Forage data internship

Name: Soyinka 'Sho' Sowoolu

email: soyinkas1@gmail.com

ANZ- Predictive Analytics (Mandatory Task)

Background Information

This task is based on a synthesised transaction dataset containing 3 months' worth of transactions for 100 hypothetical customers. It contains purchases, recurring transactions, and salary transactions.

Using the same transaction dataset, identify the annual salary for each customer

Explore correlations between annual salary and various customer attributes (e.g. age). These attributes could be those that are readily available in the data (e.g. age) or those that you construct or derive yourself (e.g. those relating to purchasing behaviour). Visualise any interesting correlations using a scatter plot.

Build a simple regression model to predict the annual salary for each customer using the attributes you identified above How accurate is your model? Should ANZ use it to segment customers (for whom it does not have this data) into income brackets for reporting purposes?

For a challenge: build a decision-tree based model to predict salary. Does it perform better? How would you accurately test the performance of this model?

Work plan

There are three main parts to the task from the above:

A. ***Import all the required libraries and import the main dataset***

B. ***Identify the annual salary and explore correlation between annual salary and various customer attributes***

```
1. Extract the salary transactions from each data set
2. Create a column for annual salary and calculate this from
monthly salary
3. Create additional columns by feature engineering (library for
this?) e.g. average merchant purchases per month, average other
purchase per month
4.Check correlation with derived features , age, location, with
```

```
scatter plot and heatmap
```

C. ***Build a simple regression model***

```
5. Use the customer ID as index and annual salary as labels
6. Split data into training and test sets (use cross-validation?)
7. Fit the data to a simple linear regression model and check the
accuracy
8. Improve the model by tuning hyperparameters (RandomSearchCV
and/or GridSearchCV)
9. Select model with the best accuracy
```

D.***Build a decision tree model to predict salary***

```
10. Use RandomForest as the decision tree based model to compare
the with the simple linear regression model.
```

A. Import all the required libraries and import the main dataset

In [1]:
```python
# Import the required python libraries
import pandas as pd
import sklearn
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```python
# Import the main dataset
df = pd.read_excel('ANZ synthesised transaction dataset.xlsx')
df.head()
```

Out[2]:

	status	card_present_flag	bpay_biller_code	account	currency	long_lat	txn_description
0	authorized	1.0	NaN	ACC-1598451071	AUD	153.41 -27.95	POS
1	authorized	0.0	NaN	ACC-1598451071	AUD	153.41 -27.95	SALES-POS
2	authorized	1.0	NaN	ACC-1222300524	AUD	151.23 -33.94	POS
3	authorized	1.0	NaN	ACC-1037050564	AUD	153.10 -27.66	SALES-POS
4	authorized	1.0	NaN	ACC-1598451071	AUD	153.41 -27.95	SALES-POS

5 rows × 23 columns

In [3]:

```
df.columns
```

Out[3]:

```
Index(['status', 'card_present_flag', 'bpay_biller_code', 'account',
       'currency', 'long_lat', 'txn_description', 'merchant_id',
       'merchant_code', 'first_name', 'balance', 'date', 'gender', 'age',
       'merchant_suburb', 'merchant_state', 'extraction', 'amount',
       'transaction_id', 'country', 'customer_id', 'merchant_long_lat',
       'movement'],
      dtype='object')
```

B. Identify the annual salary and explore correlation between annual salary and various customer attributes

In [4]:

```
# Create a DataFrame with the salary transactions only
df_salary =df[df['txn_description']=='PAY/SALARY']
df_salary
```

Out[4]:

	status	card_present_flag	bpay_biller_code	account	currency	long_lat	txn_descriptio
50	posted	NaN	0	ACC-588564840	AUD	151.27 -33.76	PAY/SALAR
61	posted	NaN	0	ACC-1650504218	AUD	145.01 -37.93	PAY/SALAR
64	posted	NaN	0	ACC-3326339947	AUD	151.18 -33.80	PAY/SALAR
68	posted	NaN	0	ACC-3541460373	AUD	145.00 -37.83	PAY/SALAR
70	posted	NaN	0	ACC-2776252858	AUD	144.95 -37.76	PAY/SALAR
...	...	...	...	...	...	...	.
11995	posted	NaN	0	ACC-1973887809	AUD	115.78 -31.90	PAY/SALAR
12000	posted	NaN	0	ACC-819621312	AUD	145.04 -37.85	PAY/SALAR
12001	posted	NaN	0	ACC-2920611728	AUD	144.96 -37.76	PAY/SALAR
12003	posted	NaN	0	ACC-1799207998	AUD	150.68 -33.79	PAY/SALAR
12004	posted	NaN	0	ACC-2171593283	AUD	146.94 -36.04	PAY/SALAR

883 rows × 23 columns

In [5]:
```
# Group by 'customer_id' to get the total salary for each customer for the 3 month
df_salary.groupby('customer_id')['amount'].sum()
```

Out[5]:
```
customer_id
CUS-1005756958    12616.11
CUS-1117979751    25050.55
CUS-1140341822    11499.06
CUS-1147642491    22248.07
CUS-1196156254    27326.11
                    ...
CUS-72755508       8703.84
CUS-809013380     13481.91
CUS-860700529     10851.72
CUS-880898248      8603.88
CUS-883482547     27842.22
Name: amount, Length: 100, dtype: float64
```

In [6]:
```
# Confirm the salary frequency by inspecting just one of the customers
df_salary[df_salary['customer_id']=='CUS-1005756958']
```

Out[6]:

	status	card_present_flag	bpay_biller_code	account	currency	long_lat	txn_descriptio
841	posted	NaN	0	ACC-2828321672	AUD	153.03 -27.51	PAY/SALAR
1744	posted	NaN	0	ACC-2828321672	AUD	153.03 -27.51	PAY/SALAR
2530	posted	NaN	0	ACC-2828321672	AUD	153.03 -27.51	PAY/SALAR
3464	posted	NaN	0	ACC-2828321672	AUD	153.03 -27.51	PAY/SALAR
4402	posted	NaN	0	ACC-2828321672	AUD	153.03 -27.51	PAY/SALAR
5328	posted	NaN	0	ACC-2828321672	AUD	153.03 -27.51	PAY/SALAR
6271	posted	NaN	0	ACC-2828321672	AUD	153.03 -27.51	PAY/SALAR
7203	posted	NaN	0	ACC-2828321672	AUD	153.03 -27.51	PAY/SALAR
8142	posted	NaN	0	ACC-2828321672	AUD	153.03 -27.51	PAY/SALAR
9072	posted	NaN	0	ACC-2828321672	AUD	153.03 -27.51	PAY/SALAR
10008	posted	NaN	0	ACC-2828321672	AUD	153.03 -27.51	PAY/SALAR
10951	posted	NaN	0	ACC-2828321672	AUD	153.03 -27.51	PAY/SALAR
11871	posted	NaN	0	ACC-2828321672	AUD	153.03 -27.51	PAY/SALAR

13 rows × 23 columns

> From above the salary is on a weekly basis. We will double check by sampling another customer

In [7]:

```
# Reconfirm the salary frequency by inspecting another customer
df_salary[df_salary['customer_id']=='CUS-809013380']
```

Out[7]:

	status	card_present_flag	bpay_biller_code	account	currency	long_lat	txn_descriptio
827	posted	NaN	0	ACC-1990648130	AUD	114.62 -28.80	PAY/SALAR
1732	posted	NaN	0	ACC-1990648130	AUD	114.62 -28.80	PAY/SALAR
2516	posted	NaN	0	ACC-1990648130	AUD	114.62 -28.80	PAY/SALAR
3453	posted	NaN	0	ACC-1990648130	AUD	114.62 -28.80	PAY/SALAR
4388	posted	NaN	0	ACC-1990648130	AUD	114.62 -28.80	PAY/SALAR
5315	posted	NaN	0	ACC-1990648130	AUD	114.62 -28.80	PAY/SALAR
6257	posted	NaN	0	ACC-1990648130	AUD	114.62 -28.80	PAY/SALAR
7189	posted	NaN	0	ACC-1990648130	AUD	114.62 -28.80	PAY/SALAR
8129	posted	NaN	0	ACC-1990648130	AUD	114.62 -28.80	PAY/SALAR
9060	posted	NaN	0	ACC-1990648130	AUD	114.62 -28.80	PAY/SALAR
10000	posted	NaN	0	ACC-1990648130	AUD	114.62 -28.80	PAY/SALAR
10944	posted	NaN	0	ACC-1990648130	AUD	114.62 -28.80	PAY/SALAR
11864	posted	NaN	0	ACC-1990648130	AUD	114.62 -28.80	PAY/SALAR

13 rows × 23 columns

> We will extract the weekly salary for each customer and multiply by 52 weeks to get the annual salary. A new annual salary column will be added to the main dataset.

In [8]:

```
# Calculate the 3 months total salary and pass to a DataFrame
df_sal_total =pd.DataFrame(df_salary.groupby('customer_id')['amount'].sum())
df_sal_total
```

Out[8]:

customer_id	amount
CUS-1005756958	12616.11
CUS-1117979751	25050.55
CUS-1140341822	11499.06
CUS-1147642491	22248.07
CUS-1196156254	27326.11
...	...
CUS-72755508	8703.84
CUS-809013380	13481.91
CUS-860700529	10851.72
CUS-880898248	8603.88
CUS-883482547	27842.22

100 rows × 1 columns

In [9]:
```
# Calculate the annual salary and create a column for this
df_sal_total['annual salary'] =(df_sal_total['amount']/13)*52

# Drop the 3 months total amount
df_sal_total.drop('amount',axis=1,inplace=True)

#DataFrame with annual salary
df_sal_total
```

Out[9]:

customer_id	annual salary
CUS-1005756958	50464.44
CUS-1117979751	100202.20
CUS-1140341822	45996.24
CUS-1147642491	88992.28
CUS-1196156254	109304.44
...	...
CUS-72755508	34815.36
CUS-809013380	53927.64
CUS-860700529	43406.88
CUS-880898248	34415.52
CUS-883482547	111368.88

100 rows × 1 columns

In [10]:
```
# We will create features representing transaction behaviours e.g. average purchase
```

```
# Create a Dataframe for merchant transaction only(card transactions)
df_merchants=df[(df['txn_description']=='POS') | (df['txn_description']=='SALES-PO

# Create a DataFrame for transactions by the customer directly (without merchants 
df_cust_only=df[(df['txn_description']=='PAYMENT') |
                (df['txn_description']=='INTER BANK')|
                (df['txn_description']=='PHONE BANK')]
```

In [11]:
```
# Create a DataFrame for average amount spent on purchases with merchants
df_mer_avg=pd.DataFrame(df_merchants.groupby('customer_id')['amount'].mean())

# name columns with appropriate description
df_mer_avg.rename(columns={'amount':'avg_mer_spend'},inplace=True)
df_mer_avg
```

Out[11]:

	avg_mer_spend
customer_id	
CUS-1005756958	37.726250
CUS-1117979751	76.458077
CUS-1140341822	67.531385
CUS-1147642491	51.128289
CUS-1196156254	30.310491
...	...
CUS-72755508	34.545111
CUS-809013380	27.297500
CUS-860700529	29.044466
CUS-880898248	28.903036
CUS-883482547	30.884917

100 rows × 1 columns

In [12]:
```
# Create a DataFrame for average amount spent on other expenses
df_othexp_avg=pd.DataFrame(df_cust_only.groupby('customer_id')['amount'].mean())

# name columns with appropriate description
df_othexp_avg.rename(columns={'amount':'avg_othexp_spend'},inplace=True)
df_othexp_avg
```

Out[12]:

	avg_othexp_spend
customer_id	
CUS-1005756958	153.500000
CUS-1117979751	120.926829
CUS-1140341822	124.666667
CUS-1147642491	98.172414
CUS-1196156254	50.453333
...	...
CUS-72755508	1180.000000
CUS-809013380	127.217391
CUS-860700529	60.238095
CUS-880898248	77.500000
CUS-883482547	99.823529

100 rows × 1 columns

In [13]:

```
# Create a DataFrame for average balance spent on customer's account
df_bal_avg=pd.DataFrame(df_cust_only.groupby('customer_id')['balance'].mean())

# name columns with appropriate description
df_bal_avg.rename(columns={'balance':'avg_bal'},inplace=True)
df_bal_avg
```

Out[13]:

	avg_bal
customer_id	
CUS-1005756958	4187.891667
CUS-1117979751	10601.552683
CUS-1140341822	5334.007778
CUS-1147642491	8321.411724
CUS-1196156254	22766.256933
...	...
CUS-72755508	5227.940000
CUS-809013380	4680.580435
CUS-860700529	3254.679048
CUS-880898248	8793.244375
CUS-883482547	10574.021569

100 rows × 1 columns

In [14]:

```
# Other attribute we will look at is the age and location of the customer.
# Get the age of each customer
df_age=pd.DataFrame(df.groupby('customer_id')['age'].mean())
df_age
```

Out[14]:

	age
customer_id	
CUS-1005756958	53.0
CUS-1117979751	21.0
CUS-1140341822	28.0
CUS-1147642491	34.0
CUS-1196156254	34.0
...	...
CUS-72755508	35.0
CUS-809013380	21.0
CUS-860700529	30.0
CUS-880898248	26.0
CUS-883482547	19.0

100 rows × 1 columns

In [15]:
```
# Get the location of each customer
df_cust_loc= pd.DataFrame(df.groupby(['customer_id','long_lat']).count())
```

In [16]:
```
df_cust_loc.reset_index(level='long_lat',inplace=True)
df_cust_loc=df_cust_loc.loc[:,['long_lat']]
df_cust_loc
```

Out[16]:

	long_lat
customer_id	
CUS-1005756958	153.03 -27.51
CUS-1117979751	115.81 -31.82
CUS-1140341822	144.97 -37.42
CUS-1147642491	151.04 -33.77
CUS-1196156254	138.52 -35.01
...	...
CUS-72755508	150.62 -33.76
CUS-809013380	114.62 -28.80
CUS-860700529	153.05 -27.61
CUS-880898248	144.89 -37.69
CUS-883482547	150.82 -34.01

100 rows × 1 columns

In [17]:
```
# We will split the coordinates into seperate longitude and latitude and also conve

# Split the 'long_lat'column into 'long' and 'lat'
df_cust_loc[['c_long','c_lat']] = df_cust_loc['long_lat'].str.split(" ",expand=Tru
```

```
# Convert the columns to float
df_cust_loc[['c_long','c_lat']]=df_cust_loc[['c_long','c_lat']].astype('float64')
df_cust_loc[['c_long','c_lat']].dtypes
```

Out[17]:
```
c_long    float64
c_lat     float64
dtype: object
```

In [18]:
```
# remove the long_lat column
df_cust_loc.drop('long_lat',axis=1,inplace=True)
df_cust_loc
```

Out[18]:

	c_long	c_lat
customer_id		
CUS-1005756958	153.03	-27.51
CUS-1117979751	115.81	-31.82
CUS-1140341822	144.97	-37.42
CUS-1147642491	151.04	-33.77
CUS-1196156254	138.52	-35.01
...	...	...
CUS-72755508	150.62	-33.76
CUS-809013380	114.62	-28.80
CUS-860700529	153.05	-27.61
CUS-880898248	144.89	-37.69
CUS-883482547	150.82	-34.01

100 rows × 2 columns

In [19]:
```
# We will create a DataFrame of the gender of the customer
# Get the location of each customer
df_cust_sex= pd.DataFrame(df.groupby(['customer_id','gender']).count())

# make gender index a column
df_cust_sex.reset_index(level='gender',inplace=True)
# Drop all other columns except gender
df_cust_sex=df_cust_sex.loc[:,['gender']]
df_cust_sex
```

Out[19]:

	gender
customer_id	
CUS-1005756958	F
CUS-1117979751	M
CUS-1140341822	M
CUS-1147642491	F
CUS-1196156254	F
...	...
CUS-72755508	F
CUS-809013380	F
CUS-860700529	M
CUS-880898248	M
CUS-883482547	F

100 rows × 1 columns

In [20]:

```
# Convert the gender to numeric by making Male =1 and Female = 0
df_cust_sex['gender'] = np.where(df_cust_sex['gender'] == 'F', 0, 1)
df_cust_sex
```

Out[20]:

	gender
customer_id	
CUS-1005756958	0
CUS-1117979751	1
CUS-1140341822	1
CUS-1147642491	0
CUS-1196156254	0
...	...
CUS-72755508	0
CUS-809013380	0
CUS-860700529	1
CUS-880898248	1
CUS-883482547	0

100 rows × 1 columns

> We can now conbine all the dataframe into a single DataFrame using the following
>
> - df_sal_total
> - df_cust_loc df_mer_avg

- df_age
- df_othexp_avg
- df_cust_sexm
- df_bal_avg

In [21]:
```
# Combined DataFrame with attributes of each customer.
df_cust_info = [df_age, df_cust_sex,df_cust_loc, df_mer_avg,df_othexp_avg,df_bal_a

df_cust_info = pd.concat(df_cust_info, join='outer', axis = 1)
df_cust_info
```

Out[21]:

	age	gender	c_long	c_lat	avg_mer_spend	avg_othexp_spend	avg_bal	a s
customer_id								
CUS-1005756958	53.0	0	153.03	-27.51	37.726250	153.500000	4187.891667	504
CUS-1117979751	21.0	1	115.81	-31.82	76.458077	120.926829	10601.552683	1002
CUS-1140341822	28.0	1	144.97	-37.42	67.531385	124.666667	5334.007778	459
CUS-1147642491	34.0	0	151.04	-33.77	51.128289	98.172414	8321.411724	889
CUS-1196156254	34.0	0	138.52	-35.01	30.310491	50.453333	22766.256933	1093
...	...	...	...	...	...	...	...	
CUS-72755508	35.0	0	150.62	-33.76	34.545111	1180.000000	5227.940000	348
CUS-809013380	21.0	0	114.62	-28.80	27.297500	127.217391	4680.580435	539
CUS-860700529	30.0	1	153.05	-27.61	29.044466	60.238095	3254.679048	434
CUS-880898248	26.0	1	144.89	-37.69	28.903036	77.500000	8793.244375	344
CUS-883482547	19.0	0	150.82	-34.01	30.884917	99.823529	10574.021569	1113

100 rows × 8 columns

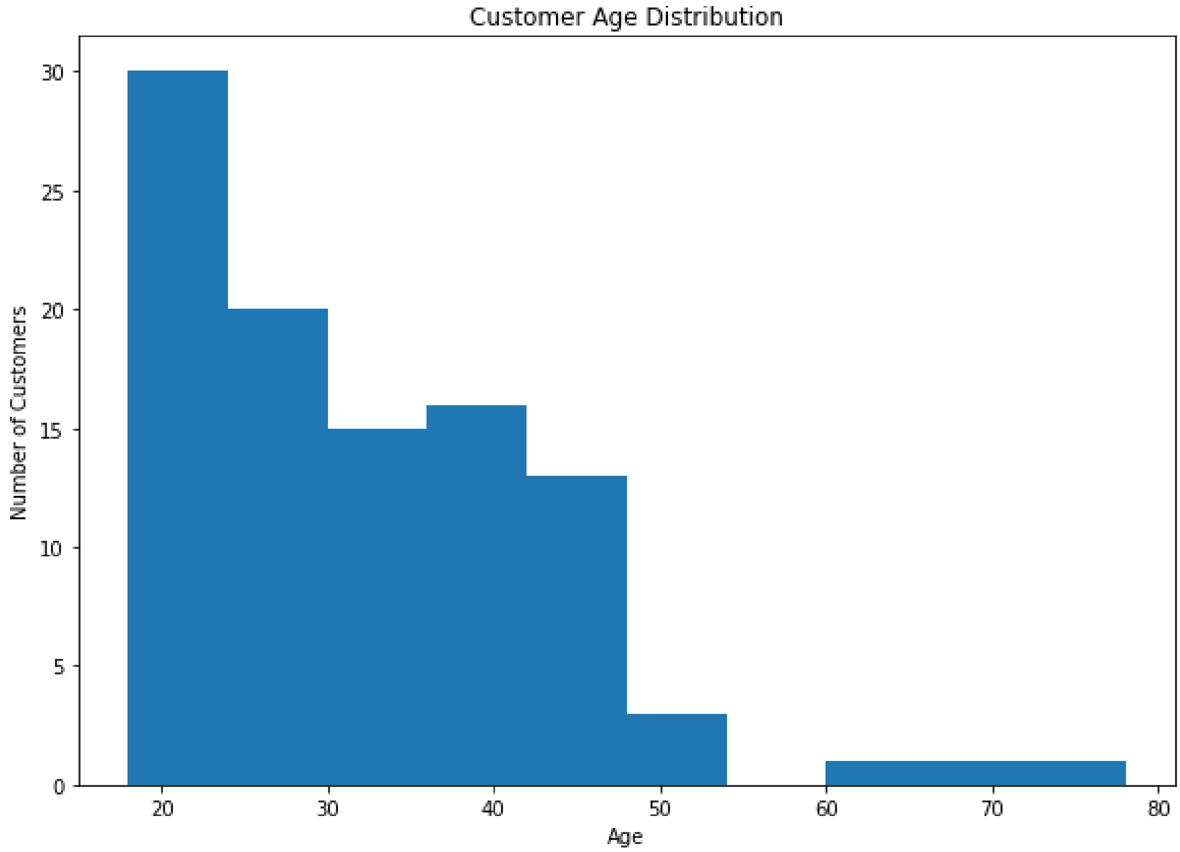
In [22]:
```
# Confirm there are no missing values
df_cust_info.isna().sum()
```

Out[22]:
```
age                 0
gender              0
c_long              0
c_lat               0
avg_mer_spend       0
avg_othexp_spend    0
avg_bal             0
annual salary       0
dtype: int64
```

In [23]:
```
# Save the DataFrame for future use
df_cust_info.to_excel('anz customer atributes.xlsx',float_format='%.3f')
```

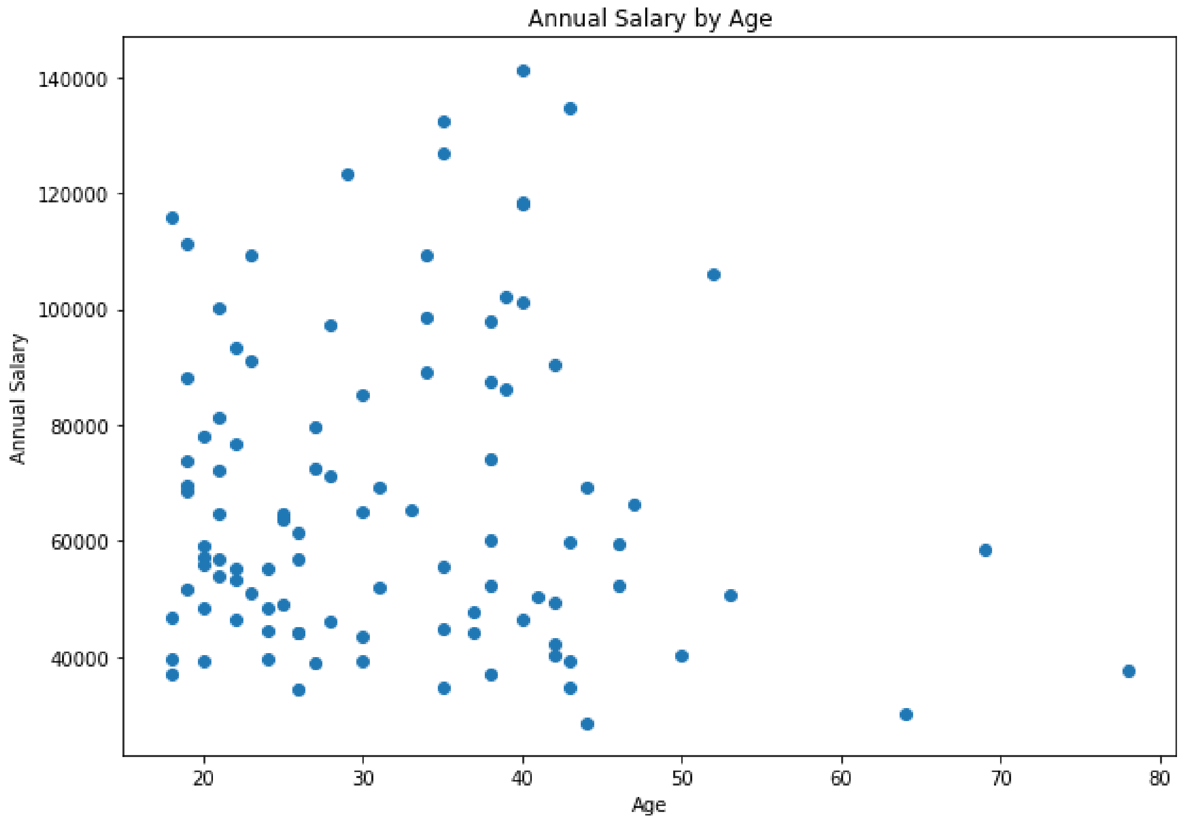
> Check the correlation of the different attributes using scatter diagram

In [24]:
```
# Plot histogram of the customer age distribution
plt.figure(figsize=(10,7))
plt.hist(df_cust_info['age']);
plt.xlabel('Age')
plt.ylabel('Number of Customers')
plt.title('Customer Age Distribution');
```



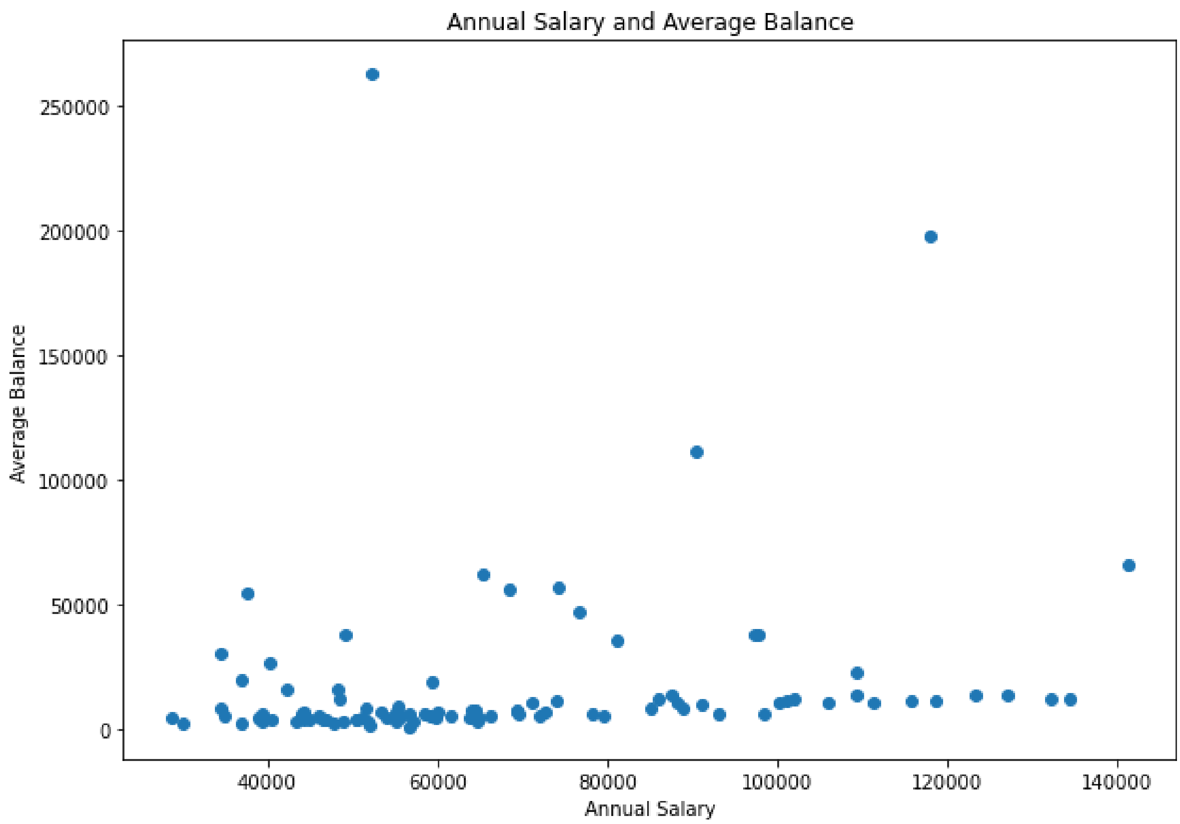
Majority of the customer are below 50 years.

In [25]:
```
# Plot a scatter diagram of the annual salary and age
plt.figure(figsize=(10,7))
plt.scatter(df_cust_info['age'],df_cust_info['annual salary'])
plt.xlabel('Age')
plt.ylabel('Annual Salary')
plt.title('Annual Salary by Age');
```



In [26]:

```
# Plot a scatter diagram of the annual salary and average balance
plt.figure(figsize=(10,7))
plt.scatter(df_cust_info['annual salary'],df_cust_info['avg_bal'])
plt.xlabel('Annual Salary')
plt.ylabel('Average Balance')
plt.title('Annual Salary and Average Balance');
```

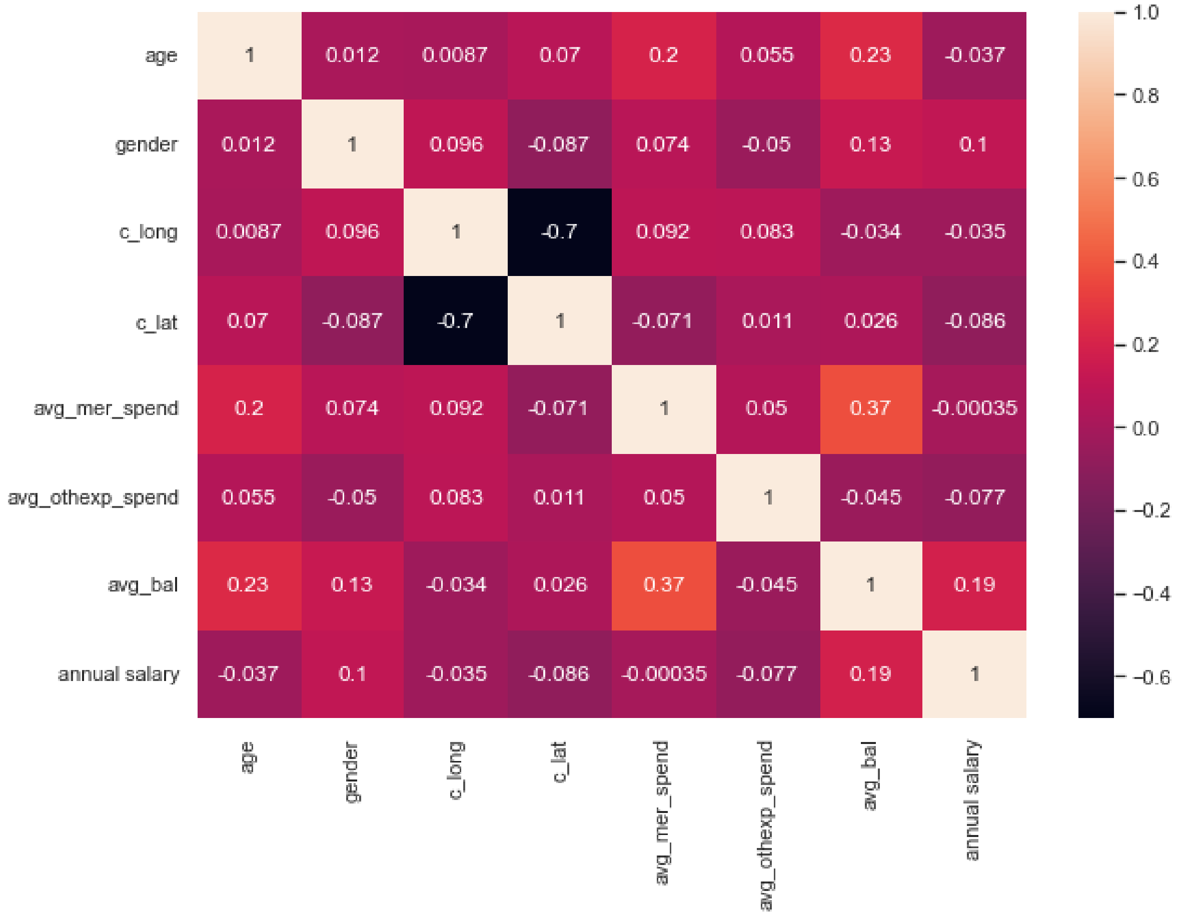


> There seems to be a very weak positive correlation between annual salary and average balance

In [27]:
```
# Plot a scatter diagram of the annual salary and location
import plotly.express as px
fig = px.scatter_geo(df_cust_info,lat='c_lat',lon='c_long', hover_name=df_cust_info
fig.update_layout(title = "Annual Salary by Location", title_x=0.2)
fig.show()
```

> There seem not be any strong linear correlation with the attributes and annual salary. We can confirm thsi holistically by checking the features correlation

In [28]:
```
# Let's check out a correlation matrix
sns.set(rc={'figure.figsize':(10,7)})
sns.heatmap(df_cust_info.corr(),annot=True);
```



> This shows there is not much correlation whether positive or negative to the annual salary as mentioned above

C. Build a simple regression model

```
5. Use the customer ID as index and annual salary as labels
6. Split data into training and test sets (use cross-validation?)
7. Fit the data to a simple linear regression model and check the
accuracy
8. Improve the model by tuning hyperparameters (RandomSearchCV
and/or GridSearchCV)
9. Select model with the best accuracy
```

We will use the scikit-learn library to train a linear regression model.

The following will be the evaluation metrics to use for evaluation:

```
* R2
* MAE
* MSE
* RMSLE
```

In [29]:
```
# Make a copy of the DataFrame for control purposes
df_cust_info_copy=df_cust_info.copy()
```

In [30]:
```
# df_cust_info=df_cust_info_copy
```

In [31]:
```
# Import the scikit-learn library to spllit the dataset
from sklearn.model_selection import train_test_split

# Split our data into features (X) and labels (y)
X= df_cust_info.drop('annual salary',axis=1)
y= df_cust_info['annual salary']
```

In [32]:
```
X
```

Out[32]:

customer_id	age	gender	c_long	c_lat	avg_mer_spend	avg_othexp_spend	avg_bal
CUS-1005756958	53.0	0	153.03	-27.51	37.726250	153.500000	4187.891667
CUS-1117979751	21.0	1	115.81	-31.82	76.458077	120.926829	10601.552683
CUS-1140341822	28.0	1	144.97	-37.42	67.531385	124.666667	5334.007778
CUS-1147642491	34.0	0	151.04	-33.77	51.128289	98.172414	8321.411724
CUS-1196156254	34.0	0	138.52	-35.01	30.310491	50.453333	22766.256933
...	...	...	...	...	...	...	...
CUS-72755508	35.0	0	150.62	-33.76	34.545111	1180.000000	5227.940000
CUS-809013380	21.0	0	114.62	-28.80	27.297500	127.217391	4680.580435
CUS-860700529	30.0	1	153.05	-27.61	29.044466	60.238095	3254.679048
CUS-880898248	26.0	1	144.89	-37.69	28.903036	77.500000	8793.244375
CUS-883482547	19.0	0	150.82	-34.01	30.884917	99.823529	10574.021569

100 rows × 7 columns

In [33]:
```
y
```

Out[33]:
```
customer_id
CUS-1005756958     50464.44
CUS-1117979751    100202.20
CUS-1140341822     45996.24
CUS-1147642491     88992.28
CUS-1196156254    109304.44
                    ...
CUS-72755508       34815.36
CUS-809013380      53927.64
CUS-860700529      43406.88
CUS-880898248      34415.52
CUS-883482547     111368.88
Name: annual salary, Length: 100, dtype: float64
```

In [34]:
```
# Confirm the shape of the two datasets
X.shape,y.shape
```

Out[34]:
```
((100, 7), (100,))
```

In [35]:
```
# Lets split our datasets into training and test data
X_train,X_test,y_train,y_test =train_test_split(X,y,test_size=0.2)

X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

Out[35]:
```
((80, 7), (80,), (20, 7), (20,))
```

In [36]:
```
# Import the linear regression model from scikit-learn
from sklearn.linear_model import LinearRegression

# Import the linear regression model evaluation metrics from scikit-learn
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_log_error


# Instantiate the model
model = LinearRegression()

# Fit the model to the dataset
model.fit (X,y)
```

Out[36]:
```
▾ LinearRegression
LinearRegression()
```

In [37]:
```
# Create a function to evaluate our model

# Create the rmsle function
def rmsle(y_train,y_pred):
    """
    Calculates root mean squared log error between prediction and true labels from
    the mean squared log error
    """
    score = np.sqrt(mean_squared_log_error(y_train,y_pred))
    return score



# Create the main evaluation function for all levels required

def model_score (model):
    """
    This function provides the score of the model evaluation from the  metrics:
    R2, MAE, MSE and RMSLE
    """

    y_pred=model.predict(X_test)
    scores={'R^2':r2_score(y_test,y_pred),
           'MAE':mean_absolute_error(y_test,y_pred),
           'MSE':mean_squared_error(y_test,y_pred),
           'RMSLE':rmsle(y_test,y_pred)}
    return scores
```

In [38]:
```
model_score(model)
```

Out[38]:
```
{'R^2': 0.002172454894987519,
 'MAE': 20454.09915452352,
 'MSE': 683109468.9918778,
 'RMSLE': 0.38530329377963685}
```

Task Question How accurate is your model? Should ANZ use it to segment customers (for whom it does not have this data) into income brackets for reporting purposes?

Answer

The simple regression model is not very accurate as there is a high error between the predicted value and the expected value from the vaue of the evaluation metrics for example, a MAE of 20,929 is a very significant error. This will thus not be recommended for use to segment customers into income bracket until a better model is built either by - getting more data to train the model or trying another model.

We will try the following decision tree model as required on the task

- Random Forest -RandomForestRegressor()
- XGBoost -XGBRegressor()

In [39]:
```
y_test
```

Out[39]:
```
customer_id
CUS-2178051368     48857.84
CUS-2738291516    132327.52
CUS-3201519139     44004.00
CUS-3716701010     66168.44
CUS-326006476      55068.16
CUS-3336454548    115702.44
CUS-127297539      59217.08
CUS-809013380      53927.64
CUS-1433879684     39426.24
CUS-3462882033     76680.24
CUS-860700529      43406.88
CUS-72755508       34815.36
CUS-3249305314     97809.40
CUS-1896554896     55408.08
CUS-331942311      44235.36
CUS-2599279756     39128.64
CUS-3989008654     64508.40
CUS-1614226872     46388.68
CUS-3378712515     55111.68
CUS-1654129794     29952.00
Name: annual salary, dtype: float64
```

In [40]:
```
y_pred=model.predict(X_test)
y_pred
```

Out[40]:
```
array([66551.3848536 , 57107.23113412, 57476.40663423, 68643.64454846,
       66303.45917078, 67997.06941676, 72706.77844382, 72171.60982439,
       68271.80041074, 75627.59199337, 64493.41356036, 51635.91361618,
       67200.47979867, 64827.87492373, 59774.23789294, 62186.4236925 ,
       76901.36276885, 59301.7988839 , 64219.98494941, 58653.14125951])
```

D.Build a decision tree model to predict salary

-Random Forest -RandomForestRegressor()

In [41]:
```
# Import the necessary library for Random Forest -RandomForestRegressor()
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
```

In [42]:
```
# Instanstiate the RF Regressor using default values
forest_reg = RandomForestRegressor()

# Fit the model to the data
forest_reg.fit(X_train,y_train)

# Evaluate the model using created function
model_score(forest_reg)
```

Out[42]:
```
{'R^2': 0.3087360798891965,
 'MAE': 15950.66302000001,
 'MSE': 473237015.4708837,
 'RMSLE': 0.2996848748074827}
```

> We can see that the Random Forest Regressor worked better as the model was slightly improved. Firstly we will use a cross-validation rather than the train_test_split to compare our results, then we will try to tune our hyperparameters to further improve the model if possible.

In [43]:
```
# Use cross-validation of 5 folds
scores = cross_val_score(forest_reg,X,y,
                         scoring="neg_mean_absolute_error",
                         cv=10,
                         n_jobs=2,
                        verbose=True)

def display_score(scores):
    scores=-scores
    print("Scores:",scores)
    print("")
    print("Mean:",scores.mean())
    print("")
    print("Standard Deviation:",scores.std())
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  10 out of  10 | elapsed:    8.5s finished
```

In [44]:
```
display_score(scores)
```

```
Scores: [14579.45872 13915.93484 19749.18336 20982.96896 12429.85456 16253.3198
 21352.55904 15543.18956 15194.40132 21354.10464]

Mean: 17135.497479999998

Standard Deviation: 3214.625554237405
```

In [45]:
```
"""sklearn.metrics.get_scorer_names()
"""
```

Out[45]:
```
'sklearn.metrics.get_scorer_names()\n'
```

> Not much difference is achieved using `cross_val_score` so we will go ahead to tune the hyperparameter using GridSearchCV and RandomizedSearchCV

In [46]:
```
# Import the scikit-learn library for hyperparameter tuning
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
```

In [47]:
```
# Select the test hyperparameters for RandomizedSearchCV
param_grid= {'bootstrap':[False],
             'n_estimators':np.arange(1,50,5),
            'max_depth':[None,2,4,10],
            'min_samples_split':np.arange(2,40,5),
            'min_samples_leaf':np.arange(1,20,2),
            'max_features':[2,5,1,'sqrt'],
            }

# Instanstiate the model
rand_search =RandomizedSearchCV(RandomForestRegressor(n_jobs=-1,
                                                      random_state=42),
                          param_distributions=param_grid,
                         n_iter=5,
                         scoring ='neg_mean_absolute_error',
                         cv=10,
                         verbose=True,
                         return_train_score=True
                               )

# Fit the model with the hypeparamter grid
rand_search.fit(X_train,y_train)
```

```
Fitting 10 folds for each of 5 candidates, totalling 50 fits
```

Out[47]:
```
RandomizedSearchCV
estimator: RandomForestRegressor
RandomForestRegressor
```

In [48]:
```
rand_search.best_params_
```

Out[48]:
```
{'n_estimators': 21,
 'min_samples_split': 12,
 'min_samples_leaf': 13,
 'max_features': 2,
 'max_depth': 10,
 'bootstrap': False}
```

In [49]:
```
model_score(rand_search)
```

Out[49]:
```
{'R^2': 0.28705129027089293,
 'MAE': 16977.520107145738,
 'MSE': 488082351.41498315,
 'RMSLE': 0.3303166527664679}
```

In [50]:
```
# save our model
# Import the joblib library to save the model
import joblib
joblib.dump(rand_search,'rand_forest_randcv_model.pkl')
```

Out[50]:
```
['rand_forest_randcv_model.pkl']
```

In [51]:
```
# Select the test hyperparameters for GridSearchCV
param_grid= {'bootstrap':[False],
             'n_estimators':np.arange(10,50,10),
            'max_depth':[None,2,4,10],
```

```
             'min_samples_split':np.arange(2,20,5),
             'min_samples_leaf':np.arange(1,20,5),
             'max_features':[2,5,1,'sqrt'],
             }

# Instanstiate the model
grid_search =GridSearchCV(RandomForestRegressor(n_jobs=-1,
                                                 random_state=42),
                          param_grid=param_grid,
                         scoring ='neg_mean_absolute_error',
                         cv=5,
                         verbose=True,
                         return_train_score=True
                               )

# Fit the model with the hypeparamter grid
grid_search.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 1024 candidates, totalling 5120 fits
```

Out[51]:

```
GridSearchCV
estimator: RandomForestRegressor
RandomForestRegressor
```

In [52]:

```
grid_search.best_params_
```

Out[52]:

```
{'bootstrap': False,
 'max_depth': 4,
 'max_features': 2,
 'min_samples_leaf': 6,
 'min_samples_split': 17,
 'n_estimators': 40}
```

In [53]:

```
model_score(grid_search)
```

Out[53]:

```
{'R^2': 0.35674515568432286,
 'MAE': 15726.009219342455,
 'MSE': 440370159.43540704,
 'RMSLE': 0.30957652029535815}
```

In [54]:

```
joblib.dump(rand_search,'rand_forest_gridcv_model.pkl')
```

Out[54]:

```
['rand_forest_gridcv_model.pkl']
```

> While we have an improved model in the RandomForest estimator compared to the linear regression model, the performance of the model is still not satisfactory. While this maybe due to the fact that that the amount of data available is not much for the model to train on, we will experiment further by trying one more decision tree model- `XGBoost Regressor`.

-XGBoost -XGBRegressor()

In [60]:

```
# We will use the XGBoost Regression model as our test further
# Import the XGBoost Regressor
from xgboost import XGBRegressor
```

In [62]:

```
# Instanstiate the XGBRegressor
```

```
xgb_model = XGBRegressor(objective='reg:squarederror', n_estimators=1000)

# Fit the model to the training data
xgb_model.fit(X_train, y_train)
```

Out[62]:

```
XGBRegressor
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categoric
al=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.300000012,
             max_delta_step=0, max_depth=6, min_child_weight=1, missi
ng=nan,
             monotone_constraints='()', n_estimators=1000, n_jobs=4,
             num_parallel_tree=1, predictor='auto', random_state=0, r
eg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_meth
```

In [63]:

```
model_score(xgb_model)
```

Out[63]:

```
{'R^2': 0.026827794055153786,
 'MAE': 17472.483453125,
 'MSE': 666230504.5614626,
 'RMSLE': 0.33803779759471375}
```

The performance of the XGBoost regression model is not any better than earlier models. To close we will try to tune the hyperparameters using RandomizedSearchCV

In [67]:

```
xgb_params = {'n_estimators' : np.arange(10,1000,100),'max_depth' : [2,3,5],'learn
                'booster': ['gbtree','dart'],'min_child_weight' : [2,4,6]}


xgb_rand_search =RandomizedSearchCV(xgb_model,param_distributions=xgb_params,
                               n_iter=2,
                                   cv=5,
                                   random_state=42,
                                   scoring='neg_mean_absolute_error')


xgb_rand_search.fit(X_train,y_train)
```

```
C:\Users\USER\OneDrive\Documents\PERSONAL\ANZ Internship\env\lib\site-packages\xgb
oost\data.py:250: FutureWarning:

pandas.Int64Index is deprecated and will be removed from pandas in a future versio
n. Use pandas.Index with the appropriate dtype instead.

C:\Users\USER\OneDrive\Documents\PERSONAL\ANZ Internship\env\lib\site-packages\xgb
oost\data.py:250: FutureWarning:

pandas.Int64Index is deprecated and will be removed from pandas in a future versio
n. Use pandas.Index with the appropriate dtype instead.

C:\Users\USER\OneDrive\Documents\PERSONAL\ANZ Internship\env\lib\site-packages\xgb
oost\data.py:250: FutureWarning:

pandas.Int64Index is deprecated and will be removed from pandas in a future versio
n. Use pandas.Index with the appropriate dtype instead.

C:\Users\USER\OneDrive\Documents\PERSONAL\ANZ Internship\env\lib\site-packages\xgb
oost\data.py:250: FutureWarning:

pandas.Int64Index is deprecated and will be removed from pandas in a future versio
n. Use pandas.Index with the appropriate dtype instead.

C:\Users\USER\OneDrive\Documents\PERSONAL\ANZ Internship\env\lib\site-packages\xgb
oost\data.py:250: FutureWarning:

pandas.Int64Index is deprecated and will be removed from pandas in a future versio
n. Use pandas.Index with the appropriate dtype instead.

C:\Users\USER\OneDrive\Documents\PERSONAL\ANZ Internship\env\lib\site-packages\xgb
oost\data.py:250: FutureWarning:

pandas.Int64Index is deprecated and will be removed from pandas in a future versio
n. Use pandas.Index with the appropriate dtype instead.

C:\Users\USER\OneDrive\Documents\PERSONAL\ANZ Internship\env\lib\site-packages\xgb
oost\data.py:250: FutureWarning:

pandas.Int64Index is deprecated and will be removed from pandas in a future versio
n. Use pandas.Index with the appropriate dtype instead.

C:\Users\USER\OneDrive\Documents\PERSONAL\ANZ Internship\env\lib\site-packages\xgb
oost\data.py:250: FutureWarning:

pandas.Int64Index is deprecated and will be removed from pandas in a future versio
n. Use pandas.Index with the appropriate dtype instead.

C:\Users\USER\OneDrive\Documents\PERSONAL\ANZ Internship\env\lib\site-packages\xgb
oost\data.py:250: FutureWarning:

pandas.Int64Index is deprecated and will be removed from pandas in a future versio
n. Use pandas.Index with the appropriate dtype instead.

C:\Users\USER\OneDrive\Documents\PERSONAL\ANZ Internship\env\lib\site-packages\xgb
oost\data.py:250: FutureWarning:

pandas.Int64Index is deprecated and will be removed from pandas in a future versio
n. Use pandas.Index with the appropriate dtype instead.

C:\Users\USER\OneDrive\Documents\PERSONAL\ANZ Internship\env\lib\site-packages\xgb
oost\data.py:250: FutureWarning:

pandas.Int64Index is deprecated and will be removed from pandas in a future versio
```

```
n. Use pandas.Index with the appropriate dtype instead.
```

Out[67]:

```
▸ RandomizedSearchCV
▸ estimator: XGBRegressor
  ▸ XGBRegressor
```

In [68]:

```
xgb_rand_search.best_params_
```

Out[68]:

```
{'objective': 'reg:squarederror',
 'n_estimators': 510,
 'min_child_weight': 4,
 'max_depth': 5,
 'learning_rate': 0.2,
 'booster': 'dart'}
```

In [69]:

```
model_score(xgb_rand_search)
```

Out[69]:

```
{'R^2': -0.010919795690088385,
 'MAE': 18469.6953203125,
 'MSE': 692072380.8587159,
 'RMSLE': 0.35112224900013633}
```

In [ ]: