Regression Model

The regression model is a supervise learning. this model used for prediction for continuous value based on correlation features.

Install xgboost

```
!pip install xgboost
```

```
Libraries used in this project:
- Python 3.12.6 (tags/v3.12.6:a4a2d2b, Sep  6 2024, 20:11:23) [MSC v.1940 64 bit (AMD64)]
- NumPy 2.1.1
- pandas 2.2.3
- Matplotlib 3.9.2
- Seaborn 0.13.2
- scikit-learn 1.6.1
- XGBoost 3.0.1
```

- Download and install xgboost library.
- xgboost library used for model training.

Load and preview the data

```
users_data = pd.read_pickle('users_data_final.pickle')

users_data.head(n = 5)
```

	user_id	# number_transactions	# total_amount_usd	# job_management	# jol
0	9231c446-cb16-4b2b-a7f7-ddfc8b25	3.0	2143.0	1	
1	bb92765a-08de-4963-b432-496524b	0.0	1369.42	0	
2	573de577-49ef-42b9-83da-d3cfb817	2.0	2.0	0	
3	d6b66b9d-7c8f-4257-a682-e136f64(	0.0	1369.42	0	
4	fade0b20-7594-4d9a-84cd-c02f79b1	1.0	1.0	0	

5 rows x 33 cols 10 per page Page 1 of 1

```
#Check the shape of the data
users_data.shape
```

```
(45179, 33)
```

```
#Check the data types
users_data.dtypes
```

- Load data from pickle file 'users_data_final.pickle'
- Display data with 5 rows only

```
user_id                    object
number_transactions       float64
total_amount_usd          float64
job_management              int64
job_technician              int64
job_entrepreneur            int64
job_blue-collar             int64
job_retired                 int64
job_admin.                  int64
job_services                int64
job_self-employed           int64
job_unemployed              int64
job_housemaid               int64
job_student                 int64
education_tertiary          int64
education_secondary         int64
education_Unknown           int64
education_primary           int64
default                      bool
housing                      bool
loan                         bool
contact_cellular            int64
contact_telephone           int64
duration                    int64
campaign                    int64
...
device_tablet               int64
single                      uint8
age_group_encoded            int8
month_joined                int64
dtype: object
```

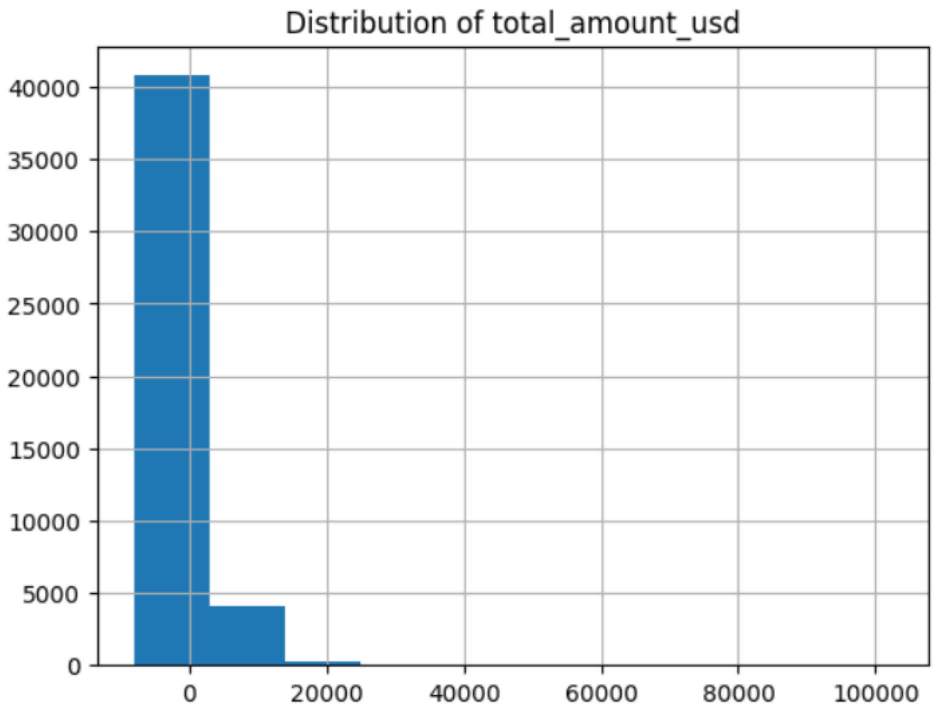
- Check features datatype

```
#Explore the distribution of the target variable
users_data.total_amount_usd.describe()
```

```
count     45179.000000
mean       1369.751283
std        2704.291321
min       -8019.000000
25%         160.000000
50%         862.000000
75%        1369.420000
max      102127.000000
Name: total_amount_usd, dtype: float64
```

