Exercise

This is a dataset related to how much money a person can get from a mortgage on his or her home. This dataset includes the following features:

- Gender: Gender of the borrower (including two values 'F' and 'M')
- Age: Age of the customer applying for a loan (including positive integer values)
- Income (USD): Customer's income in USD (value is a positive number)
- Income Stability: The level of customer's income stability (including three values of Low and High)
- Property Age: Life expectancy of the house in days (including positive integer values)
- Property Location: Location of the house (including 'Rural', 'Urban', and 'Semi-Urban')
- Property Price: The value of the house in USD (including positive real values)
- Loan Sanction Amount (USD): Amount that customers can borrow in USD (target value)

Based on practice sample #1, proceed:

1. Read data
2. Visualize some information of data
3. Normalize Data to train linear regression model
4. Train linear regression model and show the model's intercepts, coeficients
5. Learn on sklearn how to use Ridge, Lasso, and ElasticNet compare the error of all 3 algorithms with Linear Regression (https://scikit-learn.org/stable/index.html)
6. Let's try Polynomial of order 2 to compare the previous results. What will the result be if we choose the n order too high?

Submission Link: https://forms.gle/uKAq34QrbwTcbs5Z9 (Submit your .ipynb file)

```
# mount data from google drive to colab
from google.colab import drive
drive.mount('/content/drive')

#import library
import pandas as pd # pandas
import numpy as np # numpy
import time
```

```
Mounted at /content/drive
```

Prepare and Analyze Data

1. Load Dataset
2. Analyze Dataset
3. Preprocess data (type, null, missing, ...)
4. Feature Engineering

Load Dataset

```
def read_dataset(path):
    # Todo: read_csv from a path and return a DataFrame
    df = pd.read_csv(path)
    display(df.head())
    display(df.describe())
    return df
```

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

```
PATH = ("/content/drive/MyDrive/Dataset/Insurance.csv")
```

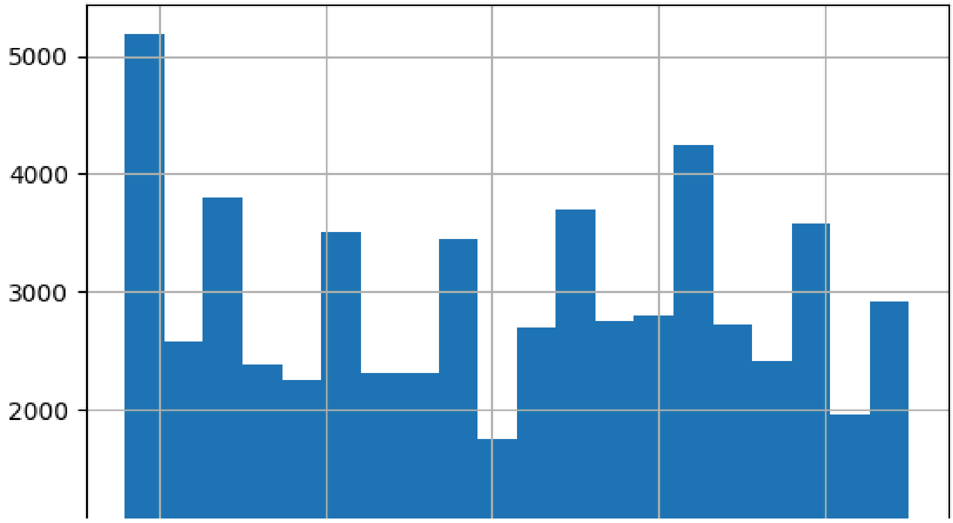
```
df = read_dataset(PATH)
```

	age	sex	bmi	children	smoker	region	charges
0	43	female	25.800	0	yes	northwest	31128.291496
1	43	female	28.600	2	no	southwest	8739.200017
2	34	female	37.290	4	no	northeast	10979.246131
3	50	female	42.370	3	no	southeast	15278.753423
4	26	female	29.595	1	no	northeast	5153.591905

