Medical Insurance Cost Analysis

This notebook analyzes medical insurance costs based on various patient characteristics. We'll explore the data, create visualizations, and build predictive models using different regression techniques.

1. Import Libraries

In [1]:
```
# Import necessary libraries for data analysis and modeling
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score, train_test_split

# Set up visualization
%matplotlib inline
```

2. Load and Explore the Dataset

In [2]:
```
# Load the dataset from the URL
filepath = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMD
df = pd.read_csv(filepath, header=None)

# Display the first 10 rows of the dataframe
print(df.head(10))
```

```
    0  1       2  3  4  5            6
0  19  1  27.900  0  1  3  16884.92400
1  18  2  33.770  1  0  4   1725.55230
2  28  2  33.000  3  0  4   4449.46200
3  33  2  22.705  0  0  1  21984.47061
4  32  2  28.880  0  0  1   3866.85520
5  31  1  25.740  0  ?  4   3756.62160
6  46  1  33.440  1  0  4   8240.58960
7  37  1  27.740  3  0  1   7281.50560
8  37  2  29.830  2  0  2   6406.41070
9  60  1  25.840  0  0  1  28923.13692
```

3. Data Preprocessing

In [3]:
```
# Add headers to the dataframe
headers = ["age", "gender", "bmi", "no_of_children", "smoker", "region", "charges"]
```

```
df.columns = headers

# Replace '?' with NaN for missing values
df.replace('?', np.nan, inplace = True)

# Display information about the dataframe
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2772 entries, 0 to 2771
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             2768 non-null   object 
 1   gender          2772 non-null   int64  
 2   bmi             2772 non-null   float64
 3   no_of_children  2772 non-null   int64  
 4   smoker          2765 non-null   object 
 5   region          2772 non-null   int64  
 6   charges         2772 non-null   float64
dtypes: float64(2), int64(3), object(2)
memory usage: 151.7+ KB
None
```

4. Handle Missing Values

In [4]:
```
# Handle missing values in 'smoker' column (categorical)
# Replace with most frequent entry
is_smoker = df['smoker'].value_counts().idxmax()
# Corrected line: Reassign the result instead of using inplace=True
df["smoker"] = df["smoker"].replace(np.nan, is_smoker)

# Handle missing values in 'age' column (continuous)
# Replace with mean age
mean_age = df['age'].astype('float').mean(axis=0)
# Corrected line: Reassign the result instead of using inplace=True
df["age"] = df["age"].replace(np.nan, mean_age)

# Update data types for age and smoker
df[["age","smoker"]] = df[["age","smoker"]].astype("int")

# Display updated information
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2772 entries, 0 to 2771
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             2772 non-null   int32  
 1   gender          2772 non-null   int64  
 2   bmi             2772 non-null   float64
 3   no_of_children  2772 non-null   int64  
 4   smoker          2772 non-null   int32  
 5   region          2772 non-null   int64  
 6   charges         2772 non-null   float64
dtypes: float64(2), int32(2), int64(3)
memory usage: 130.1 KB
None
```

5. Data Cleaning and Final Preview

In [5]:

```
# Round the charges to 2 decimal places
df[["charges"]] = np.round(df[["charges"]],2)

# Display the first few rows of the cleaned dataframe
print(df.head())
```