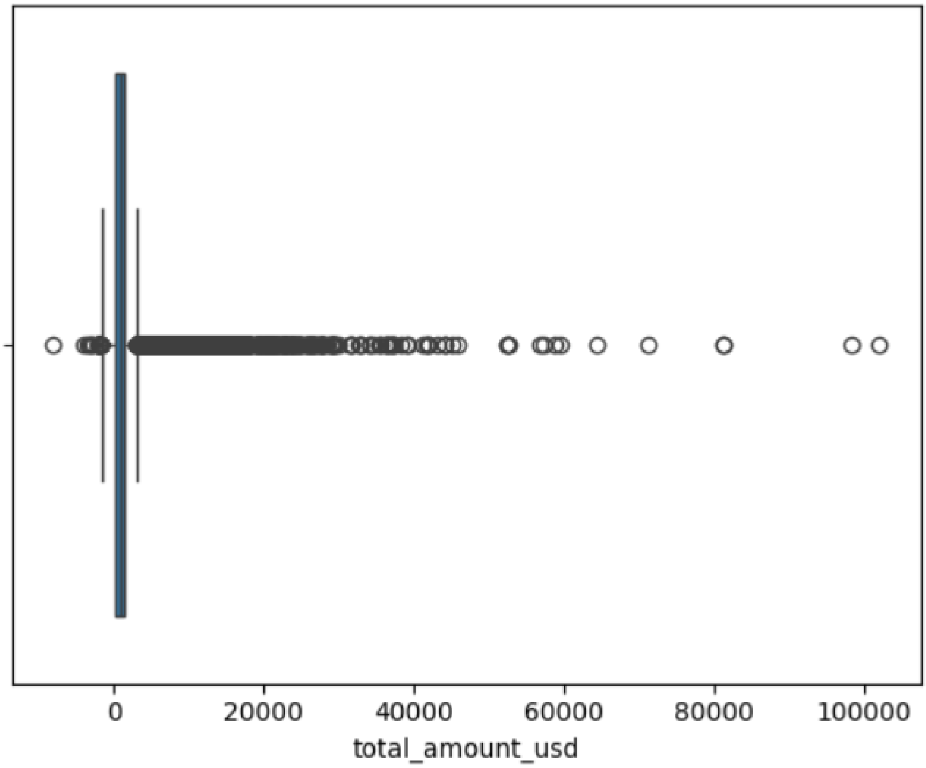
- Check info data about count,unique,top,frequent,mean,min,25%,50%,75%, max and standard for **total_amount_usd**

```
#Visualize the distribution of the target variable
users_data.total_amount_usd.hist()
plt.title('Distribution of total_amount_usd');
```



- Plot the histogram for features **total_amount_usd**

```
sns.boxplot(users_data.total_amount_usd, orient = 'h');
```



- Plot the boxplot for features **total_amount_usd**
- Check the outliers for lower bound and upper bound

```python
#Identify and remove the outliers
q1 = np.percentile(users_data.total_amount_usd, 25)
q3 = np.percentile(users_data.total_amount_usd, 75)
iqr = q3 - q1

lb = q1 - 1.5 * iqr
ub = q3 + 1.5 * iqr

print('Lower bound:', round(lb, 2))
print('Upper bound:', round(ub, 2))
```

```
Lower bound: -1654.13
Upper bound: 3183.55
```

```python
print('Number of users with total_amount_usd greater than UB:',
      users_data[(users_data.total_amount_usd >= ub)].shape[0])
print('Number of users with total_amount_usd lower than LB:  ',
      users_data[(users_data.total_amount_usd <= lb)].shape[0])
```

```
Number of users with total_amount_usd greater than UB: 4110
Number of users with total_amount_usd lower than LB:   26
```

- Get the total number of users with **total_amount_usd** greater than Upper Bound and lower then Lower Bound.

```python
users_data_wout_outliers = \
users_data[(users_data.total_amount_usd < ub) \
            & (users_data.total_amount_usd > lb)]

users_data_wout_outliers.shape
```

