Predicting Insurance Premiums

- Our simple dataset contains a few attributes for each person such as
- Age, Sex, BMI, Children, Smoker, Region and their charges

Aim

- To use this info to predict charges for new customers

In [2]:
```
import pandas as pd
file_name = "insurance.csv"
insurance = pd.read_csv(file_name) #you can use the data uploaded in the same folde
# or not

# Preview our data
insurance.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

In [3]:
```
insurance.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 [4]:
```
insurance.describe()
```

Out[4]:

	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 [5]:

```
print ("Rows     : " , insurance.shape[0])
print ("Columns  : " , insurance.shape[1])
print ("\nFeatures : \n" , insurance.columns.tolist())
print ("\nMissing values :  ", insurance.isnull().sum().values.sum())
print ("\nUnique values :  \n",insurance.nunique())
```

```
Rows     :  1338
Columns  :  7

Features :
 ['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges']

Missing values :   0

Unique values :
 age           47
sex            2
bmi          548
children       6
smoker         2
region         4
charges     1337
dtype: int64
```

In [29]:

```
insurance.corr(numeric_only = True)
```

Out[29]:

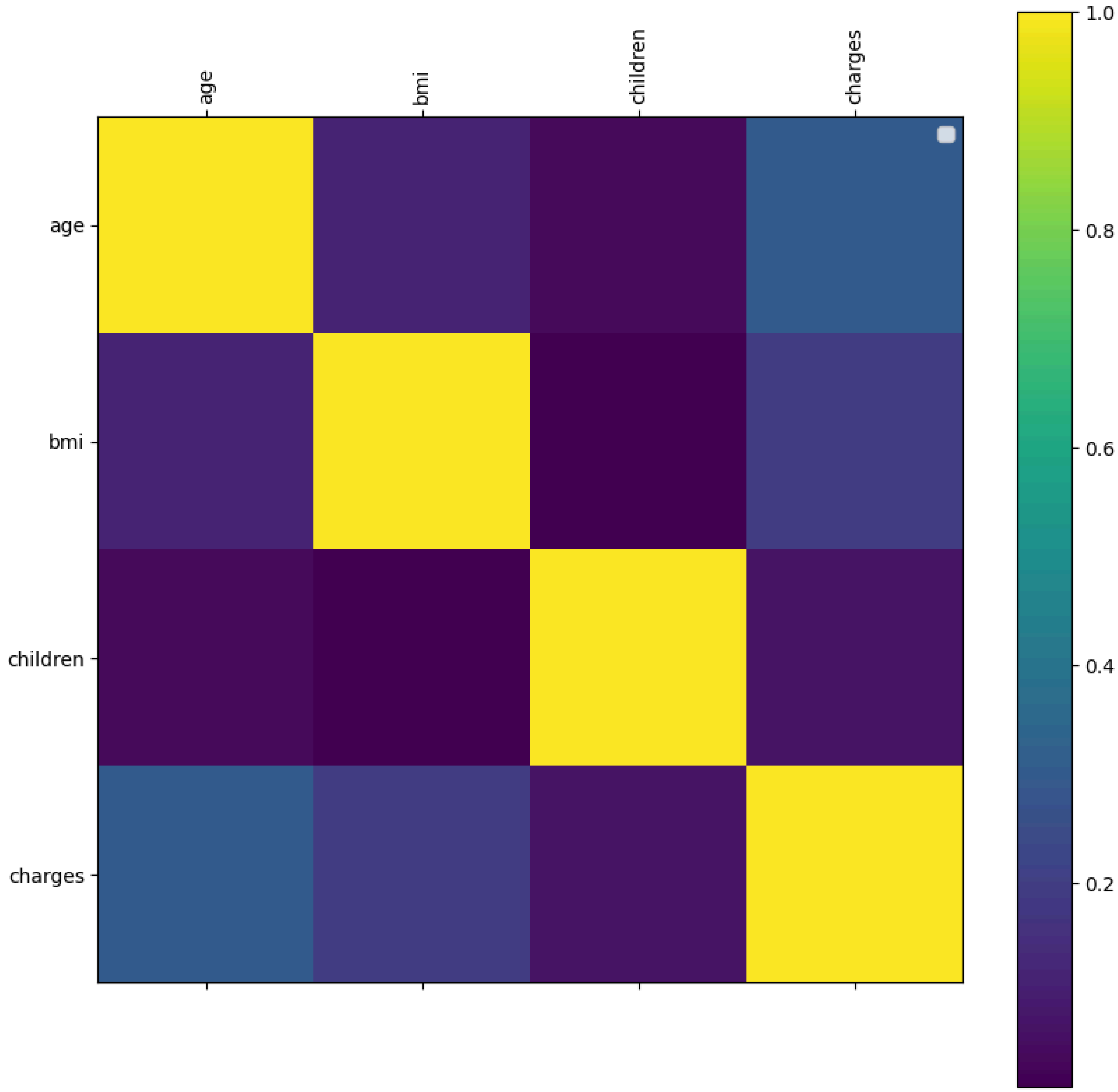
	age	bmi	children	charges
age	1.000000	0.109272	0.042469	0.299008
bmi	0.109272	1.000000	0.012759	0.198341
children	0.042469	0.012759	1.000000	0.067998
charges	0.299008	0.198341	0.067998	1.000000

In [35]:

