

دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

گزارش پروژه اول درس یادگیری ماشین

مهرآذین مرزوق – ۴۰۰۳۶۱۳۰۵۵

فهرست

[پکیج‌ها و کتابخانه‌ها 3](#_Toc167743840)

[حذف داده تکراری و ستون‌های بی‌اهمیت 3](#_Toc167743841)

[عددی کردن داده‌های categorical 4](#_Toc167743842)

[حذف داده‌هایی که ویژگی NaN بسیار زیادی دارند 5](#_Toc167743843)

[بازسازی مقادیر گمشده 5](#_Toc167743844)

[ابرپارامتر‌های تابع KNN\_Imputer 5](#_Toc167743845)

[حذف داده‌های پرت 6](#_Toc167743846)

[تقسیم داده‌ها به Train و Test 7](#_Toc167743847)

[نرمال‌سازی داده‌ها 7](#_Toc167743848)

[انتخاب رگرسور مناسب 7](#_Toc167743849)

[استفاده از مدل RandomForestRegressor 8](#_Toc167743850)

[ابرپارامترهای RandomForestRegressor 9](#_Toc167743851)

[تست 9](#_Toc167743852)

# پکیج‌ها و کتابخانه‌ها

import pandas as pd

from sklearn import preprocessing

from sklearn.impute import KNNImputer

from sklearn.model\_selection import train\_test\_split

from pycaret.regression import \*

import numpy as np

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error

from sklearn.metrics import r2\_score

pandas : این کتابخانه برای دستکاری داده‌ها به شکل dataframe استفاده می‌شود.

sklearn : به‌طور کلی این کتابخانه در یادگیری ماشین استفاده می‌شود. کاربرد توابع یادشده به شرح زیر است.

preprocessing : نرمال‌سازی داده‌ها

KNNImputer : بازسازی مقادیر گمشده

train\_test\_split : تقسیم داده‌ها به train و test

RandomForestRegressor : رگرسوری به همین نام

mean\_squared\_error : یک معیار برای بررسی کیفیت رگرسور

pycaret : برای بخش انتخاب رگرسور مناسب استفاده می‌شود.

r2\_score : یک معیار دیگر برای بررسی کیفیت رگرسور

# حذف داده تکراری و ستون‌های بی‌اهمیت

df = pd.read\_csv("CreditPrediction.csv")

df.drop\_duplicates(inplace=True)

df.drop(columns='Marital\_Status', inplace=True)

df.drop(columns='CLIENTNUM', inplace=True)

df.drop(columns=['Unnamed: 19'], inplace=True)

در ابتدا پس از خواندن فایل در یک دیتافریم، داده‌های تکراری را حذف می‌کنیم. سپس ستون‌های وضعیت تاهل، شماره مشتری و ستون آخر (که هیچ‌گونه داده‌ای در آن قرار نداشت) را حذف می‌کنیم.

# عددی کردن داده‌های categorical

df['Card\_Category'] = df['Card\_Category'].replace({'Blue': 1, 'Silver': 2, 'Gold': 3, 'Platinum': 4})

df['Income\_Category'] = df['Income\_Category'].replace({'Less than $40K': 1, '$40K - $60K': 2, '$60K - $80K': 3,

                                                       '$80K - $120K': 4, '$120K +': 5, 'Unknown': np.nan})

df['Education\_Level'] = df['Education\_Level'].replace({'Uneducated': 1, 'High School': 2, 'College': 3, 'Graduate': 4,

                                                       'Post-Graduate': 5, 'Doctorate': 6,

                                                       'Unknown': np.nan})

df = pd.get\_dummies(df, columns=['Gender'])

df.reset\_index(level=None, drop=True, inplace=True)

