In [1]:

```
#importing data

import pandas as pd
insurance = pd.read_csv("D:/Data Science and Deep Learning for Business/datasciencef
insurance.head()
```

Out[1]:

	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 [2]:

```
#describing dataset

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

```
Rows     :  1338
Columns  :  7

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

Missing values :
 age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64

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

In [3]:

```
#correlation
insurance.corr()
```

```
C:\Users\Pradeep\AppData\Local\Temp\ipykernel_23608\641068510.py:2: Future
Warning: The default value of numeric_only in DataFrame.corr is deprecate
d. In a future version, it will default to False. Select only valid column
s or specify the value of numeric_only to silence this warning.
  insurance.corr()
```

Out[3]:

	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

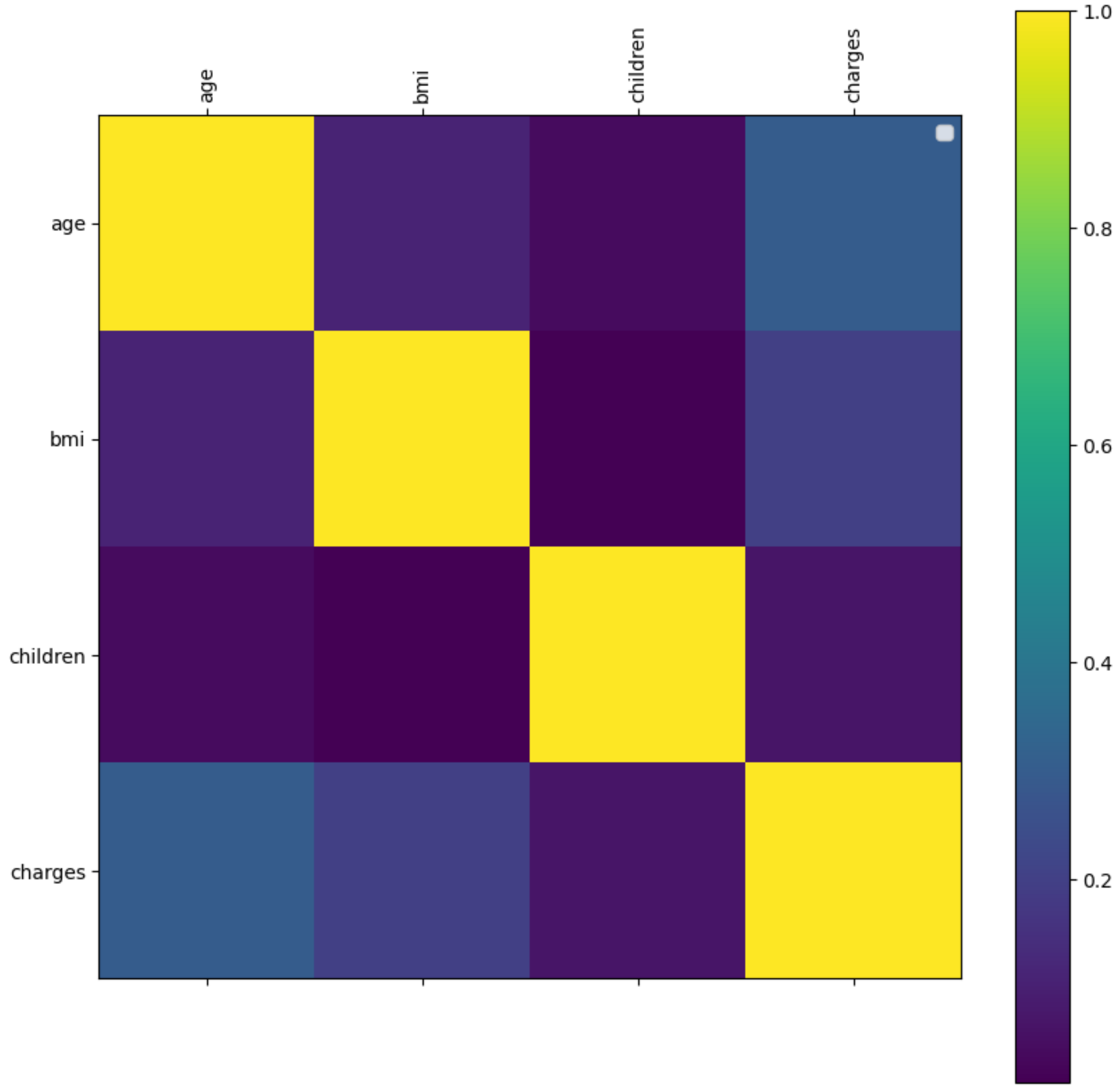
In [4]:

```
#correlation plot
import matplotlib.pyplot as plt

def plot_corr(df):
    corr = df.corr()
    fig, ax = plt.subplots(figsize=(10, 10))
    ax.legend()
    cax = ax.matshow(corr) #array as matrix
    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\Pradeep\AppData\Local\Temp\ipykernel_23608\4274883086.py:5: Futur
eWarning: The default value of numeric_only in DataFrame.corr is deprecate
d. In a future version, it will default to False. Select only valid column
s or specify the value of numeric_only to silence this warning.
  corr = df.corr()
No artists with labels found to put in legend.  Note that artists whose la
bel start with an underscore are ignored when legend() is called with no a
rgument.
```

