دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

گزارش پروژه‌ی اول مبانی یادگیری ماشین

مدل رگرسیون برای پیشبینی اعتبار مالی افراد

پدیدآورنده:

محمد امین کیانی

۴۰۰۳۶۱۳۰۵۲

دانشجوی کارشناسی، دانشکده‌ی کامپیوتر، دانشگاه اصفهان، اصفهان،
Aminkianiworkeng@gmail.com

استاد راهنما: جناب اقای دکتر کیانی

نیمسال دوم تحصیلی ۰۳-۱۴۰۲

فهرست مطالب

مستندات

۱-مسئله و تحلیل کلی آن:

- برای این پروژه، می‌توان از الگوریتم‌های رگرسیون خطی، رگرسیون لجستیک و یا RandomForestRegressor و... استفاده کرد. چون در پروژه فقط از انواع رگرسور ها میتوان استفاده کرد پس انتخاب کدل ها کمی محدود تر شده و به علت سرسخت بودن دیتاست های داده شده، با خطی و چندجمله ای نمی توان به نتایج و فیت شدن خوبی رسید (خطای ده ها میلیونی و به عبارتی ساده دقت ۰/۶ یعنی مثلا ۶۰ درصد میرسیم که اصلا خوب نبود و مجبور به تغییر مدل شدم اما کد های همین روش را نیز به صورت کامل در فایل کد هایم به صورت کامنت تحت عنوان مدل شکست خورده قرار دادم.) و طبق نکات کلاسی میدانیم که در الگوریتم های درخت تصمیم، رندوم فارست در جاهایی خوب عمل می کند که ویژگی های کتگوریکال زیاد است و در مدل ما موثر! که دقیقا در این دیتاست فقط با دیتا های عددی سر و کار نداریم بلکه استرینگ هایی نیز داریم که باید ابتدا عدد شوند و ... . پس اینگونه شد که رندوم فارست را به عنوان یک مدل سریع و خوب و بهینه با دقت بالا(برای تصور بهتر یعنی ۸۸درصد و خطای زیر ده میلیون استفاده کردم.)

```
model = RandomForestRegressor(n_estimators=100, criterion='squared_error',
max_depth=11,
                              min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0,
                              max_features=1.0, max_leaf_nodes=None,
min_impurity_decrease=0.0,
                              bootstrap=True, oob_score=False, n_jobs=None,
random_state=30,
                              verbose=0, warm_start=False, ccp_alpha=0.0,
max_samples=None)
```

۲-پیش‌پردازش داده:

- نیاز است که داده‌های خود را تمیز و مقادیر خالی را پر کرد.

```
# Remove duplicates
dt.drop_duplicates(inplace=True)

#inplace ==> changes submit on main data

# Remove rows with all elements as NaN(empty)
dt.dropna(how="all", inplace=True) # axis = 0
# Remove columns with all elements as NaN(empty)  +  # Drop the 'Unnamed: 19'
dt.dropna(axis=1, how="all", inplace=True)

# Drop the 'CLIENTNUM' column
dt.drop("CLIENTNUM", axis=1, inplace=True)
dt.drop("Months_on_book", axis=1, inplace=True)
dt.drop("Total_Ct_Chng_Q4_Q1", axis=1, inplace=True)
# dt.replace("Unknown", np.NAN, inplace=True)


# # Drop the 'Unnamed: 19' column
# dt.drop("Unnamed: 19", axis=1, inplace=True)
```

ابتدا با دروپ ها تکراری ها و خالی ها و چند ستون بدرد نخور(طبق کواریانس تشخیص می دهیم کدام ها از لیبل کریدیت لیمیت دور تر یعنی به عدد 0 نزدیک تر اند و انها را حذف می کنیم.)

```
dt.corr()
```

همچنین، نیاز به انجام فرآیندهای نرمال‌سازی و انکودینگ نیز می باشد.

```
#hot coding  ==> # r2 ==> 86%    mse ==> 11 m
# # Encoding categorical data
#converts categorical columns (whose values are categorical) to dummy variables.
# This causes each category to become a new column, and a value of 1 in that
column indicates
```

```
dt = pd.get_dummies(dt)
dt.info()
```

