Vehicle Insurance Prediction

In [59]:

```
import pandas as pd
from hdfs import InsecureClient
import os
from pyspark.sql import SparkSession
import matplotlib.pyplot as plt
import seaborn as sns
```

In [4]:

```
sparkSession = SparkSession.builder.appName("EDA Vehicle Insurance").getOrCrea
te()
client_hdfs = InsecureClient('hdfs://localhost:9820')
```

In [5]:
```python
df = sparkSession.read.csv('hdfs://localhost:9820/test/Merge.csv', header=True
, inferSchema= True)
df.show()
```

```
+---+------+---+---------------+-----------+------------------+-----------+--
------------+--------------+--------------------+-------+--------+
| id|Gender|Age|Driving_License|Region_Code|Previously_Insured|Vehicle_Age|Ve
hicle_Damage|Annual_Premium|Policy_Sales_Channel|Vintage|Response|
+---+------+---+---------------+-----------+------------------+-----------+--
------------+--------------+--------------------+-------+--------+
|  1|  Male| 44|              1|         28|                 0|  > 2 Years|
Yes|         40454|                  26|    217|       1|
|  2|  Male| 76|              1|          3|                 0|   1-2 Year|
No|         33536|                  26|    183|       0|
|  3|  Male| 47|              1|         28|                 0|  > 2 Years|
Yes|         38294|                  26|     27|       1|
|  4|  Male| 21|              1|         11|                 1|   < 1 Year|
No|         28619|                 152|    203|       0|
|  5|Female| 29|              1|         41|                 1|   < 1 Year|
No|         27496|                 152|     39|       0|
|  6|Female| 24|              1|         33|                 0|   < 1 Year|
Yes|          2630|                 160|    176|       0|
|  7|  Male| 23|              1|         11|                 0|   < 1 Year|
Yes|         23367|                 152|    249|       0|
|  8|Female| 56|              1|         28|                 0|   1-2 Year|
Yes|         32031|                  26|     72|       1|
|  9|Female| 24|              1|          3|                 1|   < 1 Year|
No|         27619|                 152|     28|       0|
| 10|Female| 32|              1|          6|                 1|   < 1 Year|
No|         28771|                 152|     80|       0|
| 11|Female| 47|              1|         35|                 0|   1-2 Year|
Yes|         47576|                 124|     46|       1|
| 12|Female| 24|              1|         50|                 1|   < 1 Year|
No|         48699|                 152|    289|       0|
| 13|Female| 41|              1|         15|                 1|   1-2 Year|
No|         31409|                  14|    221|       0|
| 14|  Male| 76|              1|         28|                 0|   1-2 Year|
Yes|         36770|                  13|     15|       0|
| 15|  Male| 71|              1|         28|                 1|   1-2 Year|
No|         46818|                  30|     58|       0|
| 16|  Male| 37|              1|          6|                 0|   1-2 Year|
Yes|          2630|                 156|    147|       1|
| 17|Female| 25|              1|         45|                 0|   < 1 Year|
Yes|         26218|                 160|    256|       0|
| 18|Female| 25|              1|         35|                 1|   < 1 Year|
No|         46622|                 152|    299|       0|
| 19|  Male| 42|              1|         28|                 0|   1-2 Year|
Yes|         33667|                 124|    158|       0|
| 20|Female| 60|              1|         33|                 0|   1-2 Year|
Yes|         32363|                 124|    102|       1|
+---+------+---+---------------+-----------+------------------+-----------+--
------------+--------------+--------------------+-------+--------+
only showing top 20 rows
```

In [7]:
```
result_df = df.select("*").toPandas()
result_df.head()
```

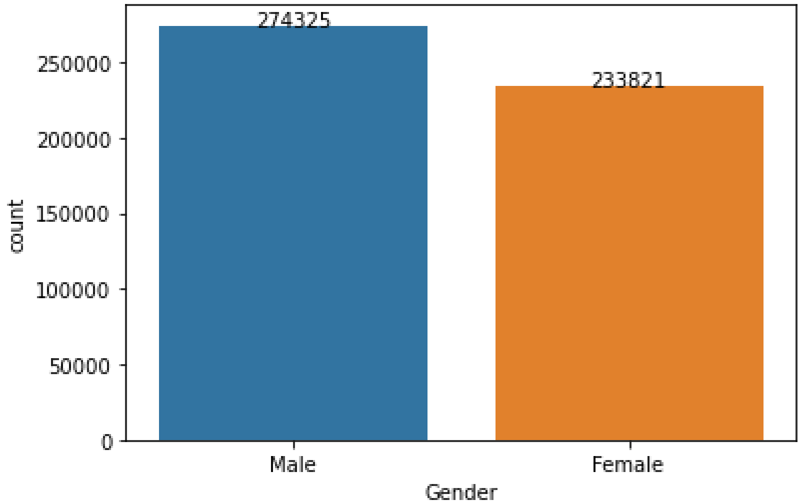
Out[7]:

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Dai
0	1	Male	44	1	28	0	> 2 Years	
1	2	Male	76	1	3	0	1-2 Year	
2	3	Male	47	1	28	0	> 2 Years	
3	4	Male	21	1	11	1	< 1 Year	
4	5	Female	29	1	41	1	< 1 Year	

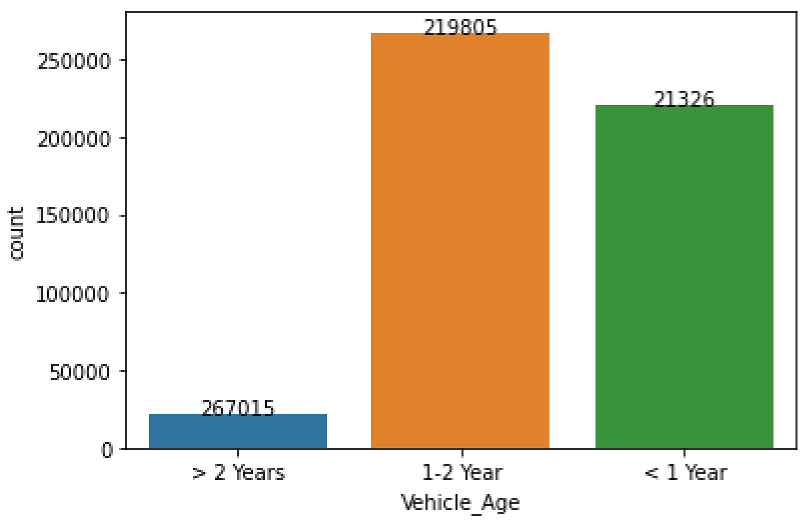
In [5]:
```
print(result_df.dtypes)
print(result_df.isnull().any())
print(result_df.isnull().sum())
```

```
id                        int32
Gender                   object
Age                       int32
Driving_License           int32
Region_Code               int32
Previously_Insured        int32
Vehicle_Age              object
Vehicle_Damage           object
Annual_Premium            int32
Policy_Sales_Channel      int32
Vintage                   int32
Response                  int32
dtype: object
id                       False
Gender                   False
Age                      False
Driving_License          False
Region_Code              False
Previously_Insured       False
Vehicle_Age              False
Vehicle_Damage           False
Annual_Premium           False
Policy_Sales_Channel     False
Vintage                  False
Response                 False
dtype: bool
id                       0
Gender                   0
Age                      0
Driving_License          0
Region_Code              0
Previously_Insured       0
Vehicle_Age              0
Vehicle_Damage           0
Annual_Premium           0
Policy_Sales_Channel     0
Vintage                  0
Response                 0
dtype: int64
```

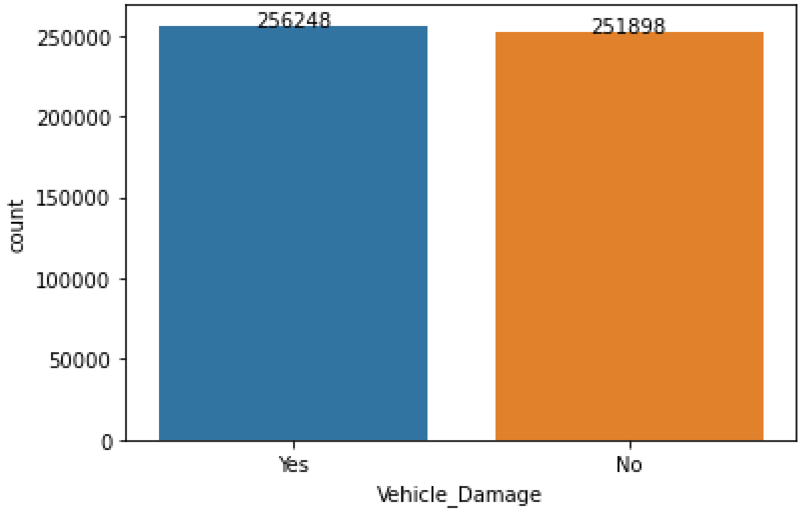
In [6]:
```
graph = sns.countplot(result_df.Gender)
graph.set_xticklabels(graph.get_xticklabels())
i=0
for p in graph.patches:
    height = p.get_height()
    graph.text(p.get_x()+p.get_width()/2., height + 0.1,
        result_df['Gender'].value_counts()[i],ha="center")
    i += 1
```



In [7]:
```
graph = sns.countplot(result_df.Vehicle_Age)
graph.set_xticklabels(graph.get_xticklabels())
i=0
for p in graph.patches:
    height = p.get_height()
    graph.text(p.get_x()+p.get_width()/2., height + 0.1,
        result_df['Vehicle_Age'].value_counts()[i],ha="center")
    i += 1
```

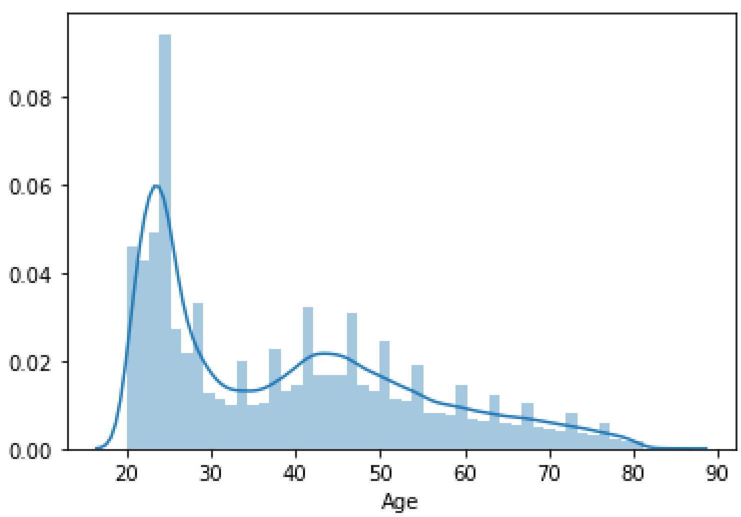


In [8]:
```
graph = sns.countplot(result_df.Vehicle_Damage)
graph.set_xticklabels(graph.get_xticklabels())
i=0
for p in graph.patches:
    height = p.get_height()
    graph.text(p.get_x()+p.get_width()/2., height + 0.1,
        result_df['Vehicle_Damage'].value_counts()[i],ha="center")
    i += 1
```