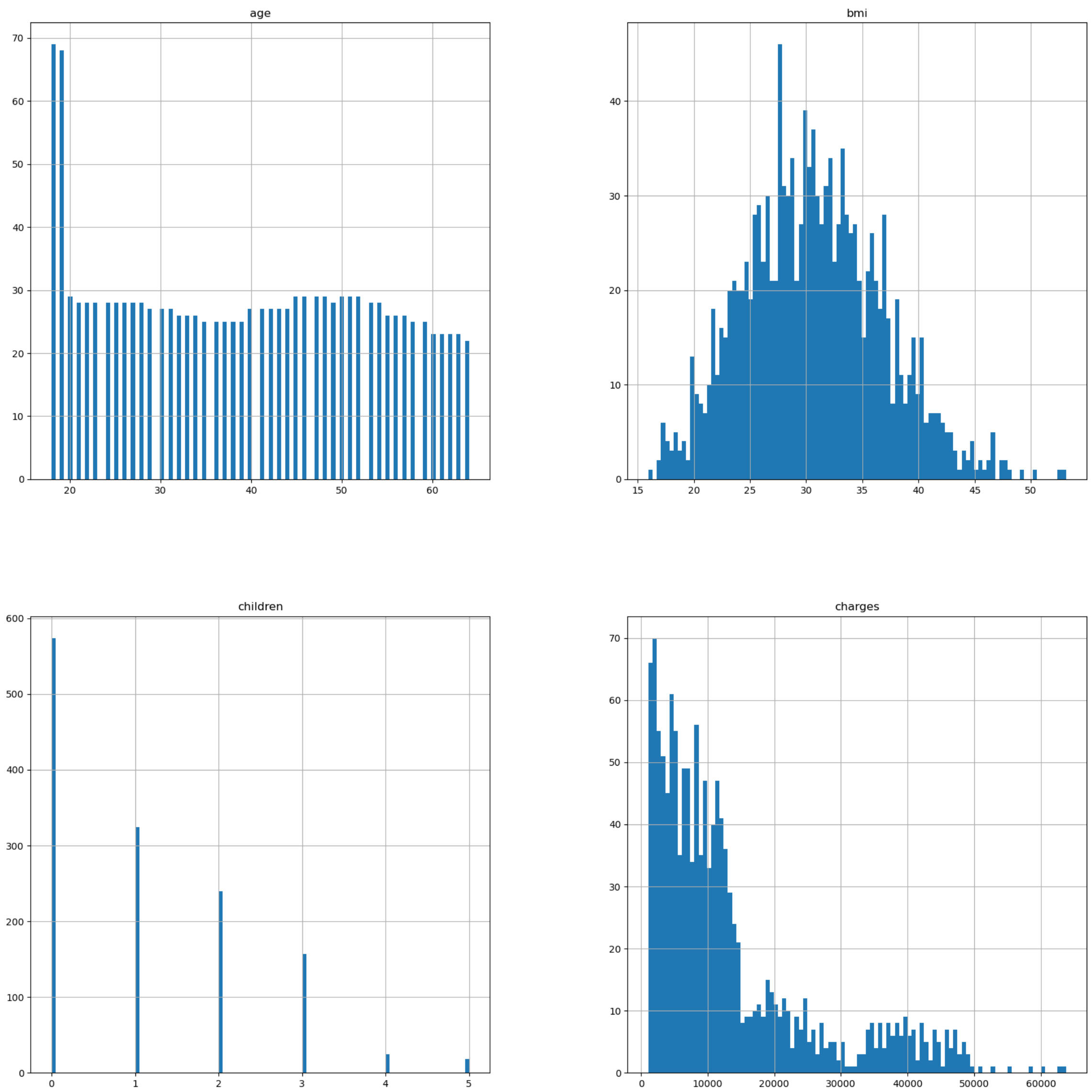


In [5]:

```
insurance.hist(bins=100,figsize=(20,20))
```

Out[5]:

```
array([[<AxesSubplot: title={'center': 'age'}>,
        <AxesSubplot: title={'center': 'bmi'}>],
