Curso Data Science

Prof Me Eng Marcelo Bianchi
Data Scientist

Aula 7 – Machine Learning
Data Exploration , Data Visualization and Multiple Linear Regression

Machine Learning

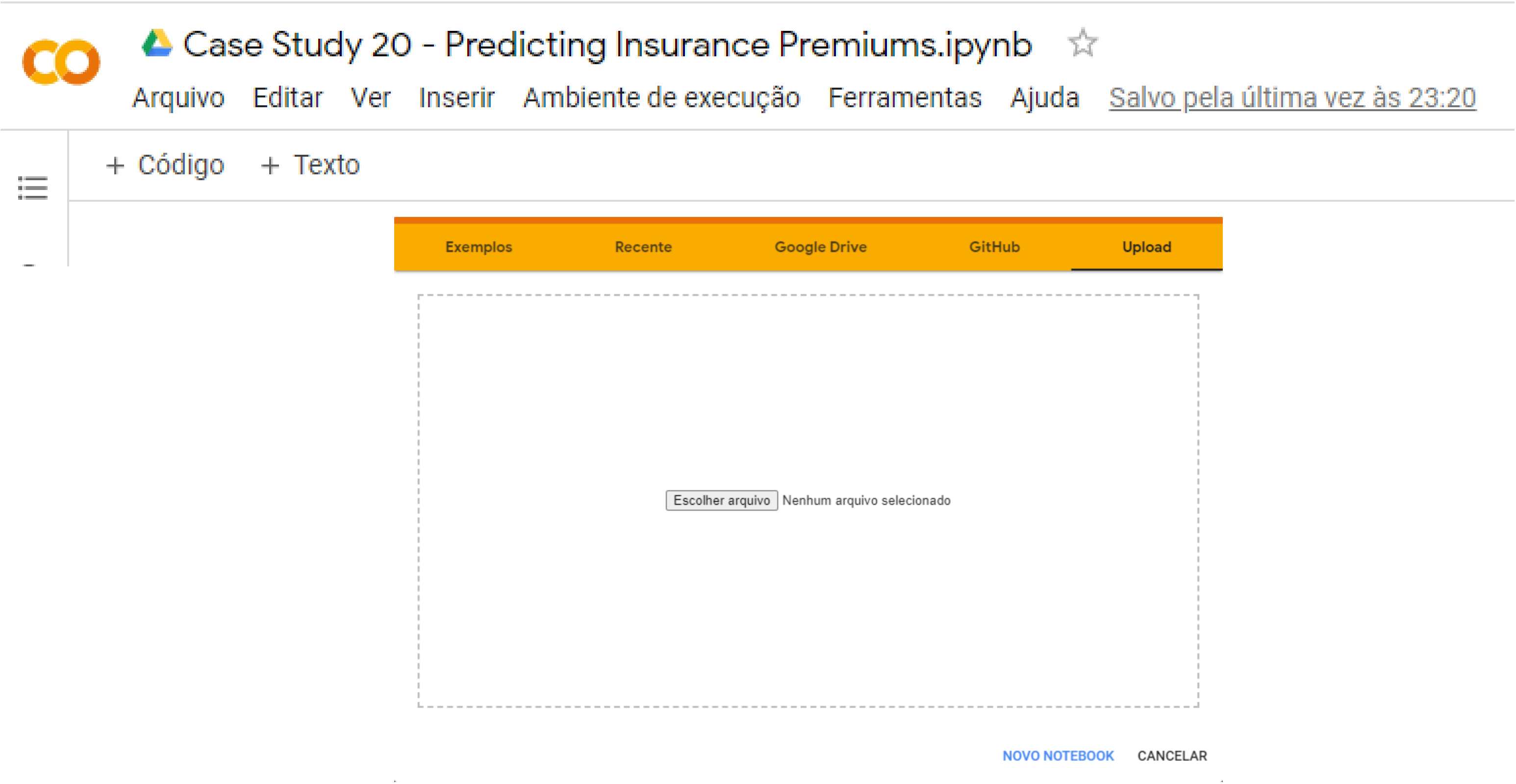
Data Exploration

Multiple Linear Regression

Step 1 – Data Exploration / Data Visualization

Step 2 – Multiple Linear Regression

Google Colaboratory



Nossa Missão

Criar um Sistema de análise preditiva para seguros

- Dataset contém os seguintes atributos:
- Age, Sex, BMI (Base Month Income) , Children, Smoker, Region and their charges

Objetivo

- Utilizar o modelo preditivo para realizar predições dos valores cobrados para os clientes (baseado no perfil).