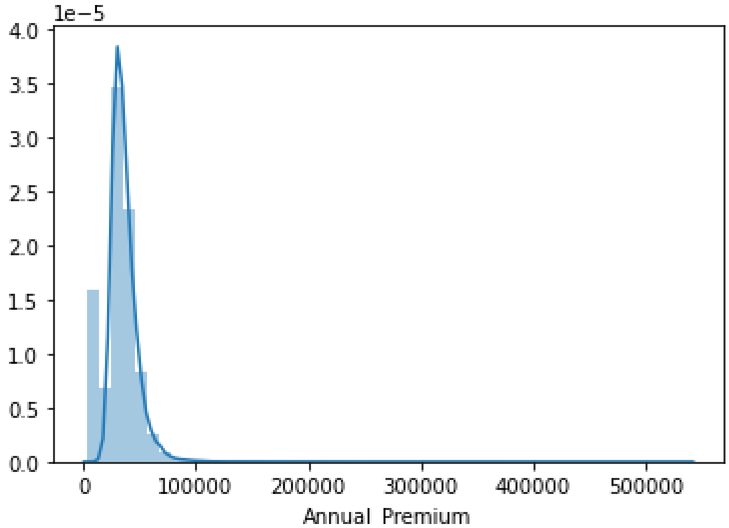
In [9]:
```
sns.distplot(result_df.Age)
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1e61a0c7730>



In [10]:
```
sns.distplot(result_df.Annual_Premium)
```

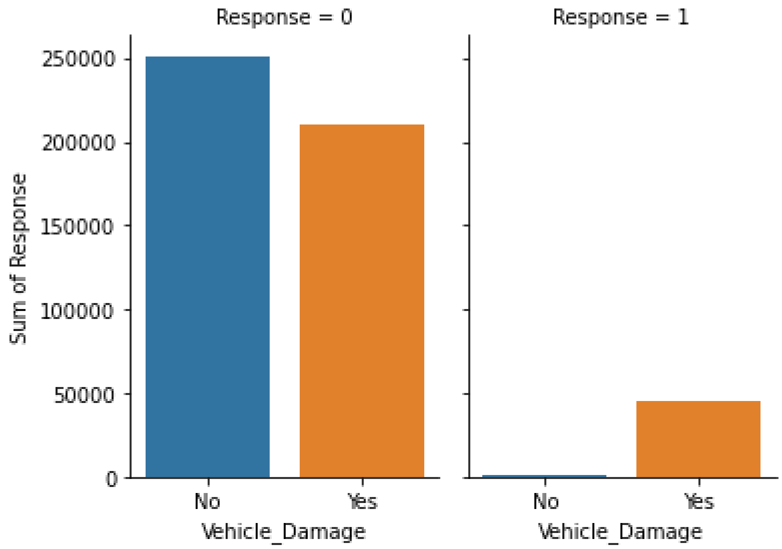
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x1e61465ce20>



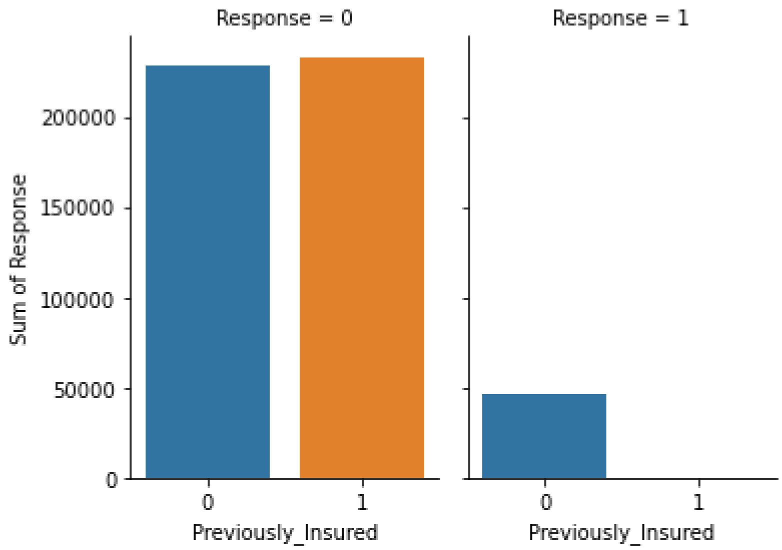
In [11]:
```
a=result_df.groupby(['Vehicle_Damage','Response']).size().sort_values(ascendin
g=False).reset_index(name='Sum of Response')
b=result_df.groupby(['Previously_Insured','Response']).size().sort_values(asce
nding=False).reset_index(name='Sum of Response')
c=result_df.groupby(['Gender','Response']).size().sort_values(ascending=False)
.reset_index(name='Sum of Response')
print(a)
print(b)
print(c)
```

```
  Vehicle_Damage  Response  Sum of Response
0             No         0           250916
1            Yes         0           210520
2            Yes         1            45728
3             No         1              982
   Previously_Insured  Response  Sum of Response
0                   1         0           232912
1                   0         0           228524
2                   0         1            46552
3                   1         1              158
   Gender  Response  Sum of Response
0    Male         0           245800
1  Female         0           215636
2    Male         1            28525
3  Female         1            18185
```

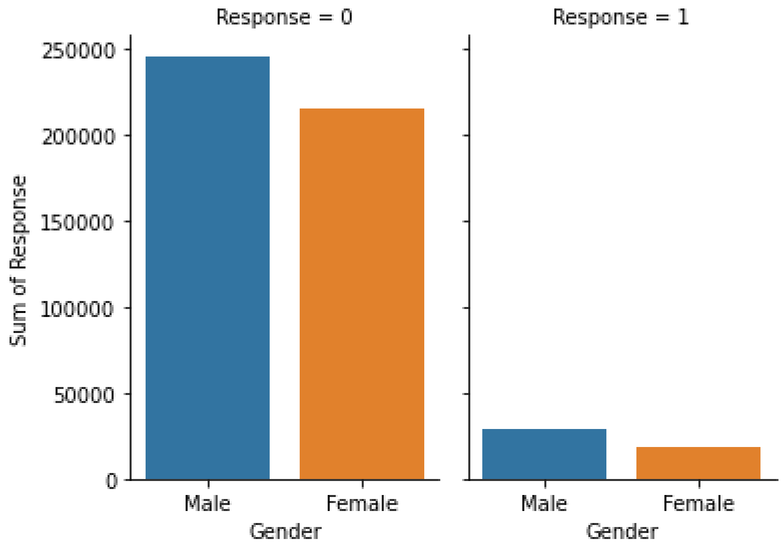
In [12]:
```
graph = sns.catplot(x="Vehicle_Damage", y="Sum of Response",col="Response",
                data=a, kind="bar",
                height=4, aspect=.7);
```



In [13]:
```
graph = sns.catplot(x="Previously_Insured", y="Sum of Response",col="Response"
,
                data=b, kind="bar",
                height=4, aspect=.7);
```



In [14]:
```
graph = sns.catplot(x="Gender", y="Sum of Response",col="Response",
                data=c, kind="bar",
                height=4, aspect=.7);
```



In [15]:
```
result_df.Vehicle_Age.unique()
```

Out[15]:
```
array(['> 2 Years', '1-2 Year', '< 1 Year'], dtype=object)
```

In [8]:
```
result_df.Gender[result_df.Gender == 'Male'] = 1
result_df.Gender[result_df.Gender == 'Female'] = 0

result_df.Vehicle_Damage[result_df.Vehicle_Damage == 'Yes'] = 1
result_df.Vehicle_Damage[result_df.Vehicle_Damage == 'No'] = 0

result_df['Gender'] = result_df['Gender'].astype(int)
result_df['Vehicle_Damage'] = result_df['Vehicle_Damage'].astype(int)
```

```
<ipython-input-8-8f978a608a47>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  result_df.Gender[result_df.Gender == 'Male'] = 1
<ipython-input-8-8f978a608a47>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  result_df.Gender[result_df.Gender == 'Female'] = 0
<ipython-input-8-8f978a608a47>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  result_df.Vehicle_Damage[result_df.Vehicle_Damage == 'Yes'] = 1
<ipython-input-8-8f978a608a47>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  result_df.Vehicle_Damage[result_df.Vehicle_Damage == 'No'] = 0
```

In [17]:
```
result_df.head()
```

Out[17]:

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Da
0	1	1	44	1	28	0	> 2 Years	
1	2	1	76	1	3	0	1-2 Year	
2	3	1	47	1	28	0	> 2 Years	
3	4	1	21	1	11	1	< 1 Year	
4	5	0	29	1	41	1	< 1 Year	

In [9]:
```
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
result_df['Vehicle_Age'] = labelencoder.fit_transform(result_df['Vehicle_Age'
])
```

In [19]:
```
result_df.head()
```

Out[19]:

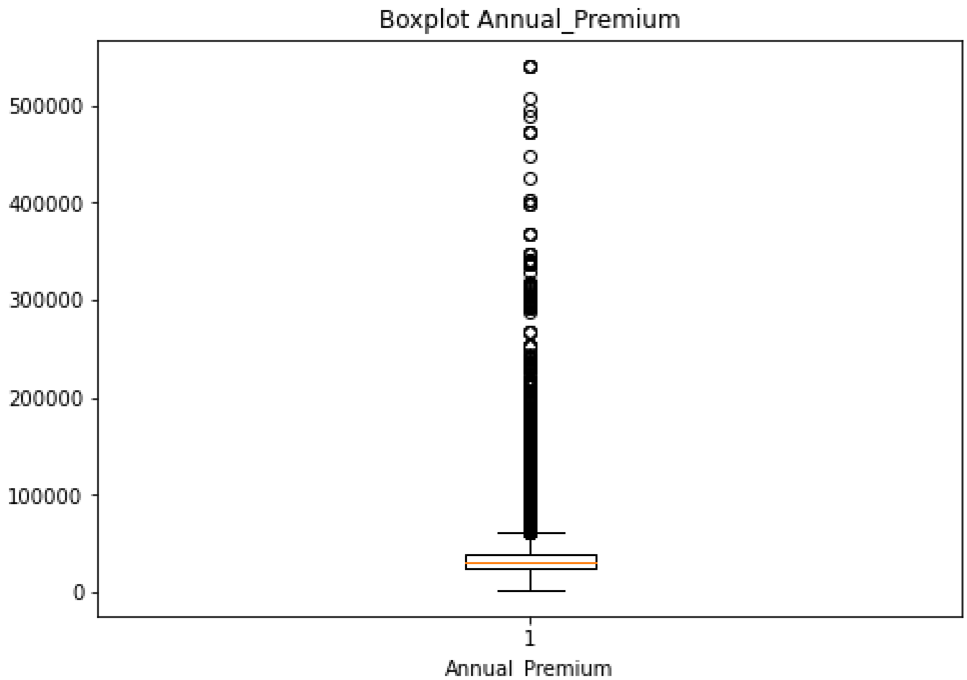
	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Dai
0	1	1	44	1	28	0	2	
1	2	1	76	1	3	0	0	
2	3	1	47	1	28	0	2	
3	4	1	21	1	11	1	1	
4	5	0	29	1	41	1	1	

In [20]:
```
numerical_columns=['Age', 'Region_Code','Annual_Premium','Vintage']
result_df[numerical_columns].describe()
```

Out[20]:

	Age	Region_Code	Annual_Premium	Vintage
count	508146.000000	508146.000000	508146.000000	508146.000000
mean	38.808413	26.406572	30554.453041	154.340123
std	15.500179	13.224921	17146.574625	83.668793
min	20.000000	0.000000	2630.000000	10.000000
25%	25.000000	15.000000	24381.000000	82.000000
50%	36.000000	28.000000	31661.000000	154.000000
75%	49.000000	35.000000	39403.750000	227.000000
max	85.000000	52.000000	540165.000000	299.000000

In [21]:
```
import matplotlib.pyplot as plt
fig = plt.figure()
# Create an axes instance
ax = fig.add_axes([0,0,1,1])
# Create the boxplot
bp = ax.boxplot(result_df['Annual_Premium'])
plt.xlabel("Annual_Premium")
plt.title("Boxplot Annual_Premium")
plt.show()
```



In [22]:
```
import numpy as np
outliers=[]
def detect_outlier(data_1):

    threshold=3
    mean_1 = np.mean(data_1)
    std_1 =np.std(data_1)


    for y in data_1:
        z_score= (y - mean_1)/std_1
        if np.abs(z_score) > threshold:
            outliers.append(y)
    return outliers
```

In [23]:
```
outlier_datapoints = detect_outlier(result_df["Annual_Premium"])
print(outlier_datapoints)
```

```
[89282, 101021, 82705, 90789, 119735, 104002, 92716, 112974, 139130, 98002, 2
67698, 125643, 85786, 95217, 101064, 86283, 136061, 117799, 84142, 91520, 872
73, 133098, 103026, 87831, 152331, 100688, 101069, 87954, 131469, 104529, 863
36, 82081, 90024, 98428, 107966, 89687, 137771, 82470, 88368, 88577, 90967, 9
1670, 83846, 99793, 101048, 90046, 95554, 88825, 82932, 123745, 103372, 8494
3, 103758, 101716, 508073, 89480, 88129, 101904, 94900, 90526, 83368, 82085,
93104, 84472, 82144, 141770, 95569, 301762, 120037, 119148, 84077, 95258, 110
204, 89463, 315565, 93104, 85772, 89902, 85670, 126671, 107748, 92211, 87105,
86793, 83912, 107266, 84056, 89637, 124345, 98425, 82879, 91565, 113820, 8935
5, 104781, 91407, 95895, 89888, 183718, 98526, 88378, 86983, 89884, 86606, 10
6578, 116045, 86416, 95598, 94647, 82422, 110973, 99324, 94506, 93656, 87578,
95221, 109361, 86885, 92190, 95817, 84122, 83433, 103906, 87763, 98337, 8287
6, 181076, 111257, 160380, 159869, 229935, 94109, 88281, 83137, 86646, 99999,
84475, 90305, 94580, 140448, 113810, 82141, 82231, 82067, 82420, 87128, 9897
9, 92575, 99851, 87307, 89687, 147075, 98152, 94333, 98073, 83196, 84432, 101
568, 100018, 168597, 82135, 119425, 97229, 83778, 101024, 85327, 83666, 8671
1, 91713, 155317, 83045, 90374, 99529, 160011, 151585, 113332, 199996, 96639,
111070, 104511, 113339, 105335, 83951, 294209, 83017, 104998, 86750, 82125, 9
4216, 90834, 107454, 85608, 86415, 89440, 336395, 95824, 83037, 82753, 89778,
95992, 111447, 95641, 214455, 82701, 89388, 95692, 88883, 87449, 96374, 9359
7, 86349, 86484, 97164, 90040, 86897, 85198, 89721, 111327, 101935, 88891, 88
746, 84472, 88038, 97467, 109546, 94843, 82128, 253362, 141131, 85340, 99873,
83203, 82424, 91146, 117793, 135054, 82089, 98327, 82033, 98152, 92928, 9793
6, 113379, 102708, 102724, 88196, 99096, 84917, 116398, 89263, 88123, 95664,
99198, 93267, 84918, 91694, 90534, 211132, 174574, 89022, 83652, 89425, 10112
3, 296891, 106843, 83957, 90972, 117563, 97505, 83338, 128329, 113088, 10711
5, 91151, 122700, 189361, 235683, 98067, 105627, 87665, 92716, 101331, 54016
5, 104003, 92518, 103511, 169127, 98650, 97693, 121384, 89063, 91415, 85063,
88638, 92860, 103968, 82129, 83808, 85553, 127493, 86860, 99950, 101025, 1001
96, 86958, 84875, 95217, 93259, 340439, 82501, 105542, 97772, 123446, 87557,
95904, 99238, 101825, 89388, 100993, 229375, 83638, 88673, 113636, 84328, 129
892, 88134, 95692, 91118, 102962, 85000, 121448, 82372, 252141, 90834, 14231
3, 206906, 106505, 131355, 84268, 85207, 83958, 108034, 91338, 107971, 84025,
218552, 82491, 86205, 240318, 105714, 101333, 83228, 111486, 135616, 110070,
98905, 85480, 86426, 86581, 82237, 167393, 83876, 120406, 126410, 107103, 107
943, 99091, 84810, 92161, 167899, 142271, 113029, 152866, 91323, 93214, 10973
8, 94946, 84671, 291169, 104539, 150468, 143525, 86013, 99248, 91164, 117237,
100718, 133747, 83683, 126158, 102747, 113553, 85514, 101380, 104388, 94899,
240809, 103439, 88128, 99339, 91487, 83182, 103059, 101639, 93971, 109080, 21
4455, 104195, 182288, 85990, 94343, 171264, 96436, 88946, 88533, 92454, 24173
5, 111384, 107115, 188591, 318706, 114021, 82195, 85683, 93902, 84046, 95274,
96364, 88388, 82649, 84112, 124520, 82204, 88369, 267698, 93195, 98418, 13190
1, 99647, 87798, 86112, 164502, 131148, 90959, 106829, 91882, 103417, 83076,
93569, 336395, 88908, 83646, 127942, 88411, 91886, 87601, 83817, 83249, 8296
8, 89059, 84025, 100985, 251853, 94593, 100377, 110308, 82788, 118594, 83554,
87133, 110498, 199154, 92267, 135229, 92300, 82826, 106909, 87675, 87130, 101
771, 85787, 110410, 92064, 92976, 112634, 99445, 84456, 313854, 109161, 11068
6, 82613, 84008, 88518, 94770, 88690, 82753, 102867, 84945, 88183, 121760, 16
5501, 94391, 93902, 88664, 134896, 286666, 88434, 98794, 159995, 111419, 8324
9, 86047, 95840, 90446, 90940, 87263, 94843, 97533, 83826, 309867, 84008, 845
14, 84046, 83027, 90600, 129706, 117544, 88811, 100278, 90335, 94035, 86343,
104229, 91094, 99866, 106392, 336395, 122795, 83979, 89643, 87386, 192034, 87
864, 84887, 84901, 93931, 199154, 101661, 303550, 100894, 86719, 87921, 11487
3, 99764, 118594, 102815, 82972, 108765, 100691, 93804, 90526, 84018, 97966,
102552, 95411, 96457, 85949, 125475, 87988, 82115, 102917, 95021, 89355, 9751
4, 112816, 128993, 86776, 120595, 211132, 313424, 84725, 83027, 103723, 11335
0, 89408, 82202, 108305, 99171, 86968, 92717, 139556, 217801, 123641, 84077,
```

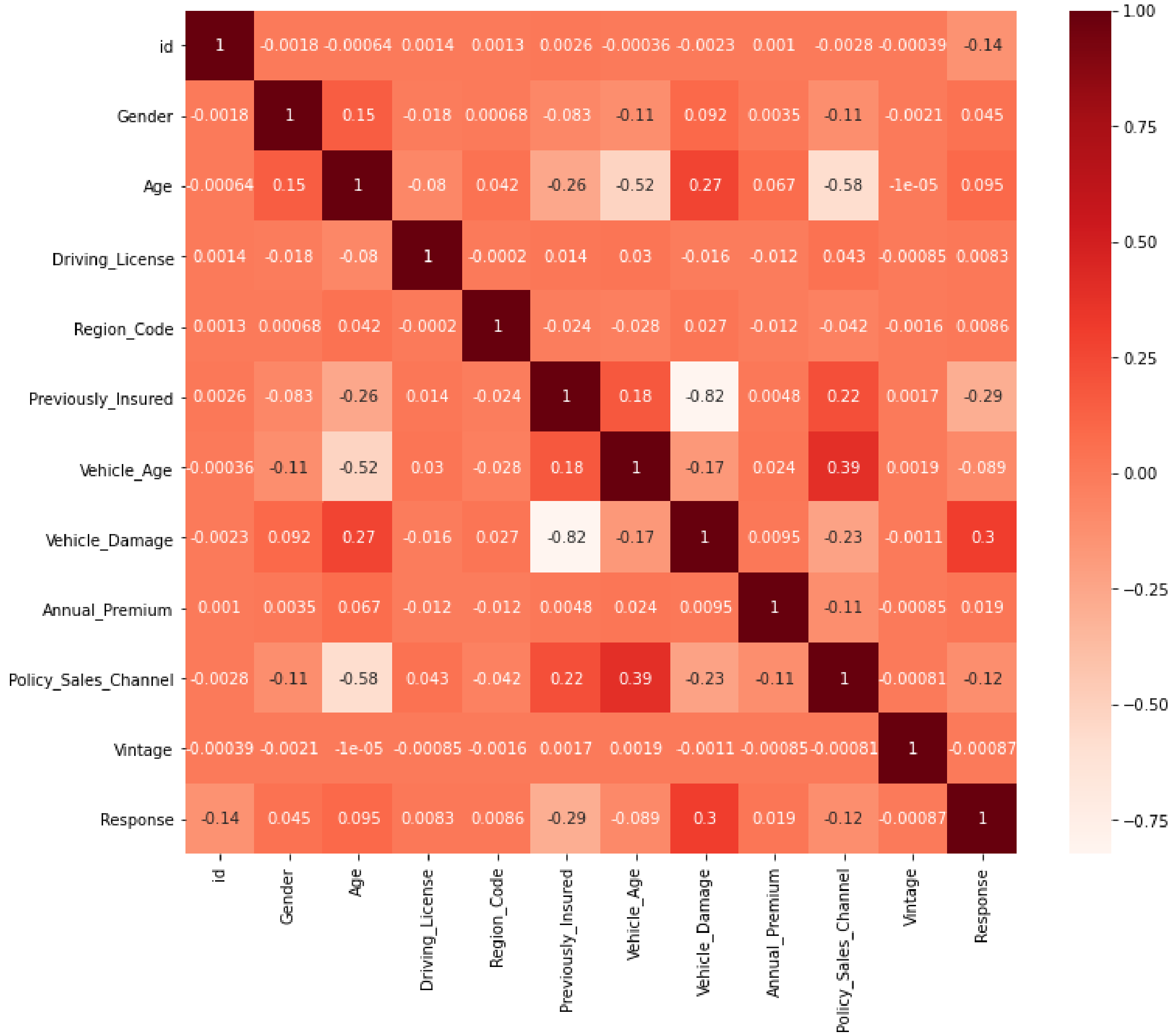
94435, 92298, 93950, 92396, 101045, 87702, 140912, 116796, 84918, 89857, 8458
2, 83086, 199303, 106670, 84609, 188591, 98893, 313854, 101070, 96120, 85851,
86349, 86659, 105998, 105637, 86624, 93335, 108182, 89237, 91625, 88522, 8439
9, 90235, 99449, 130427, 97733, 156270, 83096, 92268, 82354, 139893, 87760, 4
48156, 83817, 156062, 82272, 87575, 88366, 97865, 86631, 85525, 84223, 10756
6, 100549, 93664, 104884, 83771, 102975, 82519, 112113, 87651, 88940, 109461,
96344, 91599, 97823, 104078, 99528, 104643, 184270, 86987, 94770, 183718, 892
37, 86592, 85760, 141582, 91407, 114393, 112667, 104726, 135653, 336395, 8384
6, 109046, 103890, 99887, 105538, 89311, 83869, 152895, 313424, 96532, 21433
0, 308615, 104643, 128686, 97803, 123233, 295106, 88995, 84715, 88446, 85209,
109865, 173513, 82968, 95165, 84046, 92440, 87403, 121207, 82195, 87556, 9153
1, 472042, 90306, 82569, 93629, 90515, 111779, 140293, 101888, 82753, 100254,
101338, 93427, 143007, 97293, 87006, 115719, 97518, 91459, 93012, 85991, 8310
2, 101382, 94810, 158307, 85040, 122625, 133280, 84025, 98417, 111683, 12291
7, 90736, 113177, 96713, 123247, 92584, 91897, 399010, 134013, 84025, 94516,
115034, 144209, 100806, 102789, 96135, 82613, 101215, 82657, 85196, 170758, 9
7223, 102997, 540165, 97868, 88072, 113350, 346200, 93995, 112785, 104233, 93
259, 98529, 96074, 227281, 94892, 101802, 114452, 82802, 83013, 90544, 85235,
93691, 90861, 98662, 109279, 84196, 88523, 95895, 82162, 140225, 85517, 8575
1, 104153, 85706, 87160, 86887, 96997, 90761, 91274, 111208, 174592, 85862, 1
26302, 90025, 88023, 103510, 111715, 103123, 104290, 83525, 313854, 89090, 89
408, 115554, 93095, 105939, 102326, 82096, 89657, 134092, 88964, 100253, 8543
1, 126576, 84080, 101981, 101322, 90830, 86378, 93050, 93367, 89126, 85888, 9
7370, 195863, 98301, 88419, 299811, 109019, 83910, 91644, 88733, 107531, 8402
8, 90024, 144004, 86482, 83859, 85947, 99328, 97986, 84405, 106773, 90151, 11
4210, 152654, 124981, 97363, 94886, 90075, 82482, 93345, 118971, 124342, 1039
34, 85104, 83151, 117057, 89336, 115719, 109099, 97259, 98112, 83261, 90109,
84025, 97265, 90359, 107217, 84085, 89163, 91452, 111883, 82358, 89960, 8373
1, 99584, 82342, 105335, 83005, 106035, 251853, 84459, 82822, 122325, 91266,
336395, 87880, 115719, 84943, 107119, 87954, 85991, 112746, 93597, 316563, 85
720, 100296, 84901, 97803, 162785, 88955, 94435, 96266, 85486, 89478, 83666,
489663, 86144, 107875, 96136, 293038, 126576, 105379, 115719, 91545, 92465, 1
73545, 93253, 83873, 83378, 234236, 83907, 96253, 92203, 87864, 118167, 8270
8, 86983, 97335, 100591, 114757, 111876, 95159, 199996, 87763, 97710, 92353,
82222, 121058, 84728, 107561, 113857, 97370, 110479, 95843, 82928, 86455, 843
97, 101205, 87930, 88669, 86485, 87651, 92249, 101146, 87891, 100574, 83586,
103059, 89604, 97799, 170661, 127279, 89287, 84630, 83757, 90834, 112988, 888
11, 91037, 110592, 96079, 93735, 110999, 101380, 95118, 88683, 121321, 86492,
91369, 83148, 87813, 83151, 83813, 101108, 86656, 84771, 85804, 86983, 82647,
98307, 100950, 143007, 104229, 110308, 83569, 90718, 366891, 84921, 98237, 82
737, 110640, 96500, 86697, 91151, 92764, 540165, 109321, 116777, 106035, 8863
4, 85555, 87796, 85290, 91969, 91379, 94080, 101892, 133280, 88841, 132387, 9
6708, 96609, 99695, 89345, 82753, 346982, 119446, 85019, 120507, 84241, 9359
7, 120245, 92631, 86816, 90771, 88469, 107078, 158356, 85549, 110012, 91369,
85479, 84928, 87386, 173513, 83423, 94606, 95117, 109279, 101333, 91545, 8324
2, 104535, 93786, 82657, 87750, 114391, 100227, 93376, 91698, 102261, 102109,
82685, 87470, 100817, 177150, 107287, 82148, 111671, 86106, 101680, 83638, 10
5148, 94216, 88557, 93705, 86806, 87901, 100227, 83187, 83527, 135545, 11348
6, 214595, 107966, 96371, 96030, 88148, 109321, 91545, 98602, 106843, 87914,
82753, 96030, 100064, 85787, 84392, 97038, 97718, 87582, 89449, 94779, 93597,
122442, 106836, 95014, 110276, 92954, 103603, 101802, 366891, 96941, 133355,
83854, 144074, 85758, 83670, 97468, 89956, 118167, 100245, 90123, 162948, 110
592, 93485, 196513, 92994, 127772, 142995, 86784, 84600, 95130, 229935, 10050
2, 87271, 84742, 110012, 111518, 329525, 91348, 233713, 83666, 83787, 109890,
87956, 91414, 93214, 95845, 86920, 89929, 89088, 117799, 89178, 82713, 93449,
90216, 347606, 85535, 108323, 95712, 87160, 129609, 185009, 86380, 105365, 10
2637, 91214, 87666, 108651, 118745, 107003, 88656, 90988, 91677, 129749, 1063

35, 90090, 91670, 93611, 83027, 95817, 87364, 90856, 84915, 82031, 84176, 954
07, 97917, 89464, 83590, 130703, 96095, 144473, 83196, 83526, 84641, 86618, 8
6482, 127575, 85055, 87363, 94135, 93345, 93950, 132308, 117793, 83590, 9369
1, 96724, 86259, 82419, 85872, 95753, 113800, 93951, 99715, 101401, 109570, 1
01190, 109292, 132387, 99445, 82342, 89139, 97979, 97325, 89334, 152009, 9189
7, 87295, 92906, 87065, 88985, 85600, 87470, 86013, 82024, 143814, 87692, 857
97, 86595, 89419, 103807, 96127, 86082, 93136, 117793, 113684, 101401, 88944,
99445, 164990, 89843, 91916, 85949, 113512, 119025, 84394, 232791, 95578, 103
415, 87386, 95875, 117237, 83646, 102757, 95135, 111040, 139130, 105842, 8263
4, 114391, 93149, 101413, 83808, 91505, 82068, 82216, 90162, 89509, 97523, 96
248, 90523, 103262, 88026, 152870, 402863, 98645, 97997, 90985, 89605, 98444,
94696, 83566, 82201, 92129, 86807, 84901, 91638, 82424, 126583, 299410, 9522
6, 98893, 117993, 89373, 82031, 102797, 91279, 110204, 86860, 94477, 96108, 9
5971, 103828, 98073, 113684, 82657, 92291, 291169, 94826, 84380, 106671, 3363
95, 113452, 105538, 105842, 106897, 87101, 82533, 218095, 90941, 88833, 8758
3, 105836, 88996, 87750, 110496, 252866, 84380, 86396, 92738, 98854, 294467,
111114, 101680, 103397, 98337, 84036, 138002, 366891, 86941, 97335, 87474, 84
984, 82998, 88433, 83963, 107748, 87274, 107195, 131469, 110266, 84772, 9102
3, 100772, 96067, 87084, 94942, 90006, 89377, 102045, 103068, 114628, 82176,
89408, 95284, 96603, 216660, 119272, 91260, 110052, 95432, 98934, 88850, 8467
0, 86320, 130272, 82409, 108768, 97413, 185009, 83666, 89101, 85568, 92790, 1
40912, 88653, 118594, 100817, 92232, 101575, 117793, 103417, 108961, 87031, 8
2284, 85657, 296891, 97523, 92166, 109448, 97305, 86182, 101670, 93089, 12845
7, 82414, 86823, 101682, 82713, 89041, 90156, 117793, 174351, 82454, 83174, 8
2753, 82295, 87228, 82375, 88018, 91606, 82682, 88363, 143445, 118613, 87545,
84786, 99474, 99685, 101401, 100445, 105212, 89549, 101397, 83666, 96347, 905
41, 85604, 399010, 86082, 87004, 201587, 115232, 94331, 111557, 83997, 84380,
124950, 87120, 229935, 100337, 88883, 92978, 92994, 89067, 106843, 162733, 54
0165, 86199, 84807, 98184, 102285, 102046, 96266, 104635, 82824, 109080, 1229
06, 99718, 96158, 91770, 83355, 85571, 336395, 140789, 87431, 243141, 94446,
82675, 91029, 83005, 291169, 89581, 91396, 82314, 99021, 472042, 82932, 11041
0, 91452, 94955, 90809, 110204, 91108, 84651, 85599, 227526, 114210, 90985, 8
5466, 94970, 82682, 114145, 92091, 91279, 84950, 103222, 88040, 104521, 9109
4, 84109, 102314, 472042, 85049, 109236, 86045, 93389, 83124, 103748, 121601,
92453, 93089, 93983, 87759, 88773, 83265, 99528, 134022, 91698, 85332, 87057,
99750, 296891, 136156, 83424, 131874, 116796, 84699, 107189, 96061, 119425, 8
2738, 91969, 110479, 92042, 114391, 97056, 119272, 96199, 118644, 108034, 828
52, 91505, 85055, 291169, 86394, 97865, 97187, 112176, 84649, 89765, 82058, 3
40439, 105379, 164759, 87507, 86182, 89375, 106396, 85442, 88623, 87679, 1736
53, 84811, 86205, 84298, 85347, 86275, 108692, 91533, 116606, 86349, 155317,
93458, 92386, 92877, 97155, 84178, 85983, 235683, 83666, 85752, 158860, 9395
0, 96425, 92795, 84396, 85922, 91713, 84715, 99987, 107446, 115014, 311948, 8
9422, 86084, 86943, 85268, 135998, 102370, 183718, 120134, 96482, 82877, 9044
6, 134092, 100544, 137041, 84818, 90007, 93036, 95362, 96884, 88518, 95014, 9
2181, 110655, 94253, 108034, 91703, 155184, 90514, 136113, 91605, 95121, 9154
5, 82958, 112115, 94181, 88433, 107566, 109965, 89575, 94931, 97748, 99730, 8
9090, 135450, 232791, 97151, 88806, 107120, 211132, 92555, 85787, 84007, 9262
1, 84055, 174094, 83117, 86092, 85993, 101852, 235683, 128025, 86512, 88955,
83654, 97391, 97510, 83979, 116796, 83196, 135038, 83233, 84807, 112024, 8963
7, 116272, 86419, 119373, 114019, 88403, 84452, 84398, 86452, 91279, 83700, 8
4084, 84098, 85087, 89637, 96026, 87971, 86781, 86378, 85370, 96879, 97545, 9
2300, 92120, 88419, 108766, 86888, 105608, 82649, 339396, 91807, 88072, 10423
7, 123396, 85722, 84944, 101013, 88124, 84824, 113663, 84085, 104710, 169021,
98645, 83360, 93597, 104939, 84912, 82060, 109803, 82705, 118922, 89419, 1130
88, 97065, 122051, 125876, 366891, 96230, 122051, 82928, 102046, 150412, 9146
9, 84328, 110049, 132193, 93337, 82632, 86574, 313854, 115554, 93042, 101568,
100543, 123783, 90559, 87498, 89646, 108768, 97070, 84844, 97869, 84004, 9324

2, 106213, 85055, 84866, 85442, 85582, 84028, 402097, 187756, 114983, 108653, 86000, 101070, 84386, 91882, 86956, 87758, 89652, 92994, 94798, 86834, 103314, 155413, 82531, 82000, 88989, 83489, 106646, 85787, 93925, 90355, 85858, 93104, 122625, 85315, 105503, 102948, 95739, 109080, 83424, 87536, 92898, 87130, 120340, 87286, 85447, 199996, 99611, 110592, 116852, 88591, 109080, 82914, 115672, 85108, 337327, 82662, 97038, 266118, 98442, 156389, 95221, 96468, 85074, 91407, 159243, 91293, 211132, 122424, 114536, 96060, 141863, 136163, 144224, 85828, 188591, 132401, 92161, 100656, 131570, 87980, 103561, 91876, 84025, 97615, 104962, 111671, 90585, 89769, 86592, 90138, 113230, 87658, 92860, 125490, 100255, 94581, 90019, 92041, 88518, 170402, 86013, 110573, 102049, 84219, 83187, 117683, 94846, 87736, 86608, 98337, 82074, 99552, 145053, 88940, 85259, 82880, 83871, 98922, 85514, 88664, 89168, 91267, 85977, 86272, 128993, 100172, 89164, 119963, 88659, 82780, 84877, 94957, 96230, 127161, 82572, 121489, 127454, 495106, 88653, 159243, 111883, 109463, 84080, 95793, 90682, 83458, 92516, 98299, 88800, 92898, 91004, 98794, 94327, 101808, 83586, 128941, 102744, 86408, 86562, 103534, 87056, 90175, 82017, 87759, 85777, 104965, 98832, 92210, 112611, 88057, 85402, 313424, 88825, 87674, 147075, 97068, 89139, 91061, 111683, 84585, 95850, 92175, 235683, 82067, 102257, 83600, 82067, 96801, 92347, 107292, 185338, 100094, 92985, 94249, 85467, 88891, 88296, 104280, 85283, 98571, 107680, 83572, 108736, 100825, 84230, 84691, 220702, 82466, 90926, 102326, 93951, 82935, 109762, 94431, 102974, 92428, 96451, 82326, 118327, 92327, 82060, 83900, 96349, 244589, 103314, 83666, 93360, 98428, 87043, 88328, 104566, 83290, 111384, 85230, 86476, 199996, 97741, 115092, 83321, 82979, 88977, 84025, 104765, 91214, 84331, 92381, 86774, 291169, 140225, 82307, 98956, 84469, 86855, 86573, 82613, 167548, 82373, 102753, 102874, 89601, 82024, 188014, 99096, 107971, 109015, 104388, 90947, 83575, 295106, 106035, 90985, 87815, 97432, 346982, 109374, 91698, 87044, 167548, 109080, 111268, 92597, 100171, 96944, 151034, 85313, 235683, 220581, 93312, 87664, 111257, 337573, 89956, 93597, 86366, 99986, 92185, 119593, 85516, 94767, 91267, 101813, 94819, 83643, 90989, 83396, 126576, 251817, 93458, 85555, 84312, 90637, 85034, 84319, 99997, 89054, 85757, 124881, 97693, 86164, 89582, 87130, 165395, 85060, 94121, 93221, 92283, 95453, 91305, 101664, 110611, 93515, 90262, 84397, 86496, 92601, 111358, 123928, 82005, 86527, 83278, 87956, 106114, 105013, 101160, 90648, 106344, 87401, 87067, 196587, 143525, 90090, 144874, 88316, 106035, 122351, 88659, 99445, 102747, 95102, 82824, 120320, 91164, 87880, 89695, 93259, 100950, 192052, 87709, 90800, 112667, 109299, 339396, 107321, 94581, 88278, 110323, 87423, 174721, 94717, 82758, 103195, 87217, 104924, 92555, 101938, 97164, 84046, 99129, 89329, 83758, 83915, 130474, 130980, 92014, 85108, 120927, 117799, 86104, 97718, 82421, 84529, 94128, 137796, 87228, 84887, 82028, 130307, 175886, 122625, 99258, 82962, 86587, 110479, 83666, 110592, 90765, 297040, 125848, 86191, 83924, 90584, 88117, 96780, 92932, 100551, 199769, 87439, 87730, 91798, 85068, 196890, 123362, 84490, 83234, 93360, 116451, 98102, 105052, 107414, 83076, 86069, 108764, 142271, 118964, 96856, 114164, 97694, 83597, 98317, 87891, 157479, 84742, 109958, 90299, 82968, 86467, 91389, 140744, 83586, 97823, 340439, 90507, 154624, 88870, 86793, 135109, 169300, 113656, 99024, 82713, 93950, 114762, 108182, 84766, 86496, 84213, 82454, 136253, 87056, 83177, 91668, 96571, 88989, 96316, 108800, 90838, 90861, 104966, 85780, 101953, 105979, 90700, 121275, 104819, 90918, 83095, 93360, 101535, 82051, 95342, 102931, 107218, 93521, 96495, 112264, 86218, 99445, 98519, 92228, 130047, 241735, 109538, 122134, 122625, 108323, 122494, 85610, 88269, 124488, 199996, 101449, 93473, 99340, 95187, 91882, 92757, 87032, 97847, 103931, 108707, 91545, 84396, 102135, 303550, 91637, 91279, 83096, 87755, 88529, 108768, 113350, 188591, 89650, 132789, 84521, 103999, 154665, 174859, 87617, 91038, 84984, 85333, 94322, 94767, 150749, 84339, 88326, 86861, 83688, 84432, 103439, 182288, 126859, 83206, 97216, 119191, 83666, 159979, 115702, 141394, 93837, 188591, 116777, 101172, 95310, 84611, 109515, 90791, 183718, 85582, 115189, 106035, 101022, 85983, 170763, 96371, 97928, 102513, 95109, 99339, 104108, 138836, 110592, 100981, 108