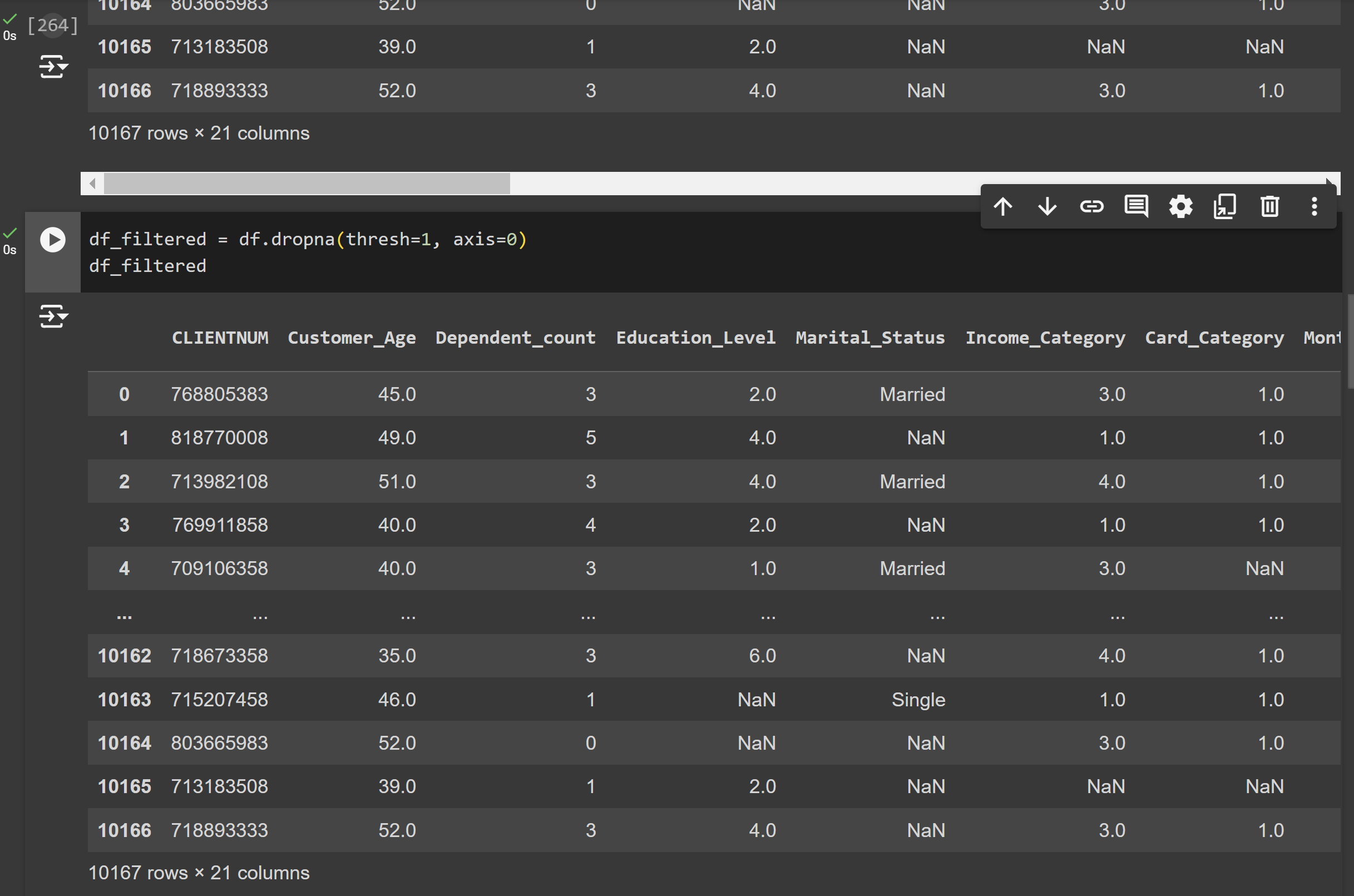
از آنجایی که رنگ کارت، نشانگر اعتبار آن است، می‌توان رنگ‌های آن را به اعداد متناظر کرد به طوری که رنگ‌ آبی با کمترین اعتبار عدد ۱ و رنگ پلاتینیوم با بیشترین اعتبار رنگ ۴ را خواهد داشت. داده های NaN همانگونه باقی می‌مانند.

داده‌های کتگوری حقوق نیز می‌تواند به اعداد متناظر شوند چرا که ترتیبی هستند و تناظر آنها به اعداد معنی‌دار است.

سطح تحصیلات هم به صورت ترتیبی است و می‌تواند به اعداد متناظر شود.

داده‌های ستون جنسیت ترتیبی نیستند پس از روش one-hot برای عددی کردن آن استفاده می‌کنیم.