       [<AxesSubplot: title={'center': 'children'}>,
        <AxesSubplot: title={'center': 'charges'}>]], dtype=object)
```

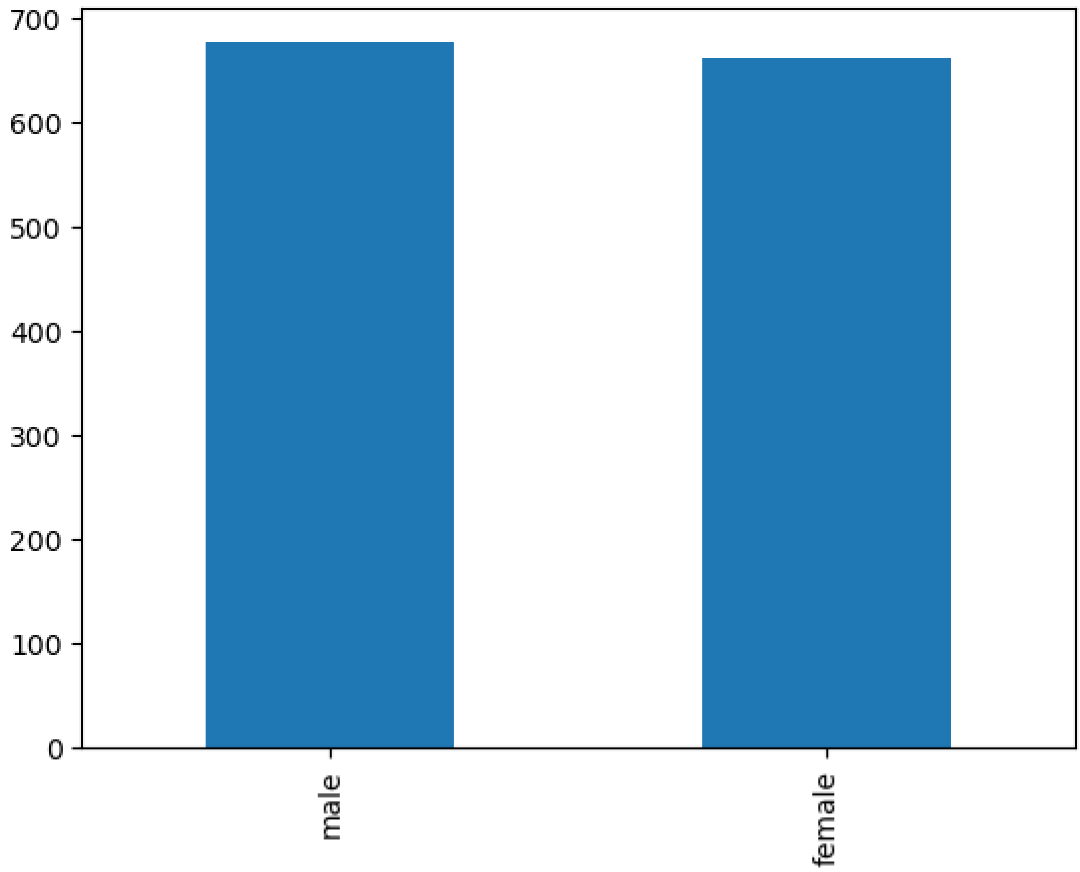


In [6]:

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

Out[6]:

```
<AxesSubplot: >
```

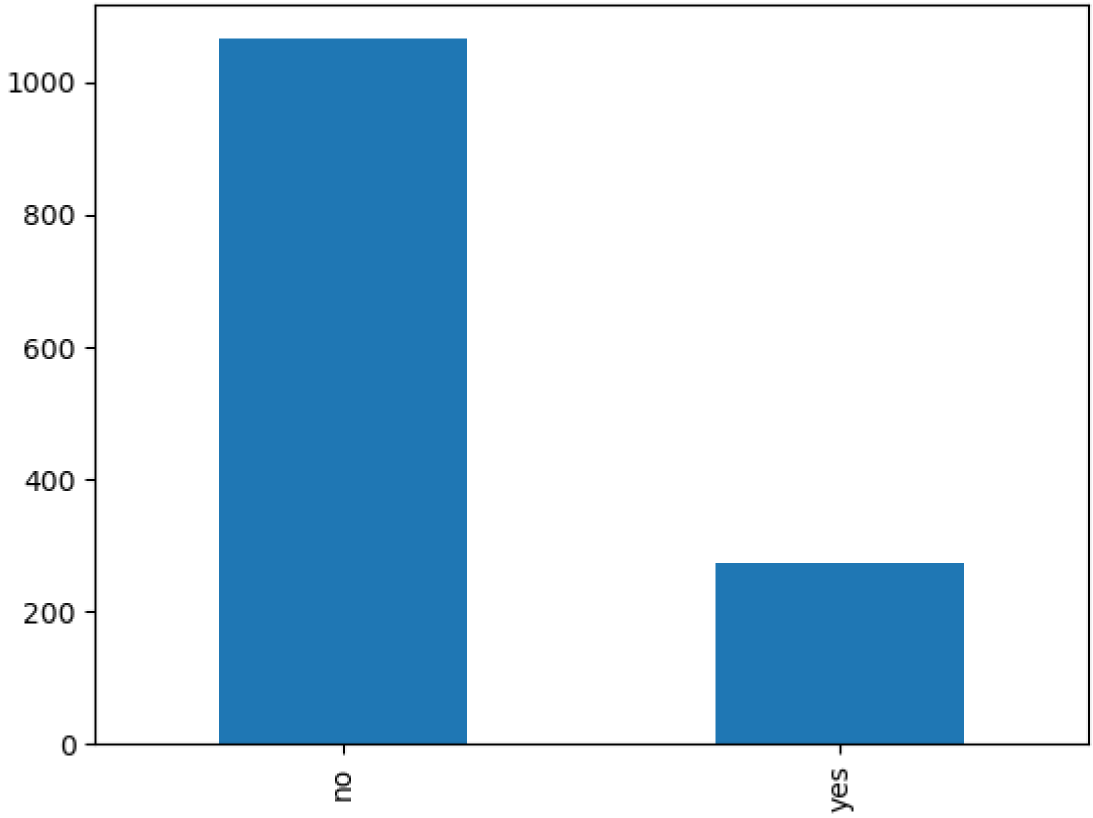


In [7]:

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

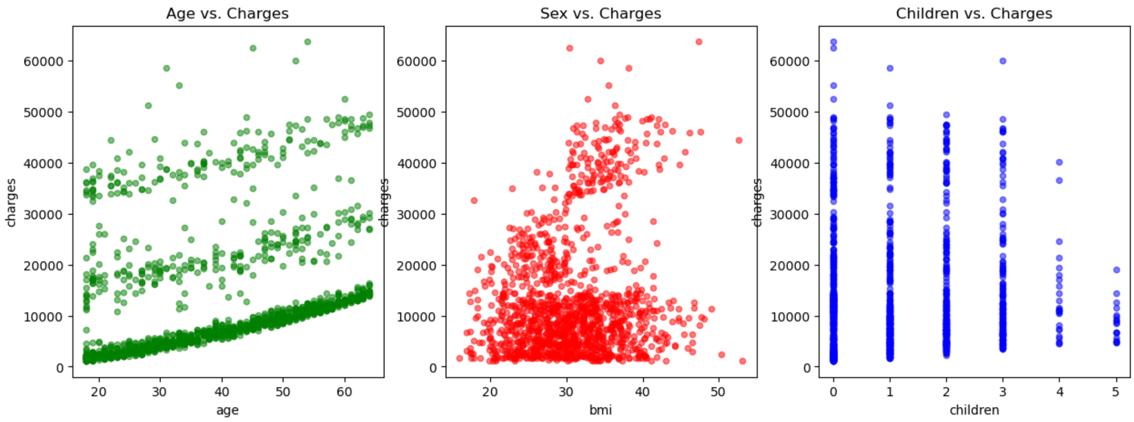
Out[7]:

<AxesSubplot: >



In [8]:

```
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=ax
insurance.plot(kind='scatter', x='bmi', y='charges', alpha=0.5, color='red', ax=axes
insurance.plot(kind='scatter', x='children', y='charges', alpha=0.5, color='blue', a
plt.show()
```

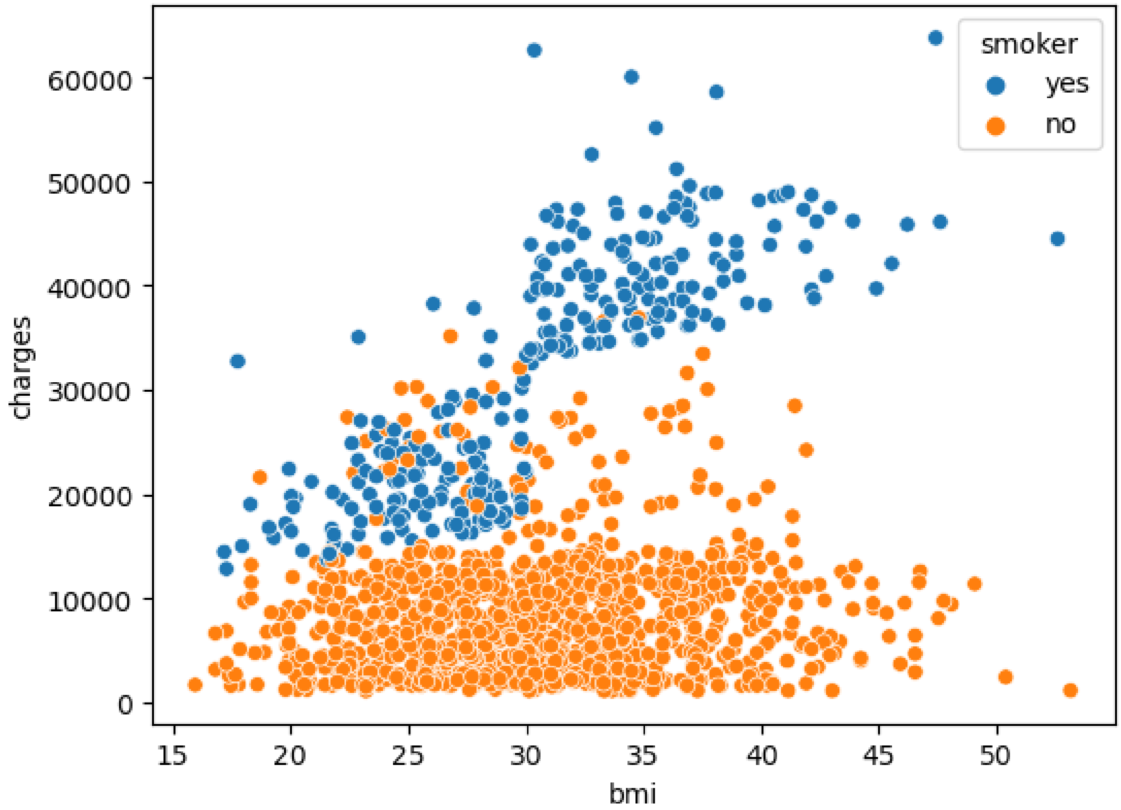


In [9]:

```
# Imorting Seaborn library
import seaborn as sns
sns.scatterplot(x="bmi", y="charges", data=insurance, hue='smoker')
```