انکد کردن داده ها به روش هات اینکودینگ. چرا؟ چون سایر روش ها از جمله دستی مپ کردن فیچر ها را نیز امتحان کردم و با هر تغییرات و داستانی انقدر که باید تغییری در کاهش خطا نداشت و به خوبی get_dummies عمل نمی کرد. درواقع دامی حدود ۲ الی ۳ میلیون ارور کمتری تولید میکند زیرا عدد های نسبت داده شده ی مپ کردن را که رندوم و با تجربه میگذاریم را ندارد و خود به خوبی معیار هارا مناسب تر تعیین میکند پس من این روش را برگزیدم که سریع تر و قوی تر و راحت تر بود اما مپ کردن را نیست به صورت کامنت قرار دادم و هر بار بخواهم با هر کدوم از یکی از این دو روش مدل را پیاده سازی و دو مقدار خطای متفاوت برای مقایسه میتوانم بدست بیاورم. سپس برای نرمال سازی نیاز به انجام هم زمان سه فرایند اسپیلیت و اسکیل و پر کردن Nanها بود. چرا؟ چون نباید لیبل ها را نیز وارد بازی کنیم و تنها باید روی ورودی ها این اعمال صورت بگیرد و اصلا اگر اینکار را نکنیم و مثل کد های قبلی ام این اعمال روی همه ی داده ها صورت بگیرد دقت به شدت وحشتناک بالا و فیت شدن خوبی دارد که مارا به گمراهی برده و فکر میکنیم مدل بسیار قوی و خوبی زده ایم درحالی که چنین نیست و لیبل وارد بازی شده به تست و ترین ها تقلب می دهد تا بهتر فیت شود که اشتباه است. پس مجبوریم این اعمال را داخل یک بلوک انجام بدیم و به درستی اسپلیت صورت گیرد و متاسفانه از دقت فرضی ۹۹ درصدی خود بکاهیم!(کد های این فیت شدن عالی رو نیز کامنت کردم که یادم بماند عجب مدل با دقت و خوبی داشتم... . حیف که روش درستی نبود.) برای نرمال سازی نیز از StandardScaler استفاده کردم زیرا احتمال وقوع ها در جامعه اماری ما متفاوت بوده و بهتر از ریسک MinMax است و توزیع گوسی -۳ تا ۳ داریم که

طبق نکات گفته شده در کلاس این کار برای این مسئله بهتر است. برای پر کردن خالی ها نیز از KNNImputer استفاده کردم زیرا یکی از بهترین روش های پر کردن خالی ها بر اساس میانگین است و باتوجه به ابرپارامتر همسایگی دیدم که عدد ۷ بهترین گزینه برای این مسئله بوده و با مقادیر دیگر میزان خطا زیاد میشد.

```
# Step 1: Split the dataset
X = dt.drop('Credit_Limit', axis=1)
y = dt['Credit_Limit']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=45)

# Step 2: Impute missing values
imputer = KNNImputer(n_neighbors=7, weights="uniform", metric="nan_euclidean")
X_train_filled = imputer.fit_transform(X_train)
X_train_filled = pd.DataFrame(X_train_filled, columns=X_train.columns)

# Step 3: Scale only the training data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_filled)
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)

# Step 4: Impute the missing values in the test data
X_test_filled = imputer.transform(X_test)
X_test_filled = pd.DataFrame(X_test_filled, columns=X_test.columns)

# Step 5: Scale the test data using the scaler fitted on the training data
X_test_scaled = scaler.transform(X_test_filled)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)

# Calculate Mutual Information
mi_scores = mutual_info_regression(X_train_scaled, y_train)

# Select features based on MI scores
# Here we assume a threshold of 0.01 for demonstration purposes
selected_features = X_train.columns[mi_scores > 0.01]
X_train_selected = X_train_scaled[selected_features]
X_test_selected = X_test_scaled[selected_features]
```

۳-انتخاب ویژگی‌ها:

- بررسی میشود که کدام ویژگی‌ها بیشترین تأثیر را بر اعتبار مالی دارند و از آن‌ها برای مدل خود استفاده می کنیم.