```
208, 107782, 88618, 101450, 119223, 129901, 88060, 148931, 101026, 84541, 835
89, 83125, 86542, 152417, 174814, 96074, 98783, 109577, 84494, 103150, 11677
7, 113577, 87295, 90837, 237697, 87178, 82578, 85127, 109003, 114589, 165395,
87988, 98507, 82833, 109515, 149375, 90138, 93983, 82002, 83512, 211132, 2009
98, 84841, 84679, 90607, 130760, 91138, 108058, 93597, 90116, 90601, 220702,
97949, 121123, 84611, 111025, 92985, 96606, 399010, 204681, 95773, 126809, 87
708, 88567, 90689, 100431, 101049, 96392, 106112, 99604, 147943, 114090, 8257
7, 102572, 150508, 96987, 87514, 90809, 336395, 84609, 106533, 84157, 90584,
108400, 424578, 103811, 101997, 101371, 82619, 105969, 91266, 87470, 92783, 8
8274, 94111, 86362, 88282, 90630, 93597, 82925, 98849, 92859, 188591, 90070,
84109, 102931, 87553, 85549, 145072, 101942, 109372, 88618, 105887, 86253, 90
967, 84396, 96620, 84080, 174298, 90452, 86579, 96045, 83499, 93389, 92042, 9
0616, 96616, 92700, 85484, 87754, 92691, 146390, 90467, 92995, 106035, 21590
5, 97710, 116363, 103191, 96759, 118916, 89535, 97852, 124626, 92181, 103886,
83008, 472042, 88128, 143163, 123008, 98800, 84414, 218552, 339396, 120319, 1
95863, 90250, 101684, 102328, 95763, 84585, 99611, 94335, 90353, 91324, 9106
0, 193637, 178137, 99445, 105969, 84471, 88409, 83683, 145414, 87395, 92228,
90690, 94557, 105379, 101953, 100690, 96135, 95136, 83132, 98794, 89440, 9945
2, 84298, 172150, 96821, 127292, 117683, 96068, 132789, 84187, 91312, 127242,
101795, 82397, 111040, 113636, 107412, 117794, 115252, 88985, 99506, 106737,
232791, 87548, 85159, 111327, 110068, 82581, 95221, 82868, 103534, 83638, 981
52, 87914, 84786, 83278, 89419, 100218, 95120, 82908, 87844, 96636, 100218, 9
5661, 88683, 99706, 89282, 105538, 87914, 101078, 90727, 82491, 113260, 8277
5, 92764, 89379, 82598, 267909, 107412, 84085, 99596, 84501, 92865, 92901, 85
956, 103150, 94320, 94506, 89162, 103629, 91672, 106274, 88405, 111484, 15435
4, 86916, 105779, 104634, 93902, 95773, 85797, 107729, 85304, 86465, 98425, 9
8018, 98645, 188591, 86455, 106112, 87244, 118626, 82598, 126576, 91267, 1011
46, 83206, 102832, 90967, 83739, 106570, 82353, 92428, 91505, 106112, 148945,
110012, 93738, 98331, 89637, 141411, 92404, 105006, 123425, 126576, 106486, 9
5962, 93195, 84560, 89705, 83666, 87336, 90541, 93597, 82740, 83666, 110323,
104774, 91255, 109582, 109067, 82028, 96495, 86659, 87096, 99941, 100189, 315
565, 100796, 83523, 87657, 84702, 106219, 83338, 82914, 133633, 91469, 93138,
89456, 92005, 89305, 116937, 83338, 93253, 84108, 82317, 99753, 103415, 9019
8, 95150, 93922, 89950, 98741, 84552, 303339, 105802, 102626, 122186, 88555,
90918, 93367, 89242, 97955, 91897, 82254, 92120, 108008, 106973, 93253, 9212
0, 122784, 109321, 87386, 94743, 138002, 84026, 151034, 83958, 87271, 86013,
83027, 134747, 199154, 109019, 107586, 119414, 124131, 105335, 83666, 87741,
82335, 154802, 90630, 162785, 108957, 91531, 100004, 98177, 84613, 91882, 837
57, 93030, 102744, 89843, 99512, 86718, 163442, 91939, 118312, 122625, 11062
3, 93021, 88891, 83172, 89718, 94506, 98289, 91969, 98164, 85148, 94781, 2272
81]
```

In [24]:
```
#Using Pearson Correlation
plt.figure(figsize=(12,10))
cor = result_df.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)
plt.show()
#Correlation with output variable
cor_target = abs(cor["Annual_Premium"])
#Selecting highly correlated features
relevant_features = cor_target[cor_target>0]
print("Correlation Coefficeint With Respect to Response")
print(relevant_features)
```