Out[9]:

```
<AxesSubplot: xlabel='bmi', ylabel='charges'>
```



In [10]:

```
insurance.head()
```

Out[10]:

	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 [11]:

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

Out[11]:

```
array(['southwest', 'southeast', 'northwest', 'northeast'], dtype=object)
```

In [12]:

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

Out[12]:

	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 [13]:

```
insurance['sex'].unique()
```

Out[13]:

```
array(['female', 'male'], dtype=object)
```

In [14]:

```
insurance['sex'] = insurance['sex'].map(lambda s :1  if s == 'female' else 0)
insurance.head()
```

Out[14]:

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

In [15]:

```
insurance['smoker'].unique()
```

Out[15]:

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

In [16]:

```
insurance['smoker'] = insurance['smoker'].map(lambda s :1  if s == 'yes' else 0)
insurance.head()
```

Out[16]:

	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 [17]:

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

In [18]:

```
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)
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))
```

```
0.7608249670500205
```

In [19]:

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

Out[19]:

	Actual	Predicted
732	4234.92700	5553.750604
708	6113.23105	7507.860180
766	8062.76400	10758.101412
1281	24535.69855	33420.147504
1000	17361.76610	27344.626895
...	...	...
911	33732.68670	25531.328016
1051	14394.55790	12859.213123
1041	1704.70015	-197.426943
482	1622.18850	2481.649508
1164	7153.55390	8243.460729

335 rows × 2 columns

In [20]:

```
# Normalize the data
from sklearn.preprocessing import StandardScaler

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

pd.DataFrame(X_train).head()
```

Out[20]:

	0	1	2	3	4
0	-1.064757	-0.991067	-0.109293	-0.921510	2.029125
1	1.697735	-0.991067	1.798254	-0.921510	-0.492823
2	-1.064757	-0.991067	-1.121243	-0.921510	-0.492823
3	0.281073	1.009014	-0.097821	-0.093307	-0.492823
4	0.847738	1.009014	1.185348	-0.093307	-0.492823

In [21]:

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

Out[21]:

	charges
1250	18648.42170
170	13405.39030
693	2352.96845
1263	7337.74800
147	9877.60770

In [22]:

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

multiple_linear_reg = LinearRegression(fit_intercept=False)
multiple_linear_reg.fit(X_train, y_train)
```

Out[22]:

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

In [23]:

```
#PolynomialFeatures
from sklearn.preprocessing import PolynomialFeatures

polynomial_features = PolynomialFeatures(degree=3)
x_train_poly = polynomial_features.fit_transform(X_train)
x_test_poly = polynomial_features.fit_transform(X_test)

polynomial_reg = LinearRegression(fit_intercept=False)
polynomial_reg.fit(x_train_poly, y_train)
```

Out[23]:

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

In [24]:

```
#Decision Tree Regression model
from sklearn.tree import DecisionTreeRegressor

decision_tree_reg = DecisionTreeRegressor(max_depth=5, random_state=13)
decision_tree_reg.fit(X_train, y_train)
```

Out[24]:

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

In [25]:

```
#Random Forest Regression model
from sklearn.ensemble import RandomForestRegressor

random_forest_reg = RandomForestRegressor(n_estimators=400, max_depth=5, random_stat
random_forest_reg.fit(X_train, y_train)
```

Out[25]:

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

In [26]:

```
#SVR model

from sklearn.svm import SVR

support_vector_reg = SVR(gamma="auto", kernel="linear", C=1000)
support_vector_reg.fit(X_train, y_train)
```

Out[26]:

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

In [27]:

```
#Importing evaluation metrics

from sklearn.model_selection import cross_val_predict
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from math import sqrt
```

In [28]:

```
#Evaluating Multiple Linear Regression Model

# 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_trair

# 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: ", acc
```

```
Training Accuracy for Multiple Linear Regression Model:  -0.44214251208022
58
Testing Accuracy for Multiple Linear Regression Model:  -0.308218253498038
2
RMSE for Training Data:  14178.647894150441
RMSE for Testing Data:  14728.890264369307
Accuracy for 10-Fold Cross Predicted Multiple Linaer Regression Model:  0.
717113419200113
```

In [29]:

```
#Evaluating Polynomial Regression Model

# 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(X

# 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: ", accuracy
```

```
Training Accuracy for Polynomial Regression Model:  0.8482275078787543
Testing Accuracy for Polynomial Regression Model:  0.8424785147329269
RMSE for Training Data:  4599.676205617919
RMSE for Testing Data:  5110.928711122368
Accuracy for 10-Fold Cross Predicted Polynomial Regression Model:  0.83910
72917688998
```

In [30]:

```
#  Evaluating Decision Tree Regression Model

# 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: ", accur
```

```
Training Accuracy for Decision Tree Regression Model:  0.8810575984972877
Testing Accuracy for Decision Tree Regression Model:  0.8312217664502496
RMSE for Training Data:  4071.9185223433237
RMSE for Testing Data:  5290.395531447086
Accuracy for 10-Fold Cross Predicted Decision Tree Regression Model:  0.84
94241031595924
```

In [31]:

```
# Evaluating Random Forest Regression Model

# 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: ", accur
```

```
Training Accuracy for Random Forest Regression Model:  0.8881034695908978
Testing Accuracy for Random Forest Regression Model:  0.8709486381615694
RMSE for Training Data:  3949.4719903815003
RMSE for Testing Data:  4626.059562572722
Accuracy for 10-Fold Cross Predicted Random Forest Regression Model:  0.85
73788696785247
```

In [32]:

```
# Evaluating Support Vector Regression Model

# 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)

# 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: ", accu
```

```
Training Accuracy for Support Vector Regression Model:  0.7114656153556187
Testing Accuracy for Support Vector Regression Model:  0.7314092980873221
RMSE for Training Data:  6342.04786792858
RMSE for Testing Data:  6673.83417232676
Accuracy for 10-Fold Cross Predicted Support Vector Regression Model:  0.7
058131221977515
```

In [33]:

```
# Compare all results in one table
training_accuracies = [accuracy_MLR_train, accuracy_PR_train, accuracy_DTR_train, ac
testing_accuracies = [accuracy_MLR_test, accuracy_PR_test, accuracy_DTR_test, accura
training_RMSE = [RMSE_MLR_train, RMSE_PR_train, RMSE_DTR_train, RMSE_RFR_train, RMSE
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_estima

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

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

Out[33]:

	Parameters	Training Accuracy	Testing Accuracy	Training RMSE	Testing RMSE	10-Fold Score
Multiple Linear Regression	fit_intercept=False	-0.442143	-0.308218	14178.647894	14728.890264	0.717113
Polynomial Regression	fit_intercept=False	0.848228	0.842479	4599.676206	5110.928711	0.839107
Decision Tree Regression	max_depth=5	0.881058	0.831222	4071.918522	5290.395531	0.849424
Random Forest Regression	n_estimators=400, max_depth=5	0.888103	0.870949	3949.471990	4626.059563	0.857379
Support Vector Regression	kernel="linear", C=1000	0.711466	0.731409	6342.047868	6673.834172	0.705813

In [34]:

```
#test on new input data
input_data = {'age': [35],
              'sex': ['male'],
              'bmi': [26],
              'children': [0],
              'smoker': ['no'],
              'region': ['southeast']}

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

Out[34]:

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

In [35]:

```
#Input data 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[35]:

	age	sex	bmi	children	smoker
0	35	0	26	0	0

In [36]:

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

Out[36]:

```
array([[-0.28559244, -0.99106682, -0.73694802, -0.92151002, -0.49282334]])
```

In [37]:

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

Out[37]:

```
array([5989.69398529])
```

In [ ]:

```

```