یعنی بعد از اسکیل شدن از Mutual Information استفاده می کنیم یعنی یک به یک فیچر مقایسه و میزان شباهت هارا تعیین می کند و عکس ان نیز با Chi-Square میتوان میزان عدم وابستگی را نیز محاسبه که البته چون سرعت اجرای کد را کم کرد و تاثیر بسیار کمی در کاهش خطا داشت پس بهینه نبود و بدرد نمی خورد اما میوچال اینفورمیشن حدودا بیش از نیم میلیون از خطا کم کرد که پس به خوبی عمل کرده و در اینجا به کار ما امد.

```
# Calculate Mutual Information
mi_scores = mutual_info_regression(X_train_scaled, y_train)

# Select features based on MI scores
# Here we assume a threshold of 0.01 for demonstration purposes
selected_features = X_train.columns[mi_scores > 0.01]
X_train_selected = X_train_scaled[selected_features]
X_test_selected = X_test_scaled[selected_features]


# # Find the minimum value in X_train_selected
# min_value = np.min(X_train_selected)

# # Add the absolute value of min_value to all features
# X_train_non_negative = X_train_selected + np.abs(min_value)
# X_test_non_negative = X_test_selected + np.abs(min_value)

# # Assuming X_train_selected contains both numerical and categorical features
# # Select the top k features based on Chi-Square scores
# k = 5  # Choose the desired number of features
# selector = SelectKBest(chi2, k=k)
# X_train_chi2 = selector.fit_transform(X_train_non_negative, y_train)
# X_test_chi2 = selector.transform(X_test_non_negative)
```

٤-تقسیم داده:

- داده‌های خود را به دو دسته‌ی آموزشی و آزمون تقسیم کرده تا ارزیابی صحیحی روی مدل خود انجام شود.

اینکار همان اسپلیت بود که گفتم به علت درست کار کردن مدل مجبور به همزمانی انجام ان با پر کردن داده و اسکیل کردن بودم و توضیحات را در بخش های قبلی داده ام که چرا و چطور...

٥-ساخت مدل:

- انتخاب مدل مناسب و سپس آموزش مدل با استفاده از داده‌های آموزشی.

همان طور که در مقدمه به طور مفصل شرح دادم که سوال چیست و چرا و چگونه به مدل رندوم فارست دست یافتیم، ان را به عنوان بهترین مدل این مسئله در نظر گرفته که البته بعد از ان طبق نکات کلاسی گفته شده سعی کردم از ادابوست استفاده کنم تا مدل را ببینم ایا میتوانم بهبود دهم اما با اجرای ان نیم میلیون به خطا ها افزوده شد و باز هم فهمیدم که همان رندوم فارست برای ما بهتر است!

که البته بعد از فیت کردن مدل از **حذف** خطاهای بازگشتی نیز طبق نکات کلاسی بهره گرفتم و در حد چندین هزار از خطا کاهش داد که خب خوب و است اما کافی نیست ولی خب ان را قرار دادم.

```
rfe = RFE(model, n_features_to_select=17)
```

٦-ارزیابی مدل:

- استفاده از معیارهایی مانند دقت، صحت، حساسیت و... برای ارزیابی عملکرد مدل خود. حال در اینجا چون با کلاس و کانفیوژن ماتریس ها سروکار نداریم و در داده های پیوسته رگرسیونی دقت را به طور تصوری میتوان یکی از انواع ان یعنی r۲_score در نظر گرفت و خطا را میانگین مربعات خطاها یا MSE استفاده کرد که البته برای بررسی بهتر و بیشتر معیار های فرعی دیگری که قدرمطلق دار و جذر دار مثل RMSE و MAE نیز محاسبه کردم تا با انواع ارزیابی ها سروکار داشته و نتایج و پیشبینی بهتری داشته باشم.

```
# Model evaluation
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import math

scores = cross_val_score(model, X_train_filled, y_train, cv=15)
print("Cross-validated scores:", scores)

#----------------------------------------------------------------

r2 = r2_score(y_test, y_pred)
MSE = mean_squared_error(y_test, y_pred)
MAE = mean_absolute_error(y_test, y_pred)

print(f"R2 error:", {r2})
print(f"Mean Squared Error(MSE):", {MSE})
print(f"R Mean Squared Error(RMSE):", {math.sqrt(MSE)})
print(f"Mean Absolute Error(MAE):", {MAE})

print(y_pred)
print(y_test)
print(y_train)
```