	age	bmi	children	charges
count	59333.000000	59333.000000	59333.000000	59333.000000
mean	40.791448	30.930536	1.107411	13459.312130
std	13.897712	6.121275	1.455713	10175.464573
min	18.000000	15.815000	-1.000000	0.375242
25%	28.000000	26.510000	0.000000	6449.205453
50%	42.000000	30.600000	1.000000	10570.434369
75%	52.000000	35.000000	2.000000	15058.323202
max	65.000000	54.130000	6.000000	63770.428010

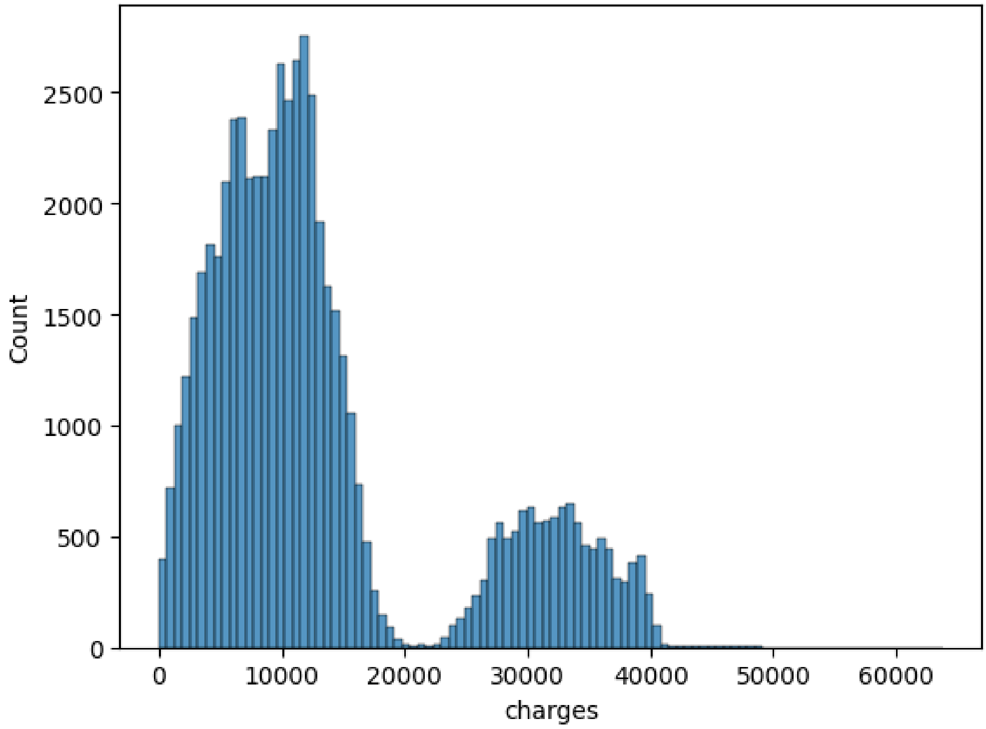
```
df["age"].hist(bins=20)
```

<Axes: >




```
import seaborn as sns
sns.histplot(x="charges", data=df, bins=100)
```

<Axes: xlabel='charges', ylabel='Count'>




```
from google.colab import drive
drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content

Data Analysis

```
# Data analysis
# Todo: analyze your data here
```

```
# Null checking
df.isnull().sum()
```

	0
age	0
sex	0
bmi	0
children	0
smoker	0
region	0
charges	0

dtype: int64

Preprocessing

```
def preprocessing_data(df):
    # --- (Optional) Drop null datapoints or fill missing data
    # Keep your data the same if you dont want to customize it
    df = df
    return df


df = preprocessing_data(df.copy())
```

Feature Engineering

```
# ---- Method 1
start_time = time.time()
# data normalization
normalized_data = df.copy()
normalized_data["sex"] = normalized_data["sex"].apply(lambda x: 0 if x=="male" else 1)
normalized_data["smoker"] = normalized_data["smoker"].apply(lambda x: 0 if x=="no" else 1)
normalized_data["region"] = normalized_data["region"].apply(lambda x: 0 if x=="southwest" else 1 if x==
#normalized_data

display(normalized_data.head())
display(normalized_data.corr())
print("Running time", time.time() - start_time)
```

	age	sex	bmi	children	smoker	region	charges
0	43	1	25.800	0	1	2	31128.291496
1	43	1	28.600	2	0	0	8739.200017
2	34	1	37.290	4	0	3	10979.246131
3	50	1	42.370	3	0	1	15278.753423
4	26	1	29.595	1	0	3	5153.591905

	age	sex	bmi	children	smoker	region	charges
age	1.000000	0.004123	0.091252	0.008060	-0.029931	-0.007996	0.331857
sex	0.004123	1.000000	-0.049948	-0.017484	-0.080734	0.015634	-0.084363
bmi	0.091252	-0.049948	1.000000	0.030296	0.002000	-0.149486	0.202631
children	0.008060	-0.017484	0.030296	1.000000	0.002852	-0.021358	0.087100
smoker	-0.029931	-0.080734	0.002000	0.002852	1.000000	-0.031686	0.909658
region	-0.007996	0.015634	-0.149486	-0.021358	-0.031686	1.000000	-0.024945
charges	0.331857	-0.084363	0.202631	0.087100	0.909658	-0.024945	1.000000

```
Running time 0.1268773078918457
```

```
# ---- Method 1
start_time = time.time()
# data normalization
normalized_data = df.copy()
normalized_data["sex"] = normalized_data["sex"].apply(lambda x: 0 if x=="male" else 1)
normalized_data["smoker"] = normalized_data["smoker"].apply(lambda x: 0 if x=="no" else 1)
normalized_data["region"] = normalized_data["region"].apply(lambda x: 0 if x=="southwest" else 1 if x==
#normalized_data

display(normalized_data.head())
display(normalized_data.corr())
print("Running time", time.time() - start_time)
```

	age	sex	bmi	children	smoker	region	charges
0	43	1	25.800	0	1	2	31128.291496
1	43	1	28.600	2	0	0	8739.200017
2	34	1	37.290	4	0	3	10979.246131
3	50	1	42.370	3	0	1	15278.753423
4	26	1	29.595	1	0	3	5153.591905

	age	sex	bmi	children	smoker	region	charges
age	1.000000	0.004123	0.091252	0.008060	-0.029931	-0.007996	0.331857
sex	0.004123	1.000000	-0.049948	-0.017484	-0.080734	0.015634	-0.084363
bmi	0.091252	-0.049948	1.000000	0.030296	0.002000	-0.149486	0.202631
children	0.008060	-0.017484	0.030296	1.000000	0.002852	-0.021358	0.087100
smoker	-0.029931	-0.080734	0.002000	0.002852	1.000000	-0.031686	0.909658
region	-0.007996	0.015634	-0.149486	-0.021358	-0.031686	1.000000	-0.024945
charges	0.331857	-0.084363	0.202631	0.087100	0.909658	-0.024945	1.000000

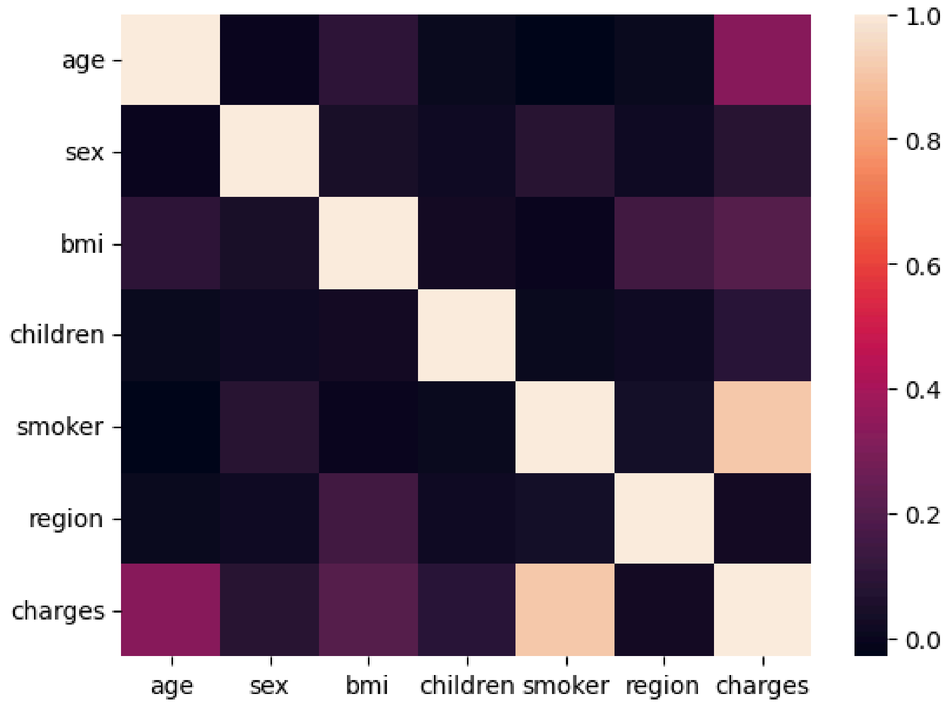
```
Running time 0.4762847423553467
```