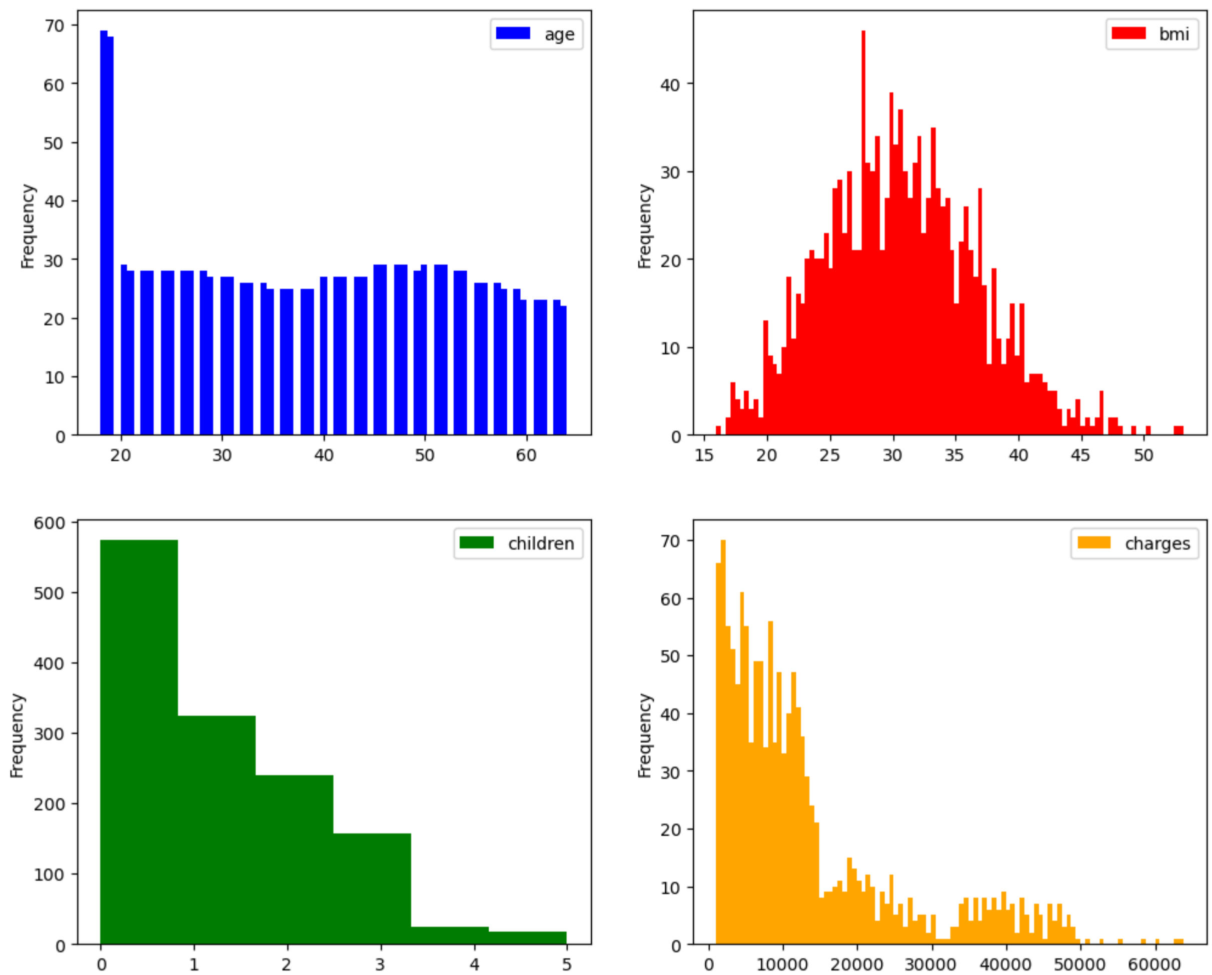
```
import matplotlib.pyplot as plt

def plot_corr(df,size=10):
```

```
    '''Function plots a graphical correlation matrix for each pair of columns in th

    Input:
        df: pandas DataFrame
        size: vertical and horizontal size of the plot'''

    corr = df.corr(numeric_only = True)
    fig, ax = plt.subplots(figsize=(size, size))
    ax.legend()
    cax = ax.matshow(corr)
    fig.colorbar(cax)
    plt.xticks(range(len(corr.columns)), corr.columns, rotation='vertical')
    plt.yticks(range(len(corr.columns)), corr.columns)

plot_corr(insurance)
```

```
C:\Users\Mahaveera Foods\AppData\Local\Temp\ipykernel_1076\3440799524.py:12: UserWar
ning: No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
  ax.legend()
```

