



**دانشگاه اصفهان**

**دانشکده مهندسی کامپیوتر**

**گزارش پروژه اول درس یادگیری ماشین**

**مهرآذین مرزوق - ۴۰۰۳۶۱۳۰۵۵**

## فهرست

۳	پکیج‌ها و کتابخانه‌ها
۳	حذف داده تکراری و ستون‌های بی‌اهمیت
۴	عددی کردن داده‌های categorical
۵	حذف داده‌هایی که ویژگی NaN بسیار زیادی دارند
۵	بازسازی مقادیر گم‌شده
۵	ابزارهای تابع KNN_Imputer
۶	حذف داده‌های پرت
۷	تقسیم داده‌ها به Train و Test
۷	نرمال‌سازی داده‌ها
۷	انتخاب رگرسور مناسب
۸	استفاده از مدل RandomForestRegressor
۹	ابزارهای RandomForestRegressor
۹	تست

## 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)
```

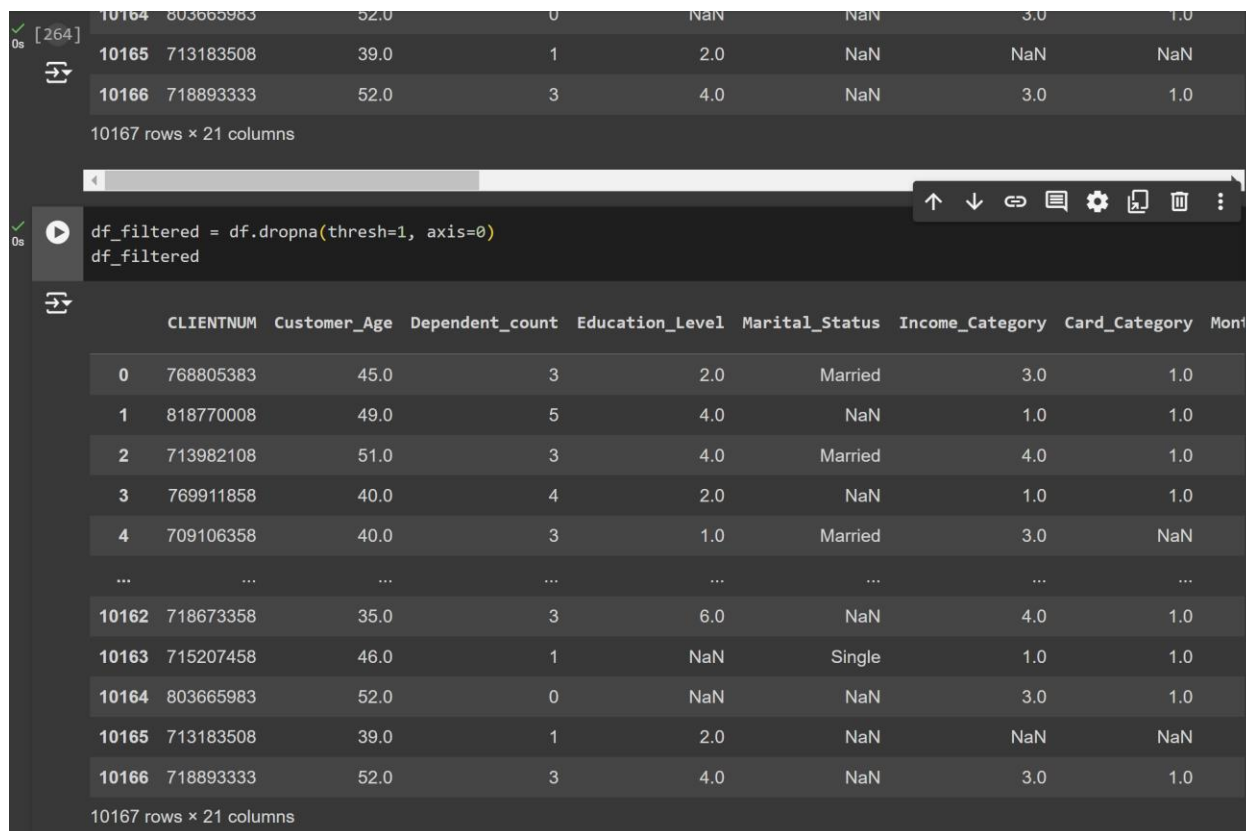
در ابتدا پس از خواندن فایل در یک دیتافریم، داده‌های تکراری را حذف می‌کنیم. سپس ستون‌های وضعیت تاهل، شماره مشتری و ستون آخر (که هیچ‌گونه داده‌ای در آن قرار نداشت) را حذف می‌کنیم.

## 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. داده‌های 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



```
df_filtered = df.dropna(thresh=1, axis=0)
df_filtered
```

	CLIENTNUM	Customer_Age	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Monthly_Payment
0	768805383	45.0	3	2.0	Married	3.0	1.0	
1	818770008	49.0	5	4.0	NaN	1.0	1.0	
2	713982108	51.0	3	4.0	Married	4.0	1.0	
3	769911858	40.0	4	2.0	NaN	1.0	1.0	
4	709106358	40.0	3	1.0	Married	3.0	NaN	
...	...	...	...	...	...	...	...	...
10162	718673358	35.0	3	6.0	NaN	4.0	1.0	
10163	715207458	46.0	1	NaN	Single	1.0	1.0	
10164	803665983	52.0	0	NaN	NaN	3.0	1.0	
10165	713183508	39.0	1	2.0	NaN	NaN	NaN	
10166	718893333	52.0	3	4.0	NaN	3.0	1.0	

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.

### ابریارامترهای 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 وزن دار 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)

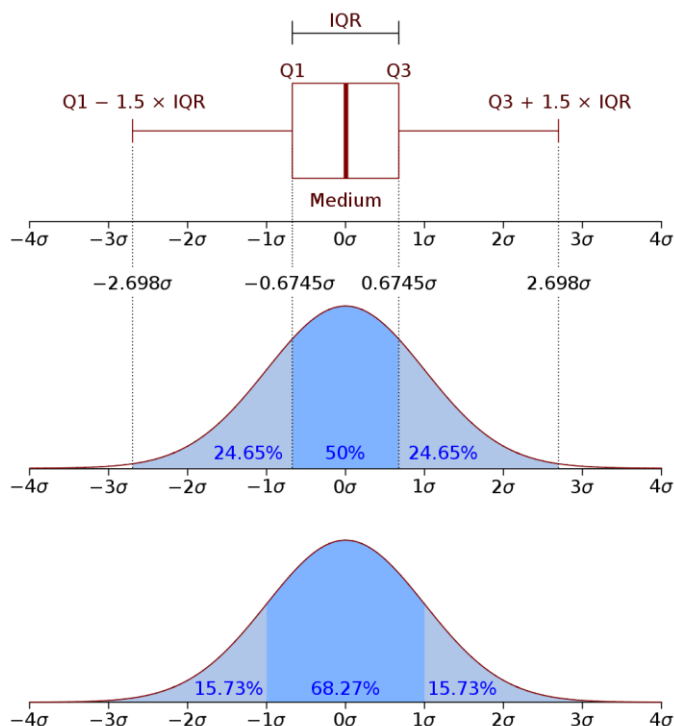
iqr_data = imputed_data[~((imputed_data < lower_fence) | (imputed_data >
upper_fence)).any(axis=1)]
```

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، دقیقاً 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. ضمناً 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 این کار 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 این کار، 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.

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
rf	Random Forest Regressor	929.8480	4621520.3734	2158.7371	0.8058	0.3643	0.2428	
lightgbm	Light Gradient Boosting Machine	954.6461	4623511.7713	2139.5796	0.8056	0.3658	0.2435	
gbr	Gradient Boosting Regressor	1102.8997	4684515.9782	2155.9430	0.8026	0.3648	0.2735	
et	Extra Trees Regressor	891.3150	4795622.3660	2180.3845	0.7977	0.3643	0.2522	
xgboost	Extreme Gradient Boosting	1034.1706	5012942.9750	2229.4069	0.7890	0.4024	0.2674	
ada	AdaBoost Regressor	1882.7261	8020816.0360	2826.5304	0.6603	0.4957	0.5046	
dt	Decision Tree Regressor	1265.6610	8695707.3722	2932.0674	0.6306	0.4760	0.3079	
ridge	Ridge Regression	2317.4474	10682724.7000	3258.1438	0.5491	0.7502	0.6192	
llar	Lasso Least Angle Regression	2319.1887	10680569.7000	3257.8642	0.5491	0.7476	0.6204	
lasso	Lasso Regression	2319.1873	10680555.7000	3257.8621	0.5491	0.7477	0.6204	
lr	Linear Regression	2320.5661	10682815.7000	3258.1878	0.5490	0.7521	0.6211	
lar	Least Angle Regression	2347.8418	10775811.5000	3272.8841	0.5449	0.7688	0.6315	
en	Elastic Net	2881.8839	16239923.7000	4019.8383	0.3138	0.6469	0.7333	
huber	Huber Regressor	2757.0771	17203417.1157	4135.5521	0.2738	0.6186	0.5872	
br	Bayesian Ridge	3643.6787	23555240.8000	4845.9805	0.0020	0.8015	0.9764	
omp	Orthogonal Matching Pursuit	3643.5425	23594096.6000	4850.1743	0.0002	0.8051	0.9850	
dummy	Dummy Regressor	3649.4126	23631119.8000	4854.4463	-0.0018	0.8060	0.9873	
knn	K Neighbors Regressor	3724.6244	26130004.0000	5105.8224	-0.1103	0.8286	0.9596	
par	Passive Aggressive Regressor	4889.6038	33776922.1542	5760.7866	-0.4338	1.0226	1.5330	

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 به دست می‌آوریم.



## ابریارامترهای 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 این کار از ۱۰۰ که دیفالت این تابع است شروع می‌کنیم و با آزمون و خطا عدد مناسب را به دست 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 این کار می‌بایست مثل قبل، با آزمون و خطا و بررسی **MSE** و **R2**، عدد مناسب را تعیین کنیم. عمق مناسب به دست آمده در این پروژه برابر با ۱۱ می‌باشد.

**random\_state**: این پارامتر میزان رندوم بودن درخت‌ها را معرفی می‌کند. مثلاً این که 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 ابریارامتر باعث می‌شود که بقیه ابریارامترها 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

```
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)
```

```
0.8356764218570025
3762469.755130294
```

### تست دوم

```
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)
```

```
0.813429803579317
4415102.198339303
```

### تست سوم

```
✓ [402] model = RandomForestRegressor(n_estimators=100, max_depth=11, random_state=42)
8s 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)
```

0.8187341897832683  
4488746.250707175

### تست چهارم

```
✓ [410] model = RandomForestRegressor(n_estimators=100, max_depth=11, random_state=42)
5s 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)
```

0.8147721648384181  
4397794.3868914815

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