

Università degli Studi di Modena e Reggio Emilia
Dipartimento di Scienze e Metodi dell'Ingegneria

CORSO DI LAUREA MAGISTRALE
in
INGEGNERIA GESTIONALE

-
DATA SCIENCE & MANAGEMENT

**MACHINE LEARNING
BENCHMARKS**

-
LINKEDIN PROFILE

Matteo Musella

A.A. 2022/2023

INDICE:

1. INTRODUZIONE

2. NOTA METODOLOGICA

2.1 FONTI DEI DATI

2.2 INDICATORI DI VALUTAZIONE

3. ANALISI DEI DATI

3.1 INTERPRETAZIONE

3.2 DATA CLEANING

3.3 DATA EXPLORATION

 3.3.1 Rimozione degli Outliners

 3.3.2 Valutazione della ridondanza dei dati e delle correlazioni

4. CONFRONTO TRA ALGORITMI DI MACHINE LEARNING

4.1 IMPOSTAZIONI PRELIMINARI PER IL CONFRONTO DEI REGRESSORI

 4.1.1 Features utilizzate

 4.1.2 K Cross Validation

4.2 LINEAR REGRESSION

4.3 LOGISTIC REGRESSION

4.4 KNEIGHBORS REGRESSOR

4.5 DECISION TREE

4.6 RANDOM FOREST

4.7 PERFORMANCE FINALE DI REGRESSIONE

1. INTRODUZIONE

Nel seguente elaborato si vogliono descrivere i procedimenti e gli step impiegati nello svolgimento di un'analisi esplorativa dei dati inerenti ad un dataset.

La base di dati è costituita da 15.000 profili LinkedIn, questa include la storia lavorativa, le generalità ma anche l'analisi della foto profilo; questi valori saranno poi utilizzati per lo studio e la comparazione di algoritmi di regressione di machine learning.

L'obiettivo della ricerca è quello di capire quale sia il gruppo di features più importante da avere sulla piattaforma LinkedIn, per cercare di ottenere un posizione lavorativa migliore, sempre se questa scelta possa derivare solo dalla sterile analisi delle caratteristiche riportate nel database.

Come criteri di confronto, si è deciso di implementare i seguenti algoritmi di regressione di machine learning: la Linear Regression, la Logistic Regression, il KNeighbors Regressor, il Decision Tree e il Random Forest.

2. NOTA METODOLOGICA

In questo paragrafo verranno descritti tutti gli indicatori utilizzati per svolgere l'analisi e il confronto tra gli algoritmi, di conseguenza, anche i dati presenti all'interno della documentazione.

2.1 FONTI DEI DATI

Nell'elaborato è stato utilizzato un dataset fornito dalla piattaforma Kaggle, consociata di Google, la quale mette a disposizione dataset a scopo didattico ed in particolare il dataset analizzato è disponibile al seguente link:

<https://www.kaggle.com/datasets/killbot/linkedin>

Per capire completamente a cosa si riferiscono le features che compongono il DataFrame, è stato descritto il compito e la funzione del social da cui sono stati estrapolati i dati.

LinkedIn è un servizio web o social network, che viene impiegato principalmente nello sviluppo di contatti professionali, tramite la pubblicazione e diffusione del curriculum vitae e nella condivisione di contenuti specifici relativi al mercato del lavoro.

La piattaforma viene utilizzata principalmente per il networking professionale e lo sviluppo della carriera e consente alle persone in cerca di lavoro di pubblicare i propri curriculum vitae e ai datori di lavoro di pubblicare offerte di impiego.

A partire dal 2015 la maggior parte delle entrate dell'azienda provengono dalla vendita dell'accesso alle informazioni dei suoi utenti a recruiter ed ad aziende.

Per comprendere la portata di connessioni fornita dalla piattaforma, basti pensare che dal marzo 2023, LinkedIn ha più di 900 milioni di membri registrati provenienti da oltre 200 paesi e territori.

Il dataset è composto da 49 features differenti riportanti caratteristiche relative al profilo, ai ruoli lavorativi ricoperti, al genere, all'etnicità ma anche a connotati e peculiarità relative all'immagine profilo scelta dall'utente iscritto alla piattaforma.

2.2 INDICATORI DI VALUTAZIONE

Nell'analisi del DataFrame sono stati elaborati algoritmi di regressione, di conseguenza vengono utilizzate tre differenti metriche per misurare la performance degli algoritmi, quali: mean absolute error, mean squared error e mean absolute deviation.

La metrica più comune tra le funzioni di costo della regressione è il **Mean Squared Error**, ed è definito come:

$$\text{mean_squared_error : } MSE = \frac{\sum_{i=1}^n (x_i - y_i)^2}{n}$$

Un modello di regressione tenta di adattare i dati tracciando una linea che riduce al minimo la distanza dai punti dati reali e dal punto sulla stessa linea. Più i valori sono vicini alla linea, migliore è il comportamento del modello per quel particolare punto; pertanto, più basso è l'MSE, meglio è.

Il **Mean Absolute Error** è simile a MSE in quanto restituisce i valori assoluti dei residui senza l'elevazione al quadrato. Non considera la direzione dell'errore, il che significa che non sapremo se gli errori negativi o positivi pesano di più sulla media complessiva.

$$\text{mean_absolute_error : } MAE = \frac{\sum_{i=1}^n |x_i - y_i|}{n}$$

Dalle nozioni di statistica si conosce che la mediana non è influenzata da valori anomali come lo è la media. Il **Median Absolute Error** indica la distanza assoluta delle previsioni dalla retta di regressione mediana.

$$\text{median_absolute_error : } MdAE = median(\{|x_i - y_i|\}_{i=1}^n)$$

Se sono presenti valori anomali e non trattati conviene utilizzare MdAE invece di MSE o MAE, poiché fornisce una migliore rappresentazione dell'errore per i punti che non sono così distanti dalla retta di regressione.

Nello studio l'intero gruppo di features è stato suddiviso in 4 gruppi di attributi in modo tale da capire quale di questi è quello correlato maggiormente al numero di followers che gli utenti riuscivano ad ottenere sui propri profili LinkedIn.

I gruppi utilizzati sono stati:

- *Employment_Data* = features inerenti ai dati lavorativi;
- *User_Data* = attributi riguardanti la sfera personale ed etnica dell'utente;
- *Picture_Data* = features dell'espressione e delle caratteristiche del volto nell'immagine del profilo utilizzata;
- *General_Data* = sono stati riunite tutte le features per provare ad ottenere una panoramica generale di tutto il DataFrame;

3. ANALISI DEI DATI

Lo studio dei profili di utenti alla piattaforma LinkedIn è stato svolto su più livelli di profondità dell'analisi ed con differenti applicazioni.

3.1 INTERPRETAZIONE

In questo paragrafo si vuole approfondire il significato degli attributi, classificando le informazioni e provando ad esplicitare la variabile target dell'analisi.

Per una comprensione maggiore, vengono riportate le features che sono state utilizzate all'interno dell'elaborato:

- **Profile data:**

c_id, age , n_followers

Le colonne appartenenti a questa categoria descrivono il numero d'iscrizione seriale dell'utente, la sua età e il numero di follower, ovvero di profili connessi all'interno della rete del profilo preso in considerazione.

- **Job data:**

avg_time_in_previous_position, avg_current_position_length, m_urn, m_urn_id, avg_previous_position_length, no_of_promotions, no_of_previous_positions_current_position_length

Le informazioni relative a questo gruppo riguardano le posizioni lavorative attuali o precedenti che ricopre o ha ricoperto l'utente, come il tempo medio di impiego o il numero di promozioni guadagnate nel precedente impiego.

- **Personal Details analysis:**

ethnicity , gender, nationality, african, celtic_english, east_asian, european, greek, hispanic, jewish, muslim, nordic, south_asian

In queste colonne, in particolare, sono riportate informazioni inerenti al genere, alla nazionalità o il grado di ‘etnicità’ corrispondente all’user.

- **Profile picture analysis:**

beauty, beauty_female, beauty_male, blur, emo_anger, emo_disgust, emo_fear, emo_happiness, emo_neutral, emo_sadness, emo_surprise, glass, head_pitch, head_roll, head_yaw, mouth_close, mouth_mask, mouth_open, mouth_other, skin_acne, skin_dark_circle, skin_health, skin_stain, smile, face_quality

In questa categoria di features vengono riportate le caratteristiche proprie che si possono assimilare dall’immagine del profilo che un utente decide caricare sulla piattaforma. Tra questi attributi sono presenti: il grado di bellezza, i gradi di emozioni, alcune caratteristiche della cute facciale e anche delle caratteristiche del sorriso e della bocca.

3.2 DATA CLEANING

Per ottenere un’analisi più accurata e omogenea, in questa fase è stato eseguito un processo di raffinamento delle features.

Per prima cosa la features **c_id** è stata convertita da *string* a *int64* mediante la seguente procedura:

```
for i, el in enumerate(LinkedIn_profile['c_id']):
    try:
        LinkedIn_profile['c_id'][i] = int(el)
    except:
        LinkedIn_profile['c_id'][i] = i+1
LinkedIn_profile['c_id'] = LinkedIn_profile['c_id'].astype('int64')
```

Figura 1 - Procedura di conversione da *string* a *int64* per la feature **c_id**

Successivamente si è deciso di convertire la feature **gender** da *string* a valore *boolean*, assegnando rispettivamente 1 e 0 a ‘Male’ e ‘Female’, utilizzando il metodo **.replace()**.

```
df_LinkedIn['gender'] = df_LinkedIn['gender'].replace('Male', 1)
df_LinkedIn['gender'] = df_LinkedIn['gender'].replace('Female', 0)
```

Figura 2 - Procedura di conversione da *string* a *boolean* per la feature **gender**

Dopo aver fixato le features, si rimuovono gli attributi non numerici o di cui non erano disponibili le informazioni. A questo punto il dataset risultante viene denominato ‘**df_LinkedIn_clean**’, il quale è formato dalle seguenti features ed è mostrato nella figura successiva.

```

RangeIndex: 62706 entries, 0 to 62705
Data columns (total 45 columns):
 #   Column           Non-Null Count Dtype  
 --- 
 0   c_id             62706 non-null  int64   
 1   avg_time_in_previous_position 62706 non-null  float64  
 2   avg_current_position_length   62706 non-null  float64  
 3   avg_previous_position_length 62706 non-null  float64  
 4   m_urn_id          62706 non-null  int64   
 5   no_of_promotions    62706 non-null  int64   
 6   no_of_previous_positions 62706 non-null  int64  
 7   current_position_length   62706 non-null  int64  
 8   age               62706 non-null  int64   
 9   beauty            62706 non-null  float64  
 10  beauty_female     62706 non-null  float64  

 36  european         62706 non-null  float64  
 37  greek            62706 non-null  float64  
 38  hispanic          62706 non-null  float64  
 39  jewish            62706 non-null  float64  
 40  muslim            62706 non-null  float64  
 41  nordic            62706 non-null  float64  
 42  south_asian       62706 non-null  float64  
 43  n_followers       62706 non-null  int64  
 44  face_quality      62706 non-null  float64  
dtypes: float64(37), int64(8)

```

Figura 3 - `.info()` di `df_LinkedIn_clean`

Il dataset risultante denominato '`df_LinkedIn_clean`' è composto da 62.706 righe rappresentanti profili LinkedIn ed è caratterizzato come di seguito:

Features → 'c_id', 'avg_time_in_previous_position', 'avg_current_position_length', 'avg_previous_position_length', 'm_urn_id', 'no_of_promotions', 'no_of_previous_positions', 'current_position_length', 'age', 'beauty', 'beauty_female', 'beauty_male', 'blur', 'emo_anger', 'emp_disgust', 'emo_fear', 'emo_happiness', 'emo_neutral', 'emo_sadness', 'emo_surprise', 'gender', 'head_pitch', 'head_roll', 'head_yaw', 'mouth_close', 'mouth_mask', 'mouth_open', 'mouth_other', 'skin_acne', 'skin_dark_circle', 'skin_health', 'skin_stain', 'smile', 'nationality', 'african', 'celtic_english', 'east_asian', 'european', 'greek', 'hispanic', 'jewish', 'muslim', 'nordic', 'south_asian', 'face_quality'

Target → 'n_followers'

La colonna target è '`n_followers`', ovvero il numero di seguaci o utenti connessi con la rete dell'account poiché si ipotizza che un profilo che abbia un alto rating nelle features analizzate, possa raggiungere un alto numero di user della piattaforma. Un'altra operazione che si svolge sul DataFrame, è quella di controllare la presenza di valori nulli all'interno dei valori che formano il set di dati.

Fortunatamente dopo aver utilizzato con il metodo `.isnull().sum()`, si è potuto constatare che all'interno del DataFrame vi sono tutti i valori, di conseguenza non si necessita dell'operazione della rimozione e successiva modifica dei valori nulli.

```

LinkedIn_profile_clean.isnull().sum()

c_id                      0
avg_time_in_previous_position 0
avg_current_position_length 0
avg_previous_position_length 0
m_urn_id                   0
no_of_promotions            0
no_of_previous_positions    0
current_position_length     0
age                         0
beauty                       0

celtic_english              0
east_asian                   0
european                     0
greek                        0
hispanic                     0
jewish                       0
muslim                       0
nordic                       0
south_asian                  0
n_followers                 0
face_quality                 0
dtype: int64

```

Figura 4 - Output del metodo `.isnull().sum()` per la rimozione dei valori nulli

3.3 DATA EXPLORATION

Il passo successivo è stato quello di esplorare i dati per comprendere meglio il contenuto del dataset.

3.3.1 Rimozione degli Outliers

Nella figure successiva sono presenti i boxplots delle distribuzioni di ogni feature numerica tramite i quali è possibile visualizzare gli intervalli dove si concentrano i casi e allo stesso tempo individuare gli eventuali elementi che si allontanano maggiormente dalla distribuzione: gli **outliers**.

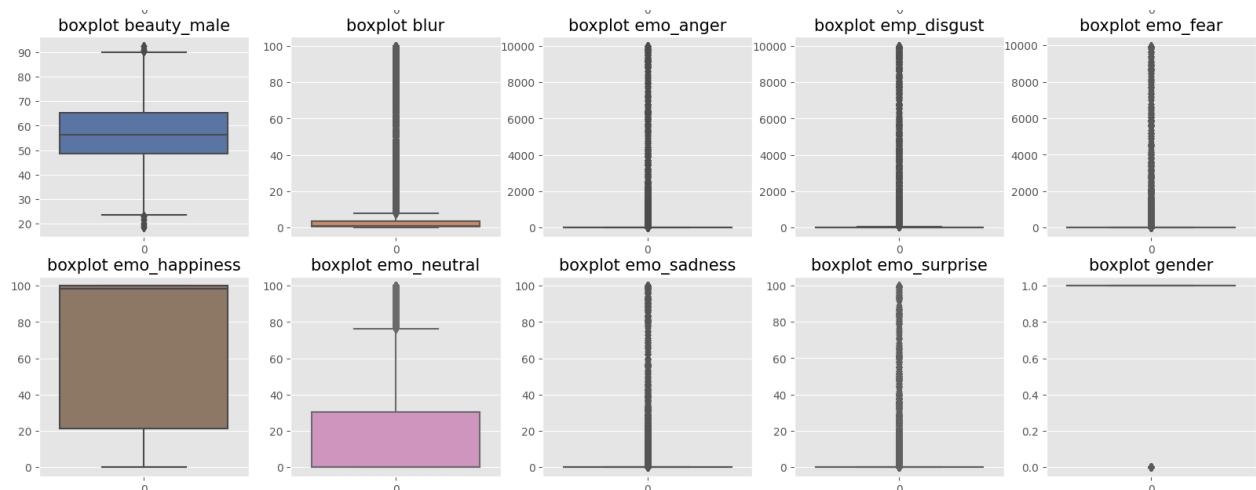


Figura 5 - Boxplot per la rappresentazione degli outliers

Dopo aver analizzato i grafi, non si trovano valori ***outliers*** all'interno delle rappresentazioni delle features, di conseguenza si procede con i prossimi step.

3.3.2 Valutazione della ridondanza dei dati e delle correlazioni

Conclusa questa fase, l'analisi si focalizza sul cercare di trovare delle correlazioni tra **features** e **target**. Come primo step si inizia con la correlazione tra l'attributo target (***n_followers***) e alcune features, rappresentandola tramite grafici '*heatmap*'.

Nella figura 6 è esposta la *heatmap* delle correlazioni tra feature e variabile target. Si nota che le variabili inerenti alle espressioni che gli utenti hanno nella propria immagine profilo influenzano relativamente poco il numero di followers, mentre le features che riguardano le posizioni lavorative attuali e precedenti riescono a influenzare maggiormente la rete di connessioni che gli utenti implementano sul proprio profilo.

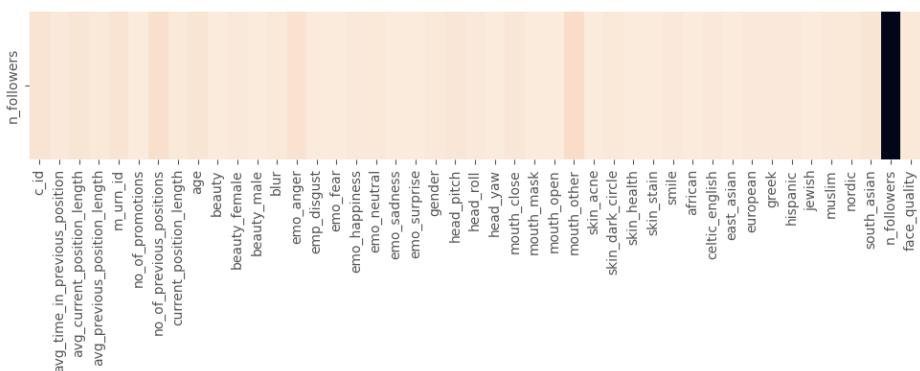


Figura 6 - Correlazione tra target e altre features

Successivamente abbiamo calcolato la matrice di correlazione tra tutte le features e le abbiamo mostrate nella heatmap della figura 7.

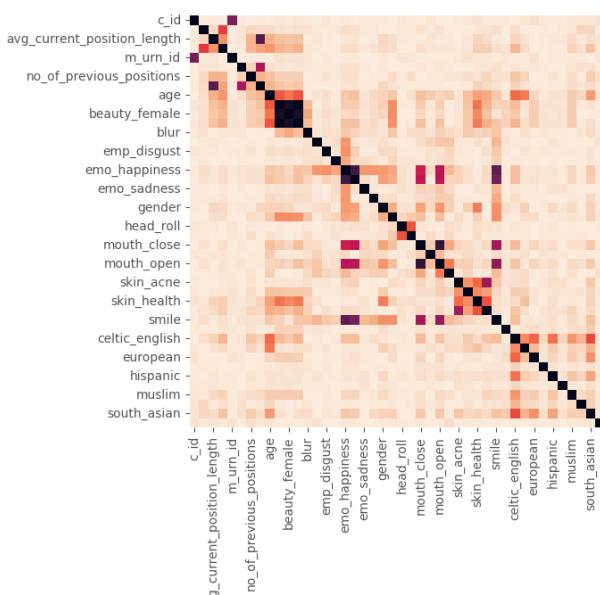


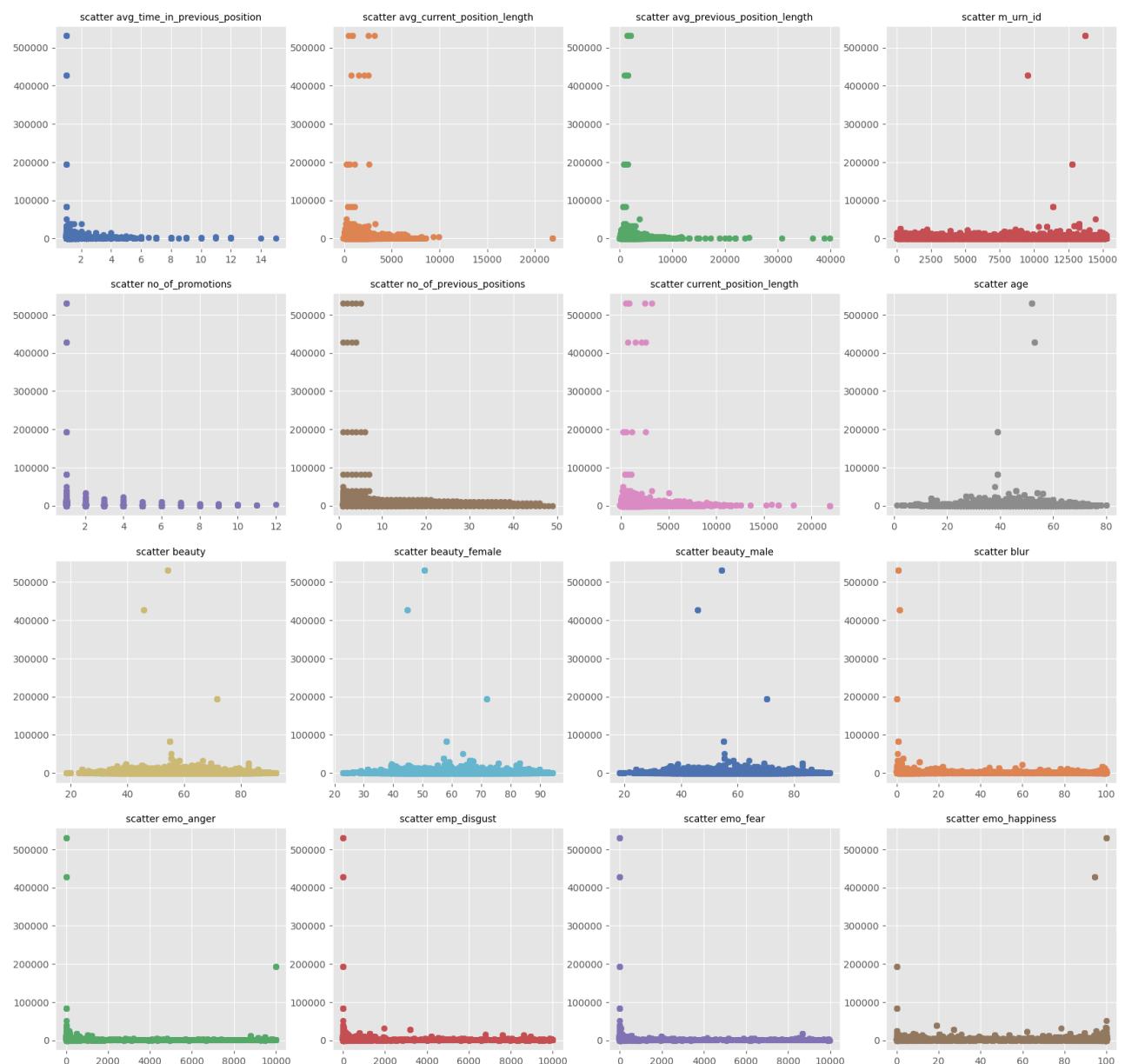
Figura 7 - Correlazione tra features

Dalla valutazione si evidenzia che gli attributi maggiormente correlati tra loro sono:

- ‘c_id’ → ‘m_urn_id’
- ‘smile’ → ‘emo_happiness’
- ‘smile’ → ‘emo_sadness’
- ‘age’ → ‘no_of_previous_positions’

Le variabili elencate sono accoppiate classificandole secondo il valore cromatico dell’*heatmap*. Successivamente le seconde features presenti in queste coppie sono state eliminate nelle prossime classificazioni per evitare ridondanze nelle informazioni e di conseguenza possibili bias.

Per rappresentare la distribuzione delle varie features si è deciso di rappresentare gli attributi correlati con la variabile target mediante scatterplot, come mostrato nella figura 8.



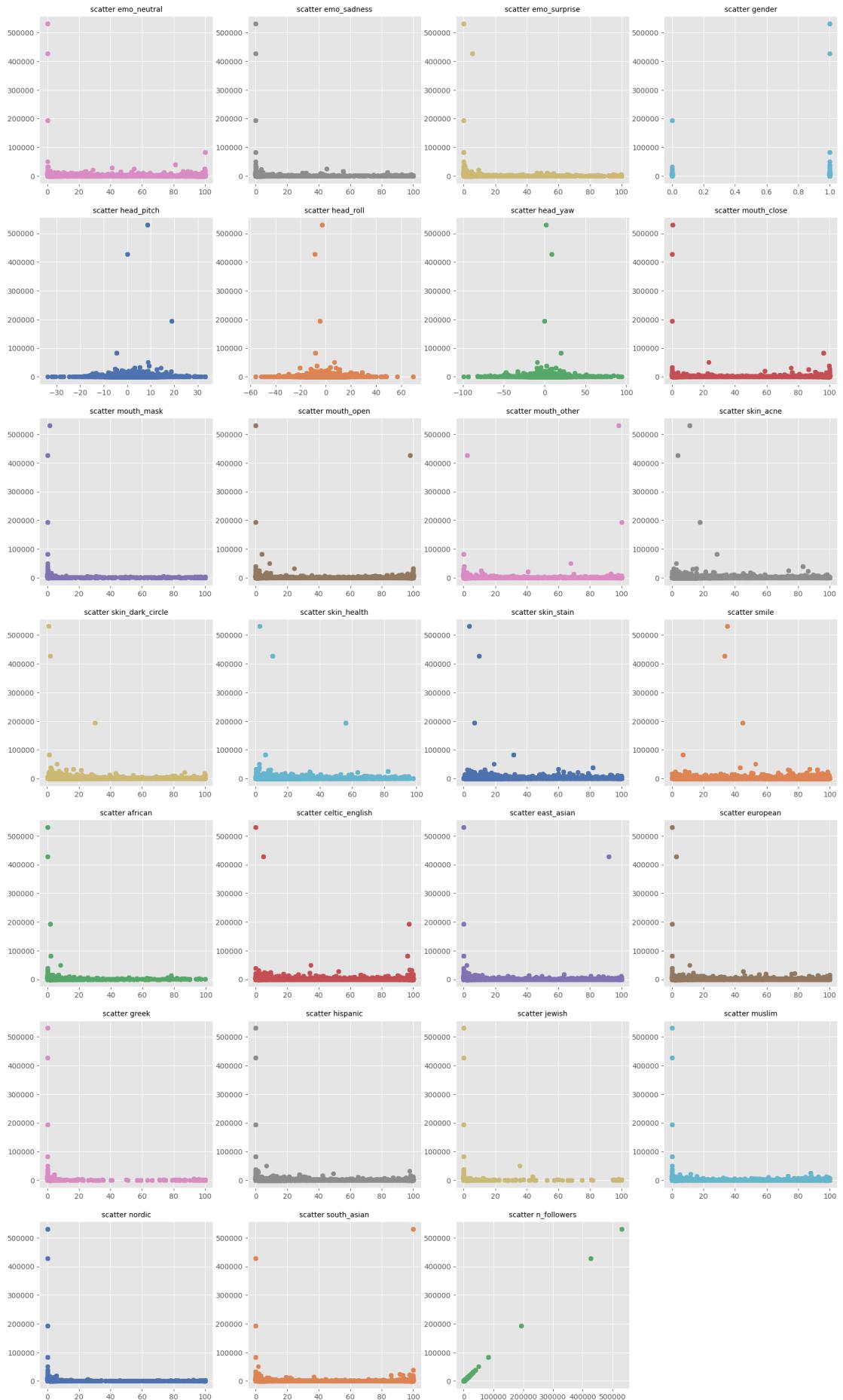


Figura 8 - Scatterplot tra features e variabile target

4. CONFRONTO TRA ALGORITMI DI MACHINE LEARNING

Nella sezione seguente verranno studiati i risultati ottenuti da cinque algoritmi di regressione (Linear Regression, Logistic Regression, KNeighbors Regressor, Decision Tree, Random Forest) applicati sui dati precedentemente descritti e analizzati, giustificando la scelta finale per la classificazione dei dati di test.

