In [55]:

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

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler, LabelEncoder

import warnings
warnings.simplefilter(action='ignore')
pd.set_option('display.max_columns', 150)
```

In [50]:

```
#read and load the data set
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
sample = pd.read_csv('sample_submission.csv')
train.head()
```

Out[50]:

	Customer_ID	months_as_customer	age	insured_sex	insured_education_level	insured_occup
0	Customer_541	239	41	FEMALE	JD	farming-f
1	Customer_440	108	31	MALE	Masters	protectiv
2	Customer_482	116	30	MALE	JD	handlers-cle
3	Customer_422	8	21	MALE	High School	handlers-cle
4	Customer_778	161	38	MALE	PhD	priv-hous

In [53]:

```
#grouping the features into categorical and numerical features
categorical_feat = {feat for feat in train.columns if train[feat].dtypes == 'O'}
num_feat = {feat for feat in train.columns if feat not in categorical_feat}
```

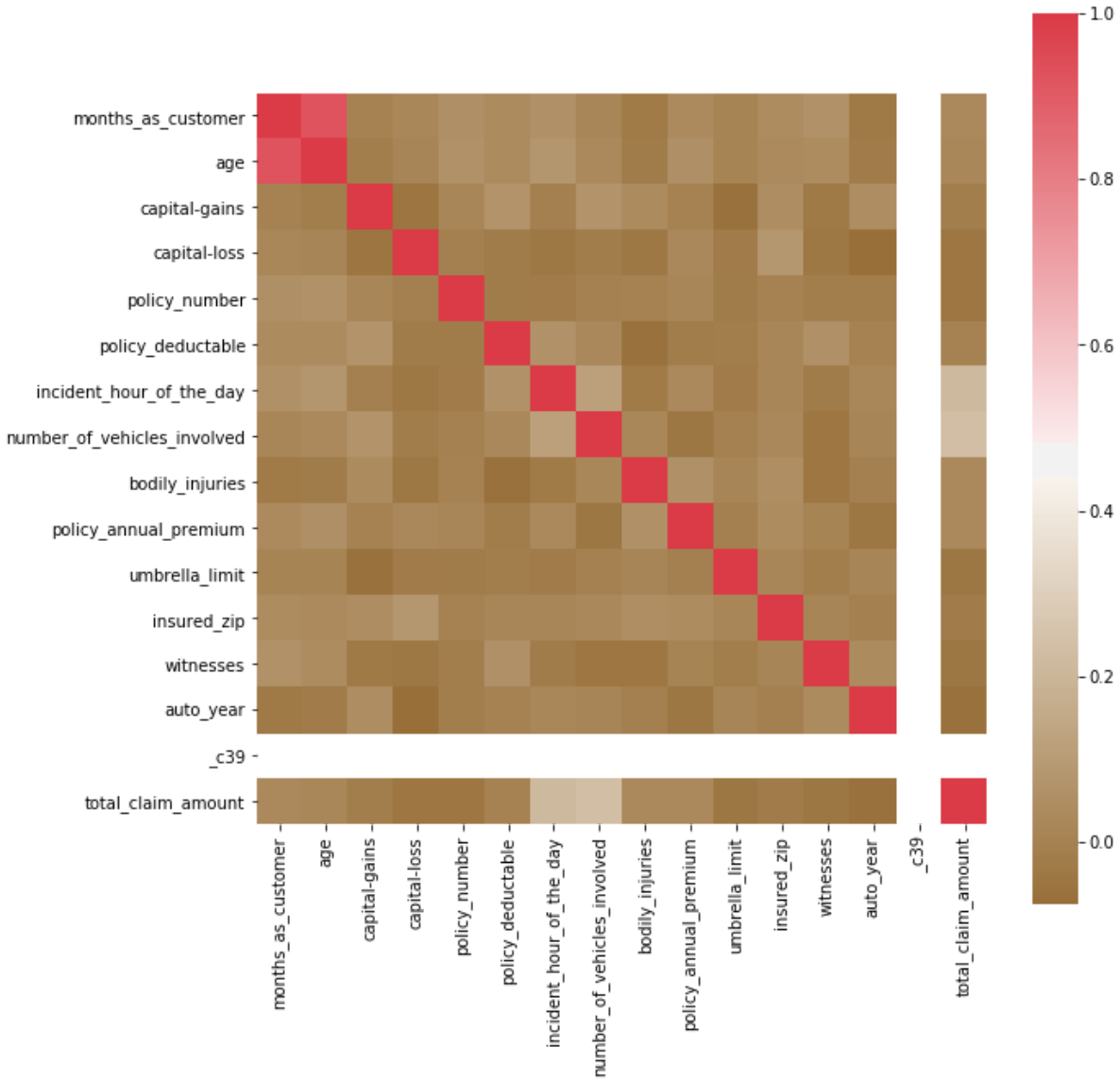
In [56]:

```
# plotting the correlation plot for the dataset

f, ax = plt.subplots(figsize = (10, 10))
corr = train.corr()
sns.heatmap(corr, mask = np.zeros_like(corr, dtype = np.bool),
            cmap = sns.diverging_palette(50, 10, as_cmap = True), square = True, ax = ax)
```

Out[56]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x27636ba2630>
```



In [57]:

```
# importing label encoder
encoder = LabelEncoder()
```

In [58]:

```
# label encoding for train dataset

train_encoded = pd.DataFrame(data=categorical_feat)
train_encoded = train[categorical_feat].apply(LabelEncoder().fit_transform)

train_encoded.head()
```

Out[58]:

	insured_hobbies	insured_sex	collision_type	insured_occupation	policy_bind_date	auto_mak
0	13	0	0	4	639	
1	19	1	2	10	444	
2	9	1	1	5	334	1
3	10	1	1	5	610	1
4	8	1	1	8	392	1

In [59]:

```
# label encoding for test dataset

test_encoded = pd.DataFrame(data=test_categorical_feat)
test_encoded = test[categorical_feat].apply(LabelEncoder().fit_transform)

test_encoded.head()
```

Out[59]:

	insured_hobbies	insured_sex	collision_type	insured_occupation	policy_bind_date	auto_mak
0	16	0	2	4	288	
1	8	0	1	3	224	
2	15	0	2	2	10	1
3	17	1	1	0	43	
4	18	0	0	6	156	

In [62]:

```
#merging the encoded categorical variables with the numerical variables
train_final = pd.merge(train[num_feat], train_encoded, on=train[num_feat].index, how='outer')
test_final = pd.merge(test[test_num_feat], test_encoded, on=test[test_num_feat].index, how='outer')
test_final.head()
```

Out[62]:

	key_0	number_of_vehicles_involved	capital-gains	policy_deductable	capital-loss	witnesses	auto_yea
0	0	1	31500	2000	0	3	200
1	1	3	61600	1000	0	3	200
2	2	3	0	1000	-61000	3	200
3	3	3	42700	2000	-64900	0	201
4	4	1	0	1000	-29900	0	200

In [63]:

```
test_final.shape
```

Out[63]:

```
(300, 37)
```

In [64]:

```
test_final.columns
```

Out[64]:

```
Index(['key_0', 'number_of_vehicles_involved', 'capital-gains',
       'policy_deductable', 'capital-loss', 'witnesses', 'auto_year',
       'bodily_injuries', 'age', 'months_as_customer', '_c39',
       'incident_hour_of_the_day', 'policy_annual_premium', 'umbrella_limit',
       'insured_zip', 'policy_number', 'insured_hobbies', 'insured_sex',
       'collision_type', 'insured_occupation', 'policy_bind_date', 'auto_mak
e',
       'incident_date', 'auto_model', 'insured_education_level',
       'incident_city', 'incident_location', 'property_damage',
       'incident_severity', 'incident_type', 'police_report_available',
       'incident_state', 'authorities_contacted', 'policy_csl', 'policy_stat
e',
       'insured_relationship', 'Customer_ID'],
      dtype='object')
```

In [66]:

```
# Remove the index column and also column with NAN values (it is not useful for modeling)
train_final.drop(['key_0', 'Customer_ID', '_c39' ], axis=1, inplace=True)
train_final.shape
```

Out[66]:

```
(700, 35)
```

In [65]:

```
# Remove the index column and also column with NAN values (it is not useful for modeling)
test_final.drop(['key_0', 'Customer_ID', '_c39' ], axis=1, inplace=True)
test_final.shape
```

Out[65]:

```
(300, 34)
```

In [18]:

```
#checking the columns
test_final.columns
```

Out[18]:

```
Index(['capital-loss', 'number_of_vehicles_involved', 'insured_zip', 'age',
       'incident_hour_of_the_day', 'umbrella_limit', 'auto_year',
       'capital-gains', 'witnesses', 'policy_deductable', 'policy_number',
       'bodily_injuries', 'months_as_customer', 'policy_annual_premium',
       'insured_hobbies', 'police_report_available', 'insured_sex',
       'insured_occupation', 'incident_state', 'policy_csl', 'auto_make',
       'incident_city', 'incident_location', 'incident_severity',
       'insured_education_level', 'insured_relationship', 'incident_type',
       'property_damage', 'auto_model', 'authorities_contacted',
       'collision_type', 'incident_date', 'policy_state', 'policy_bind_dat
e'],
      dtype='object')
```