Python

```
(41043, 33)
```

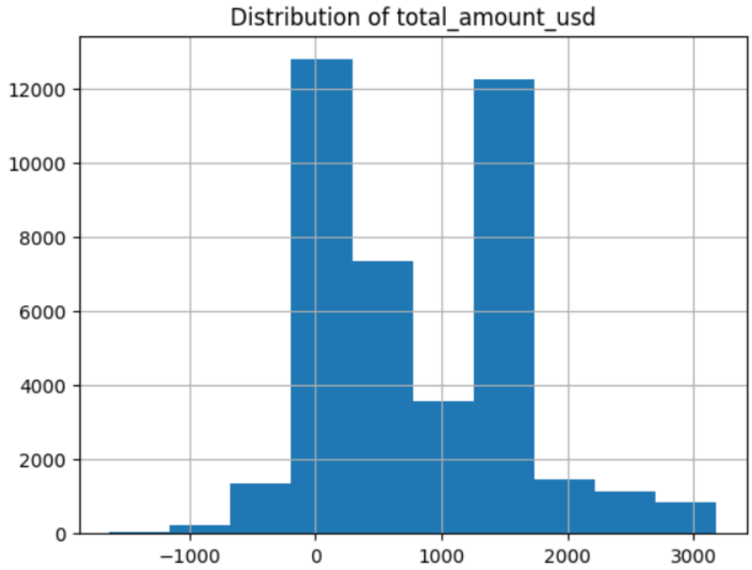
```python
#Explore the dataset with the outliers removed
users_data_wout_outliers.describe()
```

Python

	# number_transactions	# total_amount_usd	# job_management	# job_technician	# job_entrepreneur	# jol
count	41043.0	41043.0	41043.0	41043.0	41043.0	
mean	3.131179494676315	793.5782525643837	0.20203201520356698	0.1692371415344882	0.03308724995736179	
std	3.9169965142843957	761.4077590704974	0.4015208688071121	0.3749658079339652	0.17886660770973054	
min	0.0	-1636.0	0.0	0.0	0.0	
25%	1.0	123.0	0.0	0.0	0.0	
50%	2.0	659.0	0.0	0.0	0.0	
75%	3.0	1369.42	0.0	0.0	0.0	
max	20.0	3181.0	1.0	1.0	1.0	

- Get the total number of users with **total_amount_usd** after remove the outliers
- Check the total rows data and total features
- Check info data about count,unique,top,frequent,mean,min,25%,50%,75%, max and standard for **total_amount_usd**

```python
plt.title('Distribution of total_amount_usd')
users_data_wout_outliers.total_amount_usd.hist();
```



- Plot the histogram for features **total_amount_usd**

```python
#Split the data into target and features
target_data = users_data_wout_outliers.total_amount_usd
features = users_data_wout_outliers.drop(['user_id', 'total_amount_usd'],
                                          axis = 1)
```

```python
target_data.info()
```

```
<class 'pandas.core.series.Series'>
Index: 41043 entries, 0 to 45215
Series name: total_amount_usd
Non-Null Count  Dtype
--------------  -----
41043 non-null  float64
dtypes: float64(1)
```

- Split the data onto Target and features
- The target will be **total_amount_usd** and will be used as output for prediction
- The features will used as input after remove **total_amount_usd as** target and **user_id** which not used in training
- Check info datatype for feature Target

```python
features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 41043 entries, 0 to 45215
Data columns (total 31 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   number_transactions  41043 non-null  float64
 1   job_management       41043 non-null  int64  
 2   job_technician       41043 non-null  int64  
 3   job_entrepreneur     41043 non-null  int64  
 4   job_blue-collar      41043 non-null  int64  
 5   job_retired          41043 non-null  int64  
 6   job_admin.           41043 non-null  int64  
 7   job_services         41043 non-null  int64  
 8   job_self-employed    41043 non-null  int64  
 9   job_unemployed       41043 non-null  int64  
 10  job_housemaid        41043 non-null  int64  
 11  job_student          41043 non-null  int64  
 12  education_tertiary   41043 non-null  int64  
 13  education_secondary  41043 non-null  int64  
 14  education_Unknown    41043 non-null  int64  
 15  education_primary    41043 non-null  int64  
 16  default              41043 non-null  bool   
 17  housing              41043 non-null  bool   
 18  loan                 41043 non-null  bool   
 19  contact_cellular     41043 non-null  int64  
...
 29  age_group_encoded    41043 non-null  int8   
 30  month_joined         41043 non-null  int64  
dtypes: bool(4), float64(1), int64(24), int8(1), uint8(1)
memory usage: 8.4 MB
```

- The Features with exclude the **total_amount_usd** and **user_id**

```python
#Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(features,
                                                    target_data,
                                                    test_size = 0.3)
```

```python
print('Training-data features: ', X_train.shape)
print('Training-data target:   ', y_train.shape)
```

```
Training-data features:  (28730, 31)
Training-data target:    (28730,)
```

```python
#Check the distribution of the test data
print('Test-data features: ', X_test.shape)
print('Test-data target:   ', y_test.shape)
```

```
Test-data features:  (12313, 31)
Test-data target:    (12313,)
```

- Split the training data and test data
- Avoid from leaked data.
- Size Test data 30% and Training will be 70%
- Check size training data for target and features
- Check size test data for target and features

Training

1) Linear Regression Model

```python
#Train a linear regression model
linreg = LinearRegression()
linreg.fit(X_train, y_train)
```

Python

```
LinearRegression
LinearRegression()
```

```python
#Make predictions using the linear regression model
linreg_y_pred = linreg.predict(X_test)
```