4.1 IMPOSTAZIONI PRELIMINARI PER IL CONFRONTO DEI REGRESSORI

Per ottimizzare i regressori si è deciso di normalizzare i dati tramite la funzione **min-max scaler**, facendo ciò non si è penalizzato nessun modello dalla forma dei dati.

4.1.1 Features utilizzate

All'interno dell'analisi è stato utilizzato un DataFrame inherente ad un data set riguardante 62706 profili LinkedIn, includendo sia la storia lavorativa, i nomi ma anche l'analisi della foto profilo. Rispetto al DataFrame utilizzato nelle sezioni precedenti, sono state eliminate quattro features per non ricadere in ridondanza nei risultati, come è stato detto in precedenza, di conseguenza è stato inizializzato un nuovo DataFrame chiamata '**df_LinkedIn_profile**'.

Features → 'c_id', 'avg_time_in_previous_position', 'avg_current_position_length',
'avg_previous_position_length', 'no_of_promotions',
'no_of_previous_positions', 'current_position_length', 'age', 'beauty',
'beauty_female', 'beauty_male', 'blur', 'emo_anger', 'emp_disgust',
'emo_fear', 'emo_neutral', 'emo_surprise', 'gender', 'head_pitch',
'head_roll', 'head_yaw', 'mouth_close', 'mouth_mask', 'mouth_open',
'mouth_other', 'skin_acne', 'skin_dark_circle', 'skin_health',
'skin_stain', 'smile', 'nationality', 'african', 'celtic_english', 'east_asian',
'european', 'greek', 'hispanic', 'jewish', 'muslim', 'nordic',
'south_asian', 'face_quality'

Target → 'n_followers'

4.1.2 K Cross Validation

In questa sezione viene riportato l'insieme dei passaggi che si devono svolgere per la valutazione delle prestazioni dei classificatori utilizzati.

Il primo passaggio consta nell'inizializzare un ciclo *for* che viene ripetuto i=10 volte, ad ogni iterazione viene estratto un insieme di valori uguali al 10% del set di dati che verrà denominato **test set**.

L'insieme di valori rimanente, ovvero il 90%, viene suddiviso nuovamente in ***training set*** (70%) e ***validation set*** (30%). Alla fine del ciclo di iterazioni, si otterranno 10 training set (***X_train***, ***y_train***), 10 validation set (***X_val***, ***y_val***) e 10 test set (***X_test***, ***y_test***).

Ogni regressore, come il Random Forest o la Logistic Regression, viene allenato sull'i-esimo training set e viene validato sull'i-esimo validation set, estrapolando le 3 metriche ad ogni iterazione.

4.2 LINEAR REGRESSOR

La ***Linear Regression*** è un algoritmo di apprendimento automatico basato sull'apprendimento supervisionato e questo viene utilizzato in molti campi diversi, tra cui finanza, economia ma anche psicologia, per comprendere o prevedere il comportamento di una particolare variabile e per scoprire la relazione tra variabili e previsioni.

Il compito della Linear Regression è quello di prevedere un valore di una variabile dipendente (y) basato su una data variabile indipendente (x) e di conseguenza prende il nome di Linear Regression.

L'algoritmo assegna delle etichette alle istanze utilizzando una funzione continua.

Ogni riga del dataset è un esempio composto da:

- *attributi* (X) = sono le variabili predittive che descrivono una categoria;
- *target* (Y) = indica alla macchina il risultato corretto se l'istanza appartiene all'etichetta Z. E' il dato che istruisce la macchina a decidere in modo giusto;

Il risultato finale è una linea retta che minimizza la distanza tra gli N esempi dell'insieme di formazione che appartengono alla stessa categoria.

L'algoritmo per implementare la Linear Regressor è il seguente:

```
for i in range (0,10):
    lir = LinearRegression()
    lir.fit(X_train_1, y_train_1)
    y_pred_1 = lir.predict(X_test_1)
```

Figura 9 - Procedura per la *Linear Regression*

Per provare a comprendere al meglio gli effetti che questo algoritmo ha generato, sono stati rappresentati i seguenti grafici scatterplot, con anche l'aggiunta della retta di regressione.

Come si può notare la distribuzione dei punti di generati è differente a seconda della tipologia di features analizzate, e di conseguenza anche la retta di regressione avrà un angolo diverso.

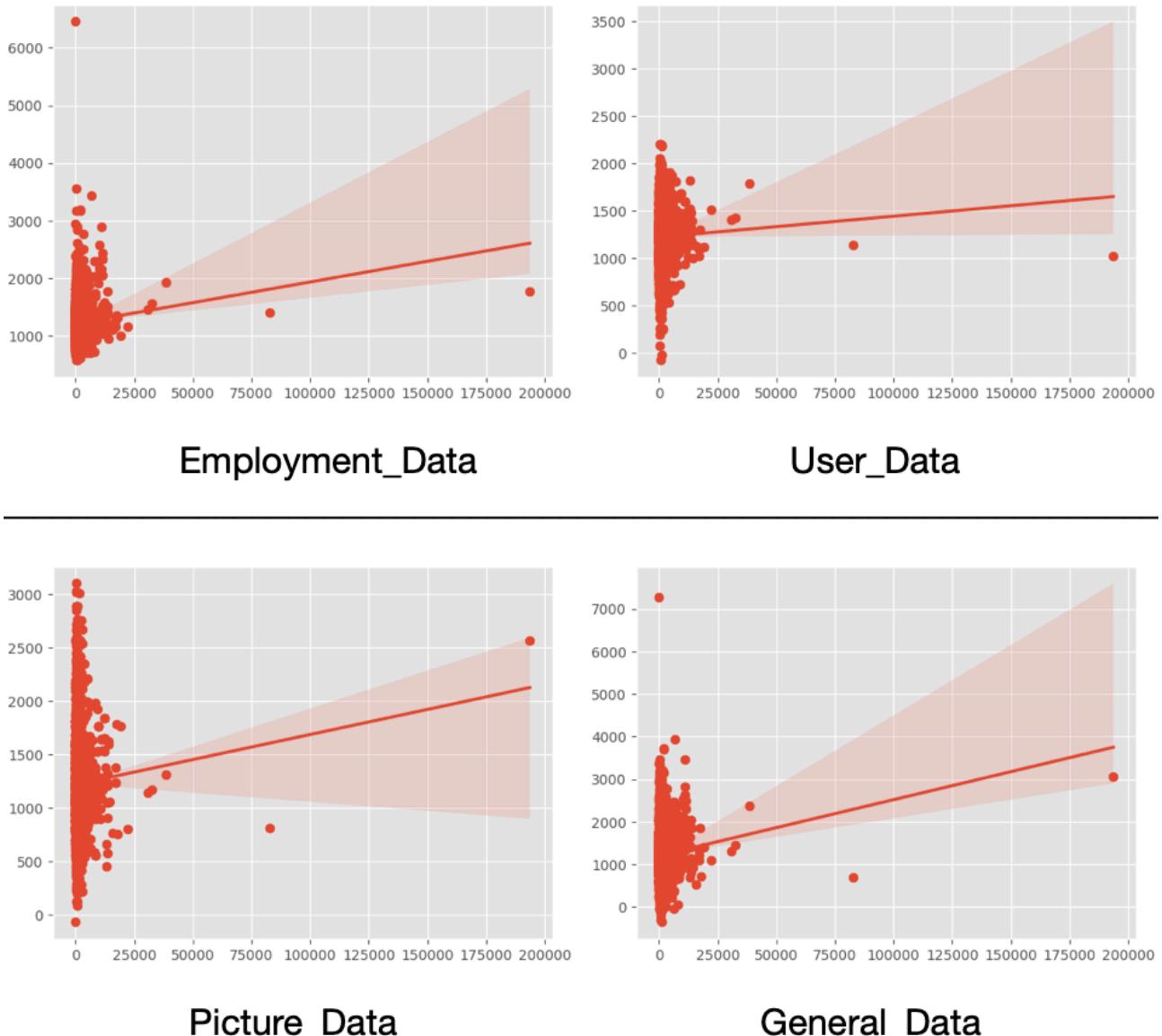


Figura 10 - Scatterplot + regression line della *Linear Regression*

4.3 LOGISTIC REGRESSOR

La **Logistic Regression** è un algoritmo di regressione di machine learning utilizzato per prevedere la probabilità di determinate classi in base ad alcune variabili dipendenti, ovvero il modello di regressione logistica calcola una somma delle caratteristiche di input e calcola la logistica del risultato.

In pratica, l'algoritmo di Logistic Regression analizza le relazioni tra features e variabile target.

Il modello di regressione logistica è una tecnica di analisi di supervised learning che aiuta a prevedere la probabilità che un evento si verifichi in futuro.

Per fare previsioni in questo scenario, sono necessari i dati dei risultati dei test precedenti.

L'algoritmo per implementare la Logistic Regressor è il seguente:

```
lor = LogisticRegression()  
lor.fit(X_train_1, y_train_1)  
y_pred_1 = lor.predict(X_test_1)
```

Figura 11 - Procedura per la *Logistic Regression*

Come si può notare dai grafici, nei primi due metodi si ottengono una tipologia di distribuzione differente dalle seconde due, le quali possono essere confrontabili.



Figura 12 - Scatterplot + regression line della *Logistic Regression*

4.4 K-NEIGHBORS REGRESSOR

Il **K-Neighbors Regressor** è un algoritmo di supervised learning automatico che può essere utilizzato per attività di classificazione o regressione. Si calcola la distanza tra le caratteristiche dei punti del test set rispetto a quelle dei punti del train set, ovvero si prende la moda o la media per calcolare i valori di previsione. L'algoritmo KNN viene utilizzato come regressore prendendo i k valori più vicini della variabile target e si calcola la media, questi poi agiscono come regressori della regressione lineare.

Nel corso dell'analisi sono stati implementati tre differenti versioni di KNN, modificando il numero di neighbors utilizzati, per poter verificare quale sia quella migliore.

L'algoritmo per implementare la K-Neighbors Regressor è il seguente:

```
for i in range(0,10):
    knn = KNeighborsRegressor(n_neighbors = 10)
    knn.fit(X_trainval_1, y_trainval_1)
    y_pred_1 = knn.predict(X_test_1)
```

Figura 13 - Procedura per il *K-Neighbors Regressor*

A seconda del numero di neighbors utilizzati si possono notare come i grafici rappresentanti le distribuzioni dei valori sono differenti, ma nonostante ciò le rette di regressione possono essere simili e confrontabili.



Figura 14 - Scatterplot + regression line del K-Neighbors Regressor ($n=1$)

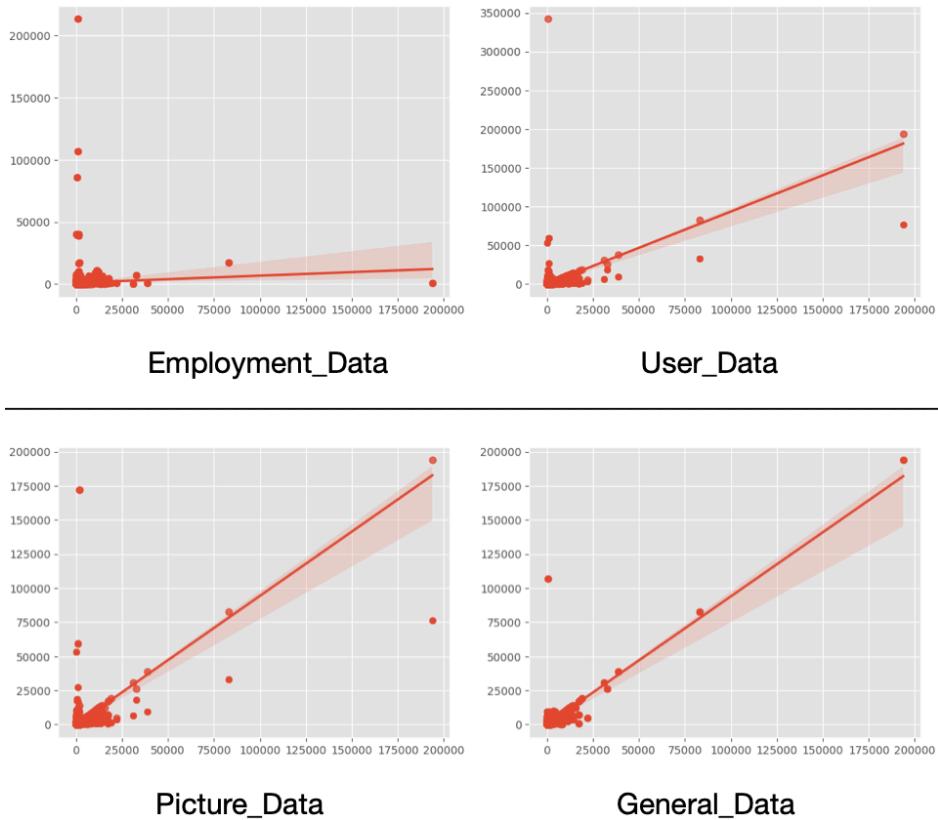


Figura 15 - Scatterplot + regression line del K-Neighbors Regressor (n=5)

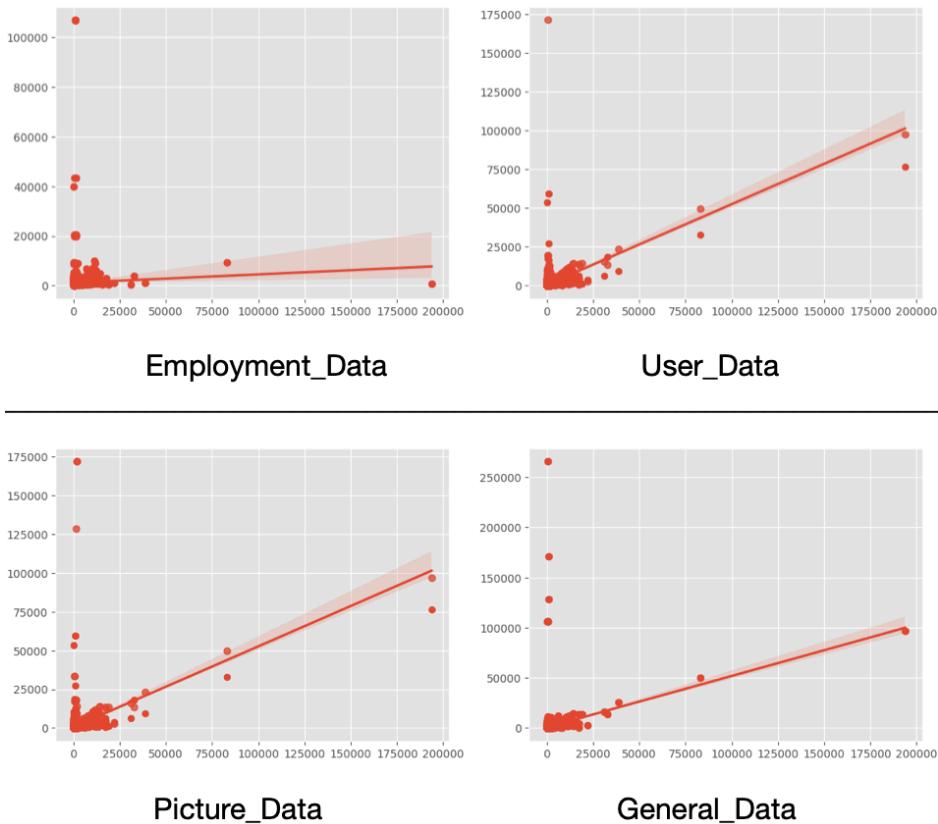


Figura 16 - Scatterplot + regression line del K-Neighbors Regressor (n=10)

4.5 DECISION TREE

Il Decision Tree è un tipo di supervised learning automatico utilizzato per classificare o fare previsioni in base a come è stata data risposta a una precedente serie di domande. Essendo il modello una forma di apprendimento supervisionato, questo significa che il modello viene addestrato e testato su un set di dati che contiene la categorizzazione desiderata.

L'albero decisionale potrebbe non fornire sempre una risposta o una decisione chiara. Invece, può presentare opzioni in modo che il data scientist possa prendere una decisione informata da solo. Gli alberi decisionali imitano il pensiero umano, quindi è generalmente facile per i data scientist comprendere e interpretare i risultati.

L'algoritmo per implementare il Decision Tree è il seguente:

```
for i in range(0,10):
    dt = DecisionTreeRegressor()
    dt.fit(X_trainval_1, y_trainval_1)
    y_pred_1 = dt.predict(X_test_1)
```

Figura 17 - Procedura per il *Decision Tree*

I grafici raffiguranti la distribuzione e la retta di regressione sono più o meno simili, eccetto lo scatterplot inerente alle features relative all'*employment_data*.



Figura 18 - Scatterplot + regression line della *Decision Tree*

4.6 RANDOM FOREST

Le Random Forest sono un metodo di apprendimento d'insieme per la classificazione, la regressione e altre attività che opera costruendo una moltitudine di alberi decisionali durante il training. Per le attività di classificazione, l'output della foresta casuale è la classe selezionata dalla maggior parte degli alberi, al contrario, per le attività di regressione viene restituita la previsione media o media dei singoli alberi.

Le Random Forest sono spesso utilizzate come modelli di black-box nelle aziende, dato che si generano previsioni ragionevoli su un'ampia gamma di dati pur richiedendo poca configurazione.

L'algoritmo per implementare le Random Forest è il seguente:

```
for i in range(0,10):
    rf = RandomForestRegressor()
    rf.fit(X_trainval_1, y_trainval_1)
    y_pred_1 = rf.predict(X_test_1)
```

Figura 19 - Procedura per le *Random Forest*

Gli scatterplot delle Random Forest sono confrontabili tra loro e denotano un agglomerato di valori con qualche punto distante.

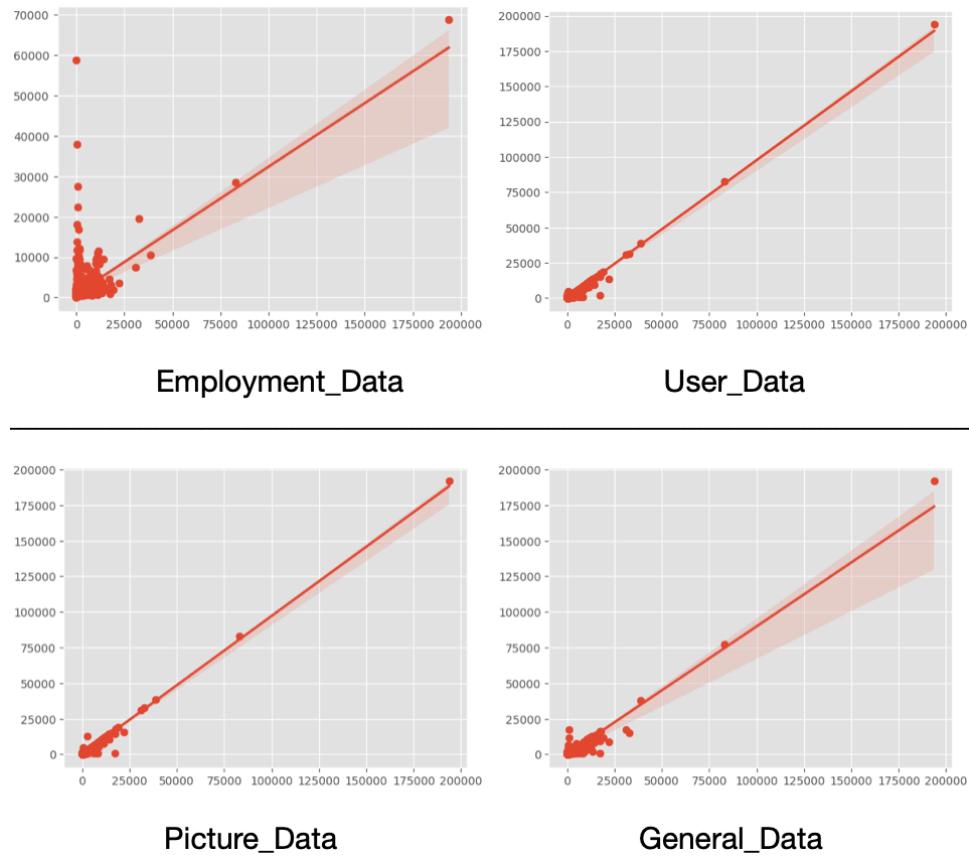


Figura 20 - Scatterplot + regression line della *Decision Tree*

4.7 PERFORMANCE FINALE DI REGRESSIONE

Per valutare le performance finali dell'analisi del DataFrame inerenti agli utenti di LinkedIn, si è deciso di riunire le metriche in forma tabulare.

	EMPLOYMENT_DATA [1st CASE]			USER_DATA [2nd CASE]		
	Mean Absolute Error	Mean Squared Error	Median Absolute Error	Mean Absolute Error	Mean Squared Error	Median Absolute Error
LINEAR REGRESSION	906,66988	9735401,10668	634,95394	919,80083	9827290,15528	654,37921
LOGISTIC REGRESSION	1166,48318	11158771,68489	718,00000	1169,68729	11202810,05278	715,00000
KNEIGHBORS REGRESSION (n = 10)	952,76715	14421327,03641	489,90000	599,21807	7763964,84324	266,79999
KNEIGHBORS REGRESSION (n = 5)	982,31016	21072565,31881	471,20000	356,74029	19304079,37341	72,00000
KNEIGHBORS REGRESSION (n = 1)	1134,50821	68407219,51028	456,00000	42,38032	153828,64120	0,00000
DECISION TREE	846,63546	48114779,16393	318,00000	38,30489	116673,32483	0,00000
RANDOM FOREST	830,66051	6676461,39137	504,81999	95,21830	138940,39884	10,81999
BEST	DT	RF	DT	DT	DT	KKN (n=1) / DT
	PICTURE_DATA [3TH CASE]			GENERAL_DATA [4th CASE]		
	Mean Absolute Error	Mean Squared Error	Median Absolute Error	Mean Absolute Error	Mean Squared Error	Median Absolute Error
LINEAR REGRESSION	937,02018	9832468,57978	635,86862	946,34752	9786537,19111	623,58358
LOGISTIC REGRESSION	1188,25099	11541657,97895	708,00000	1118,53899	11131661,09169	674,00000
KNEIGHBORS REGRESSION (n = 10)	658,81881	15256323,24205	274,79999	717,06573	28618093,22100	271,20000
KNEIGHBORS REGRESSION (n = 5)	361,21285	9933962,30614	69,79999	321,30748	2451832,02200	77,60000
KNEIGHBORS REGRESSION (n = 1)	38,80007	114640,82171	0,00000	46,40041	131633,28735	0,00000
DECISION TREE	43,26694	140975,86748	0,00000	280,13251	1361570,14559	0,00000
RANDOM FOREST	100,04158	151001,01990	11,22000	317,68747	576096,49917	163,50000
BEST	KKN (n=1)	KKN (n=1)	KKN (n=1) / DT	KKN (n=1)	KKN (n=1)	KKN (n=1) / DT

Figura 21 - Risultati finali delle performance

Dalla tabella si può notare che per le features inerenti al proprio impiego, ovvero all'*Employment_Data*, l'algoritmo che riesce a performare meglio è il **Decision Tree**, in quanto sia nel mean squared error che nel median absolute error è il migliore dei metodi. Anche per le features dell'*User_Data*, l'algoritmo migliore è quello del **Decision Tree** sulle prime due metriche utilizzate, mentre nella terza e a pari con il **K-Neghbors Regression** con un solo neighbors.

Il metodo **K-Neghbors Regression** con il numero di neighbors pari a 1 è quello che performa meglio sia per il gruppo delle features relative alla Picture_Data, ma anche a tutte le features unite, ovvero le General_Data.

Oltre a confrontare gli algoritmi rispetto alla categoria di features utilizzata, si possono anche confrontare i gruppi di attributi tra loro. Dalle metriche si può

constatare che l'insieme degli attributi ad essere correlato maggiormente alla variabile target '*n_followers*' è sicuramente quello dell'***User_Data***.

A constatare questa tendenza è il confronto di tutti i risultati delle metriche tra gli algoritmi, ma nonostante ciò, i valori ottenuti non sono troppo soddisfacenti.

I risultati delle performance esprimono il fatto che non vi è una forte correlazione tra una determinata classe di attributi e il '*n_followers*', questo può derivare dalla tipologia di social network da cui sono stati estrapolati i dati. LinkedIn, al contrario di Instagram o Facebook, è un social dove il numero di followers non è relativo alla popolarità di un utente, infatti viene utilizzato per la ricerca di un lavoro o per mostrare la propria carriera lavorativa.

Concludendo questa analisi si mostra che in LinkedIn, il gruppo di features da implementare per poter aumentare il numero di seguaci è quella degli ***User_data***.

▼ Project Code - Profili LinkedIn | [Matteo Musella]

▼ 03 - ANALISI DEI DATI

▼ 3.1 - General

As always, you start by importing all the libraries your code needs.

