Vehicle Insurance Claim Prediction using ML Algorithms

The Dataset was taken from (https://www.kaggle.com/competitions/actuarial-loss-estimation/data?select=test.csv (https://www.kaggle.com/competitions/actuarial-loss-estimation/data?select=test.csv)) which has 90,000 realistic, synthetically generated worker compensation insurance policies in the dataset, all of which have been involved in an accident. There is demographic and worker information, as well as a text description of the accident, for each report.

Train.csv - The training set containing 54,000 insurance policies to train the model.

Test.csv - The Test dataset to Predict total payment of insurance claim as final result.

Sample_submission.csv - sample file for submission in the correct format

Loading the Required libraries

In [ ]:

```
import pandas as pd # data analytical library
import matplotlib.pyplot as plt #visualization
import seaborn as sns #statistical visualization
```

Reading the Vehicle Insurance test and train dataset

In [ ]:

```
Train_Dataset=pd.read_csv("/content/Train.csv")  #loaded the train dataset
Test_Dataset=pd.read_csv("/content/Test.csv") #loaded the test dataset
```

In [ ]:

```
#Checked the dataframe for training dataset
Train_Dataset.head()
```

Out[ ]:

	ClaimNumber	DateTimeOfAccident	DateReported	Age	Gender	MaritalStatus	Dependent(
0	WC8285054	2002-04-09T07:00:00Z	2002-07-05T00:00:00Z	48	M	M	
1	WC6982224	1999-01-07T11:00:00Z	1999-01-20T00:00:00Z	43	F	M	
2	WC5481426	1996-03-25T00:00:00Z	1996-04-14T00:00:00Z	30	M	U	
3	WC9775968	2005-06-22T13:00:00Z	2005-07-22T00:00:00Z	41	M	S	
4	WC2634037	1990-08-29T08:00:00Z	1990-09-27T00:00:00Z	36	M	M	

Renaming the column names in train dataset headings as it is "unnamed"

In [ ]:

```
Train_Dataset=Train_Dataset.rename(columns={"Unnamed: 0":"ClaimNumber","Unnamed: 1":"Da
teTimeOfAccident","Unnamed: 3":"Age","Unnamed: 4":"Gender",
                      "Unnamed: 5":"MaritalStatus","Unnamed: 6":"DependentChildren","Un
named: 8":"WeeklyWages",
                      "Unnamed: 9":"PartTimeFullTime","Unnamed: 10":"HoursWorkedPerWee
k","Unnamed: 12":"ClaimDescription",
                      "Unnamed: 13":"InitialIncurredCalimsCost","Unnamed: 14":'Ultimate
IncurredClaimCost'},inplace=False)
```

Dropping the first row as it has the few repeated column names in train dataset

In [ ]:

```
Train_Dataset=Train_Dataset.drop(Train_Dataset.index[0])
Train_Dataset.head()
```

Out[ ]:

	ClaimNumber	DateTimeOfAccident	DateReported	Age	Gender	MaritalStatus	Dependent(
1	WC6982224	1999-01-07T11:00:00Z	1999-01-20T00:00:00Z	43	F	M	
2	WC5481426	1996-03-25T00:00:00Z	1996-04-14T00:00:00Z	30	M	U	
3	WC9775968	2005-06-22T13:00:00Z	2005-07-22T00:00:00Z	41	M	S	
4	WC2634037	1990-08-29T08:00:00Z	1990-09-27T00:00:00Z	36	M	M	
5	WC6828422	1999-06-21T11:00:00Z	1999-09-09T00:00:00Z	50	M	M	

Data Frame Summary

In [ ]:

```
#summary of training data
Train_Dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53999 entries, 1 to 53999
Data columns (total 15 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   ClaimNumber                53999 non-null  object 
 1   DateTimeOfAccident         53999 non-null  object 
 2   DateReported               53999 non-null  object 
 3   Age                        53999 non-null  int64  
 4   Gender                     53999 non-null  object 
 5   MaritalStatus              53970 non-null  object 
 6   DependentChildren          53999 non-null  int64  
 7   DependentsOther            53999 non-null  int64  
 8   WeeklyWages                53999 non-null  float64
 9   PartTimeFullTime           53999 non-null  object 
 10  HoursWorkedPerWeek         53999 non-null  float64
 11  DaysWorkedPerWeek          53999 non-null  int64  
 12  ClaimDescription           53999 non-null  object 
 13  InitialIncurredCalimsCost  53999 non-null  int64  
 14  UltimateIncurredClaimCost  53999 non-null  float64
dtypes: float64(3), int64(5), object(7)
memory usage: 6.6+ MB
```

In [ ]:

```
#summary of test data
Test_Dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36000 entries, 0 to 35999
Data columns (total 14 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   ClaimNumber                36000 non-null  object 
 1   DateTimeOfAccident         36000 non-null  object 
 2   DateReported               36000 non-null  object 
 3   Age                        36000 non-null  int64  
 4   Gender                     36000 non-null  object 
 5   MaritalStatus              35982 non-null  object 
 6   DependentChildren          36000 non-null  int64  
 7   DependentsOther            36000 non-null  int64  
 8   WeeklyWages                36000 non-null  float64
 9   PartTimeFullTime           36000 non-null  object 
 10  HoursWorkedPerWeek         36000 non-null  float64
 11  DaysWorkedPerWeek          36000 non-null  int64  
 12  ClaimDescription           36000 non-null  object 
 13  InitialIncurredCalimsCost  36000 non-null  int64  
dtypes: float64(2), int64(5), object(7)
memory usage: 3.8+ MB
```

In [ ]:

```
#Checking for Categorical Data based on datatypes in train data
Train_Dataset.select_dtypes(exclude=['int64','float64']).columns
```

Out[ ]:

```
Index(['ClaimNumber', 'DateTimeOfAccident', 'DateReported', 'Gender',
       'MaritalStatus', 'PartTimeFullTime', 'ClaimDescription'],
      dtype='object')
```

Changing the data type for required columns in train data (Converting to Numeric Data)

In [ ]:

```
Train_Dataset['Age'] = pd.to_numeric(Train_Dataset['Age'])
Train_Dataset['DependentChildren'] = pd.to_numeric(Train_Dataset['DependentChildren'])
Train_Dataset['DependentsOther'] = pd.to_numeric(Train_Dataset['DependentsOther'])
Train_Dataset['WeeklyWages'] = pd.to_numeric(Train_Dataset['WeeklyWages'])
Train_Dataset['HoursWorkedPerWeek'] = pd.to_numeric(Train_Dataset['HoursWorkedPerWeek'
])
Train_Dataset['DaysWorkedPerWeek'] = pd.to_numeric(Train_Dataset['DaysWorkedPerWeek'])
Train_Dataset['InitialIncurredCalimsCost'] = pd.to_numeric(Train_Dataset['InitialIncurr
edCalimsCost'])
Train_Dataset['UltimateIncurredClaimCost'] = pd.to_numeric(Train_Dataset['UltimateIncur
redClaimCost'])
```

To check Updated Dtype - if the data type changed or not

In [ ]:

```
Train_Dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53999 entries, 1 to 53999
Data columns (total 15 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   ClaimNumber                53999 non-null  object
 1   DateTimeOfAccident         53999 non-null  object
 2   DateReported               53999 non-null  object
 3   Age                        53999 non-null  int64
 4   Gender                     53999 non-null  object
 5   MaritalStatus              53970 non-null  object
 6   DependentChildren          53999 non-null  int64
 7   DependentsOther            53999 non-null  int64
 8   WeeklyWages                53999 non-null  float64
 9   PartTimeFullTime           53999 non-null  object
 10  HoursWorkedPerWeek         53999 non-null  float64
 11  DaysWorkedPerWeek          53999 non-null  int64
 12  ClaimDescription           53999 non-null  object
 13  InitialIncurredCalimsCost  53999 non-null  int64
 14  UltimateIncurredClaimCost  53999 non-null  float64
dtypes: float64(3), int64(5), object(7)
memory usage: 6.6+ MB
```

In [ ]:

```
#checking the description of the train data
Train_Dataset.describe()
```

Out[ ]:

	Age	DependentChildren	DependentsOther	WeeklyWages	HoursWorkedPerWe
count	53999.000000	53999.000000	53999.000000	53999.000000	53999.00C
mean	33.842108	0.119187	0.009945	416.363259	37.735
std	12.122124	0.517785	0.109349	248.640710	12.568
min	13.000000	0.000000	0.000000	1.000000	0.00C
25%	23.000000	0.000000	0.000000	200.000000	38.00C
50%	32.000000	0.000000	0.000000	392.200000	38.00C
75%	43.000000	0.000000	0.000000	500.000000	40.00C
max	81.000000	9.000000	5.000000	7497.000000	640.00C

Working on the train data

Checking the shape of dataset (Number of Rows and Columns)

In [ ]:

```
Train_Dataset.shape
```

Out[ ]:

```
(53999, 15)
```

Checking for duplicate values

In [ ]:

```
Train_Dataset.duplicated().sum()
```

Out[ ]:

```
0
```

Checking for missing values

In [ ]:

```
Train_Dataset.isnull().sum()
```

Out[ ]:

```
ClaimNumber                 0
DateTimeOfAccident          0
DateReported                0
Age                         0
Gender                      0
MaritalStatus              29
DependentChildren           0
DependentsOther             0
WeeklyWages                 0
PartTimeFullTime            0
HoursWorkedPerWeek          0
DaysWorkedPerWeek           0
ClaimDescription            0
InitialIncurredCalimsCost   0
UltimateIncurredClaimCost   0
dtype: int64
```

Using mean and mode imputation to treating the missing values

In [ ]:

```
#Train_Dataset['WeeklyWages']=Train_Dataset['WeeklyWages'].fillna(Train_Dataset['WeeklyWages'].mean())
#Train_Dataset['HoursWorkedPerWeek']=Train_Dataset['HoursWorkedPerWeek'].fillna(Train_Dataset['HoursWorkedPerWeek'].mean())
Train_Dataset['MaritalStatus']=Train_Dataset['MaritalStatus'].fillna(Train_Dataset['MaritalStatus'].mode()[0])
```

In [ ]:

```
#To verify if there are any more missing values
Train_Dataset.isnull().sum()
```