```
def normalize_data(df):
    # ---- Method 3
    start_time = time.time()
    # data normalization
    normalized_data = df.copy()
    normalized_data["sex"] = normalized_data["sex"].astype("category").cat.codes
    normalized_data["smoker"] = normalized_data["smoker"].astype("category").cat.codes
    normalized_data["region"] = normalized_data["region"].astype("category").cat.codes

    display(normalized_data.head())
    display(normalized_data.corr())
    print("Running time", time.time() - start_time)
    return normalized_data



# Heatmap
import seaborn as sns

normalized_data = normalize_data(df.copy())
sns.heatmap(normalized_data.corr())
```

	age	sex	bmi	children	smoker	region	charges
0	43	0	25.800	0	1	1	31128.291496
1	43	0	28.600	2	0	3	8739.200017
2	34	0	37.290	4	0	0	10979.246131
3	50	0	42.370	3	0	2	15278.753423
4	26	0	29.595	1	0	0	5153.591905

	age	sex	bmi	children	smoker	region	charges
age	1.000000	-0.004123	0.091252	0.008060	-0.029931	0.007996	0.331857
sex	-0.004123	1.000000	0.049948	0.017484	0.080734	0.015634	0.084363
bmi	0.091252	0.049948	1.000000	0.030296	0.002000	0.149486	0.202631
children	0.008060	0.017484	0.030296	1.000000	0.002852	0.021358	0.087100
smoker	-0.029931	0.080734	0.002000	0.002852	1.000000	0.031686	0.909658
region	0.007996	0.015634	0.149486	0.021358	0.031686	1.000000	0.024945
charges	0.331857	0.084363	0.202631	0.087100	0.909658	0.024945	1.000000