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
import sklearn
import warnings

from scipy import stats

warnings.filterwarnings("ignore")
plt.style.use('ggplot')
```

The dataset is loaded, which contains information relating to LinkedIn profiles.

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

Drive already mounted at /content/drive; to attempt to forcibly remount, ca

```
df_LinkedIn = "/content/drive/MyDrive/LinkedIn_profile.csv"
df_LinkedIn = pd.read_csv(df_LinkedIn)
```

Salvataggio automatico non riuscito Questo file è stato aggiornato da remoto o in un'altra scheda.

[Mostra differenze](#)

- the number of lines
- the number of columns
- the data type for each column
- the number of non-zero rows for each column

df_LinkedIn.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 62706 entries, 0 to 62705
Data columns (total 49 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   c_id             62706 non-null   object 
 1   avg_time_in_previous_position 62706 non-null   float64
 2   avg_current_position_length 62706 non-null   float64
 3   avg_previous_position_length 62706 non-null   float64
 4   m_urn             62706 non-null   object 
 5   m_urn_id          62706 non-null   int64  
 6   no_of_promotions 62706 non-null   int64  
 7   no_of_previous_positions 62706 non-null   int64  
 8   current_position_length 62706 non-null   int64  
 9   age               62706 non-null   int64  
 10  beauty            62706 non-null   float64
 11  beauty_female    62706 non-null   float64
 12  beauty_male      62706 non-null   float64
 13  blur              62706 non-null   float64
 14  emo_anger         62706 non-null   float64
 15  emp_disgust       62706 non-null   float64
 16  emo_fear          62706 non-null   float64
 17  emo_happiness     62706 non-null   float64
 18  emo_neutral       62706 non-null   float64
 19  emo_sadness        62706 non-null   float64
 20  emo_surprise       62706 non-null   float64
 21  ethnicity          62706 non-null   object 
 22  gender             62706 non-null   object 
 23  glass              62706 non-null   object 
 24  head_pitch         62706 non-null   float64
 25  head_roll          62706 non-null   float64
 26  head_yaw           62706 non-null   float64
 27  mouth_close         62706 non-null   float64
 28  mouth_mask          62706 non-null   float64
 29  mouth_open          62706 non-null   float64
 30  mouth_other         62706 non-null   float64
 31  skin_acne           62706 non-null   float64
 32  skin_dark_circle    62706 non-null   float64
 33  skin_health          62706 non-null   float64
 34  skin_stain           62706 non-null   float64
 35  smile               62706 non-null   float64
 36  nationality          62706 non-null   object 
```

```

    nationality          object
37 african            62706 non-null float64
38 celtic_english     62706 non-null float64
39 east_asian          62706 non-null float64
40 european           62706 non-null float64
41 greek              62706 non-null float64
42 hispanic            62706 non-null float64
43 jewish              62706 non-null float64
44 muslim              62706 non-null float64
45 nordic              62706 non-null float64
46 south_asian         62706 non-null float64
47 n_followers        62706 non-null int64
48 face_quality        62706 non-null float64
dtypes: float64(37), int64(6), object(6)
memory usage: 23.4+ MB

```

Using the `.head()` method, let's look at the first few rows of the DataFrame:

```
df_LinkedIn.head(5)
```

	c_id	avg_time_in_previous_position	avg_current_position_length	avg_pre
0	1	2.000000		457.0
1	2	1.500000		212.0
2	3	1.333333		243.0
3	4	1.250000		123.0
4	5	1.200000		244.0

5 rows × 49 columns



▼ 3.2 - Data Cleaning

```

for i, el in enumerate(df_LinkedIn['c_id']):
    try:
        df_LinkedIn['c_id'][i] = int(el)
    except:
        df_LinkedIn['c_id'][i] = i+1
df_LinkedIn['c_id'] = df_LinkedIn['c_id'].astype('int64')

```

Convert the feature gender from string to binary, using the `.replace()` method:

```
df_LinkedIn['gender'] = df_LinkedIn['gender'].replace('Male', 1)
df_LinkedIn['gender'] = df_LinkedIn['gender'].replace('Female', 0)

df_LinkedIn_clean = df_LinkedIn[['c_id', 'avg_time_in_previous_position', 'avg_c
                           'current_position_length', 'age', 'beauty', 'be
                           'emo_surprise', 'gender', 'head_pitch', 'head_r
                           'skin_stain', 'smile', 'african', 'celtic_engli

df_LinkedIn_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 62706 entries, 0 to 62705
Data columns (total 45 columns):
 #   Column           Non-Null Count Dtype  
 --- 
 0   c_id             62706 non-null  int64   
 1   avg_time_in_previous_position 62706 non-null  float64  
 2   avg_current_position_length 62706 non-null  float64  
 3   avg_previous_position_length 62706 non-null  float64  
 4   m_urn_id          62706 non-null  int64   
 5   no_of_promotions 62706 non-null  int64   
 6   no_of_previous_positions 62706 non-null  int64  
 7   current_position_length 62706 non-null  int64  
 8   age              62706 non-null  int64   
 9   beauty            62706 non-null  float64  
 10  beauty_female    62706 non-null  float64  
 11  beauty_male     62706 non-null  float64  
 12  blur              62706 non-null  float64  
 13  emo_anger        62706 non-null  float64  
 14  emp_disgust      62706 non-null  float64  
 15  emo_fear         62706 non-null  float64  
 16  emo_happiness    62706 non-null  float64  
 17  emo_neutral      62706 non-null  float64  
 18  emo_sadness      62706 non-null  float64  
 19  emo_surprise     62706 non-null  float64  
 20  gender            62706 non-null  int64   
 21  head_pitch       62706 non-null  float64  
 22  head_roll         62706 non-null  float64  
 23  head_yaw          62706 non-null  float64  
 24  mouth_close       62706 non-null  float64  
 25  mouth_mask        62706 non-null  float64  
 26  mouth_open        62706 non-null  float64  
 27  mouth_other       62706 non-null  float64  
 28  skin_acne         62706 non-null  float64  
 29  skin_dark_circle 62706 non-null  float64  
 30  skin_health       62706 non-null  float64  
 31  skin_stain        62706 non-null  float64  
 32  smile             62706 non-null  float64  
 33  african           62706 non-null  float64  
 34  celtic_english    62706 non-null  float64  
 35  east_asian        62706 non-null  float64  
 36  european          62706 non-null  float64  
 37  greek             62706 non-null  float64  
 38  hispanic          62706 non-null  float64  
 39  jewish            62706 non-null  float64  
 40  muslim             62706 non-null  float64  
 41  nordic            62706 non-null  float64  
 42  south_asian       62706 non-null  float64  
 43  n_followers       62706 non-null  int64   
 44  face_quality      62706 non-null  float64  
dtypes: float64(37), int64(8)
memory usage: 21.5 MB
```

The `.describe()` function does exactly that: it provides purely descriptive information about the dataset. This information includes statistics summarizing the central tendency of the variable, their dispersion, the presence of empty values, and their shape.

```
df_LinkedIn_clean.describe()
```

	c_id	avg_time_in_previous_position	avg_current_position_leng
count	62706.000000	62706.000000	62706.0000
mean	31353.500000	1.194328	765.6341
std	18101.807327	0.506724	750.7091
min	1.000000	1.000000	-120.0000
25%	15677.250000	1.000000	274.0000
50%	31353.500000	1.000000	578.0000
75%	47029.750000	1.200000	1035.0000
max	62706.000000	15.000000	21884.0000

8 rows × 45 columns



You can check through the `.isnull().sum()` method if there are null values inside the DataFrame:

```
df_LinkedIn_clean.isnull().sum()
```

c_id	0
avg_time_in_previous_position	0
avg_current_position_length	0
avg_previous_position_length	0
m_urn_id	0
no_of_promotions	0
no_of_previous_positions	0
current_position_length	0
age	0
beauty	0
beauty_female	0
beauty_male	0
blur	0
emo_anger	0
emp_disgust	0
emo_fear	0
emo_happiness	0
emo_neutral	0
emo_sadness	0
emo_surprise	0
gender	0
head_pitch	0
head_roll	0
head_yaw	0
mouth_close	0
mouth_mask	0
mouth_open	0
mouth_other	0
skin_acne	0
skin_dark_circle	0
skin_health	0
skin_stain	0
smile	0
african	0
celtic_english	0
east_asian	0
european	0
greek	0
hispanic	0
jewish	0
muslim	0
nordic	0
south_asian	0
n_followers	0
face_quality	0
dtype:	int64

▼ 3.3 - Data Exploration

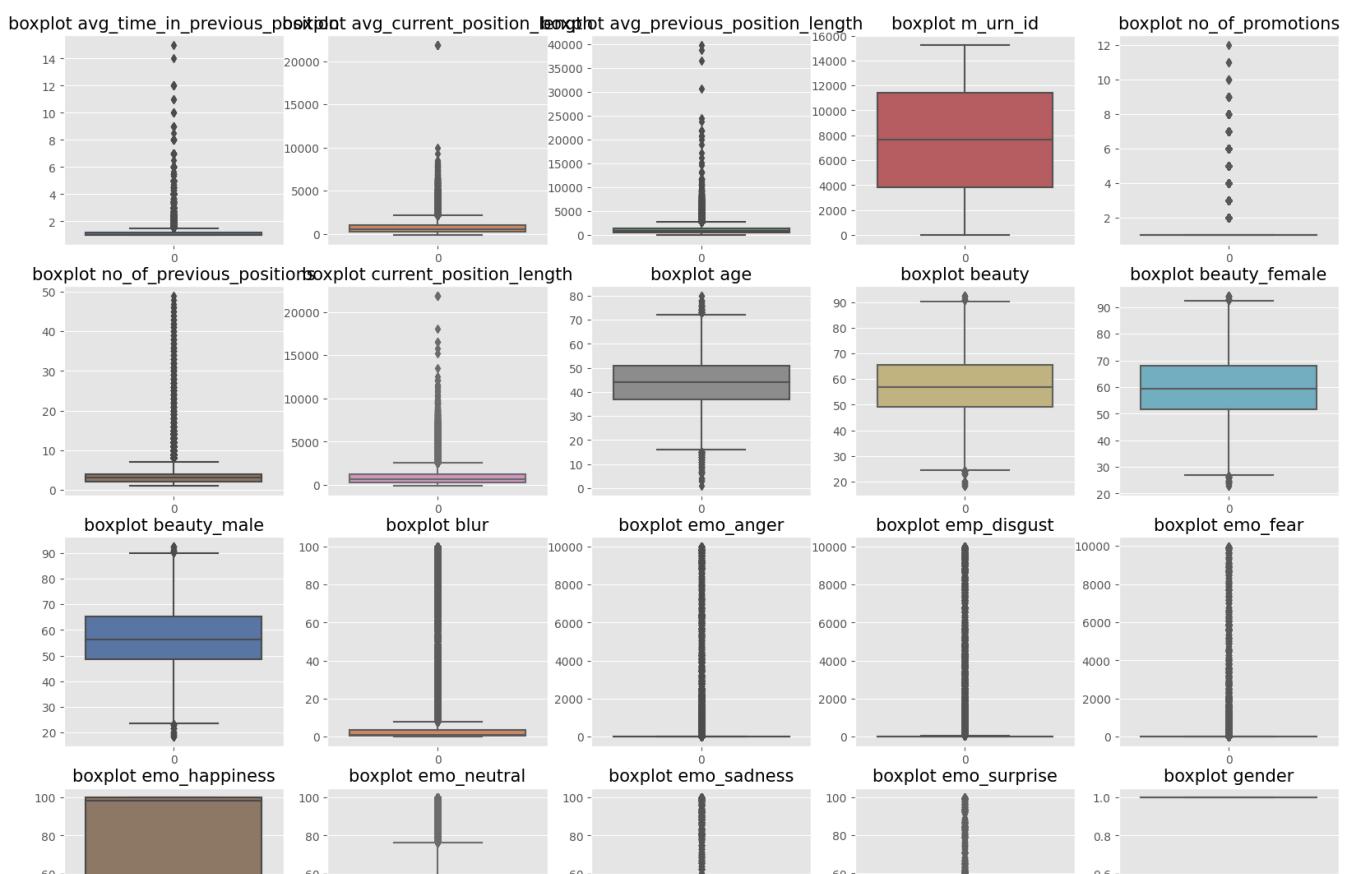
▼ Outliers

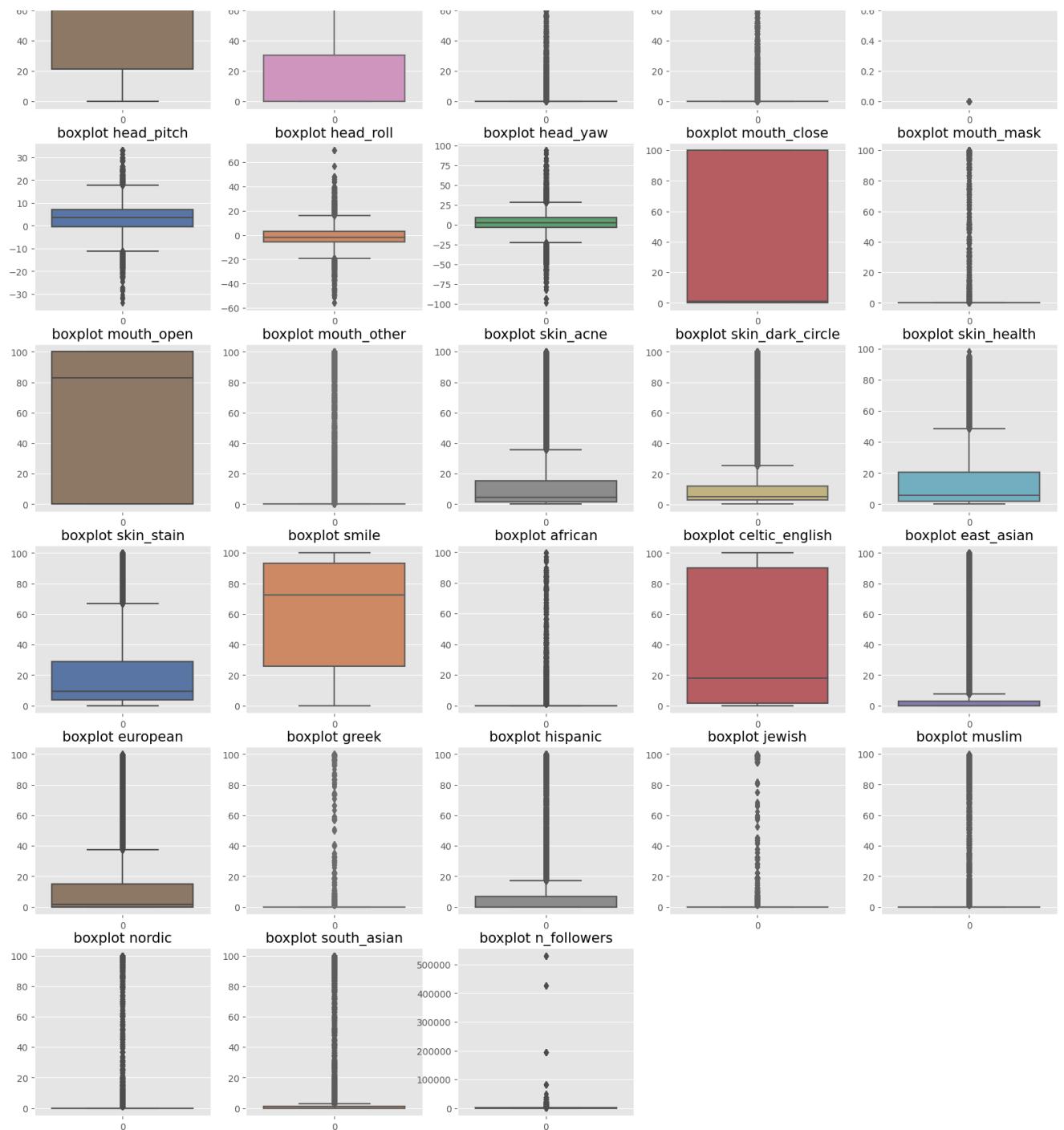
```
plt.subplots(figsize=(20,35))
c_ = sns.color_palette("deep", n_colors = 45)
i = 1

for col in df_LinkedIn_clean.columns[1:-1]:
    plt.subplot(9,5,i)
    sns.boxplot(data = df_LinkedIn_clean[col],color=c_[i-1])
    plt.title('boxplot '+ col, fontsize=15)
    i+=1

plt.suptitle("Boxplots features:", fontsize=24)
plt.show()
```

Boxplots features:





▼ Heatmap

As a first step, we start with the correlation between the target attribute (`n_followers`) and some features, representing it through 'heatmap' graphs:

```
df_corr_hs = df_LinkedIn_clean.iloc[:, :]
mat_hs = df_corr_hs.corr()
mat_hs_p = mat_hs[mat_hs.index=='n_followers']
mat_hs_p
```

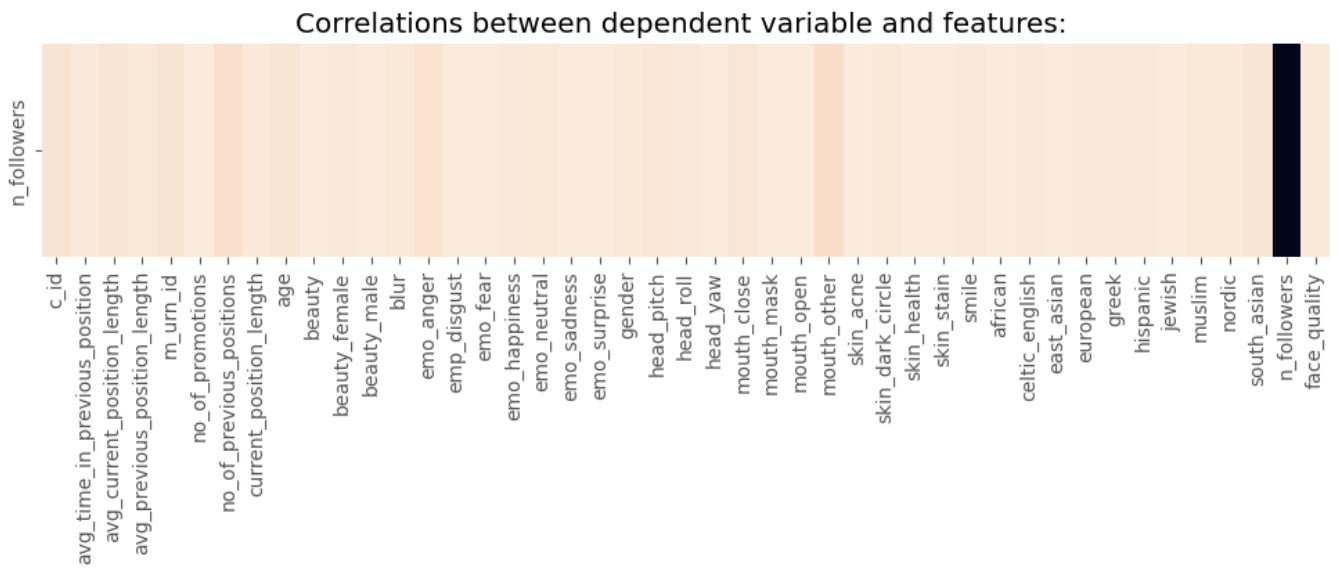
c_id	avg_time_in_previous_position	avg_current_position_len
n_followers	0.020338	-0.005338

1 rows × 45 columns



```
f, ax = plt.subplots(figsize=(12,2))
sns.heatmap(abs(mat_hs_p),
            ax = ax,
            cmap = sns.cm.rocket_r,
            cbar=False
            )

ax.set_title('Correlations between dependent variable and features: ')
plt.show()
```



Then we calculated the correlation matrix between all the features and showed them in the heatmap:

```
df_corr = df_LinkedIn_clean.iloc[:-1,:-1]
mat = df_corr.corr()
mat
```

	c_id	avg_time_in_previous_position	avg_curr
c_id	1.000000	-0.001257	
avg_time_in_previous_position	-0.001257	1.000000	
avg_current_position_length	0.010797	-0.051859	

	avg_previous_position_length	0.001000
avg_previous_position_length	-0.003919	0.416505
m_urn_id	0.692419	-0.006169
no_of_promotions	0.004927	0.012136
no_of_previous_positions	-0.002354	-0.049366
current_position_length	0.009434	-0.039971
age	-0.003121	0.038515
beauty	-0.016537	-0.025814
beauty_female	-0.008958	-0.020051
beauty_male	-0.015504	-0.027871
blur	0.010409	-0.019841
emo_anger	-0.002093	-0.008243
emo_disgust	-0.017102	-0.009419
emo_fear	0.000031	-0.007316
emo_happiness	0.009099	0.032024
emo_neutral	-0.000218	-0.026279
emo_sadness	-0.006990	-0.007601
emo_surprise	-0.009085	-0.004101
gender	0.008949	-0.001362
head_pitch	0.002288	-0.001787
head_roll	-0.014397	0.006674
head_yaw	0.008081	-0.009377
mouth_close	0.002663	-0.026970
mouth_mask	-0.017018	-0.005897
mouth_open	0.000974	0.027088
mouth_other	-0.000439	0.000614
skin_acne	0.006718	-0.000115
skin_dark_circle	0.013783	0.001657
skin_health	-0.021904	-0.013224
skin_stain	0.007939	0.020032

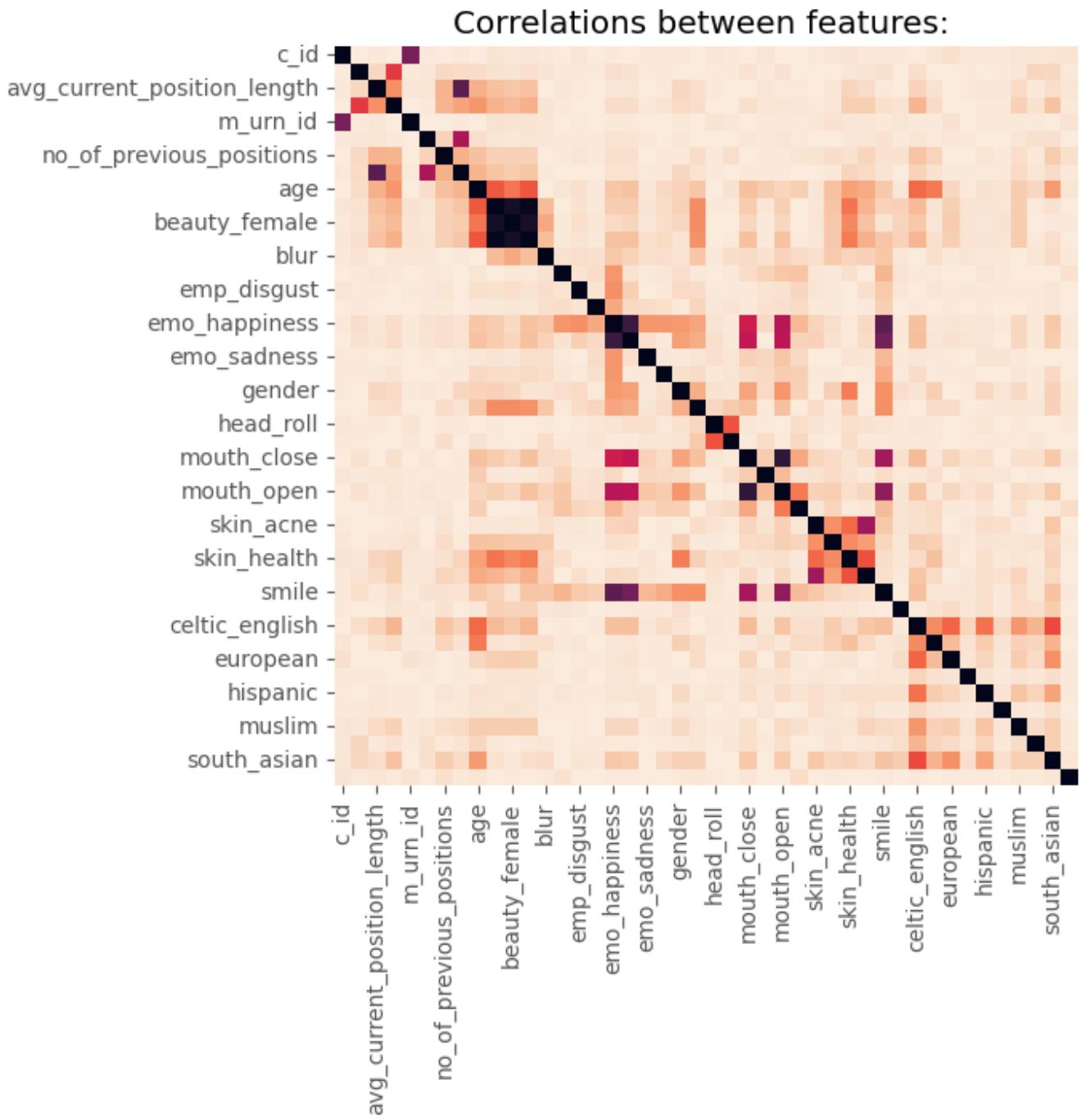
smile	0.006928	0.029563
african	0.010207	0.001705
celtic_english	0.002722	0.031994
east_asian	0.019252	-0.002799
european	-0.029367	-0.000629
greek	0.008192	0.007766
hispanic	0.004496	-0.001537
jewish	0.009763	-0.007836
muslim	0.017726	-0.014504
nordic	0.013407	0.056782
south_asian	-0.015698	-0.050940
n_followers	0.020343	-0.005339

44 rows × 44 columns



```
f, ax = plt.subplots(figsize=(6,6))
sns.heatmap(abs(mat),
            ax = ax,
            cmap = sns.cm.rocket_r,
            cbar=False
            )

ax.set_title('Correlations between features: ')
plt.show()
```



▼ Scatterplot

```
plt.subplots(figsize=(20,60))
c_ = sns.color_palette("deep", n_colors=45)
i = 1

for col in df_LinkedIn_clean.columns[1:-1]:
```