Dataset

	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

```python
import pandas as pd

# Uncomment this line if using this notebook locally
# insurance = pd.read_csv('./data/insurance/insurance.csv')

file_name = "https://raw.githubusercontent.com/rajeevratan84/datascienceforbusiness/master/insurance.csv"
insurance = pd.read_csv(file_name)

# Preview our data
insurance.head()
```

```
[2] 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
```

```
[3] insurance.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

```
[4] 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
```

```
[5] insurance.corr()
```

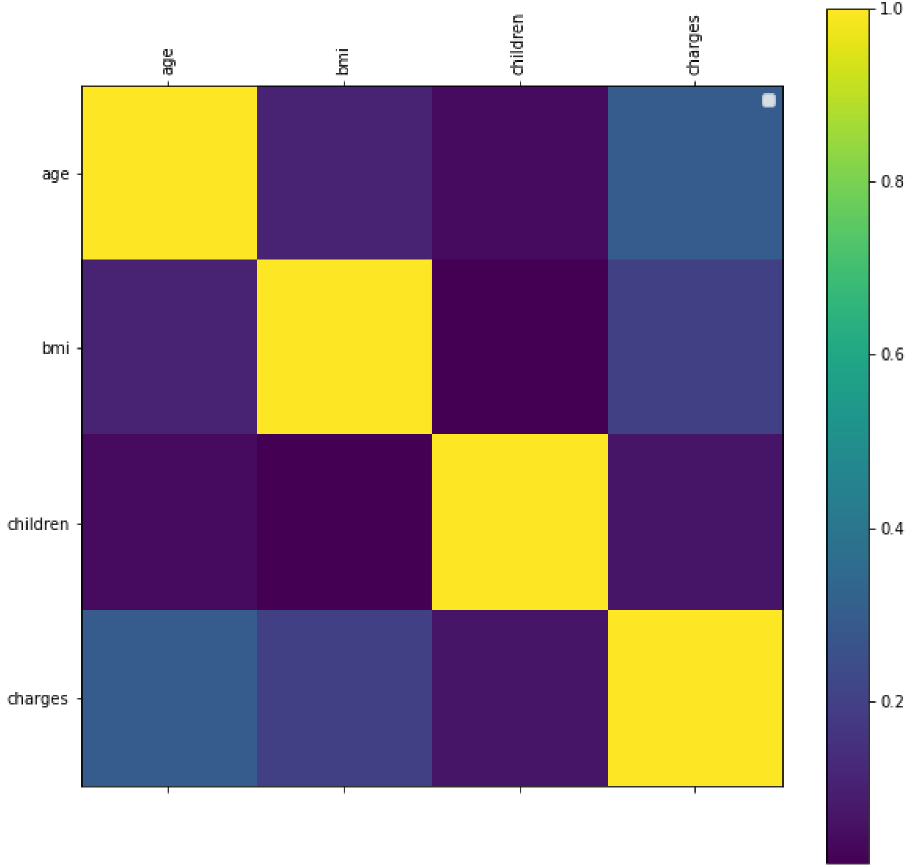
	age	bmi	children	charges
age	1.000000	0.109272	0.042469	0.299008
bmi	0.109272	1.000000	0.012759	0.198341
children	0.042469	0.012759	1.000000	0.067998
charges	0.299008	0.198341	0.067998	1.000000

```
[6] import matplotlib.pyplot as plt

    def plot_corr(df,size=10):
        '''Function plots a graphical correlation matrix for each pair of columns in the dataframe.

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

        corr = df.corr()
        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)
```

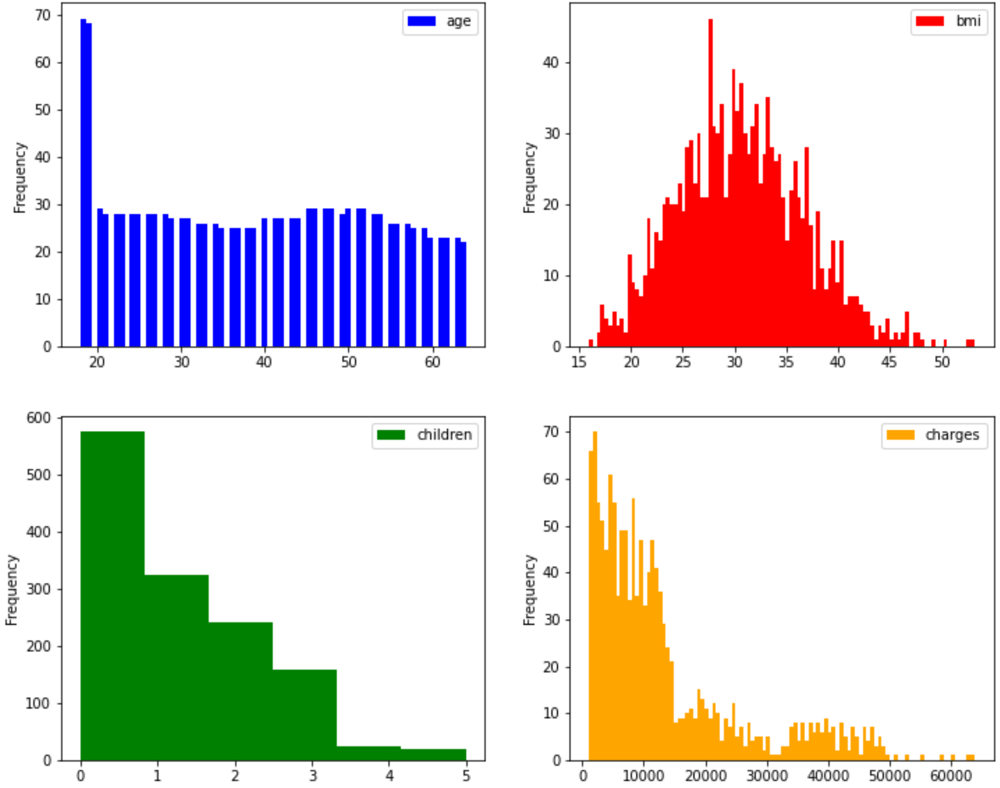
```
No handles with labels found to put in legend.
```

```
plot_corr(insurance)
```

```
No handles with labels found to put in legend.
```

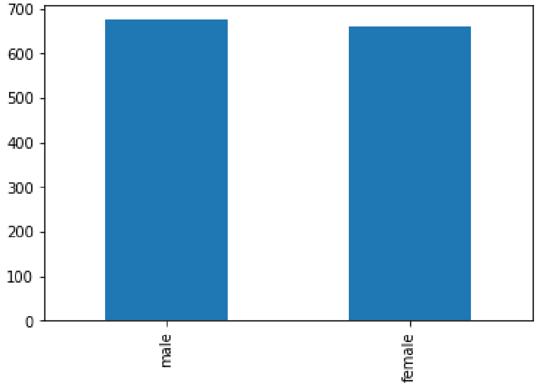


```
[7] fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 10))
    insurance.plot(kind="hist", y="age", bins=70, color="b", ax=axes[0][0])
    insurance.plot(kind="hist", y="bmi", bins=100, color="r", ax=axes[0][1])
    insurance.plot(kind="hist", y="children", bins=6, color="g", ax=axes[1][0])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



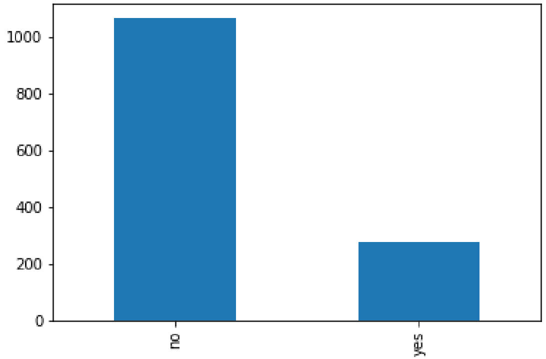
```
insurance['sex'].value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2ecdc09410>
```