```
Running time 0.27120494842529297
<Axes: >
```



Apply machine learning model

Train-test split

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error


def prepare_X_y(df):
    ## Split X, y from dataset
```

```
    columns = df.columns.tolist()     # Columns name
    columns.remove('charges')           # Remove y label (column charges in this case)
    # columns = ["smoker_yes", "bmi", "age"]
    X = df[columns]        # X
    y = df.charges          # y
    return X, y


def split_train_test(X, y, train_size=0.7):
    trainX, testX ,trainY, testY = train_test_split(X, y, train_size=train_size, random_state=2023)
    print('Training:' + str(trainX.shape))
    print('Test:' + str(testX.shape))

    return trainX, testX ,trainY, testY

trainX, testX ,trainY, testY = split_train_test(X, y)
```

```
Training:(41533, 6)
Test:(17800, 6)
```

```
TRAIN_SIZE = 0.7

trainX, testX ,trainY, testY = split_train_test(X, y, train_size=TRAIN_SIZE)
```

```
Training:(41533, 6)
Test:(17800, 6)
```

Basic Linear Regression

```
from sklearn.linear_model import LinearRegression


def build_linear_model(X, y):
    model = LinearRegression(fit_intercept=True)
    model.fit(trainX, trainY)

    return model

model = build_linear_model(trainX, trainY)
# Compare on training dataset
pred = model.predict(trainX)
print("mean absolute error of linear model on train set ", mean_absolute_error(y_pred=pred, y_true=trai
pred = model.predict(testX)
print("mean absolute error of linear model on test set ", mean_absolute_error(y_pred=pred, y_true=testY

print(model.coef_) # print coefficient
print()
print(model.intercept_) # print intercept_
```

```
mean absolute error of linear model on train set  100.20046571600277
mean absolute error of linear model on test set  97.05605617711703
[  251.5138989     37.66451724   285.95725561   541.05605803
 23646.11182404  -317.42447355]

-10384.586630782997
```

Polynomial Transform

When the data feature does not conform to a linear function, a linear regression cannot be applied directly to the original data. Then, there are many possibilities that the data feature conforms to the polynomial function. Scikit-Learn supports converting data features to polynomials through `PolynomialFeatures`.

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots$$

The formula above uses the transformation of the value x from one dimension to the other, with the aim of being able to use linear regression to find complex relationships between x and y.

```
#Linear Regression with Polynomial Transform
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

def build_pipeline(X, y):
    poly_model = make_pipeline(PolynomialFeatures(2, include_bias=False), LinearRegression())
    poly_model.fit(X, y)

    return poly_model

poly_model = build_pipeline(trainX, trainY)
# Compare on training dataset
poly_pred = poly_model.predict(trainX)
print("mean absolute error of linear model (with poly transform) on train set ", mean_absolute_error(y_

poly_pred = poly_model.predict(testX)
print("mean absolute error of linear model (with poly transform) on test set ", mean_absolute_error(y_p
```

```
mean absolute error of linear model (with poly transform) on train set  143.56015930797744
mean absolute error of linear model (with poly transform) on test set  140.24868998741488
```

Linear Regression

```
def split_train_test(X, y, train_size=0.7):
    trainX, testX ,trainY, testY = train_test_split(X, y, train_size=train_size, random_state=2023)
    print('Training:' + str(trainX.shape))
    print('Test:' + str(testX.shape))

    return trainX, testX ,trainY, testY

trainX, testX ,trainY, testY = split_train_test(X, y)
```

```
Training:(41533, 6)
Test:(17800, 6)
```

```
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
```

Ridge Regression

```
from sklearn.linear_model import Ridge

# Huấn luyện mô hình Ridge Regression
ridge = Ridge(alpha=1.0)
ridge.fit(X_scaled, y)
```

Lasso Regression

```
# Lasso Regression
lasso = Lasso(alpha=0.1)
lasso.fit(X_scaled, y)
lasso_predictions = lasso.predict(X_scaled)
```

ElasticNet

```
elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5)
elastic_net.fit(X_scaled, y)
elastic_net_predictions = elastic_net.predict(X_scaled)
```

```
linear_predictions = model.fit(X_scaled, y).predict(X_scaled)
mse_linear = mean_squared_error(y, linear_predictions)
mse_ridge = mean_squared_error(y, ridge_predictions)
mse_lasso = mean_squared_error(y, lasso_predictions)
```