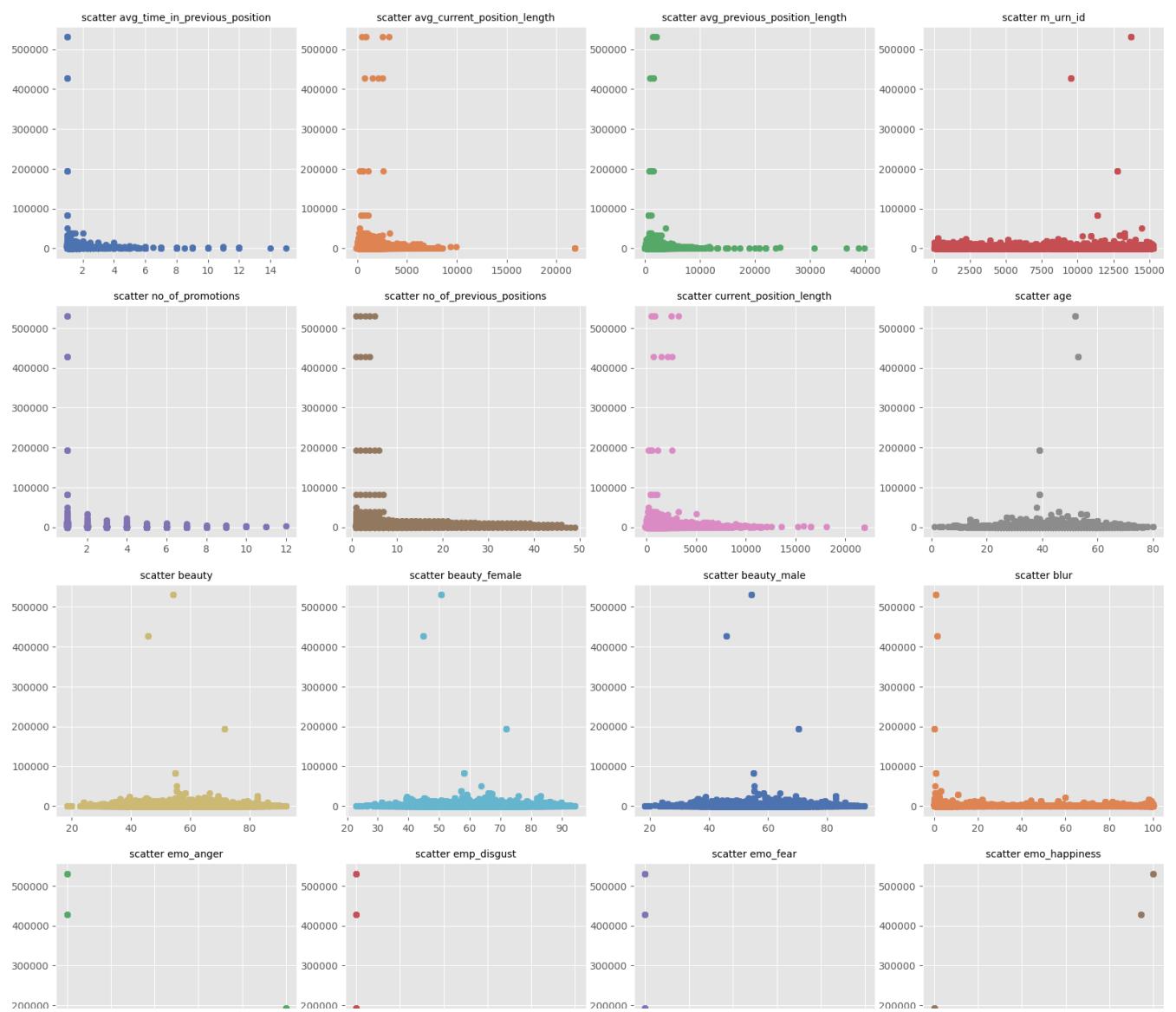
```

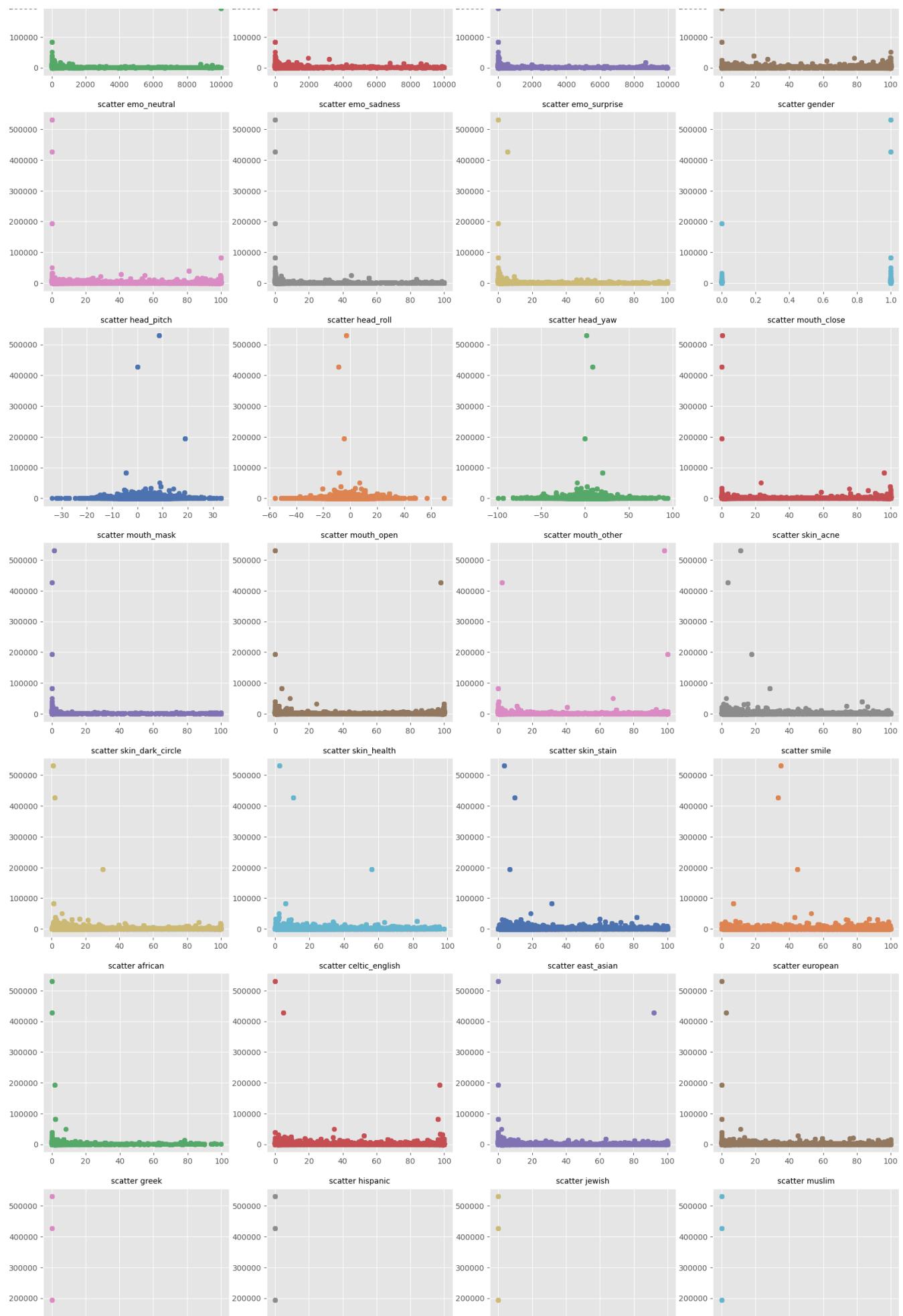
plt.subplot(12,4,i)
plt.scatter(df_LinkedIn_clean[col], df_LinkedIn_clean['n_followers'], color=c)
plt.title('scatter '+ col, fontsize=10)
i+=1

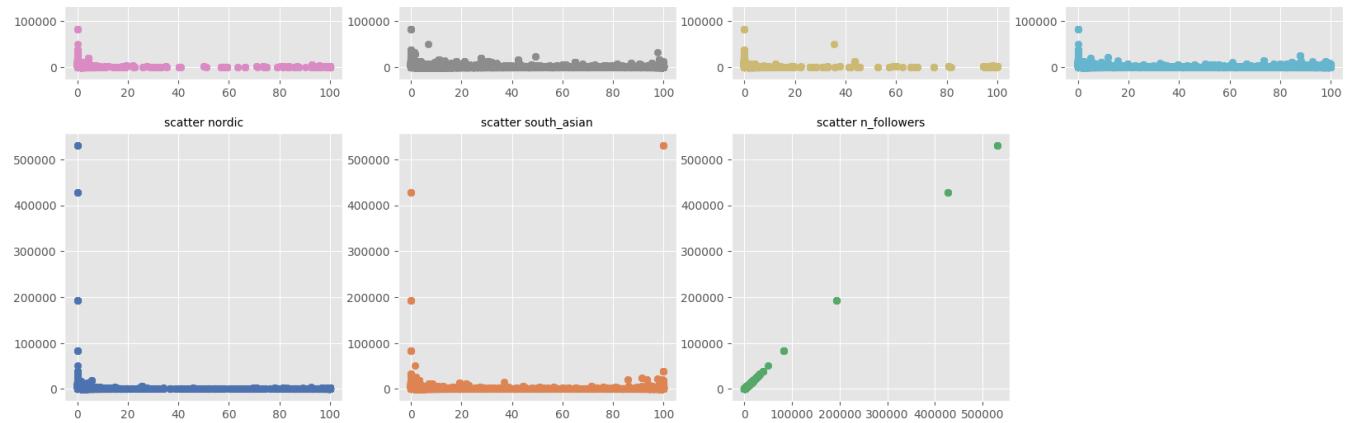
plt.suptitle("Scatters features:", fontsize=30)
plt.show()

```

Scatters features:







▼ 04 - REGRESSION

▼ General

You import the libraries needed for machine learning classification algorithm implementations.

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn import tree

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import median_absolute_error
```

The new DataFrame is formed by eliminating the features that could have caused redundancy:

```
df_LinkedIn_profile = df_LinkedIn_clean[['c_id', 'avg_time_in_previous_position',
                                         'current_position_length', 'age', 'bea',
                                         'gender', 'head_pitch', 'head_roll', 'skin_stain', 'smile', 'african', 'cel
```

▼ Employment_Data [1st case]

▼ General

The first set of attributes is related to employment data:

```
X_1 = df_LinkedIn_profile[['c_id', 'avg_time_in_previous_position', 'avg_current
y_1 = df_LinkedIn_profile['n_followers'].to_numpy()
```

We normalize the data with the min max scaler function

```
scaler = MinMaxScaler()
scaler.fit(X_1)
X_1 = scaler.transform(X_1)
```

X_1

```
array([[0.0000000e+00, 7.14285714e-02, 2.62225050e-02, ...,
       0.0000000e+00, 2.62225050e-02, 4.55696203e-01],
      [1.59476916e-05, 3.57142857e-02, 1.50881658e-02, ...,
       2.08333333e-02, 1.50881658e-02, 4.55696203e-01],
      [3.18953831e-05, 2.38095238e-02, 1.64970005e-02, ...,
       4.16666667e-02, 1.64970005e-02, 4.55696203e-01],
      ...,
      [9.99968105e-01, 0.00000000e+00, 1.10434466e-02, ...,
       1.25000000e-01, 1.10434466e-02, 7.46835443e-01],
      [9.99984052e-01, 0.00000000e+00, 1.03662970e-01, ...,
       1.45833333e-01, 1.03662970e-01, 7.46835443e-01],
      [1.00000000e+00, 0.00000000e+00, 3.32212325e-02, ...,
       1.66666667e-01, 3.32212325e-02, 7.46835443e-01]])
```

y_1

```
array([420, 420, 420, ..., 279, 279, 279])
```

```
print('shape of X_1:')
print(X_1.shape)
print('type of X_1:')
print(type(X_1))
print('shape of y_1:')
print(y_1.shape)
print('type of y_1:')
print(type(y_1))
```

```
shape of X_1:
(62706, 8)
type of X_1:
<class 'numpy.ndarray'>
shape of y_1:
(62706,)
type of y_1:
<class 'numpy.ndarray'>
```

We test the train_test_split function in order to check if the set created are the same in each iteration:

```
X_trainval_1, X_test_1, y_trainval_1, y_test_1 = train_test_split(X_1, y_1, test_size=0.2)
X_train_1, X_val_1, y_train_1, y_val_1 = train_test_split(X_trainval_1, y_trainval_1, test_size=0.2)
```

```
print(len(X_train_1))
print(len(X_test_1))
print(len(X_val_1))
```

```
39504
6271
16931
```

```
X_train_1
```

```
array([[0.64399968, 0.01428571, 0.00822578, ..., 0.08333333, 0.00822578,
       0.49367089],
       [0.09327805, 0.           , 0.06080713, ..., 0.04166667, 0.06080713,
       0.59493671],
       [0.01840364, 0.           , 0.07044174, ..., 0.125       , 0.07044174,
       0.6835443 ],
       ...,
       [0.55056216, 0.           , 0.07044174, ..., 0.02083333, 0.07044174,
       0.49367089],
       [0.75006778, 0.           , 0.04331031, ..., 0.02083333, 0.19473732,
       0.64556962],
       [0.89275177, 0.07142857, 0.03722051, ..., 0.           , 0.03722051,
       0.26582278]])
```

```
X_test_1
```

```
array([[0.82754166, 0.03571429, 0.23513907, ..., 0.02083333, 0.23513907,
       0.59493671],
       [0.84385615, 0.           , 0.03632673, ..., 0.04166667, 0.09807308,
       0.74683544],
       [0.90453712, 0.           , 0.11338847, ..., 0.04166667, 0.11338847,
       0.59493671],
       ...,
       [0.05819313, 0.           , 0.00686239, ..., 0.02083333, 0.00686239,
       0.26582278],
       [0.6687186 , 0.           , 0.08155335, ..., 0.04166667, 0.15765315,
       0.44303797],
       [0.50474444, 0.01785714, 0.02758589, ..., 0.0625     , 0.02758589,
       0.64556962]])
```

```
X_val_1
```

```
array([[0.74558648, 0.03571429, 0.02613161, ..., 0.0625      , 0.02613161,
       0.35443038],
       [0.50007176, 0.          , 0.05385384, ..., 0.          , 0.05385384,
       0.40506329],
       [0.19649151, 0.          , 0.08843847, ..., 0.02083333, 0.08843847,
       0.65822785],
       ...,
       [0.94394386, 0.          , 0.01508817, ..., 0.04166667, 0.01508817,
       0.65822785],
       [0.74911092, 0.07142857, 0.027495  , ..., 0.04166667, 0.027495  ,
       0.46835443],
       [0.01811658, 0.          , 0.0373114 , ..., 0.04166667, 0.0373114 ,
       0.48101266]])
```

▼ Liner Regression

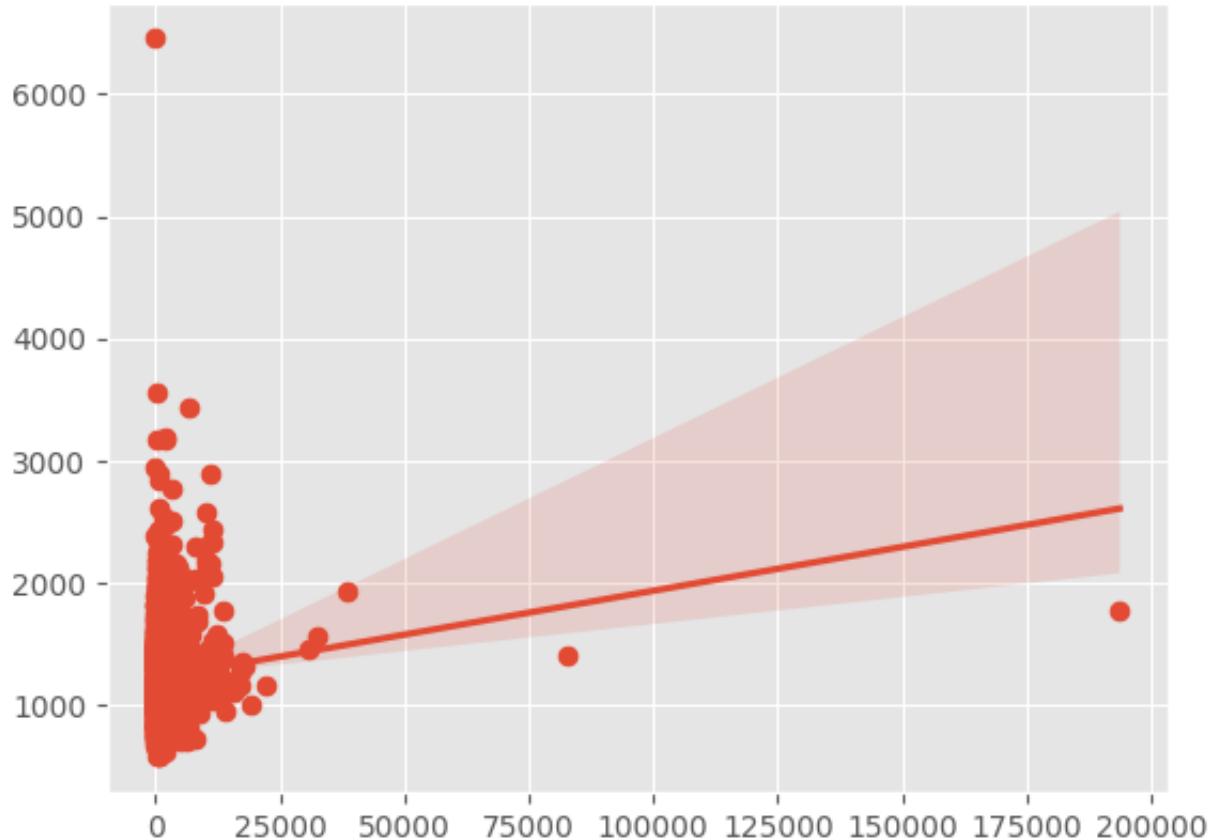
```
for i in range (0,10):
    lir = LinearRegression()
    lir.fit(X_train_1, y_train_1)
    y_pred_1 = lir.predict(X_test_1)

print(' --- METRICHE ---')
print(mean_absolute_error(y_test_1, y_pred_1))
print(mean_squared_error(y_test_1, y_pred_1))
print(median_absolute_error(y_test_1, y_pred_1))

--- METRICHE ---
906.6698816234795
9735401.10668005
634.9539407774275
```

```
plt.scatter(y_test_1, y_pred_1)
sns.regplot(x=y_test_1, y=y_pred_1, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



▼ Logistic Regression

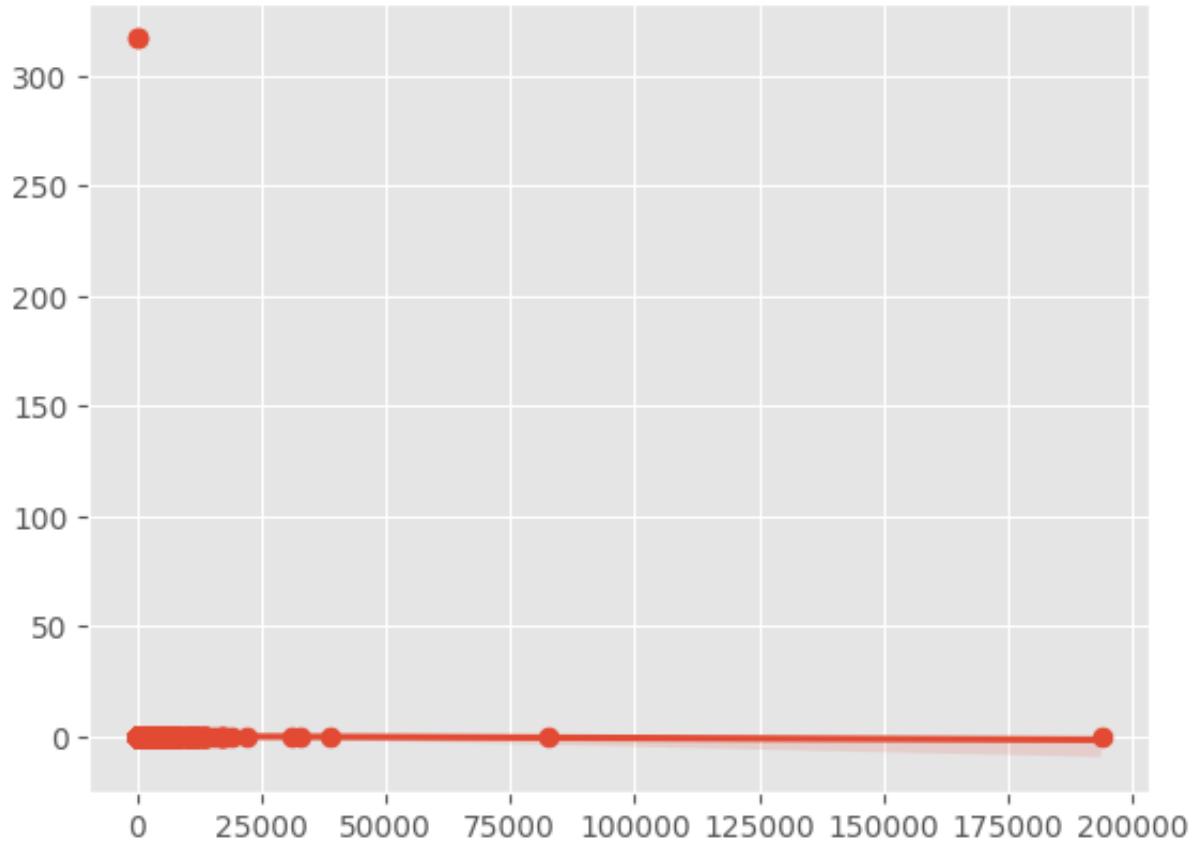
```
lor = LogisticRegression()
lor.fit(X_train_1, y_train_1)
y_pred_1 = lor.predict(X_test_1)

print(' --- METRICHE ---')
print(mean_absolute_error(y_test_1, y_pred_1))
print(mean_squared_error(y_test_1, y_pred_1))
print(median_absolute_error(y_test_1, y_pred_1))

    --- METRICHE ---
1166.4831765268698
11158771.68489874
718.0
```

```
plt.scatter(y_test_1, y_pred_1)
sns.regplot(x=y_test_1, y=y_pred_1, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



▼ KNeighbors Regressor

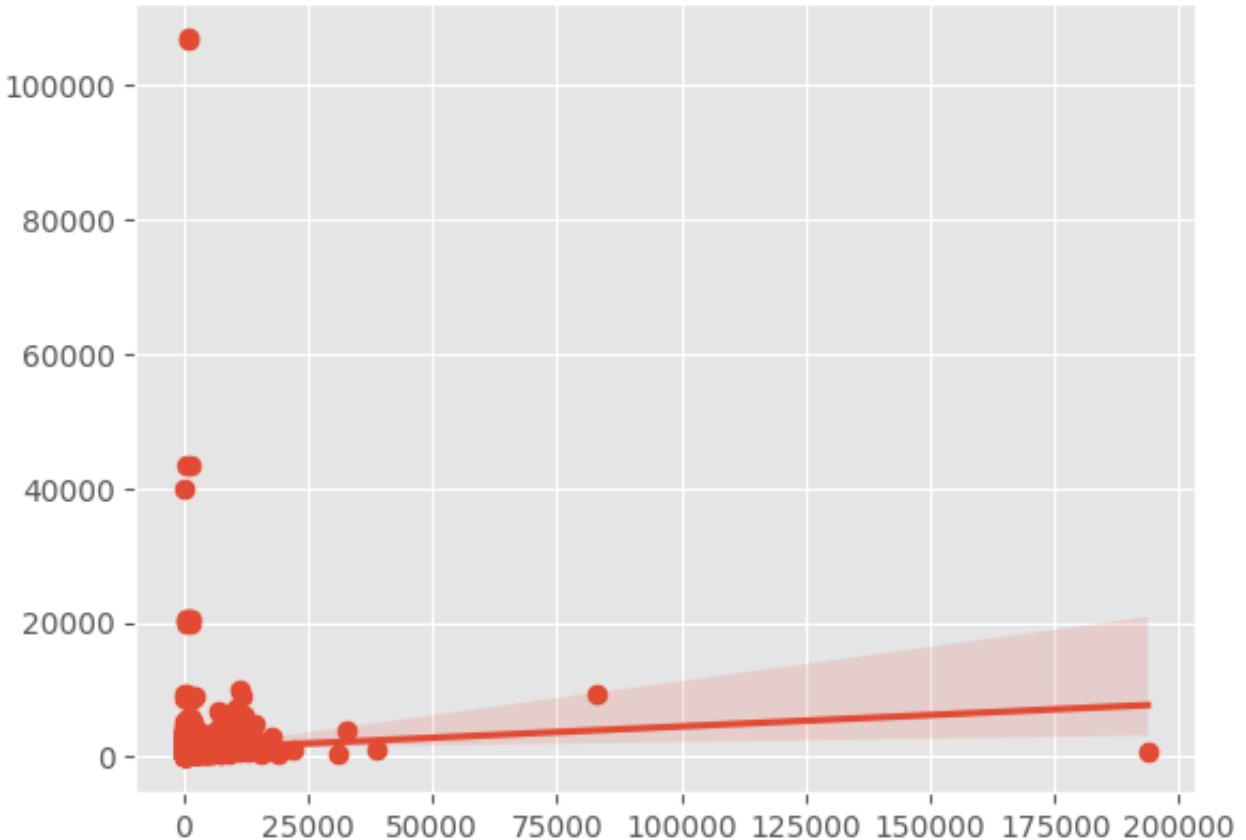
```
for i in range(0,10):
    knn = KNeighborsRegressor(n_neighbors = 10)
    knn.fit(X_trainval_1, y_trainval_1)
    y_pred_1 = knn.predict(X_test_1)

    print(' --- METRICHE ---')
    print(mean_absolute_error(y_test_1, y_pred_1))
    print(mean_squared_error(y_test_1, y_pred_1))
    print(median_absolute_error(y_test_1, y_pred_1))

    --- METRICHE ---
    952.7671503747409
    14421327.036415242
    498.9
```

```
plt.scatter(y_test_1, y_pred_1)
sns.regplot(x=y_test_1, y=y_pred_1, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