```
Correlation Coefficeint With Respect to Response
id                      0.001042
Gender                  0.003502
Age                     0.067392
Driving_License         0.012084
Region_Code             0.012012
Previously_Insured      0.004776
Vehicle_Age             0.023545
Vehicle_Damage          0.009504
Annual_Premium          1.000000
Policy_Sales_Channel    0.114035
Vintage                 0.000855
Response                0.019477
Name: Annual_Premium, dtype: float64
```

Linear Regression

In [25]:
```
x=result_df.drop(['Annual_Premium','id'], axis = 1)
y=result_df['Annual_Premium']
print(x.shape)
print(y.shape)
```

```
(508146, 10)
(508146,)
```

In [26]:
```
from sklearn.model_selection import train_test_split
train_features, test_features, train_labels, test_labels = train_test_split(x,
y, test_size = 0.3, random_state = 0)
```

In [27]:
```
print('Training Features Shape:', train_features.shape)
print('Training Labels Shape:', train_labels.shape)
print('Testing Features Shape:', test_features.shape)
print('Testing Labels Shape:', test_labels.shape)
```

```
Training Features Shape: (355702, 10)
Training Labels Shape: (355702,)
Testing Features Shape: (152444, 10)
Testing Labels Shape: (152444,)
```

In [28]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(train_features, train_labels)
y_pred = model.predict(test_features)
```

In [29]:
```
print("Training set score: {:.4f}".format(model.score(train_features,train_lab
els)))

print("Test set score: {:.4f}".format(model.score(test_features,test_labels)))
```

```
Training set score: 0.0216
Test set score: 0.0202
```

In [30]:
```
print(model.coef_)
print(model.intercept_)
```

```
[-2.20296565e+02  5.68341300e+01 -2.40386630e+03 -2.20994504e+01
  1.84695901e+03  2.87446966e+03  7.89872640e+02 -4.05438669e+01
 -2.30672795e-01  9.38180788e+02]
33231.52629203656
```

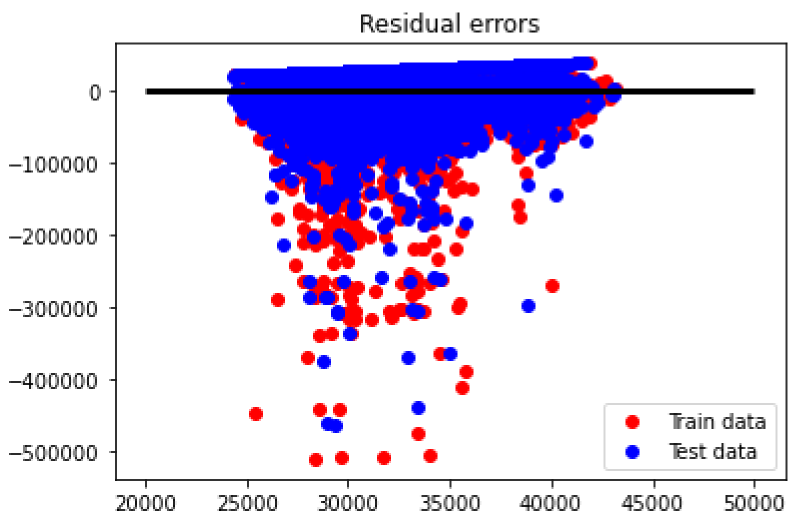
In [31]:
```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(test_labels, y_pred)
rmse = np.sqrt(mse)
print("RMSE value: {:.4f}".format(rmse))
```

```
RMSE value: 16885.9328
```

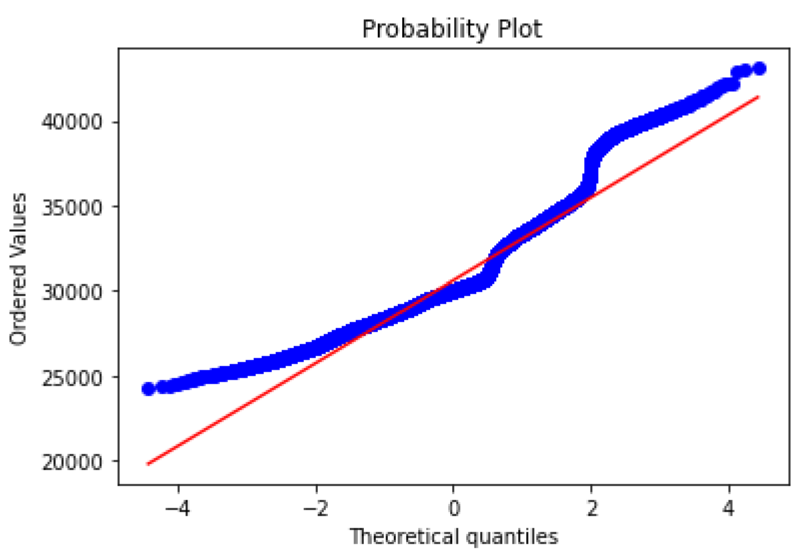
In [32]:
```
# Calculate and print r2_score

from sklearn.metrics import r2_score
print ("R2 Score value: {:.4f}".format(r2_score(test_labels, y_pred)))
```

```
R2 Score value: 0.0202
```

In [33]:
```
# Plotting residual errors
plt.scatter(model.predict(train_features), model.predict(train_features) - tra
in_labels, color = 'red', label = 'Train data')
plt.scatter(model.predict(test_features), model.predict(test_features) - test_
labels, color = 'blue', label = 'Test data')
plt.hlines(xmin = 20000, xmax = 50000, y = 0, linewidth = 3)
plt.title('Residual errors')
plt.legend(loc = 4)
plt.show()
```

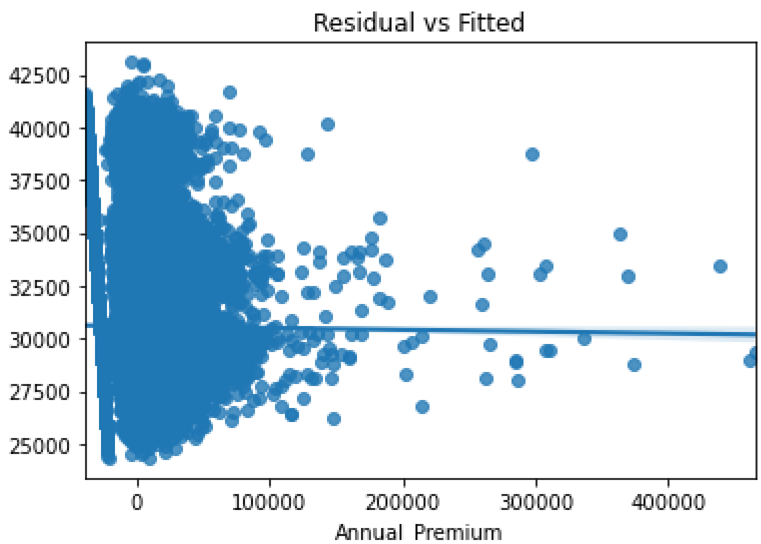


In [34]:
```
import statsmodels.api as sm
import pylab
import scipy.stats as stats
stats.probplot(y_pred.reshape(-1), dist="norm", plot=pylab)
pylab.show()
```



In [35]:
```
import seaborn as sns
residuals = test_labels - y_pred
ax = sns.regplot(x=residuals, y=y_pred)
ax.set_title('Residual vs Fitted')
```

Out[35]:
```
Text(0.5, 1.0, 'Residual vs Fitted')
```



Lasso, Ridge

In [36]:
```
from sklearn.linear_model import Lasso
```

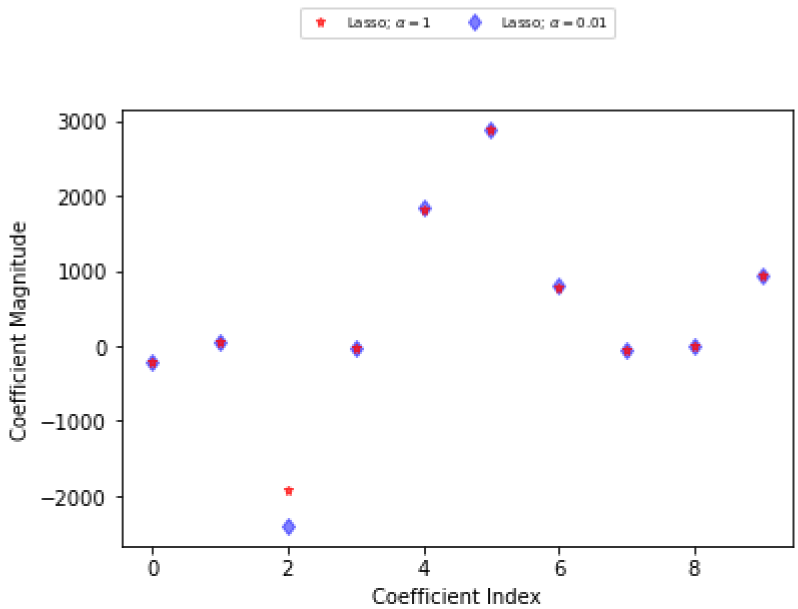
In [37]:
```
lasso = Lasso()
lasso.fit(train_features,train_labels)
train_score=lasso.score(train_features,train_labels)
test_score=lasso.score(test_features,test_labels)
coeff_used = np.sum(lasso.coef_!=0)
print("training score:", train_score)
print("test score: ", test_score)
print("number of features used: ", coeff_used)
lasso001 = Lasso(alpha=0.01, max_iter=10e5)
lasso001.fit(train_features,train_labels)
train_score001=lasso001.score(train_features,train_labels)
test_score001=lasso001.score(test_features,test_labels)
coeff_used001 = np.sum(lasso001.coef_!=0)
print("training score for alpha=0.01:", train_score001)
print("test score for alpha =0.01: ", test_score001)
print("number of features used: for alpha =0.01:", coeff_used001)
lasso00001 = Lasso(alpha=0.0001, max_iter=10e5)
lasso00001.fit(train_features,train_labels)
train_score00001=lasso00001.score(train_features,train_labels)
test_score00001=lasso00001.score(test_features,test_labels)
coeff_used00001 = np.sum(lasso00001.coef_!=0)
print("training score for alpha=0.0001:", train_score00001)
print("test score for alpha =0.0001: ", test_score00001)
print("number of features used: for alpha =0.0001:", coeff_used00001)
```

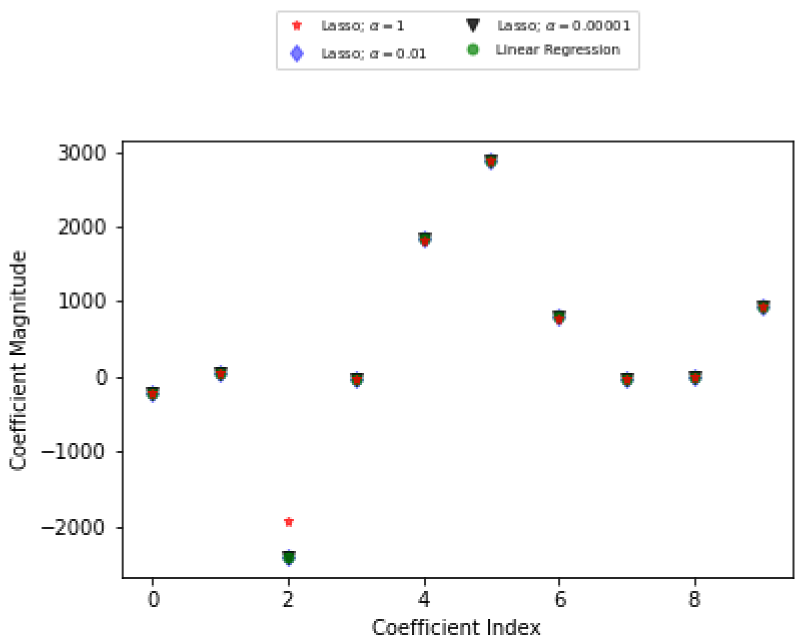
```
training score: 0.021558759070389066
test score:  0.020205815285361806
number of features used:  10
training score for alpha=0.01: 0.02156062130858072
test score for alpha =0.01:  0.02020252115769039
number of features used: for alpha =0.01: 10
training score for alpha=0.0001: 0.02156062149479465
test score for alpha =0.0001:  0.020202470244255255
number of features used: for alpha =0.0001: 10
```

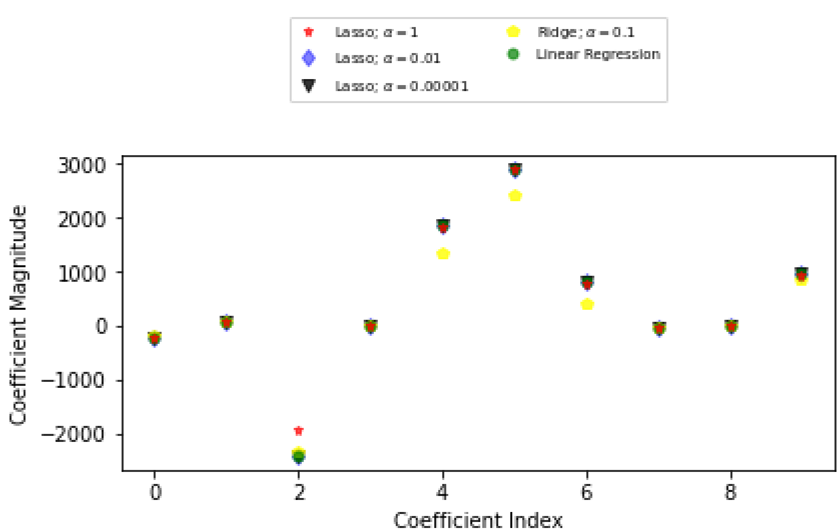
In [38]:
```
from sklearn.linear_model import Ridge
ridgereg = Ridge(alpha=0, normalize=True)
ridgereg.fit(train_features, train_labels)
y_pred = ridgereg.predict(test_features)
from sklearn import metrics
print("R-Square Value",r2_score(test_labels,y_pred))
print ("mean_absolute_error :",metrics.mean_absolute_error(test_labels, y_pred
))
print ("mean_squared_error : ",metrics.mean_squared_error(test_labels, y_pred
))
print ("root_mean_squared_error : ",np.sqrt(metrics.mean_squared_error(test_la
bels, y_pred)))
ridgereg = Ridge(alpha=0.1, normalize=True)
ridgereg.fit(train_features, train_labels)
y_pred = ridgereg.predict(test_features)
print("R-Square Value",r2_score(test_labels,y_pred))
print ("mean_absolute_error :",metrics.mean_absolute_error(test_labels, y_pred
))
print ("mean_squared_error : ",metrics.mean_squared_error(test_labels, y_pred
))
print ("root_mean_squared_error : ",np.sqrt(metrics.mean_squared_error(test_la
bels, y_pred)))
print(ridgereg.coef_)
```

```
R-Square Value 0.02020246972837836
mean_absolute_error : 11950.144186561989
mean_squared_error :  285134727.5654296
root_mean_squared_error :  16885.932830774545
R-Square Value 0.020080346450857656
mean_absolute_error : 11939.191499945779
mean_squared_error :  285170267.1400758
root_mean_squared_error :  16886.985140636436
[-1.83744566e+02  5.12760030e+01 -2.35563745e+03 -1.96128702e+01
  1.30913268e+03  2.40050122e+03  3.91052677e+02 -3.56696578e+01
 -2.03058346e-01  8.41517289e+02]
```

In [39]:
```
plt.plot(lasso.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,color=
'red',label=r'Lasso; $\alpha = 1$',zorder=7)
plt.plot(lasso001.coef_,alpha=0.5,linestyle='none',marker='d',markersize=6,col
or='blue',label=r'Lasso; $\alpha = 0.01$')
plt.xlabel('Coefficient Index')
plt.ylabel('Coefficient Magnitude')
plt.legend(fontsize=7,loc="lower center", bbox_to_anchor=(0.5, 1.15), ncol=2)
plt.show()

plt.plot(lasso.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,color=
'red',label=r'Lasso; $\alpha = 1$',zorder=7)
plt.plot(lasso001.coef_,alpha=0.5,linestyle='none',marker='d',markersize=6,col
or='blue',label=r'Lasso; $\alpha = 0.01$')
plt.plot(lasso00001.coef_,alpha=0.8,linestyle='none',marker='v',markersize=6,c
olor='black',label=r'Lasso; $\alpha = 0.00001$')
plt.plot(model.coef_,alpha=0.7,linestyle='none',marker='o',markersize=5,color=
'green',label='Linear Regression',zorder=2)
plt.xlabel('Coefficient Index')
plt.ylabel('Coefficient Magnitude')
plt.legend(fontsize=7,loc="lower center", bbox_to_anchor=(0.5, 1.15), ncol=2)
plt.show()

plt.plot(lasso.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,color=
'red',label=r'Lasso; $\alpha = 1$',zorder=7)
plt.plot(lasso001.coef_,alpha=0.5,linestyle='none',marker='d',markersize=6,col
or='blue',label=r'Lasso; $\alpha = 0.01$')
plt.plot(lasso00001.coef_,alpha=0.8,linestyle='none',marker='v',markersize=6,c
olor='black',label=r'Lasso; $\alpha = 0.00001$')
plt.plot(ridgereg.coef_,alpha=0.8,linestyle='none',marker='p',markersize=6,col
or='yellow',label=r'Ridge; $\alpha = 0.1$')
plt.plot(model.coef_,alpha=0.7,linestyle='none',marker='o',markersize=5,color=
'green',label='Linear Regression',zorder=2)
plt.xlabel('Coefficient Index')
plt.ylabel('Coefficient Magnitude')
plt.legend(fontsize=7,loc="lower center", bbox_to_anchor=(0.5, 1.15), ncol=2)
plt.tight_layout()
plt.show()
```


Lasso; $\alpha = 1$
Lasso; $\alpha = 0.01$
Coefficient Magnitude
Coefficient Index


Lasso; $\alpha = 1$
Lasso; $\alpha = 0.01$
Lasso; $\alpha = 0.00001$
Linear Regression
Coefficient Magnitude
Coefficient Index


Lasso; $\alpha = 1$
Lasso; $\alpha = 0.01$
Lasso; $\alpha = 0.00001$
Ridge; $\alpha = 0.1$
Linear Regression
Coefficient Magnitude
Coefficient Index

Logistic Regression

In [34]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
```

In [35]:
```
x=result_df.drop('Response', axis = 1)
y=result_df['Response']
```

In [36]:
```
from sklearn.model_selection import train_test_split
train_features, test_features, train_labels, test_labels = train_test_split(x,
y, test_size = 0.3, random_state = 0)
```

In [37]:
```
from imblearn.under_sampling import NearMiss
nr = NearMiss()
train_features, train_labels = nr.fit_resample(train_features, train_labels)
```

In [38]:
```
print('Training Features Shape:', train_features.shape)
print('Training Labels Shape:', train_labels.shape)
print('Testing Features Shape:', test_features.shape)
print('Testing Labels Shape:', test_labels.shape)
```

```
Training Features Shape: (65378, 11)
Training Labels Shape: (65378,)
Testing Features Shape: (152444, 11)
Testing Labels Shape: (152444,)
```

In [39]:
```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(max_iter=1000)

logreg.fit(train_features, train_labels)

y_pred=logreg.predict(test_features)

print(classification_report(test_labels, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.30      0.46    138423
           1       0.11      0.82      0.19     14021

    accuracy                           0.35    152444
   macro avg       0.53      0.56      0.32    152444
weighted avg       0.87      0.35      0.43    152444
```

In [40]:
```
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(test_labels, y_pred)*100)
```

```
Accuracy: 35.086982760882684
```

In [41]:
```
print("Accuracy:",metrics.accuracy_score(test_labels, y_pred))
print("Precision:",metrics.precision_score(test_labels, y_pred))
print("Recall:",metrics.recall_score(test_labels, y_pred))
```

```
Accuracy: 0.35086982760882685
Precision: 0.1067787664700599
Recall: 0.8224805648669853
```

In [42]:
```
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(test_labels, y_pred)
cnf_matrix
```

Out[42]:
```
array([[41956, 96467],
       [ 2489, 11532]], dtype=int64)
```

In [43]:
```
import numpy as np
predictions = logreg.predict(test_features)
errors = abs(predictions - test_labels)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
```

```
Mean Absolute Error: 0.65 degrees.
```

In [44]:
```
from sklearn.feature_selection import RFE
rfe = RFE(logreg, n_features_to_select=8)
fit = rfe.fit(train_features, train_labels)
print("Num Features: %d" % fit.n_features_)
print("Selected Features: %s" % fit.support_)
print("Feature Ranking: %s" % fit.ranking_)
print("Features:", train_features.columns)
```

```
Num Features: 8
Selected Features: [False  True  True  True  True  True  True  True False  Tr
ue False]
Feature Ranking: [4 1 1 1 1 1 1 1 3 1 2]
Features: Index(['id', 'Gender', 'Age', 'Driving_License', 'Region_Code',
       'Previously_Insured', 'Vehicle_Age', 'Vehicle_Damage', 'Annual_Premiu
m',
       'Policy_Sales_Channel', 'Vintage'],
      dtype='object')
```

In [45]:
```
logreg_imp = LogisticRegression(max_iter=1000)
train_important = train_features.drop(['id','Annual_Premium','Vintage'], axis=
1)
test_important = test_features.drop(['id','Annual_Premium','Vintage'], axis=1)
logreg_imp.fit(train_important, train_labels)
predictions = logreg_imp.predict(test_important)
errors = abs(predictions - test_labels)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
print(classification_report(test_labels, predictions))
```

```
Mean Absolute Error: 0.38 degrees.
              precision    recall  f1-score   support

           0       0.99      0.58      0.73    138423
           1       0.19      0.97      0.32     14021

    accuracy                           0.62    152444
   macro avg       0.59      0.77      0.52    152444
weighted avg       0.92      0.62      0.69    152444
```

In [46]:
```
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(test_labels, predictions)
cnf_matrix
```

Out[46]:
```
array([[80271, 58152],
       [  488, 13533]], dtype=int64)
```

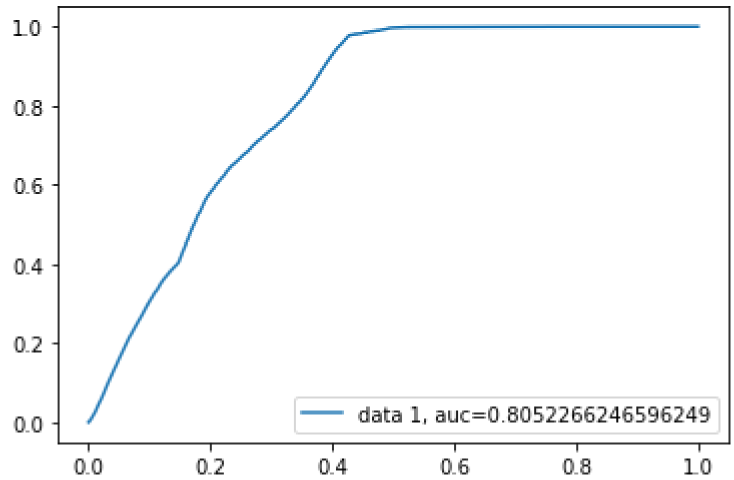
In [47]:
```
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(test_labels, predictions)*100)
```

```
Accuracy: 61.53341554931647
```

In [48]:
```
print("Accuracy:",metrics.accuracy_score(test_labels, predictions))
print("Precision:",metrics.precision_score(test_labels, predictions))
print("Recall:",metrics.recall_score(test_labels, predictions))
```

```
Accuracy: 0.6153341554931647
Precision: 0.1887842644904792
Recall: 0.9651950645460381
```

In [49]:
```
y_pred_proba = logreg_imp.predict_proba(test_important)[::,1]
fpr, tpr, _ = metrics.roc_curve(test_labels,  y_pred_proba)
auc = metrics.roc_auc_score(test_labels, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



Assignment 3

Decision Tree

In [56]:
```
import timeit
start = timeit.default_timer()
x=result_df.drop('Response', axis = 1)
y=result_df['Response']
from sklearn.model_selection import train_test_split
train_features, test_features, train_labels, test_labels = train_test_split(x,
y, test_size = 0.3, random_state = 0)
```

In [57]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(train_features, train_labels)
dt_pred = dt.predict(test_features)
stop = timeit.default_timer()
print('Time: ', stop - start)
```

```
Time:  2.209480400000004
```

In [59]:
```
param_dict = {
    "criterion" : ['gini','entropy'],
    "max_depth":range(1,10),
    "min_samples_split":range(1,10),
    "min_samples_leaf":range(1,5)
}
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(dt,param_grid = param_dict,cv=10,verbose=1,n_jobs=-1)
grid.fit(train_features,train_labels)
```

```
Fitting 10 folds for each of 648 candidates, totalling 6480 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  52 tasks      | elapsed:    2.6s
[Parallel(n_jobs=-1)]: Done 352 tasks      | elapsed:   18.4s
[Parallel(n_jobs=-1)]: Done 634 tasks      | elapsed:   45.5s
[Parallel(n_jobs=-1)]: Done 984 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done 1434 tasks      | elapsed:  2.8min
[Parallel(n_jobs=-1)]: Done 1984 tasks      | elapsed:  4.6min
[Parallel(n_jobs=-1)]: Done 2634 tasks      | elapsed:  7.5min
[Parallel(n_jobs=-1)]: Done 3384 tasks      | elapsed: 10.8min
[Parallel(n_jobs=-1)]: Done 4234 tasks      | elapsed: 12.4min
[Parallel(n_jobs=-1)]: Done 5184 tasks      | elapsed: 15.6min
[Parallel(n_jobs=-1)]: Done 6234 tasks      | elapsed: 20.8min
[Parallel(n_jobs=-1)]: Done 6480 out of 6480 | elapsed: 22.2min finished
```

Out[59]:
```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(1, 10),
                         'min_samples_leaf': range(1, 5),
                         'min_samples_split': range(1, 10)},
             verbose=1)
```

In [60]:
```
grid.best_params_
```

Out[60]:
```
{'criterion': 'entropy',
 'max_depth': 8,
 'min_samples_leaf': 3,
 'min_samples_split': 7}
```

In [61]:
```
start = timeit.default_timer()
dt = DecisionTreeClassifier(criterion='entropy', max_depth=8, min_samples_leaf
=3, min_samples_split=7)
dt.fit(train_features, train_labels)
dt_pred = dt.predict(test_features)
stop = timeit.default_timer()
print('Time: ', stop - start)
```

```
Time:  1.1883704000001671
```

In [62]:
```
errors = abs(dt_pred - test_labels)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
print("Accuracy:",metrics.accuracy_score(test_labels, dt_pred)*100)
conf = confusion_matrix(test_labels,dt_pred)
print(conf)
```

```
Mean Absolute Error: 0.09 degrees.
Accuracy: 90.79596441972134
[[138412     11]
 [ 14020      1]]
```

In [63]:
```
print("Accuracy:",metrics.accuracy_score(test_labels, dt_pred))
print("Precision:",metrics.precision_score(test_labels, dt_pred))
print("Recall:",metrics.recall_score(test_labels, dt_pred))
```

```
Accuracy: 0.9079596441972134
Precision: 0.08333333333333333
Recall: 7.132158904500393e-05
```

In [64]:
```
importances = list(dt.feature_importances_)
feature_importances = [(feature, round(importance, 2)) for feature, importance
in zip(x.columns, importances)]
feature_importances = sorted(feature_importances, key = lambda x: x[1], revers
e = True)
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_imp
ortances];
```

```
Variable: Previously_Insured   Importance: 0.55
Variable: id                   Importance: 0.28
Variable: Age                  Importance: 0.07
Variable: Vehicle_Damage       Importance: 0.07
Variable: Policy_Sales_Channel Importance: 0.02
Variable: Gender               Importance: 0.0
Variable: Driving_License      Importance: 0.0
Variable: Region_Code          Importance: 0.0
Variable: Vehicle_Age          Importance: 0.0
Variable: Annual_Premium       Importance: 0.0
Variable: Vintage              Importance: 0.0
```

In [66]:
```
rf_most_important = DecisionTreeClassifier(criterion='entropy', max_depth=8, m
in_samples_leaf=3, min_samples_split=7)
train_important = train_features.loc[:, ['Annual_Premium','Vintage','Age','Reg
ion_Code','Vehicle_Damage','Policy_Sales_Channel','Gender','Previously_Insure
d']]
test_important = test_features.loc[:, ['Annual_Premium','Vintage','Age','Regio
n_Code','Vehicle_Damage','Policy_Sales_Channel','Gender','Previously_Insured'
]]
rf_most_important.fit(train_important, train_labels)
predictions = rf_most_important.predict(test_important)
errors = abs(predictions - test_labels)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
```

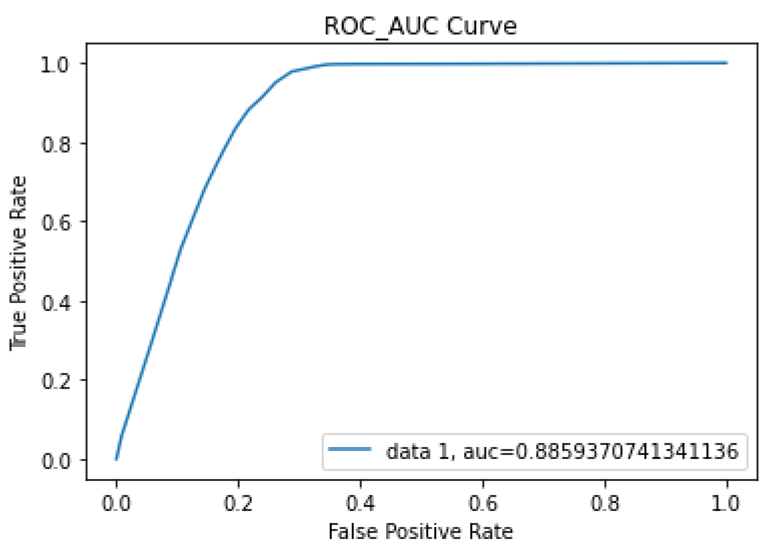
```
Mean Absolute Error: 0.09 degrees.
```

In [67]:
```
print("Accuracy:",metrics.accuracy_score(test_labels, predictions))
print("Precision:",metrics.precision_score(test_labels, predictions))
print("Recall:",metrics.recall_score(test_labels, predictions))
```

```
Accuracy: 0.9079137256959933
Precision: 0.13043478260869565
Recall: 0.00021396476713501176
```

In [68]:
```
y_pred_proba = dt.predict_proba(test_features)[::,1]
fpr, tpr, _ = metrics.roc_curve(test_labels,  y_pred_proba)
auc = metrics.roc_auc_score(test_labels, y_pred_proba)
print(auc)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.title("ROC_AUC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```

```
0.8859370741341136
```



Random Forest

In [50]:
```
import timeit
start = timeit.default_timer()
x=result_df.drop('Response', axis = 1)
y=result_df['Response']
from sklearn.model_selection import train_test_split
train_features, test_features, train_labels, test_labels = train_test_split(x,
y, test_size = 0.3, random_state = 0)
```

In [51]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 1000, random_state=42)
rf.fit(train_features, train_labels)
predictions = rf.predict(test_features)
stop = timeit.default_timer()
print('Time: ', stop - start)
```

```
Time:  1021.1808909000006
```

In [52]:
```
errors = abs(predictions - test_labels)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
print("Accuracy:",metrics.accuracy_score(test_labels, predictions))
print("Precision:",metrics.precision_score(test_labels, predictions))
print("Recall:",metrics.recall_score(test_labels, predictions))
conf = confusion_matrix(test_labels,predictions)
print(conf)
```

```
Mean Absolute Error: 0.1 degrees.
Accuracy: 0.9047781480412479
Precision: 0.39983812221772563
Recall: 0.07046572997646387
[[136940   1483]
 [ 13033    988]]
```

In [53]:
```
importances = list(rf.feature_importances_)
feature_importances = [(feature, round(importance, 2)) for feature, importance
in zip(x.columns, importances)]
feature_importances = sorted(feature_importances, key = lambda x: x[1], revers
e = True)
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_imp
ortances];
```

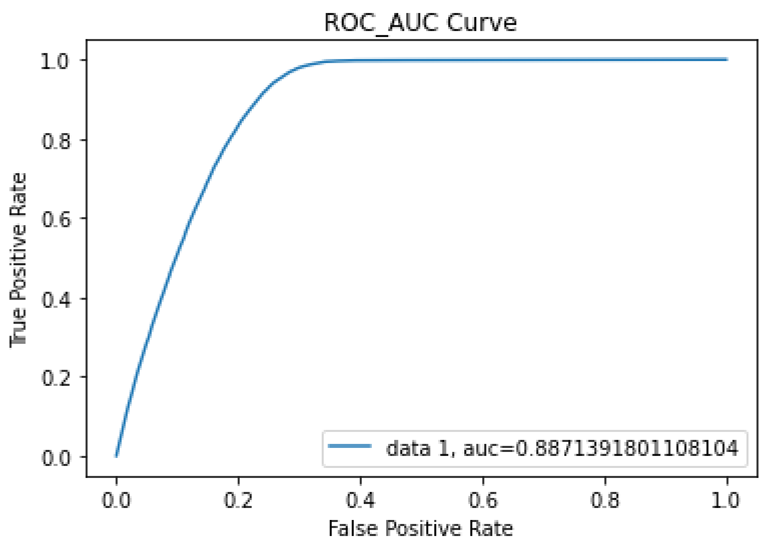
```
Variable: id                   Importance: 0.26
Variable: Vintage              Importance: 0.19
Variable: Annual_Premium       Importance: 0.17
Variable: Age                  Importance: 0.12
Variable: Region_Code          Importance: 0.08
Variable: Vehicle_Damage       Importance: 0.06
Variable: Policy_Sales_Channel Importance: 0.05
Variable: Previously_Insured   Importance: 0.04
Variable: Gender               Importance: 0.01
Variable: Vehicle_Age          Importance: 0.01
Variable: Driving_License      Importance: 0.0
```

In [54]:
```
print("Accuracy:",metrics.accuracy_score(test_labels, predictions))
print("Precision:",metrics.precision_score(test_labels, predictions))
print("Recall:",metrics.recall_score(test_labels, predictions))
```

```
Accuracy: 0.9047781480412479
Precision: 0.39983812221772563
Recall: 0.07046572997646387
```

In [55]:
```
y_pred_proba = rf.predict_proba(test_features)[::,1]
fpr, tpr, _ = metrics.roc_curve(test_labels,  y_pred_proba)
auc = metrics.roc_auc_score(test_labels, y_pred_proba)
print(auc)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.title("ROC_AUC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```

```
0.8871391801108104
```



GBM

In [87]:
```
import timeit
start = timeit.default_timer()
x=result_df.drop('Response', axis = 1)
y=result_df['Response']
from sklearn.model_selection import train_test_split
train_features, test_features, train_labels, test_labels = train_test_split(x,
y, test_size = 0.3, random_state = 0)
from imblearn.over_sampling import SMOTE
nr = SMOTE()
train_features, train_labels = nr.fit_sample(train_features, train_labels)
```

In [88]:
```
from sklearn.ensemble import GradientBoostingClassifier
gbm = GradientBoostingClassifier()
gbm.fit(train_features,train_labels)
predictions = gbm.predict(test_features)
stop = timeit.default_timer()
print('Time: ', stop - start)
```

```
Time:  97.52382639999996
```

In [ ]:
```
param ={
        "learning_rate" : [1, 0.5, 0.25, 0.1, 0.05, 0.01],
        "n_estimators" : [100, 200]}

grid_search = GridSearchCV(estimator = gbm, param_grid = param,
                          cv = 3, n_jobs = -1, verbose = 2)
grid_search.fit(train_features, train_labels)
grid_search.best_params_
```

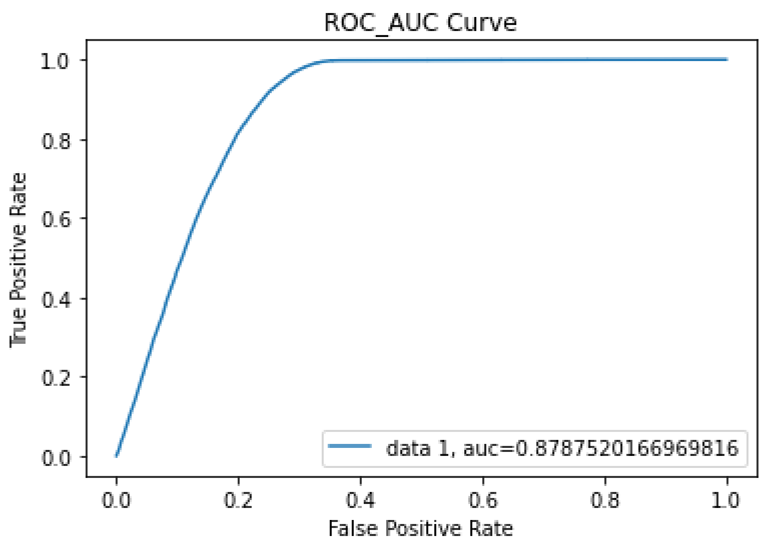
In [90]:
```
start = timeit.default_timer()
gbm = GradientBoostingClassifier(learning_rate=0.5, n_estimators=200)
gbm.fit(train_features,train_labels)
predictions = gbm.predict(test_features)
stop = timeit.default_timer()
print('Time: ', stop - start)
```

In [91]:
```
errors = abs(predictions - test_labels)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
print("Accuracy:",metrics.accuracy_score(test_labels, predictions))
print("Precision:",metrics.precision_score(test_labels, predictions))
print("Recall:",metrics.recall_score(test_labels, predictions))
conf = confusion_matrix(test_labels,predictions)
print(conf)
```

```
Mean Absolute Error: 0.15 degrees.
Accuracy: 0.8491577234919052
Precision: 0.31757196292080014
Recall: 0.5570929320305257
[[121638  16785]
 [  6210   7811]]
```

In [92]:
```
y_pred_proba = gbm.predict_proba(test_features)[::,1]
fpr, tpr, _ = metrics.roc_curve(test_labels,  y_pred_proba)
auc = metrics.roc_auc_score(test_labels, y_pred_proba)
print(auc)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.title("ROC_AUC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```

```
0.8787520166969816
```



XGM

In [75]:
```
import timeit
start = timeit.default_timer()
x=result_df.drop('Response', axis = 1)
y=result_df['Response']
from sklearn.model_selection import train_test_split
train_features, test_features, train_labels, test_labels = train_test_split(x,
y, test_size = 0.3, random_state = 0)
from imblearn.over_sampling import SMOTE
nr = SMOTE()
train_features, train_labels = nr.fit_sample(train_features, train_labels)
```

In [84]:
```
from xgboost import XGBClassifier as xgb
model_xgb = xgb()
model_xgb.fit(train_features,train_labels)
best_preds = model_xgb.predict(test_features)
stop = timeit.default_timer()
print('Time: ', stop - start)
```

```
Time:  3162.954396
```

In [ ]:
```
from sklearn.model_selection import GridSearchCV
parameters = {
     "eta"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
     "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15]}

grid = GridSearchCV(model_xgb,
                    parameters, n_jobs=-1,
                    scoring="neg_log_loss",
                    cv=3, verbose=2)
grid.fit(train_features, train_labels)
grid.best_params_
```

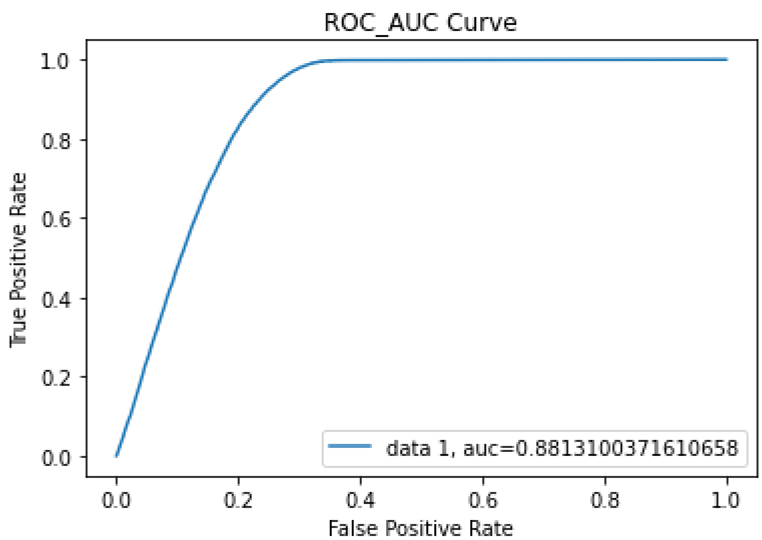
In [ ]:
```
start = timeit.default_timer()
model_xgb = xgb(eta=0.05, max_depth=12)
model_xgb.fit(train_features,train_labels)
best_preds = model_xgb.predict(test_features)
stop = timeit.default_timer()
print('Time: ', stop - start)
```

In [85]:
```
errors = abs(best_preds - test_labels)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
print("Accuracy:",metrics.accuracy_score(test_labels, best_preds))
print("Precision:",metrics.precision_score(test_labels, best_preds))
print("Recall:",metrics.recall_score(test_labels, best_preds))
conf = confusion_matrix(test_labels,best_preds)
print(conf)
```

```
Mean Absolute Error: 0.15 degrees.
Accuracy: 0.8471897877253286
Precision: 0.3198943525207799
Recall: 0.5874046073746523
[[120913  17510]
 [  5785   8236]]
```

In [86]:
```
y_pred_proba = model_xgb.predict_proba(test_features)[::,1]
fpr, tpr, _ = metrics.roc_curve(test_labels,  y_pred_proba)
auc = metrics.roc_auc_score(test_labels, y_pred_proba)
print(auc)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.title("ROC_AUC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```

```
0.8813100371610658
```



Neural Network Classifier

In [1]:
```
from keras import Sequential
from keras.layers import Dense
```

In [28]:
```
x=result_df.drop('Response', axis = 1)
y=result_df['Response']
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x = sc.fit_transform(x)
```

In [29]:
```
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
train_features, test_features, train_labels, test_labels = train_test_split(x,
y, test_size = 0.3)
nr = SMOTE()
train_features, train_labels = nr.fit_sample(train_features, train_labels)
```

In [12]:
```
print('Training Features Shape:', train_features.shape)
print('Training Labels Shape:', train_labels.shape)
print('Testing Features Shape:', test_features.shape)
print('Testing Labels Shape:', test_labels.shape)
```

```
Training Features Shape: (645800, 11)
Training Labels Shape: (645800,)
Testing Features Shape: (152444, 11)
Testing Labels Shape: (152444,)
```

In [13]:
```
def build_model():
    classifier = Sequential()
    classifier.add(Dense(6, activation='relu', kernel_initializer='random_norm
al', input_dim=11))
    classifier.add(Dense(6, activation='relu', kernel_initializer='random_norm
al'))
    classifier.add(Dense(1, activation='sigmoid', kernel_initializer='random_n
ormal'))
    classifier.compile(optimizer ='adam',loss='binary_crossentropy', metrics =
['accuracy'])
    return classifier
```

In [14]:
```
keras_model = build_model()
keras_model.fit(train_features,train_labels, batch_size=64, epochs=100)
```

```
Epoch 1/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3795 -
accuracy: 0.8377
Epoch 2/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3604 -
accuracy: 0.8446
Epoch 3/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3580 -
accuracy: 0.8448
Epoch 4/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3566 -
accuracy: 0.8451
Epoch 5/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3554 -
accuracy: 0.8454
Epoch 6/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3547 -
accuracy: 0.8454
Epoch 7/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3543 -
accuracy: 0.8453
Epoch 8/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3540 -
accuracy: 0.8453
Epoch 9/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3540 -
accuracy: 0.8453
Epoch 10/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3539 -
accuracy: 0.8454
Epoch 11/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3538 -
accuracy: 0.8455
Epoch 12/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3537 -
accuracy: 0.8454
Epoch 13/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3537 -
accuracy: 0.8453
Epoch 14/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3536 -
accuracy: 0.8454
Epoch 15/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3535 -
accuracy: 0.8455
Epoch 16/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3534 -
accuracy: 0.8457
Epoch 17/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3530 -
accuracy: 0.8456
Epoch 18/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3528 -
accuracy: 0.8458
Epoch 19/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3527 -
accuracy: 0.8459
```

```
Epoch 20/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3526 -
accuracy: 0.8459
Epoch 21/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3527 -
accuracy: 0.8460
Epoch 22/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3525 -
accuracy: 0.8460
Epoch 23/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3524 -
accuracy: 0.8460
Epoch 24/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3523 -
accuracy: 0.8462
Epoch 25/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3523 -
accuracy: 0.8461
Epoch 26/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3523 -
accuracy: 0.8461
Epoch 27/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3521 -
accuracy: 0.8461
Epoch 28/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3520 -
accuracy: 0.8462
Epoch 29/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3519 -
accuracy: 0.8464
Epoch 30/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3517 -
accuracy: 0.8464
Epoch 31/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3515 -
accuracy: 0.8467
Epoch 32/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3515 -
accuracy: 0.8464
Epoch 33/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3513 -
accuracy: 0.8465
Epoch 34/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3514 -
accuracy: 0.8466
Epoch 35/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3511 -
accuracy: 0.8465
Epoch 36/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3510 -
accuracy: 0.8468
Epoch 37/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3510 -
accuracy: 0.8467
Epoch 38/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3508 -
accuracy: 0.8469
```

```
Epoch 39/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3507 -
accuracy: 0.8472
Epoch 40/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3506 -
accuracy: 0.8471
Epoch 41/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3506 -
accuracy: 0.8471
Epoch 42/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3506 -
accuracy: 0.8471
Epoch 43/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3505 -
accuracy: 0.8470
Epoch 44/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3505 -
accuracy: 0.8472
Epoch 45/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3504 -
accuracy: 0.8471
Epoch 46/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3505 -
accuracy: 0.8471
Epoch 47/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3504 -
accuracy: 0.8471
Epoch 48/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3504 -
accuracy: 0.8473
Epoch 49/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3504 -
accuracy: 0.8472
Epoch 50/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3504 -
accuracy: 0.8471
Epoch 51/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3504 -
accuracy: 0.8472
Epoch 52/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3503 -
accuracy: 0.8472
Epoch 53/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3504 -
accuracy: 0.8472
Epoch 54/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3503 -
accuracy: 0.8472
Epoch 55/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3503 -
accuracy: 0.8472
Epoch 56/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3503 -
accuracy: 0.8473
Epoch 57/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3502 -
accuracy: 0.8474
```

```
Epoch 58/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3502 -
accuracy: 0.8472
Epoch 59/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3502 -
accuracy: 0.8473
Epoch 60/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3503 -
accuracy: 0.8471
Epoch 61/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3502 -
accuracy: 0.8473
Epoch 62/100
10091/10091 [==============================] - ETA: 0s - loss: 0.3502 - accur
acy: 0.84 - 17s 2ms/step - loss: 0.3502 - accuracy: 0.8473
Epoch 63/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3503 -
accuracy: 0.8473
Epoch 64/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3502 -
accuracy: 0.8474
Epoch 65/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3502 -
accuracy: 0.8472
Epoch 66/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3501 -
accuracy: 0.8473
Epoch 67/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3502 -
accuracy: 0.8473
Epoch 68/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3501 -
accuracy: 0.8474
Epoch 69/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3502 -
accuracy: 0.8474
Epoch 70/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3502 -
accuracy: 0.8473
Epoch 71/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3502 -
accuracy: 0.8471
Epoch 72/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3500 -
accuracy: 0.8474
Epoch 73/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3501 -
accuracy: 0.8473
Epoch 74/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3501 -
accuracy: 0.8473
Epoch 75/100
10091/10091 [==============================] - 16s 2ms/step - loss: 0.3501 -
accuracy: 0.8474
Epoch 76/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3501 -
accuracy: 0.8472
```

```
Epoch 77/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3500 -
accuracy: 0.8473
Epoch 78/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3501 -
accuracy: 0.8473
Epoch 79/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3501 -
accuracy: 0.8472
Epoch 80/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3501 -
accuracy: 0.8474
Epoch 81/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3500 -
accuracy: 0.8473
Epoch 82/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3500 -
accuracy: 0.8474
Epoch 83/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3501 -
accuracy: 0.8474
Epoch 84/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3500 -
accuracy: 0.8474
Epoch 85/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3500 -
accuracy: 0.8472
Epoch 86/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3500 -
accuracy: 0.8474
Epoch 87/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3499 -
accuracy: 0.8475
Epoch 88/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3499 -
accuracy: 0.8472
Epoch 89/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3499 -
accuracy: 0.8474
Epoch 90/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3499 -
accuracy: 0.8474
Epoch 91/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3499 -
accuracy: 0.8474
Epoch 92/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3499 -
accuracy: 0.8473
Epoch 93/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3499 -
accuracy: 0.8475
Epoch 94/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3499 -
accuracy: 0.8473
Epoch 95/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3499 -
accuracy: 0.8474
```

```
Epoch 96/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3499 -
accuracy: 0.8472
Epoch 97/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3499 -
accuracy: 0.8473
Epoch 98/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3499 -
accuracy: 0.8472
Epoch 99/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3499 -
accuracy: 0.8473
Epoch 100/100
10091/10091 [==============================] - 17s 2ms/step - loss: 0.3499 -
accuracy: 0.8474
```

Out[14]:

```
<tensorflow.python.keras.callbacks.History at 0x122c3422670>
```

In [30]:

```
eval_model=keras_model.evaluate(train_features, train_labels)
eval_model
```

```
20176/20176 [==============================] - 40s 2ms/step - loss: 0.3513 -
accuracy: 0.8465
```

Out[30]:

```
[0.35126250982284546, 0.8464527726173401]
```

In [31]:

```
y_pred=keras_model.predict(test_features)
y_pred =(y_pred>0.5)
```

In [32]:

```
print("Accuracy:",metrics.accuracy_score(test_labels, y_pred))
print("Precision:",metrics.precision_score(test_labels, y_pred))
print("Recall:",metrics.recall_score(test_labels, y_pred))
cm = confusion_matrix(test_labels, y_pred)
print(cm)
```

```
Accuracy: 0.7590918632415838
Precision: 0.26553637610827857
Recall: 0.9380787037037037
[[102751  35869]
 [   856  12968]]
```

In [33]:
```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.metrics import auc
from sklearn.metrics import roc_curve
y_pred_keras = keras_model.predict(test_features).ravel()
fpr_keras, tpr_keras, thresholds_keras = roc_curve(test_labels, y_pred_keras)
auc_keras = auc(fpr_keras, tpr_keras)
plt.plot(fpr_keras,tpr_keras,label="data 1, auc="+str(auc_keras))
plt.legend(loc=4)
plt.title("ROC_AUC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```