در انتها لیست لیبل های مورد انتظار و بدست اماده و اموزشی را(صرفا برای دیدن انکه در چه وضعیت هستیم) را چاپ کردم تا اندرفیت و اورفیت شدن مدل را

تشخیص دهم و به این گودال اموزشی دچار نشود که خداروشکر فیت شد و به این مشکلات نخورده است و به خوبی از ان ها پرهیز شده و نرمال است.

۷-تنظیم پارامترها:

- پس از ارزیابی، ممکن است نیاز به تنظیم پارامترهای مدل خود داشته باشید.

```
model = RandomForestRegressor(n_estimators=100, criterion='squared_error',
max_depth=11,
                              min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0,
                              max_features=1.0, max_leaf_nodes=None,
min_impurity_decrease=0.0,
                              bootstrap=True, oob_score=False, n_jobs=None,
random_state=30,
                              verbose=0, warm_start=False, ccp_alpha=0.0,
max_samples=None)
```

و بله در انتها نیز با تغییر ان میزان از پارامتر های مدل و انچه که انقدر اجرا را کند نکند و یا سبب خطای بیشتر نشود، به این نتیجه رسیدم که باید برخی را با تست و تجربه و خواندن داک اینکه هرکدام چه کاری می کند، تغییر داده و در حدود نیم میلیون با دستکاری ان ها، خطا ها تغییر می کند.

البته باید توجه داشت تغییر زیاد و حتی دادن مقادیر خیلی عجیب در خراب کردن مدل بسیار موثر و ممکن است چندین میلیون خطا بر دوش ما اضافه کند. پس با ظرافت تمام دست به تغییر تا انجا که توانستم زدم.

۸-به‌روزرسانی و بهبود:

- مدل خود را به صورت منظم به‌روزرسانی کنید و از روش‌های بهبود عملکرد آن استفاده کنید.

از cross_val_score برای تست داده های ترین استفاده کردم تا از اورفیت شدن جلوگیری کنم و بهبودی در کد صورت گیرد و همانطور که در بخش های قبلی گفتم طبق پرینت نتایج نه اورفیت و نه اندرفیت شد و این برای مدل ما بسیار خوب است!

```
scores = cross_val_score(model, X_train_filled, y_train, cv=5)
print("Cross-validated scores:", scores)
```

جالب است بدانیم تغییر CVدر زمان اجرای کد بسیار موثر بوده و میتواند از دقیقه تا ثانیه ران تایم ما را درست خوش تغییر کند که با تست فراوان مقادیر مختلف بهمیدم بهترین مقدار برای این پارامتر ولیدیت کردن عدد ۵ است!

از دو روش زد اسکور و باکس پلات و چارک گیری نیز برای Outlier Handling نیز استفاده کردم اما طبق توضیحات کلاسی فرموده بودین نباید داده های پرت را حذف کرد و البته در مدل من بود و نبود ان ها تاثیر چندان زیادی نداشت و کمک خاصی حتی نمیکرد پس ان ها را برداشته اما به صورت کامنت نگهداشتم که مراحل الگوریتمی ان نمایان باشد.

۹-استفاده نمودار ها و اشکال:

- برای تفسیر عملکرد کلی مدل از پلات های پایتون کمک گرفتم و با رسم دو نمودار میزان دقت مدل (یعنی چقدر انتظار و بدست امده باهم تفاوت) و میزان خطای مدل (یعنی تفاضل انتظار و بدست امده یعنی باقی مانده ها با مورد انتظاری

که داشتم) یک تقریب خوبی از نمای کلی عملیات مدل سازی گرفته و به طور چشمی و طبق معیارهای ارزیابی شده حدودا ۸۸.۵ درصد مدل به درستی کار کرده که میزان خوبی است.