# حذف داده‌هایی که ویژگی NaN بسیار زیادی دارند



برای حذف این داده‌ها از متد dropna(thresh=threshold) استفاده می‌کنیم. همانطور که عکس بالا مشاهده می‌شود، هیچ ردیفی نیست که بیش از ۱ ویژگی missed داشته باشد. پس نیازی به حذف این داده‌ها وجود ندارد و می‌توان آن داده‌ها را دوباره ساخت.

# بازسازی مقادیر گمشده

knn\_imputer = KNNImputer(n\_neighbors=4)

imputed\_data = knn\_imputer.fit\_transform(df)

imputed\_data = pd.DataFrame(imputed\_data,columns=df.columns)

یکی از ساده‌ترین استراتژی‌ها برای انجام این امر، استفاده از الگوریتم KNN می‌باشد.

## ابرپارامتر‌های تابع KNN\_Imputer

n\_neighbors : عدد انتخاب شده در این پروژه ۴ می‌باشد. چرا که با آزمایش‌هایی که انجام شد، تفاوت زیادی بین ۴ و ۵ (دیفالت) وجود ندارد و ترجیح بر این است که سربار محاسباتی با حفظ دقت، کم شود.

weight : از حالت دیفالت که «یکسان» است استفاده شده است. چرا که ۴ همسایه تعداد کمی برای وزن‌دار کردن نتایج آن‌هاست.

metric : از حالت دیفالت فاصله اقلیدسی استفاده شد.

# حذف داده‌های پرت

Q1 = imputed\_data.quantile(0.25)

Q3 = imputed\_data.quantile(0.75)

IQR = Q3 - Q1

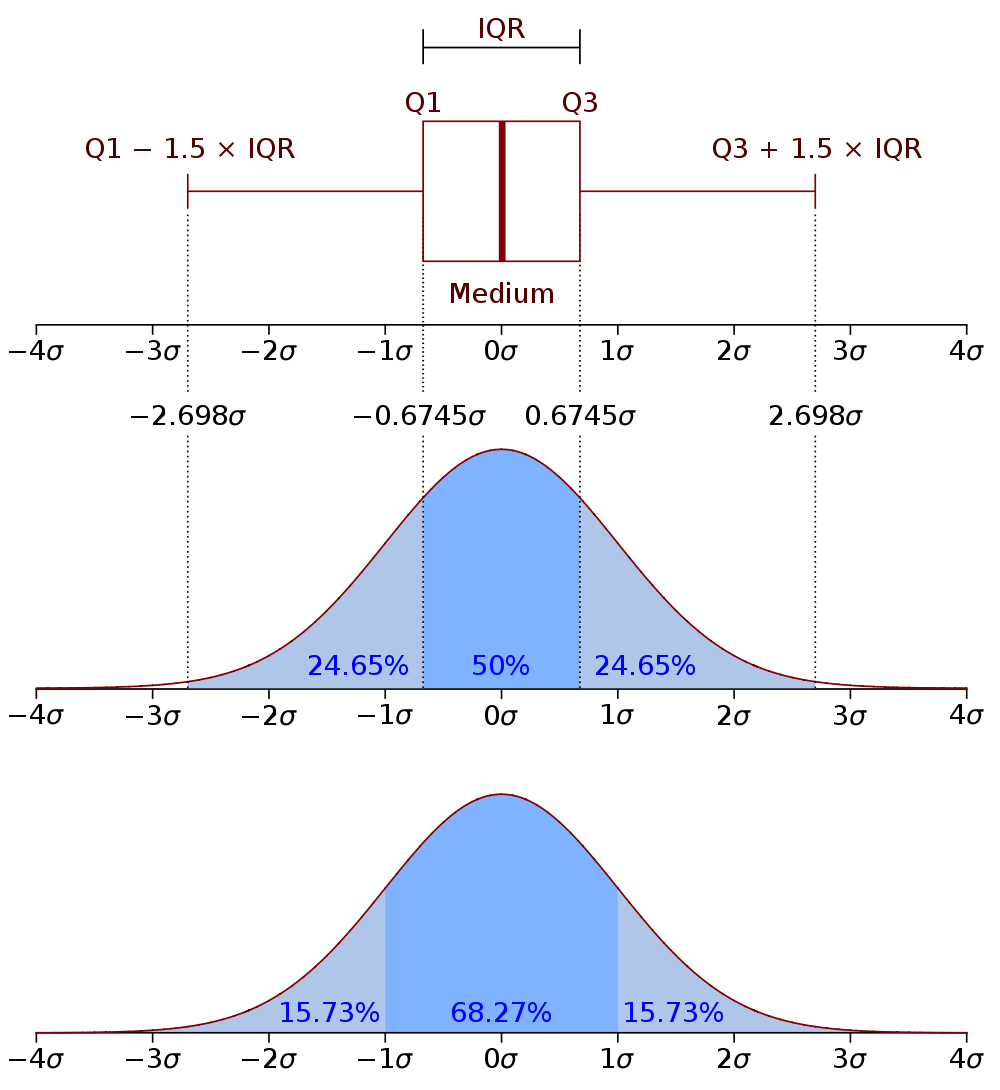
lower\_fence = Q1 - (1.5 \* IQR)