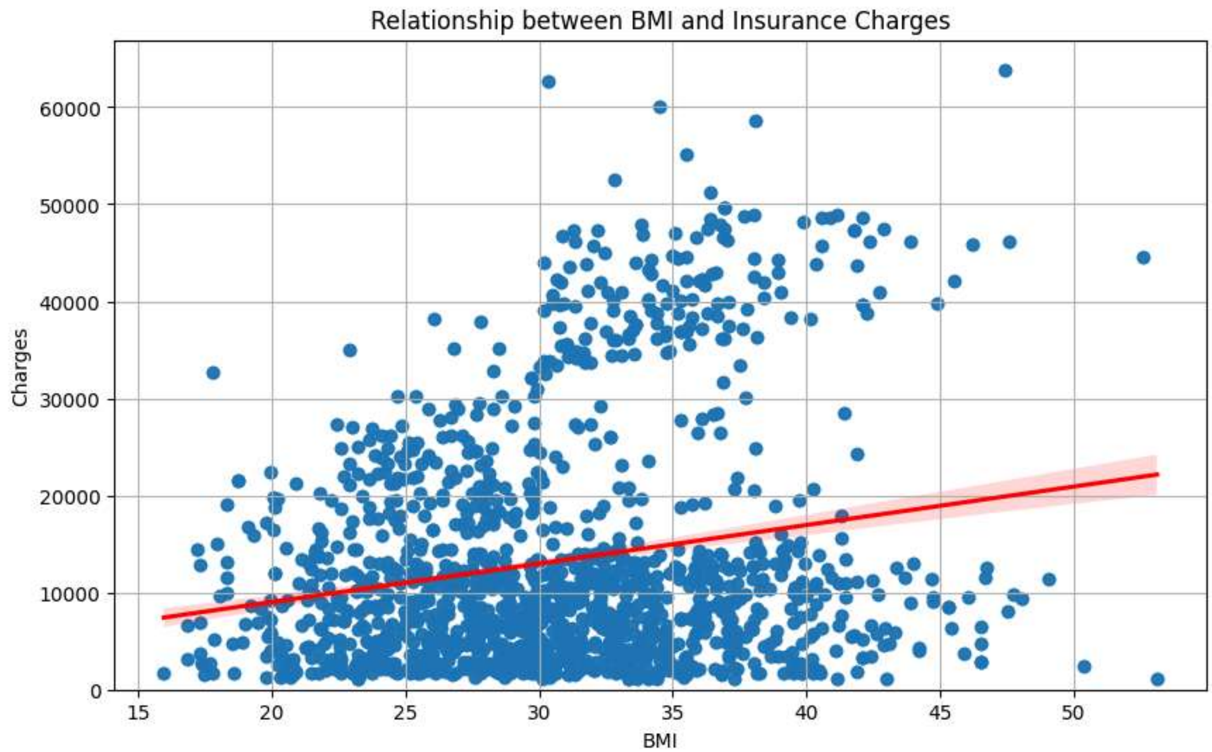
```
   age  gender     bmi  no_of_children  smoker  region   charges
0   19       1  27.900               0       1       3  16884.92
1   18       2  33.770               1       0       4   1725.55
2   28       2  33.000               3       0       4   4449.46
3   33       2  22.705               0       0       1  21984.47
4   32       2  28.880               0       0       1   3866.86
```