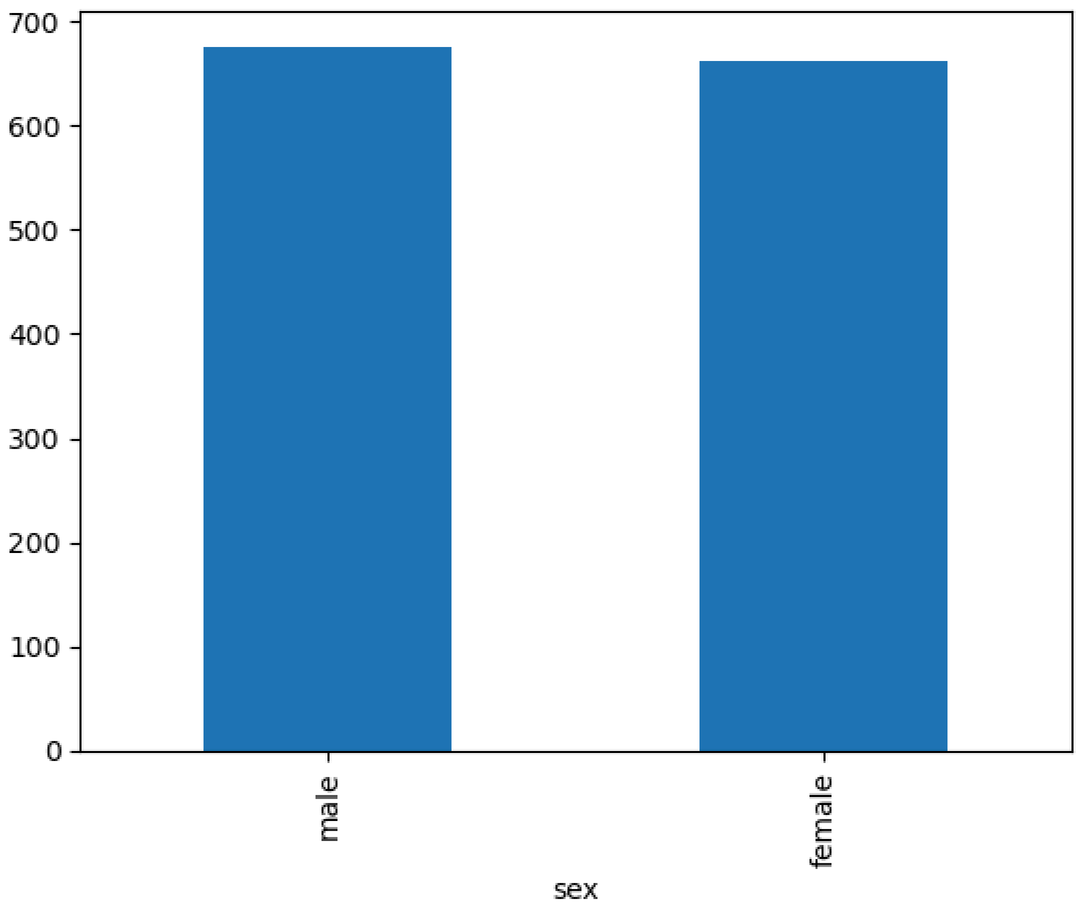


In [37]:
```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 10))
insurance.plot(kind="hist", y="age", bins=70, color="b", ax=axes[0][0])
insurance.plot(kind="hist", y="bmi", bins=100, color="r", ax=axes[0][1])
insurance.plot(kind="hist", y="children", bins=6, color="g", ax=axes[1][0])
insurance.plot(kind="hist", y="charges", bins=100, color="orange", ax=axes[1][1])
plt.show()
```



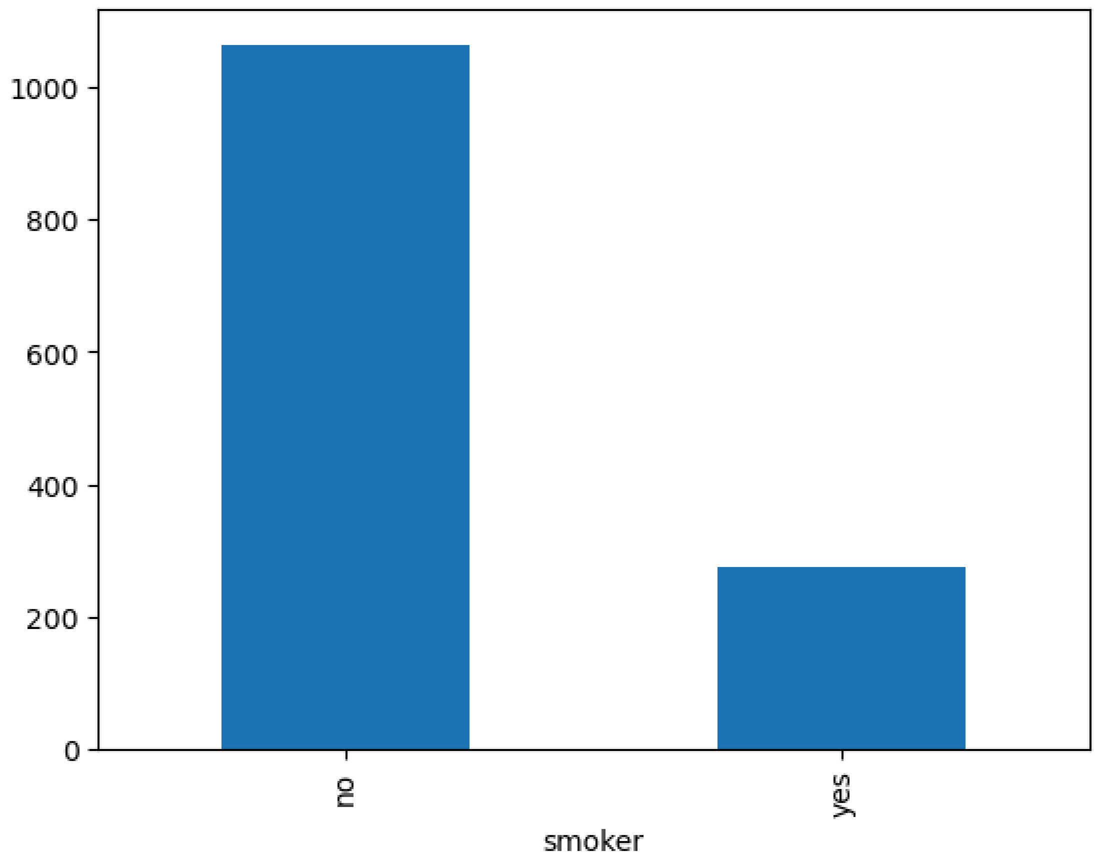
In [39]:
```
insurance['sex'].value_counts().plot(kind='bar')
```