Out[ ]:

```
ClaimNumber                  0
DateTimeOfAccident           0
DateReported                 0
Age                          0
Gender                       0
MaritalStatus                0
DependentChildren            0
DependentsOther              0
WeeklyWages                  0
PartTimeFullTime             0
HoursWorkedPerWeek           0
DaysWorkedPerWeek            0
ClaimDescription             0
InitialIncurredCalimsCost    0
UltimateIncurredClaimCost    0
dtype: int64
```

Finally, there are no missing values in Train Dataset

Dividing the data into categorical and numerical data

In [ ]:

```
Train_Dataset_num=Train_Dataset[{'Age','DependentChildren', 'DependentsOther', 'WeeklyW
ages', 'HoursWorkedPerWeek', 'DaysWorkedPerWeek',
           'InitialIncurredCalimsCost','UltimateIncurredClaimCost', }]
Train_Dataset_cat=Train_Dataset[{'ClaimNumber', 'DateTimeOfAccident', 'DateReported','G
ender' ,'MaritalStatus','PartTimeFullTime','ClaimDescription'}]
```

For understanding categorical data

In [ ]:

```
Train_Dataset['MaritalStatus'].value_counts()
```

Out[ ]:

```
S    26190
M    22515
U     5294
Name: MaritalStatus, dtype: int64
```

In [ ]:

```
Train_Dataset['Gender'].value_counts()
```

Out[ ]:

```
M    41659
F    12338
U        2
Name: Gender, dtype: int64
```

In [ ]:

```
Train_Dataset['PartTimeFullTime'].value_counts()
```

Out[ ]:

```
F    49111
P     4888
Name: PartTimeFullTime, dtype: int64
```

In [ ]:

```
Train_Dataset['ClaimDescription'].nunique()
```

Out[ ]:

```
28113
```

There are 28113 unique claims made.

In [ ]:

```
Train_Dataset.ClaimNumber.count()
```

Out[ ]:

```
53999
```

The total number of claims filed is 53999.

In [ ]:

```
Train_Dataset['ClaimNumber'].nunique()
```

Out[ ]:

```
53999
```

The total number of claims that were filed is 53999 and all number of unique claims are Unique.

Data Transformation

Data binning

In [ ]:

```
Train_Dataset['Age'].value_counts
```

Out[ ]:

```
<bound method IndexOpsMixin.value_counts of 1        43
2        30
3        41
4        36
5        50
         ..
53995    32
53996    20
53997    19
53998    24
53999    22
Name: Age, Length: 53999, dtype: int64>
```

In [ ]:

```
Train_Dataset['Age'].min()
```

Out[ ]:

```
13
```

In [ ]:

```
Train_Dataset['Age'].max()
```

Out[ ]:

```
81
```

In [ ]:

```
Train_Dataset['Age_Bin']=pd.cut(Train_Dataset['Age'],bins=[1,25,50,80] , labels=['Youn
g','Middle-Age','Old'])
Train_Dataset['Age_Bin']
```

Out[ ]:

```
1        Middle-Age
2        Middle-Age
3        Middle-Age
4        Middle-Age
5        Middle-Age
            ...
53995    Middle-Age
53996         Young
53997         Young
53998         Young
53999         Young
Name: Age_Bin, Length: 53999, dtype: category
Categories (3, object): ['Young' < 'Middle-Age' < 'Old']
```

In [ ]:

```
Train_Dataset['WeeklyWages'].value_counts
```

Out[ ]:

```
<bound method IndexOpsMixin.value_counts of 1        509.34
2        709.10
3        555.46
4        377.10
5        200.00
          ...
53995    500.00
53996    500.00
53997    283.00
53998    200.00
53999    200.00
Name: WeeklyWages, Length: 53999, dtype: float64>
```

In [ ]:

```
Train_Dataset['WeeklyWages'].max()
```

Out[ ]:

```
7497.0
```

In [ ]:

```
Train_Dataset['WeeklyWages'].min()
```

Out[ ]:

```
1.0
```

In [ ]:

```
Train_Dataset['WeeklyWages_Bin']=pd.cut(Train_Dataset['WeeklyWages'],bins=[0,1000,2000,
4000,7000,8000] , labels=['Low','Below Average','Average Wage','Above Average','High'])
Train_Dataset['WeeklyWages_Bin']
```

Out[ ]:

```
1        Low
2        Low
3        Low
4        Low
5        Low
        ...
53995    Low
53996    Low
53997    Low
53998    Low
53999    Low
Name: WeeklyWages_Bin, Length: 53999, dtype: category
Categories (5, object): ['Low' < 'Below Average' < 'Average Wage' < 'Above
Average' < 'High']
```

Exploratory Data Analysis :-

1. Univariate Analysis

In [ ]:

```
#Checking the target variable
Train_Dataset['UltimateIncurredClaimCost'].describe()
```

Out[ ]:

```
count    5.399900e+04
mean     1.100349e+04
std      3.339129e+04
min      1.218868e+02
25%      9.263174e+02
50%      3.371116e+03
75%      8.197288e+03
max      4.027136e+06
Name: UltimateIncurredClaimCost, dtype: float64
```

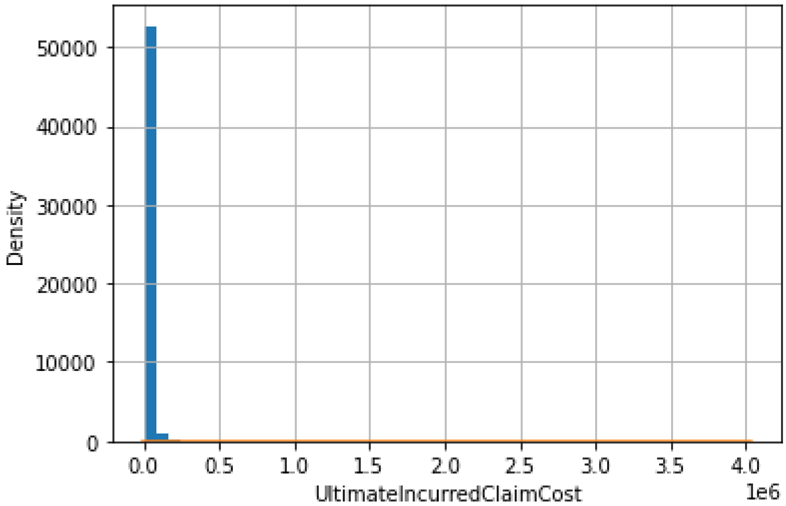
In [ ]:

```
#Checking the skewness of the target variable
Train_Dataset['UltimateIncurredClaimCost'].hist(bins=50)
sns.distplot(Train_Dataset['UltimateIncurredClaimCost'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Futu
reWarning: `distplot` is a deprecated function and will be removed in a fu
ture version. Please adapt your code to use either `displot` (a figure-lev
el function with similar flexibility) or `histplot` (an axes-level functio
n for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f75ae39ef10>
```



[link text (https://)](https://)##### The data for UltimateIncurredClaimCost is right skewed.

In [ ]:

```
Train_Dataset['InitialIncurredCalimsCost'].describe()
```

Out[ ]:

```
count    5.399900e+04
mean     7.841263e+03
std      2.058425e+04
min      1.000000e+00
25%      7.000000e+02
50%      2.000000e+03
75%      9.500000e+03
max      2.000000e+06
Name: InitialIncurredCalimsCost, dtype: float64
```

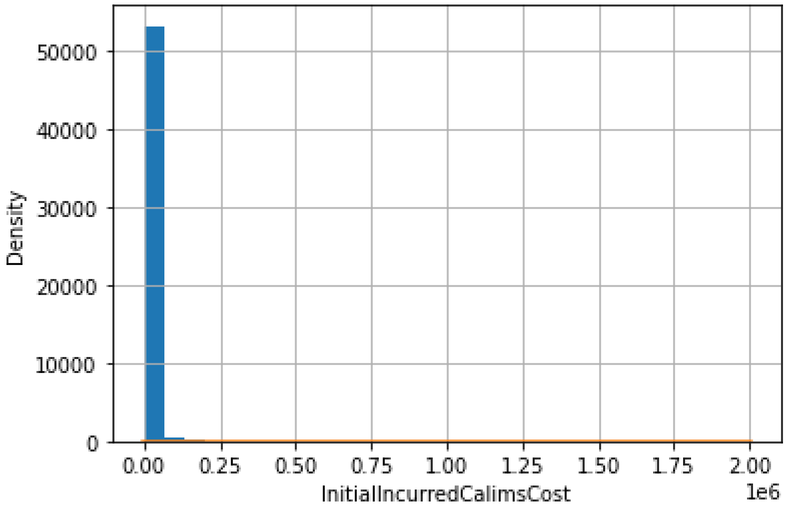
In [ ]:

```
Train_Dataset['InitialIncurredCalimsCost'].hist(bins=30)
sns.distplot(Train_Dataset['InitialIncurredCalimsCost'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Futu
reWarning: `distplot` is a deprecated function and will be removed in a fu
ture version. Please adapt your code to use either `displot` (a figure-lev
el function with similar flexibility) or `histplot` (an axes-level functio
n for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f75ae1e0e50>
```



The data for InitialIncurredClaimCost is right skewed.