In [67]:

```
#checking the columns
train_final.columns
```

Out[67]:

```
Index(['total_claim_amount', 'number_of_vehicles_involved', 'capital-gains',
       'policy_deductable', 'capital-loss', 'witnesses', 'auto_year',
       'bodily_injuries', 'age', 'months_as_customer',
       'incident_hour_of_the_day', 'policy_annual_premium', 'umbrella_limit',
       'insured_zip', 'policy_number', 'insured_hobbies', 'insured_sex',
       'collision_type', 'insured_occupation', 'policy_bind_date', 'auto_mak
e',
       'incident_date', 'auto_model', 'insured_education_level',
       'incident_city', 'incident_location', 'property_damage',
       'incident_severity', 'incident_type', 'police_report_available',
       'incident_state', 'authorities_contacted', 'policy_csl', 'policy_stat
e',
       'insured_relationship'],
      dtype='object')
```

In [16]:

```
#checking for null values
print(train_final.isnull().values.any())
```

```
False
```

In [17]:

```
#checking for null values
print(test_final.isnull().values.any())
```

```
False
```

In [68]:

```
# splitting the dependent and independent variable
X = train_final.loc[:, train_final.columns != 'total_claim_amount']
y = train_final['total_claim_amount']

print(X.shape)
print(y.shape)
```

```
(700, 34)
(700,)
```

In [42]:

```
#Obtain training and validation sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
30)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(560, 34)
(140, 34)
(560,)
(140,)
```

In [43]:

```
# standard scaling

from sklearn.preprocessing import StandardScaler

# creating a standard scaler
sc = StandardScaler()

# feeding independents sets into the standard scaler
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

In [22]:

```
# Linear Regression modelling
from sklearn.linear_model import LinearRegression

# creating the Linear Regression model
lR = LinearRegression()

# # feeding the training data to the model
lR.fit(X_train, y_train)

y_pred_lR = lR.predict(X_test)

lR.score(X_train,y_train)
```

Out[22]:

```
0.43043629244369813
```

In [23]:

```
#RandomForestRegressor modelling
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 10, random_state = 0)
regressor.fit(X_train, y_train)

# Predicting a new result
y_pred = regressor.predict(X_test)
regressor.score(X_train,y_train)
```

Out[23]:

```
0.9318726859292086
```

In [24]:

```
# feeding independents sets into the standard scaler
testing = sc.fit_transform(test_final)

zpred = regressor.predict(testing)
zpred.shape
```

Out[24]:

```
(300,)
```

In [235]:

```
#Create submission file
submission = pd.DataFrame({'Customer_ID': test['Customer_ID'],'total_claim_amount': zpred
})
print(submission.head())
```

```
    Customer_ID  total_claim_amount
0  Customer_521           79276.000
1  Customer_737           90176.001
2  Customer_740           73102.666
3  Customer_660           82742.667
4  Customer_411            7825.335
```

In [236]:

```
submission.shape
```

Out[236]:

```
(300, 2)
```

In [238]:

```
submission.to_csv('Ndidi_submission', index=False)
```

In [32]:

```
#DecisionTreeRegressor modelling
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score


# creating the model
model = DecisionTreeRegressor()

# feeding the training data to the model
model.fit(X_train, y_train)

# predicting the test set results
y_pred2 = model.predict(X_test)

print(model.score(X_train, y_train))
print(model.score(X_test, y_test))
```

```
1.0
0.43433972770243023
```

In [44]:

```
# feeding independents sets into the standard scaler
testing = sc.fit_transform(test_final)

y_pred2 = regressor.predict(testing)
y_pred2.shape
```

Out[44]:

```
(300,)
```

In [45]:

```
#Create submission file
submission2 = pd.DataFrame({'Customer_ID': test['Customer_ID'],'total_claim_amount': y_pre
d2})
print(submission2.head())
```

```
    Customer_ID  total_claim_amount
0  Customer_521           81062.667
1  Customer_737           84318.666
2  Customer_740           77394.667
3  Customer_660           79838.667
4  Customer_411            7417.335
```

In [46]:

```
submission2.to_csv('Ndidi_submission2', index=False)
```

In [ ]:

```

```