Out[39]: <Axes: xlabel='sex'>

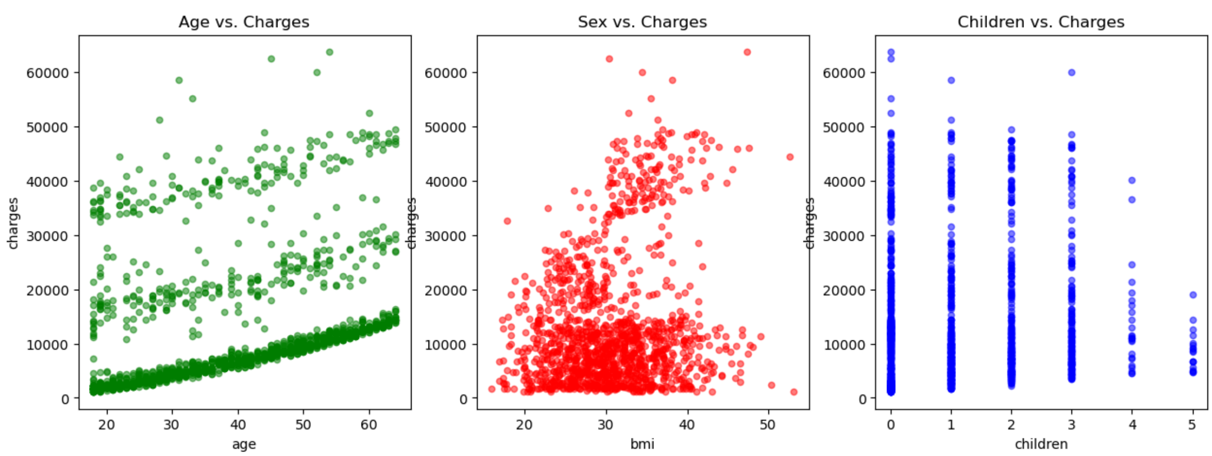


In [41]:
```
insurance['smoker'].value_counts().plot(kind='bar')
```

Out[41]: <Axes: xlabel='smoker'>

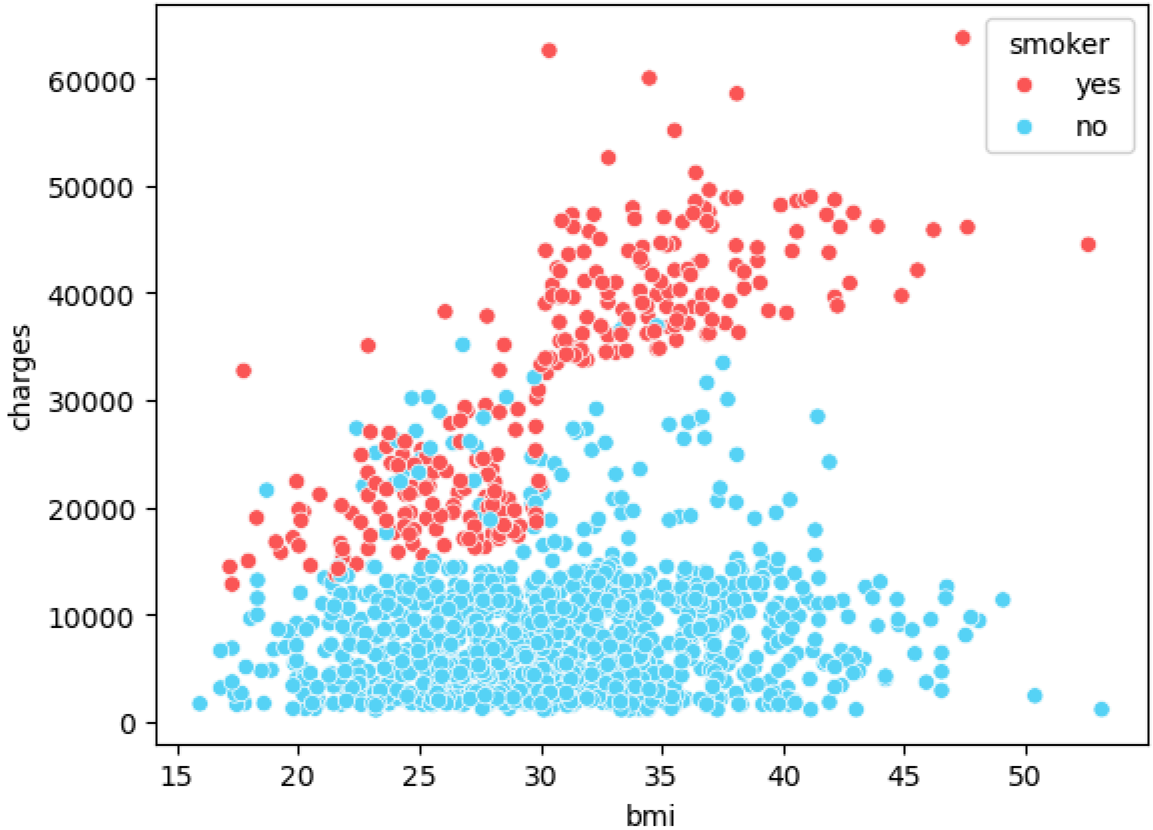



```
In [43]: fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))
         insurance.plot(kind='scatter', x='age', y='charges', alpha=0.5, color='green', ax=a
         insurance.plot(kind='scatter', x='bmi', y='charges', alpha=0.5, color='red', ax=axe
         insurance.plot(kind='scatter', x='children', y='charges', alpha=0.5, color='blue',
         plt.show()
```



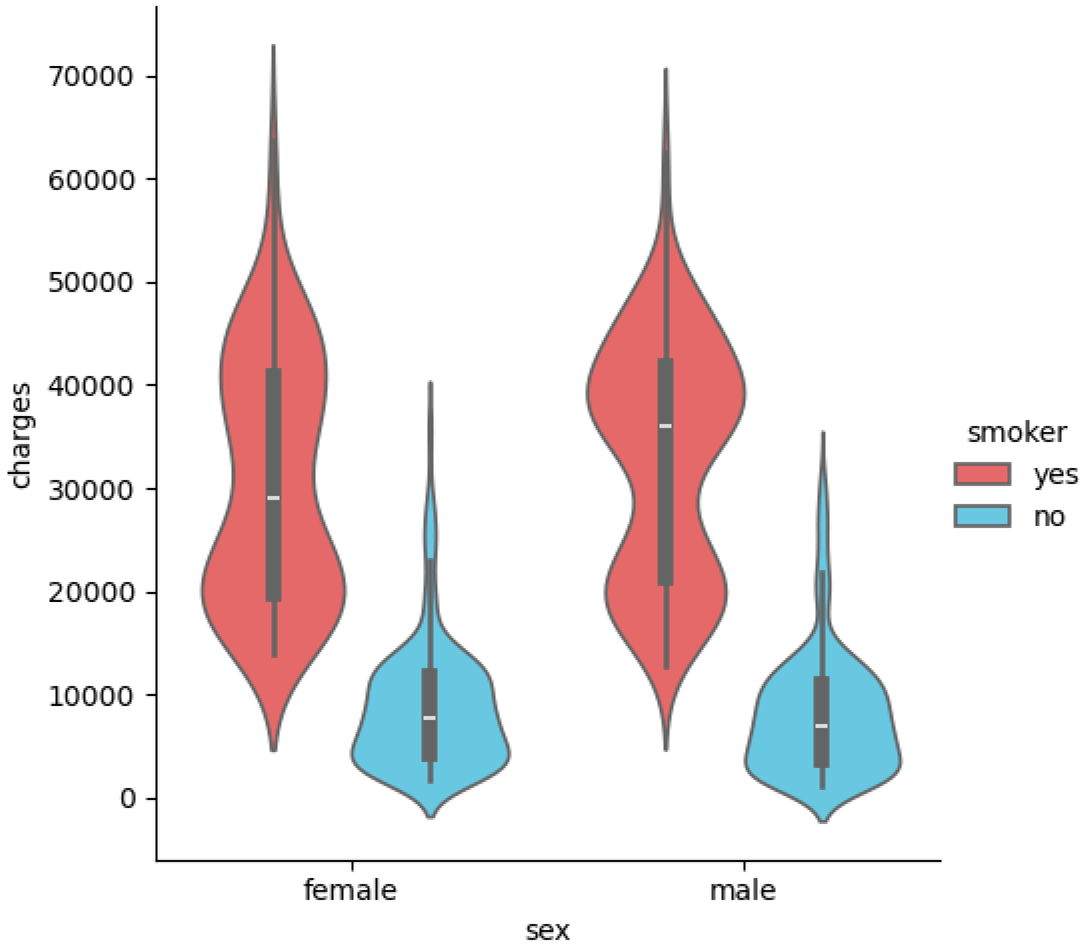

```
In [45]: import seaborn as sns  # Imorting Seaborn Library
         pal = ["#FA5858", "#58D3F7"]
         sns.scatterplot(x="bmi", y="charges", data=insurance, palette=pal, hue='smoker')
```

```
Out[45]: <Axes: xlabel='bmi', ylabel='charges'>
```



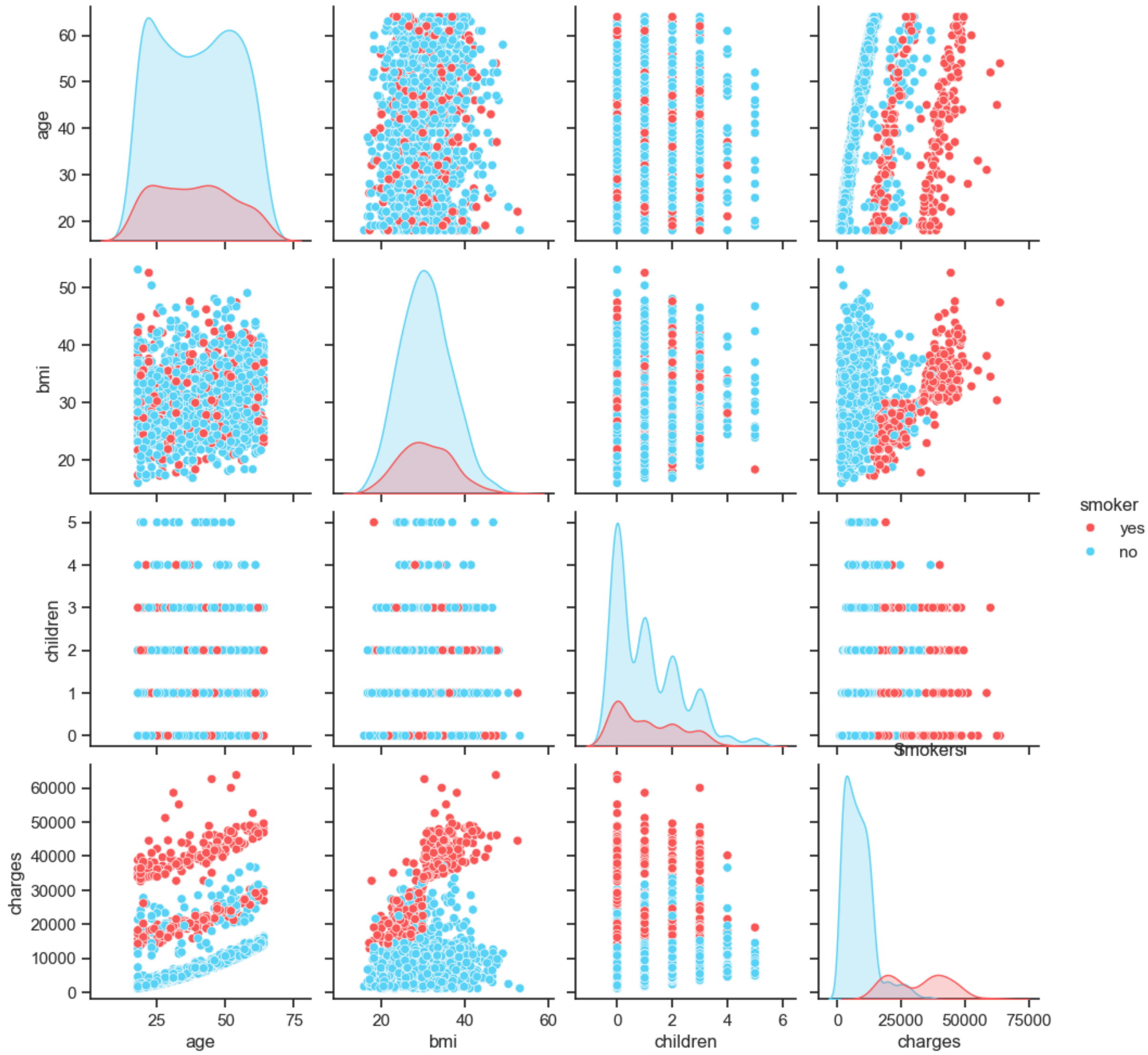
In [47]:
```
pal = ["#FA5858", "#58D3F7"]
sns.catplot(x="sex", y="charges", hue="smoker",
            kind="violin", data=insurance, palette = pal)
```

Out[47]: <seaborn.axisgrid.FacetGrid at 0x23e23029a30>



In [48]:
```
import seaborn as sns

sns.set(style="ticks")
pal = ["#FA5858", "#58D3F7"]

sns.pairplot(insurance, hue="smoker", palette=pal)
plt.title("Smokers")
```