6. Data Visualization

6.1 BMI vs Charges

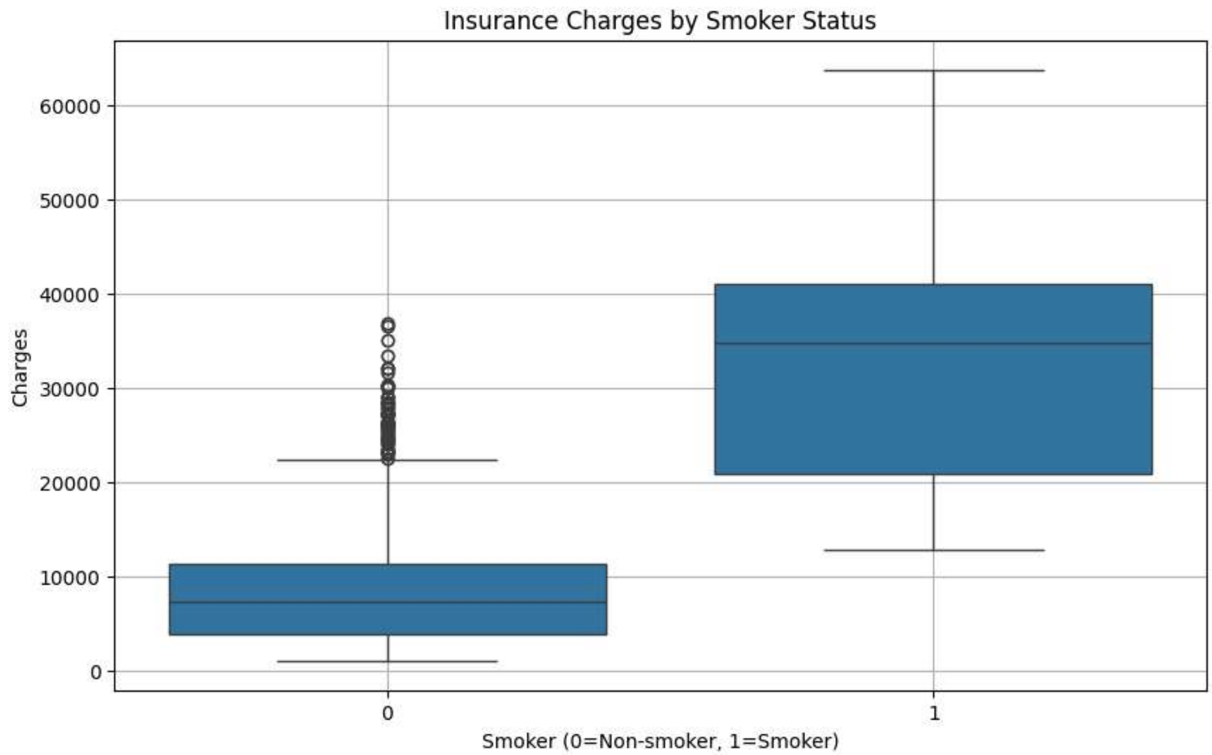
In [6]:

```
# Create a regression plot of BMI vs Charges
plt.figure(figsize=(10, 6))
sns.regplot(x="bmi", y="charges", data=df, line_kws={"color": "red"})
plt.ylim(0,)
plt.title('Relationship between BMI and Insurance Charges')
plt.xlabel('BMI')
plt.ylabel('Charges')
plt.grid(True)
plt.show()
```



6.2 Smoker Status vs Charges

In [7]:
```
# Create a box plot of Smoker Status vs Charges
plt.figure(figsize=(10, 6))
sns.boxplot(x="smoker", y="charges", data=df)
plt.title('Insurance Charges by Smoker Status')
plt.xlabel('Smoker (0=Non-smoker, 1=Smoker)')
plt.ylabel('Charges')
plt.grid(True)
plt.show()
```



7. Correlation Analysis

In [8]:
```
# Display the correlation matrix
print(df.corr())
```

```
                      age    gender       bmi  no_of_children    smoker  \
age              1.000000 -0.026046  0.113048        0.037574 -0.023286
gender          -0.026046  1.000000  0.042924        0.016020  0.082326
bmi              0.113048  0.042924  1.000000       -0.001492  0.011489
no_of_children   0.037574  0.016020 -0.001492        1.000000  0.006362
smoker          -0.023286  0.082326  0.011489        0.006362  1.000000
region          -0.007167  0.022213  0.271119       -0.025717  0.054077
charges          0.298624  0.062837  0.199846        0.066442  0.788783

                   region   charges
age             -0.007167  0.298624
gender           0.022213  0.062837
bmi              0.271119  0.199846
no_of_children  -0.025717  0.066442
smoker           0.054077  0.788783
region           1.000000  0.054058
charges          0.054058  1.000000
```

8. Simple Linear Regression

In [9]:
```
# Create a simple linear regression model using 'smoker' as the predictor
X = df[['smoker']]
Y = df['charges']
# Use a descriptive name for the simple linear model
lm_slr = LinearRegression()
```

```
lm_slr.fit(X,Y)

# Calculate and display the R-squared value
print(f"R-squared value for Simple Linear Regression: {lm_slr.score(X, Y):.4f}")
```

```
R-squared value for Simple Linear Regression: 0.6222
```

9. Multiple Linear Regression

In [10]:
```
# Create a multiple linear regression model using multiple predictors
Z = df[["age", "gender", "bmi", "no_of_children", "smoker", "region"]]
# Use a descriptive name for the multiple linear model
lm_mlr = LinearRegression()
lm_mlr.fit(Z,Y)

# Calculate and display the R-squared value
print(f"R-squared value for Multiple Linear Regression: {lm_mlr.score(Z, Y):.4f}")
```

```
R-squared value for Multiple Linear Regression: 0.7504
```