Python

```python
results = pd.concat([y_test.iloc[:5], X_test.iloc[:5]], axis = 1)
results.insert(1, 'total_pred', linreg_y_pred[:5].round(2))
results
```

Python

	# total_amount_usd	# total_pred	# number_transactions	# job_management	# job_technician	# job
33025	104.0	856.8	3.0	0	0	
1893	1369.42	1129.94	0.0	0	1	
26956	219.0	994.59	2.0	0	0	
38050	170.0	829.65	2.0	0	0	
19663	53.0	816.78	3.0	1	0	

5 rows x 33 cols 10 per page Page 1 of 1

- Fit to start training the features and target
- Predict the model with test data
- Get the 5 rows prediction and test data. Concatenate these data

```python
#Obtain the linear regression model's score
r2_score(y_test, linreg_y_pred)
```

71]

```
0.23321527124088082
```

- Evaluate the model
- Near to 1 is a good model, the result still far.

2) Decision Tree Model

```
#Train a decision-tree model

reg_tree = DecisionTreeRegressor()
reg_tree.fit(X_train, y_train)
```

Pythor

```
DecisionTreeRegressor
DecisionTreeRegressor()
```

```
#Make predictions using the decision-tree model
reg_tree_y_pred = reg_tree.predict(X_test)
```

Pythor

```
results['total_pred'] = reg_tree_y_pred[:5]
results
```

Pythor

	# total_amount_usd	# total_pred	# number_transactions	# job_management	# job_technician	# jo
33025	104.0	2958.0	3.0	0	0	
1893	1369.42	1369.4199999999892	0.0	0	1	
26956	219.0	2787.0	2.0	0	0	
38050	170.0	59.0	2.0	0	0	
19663	53.0	862.0	3.0	1	0	

5 rows x 33 cols 10 per page Page 1 of 1

- Used for complex data and branched, cannot used for linear
- Fit to start training the features and target
- Predict the model with test data
- Get the 5 rows prediction and test data. Concatenate these data

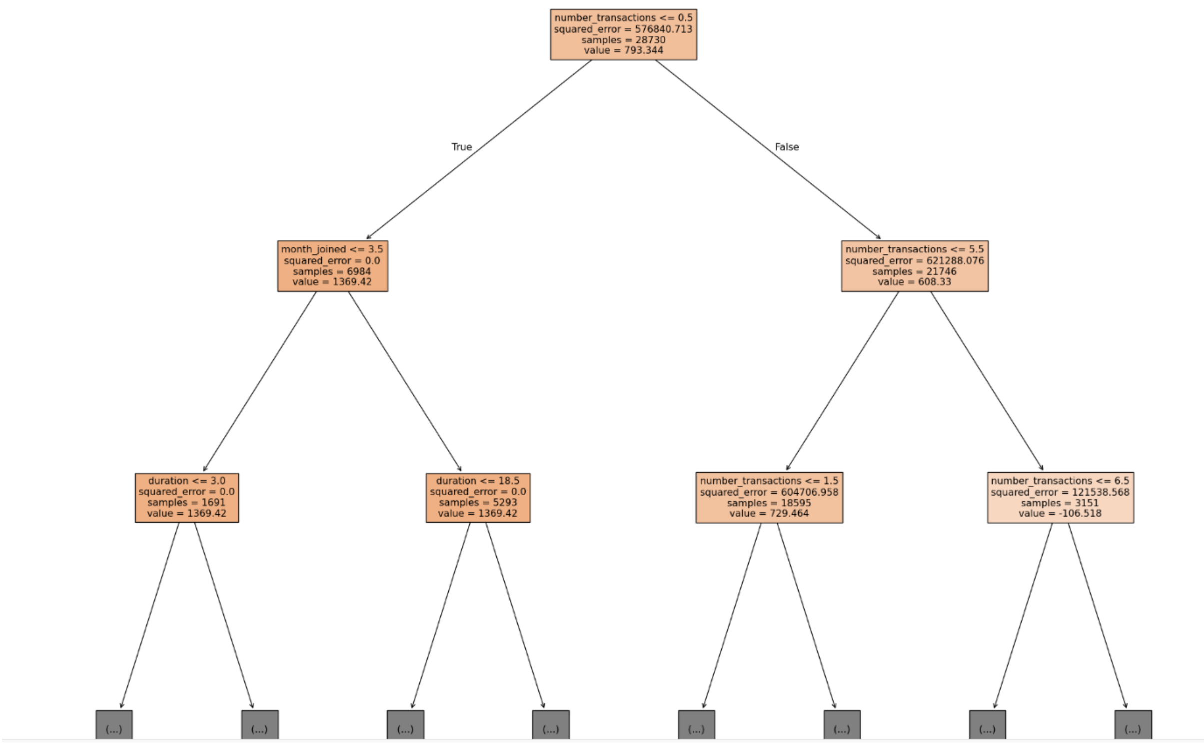
```
#Obtain the decision-tree model's score
r2_score(y_test, reg_tree_y_pred)
```

```
-0.40313871451252936
```

- R2 score below than 0, it a worse prediction. Means it underfitting, couldn't learn from data.

```
#Visualize the decision tree
fig = plt.figure(figsize = (25, 20))
_ = tree.plot_tree(reg_tree,
                   feature_names = list(X_train.columns),
                   max_depth = 2,
                   filled = True)
```

- Plot the decision Tree with 2 top layers, so easily to read.