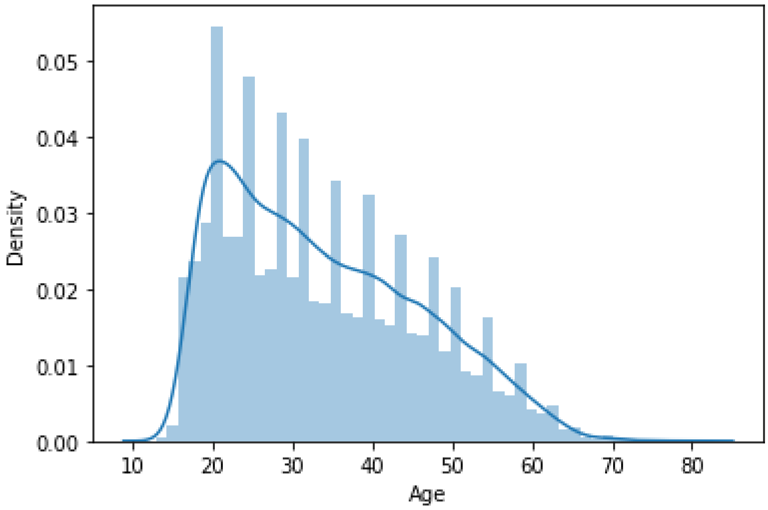
In [ ]:

```
sns.distplot(Train_Dataset['Age'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Futu
reWarning: `distplot` is a deprecated function and will be removed in a fu
ture version. Please adapt your code to use either `displot` (a figure-lev
el function with similar flexibility) or `histplot` (an axes-level functio
n for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f75a991a550>
```

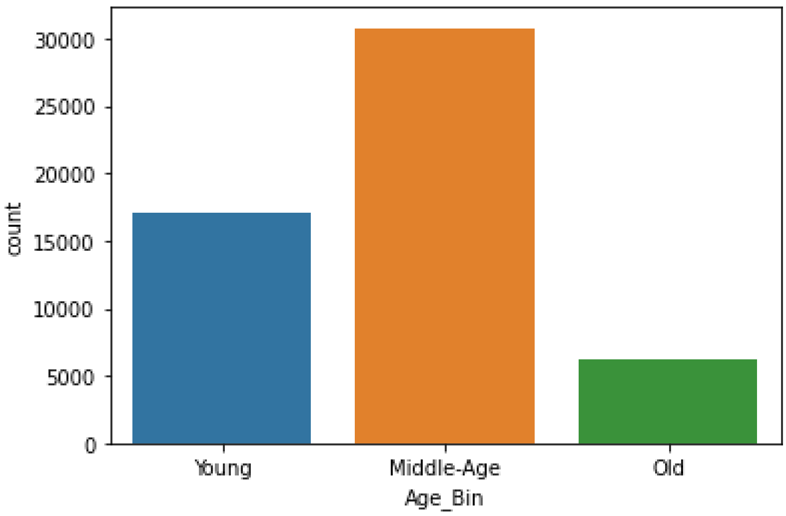


In [ ]:

```
sns.countplot(x = 'Age_Bin', data = Train_Dataset)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f75a91d5350>
```



In [ ]:

```
Train_Dataset['Age_Bin'].value_counts(normalize=True)*100
```

Out[ ]:

```
Middle-Age    56.944702
Young         31.638209
Old           11.417090
Name: Age_Bin, dtype: float64
```

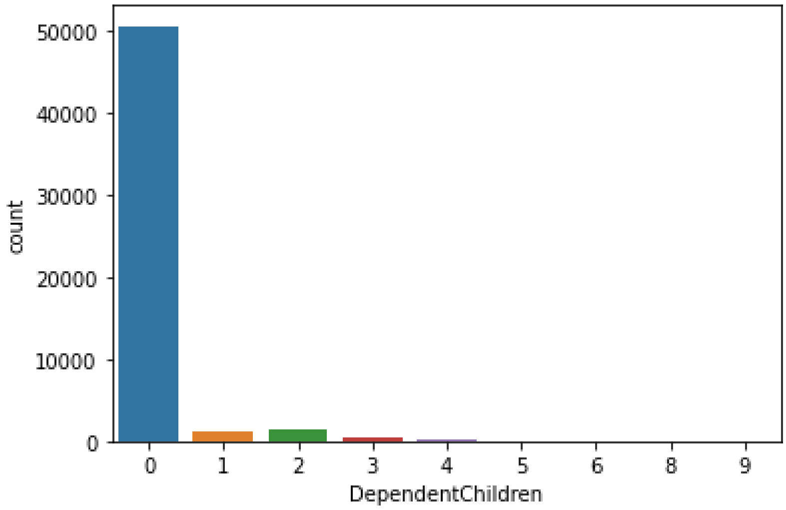
From both the plots we can see,when compare with Yound and old age,the claims are higher from the middle age group (25-40) which is close to 57%

In [ ]:

```
sns.countplot(x = 'DependentChildren', data = Train_Dataset)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f75a9159690>
```



In [ ]:

```
Train_Dataset['DependentChildren'].value_counts(normalize=True)*100
```

Out[ ]:

```
0    93.775811
2     2.520417
1     2.357451
3     0.977796
4     0.277783
5     0.077779
6     0.009259
9     0.001852
8     0.001852
Name: DependentChildren, dtype: float64
```

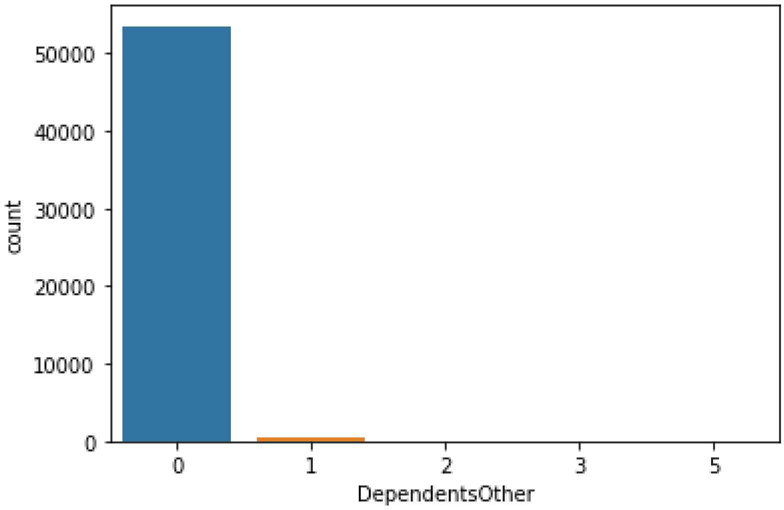
The claims made from people with no(zero) children as dependents is really high(close to 94%).

In [ ]:

```
sns.countplot(x = 'DependentsOther', data = Train_Dataset)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f75a9141c90>
```



In [ ]:

```
Train_Dataset['DependentsOther'].value_counts(normalize=True)*100
```

Out[ ]:

```
0    99.085168
1     0.855571
2     0.042593
3     0.014815
5     0.001852
Name: DependentsOther, dtype: float64
```

The claims made from people with no other dependents is higher than people with other dependents (morethan 99%).

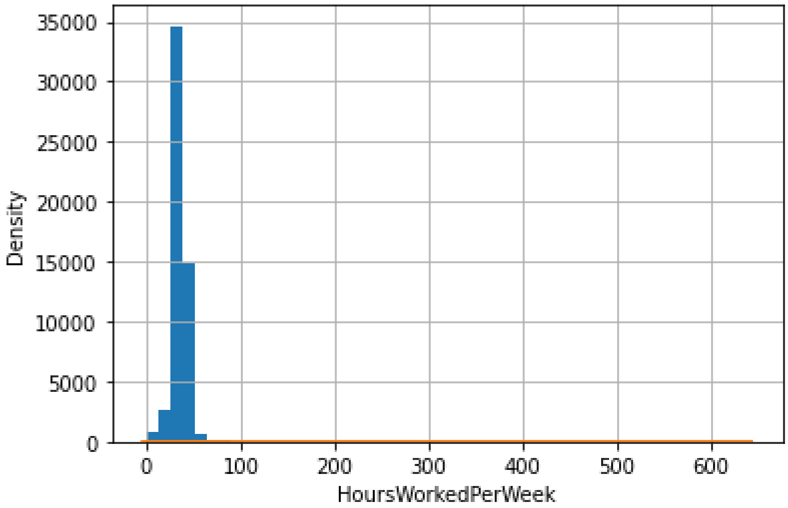
In [ ]:

```
Train_Dataset['HoursWorkedPerWeek'].hist(bins=50)
sns.distplot(Train_Dataset['HoursWorkedPerWeek'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Futu
reWarning: `distplot` is a deprecated function and will be removed in a fu
ture version. Please adapt your code to use either `displot` (a figure-lev
el function with similar flexibility) or `histplot` (an axes-level functio
n for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f75a90b5810>
```



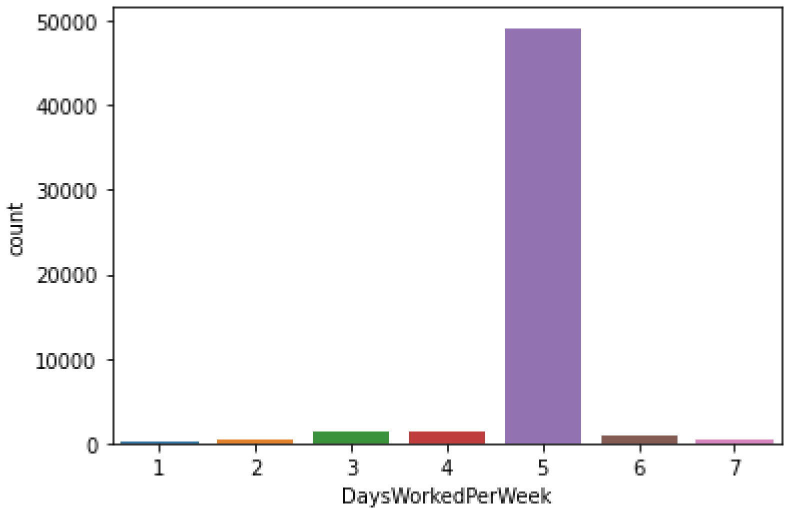
The data for HoursWorkedPerWeek is positively skewed.

In [ ]:

```
sns.countplot(x = 'DaysWorkedPerWeek', data = Train_Dataset)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f75a9008d50>
```



In [ ]:

```
Train_Dataset['DaysWorkedPerWeek'].value_counts(normalize=True)*100
```

Out[ ]:

```
5    91.083168
4     2.733384
3     2.659309
6     1.637067
2     0.950018
7     0.598159
1     0.338895
Name: DaysWorkedPerWeek, dtype: float64
```

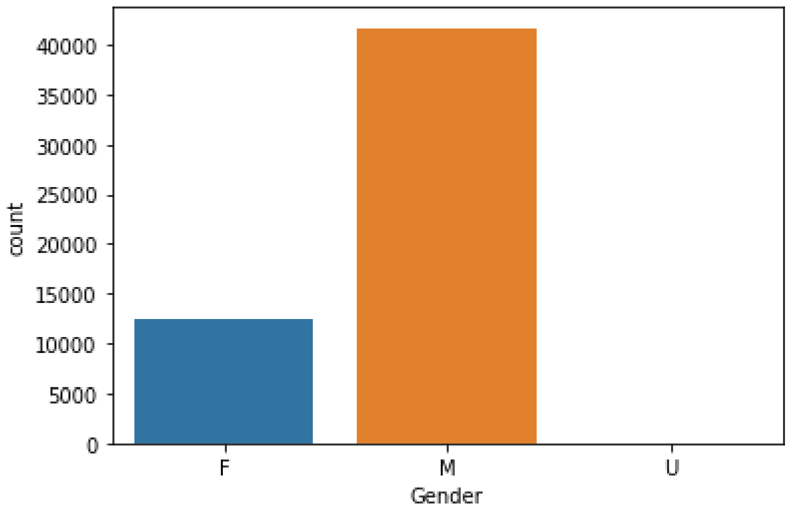
From the above plot states that 91% of the 5days working (5 days/week)people from our dataset have claimed for insurance.

In [ ]:

```
sns.countplot(x = 'Gender', data = Train_Dataset)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f75a8ec6510>
```



In [ ]:

```
Train_Dataset['Gender'].value_counts(normalize=True)*100
```

Out[ ]:

```
M    77.147725
F    22.848571
U     0.003704
Name: Gender, dtype: float64
```

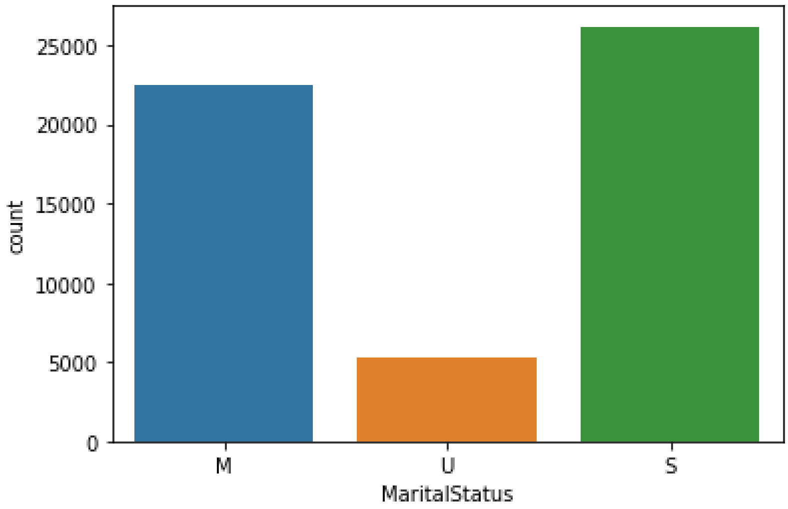
From the above plot we can see more than 77% males gender claimed insurance compare with females(22%).

In [ ]:

```
sns.countplot(x = 'MaritalStatus', data = Train_Dataset)
```

Out[ ]:

```
&lt;matplotlib.axes._subplots.AxesSubplot at 0x7f75a8e4d6d0&gt;
```



In [ ]:

```
Train_Dataset['MaritalStatus'].value_counts(normalize=True)*100
```

Out[ ]:

```
S    48.500898
M    41.695217
U     9.803885
Name: MaritalStatus, dtype: float64
```

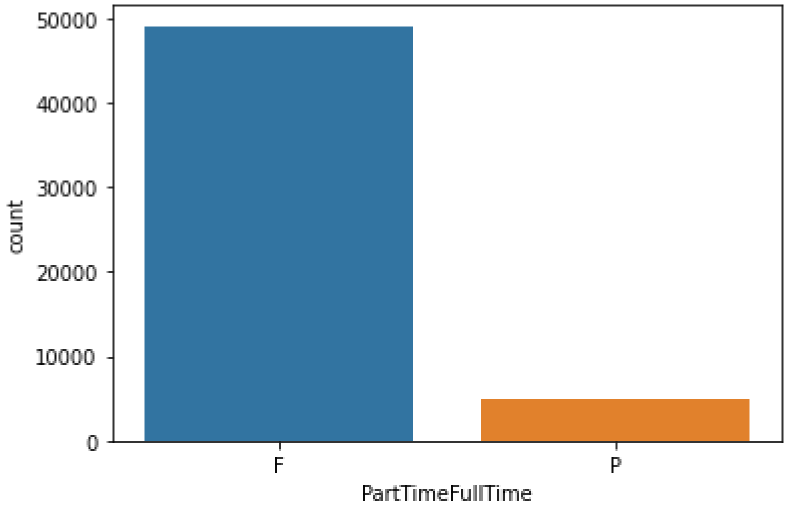
From the above plot, we can say almost 48% Single people claimed for insurance with compare with married and unmarried.

In [ ]:

```
sns.countplot(x = 'PartTimeFullTime', data = Train_Dataset)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f75a9335210>
```



In [ ]:

```
Train_Dataset['PartTimeFullTime'].value_counts(normalize=True)*100
```

Out[ ]:

```
F    90.947981
P     9.052019
Name: PartTimeFullTime, dtype: float64
```

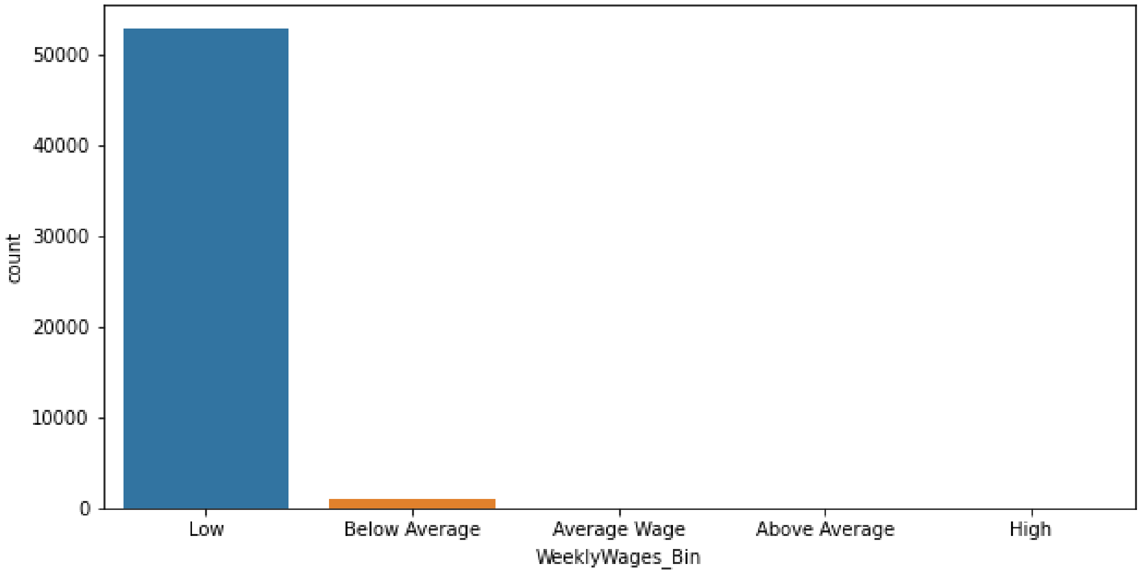
From the above plot we can say closely 91% of the people hold full time jobs.

In [ ]:

```
plt.figure(figsize=(10,5))
sns.countplot(x = 'WeeklyWages_Bin', data = Train_Dataset)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f75a91ec690>
```



In [ ]:

```
Train_Dataset['WeeklyWages_Bin'].value_counts(normalize=True)*100
```

Out[ ]:

```
Low              97.905517
Below Average     1.972259
Average Wage      0.107409
Above Average     0.007408
High              0.007408
Name: WeeklyWages_Bin, dtype: float64
```

From the above plot we can state that most of the people who claimed for insurance have low wages

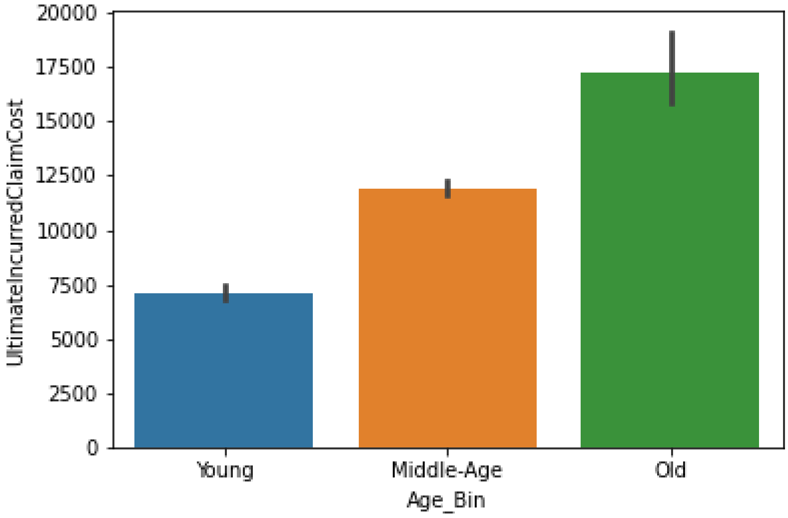
2. Bivariate analysis

In [ ]:

```
sns.barplot(x='Age_Bin',y='UltimateIncurredClaimCost',data=Train_Dataset)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f75a8d315d0>
```



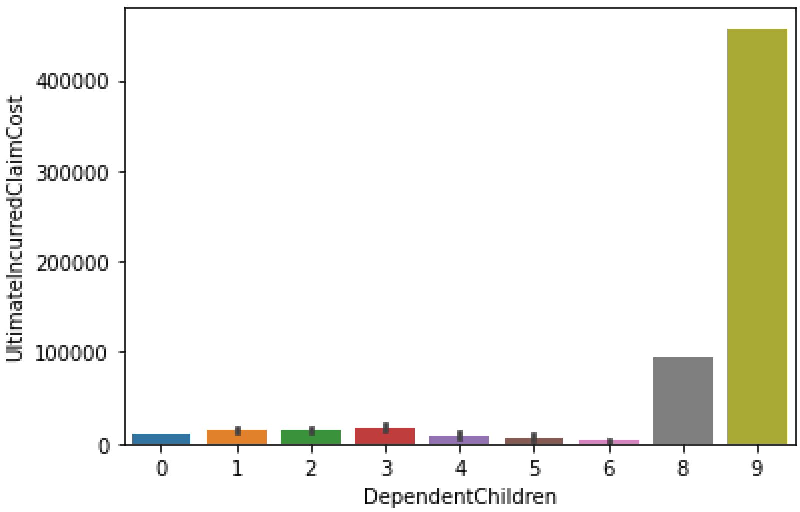
The old age group people (50-80) got more total claims payments by the insurance company.

In [ ]:

```
#plt.figure(figsize=(10,5))
sns.barplot(x='DependentChildren',y='UltimateIncurredClaimCost',data=Train_Dataset)
#plt.show()
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f75a8cb6310>
```



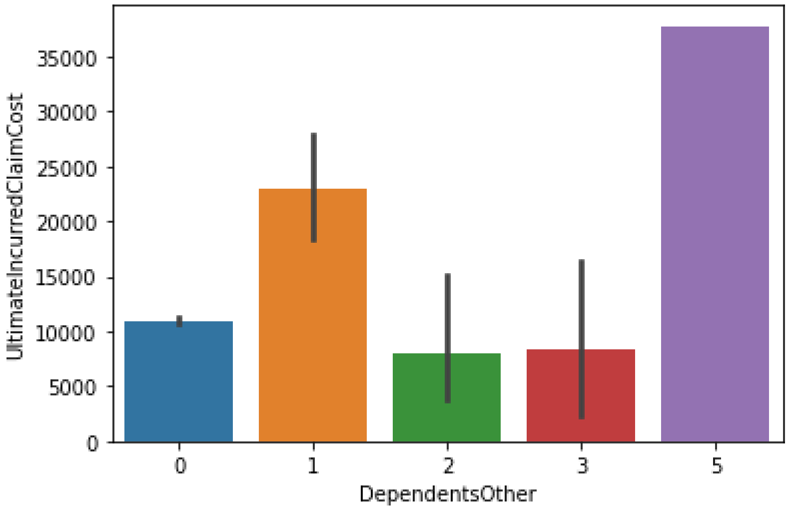
People who have more number of children as dependents got more insurance payments from the insurance company.

In [ ]:

```
sns.barplot(x='DependentsOther',y='UltimateIncurredClaimCost',data=Train_Dataset)
```

Out[ ]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f75a8be8d50>

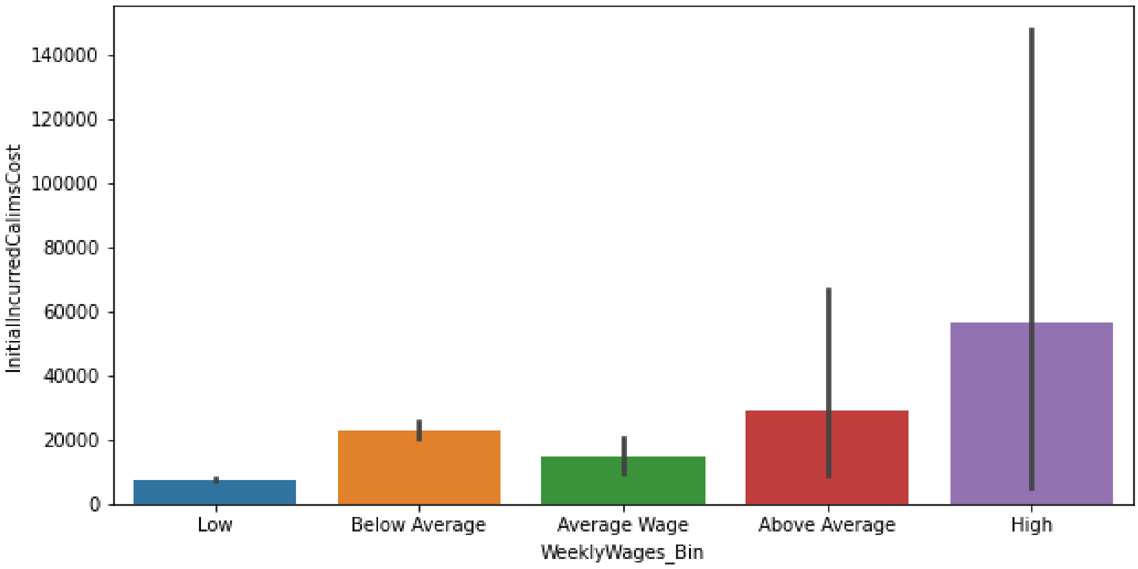


In [ ]:

```
plt.figure(figsize=(10,5))
sns.barplot(x='WeeklyWages_Bin',y='InitialIncurredCalimsCost',data=Train_Dataset)
```

Out[ ]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f75a8bbf410>



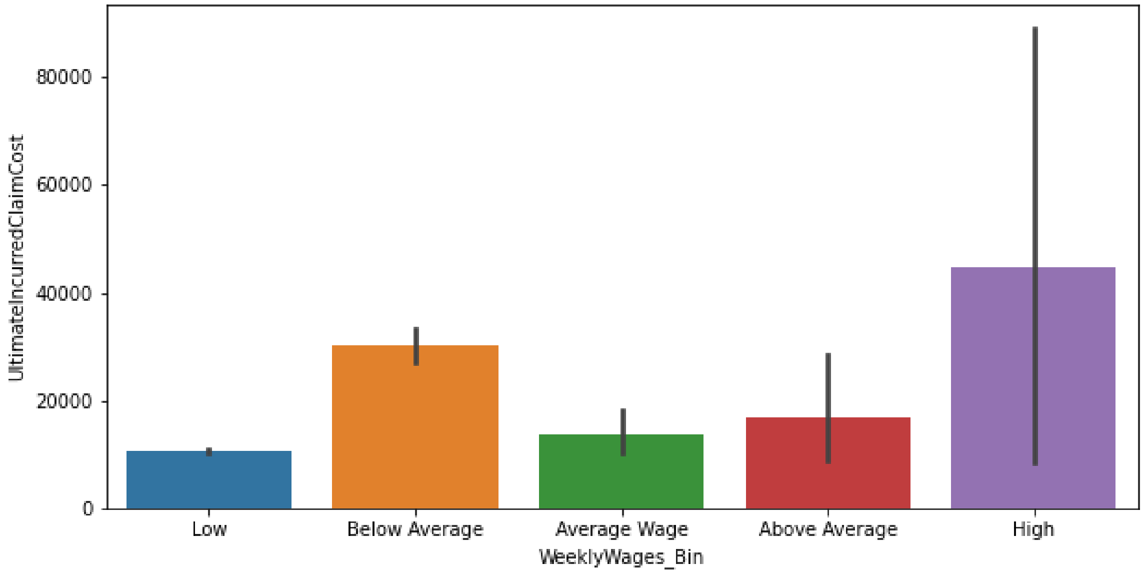
People whose wages are above average claimed for more claim cost.

In [ ]:

```
plt.figure(figsize=(10,5))
sns.barplot(x='WeeklyWages_Bin',y='UltimateIncurredClaimCost',data=Train_Dataset)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f75a8af23d0>
```



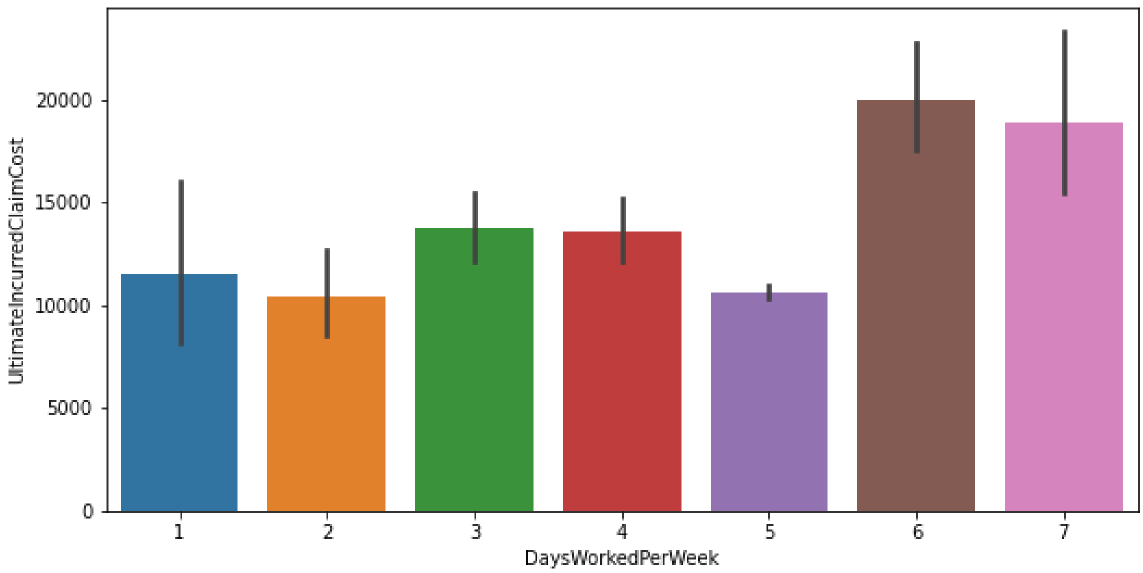
People who's wages are below average and above average got more total claims payments by the insurance company.