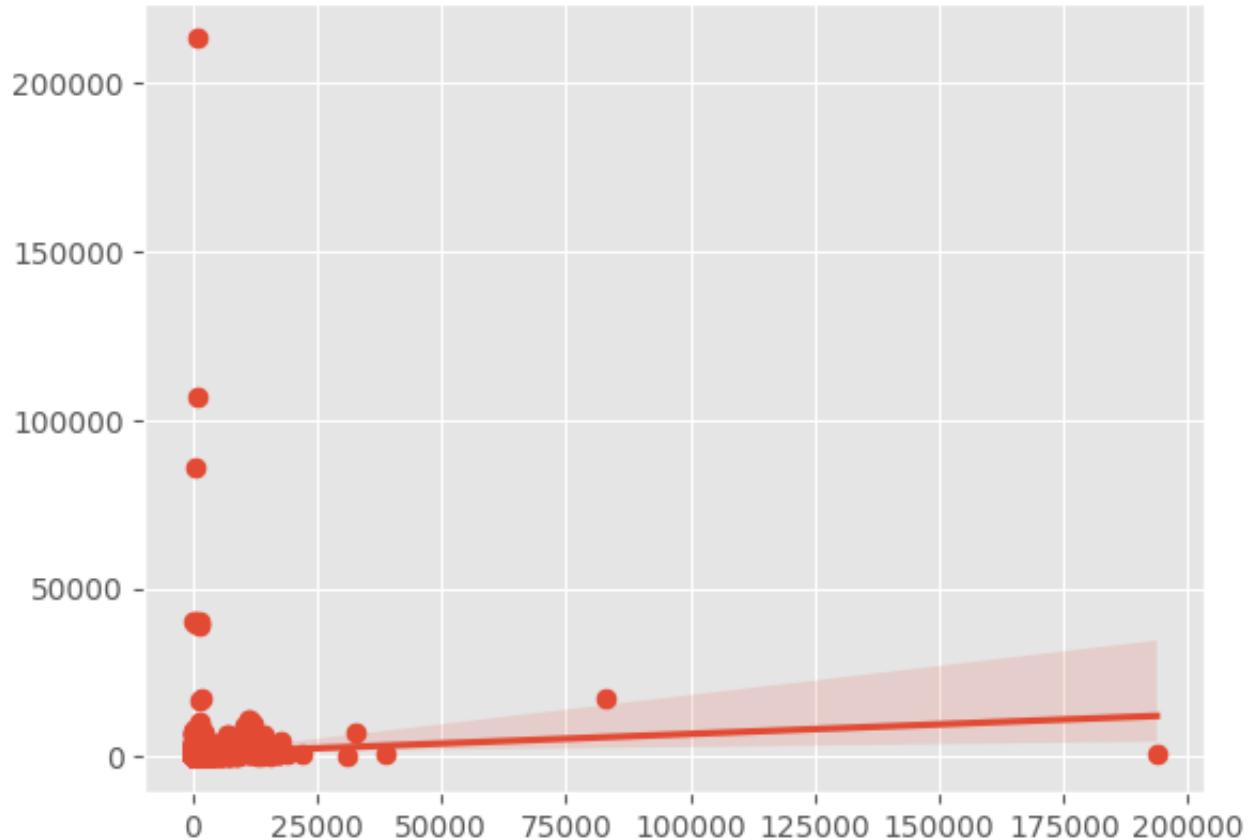
```
for i in range(0,10):
    knn = KNeighborsRegressor(n_neighbors = 5)
    knn.fit(X_trainval_1, y_trainval_1)
    y_pred_1 = knn.predict(X_test_1)

    print(' --- METRICHE ---')
    print(mean_absolute_error(y_test_1, y_pred_1))
    print(mean_squared_error(y_test_1, y_pred_1))
    print(median_absolute_error(y_test_1, y_pred_1))

    --- METRICHE ---
    982.3101578695583
    21072565.318813585
    471.2000000000005
```

```
plt.scatter(y_test_1, y_pred_1)
sns.regplot(x=y_test_1, y=y_pred_1, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



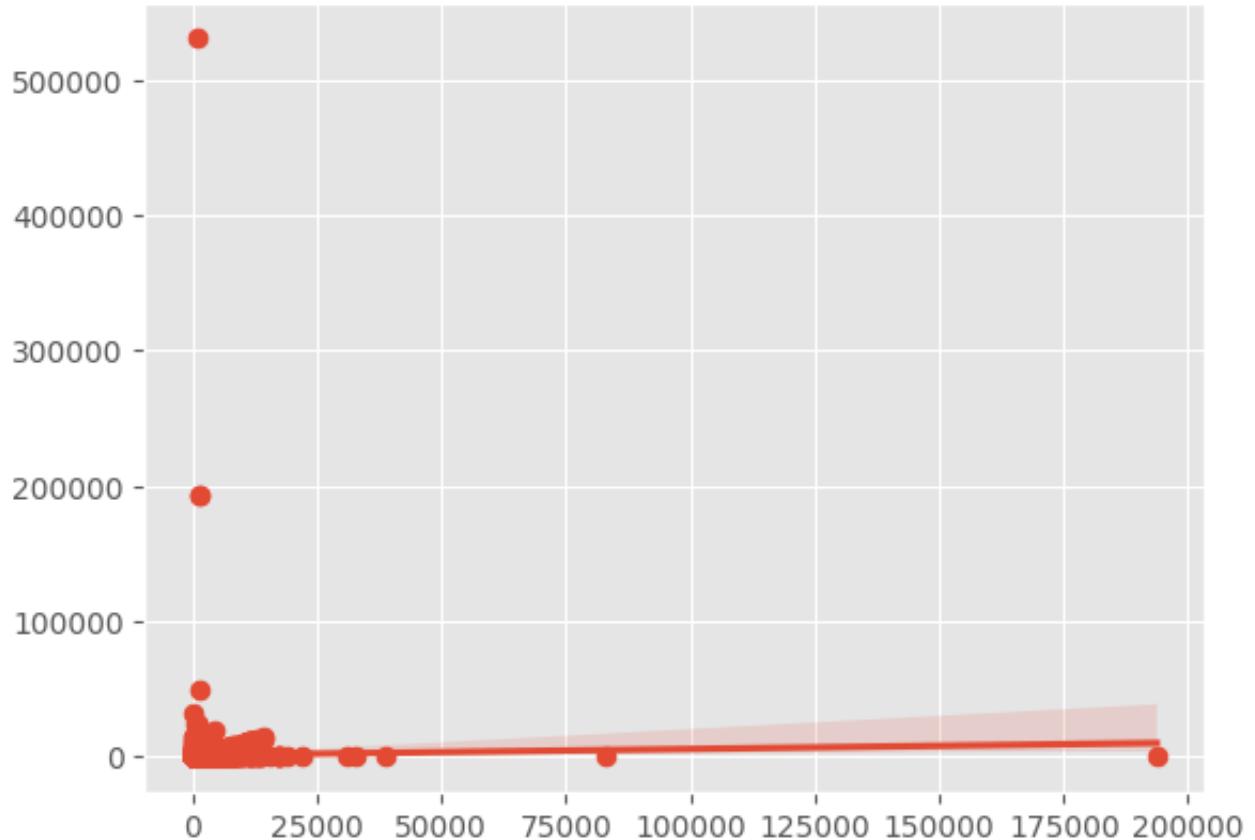
```
for i in range(0,10):
    knn = KNeighborsRegressor(n_neighbors = 1)
    knn.fit(X_trainval_1, y_trainval_1)
    y_pred_1 = knn.predict(X_test_1)

    print(' --- METRICHE ---')
    print(mean_absolute_error(y_test_1, y_pred_1))
    print(mean_squared_error(y_test_1, y_pred_1))
    print(median_absolute_error(y_test_1, y_pred_1))

    --- METRICHE ---
    1134.5082124063147
    68407219.51028544
    456.0
```

```
plt.scatter(y_test_1, y_pred_1)
sns.regplot(x=y_test_1, y=y_pred_1, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



▼ Decision Tree

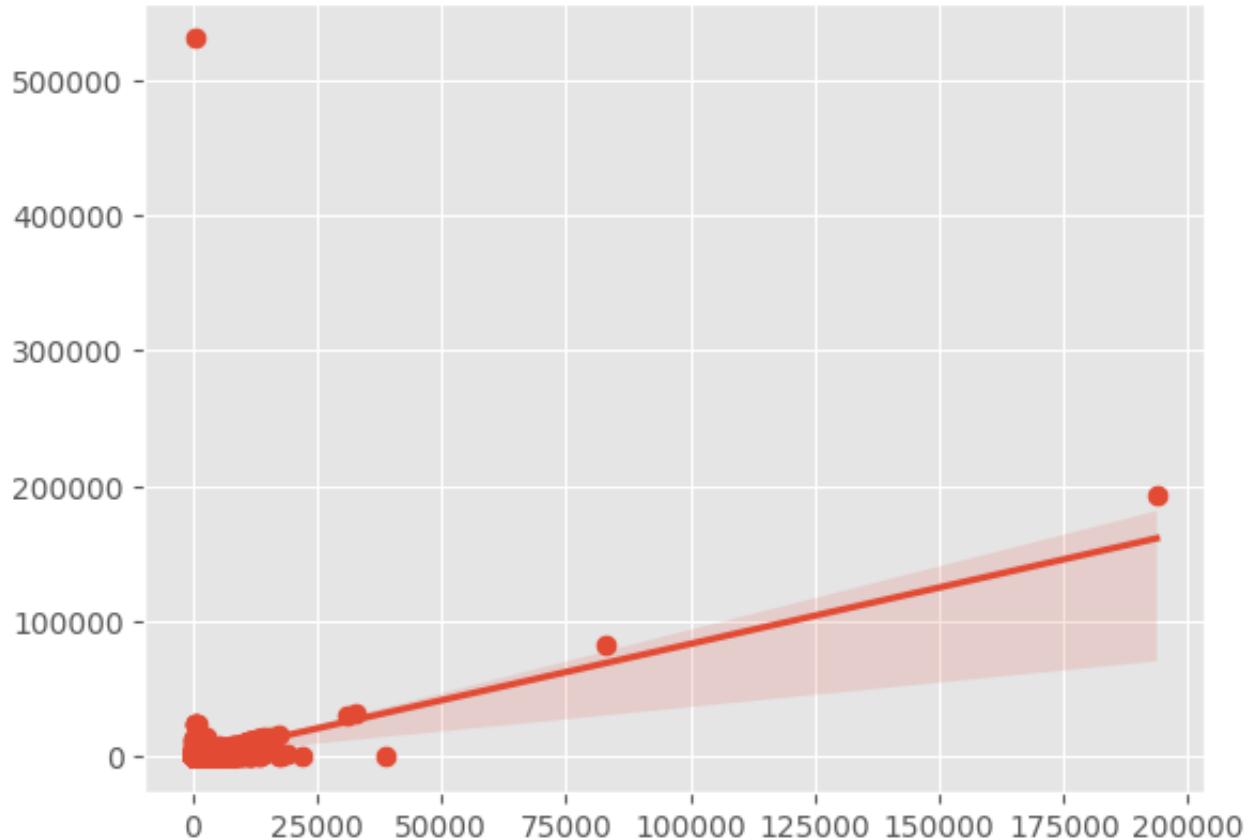
```
for i in range(0,10):
    dt = DecisionTreeRegressor()
    dt.fit(X_trainval_1, y_trainval_1)
    y_pred_1 = dt.predict(X_test_1)

    print(' --- METRICHE ---')
    print(mean_absolute_error(y_test_1, y_pred_1))
    print(mean_squared_error(y_test_1, y_pred_1))
    print(median_absolute_error(y_test_1, y_pred_1))

    --- METRICHE ---
    846.6354648381439
    48114779.163929194
    318.0
```

```
plt.scatter(y_test_1, y_pred_1)
sns.regplot(x=y_test_1, y=y_pred_1, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



▼ Random Forest

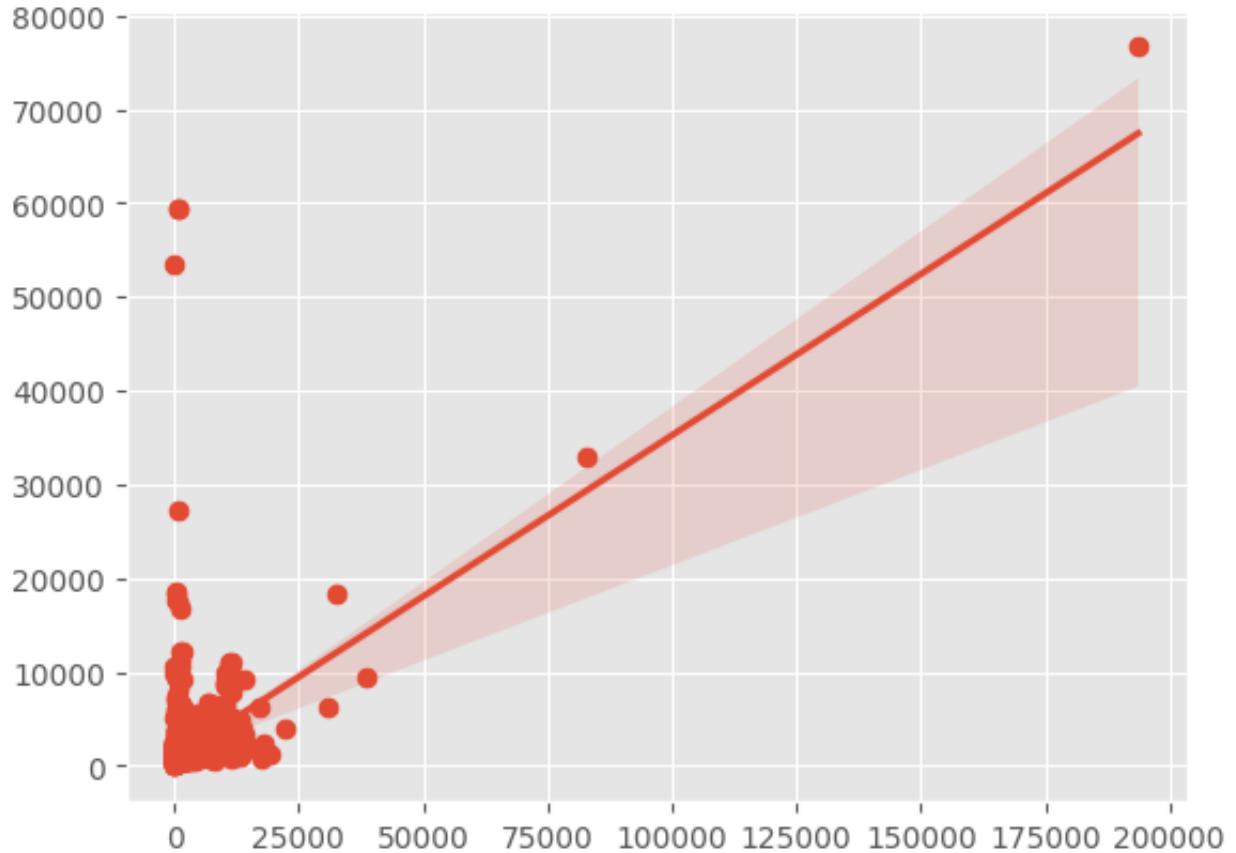
```
for i in range(0,10):
    rf = RandomForestRegressor()
    rf.fit(X_trainval_1, y_trainval_1)
    y_pred_1 = rf.predict(X_test_1)

    print(' --- METRICHE ---')
    print(mean_absolute_error(y_test_1, y_pred_1))
    print(mean_squared_error(y_test_1, y_pred_1))
    print(median_absolute_error(y_test_1, y_pred_1))

    --- METRICHE ---
    830.660510285441
    6676461.391370275
    504.819999999994
```

```
plt.scatter(y_test_1, y_pred_1)
sns.regplot(x=y_test_1, y=y_pred_1, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



▼ User_Data [2nd case]

▼ General

The second set of attributes is that relating to general user data:

```
X_2 = df_LinkedIn_profile[['c_id', 'age', 'gender', 'african', 'celtic_english',
y_2 = df_LinkedIn_profile['n_followers'].to_numpy()
```

We normalize the data with the min max scaler function

```
scaler = MinMaxScaler()
scaler.fit(X_2)
X_2 = scaler.transform(X_2)
```

X_2

```
array([[0.0000000e+00, 4.55696203e-01, 1.0000000e+00, ...,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00],
      [1.59476916e-05, 4.55696203e-01, 1.0000000e+00, ...,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00],
      [3.18953831e-05, 4.55696203e-01, 1.0000000e+00, ...,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00],
      ...,
      [9.99968105e-01, 7.46835443e-01, 1.0000000e+00, ...,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00],
      [9.99984052e-01, 7.46835443e-01, 1.0000000e+00, ...,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00],
      [1.0000000e+00, 7.46835443e-01, 1.0000000e+00, ...,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00]])
```

y_2

```
array([420, 420, 420, ..., 279, 279, 279])
```

```
print('shape of X_2:')
print(X_2.shape)
print('type of X_2:')
print(type(X_2))
print('shape of y_2:')
print(y_2.shape)
print('type of y_2:')
print(type(y_2))
```

```
shape of X_2:
(62706, 13)
type of X_2:
<class 'numpy.ndarray'>
shape of y_2:
(62706,)
type of y_2:
<class 'numpy.ndarray'>
```

We test the train_test_split function in order to check if the set created are the same in each iteration:

```
X_trainval_2, X_test_2, y_trainval_2, y_test_2 = train_test_split(X_2, y_2, test_size=0.2, random_state=42)
X_train_2, X_val_2, y_train_2, y_val_2 = train_test_split(X_trainval_2, y_trainval_2, test_size=0.2, random_state=42)
```

```
print(len(X_train_2))
print(len(X_test_2))
print(len(X_val_2))
```

```
39504
6271
16931
```

X_train_2

```
array([[0.64399968, 0.49367089, 0.04160416], [0.09327805, 0.59493671, 1.0], [0.01840364, 0.6835443 , 1.0], ..., [0.55056216, 0.49367089, 1.0], [0.75006778, 0.64556962, 1.0], [0.89275177, 0.26582278, 1.0]]])
```

X_test_2

```
array([[0.82754166, 0.59493671, 1.0], [0.84385615, 0.74683544, 1.0], [0.90453712, 0.59493671, 1.0], ..., [0.05819313, 0.26582278, 1.0], [0.6687186 , 0.44303797, 1.0], [0.50474444, 0.64556962, 1.0], ..., [0.01370137], [0.50474444, 0.64556962, 1.0], ..., [0.01370137], [0.50474444, 0.64556962, 1.0], ..., [0.01370137]]])
```

```
X_val_2
```

```
array([[0.74558648, 0.35443038, 1.         , ..., 0.         , 0.         ,
       0.99949995],
      [0.50007176, 0.40506329, 1.         , ..., 0.         , 0.         ,
       0.99389939],
      [0.19649151, 0.65822785, 0.         , ..., 0.03665498, 0.         ,
       0.03830383],
      ...,
      [0.94394386, 0.65822785, 1.         , ..., 0.         , 0.         ,
       0.         ],
      [0.74911092, 0.46835443, 1.         , ..., 0.         , 0.         ,
       0.         ],
      [0.01811658, 0.48101266, 1.         , ..., 0.         , 0.         ,
       0.         ]])
```

▼ Liner Regression

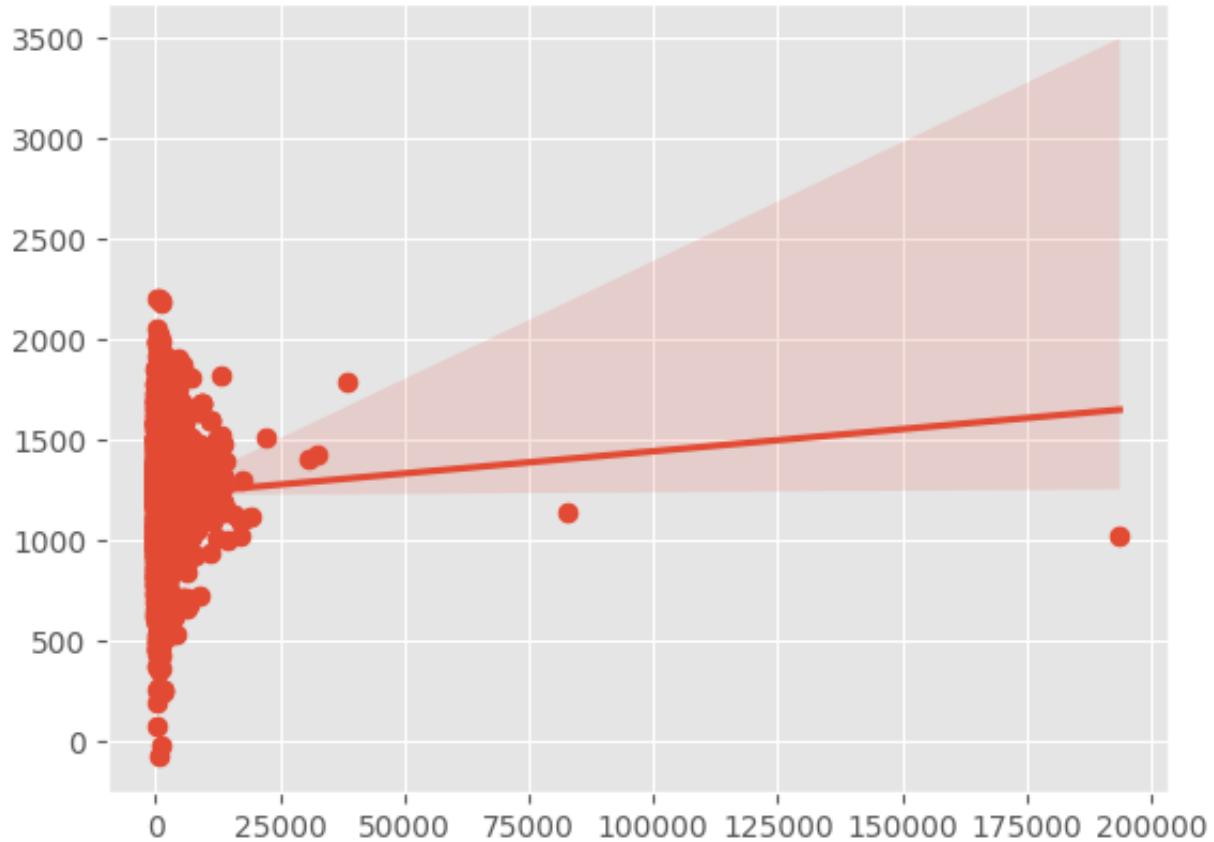
```
for i in range(0,10):
    lir = LinearRegression()
    lir.fit(X_train_2, y_train_2)
    y_pred_2 = lir.predict(X_test_2)

print(' --- METRICHE ---')
print(mean_absolute_error(y_test_2, y_pred_2))
print(mean_squared_error(y_test_2, y_pred_2))
print(median_absolute_error(y_test_2, y_pred_2))

--- METRICHE ---
919.8008264198359
9827290.155282123
654.3792133310501
```

```
plt.scatter(y_test_2, y_pred_2)
sns.regplot(x=y_test_2, y=y_pred_2, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



▼ Logistic Regression

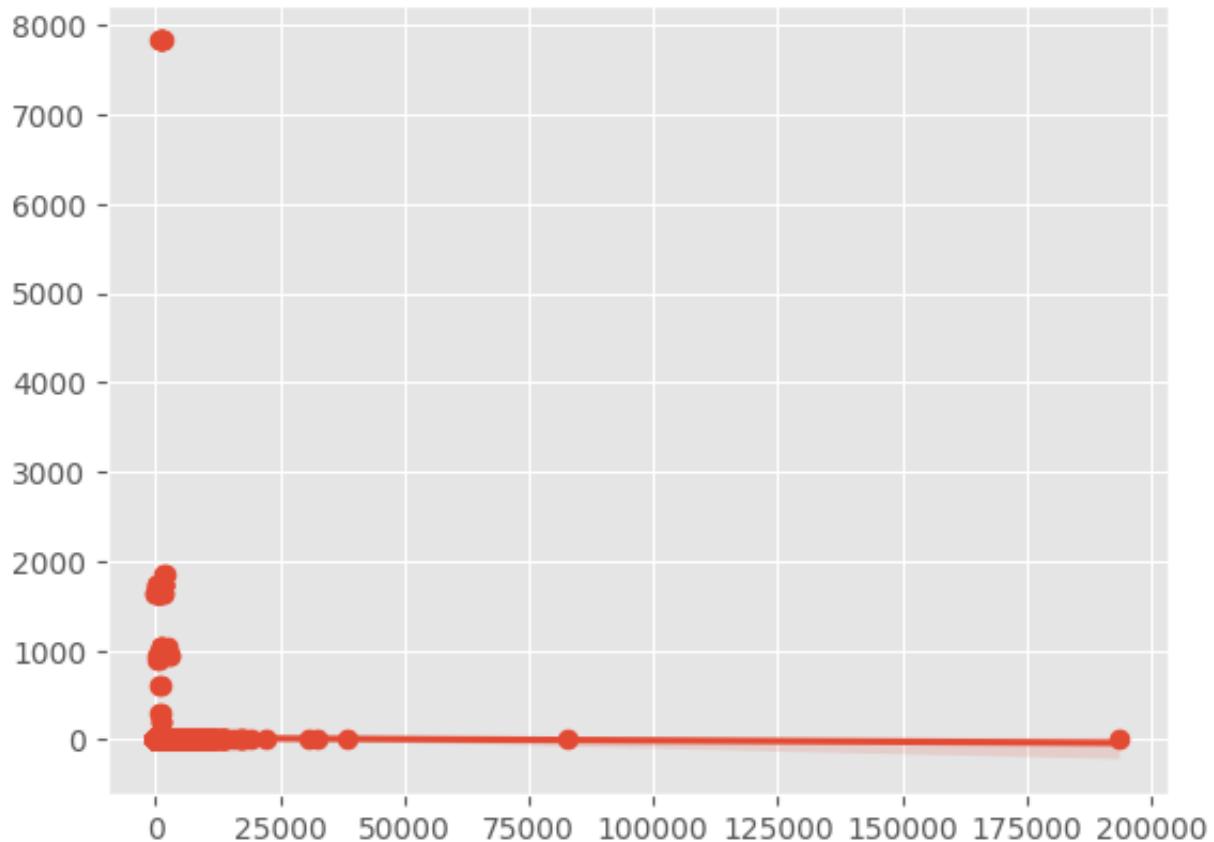
```
lor = LogisticRegression()
lor.fit(X_train_2, y_train_2)
y_pred_2 = lor.predict(X_test_2)

print(' --- METRICHE ---')
print(mean_absolute_error(y_test_2, y_pred_2))
print(mean_squared_error(y_test_2, y_pred_2))
print(median_absolute_error(y_test_2, y_pred_2))

--- METRICHE ---
1169.687290703237
11202810.052782651
715.0
```

```
plt.scatter(y_test_2, y_pred_2)
sns.regplot(x=y_test_2, y=y_pred_2, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