3) Random Forest Model

```python
#Train a random-forest model
rf = RandomForestRegressor()
rf.fit(X_train, y_train)
```

Python

```
▾ RandomForestRegressor
RandomForestRegressor()
```

```python
#Make predictions using the random-forest model
rf_y_pred = rf.predict(X_test)
```

Python

```python
results['total_pred'] = rf_y_pred[:5]
results
```

Python

	# total_amount_usd	# total_pred	# number_transactions	# job_management	# job_technician	# job
33025	104.0	1187.61	3.0	0	0	
1893	1369.42	1369.4199999999496	0.0	0	1	
26956	219.0	1134.88	2.0	0	0	
38050	170.0	514.93	2.0	0	0	
19663	53.0	969.48	3.0	1	0	

5 rows x 33 cols 10 per page Page 1 of 1

- Used ensemble from multiple decision tree and run simultaneously, more stable and accurate.
- Fit to start training the features and target
- Predict the model with test data
- Get the 5 rows prediction and test data. Concatenate these data

```
#Obtain the random-forest model's score
r2_score(y_test, rf_y_pred)
```

```
0.29556907062128646
```

- Evaluate the model
- Near to 1 is a good model, the result still far.

4) XGBoost Model

```
#Train a gradient-boosting model
xgb = XGBRegressor()
xgb.fit(X_train, y_train)
```

```
XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             feature_weights=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=None, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
             max_leaves=None, min_child_weight=None, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=None,
             n_jobs=None, num_parallel_tree=None, ...)
```

```
#Make predictions using the gradient-boosting model
xgb_y_pred = xgb.predict(X_test)
```

```
results['total_pred'] = xgb_y_pred[:5]
results
```

	# total_amount_usd	# total_pred	# number_transactions	# job_management	# job_technician
33025	104.0	770.8076	3.0	0	0
1893	1369.42	1427.2244	0.0	0	1
26956	219.0	923.41144	2.0	0	0
38050	170.0	461.28928	2.0	0	0

- Powerful model boosting
- Fit to start training the features and target
- Predict the model with test data
- Get the 5 rows prediction and test data. Concatenate these data

```
#Obtain the gradient-boosting model's score
r2_score(y_test, xgb_y_pred)
```

```
0.3066909539555063
```

- Evaluate the model
- Near to 1 is a good model, the result still far.

Tuning

```
#Define the parameter grid used to tune the linear regression model

param_grid = {'l1_ratio': [0.1, 0.5, 0.9],
              'alpha': [0.0001, 0.01, 0.1],
              'max_iter': [100, 1000, 10000]}

print(param_grid)
```

```
{'l1_ratio': [0.1, 0.5, 0.9], 'alpha': [0.0001, 0.01, 0.1], 'max_iter': [100, 1000, 10000]}
```

- Tuning for linear regression
- **L1_ratio**, if 1 it Lasso regression, if 0 is ridge regression, if half/middle, can be both.
- **Alpa** is strength of regularization (strength penalty)
- **Max_iter** is Iteration process

```python
#Perform a grid search for optimal elastic net hyperparameters

model = ElasticNet()
gs = GridSearchCV(estimator = model,
                  param_grid = param_grid,
                  n_jobs = -1,
                  verbose = 2)

gs.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
```

```
▸ GridSearchCV
▸ best_estimator_: ElasticNet
  ▸ ElasticNet
```

```python
print('Best R2 score:    ', round(gs.best_score_, 4))
print('Best parameters: ', gs.best_params_)
```

```
Best R2 score:    0.2343
Best parameters:  {'alpha': 0.01, 'l1_ratio': 0.9, 'max_iter': 100}
```