In [ ]:

```
plt.figure(figsize=(10,5))
sns.barplot(x='DaysWorkedPerWeek',y='UltimateIncurredClaimCost',data=Train_Dataset)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f75a8ac8790>
```



In [ ]:

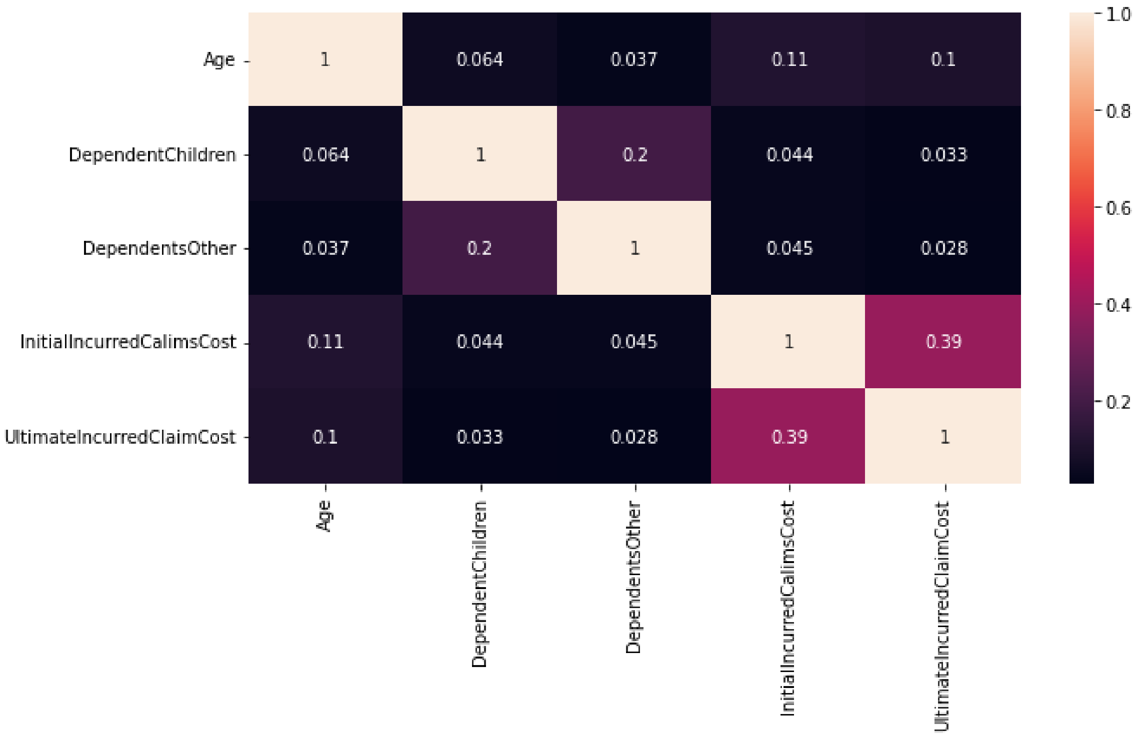
```
Train_Dataset_drop=Train_Dataset.drop(columns=['WeeklyWages', 'WeeklyWages_Bin','HoursW
orkedPerWeek','DaysWorkedPerWeek','Age_Bin'])
Train_Dataset_drop.head()
```

Out[ ]:

	ClaimNumber	DateTimeOfAccident	DateReported	Age	Gender	MaritalStatus	Dependent(
1	WC6982224	1999-01-07T11:00:00Z	1999-01-20T00:00:00Z	43	F	M	
2	WC5481426	1996-03-25T00:00:00Z	1996-04-14T00:00:00Z	30	M	U	
3	WC9775968	2005-06-22T13:00:00Z	2005-07-22T00:00:00Z	41	M	S	
4	WC2634037	1990-08-29T08:00:00Z	1990-09-27T00:00:00Z	36	M	M	
5	WC6828422	1999-06-21T11:00:00Z	1999-09-09T00:00:00Z	50	M	M	

In [ ]:

```
plt.figure(figsize=(10,5))
sns.heatmap(Train_Dataset_drop[['ClaimNumber', 'DateTimeOfAccident', 'DateReported', 'A
ge', 'DependentChildren',
       'DependentsOther','InitialIncurredCalimsCost', 'UltimateIncurredClaimCost', ]].c
orr(),annot=True)
plt.show()
```



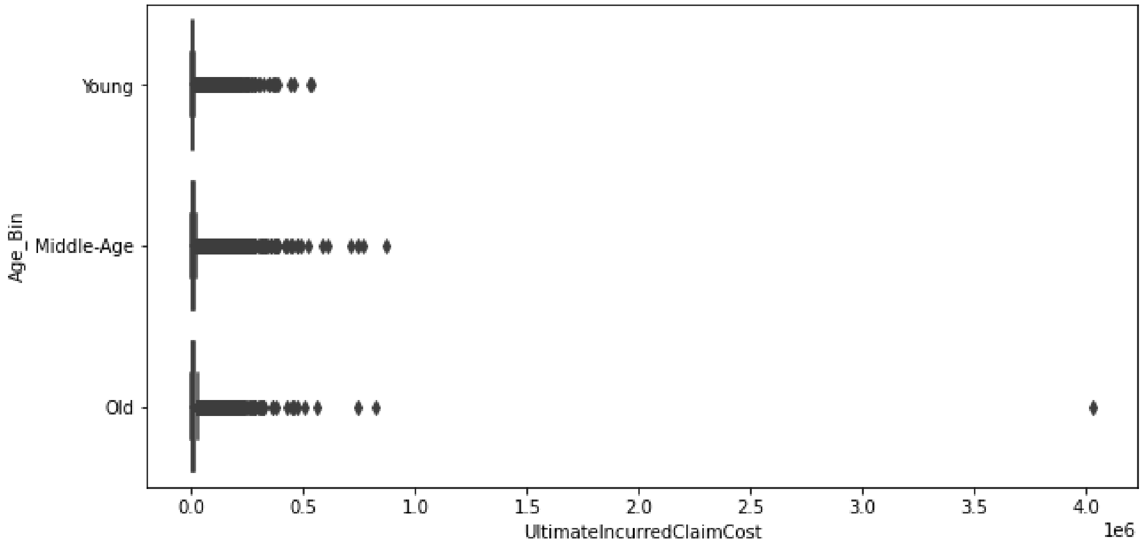
There is no correlation between the numerical columns.

In [ ]:

```
plt.figure(figsize=(10,5))
sns.boxplot(x='UltimateIncurredClaimCost', y='Age_Bin',data=Train_Dataset)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f75a8976a90>
```



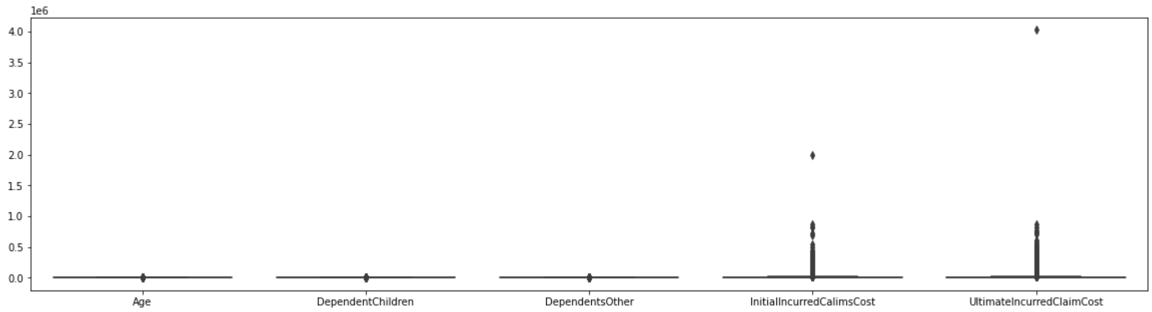
3. Outlier Analysis

In [ ]:

```
plt.figure(figsize=(20,5))
sns.boxplot(data=Train_Dataset_drop)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f75a89d9c90>
```



In [ ]:

```
Train_Dataset_drop.shape
```

Out[ ]:

```
(53999, 12)
```

From the above boxplot we can see that there are a lot of outliers in "InitialIncurredCalimsCost" and "UltimateIncurredClaimCost".

In [ ]:

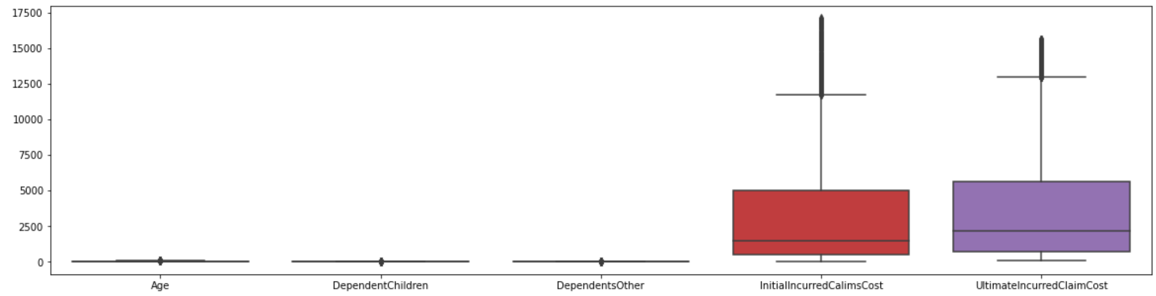
```
for i in range(4):

    limit1=3*Train_Dataset_drop['InitialIncurredCalimsCost'].std()

    lower_limit1=Train_Dataset_drop['InitialIncurredCalimsCost'].mean()-limit1
    upper_limit1=Train_Dataset_drop['InitialIncurredCalimsCost'].mean()+limit1

    Train_Dataset_drop=Train_Dataset_drop[(Train_Dataset_drop['InitialIncurredCalimsCos
t']>lower_limit1)&(Train_Dataset_drop['InitialIncurredCalimsCost']<upper_limit1)]

    limit2=3*Train_Dataset_drop['UltimateIncurredClaimCost'].std()

    lower_limit2=Train_Dataset_drop['UltimateIncurredClaimCost'].mean()-limit2
    upper_limit2=Train_Dataset_drop['UltimateIncurredClaimCost'].mean()+limit2

    Train_Dataset_drop=Train_Dataset_drop[(Train_Dataset_drop['UltimateIncurredClaimCos
t']>lower_limit2)&(Train_Dataset_drop['UltimateIncurredClaimCost']<upper_limit2)]
```

In [ ]:

```
plt.figure(figsize=(20,5))
sns.boxplot(data=Train_Dataset_drop)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f75a88881d0>
```



Machine Learning Models

In [ ]:

```
#Importing the libraries for the modeling

from sklearn.linear_model import LinearRegression
import sklearn.preprocessing as pre
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

le=pre.LabelEncoder()
```

In [ ]:

```
for x in Train_Dataset.select_dtypes(include='object').columns:
    Train_Dataset[x]=le.fit_transform(Train_Dataset[x])
```

In [ ]:

```
X_scale=Train_Dataset_drop.drop(['UltimateIncurredClaimCost'],axis='columns')
cat_Train_Dataset=X_scale.select_dtypes(exclude=[float,int]).columns
for i in cat_Train_Dataset :
    X_scale[str(i)]=le.fit_transform(X_scale[str(i)])
X_scale=X_scale.apply(pre.minmax_scale)
Y=Train_Dataset_drop['UltimateIncurredClaimCost']
```