Out[48]:
```
Text(0.5, 1.0, 'Smokers')
```



Preparing Data for Machine Learning Algorithms

In [51]:
```
insurance.head()
```

Out[51]:

	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 [54]:
```
insurance['region'].unique()
```

Out[54]: array(['southwest', 'southeast', 'northwest', 'northeast'], dtype=object)

In [56]:
```
insurance.drop(["region"], axis=1, inplace=True)
insurance.head()
```

Out[56]:

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

In [58]:
```
# Changing binary categories to 1s and 0s
insurance['sex'] = insurance['sex'].map(lambda s :1  if s == 'female' else 0)
insurance['smoker'] = insurance['smoker'].map(lambda s :1  if s == 'yes' else 0)

insurance.head()
```

Out[58]:

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

In [60]:
```
X = insurance.drop(['charges'], axis = 1)
y = insurance.charges
```

Modeling our Data

In [63]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)
lr = LinearRegression().fit(X_train, y_train)

y_train_pred = lr.predict(X_train)
y_test_pred = lr.predict(X_test)

print(lr.score(X_test, y_test))
```

```
0.7952171980481992
```

Score is the R2 score, which varies between 0 and 100%. It is closely related to the MSE but not the same.

Wikipedia defines r2 like this, " ... is the proportion of the variance in the dependent variable that is predictable from the independent variable(s)." Another definition is "(total variance explained by model) / total variance." So if it is 100%, the two variables are perfectly correlated, i.e., with no variance at all. A low value would show a low level of correlation, meaning a regression model that is not valid, but not in all cases.

In [66]:
```
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_test_pred})
results
```

Out[66]:

	Actual	Predicted
578	9724.53000	11457.247488
610	8547.69130	9925.930740
569	45702.02235	37768.549419
1034	12950.07120	15853.346790
198	9644.25250	6939.119725
...	...	...
574	13224.05705	14429.077741
1174	4433.91590	6705.247131
1327	9377.90470	11152.092298
817	3597.59600	7200.555548
1337	29141.36030	36542.082417

335 rows × 2 columns

In [68]:
```
# Normalize the data
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [70]:
```
pd.DataFrame(X_train).head()
```

Out[70]:

	0	1	2	3	4
0	-0.514853	0.985155	-0.181331	-0.063607	-0.503736
1	1.548746	0.985155	-1.393130	-0.892144	-0.503736
2	-1.439915	-1.015069	-0.982242	-0.063607	-0.503736
3	-1.368757	0.985155	-1.011133	-0.892144	1.985167
4	-0.941805	0.985155	-1.362635	-0.892144	-0.503736

In [72]:

```
pd.DataFrame(y_train).head()
```

Out[72]:

	charges
1075	4562.84210
131	13616.35860
15	1837.23700
1223	26125.67477
1137	3176.28770

In [74]:

```
from sklearn.linear_model import LinearRegression  # Import Linear Regression model

multiple_linear_reg = LinearRegression(fit_intercept=False)  # Create a instance fo
multiple_linear_reg.fit(X_train, y_train)  # Fit data to the model
```

Out[74]:

```
▾ LinearRegression
LinearRegression(fit_intercept=False)
```

In [76]:

```
from sklearn.preprocessing import PolynomialFeatures

polynomial_features = PolynomialFeatures(degree=3)  # Create a PolynomialFeatures i
x_train_poly = polynomial_features.fit_transform(X_train)  # Fit and transform the
x_test_poly = polynomial_features.fit_transform(X_test)  # Fit and transform the te

polynomial_reg = LinearRegression(fit_intercept=False)  # Create a instance for Lin
polynomial_reg.fit(x_train_poly, y_train)  # Fit data to the model
```

Out[76]:

```
▾ LinearRegression
LinearRegression(fit_intercept=False)
```

In [78]:

```
from sklearn.tree import DecisionTreeRegressor  # Import Decision Tree Regression m

decision_tree_reg = DecisionTreeRegressor(max_depth=5, random_state=13)  # Create a
decision_tree_reg.fit(X_train, y_train)  # Fit data to the model
```

Out[78]:

```
DecisionTreeRegressor
DecisionTreeRegressor(max_depth=5, random_state=13)
```

In [80]:

```
from sklearn.ensemble import RandomForestRegressor  # Import Random Forest Regressi

random_forest_reg = RandomForestRegressor(n_estimators=400, max_depth=5, random_sta
random_forest_reg.fit(X_train, y_train)  # Fit data to the model
```

Out[80]:

```
RandomForestRegressor
RandomForestRegressor(max_depth=5, n_estimators=400, random_state=13)
```

NOTE: n_estimators represents the number of trees in the forest. Usually the higher the number of trees the better to learn the data. However, adding a lot of trees can slow down the training process considerably, therefore we do a parameter search to find the sweet spot.

In [83]:

```
from sklearn.svm import SVR  # Import SVR model

support_vector_reg = SVR(gamma="auto", kernel="linear", C=1000)  # Create a instanc
support_vector_reg.fit(X_train, y_train)  # Fit data to the model
```

Out[83]:

```
SVR
SVR(C=1000, gamma='auto', kernel='linear')
```

In [85]:

```
from sklearn.model_selection import cross_val_predict  # For K-Fold Cross Validatio
from sklearn.metrics import r2_score  # For find accuracy with R2 Score
from sklearn.metrics import mean_squared_error  # For MSE
from math import sqrt  # For squareroot operation
```