▼ KNeighbors Regressor

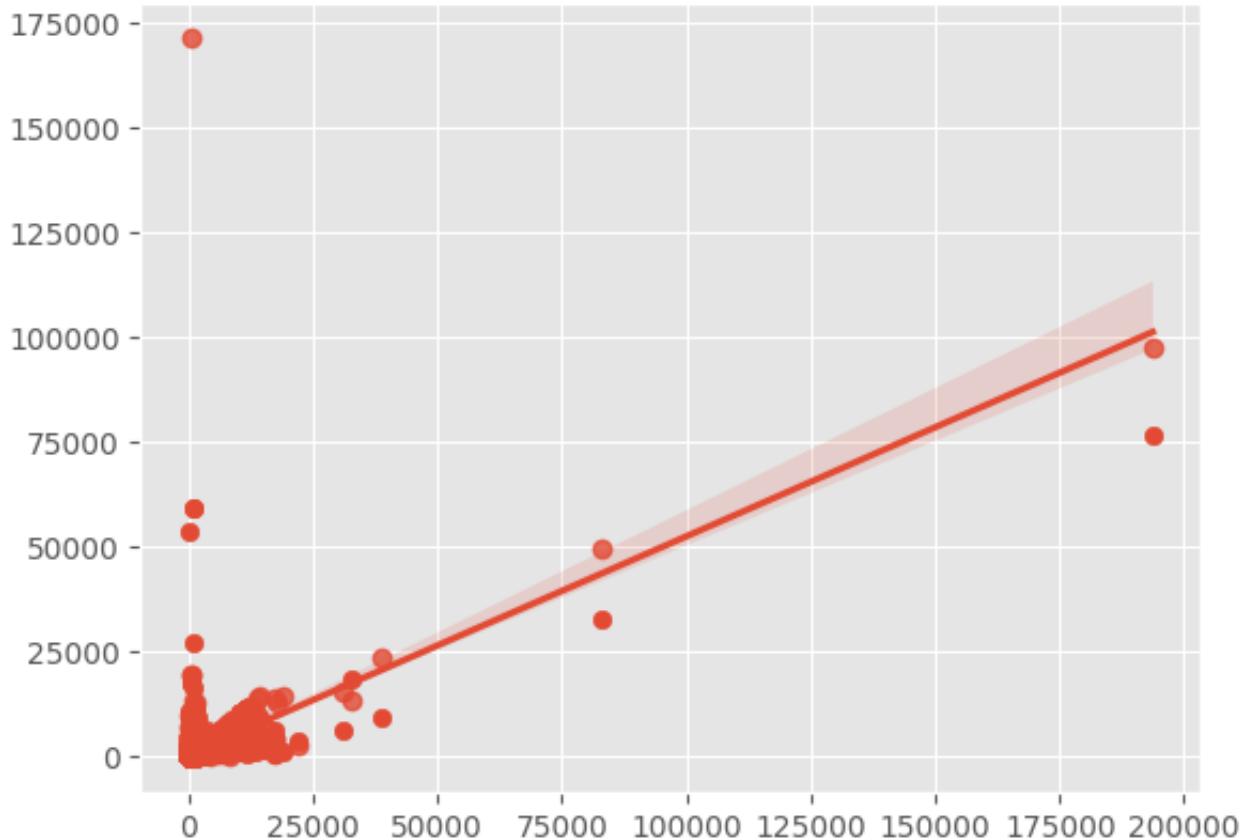
```
for i in range(0,10):
    knn = KNeighborsRegressor(n_neighbors = 10)
    knn.fit(X_trainval_2, y_trainval_2)
    y_pred_2 = knn.predict(X_test_2)

    print(' --- METRICHE ---')
    print(mean_absolute_error(y_test_2, y_pred_2))
    print(mean_squared_error(y_test_2, y_pred_2))
    print(median_absolute_error(y_test_2, y_pred_2))

    --- METRICHE ---
    599.2180672938925
    7763964.843420507
    266.7999999999995
```

```
plt.scatter(y_test_1, y_pred_1)
sns.regplot(x=y_test_2, y=y_pred_2, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



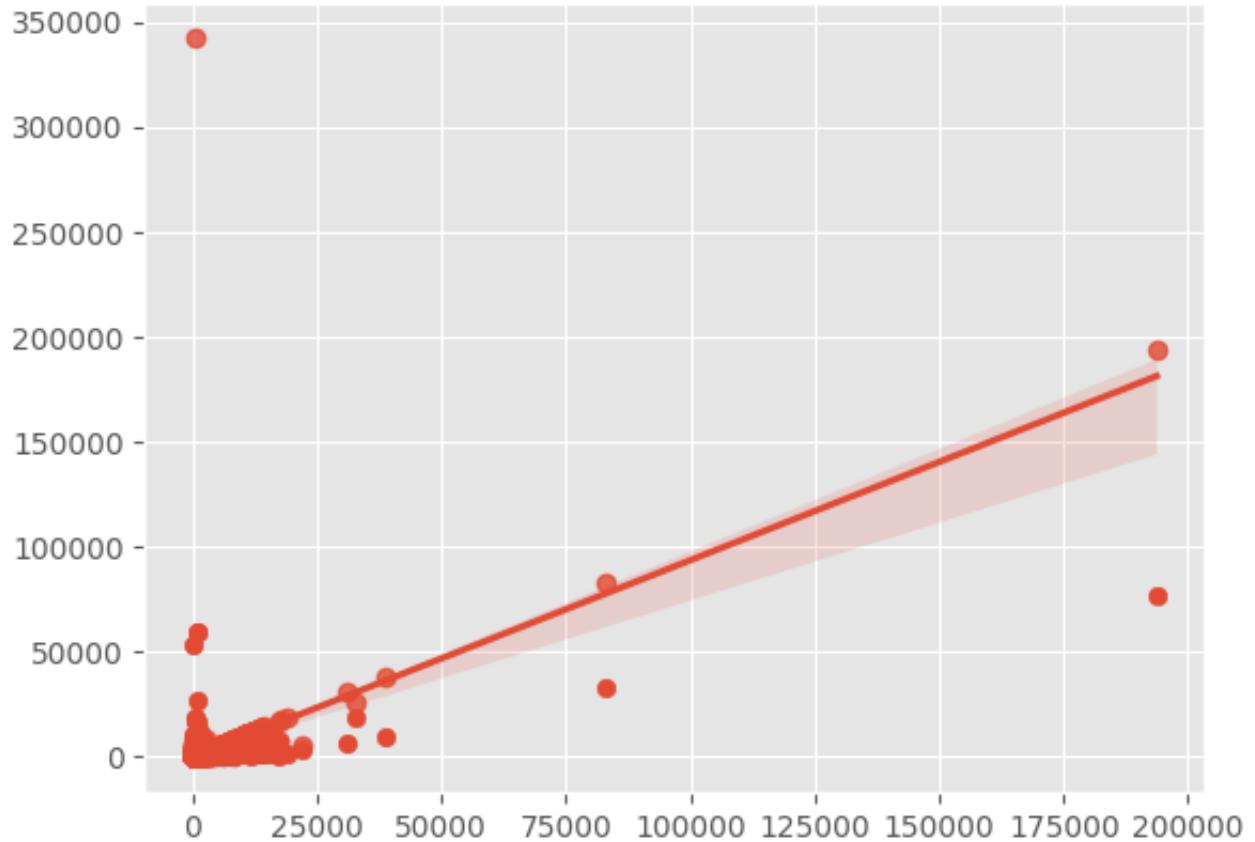
```
for i in range(0,10):
    knn = KNeighborsRegressor(n_neighbors = 5)
    knn.fit(X_trainval_2, y_trainval_2)
    y_pred_2 = knn.predict(X_test_2)

    print(' --- METRICHE ---')
    print(mean_absolute_error(y_test_2, y_pred_2))
    print(mean_squared_error(y_test_2, y_pred_2))
    print(median_absolute_error(y_test_2, y_pred_2))

    --- METRICHE ---
    356.7402966034125
    19304079.37340775
    72.0
```

```
plt.scatter(y_test_1, y_pred_1)
sns.regplot(x=y_test_2, y=y_pred_2, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



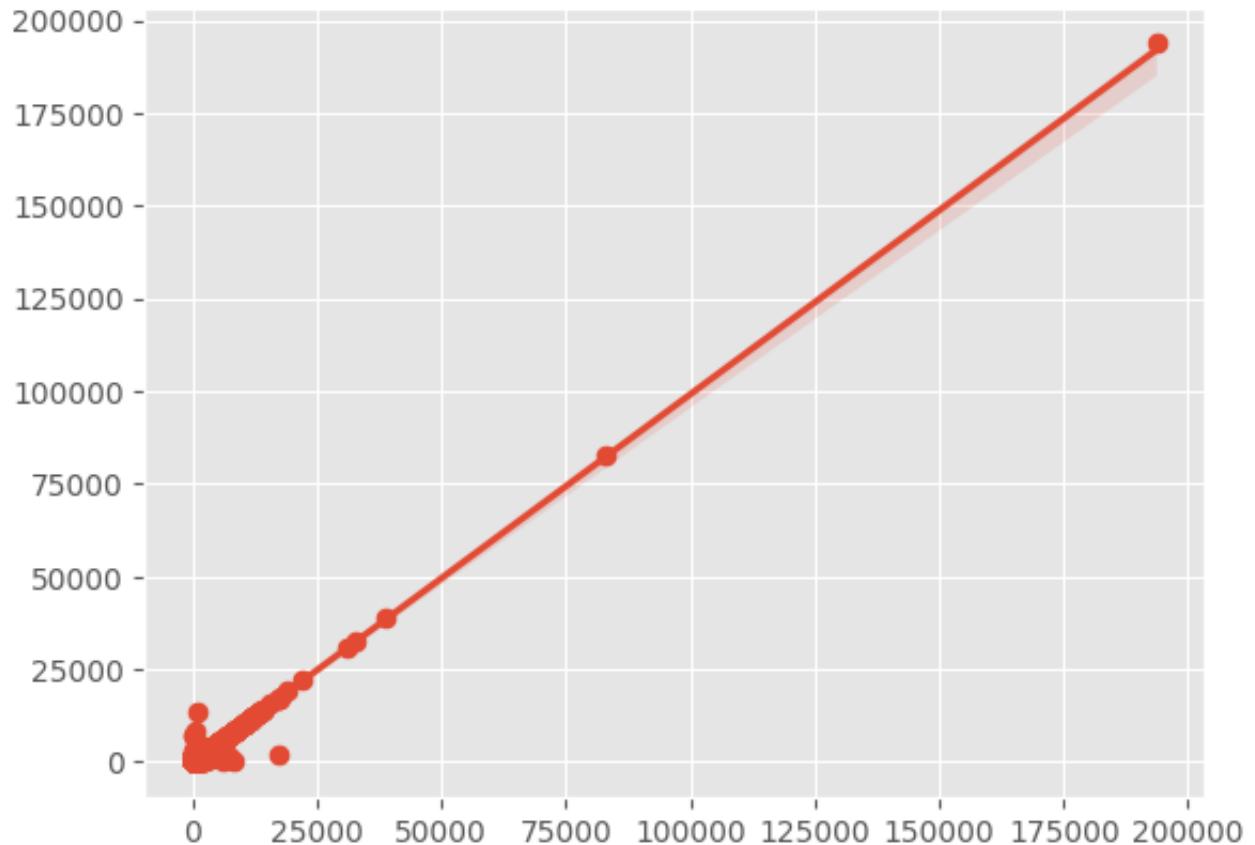
```
for i in range(0,10):
    knn = KNeighborsRegressor(n_neighbors = 1)
    knn.fit(X_trainval_2, y_trainval_2)
    y_pred_2 = knn.predict(X_test_2)

    print(' --- METRICHE ---')
    print(mean_absolute_error(y_test_2, y_pred_2))
    print(mean_squared_error(y_test_2, y_pred_2))
    print(median_absolute_error(y_test_2, y_pred_2))

    --- METRICHE ---
    42.38032211768458
    153828.64120554936
    0.0
```

```
plt.scatter(y_test_2, y_pred_2)
sns.regplot(x=y_test_2, y=y_pred_2, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



▼ Decision Tree

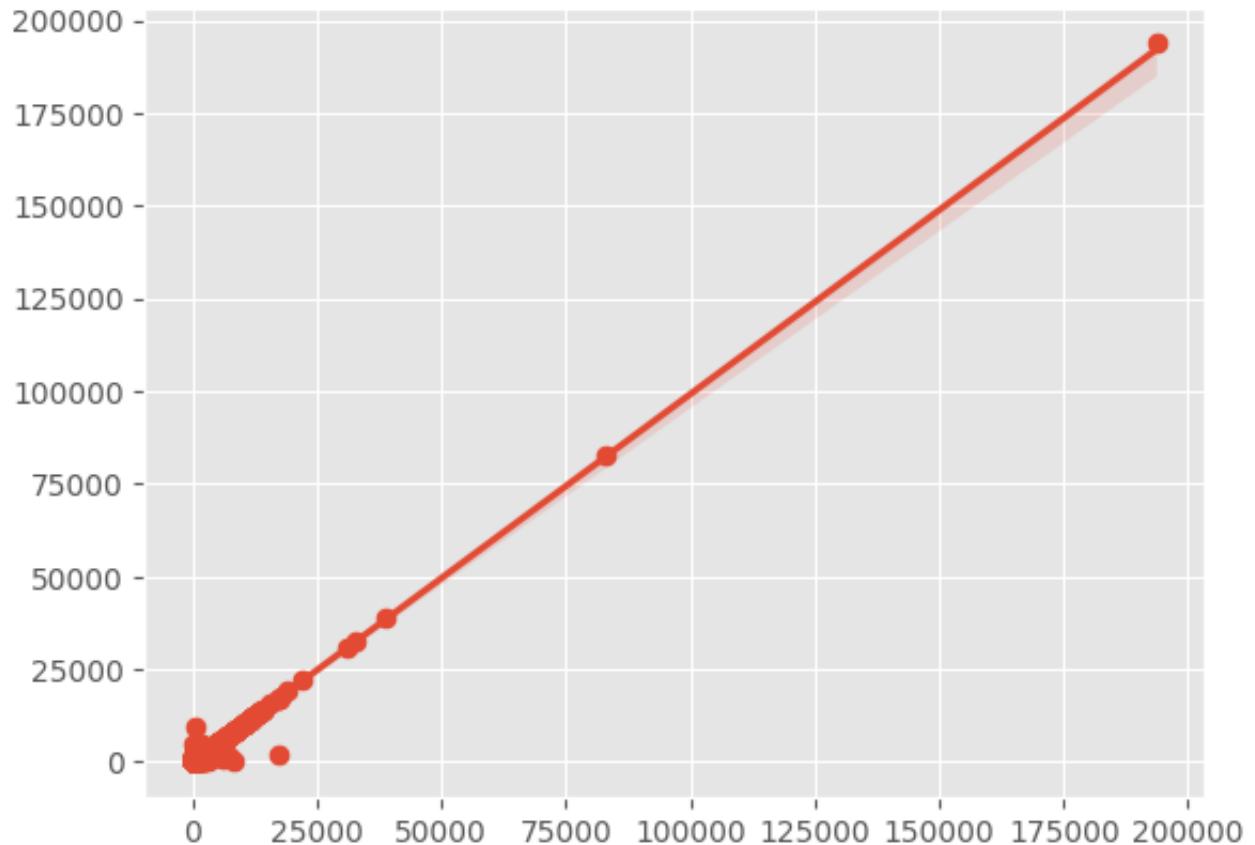
```
for i in range(0,10):
    dt = DecisionTreeRegressor()
    dt.fit(X_trainval_2, y_trainval_2)
    y_pred_2 = dt.predict(X_test_2)

    print(' --- METRICHE ---')
    print(mean_absolute_error(y_test_2, y_pred_2))
    print(mean_squared_error(y_test_2, y_pred_2))
    print(median_absolute_error(y_test_2, y_pred_2))

    --- METRICHE ---
    38.304895550948814
    116673.32482857599
    0.0
```

```
plt.scatter(y_test_2, y_pred_2)
sns.regplot(x=y_test_2, y=y_pred_2, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



▼ Random Forest

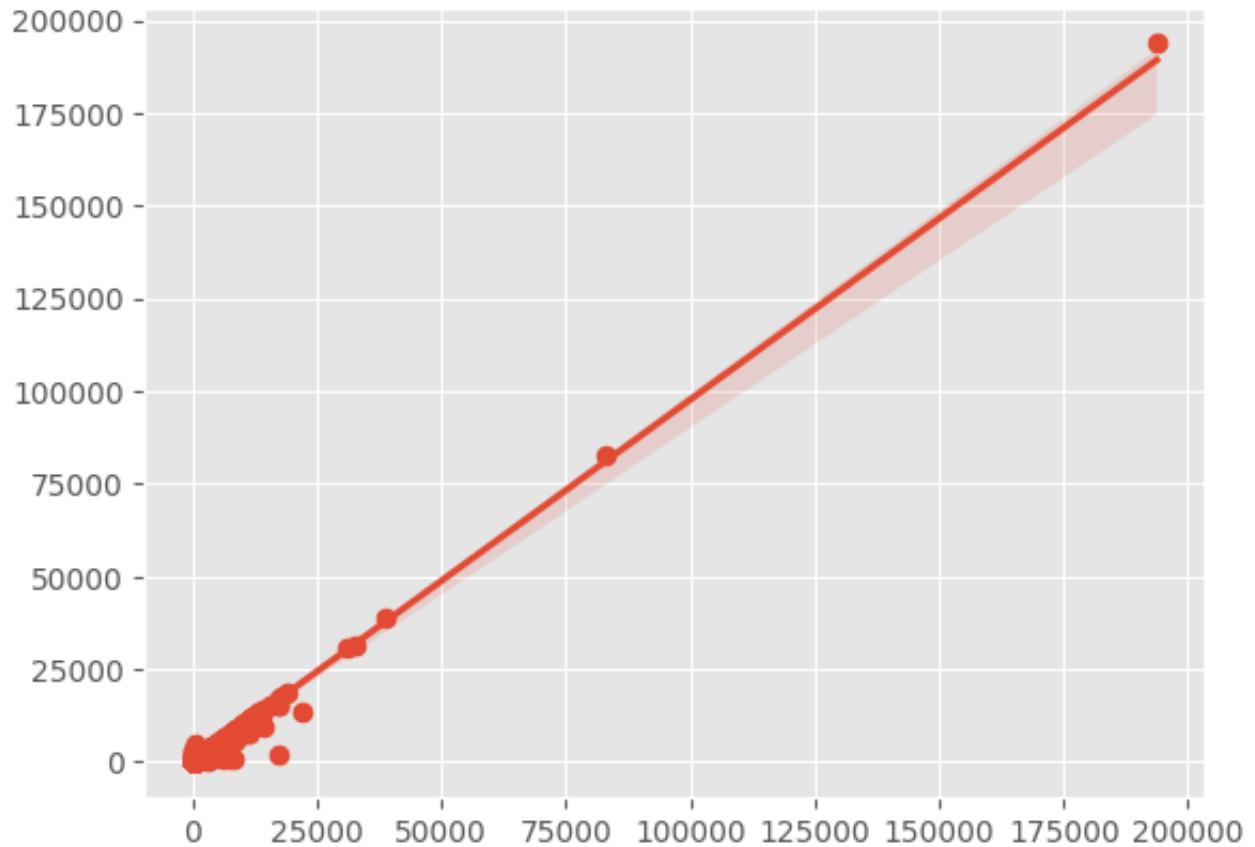
```
for i in range(0,10):
    rf = RandomForestRegressor()
    rf.fit(X_trainval_2, y_trainval_2)
    y_pred_2 = rf.predict(X_test_2)

    print(' --- METRICHE ---')
    print(mean_absolute_error(y_test_2, y_pred_2))
    print(mean_squared_error(y_test_2, y_pred_2))
    print(median_absolute_error(y_test_2, y_pred_2))

    --- METRICHE ---
    95.21830011162493
    138940.39884621272
    10.81999999999993
```

```
plt.scatter(y_test_2, y_pred_2)
sns.regplot(x=y_test_2, y=y_pred_2, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



▼ Picture_Data [3rd case]

▼ General

The third set of attributes is related to the uploaded profile picture data:

```
X_3 = df_LinkedIn_profile[['beauty', 'beauty_female', 'beauty_male', 'blur', 'emo',
                             'mouth_open', 'mouth_other', 'skin_acne', 'skin_dark_c
y_3 = df_LinkedIn_profile['n_followers'].to_numpy()
```

We normalize the data with the min max scaler function

```
scaler = MinMaxScaler()
scaler.fit(X_3)
X_3 = scaler.transform(X_3)
```

X_3

```
array([[0.52908995, 0.57329561, 0.52908995, ..., 0.01411   , 0.77587795,
       0.71323452],
       [0.52908995, 0.57329561, 0.52908995, ..., 0.01411   , 0.77587795,
       0.71323452],
       [0.52908995, 0.57329561, 0.52908995, ..., 0.01411   , 0.77587795,
       0.71323452],
       ...,
       [0.33682746, 0.46563196, 0.33682746, ..., 0.00995   , 0.95099708,
       0.86315334],
       [0.33682746, 0.46563196, 0.33682746, ..., 0.00995   , 0.95099708,
       0.86315334],
       [0.33682746, 0.46563196, 0.33682746, ..., 0.00995   , 0.95099708,
       0.86315334]])
```

y_3

```
array([420, 420, 420, ..., 279, 279, 279])
```

```
print('shape of X_3:')
print(X_3.shape)
print('type of X_3:')
print(type(X_3))
print('shape of y_3:')
print(y_3.shape)
print('type of y_3:')
print(type(y_3))
```

```
shape of X_3:
(62706, 23)
type of X_3:
<class 'numpy.ndarray'>
shape of y_3:
(62706,)
type of y_3:
<class 'numpy.ndarray'>
```

We test the train_test_split function in order to check if the set created are the same in each iteration:

```
X_trainval_3, X_test_3, y_trainval_3, y_test_3 = train_test_split(X_3, y_3, test_size=0.2, random_state=42)
X_train_3, X_val_3, y_train_3, y_val_3 = train_test_split(X_trainval_3, y_trainval_3, test_size=0.2, random_state=42)
```

```
print(len(X_train_3))
print(len(X_test_3))
print(len(X_val_3))
```

```
39504
6271
16931
```

X_train_3

```
array([[0.63638574, 0.59688467, 0.61896014, ..., 0.01151    , 0.9470328 ,
       0.93764364],
      [0.53318294, 0.57196487, 0.53318294, ..., 0.08878    , 0.76868017,
       0.4705349 ],
      [0.44581178, 0.45267478, 0.44581178, ..., 0.04431    , 0.42942378,
       0.85985368],
      ...,
      [0.65344662, 0.57556486, 0.65344662, ..., 0.02782    , 0.98048893,
       0.        ],
      [0.49326615, 0.56067461, 0.49326615, ..., 0.02005    , 0.9553718 ,
       0.86683253],
      [0.68486674, 0.70026195, 0.68486674, ..., 0.24699    , 0.9607376 ,
       0.92976871]])
```

X_test_3

```
array([[0.61687987, 0.60789477, 0.61687987, ..., 0.03191    , 0.4885176 ,
       0.        ],
      [0.55855138, 0.60138117, 0.55855138, ..., 0.11466    , 0.00770832,
       0.97139935],
      [0.40516554, 0.51802098, 0.40516554, ..., 0.14691    , 0.9753734 ,
       0.72015012],
      ...,
      [0.62033798, 0.63040524, 0.62033798, ..., 0.29348    , 0.14788972,
       0.94017373],
      [0.79073066, 0.78390228, 0.79073066, ..., 0.02297    , 0.60607456,
       0.58769951],
      [0.69903686, 0.75428988, 0.69903686, ..., 0.14025    , 0.06299804,
       0.88463809]])
```

```
X_val_3
```

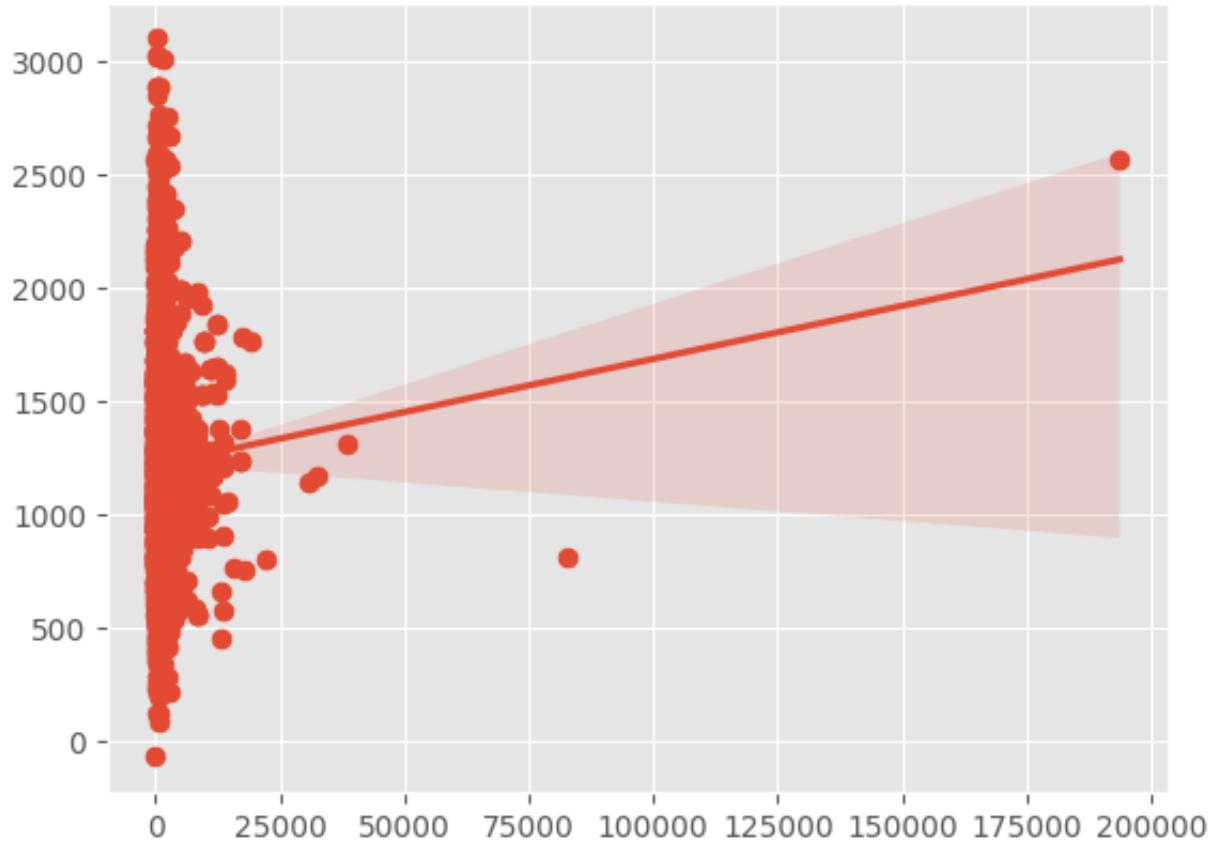
```
array([[0.61113888, 0.55078513, 0.61113888, ..., 0.05851 , 0.0745405 ,  
       0.06346328],  
      [0.46691162, 0.47022651, 0.46691162, ..., 0.04251 , 0.69453009,  
       0.          ],  
      [0.55405314, 0.51150738, 0.49615691, ..., 0.05216 , 0.82652264,  
       0.98513568],  
      ...,  
      [0.52205217, 0.50525991, 0.52205217, ..., 0.01958 , 0.41186481,  
       0.30317949],  
      [0.55617393, 0.54768942, 0.55617393, ..., 0.01115 , 0.69544108,  
       0.          ],  
      [0.67107485, 0.65828069, 0.67107485, ..., 0.07102 , 0.86816762,  
       0.95135887]])
```

▼ Liner Regression

```
for i in range(0,10):  
    lir = LinearRegression()  
    lir.fit(X_train_3, y_train_3)  
    y_pred_3 = lir.predict(X_test_3)  
  
print(' --- METRICHE ---')  
print(mean_absolute_error(y_test_3, y_pred_3))  
print(mean_squared_error(y_test_3, y_pred_3))  
print(median_absolute_error(y_test_3, y_pred_3))  
  
--- METRICHE ---  
937.0201855497633  
9832468.579783091  
635.8686214536428
```

```
plt.scatter(y_test_3, y_pred_3)
sns.regplot(x=y_test_3, y=y_pred_3, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



▼ Logistic Regression

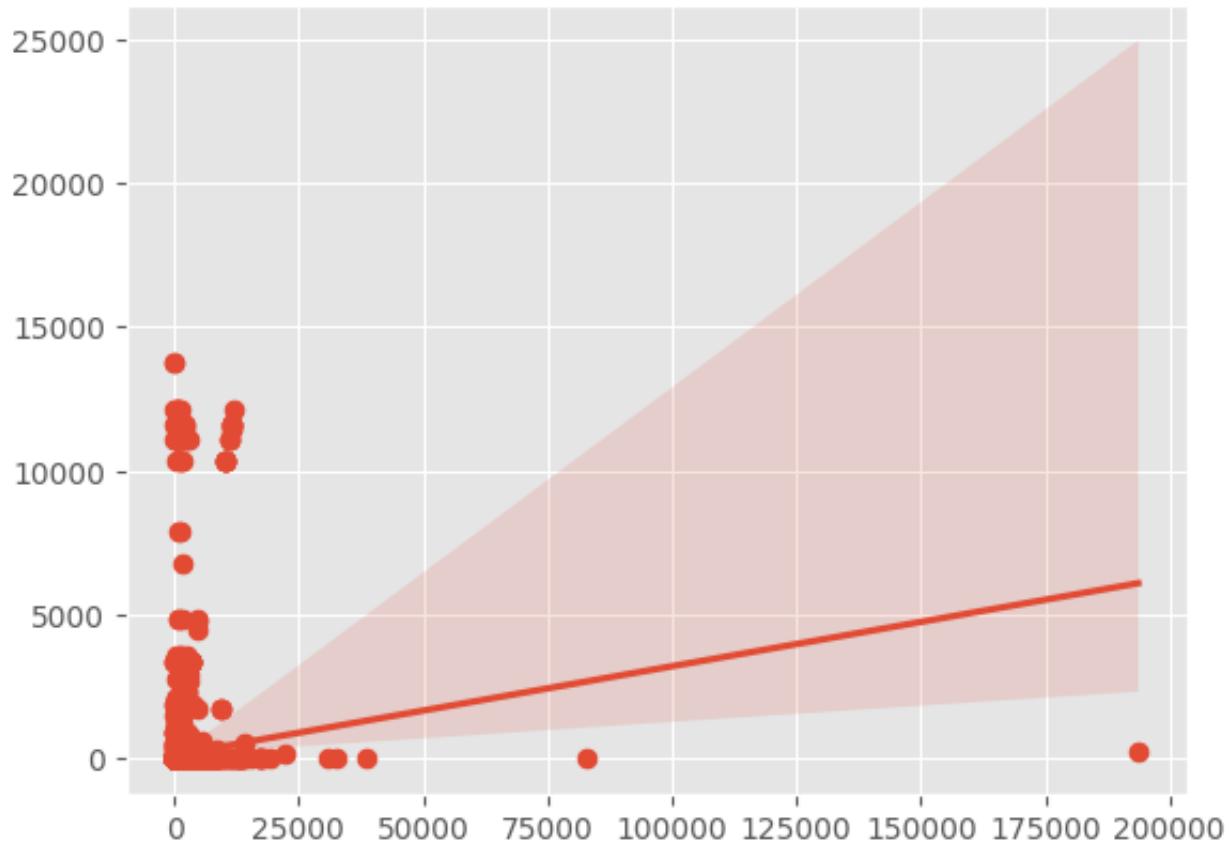
```
lor = LogisticRegression()
lor.fit(X_train_3, y_train_3)
y_pred_3 = lor.predict(X_test_3)

print(' --- METRICHE ---')
print(mean_absolute_error(y_test_3, y_pred_3))
print(mean_squared_error(y_test_3, y_pred_3))
print(median_absolute_error(y_test_3, y_pred_3))

--- METRICHE ---
1188.2509966512519
11541657.978950726
708.0
```

```
plt.scatter(y_test_3, y_pred_3)
sns.regplot(x=y_test_3, y=y_pred_3, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



▼ KNeighbors Regressor

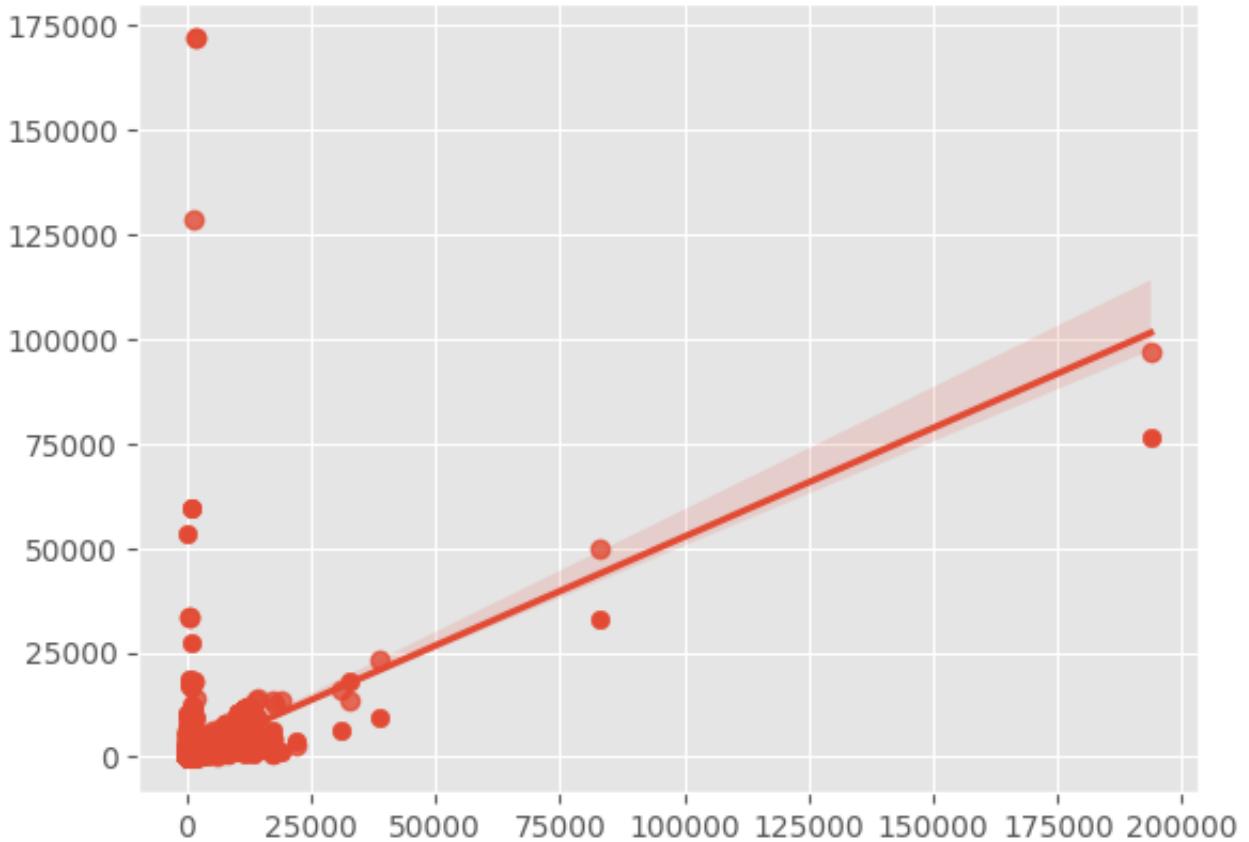
```
for i in range(0,10):
    knn = KNeighborsRegressor(n_neighbors = 10)
    knn.fit(X_trainval_3, y_trainval_3)
    y_pred_3 = knn.predict(X_test_3)

    print(' --- METRICHE ---')
    print(mean_absolute_error(y_test_3, y_pred_3))
    print(mean_squared_error(y_test_3, y_pred_3))
    print(median_absolute_error(y_test_3, y_pred_3))

    --- METRICHE ---
    658.8188167756339
    15256323.242050711
    274.7999999999995
```

```
plt.scatter(y_test_1, y_pred_1)
sns.regplot(x=y_test_3, y=y_pred_3, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



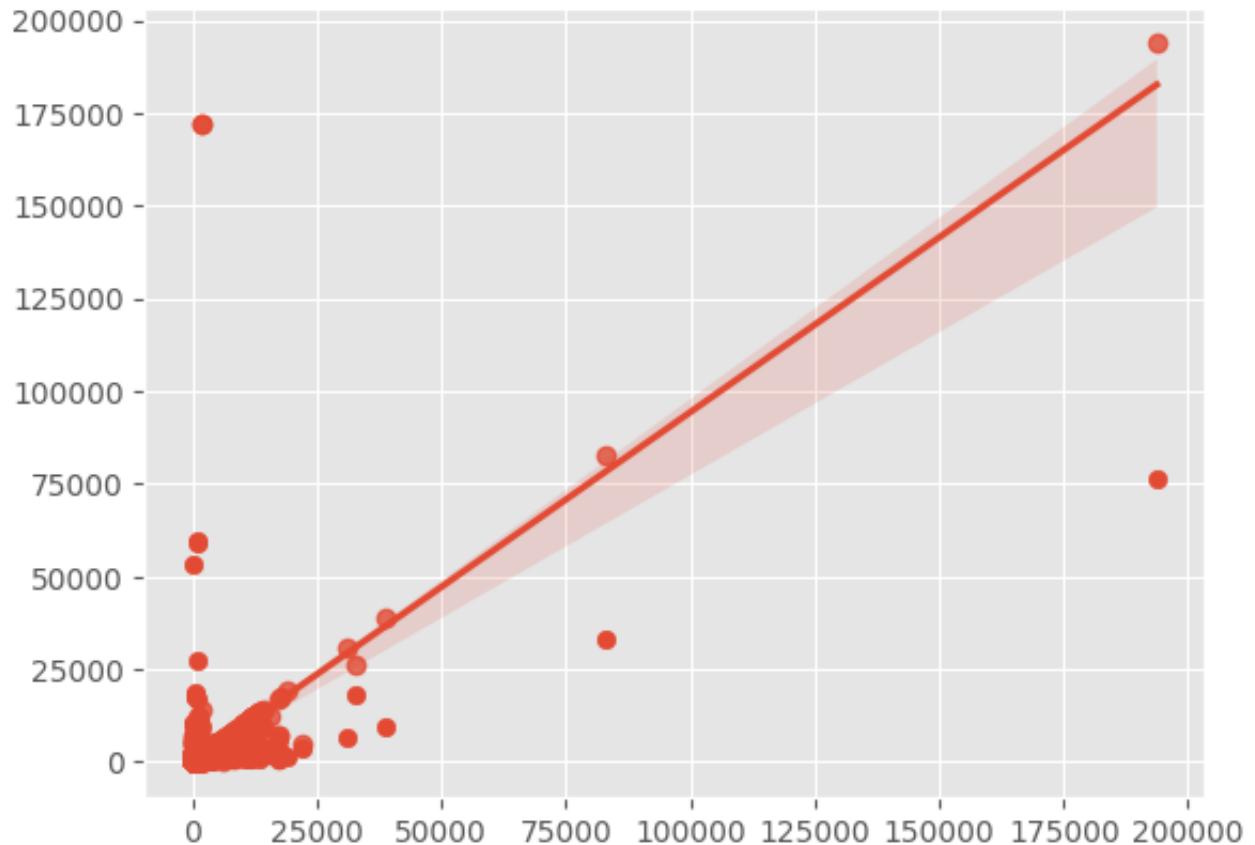
```
for i in range(0,10):
    knn = KNeighborsRegressor(n_neighbors = 5)
    knn.fit(X_trainval_3, y_trainval_3)
    y_pred_3 = knn.predict(X_test_3)

print(' --- METRICHE ---')
print(mean_absolute_error(y_test_3, y_pred_3))
print(mean_squared_error(y_test_3, y_pred_3))
print(median_absolute_error(y_test_3, y_pred_3))

--- METRICHE ---
361.2128528145431
9933962.30613937
69.79999999999995
```

```
plt.scatter(y_test_1, y_pred_1)
sns.regplot(x=y_test_3, y=y_pred_3, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



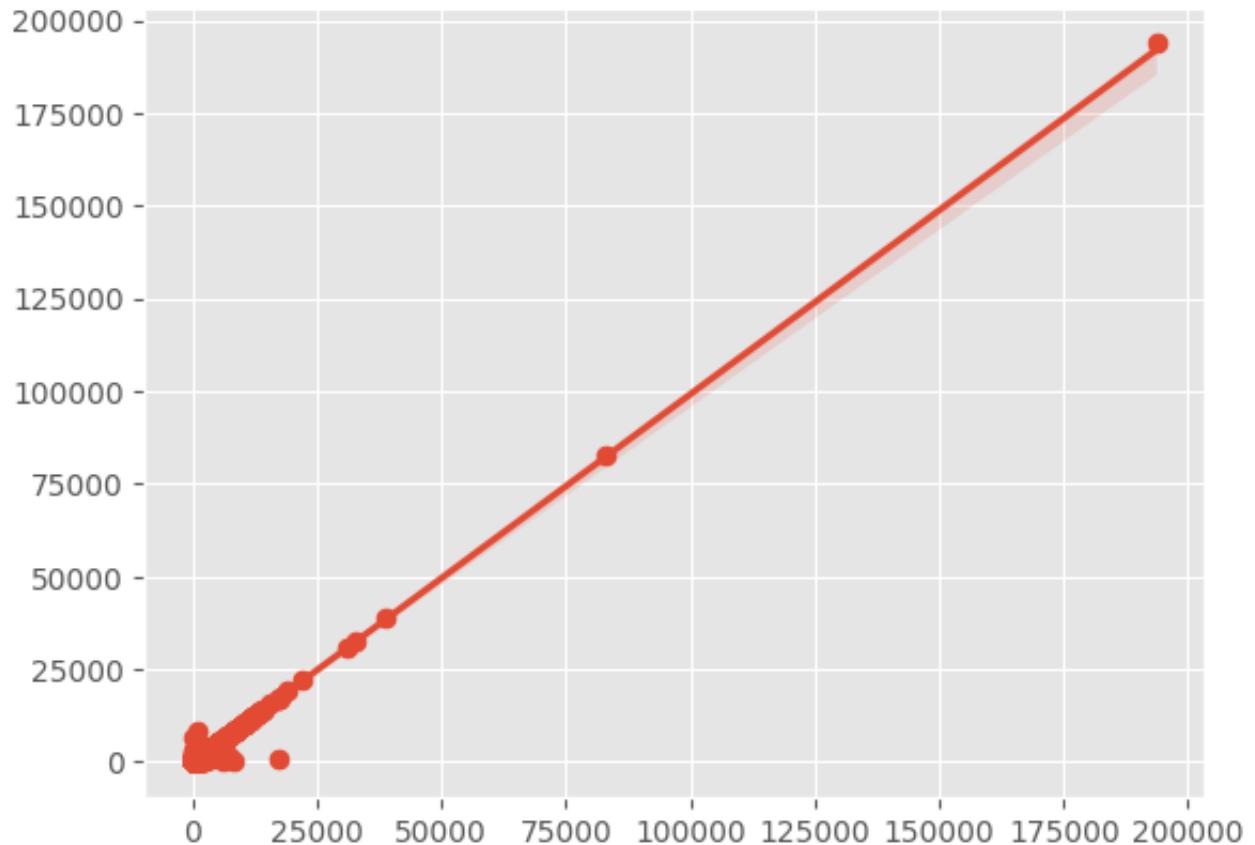
```
for i in range(0,10):
    knn = KNeighborsRegressor(n_neighbors = 1)
    knn.fit(X_trainval_3, y_trainval_3)
    y_pred_3 = knn.predict(X_test_3)

print(' --- METRICHE ---')
print(mean_absolute_error(y_test_3, y_pred_3))
print(mean_squared_error(y_test_3, y_pred_3))
print(median_absolute_error(y_test_3, y_pred_3))
```

```
--- METRICHE ---
38.80066974964121
114640.82171902408
0.0
```

```
plt.scatter(y_test_3, y_pred_3)
sns.regplot(x=y_test_3, y=y_pred_3, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



▼ Decision Tree

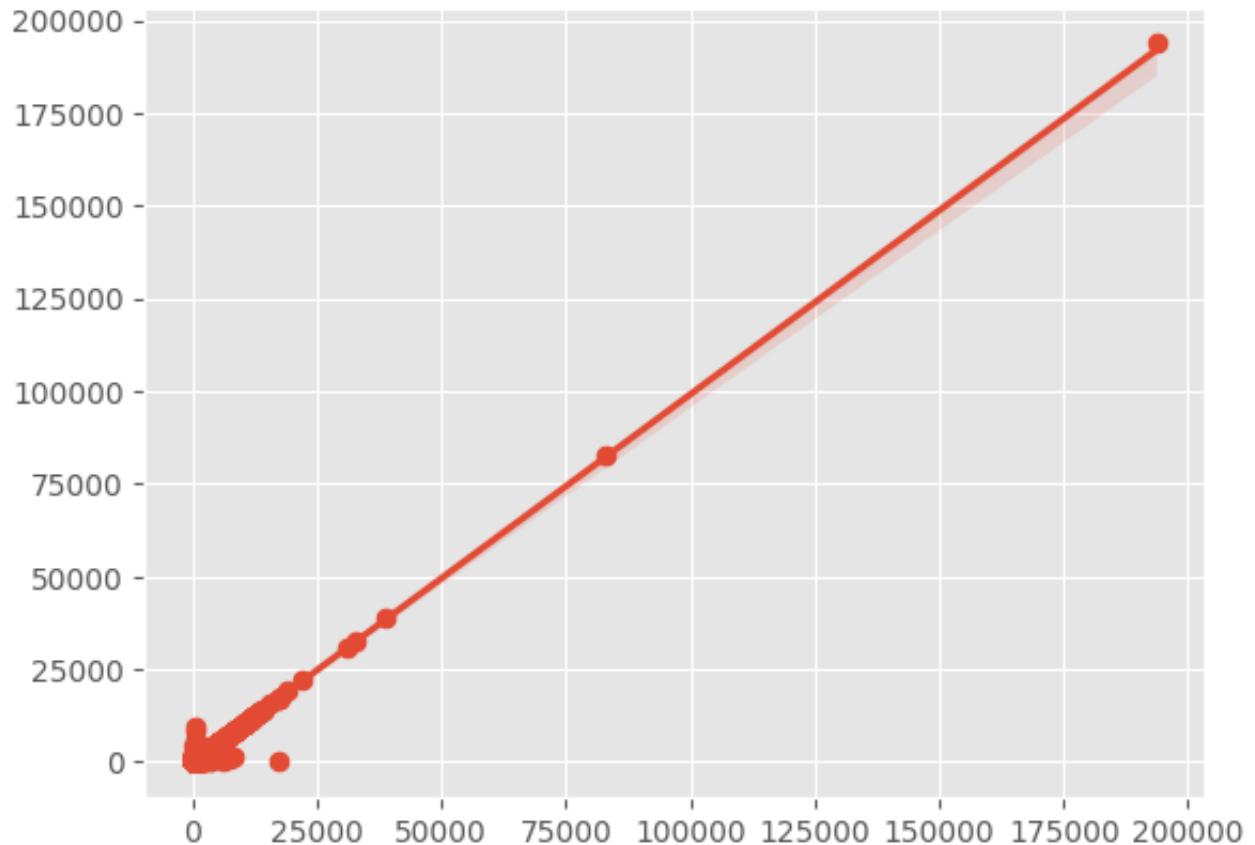
```
for i in range(0,10):
    dt = DecisionTreeRegressor()
    dt.fit(X_trainval_3, y_trainval_3)
    y_pred_3 = dt.predict(X_test_3)

    print(' --- METRICHE ---')
    print(mean_absolute_error(y_test_3, y_pred_3))
    print(mean_squared_error(y_test_3, y_pred_3))
    print(median_absolute_error(y_test_3, y_pred_3))

    --- METRICHE ---
    43.2669430712805
    140975.86748524956
    0.0
```

```
plt.scatter(y_test_3, y_pred_3)
sns.regplot(x=y_test_3, y=y_pred_3, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



▼ Random Forest

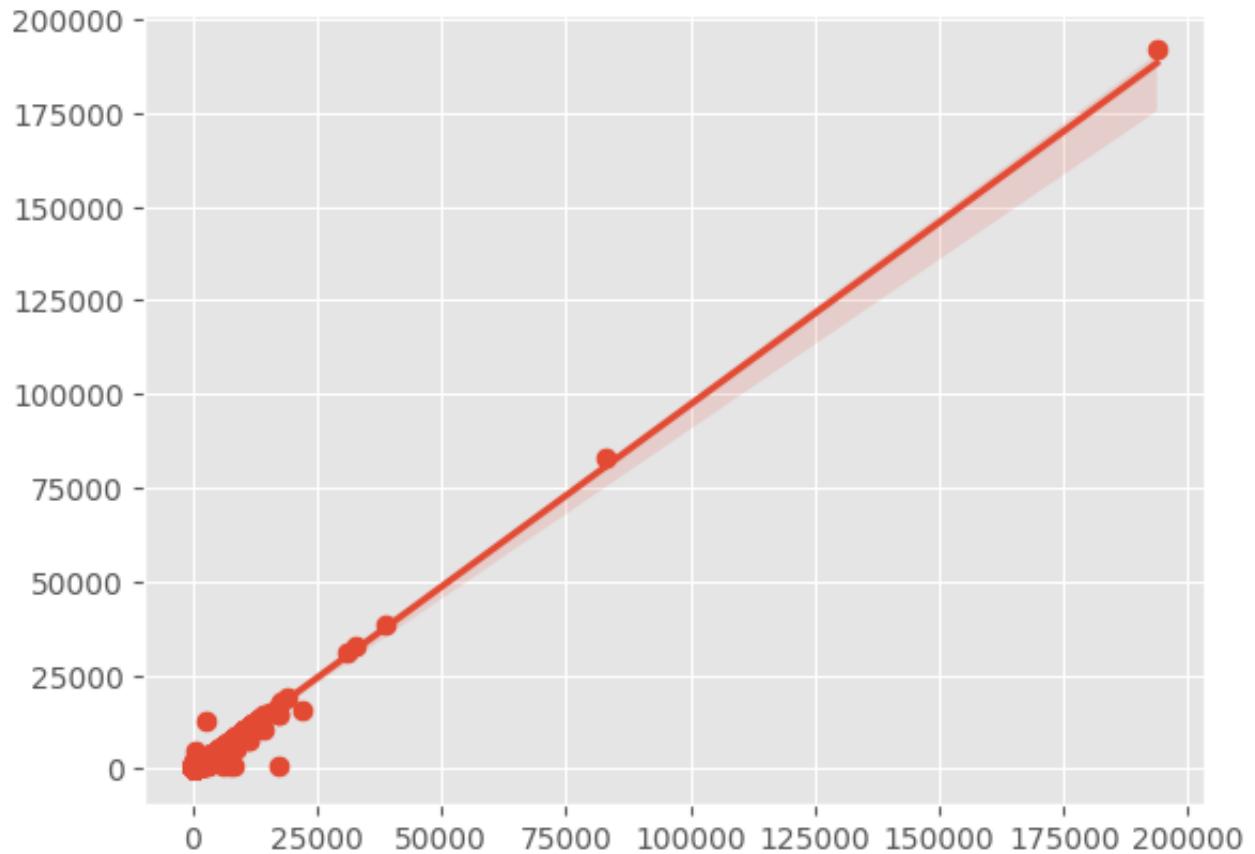
```
for i in range(0,10):
    rf = RandomForestRegressor()
    rf.fit(X_trainval_3, y_trainval_3)
    y_pred_3 = rf.predict(X_test_3)

    print(' --- METRICHE ---')
    print(mean_absolute_error(y_test_3, y_pred_3))
    print(mean_squared_error(y_test_3, y_pred_3))
    print(median_absolute_error(y_test_3, y_pred_3))

    --- METRICHE ---
    100.04158029022484
    151001.0199054537
    11.22000000000027
```

```
plt.scatter(y_test_3, y_pred_3)
sns.regplot(x=y_test_3, y=y_pred_3, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



▼ General_Data [4th case]

▼ General

Attributes X and y are created, the second is the target attribute while the second is made up of all the other features:

```
X_4 = df_LinkedIn_profile[['c_id', 'avg_time_in_previous_position', 'avg_current
                           'current_position_length', 'age', 'beauty', 'beauty_fem
                           'gender', 'head_pitch', 'head_roll', 'head_yaw', 'mouth
                           'skin_stain', 'smile', 'african', 'celtic_english', 'ea
y_4 = df_LinkedIn_profile['n_followers'].to_numpy()
```

We normalize the data with the min max scaler function

```
scaler = MinMaxScaler()
scaler.fit(X_4)
X_4 = scaler.transform(X_4)
```

X_4

```
array([[0.00000000e+00, 7.14285714e-02, 2.62225050e-02, ...,
       0.00000000e+00, 0.00000000e+00, 7.13234519e-01],
      [1.59476916e-05, 3.57142857e-02, 1.50881658e-02, ...,
       0.00000000e+00, 0.00000000e+00, 7.13234519e-01],
      [3.18953831e-05, 2.38095238e-02, 1.64970005e-02, ...,
       0.00000000e+00, 0.00000000e+00, 7.13234519e-01],
      ...,
      [9.99968105e-01, 0.00000000e+00, 1.10434466e-02, ...,
       0.00000000e+00, 0.00000000e+00, 8.63153345e-01],
      [9.99984052e-01, 0.00000000e+00, 1.03662970e-01, ...,
       0.00000000e+00, 0.00000000e+00, 8.63153345e-01],
      [1.00000000e+00, 0.00000000e+00, 3.32212325e-02, ...,
       0.00000000e+00, 0.00000000e+00, 8.63153345e-01]])
```

y_4

```
array([420, 420, 420, ..., 279, 279, 279])
```

```
print('shape of X_4: ')
print(X_4.shape)
print('type of X_4: ')
print(type(X_4))
print('shape of y_4: ')
print(y_4.shape)
print('type of y_4: ')
print(type(y_4))
```

```
shape of X_4:
(62706, 41)
type of X_4:
<class 'numpy.ndarray'>
shape of y_4:
(62706,)
type of y_4:
<class 'numpy.ndarray'>
```

We test the train_test_split function in order to check if the set created are the same in each iteration:

```
X_trainval_4, X_test_4, y_trainval_4, y_test_4 = train_test_split(X_4, y_4, test_size=0.2)
X_train_4, X_val_4, y_train_4, y_val_4 = train_test_split(X_trainval_4, y_trainval_4, test_size=0.2)
```

```
print(len(X_train_4))
print(len(X_test_4))
print(len(X_val_4))
```

```
39504
6271
16931
```

X_train_4

```
array([[0.64399968, 0.01428571, 0.00822578, ..., 0.         , 0.04160416,
       0.93764364],
      [0.09327805, 0.         , 0.06080713, ..., 0.         , 0.         ,
       0.4705349 ],
      [0.01840364, 0.         , 0.07044174, ..., 0.         , 0.         ,
       0.85985368],
      ...,
      [0.55056216, 0.         , 0.07044174, ..., 0.         , 0.         ,
       0.         ],
      [0.75006778, 0.         , 0.04331031, ..., 0.         , 0.         ,
       0.86683253],
      [0.89275177, 0.07142857, 0.03722051, ..., 0.         , 0.         ,
       0.92976871]])
```

X_test_4

```
array([[0.82754166, 0.03571429, 0.23513907, ..., 0.         , 0.         ,
       0.         ],
      [0.84385615, 0.         , 0.03632673, ..., 0.         , 0.98629863,
       0.97139935],
      [0.90453712, 0.         , 0.11338847, ..., 0.         , 0.         ,
       0.72015012],
      ...,
      [0.05819313, 0.         , 0.00686239, ..., 0.         , 0.         ,
       0.94017373],
      [0.6687186 , 0.         , 0.08155335, ..., 0.         , 0.01370137,
       0.58769951],
      [0.50474444, 0.01785714, 0.02758589, ..., 0.         , 0.         ,
       0.88463809]])
```

X_val_4

```
array([[0.74558648, 0.03571429, 0.02613161, ..., 0.         , 0.99949995,
       0.06346328],
      [0.50007176, 0.         , 0.05385384, ..., 0.         , 0.99389939,
       0.         ],
      [0.19649151, 0.         , 0.08843847, ..., 0.         , 0.03830383,
       0.98513568],
      ...,
      [0.94394386, 0.         , 0.01508817, ..., 0.         , 0.         ,
       0.30317949],
      [0.74911092, 0.07142857, 0.027495 , ..., 0.         , 0.         ,
       0.         ],
      [0.01811658, 0.         , 0.0373114 , ..., 0.         , 0.         ,
       0.95135887]])
```

▼ Liner Regression

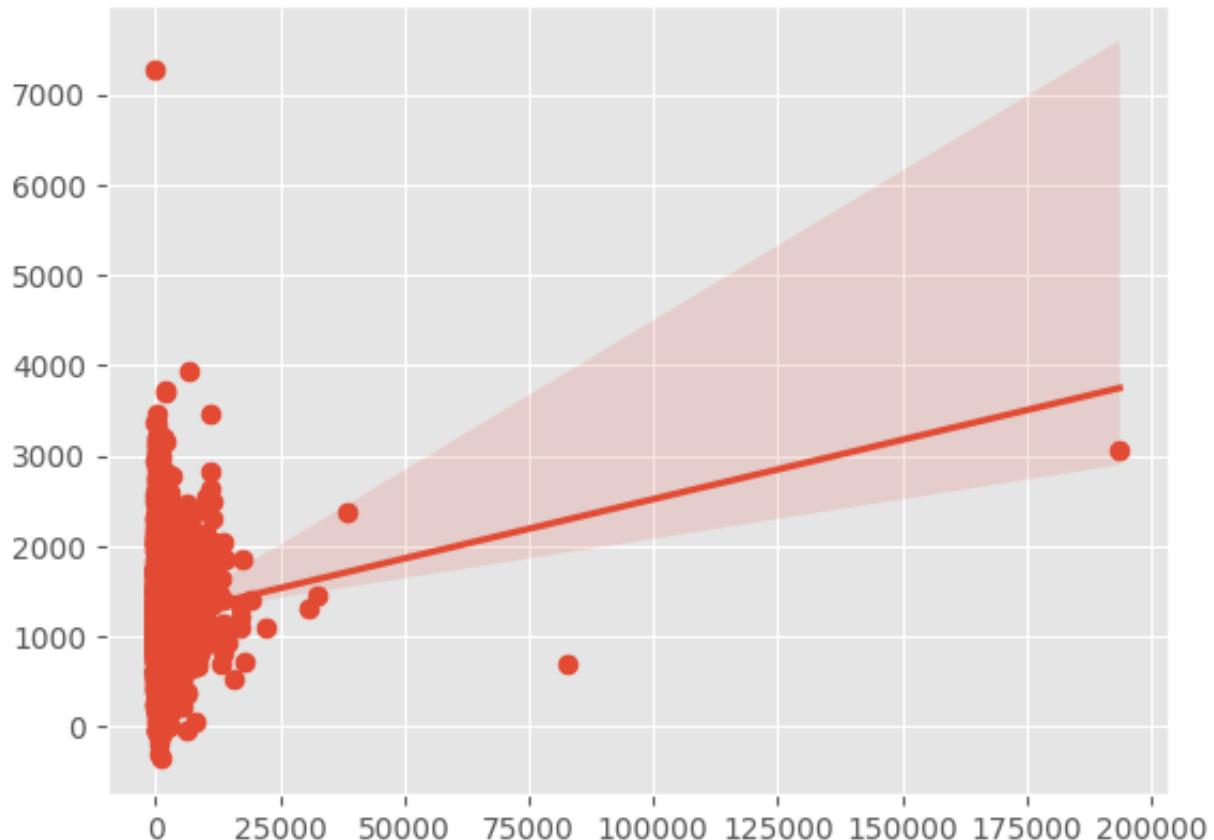
```
for i in range(0,10):
    lir = LinearRegression()
    lir.fit(X_train_4, y_train_4)
    y_pred_4 = lir.predict(X_test_4)

print(' --- METRICHE ---')
print(mean_absolute_error(y_test_4, y_pred_4))
print(mean_squared_error(y_test_4, y_pred_4))
print(median_absolute_error(y_test_4, y_pred_4))
```

```
--- METRICHE ---
946.3475252238491
9786537.191111144
623.5835855714977
```

```
plt.scatter(y_test_4, y_pred_4)
sns.regplot(x=y_test_4, y=y_pred_4, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



▼ Logistic Regression

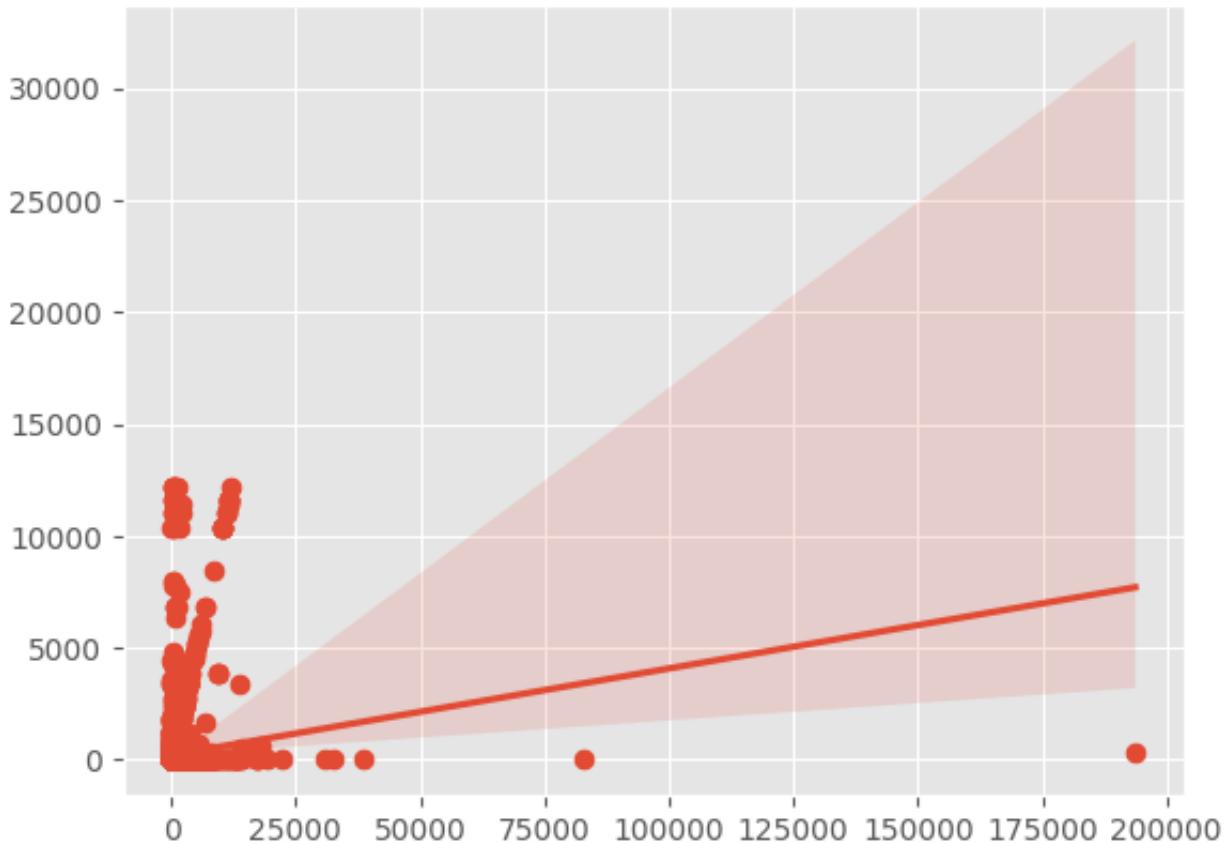
```
lor = LogisticRegression()
lor.fit(X_train_4, y_train_4)
y_pred_4 = lor.predict(X_test_4)

print(' --- METRICHE ---')
print(mean_absolute_error(y_test_4, y_pred_4))
print(mean_squared_error(y_test_4, y_pred_4))
print(median_absolute_error(y_test_4, y_pred_4))
```

```
--- METRICHE ---
1118.5389889969701
11131661.091691915
674.0
```

```
plt.scatter(y_test_4, y_pred_4)
sns.regplot(x=y_test_4, y=y_pred_4, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



▼ KNeighbors Regressor

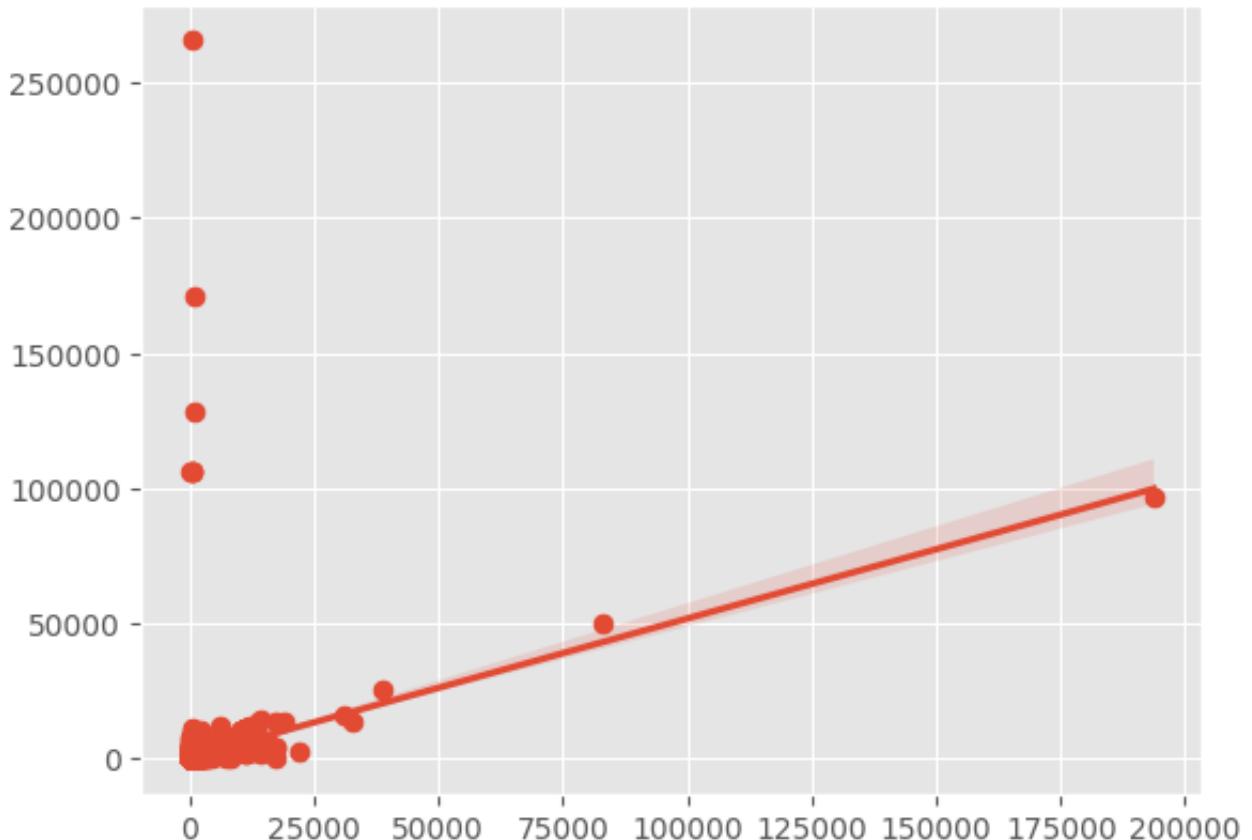
```
for i in range(0,10):
    knn = KNeighborsRegressor(n_neighbors = 10)
    knn.fit(X_trainval_4, y_trainval_4)
    y_pred_4 = knn.predict(X_test_4)

print(' --- METRICHE ---')
print(mean_absolute_error(y_test_4, y_pred_4))
print(mean_squared_error(y_test_4, y_pred_4))
print(median_absolute_error(y_test_4, y_pred_4))
```

```
--- METRICHE ---
717.0657311433582
28618093.221001428
271.2000000000005
```

```
plt.scatter(y_test_4, y_pred_4)
sns.regplot(x=y_test_4, y=y_pred_4, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



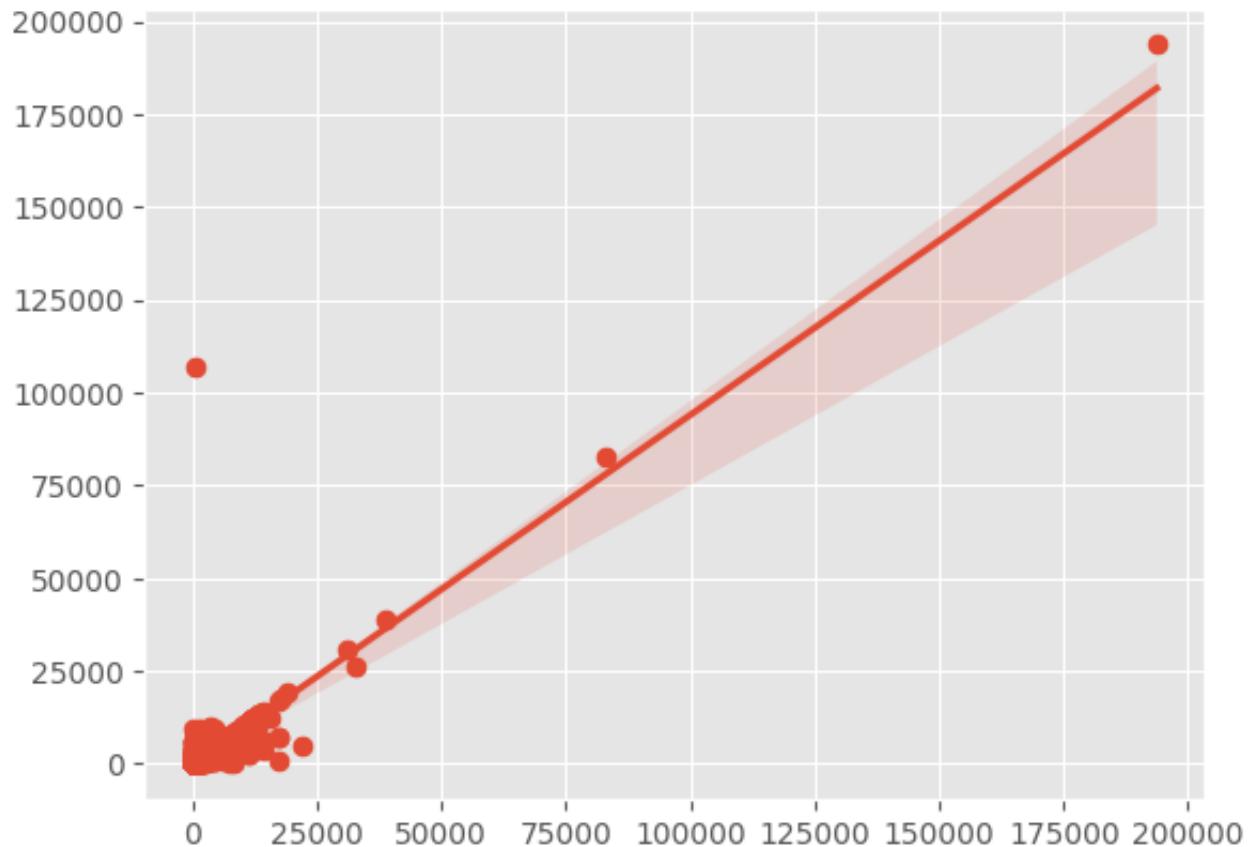
```
for i in range(0,10):
    knn = KNeighborsRegressor(n_neighbors = 5)
    knn.fit(X_trainval_4, y_trainval_4)
    y_pred_4 = knn.predict(X_test_4)

    print(' --- METRICHE ---')
    print(mean_absolute_error(y_test_4, y_pred_4))
    print(mean_squared_error(y_test_4, y_pred_4))
    print(median_absolute_error(y_test_4, y_pred_4))
```

```
--- METRICHE ---
321.30747887099346
2451832.0220060595
77.60000000000002
```

```
plt.scatter(y_test_4, y_pred_4)
sns.regplot(x=y_test_4, y=y_pred_4, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



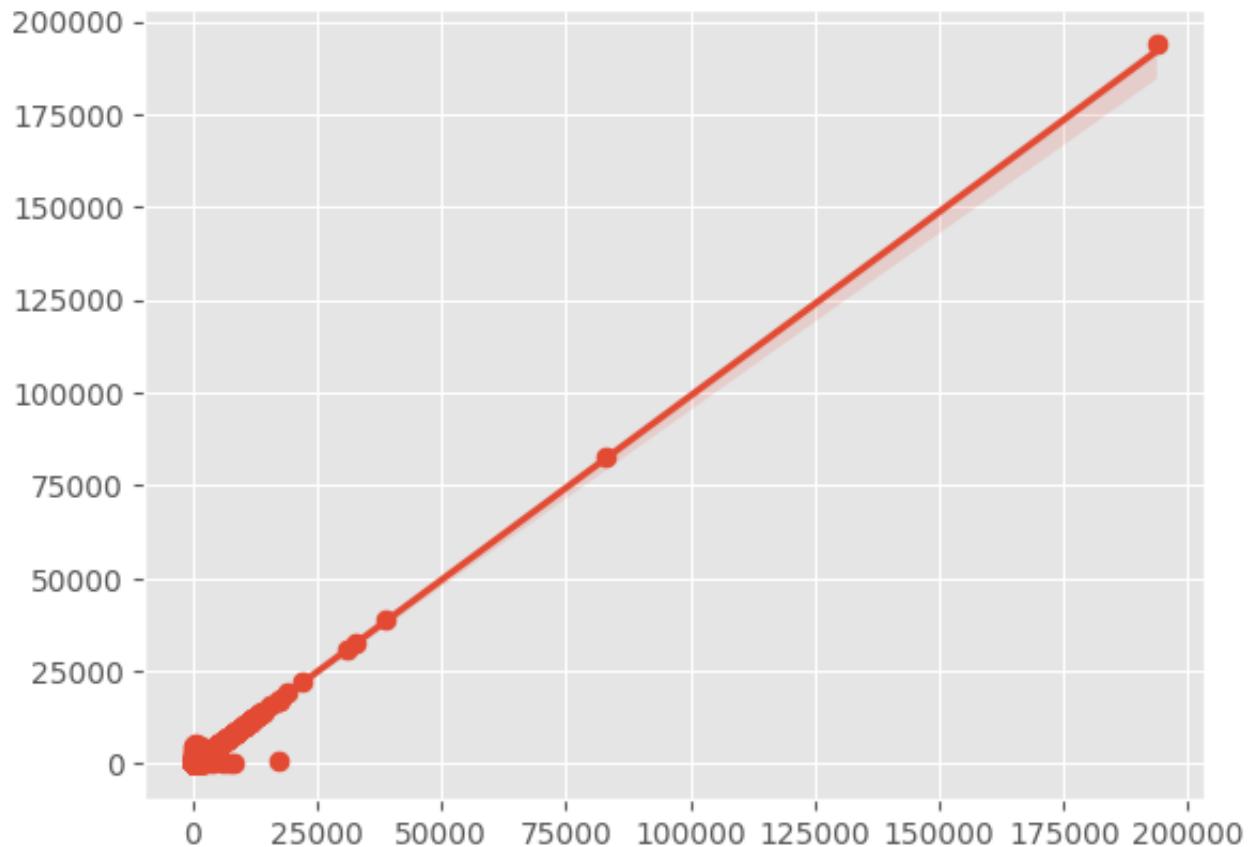
```
for i in range(0,10):
    knn = KNeighborsRegressor(n_neighbors = 1)
    knn.fit(X_trainval_4, y_trainval_4)
    y_pred_4 = knn.predict(X_test_4)

    print(' --- METRICHE ---')
    print(mean_absolute_error(y_test_4, y_pred_4))
    print(mean_squared_error(y_test_4, y_pred_4))
    print(median_absolute_error(y_test_4, y_pred_4))
```

```
--- METRICHE ---
46.40041460692075
131633.28735448892
0.0
```

```
plt.scatter(y_test_4, y_pred_4)
sns.regplot(x=y_test_4, y=y_pred_4, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



▼ Decision Tree

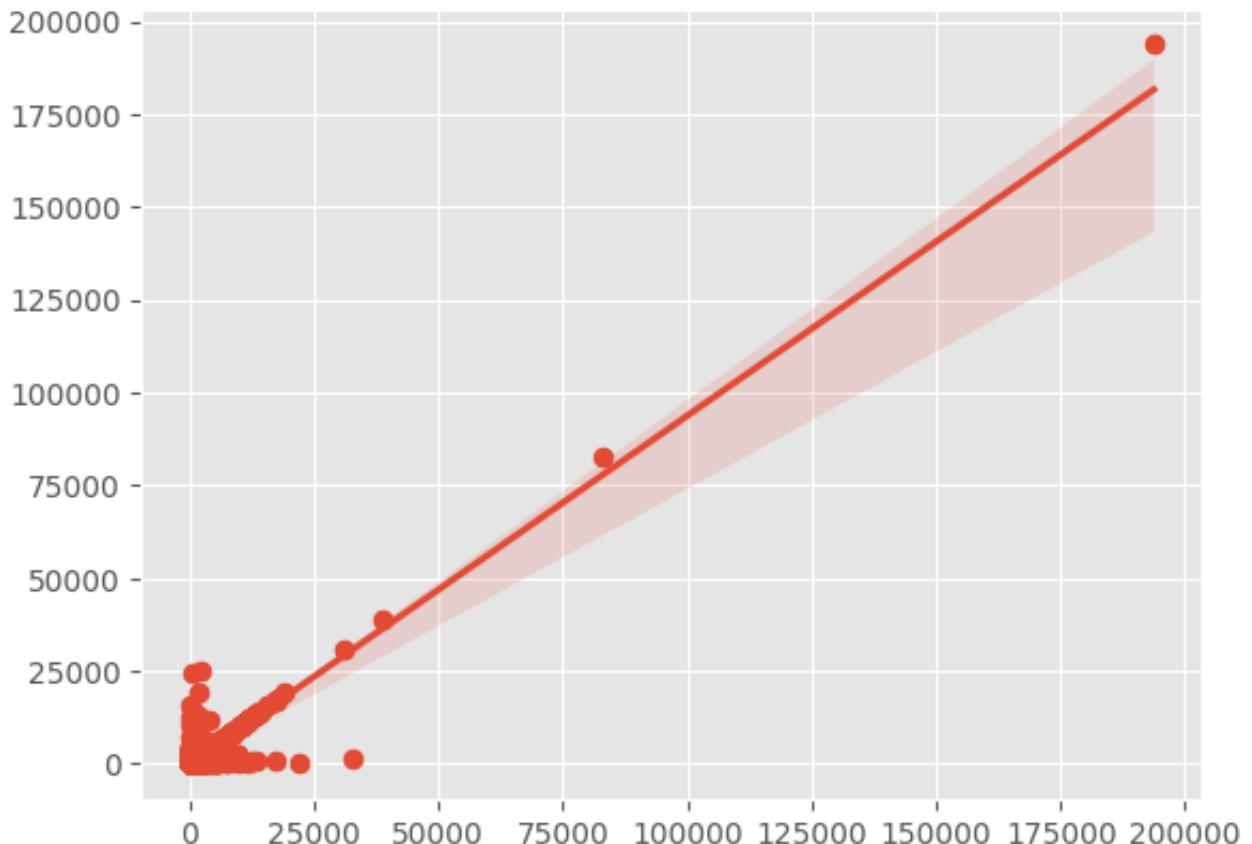
```
for i in range(0,10):
    dt = DecisionTreeRegressor()
    dt.fit(X_trainval_4, y_trainval_4)
    y_pred_4 = dt.predict(X_test_4)

print(' --- METRICHE ---')
print(mean_absolute_error(y_test_4, y_pred_4))
print(mean_squared_error(y_test_4, y_pred_4))
print(median_absolute_error(y_test_4, y_pred_4))
```

```
--- METRICHE ---
280.13251475043853
1361570.1455908148
0.0
```

```
plt.scatter(y_test_4, y_pred_4)
sns.regplot(x=y_test_4, y=y_pred_4, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



▼ Random Forest

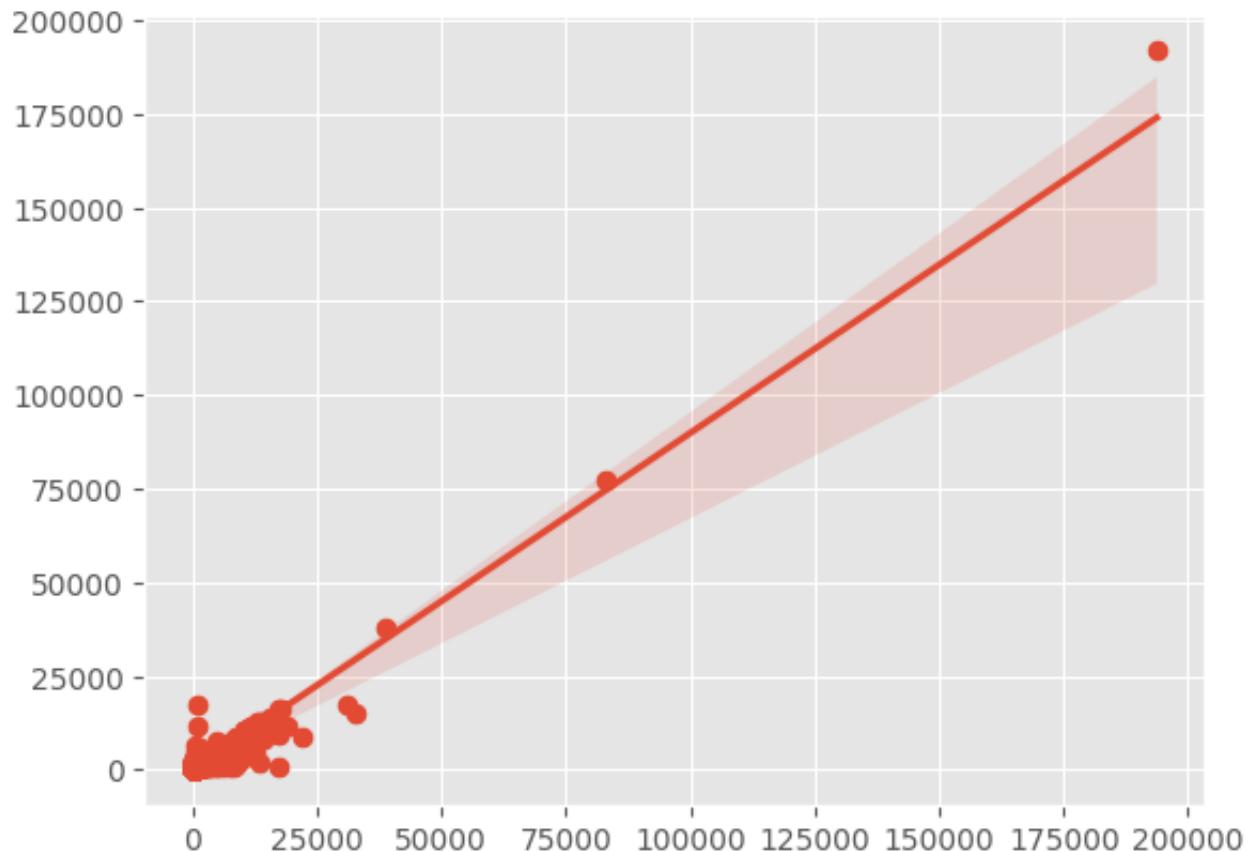
```
for i in range(0,10):
    rf = RandomForestRegressor()
    rf.fit(X_trainval_4, y_trainval_4)
    y_pred_4 = rf.predict(X_test_4)

print(' --- METRICHE ---')
print(mean_absolute_error(y_test_4, y_pred_4))
print(mean_squared_error(y_test_4, y_pred_4))
print(median_absolute_error(y_test_4, y_pred_4))
```

```
--- METRICHE ---
317.6874677084994
576096.4991705151
163.5
```

```
plt.scatter(y_test_4, y_pred_4)
sns.regplot(x=y_test_4, y=y_pred_4, data=df_LinkedIn_clean)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Prodotti Colab a pagamento - Annulla i contratti qui

✓ 1 s data/ora di completamento: 13:21