Splitting the data into test and train data

In [ ]:

```
#Splitting the data into test and train data
x_train,x_test,y_train,y_test=train_test_split(X_scale,Y,test_size=0.3,random_state=42)
```

checking the shape of the test and train set

In [ ]:

```
#checking the shape of the test and train set
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[ ]:

```
((31850, 11), (13650, 11), (31850,), (13650,))
```

1. Linear Regression Model

In [ ]:

```
import sklearn.linear_model as lm

#creating the linear regression model
firstmodel=lm.LinearRegression()

#Fitting the model
firstmodel.fit(x_train,y_train)
```

Out[ ]:

```
LinearRegression()
```

In [ ]:

```
#Checking the train score
firstmodel.score(x_train,y_train)
print("Training set accuracy: ", +(firstmodel.score(x_train, y_train)))

#Checking the test score
firstmodel.score(x_test,y_test)
print("Test set accuracy:" , +(firstmodel.score(x_test, y_test)))
```

```
Training set accuracy:  0.6798255481178431
Test set accuracy: 0.6854008835218999
```

In [ ]:

```
#Predictions on the test data set.
y_pred = firstmodel.predict(x_test)

print('The predict values are:\n',y_pred)
```

```
The predict values are:
 [ 3191.13789665 10054.30742874  1981.2219766  ...  4914.80242607
  1362.63235906  1414.83045855]
```

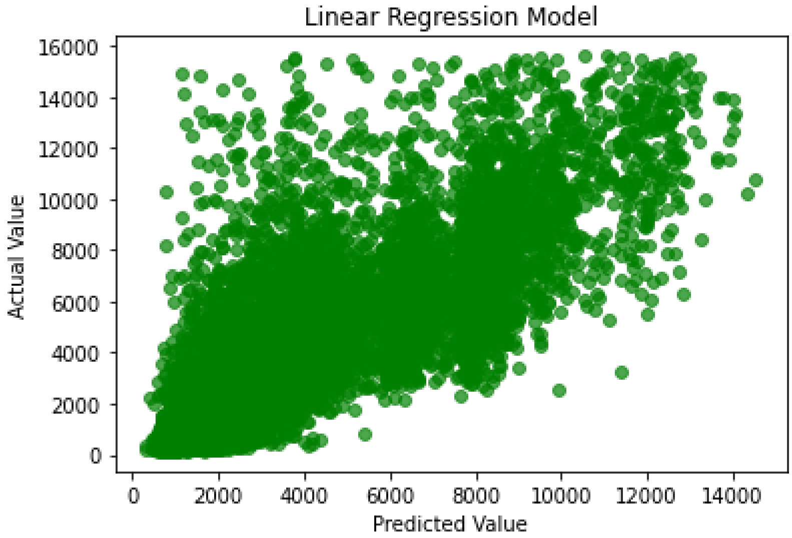
In [ ]:

```
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

print('Linear Regression Model')

# Checking the R squared(R2) value
r2 = r2_score(y_test,y_pred)
print("R^2 is: " , r2)

# Checking the Mean Absolute Error (MAE) value
print("MAE",mean_absolute_error(y_test,y_pred))

# Checking the Mean Squared Error (MSE) value
print("MSE",mean_squared_error(y_test,y_pred))

# Checking the Root Mean Squared Error (RMSE) value
print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred)))

#Checking Root Mean Squared Log Error(RMSLE)
print("RMSLE",np.log(np.sqrt(mean_squared_error(y_test,y_pred))))
```

```
Linear Regression Model
R^2 is:  0.6854008835218999
MAE 1330.9105731217392
MSE 3625104.373780617
RMSE 1903.9706861663121
RMSLE 7.551696819311737
```

In [ ]:

```
#Plotting the scatter plot between the predicted value and the actual values
actual_values = y_test
plt.scatter(y_pred, actual_values, alpha=.7,
            color='g')
plt.xlabel('Predicted Value')
plt.ylabel('Actual Value')
plt.title('Linear Regression Model')
plt.show()
```



2. Decision Tree Regression Model

In [ ]:

```
#Importing the decision tree regressor
from sklearn.tree import DecisionTreeRegressor

# creating the model
secondmodel= DecisionTreeRegressor()

#Fitting the data
secondmodel.fit(x_train, y_train)
```

Out[ ]:

```
DecisionTreeRegressor()
```

In [ ]:

```
#Checking the train score
secondmodel.score(x_train,y_train)
print("Training set accuracy: ", +(secondmodel.score(x_train, y_train)))

#Checking the test score
secondmodel.score(x_test,y_test)
print("Test set accuracy:" , +(secondmodel.score(x_test, y_test)))
```

```
Training set accuracy:  1.0
Test set accuracy: 0.5186011162312603
```

In [ ]:

```
# predicting the test set results
y_pred1= secondmodel.predict(x_test)
print('The predict values are:\n',y_pred1)
```

```
The predict values are:
 [5895.406111  8579.471902  2115.35383   ... 4848.11682    452.1326662
  199.3743564]
```

In [ ]:

```
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

print('Decision Tree Regression Model')


# Checking the R squared(R2) value
r2 = r2_score(y_test,y_pred1)
print("R^2 is: " , r2)

# Checking the Mean Absolute Error (MAE) value
print("MAE",mean_absolute_error(y_test,y_pred1))

# Checking the Mean Squared Error (MSE) value
print("MSE",mean_squared_error(y_test,y_pred1))

# Checking the oot Mean Squared Error (RMSE) value
print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred1)))

#Checking Root Mean Squared Log Error(RMSLE)
print("RMSLE",np.log(np.sqrt(mean_squared_error(y_test,y_pred1))))
```

```
Decision Tree Regression Model
R^2 is:  0.5186011162312603
MAE 1483.7112169384102
MSE 5547126.828007627
RMSE 2355.2339221418383
RMSLE 7.764395331514155
```

3. Random Forest Regression Model

In [ ]:

```
from sklearn.ensemble import RandomForestRegressor

# creating the model
thirdmodel = RandomForestRegressor(n_estimators = 10, random_state = 0)

#Fitting the data
thirdmodel.fit(x_train, y_train)
```

Out[ ]:

```
RandomForestRegressor(n_estimators=10, random_state=0)
```

In [ ]:

```
#Checking the train score
thirdmodel.score(x_train,y_train)
print("Training set accuracy: ", +(thirdmodel.score(x_train, y_train)))

#Checking the test score
thirdmodel.score(x_test,y_test)
print("Test set accuracy:" , +(thirdmodel.score(x_test, y_test)))
```

```
Training set accuracy:  0.9509445717296763
Test set accuracy: 0.726165582100248
```

In [ ]:

```
# predicting the test set results
y_pred2= thirdmodel.predict(x_test)
print('The predict values are:\n',y_pred2)
```

```
The predict values are:
 [3258.3329582  9237.0789959  1529.65064538 ... 7384.3224032   665.6282603
5
  381.56254478]
```

In [ ]:

```
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score


print('Random Forest Regression Model')


# Checking the R squared(R2) value
r2 = r2_score(y_test,y_pred2)
print("R^2 is: " , r2)

# Checking the Mean Absolute Error (MAE) value
print("MAE",mean_absolute_error(y_test,y_pred2))

# Checking the Mean Squared Error (MSE) value
print("MSE",mean_squared_error(y_test,y_pred2))

# Checking the oot Mean Squared Error (RMSE) value
print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred2)))

#Checking Root Mean Squared Log Error(RMSLE)
print("RMSLE",np.log(np.sqrt(mean_squared_error(y_test,y_pred2))))
```

```
Random Forest Regression Model
R^2 is:  0.726165582100248
MAE 1151.3787852244966
MSE 3155375.504969554
RMSE 1776.3376663713332
RMSLE 7.482309032941516
```

4. Support Vector Regression (SVM)

In [ ]:

```
from sklearn.svm import SVR

# creating the model
fourthmodel = SVR()

# fitting the training data to the model
fourthmodel.fit(x_train, y_train)
```

Out[ ]:

```
SVR()
```

In [ ]:

```
#Checking the train score
fourthmodel.score(x_train,y_train)
print("Training set accuracy: ", +(fourthmodel.score(x_train, y_train)))

#Checking the test score
fourthmodel.score(x_test,y_test)
print("Test set accuracy:" , +(fourthmodel.score(x_test, y_test)))
```

```
Training set accuracy:  0.1949175403714759
Test set accuracy: 0.19737410156975255
```

In [ ]:

```
# predicting the test set results
y_pred3 = fourthmodel.predict(x_test)
print('The predict values are:\n',y_pred3)
```

```
The predict values are:
 [2452.52791317 4075.1819088  2213.84214469 ... 2859.12818538 1812.7421726
6
 1989.50855096]
```

In [ ]:

```
import numpy as np
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

print('Support Vector Machine Regression Model')


# Checking the R squared(R2) value
r2 = r2_score(y_test,y_pred3)
print("R^2 is: " , r2)

# Checking the Mean Absolute Error (MAE) value
print("MAE",mean_absolute_error(y_test,y_pred3))

# Checking the Mean Squared Error (MSE) value
print("MSE",mean_squared_error(y_test,y_pred3))

# Checking the oot Mean Squared Error (RMSE) value
print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred3)))

#Checking Root Mean Squared Log Error(RMSLE)
print("RMSLE",np.log(np.sqrt(mean_squared_error(y_test,y_pred3))))
```

```
Support Vector Machine Regression Model
R^2 is:  0.19737410156975255
MAE 2148.6818928548455
MSE 9248604.024962768
RMSE 3041.1517596073313
RMSLE 8.019991590939684
```

Comparision of All 4 Models Training, Test Score, R2 and RMSE

Evaluation Metrics	Linear	Decision Tree	Random Forest	Support Vector Machine
R^2	0.68	0.51	0.72	0.19
MAE	1330	1485	1151	2148
MSE	3625104	5551583	3155375	9248580
RMSE	1903	2356	1776	3041
RMSLE	7.5	7.7	7.4	8.01

Working on Test Data

In [ ]:

```
Test_Dataset=pd.read_csv("/content/Test.csv")
```

In [ ]:

```
#Checking information on test data
Test_Dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36000 entries, 0 to 35999
Data columns (total 14 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   ClaimNumber                36000 non-null  object 
 1   DateTimeOfAccident         36000 non-null  object 
 2   DateReported               36000 non-null  object 
 3   Age                        36000 non-null  int64  
 4   Gender                     36000 non-null  object 
 5   MaritalStatus              35982 non-null  object 
 6   DependentChildren          36000 non-null  int64  
 7   DependentsOther            36000 non-null  int64  
 8   WeeklyWages                36000 non-null  float64
 9   PartTimeFullTime           36000 non-null  object 
 10  HoursWorkedPerWeek         36000 non-null  float64
 11  DaysWorkedPerWeek          36000 non-null  int64  
 12  ClaimDescription           36000 non-null  object 
 13  InitialIncurredCalimsCost  36000 non-null  int64  
dtypes: float64(2), int64(5), object(7)
memory usage: 3.8+ MB
```

In [ ]:

```
#checking the description of test data
Test_Dataset.describe()
```

Out[ ]:

	Age	DependentChildren	DependentsOther	WeeklyWages	HoursWorkedPerWe
count	36000.000000	36000.000000	36000.000000	36000.00000	36000.000C
mean	33.856556	0.120000	0.009611	416.37745	37.758ε
std	12.124416	0.522437	0.108357	242.49109	11.951
min	13.000000	0.000000	0.000000	1.00000	0.000C
25%	23.000000	0.000000	0.000000	200.00000	38.000C
50%	32.000000	0.000000	0.000000	395.18500	38.000C
75%	43.000000	0.000000	0.000000	500.00000	40.000C
max	80.000000	8.000000	5.000000	7400.00000	700.000C

In [ ]:

```
#checking the test dataframe
Test_Dataset.head(5)
```

Out[ ]:

	ClaimNumber	DateTimeOfAccident	DateReported	Age	Gender	MaritalStatus	Dependent(
0	WC8145235	2002-04-02T10:00:00Z	2002-05-07T00:00:00Z	26	M	S	
1	WC2005111	1988-04-06T16:00:00Z	1988-04-15T00:00:00Z	31	M	M	
2	WC6899143	1999-03-08T09:00:00Z	1999-04-04T00:00:00Z	57	M	M	
3	WC5502023	1996-07-26T09:00:00Z	1996-09-04T00:00:00Z	33	M	M	
4	WC4785156	1994-04-13T14:00:00Z	1994-07-07T00:00:00Z	32	F	M	

In [ ]:

```
#Checking the shape of test data
Test_Dataset.shape
```

Out[ ]:

```
(36000, 14)
```

In [ ]:

```
#Checking for duplicate values
Test_Dataset.duplicated().sum()
```

Out[ ]:

```
0
```

There are no duplicate values

In [ ]:

```
#Checking for the null values
Test_Dataset.isnull().sum()
```

Out[ ]:

```
ClaimNumber                  0
DateTimeOfAccident           0
DateReported                 0
Age                          0
Gender                       0
MaritalStatus               18
DependentChildren            0
DependentsOther              0
WeeklyWages                  0
PartTimeFullTime             0
HoursWorkedPerWeek           0
DaysWorkedPerWeek            0
ClaimDescription             0
InitialIncurredCalimsCost    0
dtype: int64
```

In [ ]:

```
#Missing Value Treatment using mode imputation
Test_Dataset['MaritalStatus']=Test_Dataset['MaritalStatus'].fillna(Test_Dataset['Marita
lStatus'].mode()[0])
```

In [ ]:

```
#To verify if there are more missing values in the dataset
Test_Dataset.isnull().sum()
```

Out[ ]:

```
ClaimNumber                  0
DateTimeOfAccident           0
DateReported                 0
Age                          0
Gender                       0
MaritalStatus                0
DependentChildren            0
DependentsOther              0
WeeklyWages                  0
PartTimeFullTime             0
HoursWorkedPerWeek           0
DaysWorkedPerWeek            0
ClaimDescription             0
InitialIncurredCalimsCost    0
dtype: int64
```

There are no missing values.

Data Transformation

Data Binning

In [ ]:

```
Test_Dataset['Age_Bin']=pd.cut(Test_Dataset['Age'],bins=[1,25,50,80] , labels=['Young',
'Middle-Age','Old'])
Test_Dataset['Age_Bin']
```

Out[ ]:

```
0        Middle-Age
1        Middle-Age
2               Old
3        Middle-Age
4        Middle-Age
            ...
35995           Old
35996         Young
35997    Middle-Age
35998    Middle-Age
35999           Old
Name: Age_Bin, Length: 36000, dtype: category
Categories (3, object): ['Young' < 'Middle-Age' < 'Old']
```

In [ ]:

```
Test_Dataset['WeeklyWages_Bin']=pd.cut(Test_Dataset['WeeklyWages'],bins=[0,1000,2000,4000,7000,8000] , labels=['Low','Below Average','Average Wage','Above Average','High'])
Test_Dataset['WeeklyWages_Bin']
```

Out[ ]:

```
0        Low
1        Low
2        Low
3        Low
4        Low
        ...
35995    Low
35996    Low
35997    Low
35998    Low
35999    Low
Name: WeeklyWages_Bin, Length: 36000, dtype: category
Categories (5, object): ['Low' < 'Below Average' < 'Average Wage' < 'Above Average' < 'High']
```

In [ ]:

```
Test_Dataset_drop=Test_Dataset.drop(columns=['WeeklyWages', 'WeeklyWages_Bin','HoursWorkedPerWeek','DaysWorkedPerWeek','Age_Bin'])
```

In [ ]:

```
Test_Dataset_drop.head(5)
```

Out[ ]:

	ClaimNumber	DateTimeOfAccident	DateReported	Age	Gender	MaritalStatus	Dependent(
0	WC8145235	2002-04-02T10:00:00Z	2002-05-07T00:00:00Z	26	M	S	
1	WC2005111	1988-04-06T16:00:00Z	1988-04-15T00:00:00Z	31	M	M	
2	WC6899143	1999-03-08T09:00:00Z	1999-04-04T00:00:00Z	57	M	M	
3	WC5502023	1996-07-26T09:00:00Z	1996-09-04T00:00:00Z	33	M	M	
4	WC4785156	1994-04-13T14:00:00Z	1994-07-07T00:00:00Z	32	F	M	

In [ ]:

```
Test_Dataset_drop.head(5)
```

Out[ ]:

	ClaimNumber	DateTimeOfAccident	DateReported	Age	Gender	MaritalStatus	Dependent(
0	WC8145235	2002-04-02T10:00:00Z	2002-05-07T00:00:00Z	26	M	S	
1	WC2005111	1988-04-06T16:00:00Z	1988-04-15T00:00:00Z	31	M	M	
2	WC6899143	1999-03-08T09:00:00Z	1999-04-04T00:00:00Z	57	M	M	
3	WC5502023	1996-07-26T09:00:00Z	1996-09-04T00:00:00Z	33	M	M	
4	WC4785156	1994-04-13T14:00:00Z	1994-07-07T00:00:00Z	32	F	M	

In [ ]:

```
Test_Dataset_drop.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36000 entries, 0 to 35999
Data columns (total 11 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   ClaimNumber               36000 non-null  object
 1   DateTimeOfAccident        36000 non-null  object
 2   DateReported              36000 non-null  object
 3   Age                       36000 non-null  int64 
 4   Gender                    36000 non-null  object
 5   MaritalStatus             36000 non-null  object
 6   DependentChildren         36000 non-null  int64 
 7   DependentsOther           36000 non-null  int64 
 8   PartTimeFullTime          36000 non-null  object
 9   ClaimDescription          36000 non-null  object
 10  InitialIncurredCalimsCost  36000 non-null  int64 
dtypes: int64(4), object(7)
memory usage: 3.0+ MB
```

In [ ]:

```
label_encoder=pre.LabelEncoder()
Test_Dataset_drop['ClaimNumber']=label_encoder.fit_transform(Test_Dataset_drop['ClaimNumber'])
Test_Dataset_drop['DateTimeOfAccident']=label_encoder.fit_transform(Test_Dataset_drop['DateTimeOfAccident'])
Test_Dataset_drop['DateReported']=label_encoder.fit_transform(Test_Dataset_drop['DateReported'])
Test_Dataset_drop['Age']=label_encoder.fit_transform(Test_Dataset_drop['Age'])
Test_Dataset_drop['Gender']=label_encoder.fit_transform(Test_Dataset_drop['Gender'])
Test_Dataset_drop['MaritalStatus']=label_encoder.fit_transform(Test_Dataset_drop['MaritalStatus'])
Test_Dataset_drop['PartTimeFullTime']=label_encoder.fit_transform(Test_Dataset_drop['PartTimeFullTime'])
Test_Dataset_drop['ClaimDescription']=label_encoder.fit_transform(Test_Dataset_drop['ClaimDescription'])
```

In [ ]:

```
Test_Dataset_drop.dtypes
```

Out[ ]:

```
ClaimNumber                 int64
DateTimeOfAccident          int64
DateReported                int64
Age                         int64
Gender                      int64
MaritalStatus               int64
DependentChildren           int64
DependentsOther             int64
PartTimeFullTime            int64
ClaimDescription            int64
InitialIncurredCalimsCost   int64
dtype: object
```

In [ ]:

```
def test_pre(data):
    import sklearn.preprocessing as pre
    from sklearn.preprocessing import minmax_scale
    #label_encoder=pre.LabelEncoder()
    data=data.apply(minmax_scale)
    #data['Age']=label_encoder.fit_transform(data['Age'])
    #data['MaritalStatus']=label_encoder.fit_transform(data['MaritalStatus'])
    #data['ClaimDescription']=label_encoder.fit_transform(data['ClaimDescription'])
    return data
```

In [ ]:

```
test=test_pre(Test_Dataset_drop)
```

In [ ]:

```
y_pred=thirdmodel.predict(test)
```

In [ ]:

```
y_pred
```

Out[ ]:

```
array([ 846.53527419, 1022.43051588,  872.05163966, ..., 1213.23701594,
       1273.96498442,  951.11595528])
```

In [ ]:

```
result=y_pred
```

In [ ]:

```
def submission(result):
     submission = pd.read_csv("/content/sample_submission.csv")
     submission = submission.drop('UltimateIncurredClaimCost',axis=1)
     submission['UltimateIncurredClaimCost'] = result
     #Writing the file
     submission.to_csv("Submit.csv",index=False)

submission(result)
```