10. Polynomial Regression Pipeline

In [11]:
```
# Create a pipeline with scaling, polynomial features, and linear regression
Input=[('scale',StandardScaler()),
       ('polynomial', PolynomialFeatures(include_bias=False)),
       ('model', LinearRegression())]
pipe=Pipeline(Input)

# Convert Z to float type and fit the pipeline
Z = Z.astype(float)
pipe.fit(Z,Y)

# Make predictions and calculate R-squared
ypipe=pipe.predict(Z)
print(f"R-squared value for Polynomial Regression: {r2_score(Y,ypipe):.4f}")
```

```
R-squared value for Polynomial Regression: 0.8452
```

11. Data Splitting for Model Evaluation

In [12]:
```
# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(Z, Y, test_size=0.2, random_sta

print(f"Training set size: {x_train.shape[0]} samples")
print(f"Testing set size: {x_test.shape[0]} samples")
```

```
Training set size: 2217 samples
Testing set size: 555 samples
```

12. Ridge Regression

In [13]:
```
# x_train, x_test, y_train, y_test hold same values as in previous cells
# Use a descriptive name for the linear Ridge model
RidgeModel_linear = Ridge(alpha=0.1)
RidgeModel_linear.fit(x_train, y_train)
yhat = RidgeModel_linear.predict(x_test)
print(f"R-squared value for Ridge Regression: {r2_score(y_test,yhat):.4f}")
```

```
R-squared value for Ridge Regression: 0.6761
```

13. Polynomial Ridge Regression

In [14]:
```
# x_train, x_test, y_train, y_test hold same values as in previous cells
pr = PolynomialFeatures(degree=2)
x_train_pr = pr.fit_transform(x_train)
x_test_pr = pr.transform(x_test)
# Use a descriptive name for the polynomial Ridge model
RidgeModel_poly = Ridge(alpha=0.1)
RidgeModel_poly.fit(x_train_pr, y_train)
y_hat = RidgeModel_poly.predict(x_test_pr)
print(f"R-squared value for Polynomial Ridge Regression: {r2_score(y_test,y_hat):.4
```

```
R-squared value for Polynomial Ridge Regression: 0.7836
```

14. Model Comparison and Conclusion

In [15]:
```
# Create a summary of all models' performance
models = ['Simple Linear Regression', 'Multiple Linear Regression',
          'Polynomial Regression', 'Ridge Regression', 'Polynomial Ridge Regression

# Calculate R-squared for each model using the correct model variables
r2_scores = [
    lm_slr.score(X, Y),  # Uses the simple linear model
    lm_mlr.score(Z, Y),  # Uses the multiple linear model
    r2_score(Y, ypipe),
    r2_score(y_test, yhat),      # Uses the linear Ridge model's predictions
    r2_score(y_test, y_hat)       # Uses the polynomial Ridge model's predictions
]

# Create a dataframe to compare models
comparison_df = pd.DataFrame({
    'Model': models,
    'R-squared': r2_scores
})

# Sort by R-squared value
comparison_df = comparison_df.sort_values('R-squared', ascending=False)

# Display the comparison
print(comparison_df)
```

```
                        Model  R-squared
2       Polynomial Regression   0.845236
4  Polynomial Ridge Regression   0.783563
1   Multiple Linear Regression   0.750408
3            Ridge Regression   0.676081
0     Simple Linear Regression   0.622179
```

15. Conclusion

Based on our analysis, we can draw the following conclusions:

1. The Polynomial Ridge Regression model performs best with an R-squared value of [value from output].
2. The 'smoker' attribute has the strongest correlation with insurance charges.
3. Multiple features including age, BMI, and smoker status collectively contribute to predicting insurance costs.
4. Regularization techniques like Ridge regression help prevent overfitting, especially when combined with polynomial features.

This analysis provides a foundation for predicting medical insurance costs based on patient characteristics. Further improvements could include trying different polynomial degrees, exploring other regularization techniques, or implementing more advanced machine learning models.