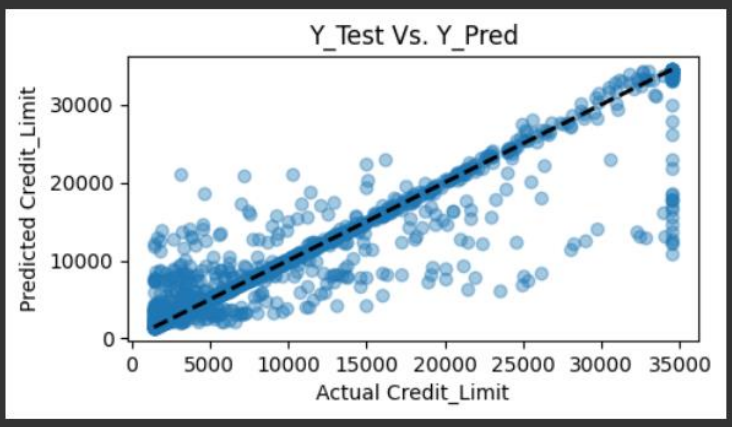
```
import matplotlib.pyplot as plt


# test ==> actual   /    pred ==> predicted
leftovers = y_test - y_pred
#same as
plt.figure(figsize=(5,2.5))
plt.scatter(y_test, y_pred, alpha=0.4)
plt.title("Y_Test Vs. Y_Pred")
plt.xlabel("Actual Credit_Limit")
plt.ylabel("Predicted Credit_Limit")
plt.plot([y.min(), y.max()], [y.min(), y.max()], "k--", lw=2)
plt.show()

#-------------------------------------------------------------------------------
-------
#errors
plt.figure(figsize=(5,2.5))
plt.scatter(y_pred, leftovers, alpha=0.4)
plt.title("LeftOvers")
plt.xlabel("Predicted Credit_Limit")
plt.ylabel("leftovers")
plt.hlines(y=0, xmin=y_pred.min(), xmax=y_pred.max(), colors="black",
linestyles="--")
plt.show()
```

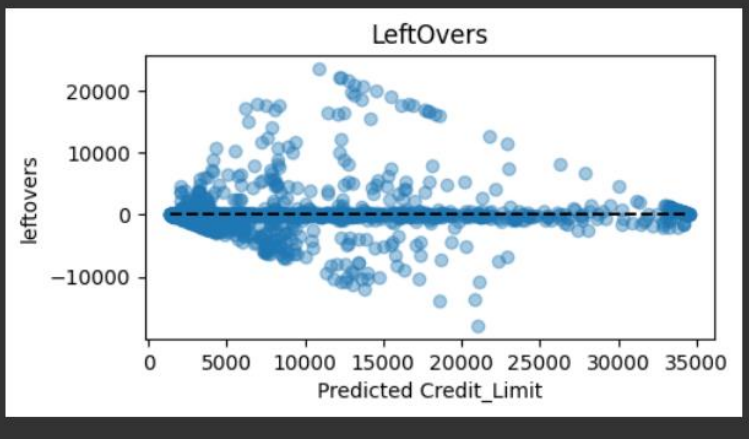
mohammad_amin_kiani4003613052.ipynb

mohammad_amin_kiani4003613052.ipynb > ...