upper\_fence = Q3 + (1.5 \* IQR)

iqr\_data = imputed\_data[~((imputed\_data <lower\_fence) | (imputed\_data > upper\_fence)).any(axis=1)]

برای اینکار داده‌ها را چارک بندی می‌کنیم. چارک اول، نقطه‌ای است که ۲۵ درصد داده‌ها زیر آن قرار دارند و ۷۵ درصد داده‌ها بالای‌ آن هستند. چارک سوم،‌ دقیقا برعکس چارک اول است.

فاصله بین چارک اول و سوم را فاصله ایمن نامگذاری می‌کنیم و داده‌هایی را که فاصله معینی با چارک اول دارند و داده‌هایی که فاصله معین دیگری با چارک سوم دارند را به عنوان داده‌ی پرت در نظر می‌گیریم و حذف می‌کنیم.

دلیل انتخاب اعداد بالا، عکس روبرو است که به وضوح نشان می‌دهد در یک توزیع نرمال می‌توان از کجا به بعد، داده‌ها را پرت در نظر گرفت.

# تقسیم داده‌ها به Train و Test

data = iqr\_data.drop('Credit\_Limit', axis=1)

target = iqr\_data['Credit\_Limit']

x\_train, x\_test, y\_train, y\_test = train\_test\_split(data, target, test\_size=0.2, shuffle=True)

x\_train.reset\_index(level=None, drop=True, inplace=True)

y\_train.reset\_index(level=None, drop=True, inplace=True)

x\_test.reset\_index(level=None, drop=True, inplace=True)

y\_test.reset\_index(level=None, drop=True, inplace=True)

در این بخش به نسبت ۸۰-۲۰ داده‌های تمیزشده را به Train و Test تقسیم میکنیم. ضمنا می‌بایست قبل از هر تقسیم، داده‌ها را یک‌دور shuffle کنیم تا فقط یک نوع داده را train نکنیم.

# نرمال‌سازی داده‌ها

scaler = preprocessing.StandardScaler()

scaled\_features = scaler.fit\_transform(x\_train)

normalized\_train = pd.DataFrame(scaled\_features, columns=[x\_train.columns])

scaled\_features = scaler.transform(x\_test)

normalized\_test = pd.DataFrame(scaled\_features, columns=[x\_test.columns])

برای این‌کار از اسکیلر استاندارد استفاده می‌کنیم؛ چرا که در قبل نیز فرض کردیم که داده به صورت نرمال پخش شده است.

# انتخاب رگرسور مناسب

s = setup(iqr\_data, target='Credit\_Limit', session\_id=123)

best = compare\_models()

برای این‌کار، همه x ها و همچنین همه y های پیش‌پردازش شده را استفاده می‌کنیم. هدف از این کار استفاده از تابع setup() است. این تابع در حقیقت مدل‌های متفاوت را روی داده ما train می‌کند و MSE و R\_Squared و بقیه معیار‌ها را بیان می‌کند.

A screenshot of a computer

Description automatically generated

نتایج نشان می‌دهد استفاده از Random Forest Regressor می‌تواند ایده خوبی باشد.

# استفاده از مدل RandomForestRegressor

model = RandomForestRegressor(n\_estimators=100, max\_depth=11, random\_state=42)

model.fit(normalized\_train,y\_train)

y\_pred = model.predict(normalized\_test)

MSE = mean\_squared\_error(y\_test, y\_pred)

R2 = r2\_score(y\_test, y\_pred)

print(R2)

print(MSE)

در این بخش از مدل یاد شده استفاده می‌کنیم و MSE و R2 را به‌دست می‌آوریم.