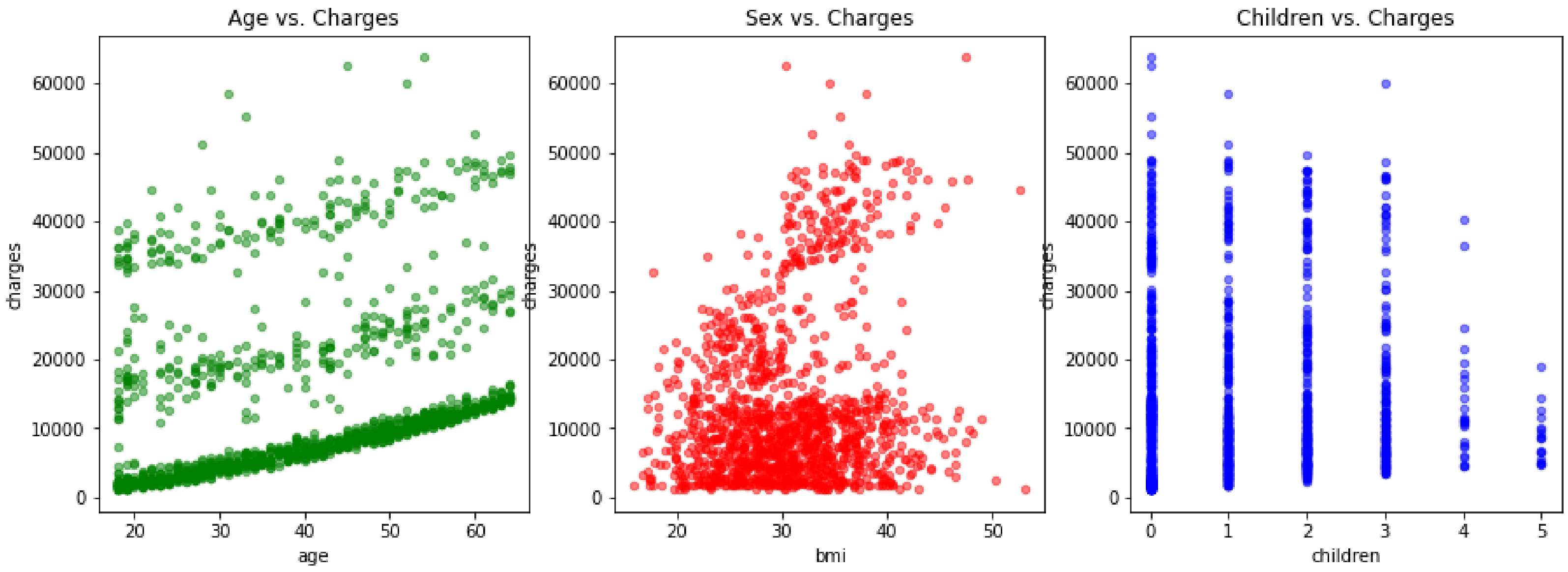



```
[9] insurance['smoker'].value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2ec6b10290>
```