Evaluating Multiple Linear Regression Model

In [88]:

```
# Prediction with training dataset:
y_pred_MLR_train = multiple_linear_reg.predict(X_train)

# Prediction with testing dataset:
y_pred_MLR_test = multiple_linear_reg.predict(X_test)

# Find training accuracy for this model:
accuracy_MLR_train = r2_score(y_train, y_pred_MLR_train)
print("Training Accuracy for Multiple Linear Regression Model: ", accuracy_MLR_trai

# Find testing accuracy for this model:
accuracy_MLR_test = r2_score(y_test, y_pred_MLR_test)
print("Testing Accuracy for Multiple Linear Regression Model: ", accuracy_MLR_test)

# Find RMSE for training data:
RMSE_MLR_train = sqrt(mean_squared_error(y_train, y_pred_MLR_train))
```

```
print("RMSE for Training Data: ", RMSE_MLR_train)

# Find RMSE for testing data:
RMSE_MLR_test = sqrt(mean_squared_error(y_test, y_pred_MLR_test))
print("RMSE for Testing Data: ", RMSE_MLR_test)

# Prediction with 10-Fold Cross Validation:
y_pred_cv_MLR = cross_val_predict(multiple_linear_reg, X, y, cv=10)

# Find accuracy after 10-Fold Cross Validation
accuracy_cv_MLR = r2_score(y, y_pred_cv_MLR)
print("Accuracy for 10-Fold Cross Predicted Multiple Linaer Regression Model: ", ac
```

```
Training Accuracy for Multiple Linear Regression Model:  -0.4895607457643889
Testing Accuracy for Multiple Linear Regression Model:  -0.324110208111029
RMSE for Training Data:  14589.30728329809
RMSE for Testing Data:  14438.166278828226
Accuracy for 10-Fold Cross Predicted Multiple Linaer Regression Model:  0.7171134192
00113
```

Evaluating Polynomial Regression Model

In [91]:

```
# Prediction with training dataset:
y_pred_PR_train = polynomial_reg.predict(x_train_poly)

# Prediction with testing dataset:
y_pred_PR_test = polynomial_reg.predict(x_test_poly)

# Find training accuracy for this model:
accuracy_PR_train = r2_score(y_train, y_pred_PR_train)
print("Training Accuracy for Polynomial Regression Model: ", accuracy_PR_train)

# Find testing accuracy for this model:
accuracy_PR_test = r2_score(y_test, y_pred_PR_test)
print("Testing Accuracy for Polynomial Regression Model: ", accuracy_PR_test)

# Find RMSE for training data:
RMSE_PR_train = sqrt(mean_squared_error(y_train, y_pred_PR_train))
print("RMSE for Training Data: ", RMSE_PR_train)

# Find RMSE for testing data:
RMSE_PR_test = sqrt(mean_squared_error(y_test, y_pred_PR_test))
print("RMSE for Testing Data: ", RMSE_PR_test)

# Prediction with 10-Fold Cross Validation:
y_pred_cv_PR = cross_val_predict(polynomial_reg, polynomial_features.fit_transform(

# Find accuracy after 10-Fold Cross Validation
accuracy_cv_PR = r2_score(y, y_pred_cv_PR)
print("Accuracy for 10-Fold Cross Predicted Polynomial Regression Model: ", accurac
```

```
Training Accuracy for Polynomial Regression Model:  0.8355072839439216
Testing Accuracy for Polynomial Regression Model:  0.881007372576663
RMSE for Training Data:  4848.181811885191
RMSE for Testing Data:  4328.2266950880685
Accuracy for 10-Fold Cross Predicted Polynomial Regression Model:  0.839107291768899
8
```

Evaluating Decision Tree Regression Model

In [94]:
```python
# Prediction with training dataset:
y_pred_DTR_train = decision_tree_reg.predict(X_train)

# Prediction with testing dataset:
y_pred_DTR_test = decision_tree_reg.predict(X_test)

# Find training accuracy for this model:
accuracy_DTR_train = r2_score(y_train, y_pred_DTR_train)
print("Training Accuracy for Decision Tree Regression Model: ", accuracy_DTR_train)

# Find testing accuracy for this model:
accuracy_DTR_test = r2_score(y_test, y_pred_DTR_test)
print("Testing Accuracy for Decision Tree Regression Model: ", accuracy_DTR_test)

# Find RMSE for training data:
RMSE_DTR_train = sqrt(mean_squared_error(y_train, y_pred_DTR_train))
print("RMSE for Training Data: ", RMSE_DTR_train)

# Find RMSE for testing data:
RMSE_DTR_test = sqrt(mean_squared_error(y_test, y_pred_DTR_test))
print("RMSE for Testing Data: ", RMSE_DTR_test)

# Prediction with 10-Fold Cross Validation:
y_pred_cv_DTR = cross_val_predict(decision_tree_reg, X, y, cv=10)

# Find accuracy after 10-Fold Cross Validation
accuracy_cv_DTR = r2_score(y, y_pred_cv_DTR)
print("Accuracy for 10-Fold Cross Predicted Decision Tree Regression Model: ", accu
```

```
Training Accuracy for Decision Tree Regression Model:  0.8694256791947466
Testing Accuracy for Decision Tree Regression Model:  0.8711939682763064
RMSE for Training Data:  4319.5096631798915
RMSE for Testing Data:  4503.167201972113
Accuracy for 10-Fold Cross Predicted Decision Tree Regression Model:  0.849424103159
5924
```

Evaluating Random Forest Regression Model

In [97]:
```python
# Prediction with training dataset:
y_pred_RFR_train = random_forest_reg.predict(X_train)

# Prediction with testing dataset:
y_pred_RFR_test = random_forest_reg.predict(X_test)

# Find training accuracy for this model:
accuracy_RFR_train = r2_score(y_train, y_pred_RFR_train)
```

```
print("Training Accuracy for Random Forest Regression Model: ", accuracy_RFR_train)

# Find testing accuracy for this model:
accuracy_RFR_test = r2_score(y_test, y_pred_RFR_test)
print("Testing Accuracy for Random Forest Regression Model: ", accuracy_RFR_test)

# Find RMSE for training data:
RMSE_RFR_train = sqrt(mean_squared_error(y_train, y_pred_RFR_train))
print("RMSE for Training Data: ", RMSE_RFR_train)

# Find RMSE for testing data:
RMSE_RFR_test = sqrt(mean_squared_error(y_test, y_pred_RFR_test))
print("RMSE for Testing Data: ", RMSE_RFR_test)

# Prediction with 10-Fold Cross Validation:
y_pred_cv_RFR = cross_val_predict(random_forest_reg, X, y, cv=10)

# Find accuracy after 10-Fold Cross Validation
accuracy_cv_RFR = r2_score(y, y_pred_cv_RFR)
print("Accuracy for 10-Fold Cross Predicted Random Forest Regression Model: ", accu
```

```
Training Accuracy for Random Forest Regression Model:  0.8786315807304423
Testing Accuracy for Random Forest Regression Model:  0.8969061711453914
RMSE for Training Data:  4164.457266795271
RMSE for Testing Data:  4028.7127866662904
Accuracy for 10-Fold Cross Predicted Random Forest Regression Model:  0.857378869678
5247
```