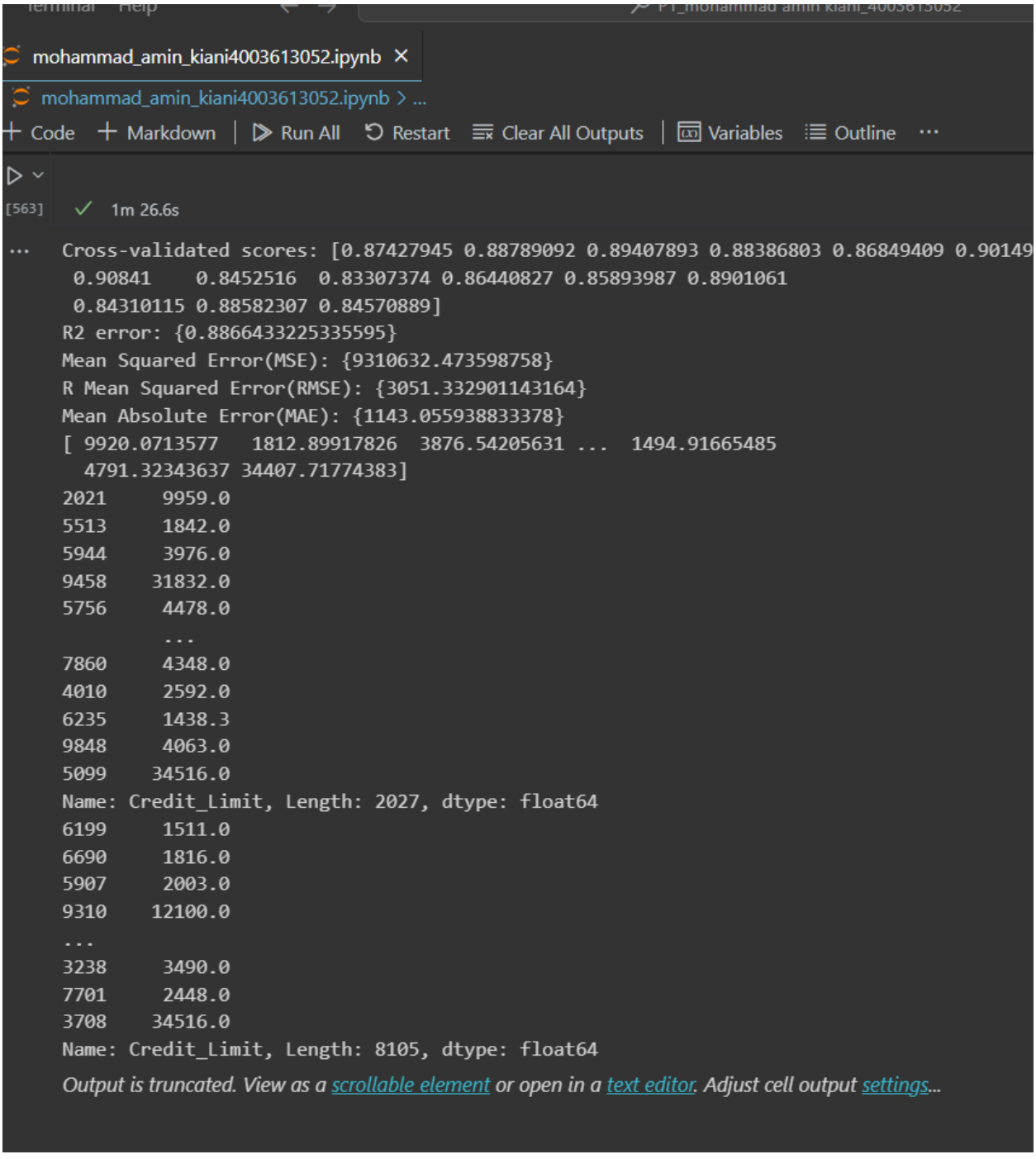
+ Code + Markdown | Interrupt Restart Clear All Outputs Go To | Variables Outline ...

```
plt.hlines(y=0, xmin=y_pred.min(), xmax=y_pred.max(), colors="black", linestyles="--")
plt.show()
```





۱۰-خروجی نهایی:



٩- مراجع

[1] https://github.com
[2] https://stackoverflow.com/questions
[3] https://www.wikipedia.org/
[4] https://scikit-learn.github.io
[5] https://scikit-learn.org/stable/supervised_learning.html
[6] https://pandas.pydata.org/
[7] https://builtin.com/data-science/random-forest-python
[8] https://www.geeksforgeeks.org/random-forest-regression-in-python/
[9] https://medium.com/@brandon93.w/regression-model-evaluation-metrics-r-squared-adjusted-r-squared-mse-rmse-and-mae-24dcc0e4cbd3
[10] https://stats.stackexchange.com/questions/618544/how-to-choose-between-r2-and-mse-scores