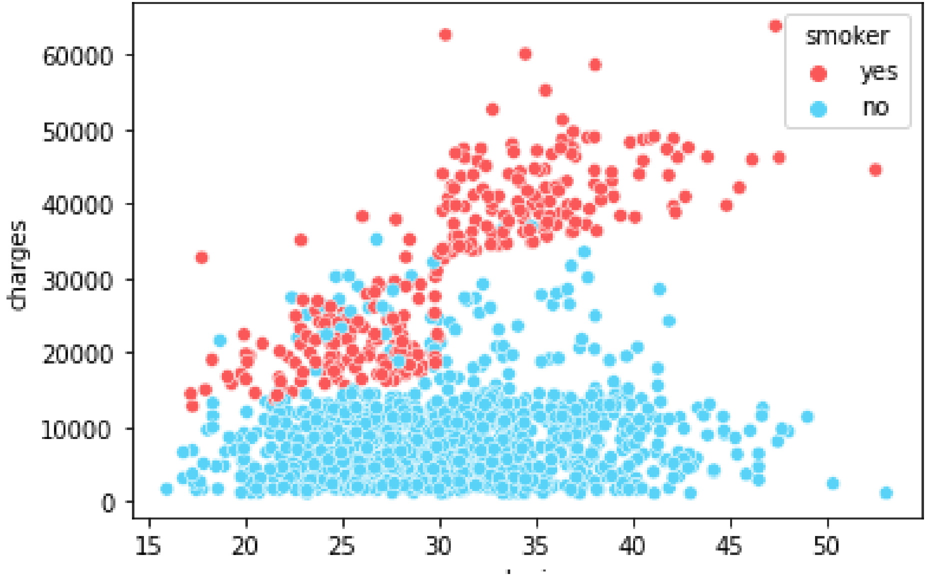


```
[10] fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))
     insurance.plot(kind='scatter', x='age', y='charges', alpha=0.5, color='green', ax=axes[0], title="Age vs. Charges")
     insurance.plot(kind='scatter', x='bmi', y='charges', alpha=0.5, color='red', ax=axes[1], title="Sex vs. Charges")
     insurance.plot(kind='scatter', x='children', y='charges', alpha=0.5, color='blue', ax=axes[2], title="Children vs. Charges")
     plt.show()
```



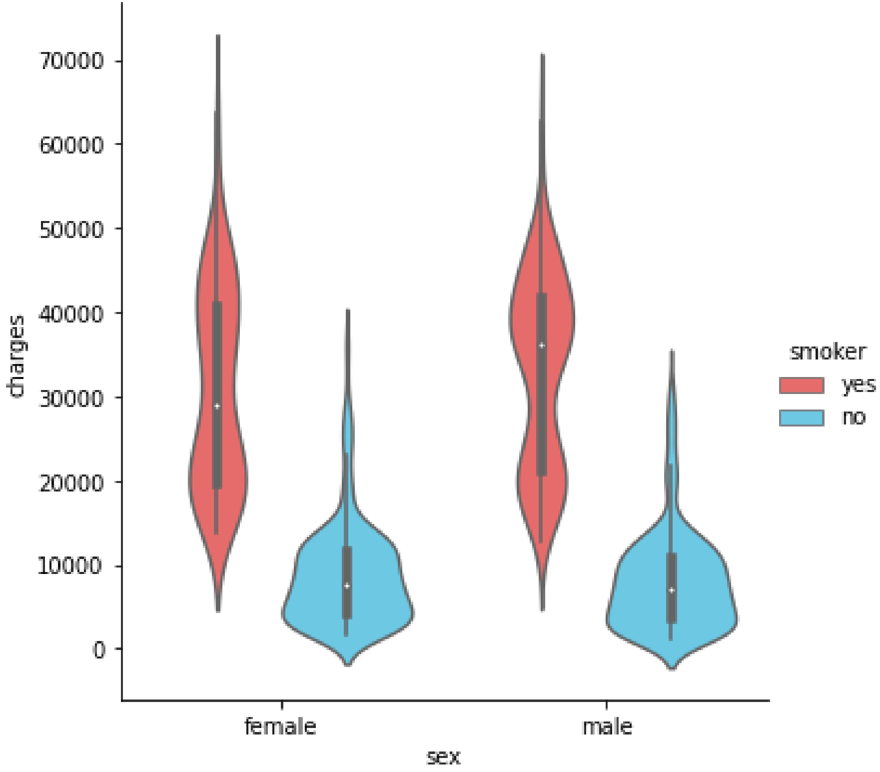
```python
import seaborn as sns  # Imorting Seaborn library
pal = ["#FA5858", "#58D3F7"]
sns.scatterplot(x="bmi", y="charges", data=insurance, palette=pal, hue='smoker')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2eb8441c10>
```



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

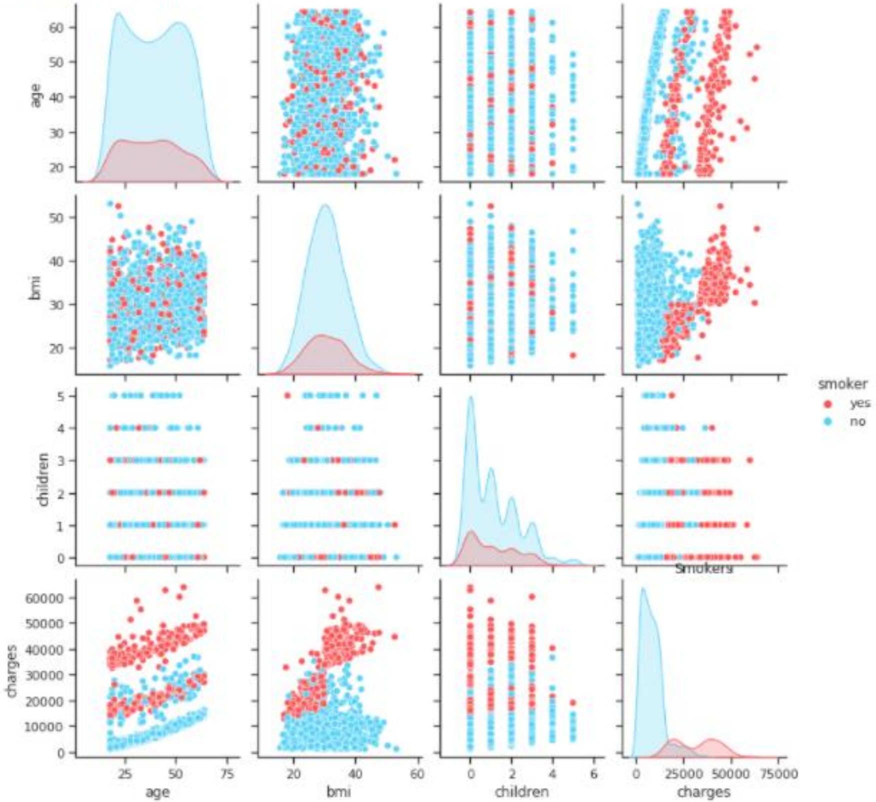
```
<seaborn.axisgrid.FacetGrid at 0x7f2eb7d68590>
```



[15]

```python
import seaborn as sns

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

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

```
Text(0.5, 1.0, 'Smokers')
```



Preparing Data for Machine Learning Algorithms

```
[16] insurance.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

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

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

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

	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

```
[19] # 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()
```

	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

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

Modeling our Data

```
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.

," ... 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.

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

	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

```
[34] # Normalize the data
     from sklearn.preprocessing import StandardScaler

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

```
[35] pd.DataFrame(X_train).head()
```

	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

```
[36] pd.DataFrame(y_train).head()
```

	charges
1075	4562.84210
131	13616.35860
15	1837.23700
1223	26125.67477
1137	3176.28770

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

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

```
LinearRegression(copy_X=True, fit_intercept=False, n_jobs=None, normalize=False)
```

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.

```
[38] from sklearn.model_selection import cross_val_predict  # For K-Fold Cross Validation
     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

```
# 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_train)

# 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: ", accuracy_cv_MLR)
```

```
Training Accuracy for Multiple Linear Regression Model:  -0.48956074576438935
Testing Accuracy for Multiple Linear Regression Model:  -0.3241102081110292
RMSE for Training Data:  14589.307283298092
RMSE for Testing Data:  14438.16627882823
Accuracy for 10-Fold Cross Predicted Multiple Linaer Regression Model:  0.717113419200113
```

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

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

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

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

```
[42] # 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
```

	age	sex	bmi	children	smoker
0	35	0	26	0	0

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

```
array([[ 3.50000000e+01,  4.25050490e-17,  2.60000000e+01,
         9.74074040e-17, -7.79259232e-17]])
```

```
[44] # 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)
array([[ 3.50000000e+01,  4.25050490e-17,  2.60000000e+01,
         9.74074040e-17, -7.79259232e-17]])
```

Predição

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

     array([174707.25423291])
```

Muito obrigado!