Evaluating Support Vector Regression Model

In [99]:

```
# Prediction with training dataset:
y_pred_SVR_train = support_vector_reg.predict(X_train)

# Prediction with testing dataset:
y_pred_SVR_test = support_vector_reg.predict(X_test)

# Find training accuracy for this model:
accuracy_SVR_train = r2_score(y_train, y_pred_SVR_train)
print("Training Accuracy for Support Vector Regression Model: ", accuracy_SVR_train

# Find testing accuracy for this model:
accuracy_SVR_test = r2_score(y_test, y_pred_SVR_test)
print("Testing Accuracy for Support Vector Regression Model: ", accuracy_SVR_test)

# Find RMSE for training data:
RMSE_SVR_train = sqrt(mean_squared_error(y_train, y_pred_SVR_train))
print("RMSE for Training Data: ", RMSE_SVR_train)

# Find RMSE for testing data:
RMSE_SVR_test = sqrt(mean_squared_error(y_test, y_pred_SVR_test))
print("RMSE for Testing Data: ", RMSE_SVR_test)

# Prediction with 10-Fold Cross Validation:
y_pred_cv_SVR = cross_val_predict(support_vector_reg, X, y, cv=10)
```

```python
# Find accuracy after 10-Fold Cross Validation
accuracy_cv_SVR = r2_score(y, y_pred_cv_SVR)
print("Accuracy for 10-Fold Cross Predicted Support Vector Regression Model: ", acc
```

```
Training Accuracy for Support Vector Regression Model:  0.6522181188488032
Testing Accuracy for Support Vector Regression Model:  0.734317356160165
RMSE for Training Data:  7049.511742429496
RMSE for Testing Data:  6467.427432129216
Accuracy for 10-Fold Cross Predicted Support Vector Regression Model:  0.70581312219
77515
```

In [100…

```python
# Compare all results in one table
training_accuracies = [accuracy_MLR_train, accuracy_PR_train, accuracy_DTR_train, a
testing_accuracies = [accuracy_MLR_test, accuracy_PR_test, accuracy_DTR_test, accur
training_RMSE = [RMSE_MLR_train, RMSE_PR_train, RMSE_DTR_train, RMSE_RFR_train, RMS
testing_RMSE = [RMSE_MLR_test, RMSE_PR_test, RMSE_DTR_test, RMSE_RFR_test, RMSE_SVR
cv_accuracies = [accuracy_cv_MLR, accuracy_cv_PR, accuracy_cv_DTR, accuracy_cv_RFR,

parameters = ["fit_intercept=False", "fit_intercept=False", "max_depth=5", "n_estim

table_data = {"Parameters": parameters, "Training Accuracy": training_accuracies, "
              "Training RMSE": training_RMSE, "Testing RMSE": testing_RMSE, "10-Fol
model_names = ["Multiple Linear Regression", "Polynomial Regression", "Decision Tre

table_dataframe = pd.DataFrame(data=table_data, index=model_names)
table_dataframe
```

Out[100…

	Parameters	Training Accuracy	Testing Accuracy	Training RMSE	Testing RMSE	10-Fold Score
Multiple Linear Regression	fit_intercept=False	-0.489561	-0.324110	14589.307283	14438.166279	0.717113
Polynomial Regression	fit_intercept=False	0.835507	0.881007	4848.181812	4328.226695	0.839107
Decision Tree Regression	max_depth=5	0.869426	0.871194	4319.509663	4503.167202	0.849424
Random Forest Regression	n_estimators=400, max_depth=5	0.878632	0.896906	4164.457267	4028.712787	0.857379
Support Vector Regression	kernel="linear", C=1000	0.652218	0.734317	7049.511742	6467.427432	0.705813

Our best classifier is our Random Forests using 400 estimators and a max_depth of 5

R^2 (coefficient of determination) regression score function.

Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

Let's test our best regression on some new data

In [107…

```
input_data = {'age': [35],
              'sex': ['male'],
              'bmi': [26],
              'children': [0],
              'smoker': ['no'],
              'region': ['southeast']}

input_data = pd.DataFrame(input_data)
input_data
```

Out[107…

	age	sex	bmi	children	smoker	region
0	35	male	26	0	no	southeast

In [109…

```
# Our simple pre-processing
input_data.drop(["region"], axis=1, inplace=True)
input_data['sex'] = input_data['sex'].map(lambda s :1  if s == 'female' else 0)
input_data['smoker'] = input_data['smoker'].map(lambda s :1  if s == 'yes' else 0)
input_data
```

Out[109…

	age	sex	bmi	children	smoker
0	35	0	26	0	0

In [111…

```
# Scale our input data
input_data = sc.transform(input_data)
input_data
```

Out[111…

```
array([[-0.30137763, -1.01506865, -0.75753763, -0.89214407, -0.50373604]])
```

In [113…

```
# Reshape our input data in the format required by sklearn models
input_data = input_data.reshape(1, -1)
print(input_data.shape)
input_data
```

```
(1, 5)
```

Out[113…

```
array([[-0.30137763, -1.01506865, -0.75753763, -0.89214407, -0.50373604]])
```

In [115…

```
# Get our predicted insurance rate for our new customer
random_forest_reg.predict(input_data)
```

Out[115…

```
array([5961.85333748])
```

In [117…

```
# Note Standard Scaler remembers your inputs so you can use it still here
print(sc.mean_)
print(sc.scale_)
```

```
[39.23529412  0.50747757 30.7197657   1.07676969  0.20239282]
[14.05311355  0.49994408  6.23040428  1.20694597  0.40178348]
```

Thus we can see the forecasting on our data is best suited for Predicting Insurance Premiumns

-----------------------------------End of Notebook---

In [ ]: