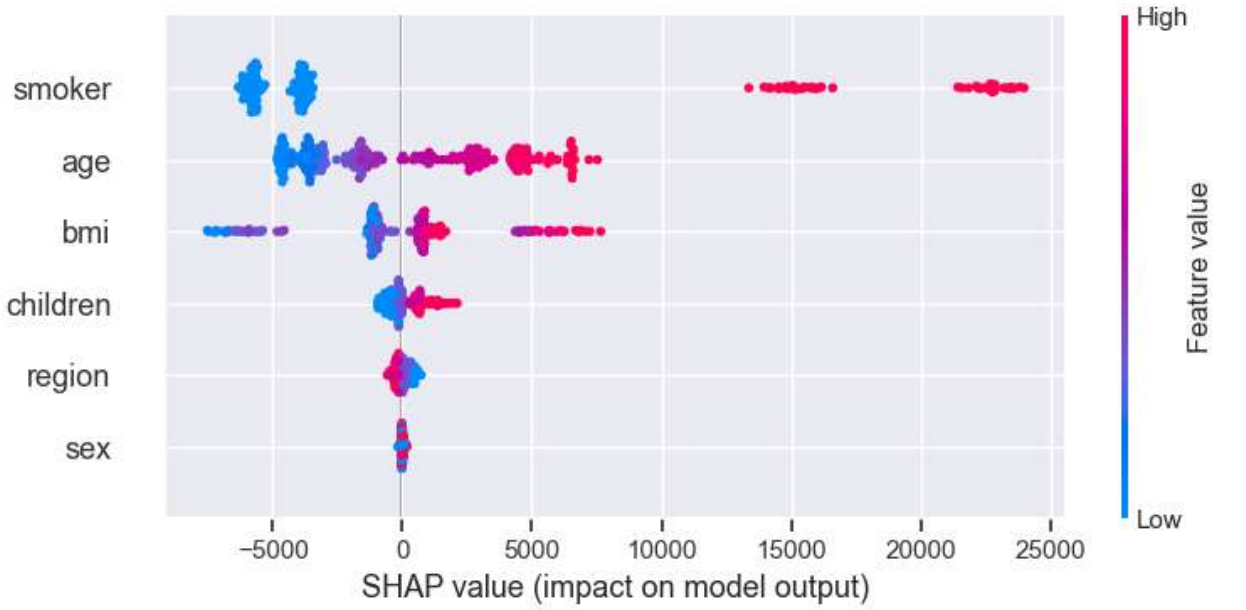
In [33]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = rf.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, y_pred)
rmse = math.sqrt(mse)

print('MAE is {}'.format(mae))
print('MAPE is {}'.format(mape))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))
print('RMSE score is {}'.format(rmse))
```

```
MAE is 2439.7944683926576
MAPE is 0.280797784076888
MSE is 19747580.91254748
R2 score is 0.8638649553585273
RMSE score is 4443.8250317207
```

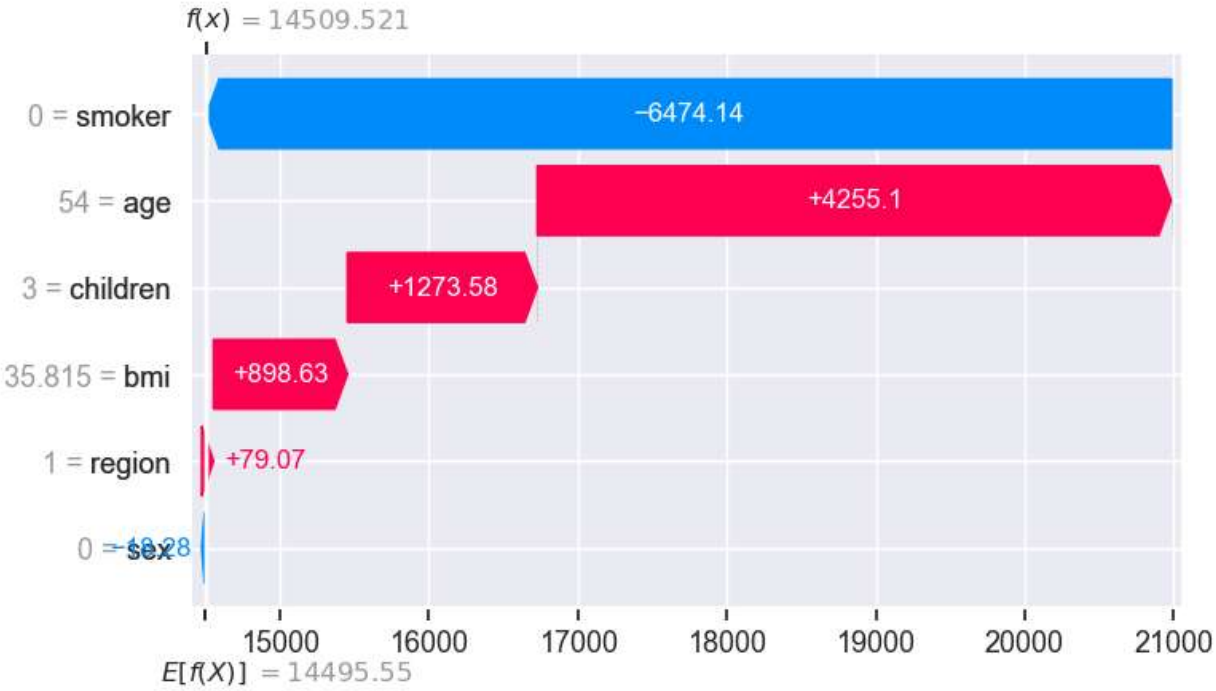
In [35]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": rf.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Feature Importance Each Attributes (Random Forest Regressor)', fontsize=18
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



In [36]:
```
import shap
explainer = shap.TreeExplainer(rf)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [37]:
```
explainer = shap.Explainer(rf, X_test, check_additivity=False)
shap_values = explainer(X_test, check_additivity=False)
shap.plots.waterfall(shap_values[0])
```



AdaBoost Regressor

In [38]:
```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.model_selection import GridSearchCV

# Define AdaBoostRegressor model
abr = AdaBoostRegressor()

# Define hyperparameters and possible values
params = {'n_estimators': [50, 100, 150],
          'learning_rate': [0.01, 0.1, 1, 10]}

# Perform GridSearchCV with 5-fold cross validation
grid_search = GridSearchCV(abr, param_grid=params, cv=5, scoring='neg_mean_squared_er
grid_search.fit(X_train, y_train)

# Print best hyperparameters and corresponding score
print("Best hyperparameters: ", grid_search.best_params_)
```

```
Best hyperparameters:  {'learning_rate': 0.01, 'n_estimators': 50}
```

In [40]:
```
from sklearn.ensemble import RandomForestRegressor
abr = AdaBoostRegressor(random_state=0, learning_rate=0.01, n_estimators=50)
abr.fit(X_train, y_train)
```

Out[40]:
```
AdaBoostRegressor(learning_rate=0.01, random_state=0)
```

In [41]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = abr.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, y_pred)
rmse = math.sqrt(mse)

print('MAE is {}'.format(mae))
print('MAPE is {}'.format(mape))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))
print('RMSE score is {}'.format(rmse))
```

```
MAE is 2864.7990524044253
MAPE is 0.38899081549024944
MSE is 21532890.049069017
R2 score is 0.8515574661488103
RMSE score is 4640.354517606281
```

In [42]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": abr.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Feature Importance Each Attributes (AdaBoost Regressor)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```