## ابرپارامترهای RandomForestRegressor

n\_estimators : تعداد درخت‌های مورد استفاده در الگوریتم. هرچه تعداد درخت‌ها بیشتر باشد، دقت بالاتر می‌رود. البته باید به overfitting نیز دقت کنیم. برای این‌کار از ۱۰۰ که دیفالت این تابع است شروع می‌کنیم و با آزمون و خطا عدد مناسب را به‌دست می‌آوریم. در این پروژه، مقدار مناسب همان ۱۰۰ می‌باشد.

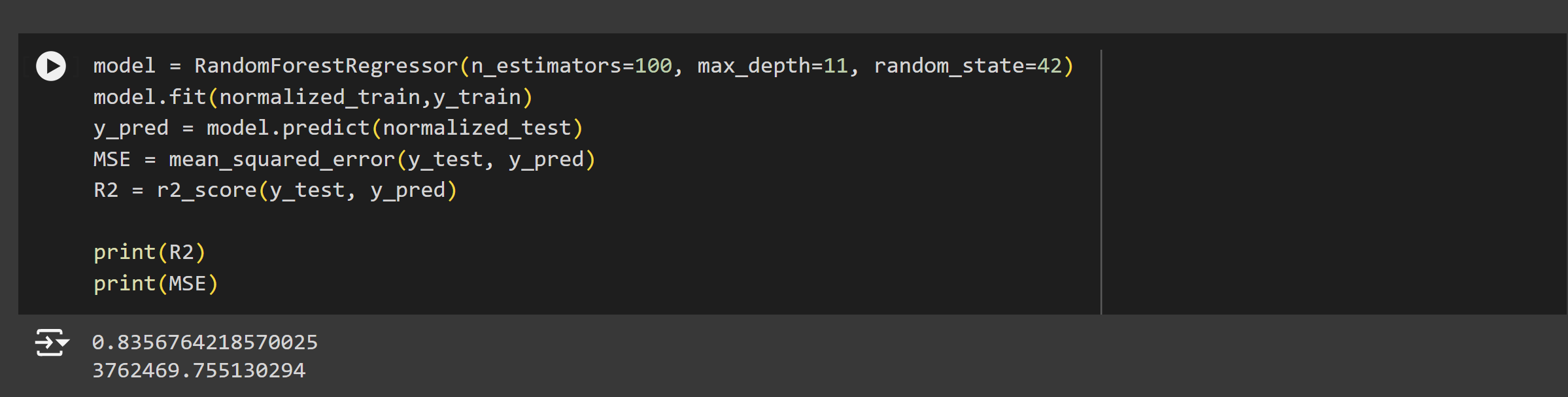
max\_depth : این پارامتر، عمق درختان را تعیین می‌کند. که دیفالت آن None است و باید تعیین شود. هرچه عمق بیشتر باشد، نتایج دقیق‌تر خواهد بود. البته باید overfitting را نیز در نظر داشته باشیم. برای این‌کار می‌بایست مثل قبل، با آزمون و خطا و بررسی MSE و R2 ، عدد مناسب را تعیین کنیم. عمق مناسب به‌دست آمده در این پروژه برابر با ۱۱ می‌باشد.

random\_state : این پارامتر میزان رندوم بودن درخت‌ها را معرفی می‌کند. مثلا این‌که در هر نود، از چه زیرمجموعه‌ای از ویژگی‌ها استفاده شود و یا bootstrapping را True یا False می‌کند که به شکلی دیگر، رندوم بودن درخت‌ها را تعیین می‌کند. استفاده از این ابرپارامتر باعث می‌شود که بقیه ابرپارامتر‌ها در این پروژه لازم نباشند.

# تست

در این بخش با استفاده از چند بار ران کردن برنامه، MSE و R2 های متفاوت را نمایش می‌دهیم.

تست اول



تست دوم

A screenshot of a computer

Description automatically generated

تست سوم

A screenshot of a computer program

Description automatically generated

تست چهارم

A screenshot of a computer program

Description automatically generated

میانگین MSE این مدل در حدود ۴۲۰۰۰۰۰۰ و R2 نیز در حدود ۰.۸۱ می‌باشد.