- GridSearchCV will test all parameter combinations (3 × 3 × 3 = 27 combinations) and select the best one according to model performance (e.g. R^2 Score, RMSE).

```python
#Define the parameter grid used to tune the decision-tree model
param_grid = {'max_depth': [5, 10, 20],
              'min_samples_split': [10, 100, 1000],
              'min_samples_leaf': [10, 100, 1000]}

print(param_grid)
```

```
{'max_depth': [5, 10, 20], 'min_samples_split': [10, 100, 1000], 'min_samples_leaf': [10, 100, 1000]}
```

```python
#Perform a grid search for optimal decision-tree hyperparameters
model = DecisionTreeRegressor()
gs = GridSearchCV(estimator = model,
                  param_grid = param_grid,
                  n_jobs = -1,
                  verbose = 2)

gs.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
```

```
GridSearchCV
best_estimator_:
DecisionTreeRegressor
DecisionTreeRegressor
```

- Tuning for decision tree
- **Max_depth** is limiting tree depth, to avoid overfitting
- **Min_samples_split** is minimum data before split
- **Min_samples_leaf is** Minimum number of samples to become a leaf

```python
print('Best R2 score:    ', round(gs.best_score_, 4))
print('Best parameters: ', gs.best_params_)
```

```
Best R2 score:     0.3357
Best parameters:  {'max_depth': 10, 'min_samples_leaf': 100, 'min_samples_split': 1000}
```

- Evaluate the model
- Get the best score from **GridSearchCV**

```
#Compare evaluation metrics for each model

models = ['Linear Regression', 'Decision Tree',
          'Random Forest', 'XGBoost', 'Dummy Regressor']

metrics = ['R2', 'MAE', 'MSE']

pred_list = ['linreg_y_pred', 'reg_tree_y_pred',
             'rf_y_pred', 'xgb_y_pred', 'dummy_y_pred']

# Baseline algorithm.
dummy = DummyRegressor()
dummy.fit(X_train, y_train)
dummy_y_pred = dummy.predict(X_test)

scores = np.empty((0, 3))

for i in pred_list:
    scores = np.append(scores,
                       np.array([[r2_score(y_test, gl
                                  mean_absolute_error (variable) y_test: Any
                                  mean_squared_error(y_test, globals()[i])]]),
                       axis = 0)

scores = np.around(scores, 4)

scoring_df = pd.DataFrame(scores, index = models, columns = metrics)
scoring_df.sort_values(by = 'MSE', ascending = True)
```

- Compare the model by evaluate each model
- List the model which is Linear regression, Decision Tree, Random Forest, XGBoost, Dummy Regressor
- Metric for evaluate the model which is R2,MAE,MSE
- Dummy Regressor will used as baseline or benchmark
- Score value will be store in data frame and sort by MSE ascending

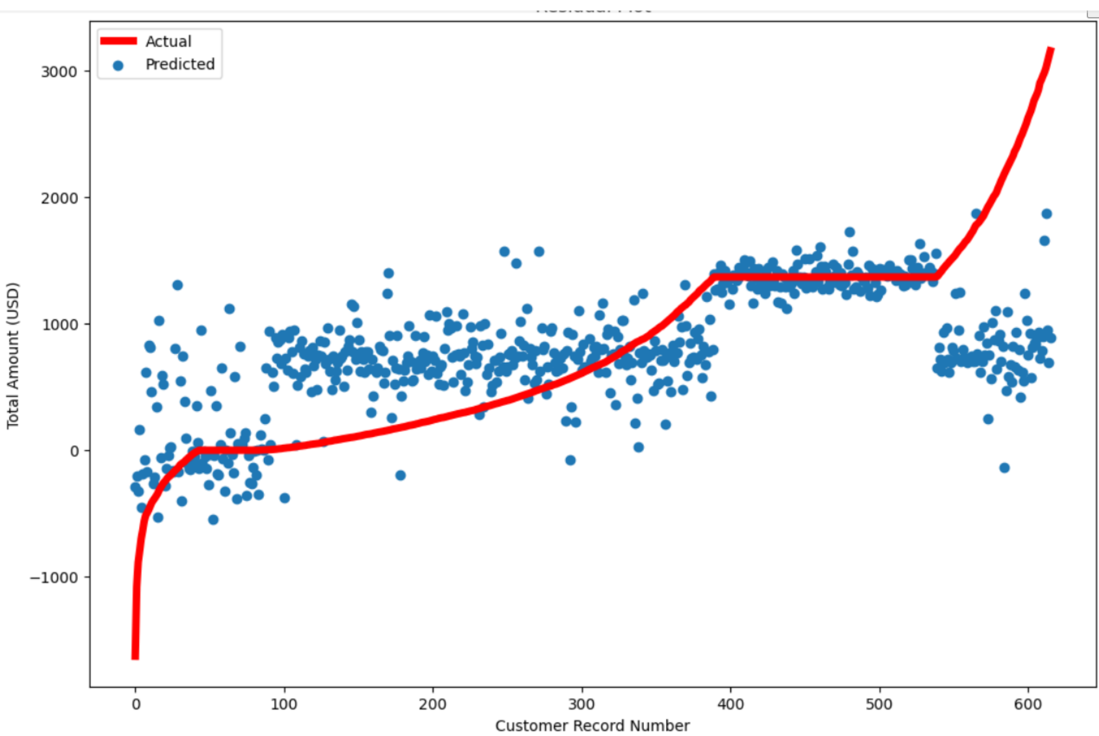
...	# R2	# MAE	# MSE
XGBoost	0.3067	433.0251	406600.3341
Random Forest	0.2956	424.5515	413122.9109
Linear Regression	0.2332	531.7977	449691.1279
Dummy Regressor	-0.0	656.6227	586463.9422
Decision Tree	-0.4031	561.2292	822889.4076

5 rows x 3 cols 10 per page Page 1 of 1

- **XGboost** is the top where MSE is the lower and it is a good model
- R2 also highest compare to the others.

```python
#Plot the residuals

# Set up data frame for plotting.

resid_df = pd.DataFrame()
resid_df['total_amount_usd'] = y_test
resid_df['total_pred'] = xgb_y_pred
resid_df['residuals'] = resid_df['total_amount_usd'] - resid_df['total_pred']
resid_df = resid_df.sort_values('total_amount_usd')[::20]
resid_df['record_num'] = np.arange(len(resid_df))
```

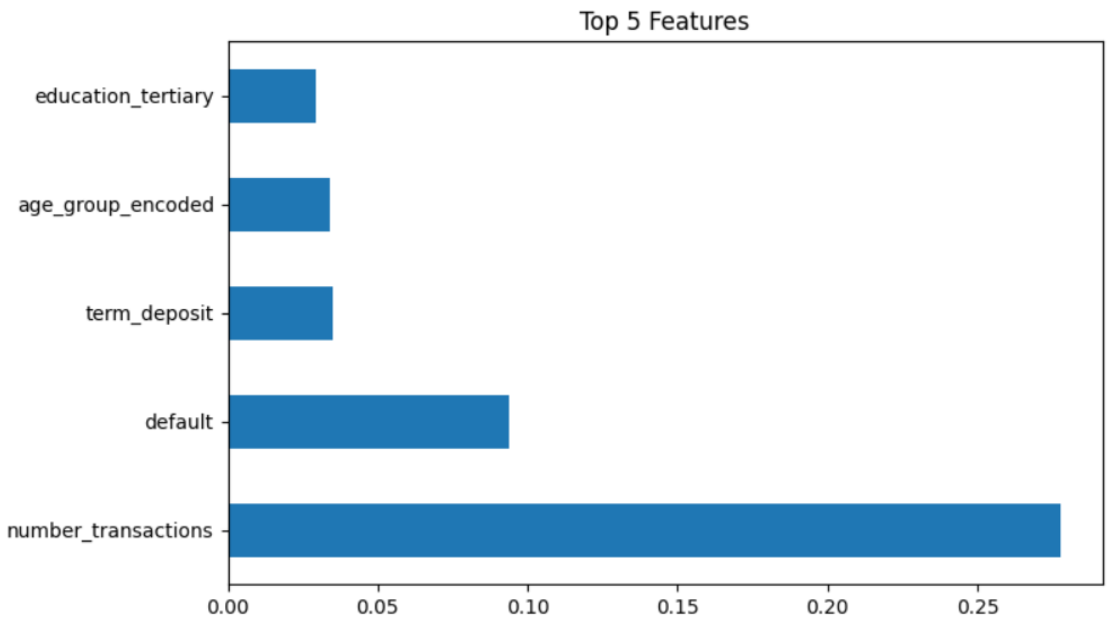
```python
plt.figure(figsize = (12, 8))

plt.plot(resid_df['record_num'], resid_df['total_amount_usd'],
         color = 'red', linewidth = 5)
plt.scatter(resid_df['record_num'], resid_df['total_pred'])

plt.legend(['Actual', 'Predicted'])
plt.title('Residual Plot')
plt.ylabel('Total Amount (USD)')
plt.xlabel('Customer Record Number')
plt.show();
```



- The blue dot is quite close to the red line, meaning the model is quite good.
- But there are certain patterns, especially at the left and right ends of the graph. On the right and left show the dot is far from the line. Which the model could not predict accurately for extreme data.
- Or the model is less extreme/complex.
- Or the middle overfitting in the middle
- Randomly distributed residuals are a sign of a good model.
- In this graph, most of the points are quite close to the line.

```python
#Generate a feature-importance plot

def feature_importance_plot(model, X_train, n):
    """Plots feature importance. Only works for ensemble learning."""
    plt.figure(figsize = (8, 5))
    feat_importances = pd.Series(model.feature_importances_,
                                 index = X_train.columns)
    feat_importances.nlargest(n).plot(kind = 'barh')
    plt.title(f'Top {n} Features')
    plt.show()
```

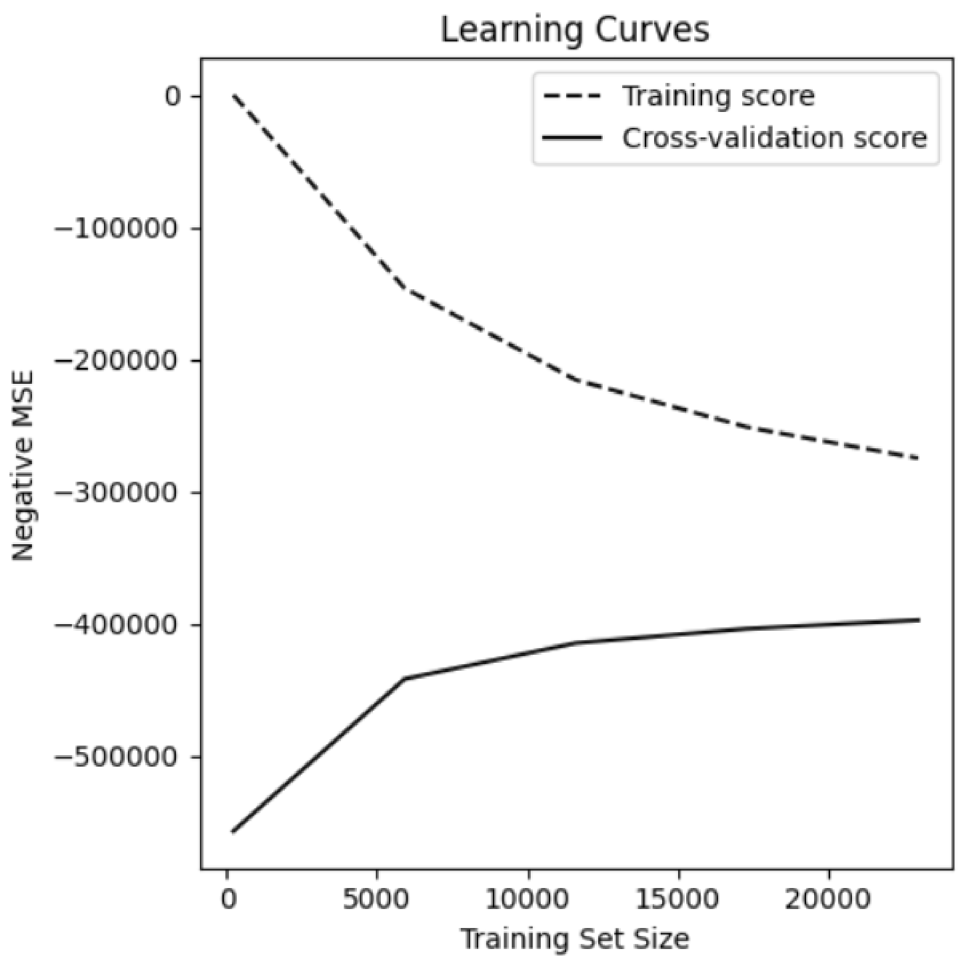
```python
feature_importance_plot(xgb, X_train, 5)
```



- What features influences the XGBoost model the most in forecasting **total_amount_usd**.
- Get Top 5 features
- Features **number_transaction** is the top 1 influence the **total_amount_usd.**

```python
#Plot learning curves

def plot_learning_curves(model, X_train, y_train):
    """Plots learning curves for model validation."""
    plt.figure(figsize = (5, 5))
    train_sizes, train_scores, test_scores = \
    learning_curve(model, X_train, y_train, cv = 5,
                   scoring = 'neg_mean_squared_error',
                   n_jobs = -1,
                   shuffle = True,
                   train_sizes = np.linspace(0.01, 1.0, 5))

    # Means of training and test set scores.
    train_mean = np.mean(train_scores, axis = 1)
    test_mean = np.mean(test_scores, axis = 1)

    # Draw lines.
    plt.plot(train_sizes, train_mean, '--',
             color = '#111111', label = 'Training score')
    plt.plot(train_sizes, test_mean,
             color = '#111111', label = 'Cross-validation score')

    # Create plot.
    plt.title('Learning Curves')
    plt.xlabel('Training Set Size')
    plt.ylabel('Negative MSE')
    plt.legend(loc = 'best')
    plt.tight_layout()

    plt.show()
```

```
plot_learning_curves(xgb, X_train, y_train)
```



- The training score (negative MSE) increasingly when the total data also increase
- The model learns better when given more data.
- But the gap with the CV score remains large — this is a sign of overfitting.
- Cross-validation Score increases slowly and levels off. The model doesn't improve much even with more data. It is possible that the model has reached saturation point.
- Big gap between Training and CV Score. Means that model is overfitting. The model performs very well on the training data. But weak when tested with new data (CV).