In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/playground-series-s4e7/sample_submission.csv
/kaggle/input/playground-series-s4e7/train.csv
/kaggle/input/playground-series-s4e7/test.csv
```

In [2]:

```
train = pd.read_csv("/kaggle/input/playground-series-s4e7/train.csv")
print(train.shape)
```

```
(11504798, 12)
```

Exploratory Data Analysis

In [3]:
```
train_num = train[['Age','Annual_Premium','Vintage']]
train_cat = train[['Gender','Driving_License','Region_Code','Previously_Insured','Vehicle_Age','Vehicle_Damage','Policy_Sales_Channel']]
```

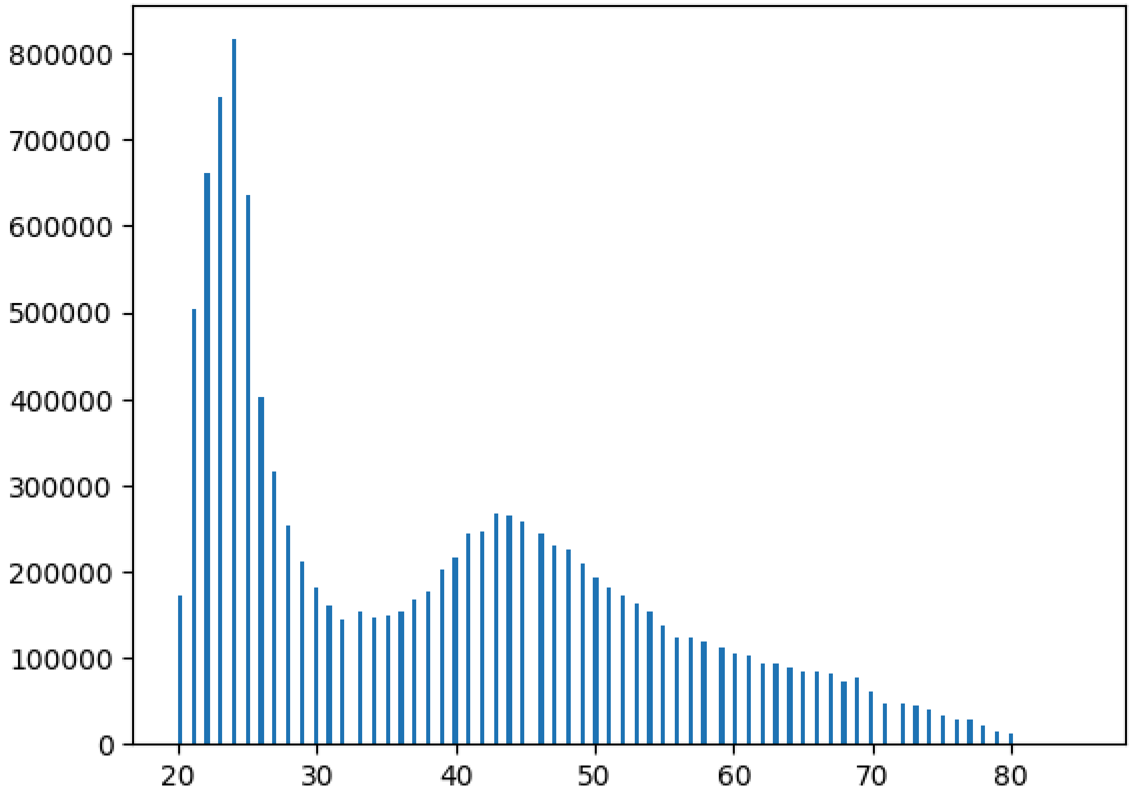
In [4]:
```
# Analyse %response by gender. Males appear to respond more positively.
gender_pivot = pd.pivot_table(train, index='Response', columns='Gender', values='id', aggfunc='count')
gender_pivot['Female%'] = gender_pivot['Female']/gender_pivot['Female'].sum()*100
gender_pivot['Male%'] = gender_pivot['Male']/gender_pivot['Male'].sum()*100
gender_pivot
```

Out[4]:

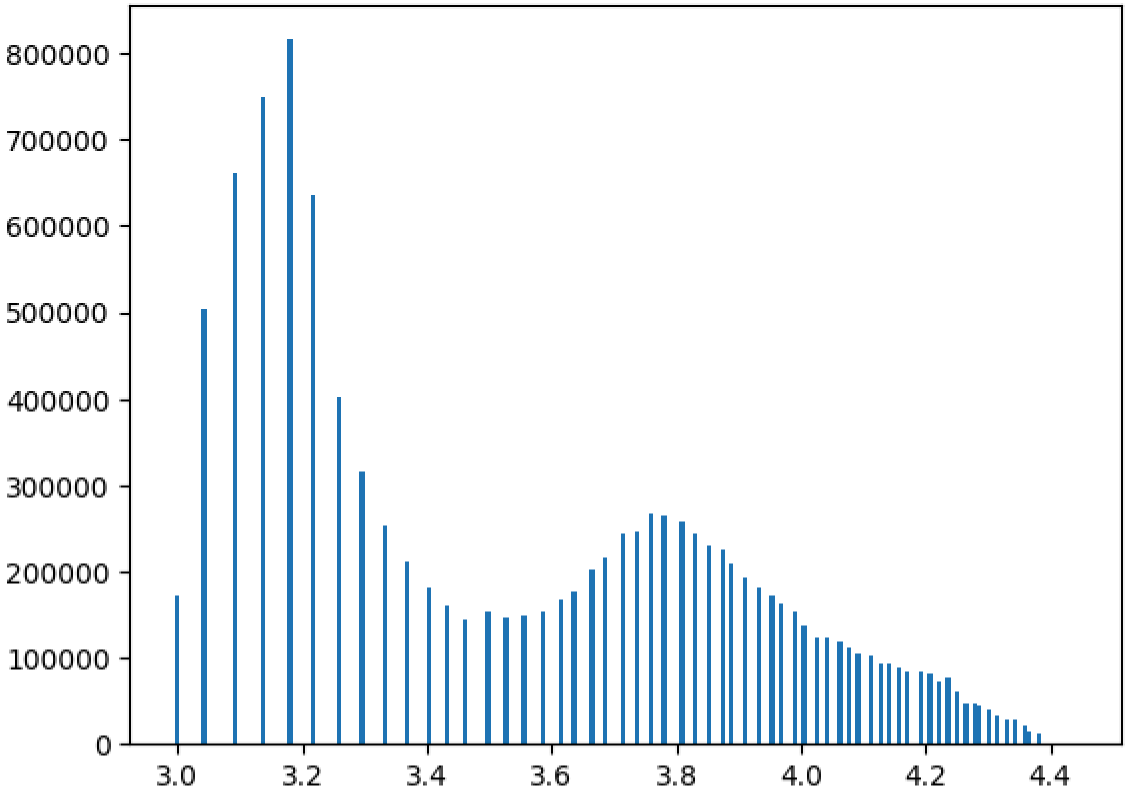
Gender	Female	Male	Female%	Male%
Response				
0	4731603	5358136	89.670349	86.031161
1	545061	869998	10.329651	13.968839

In [5]:

```
# Plot histogram by age
plt.hist(train['Age'], bins=200);
```

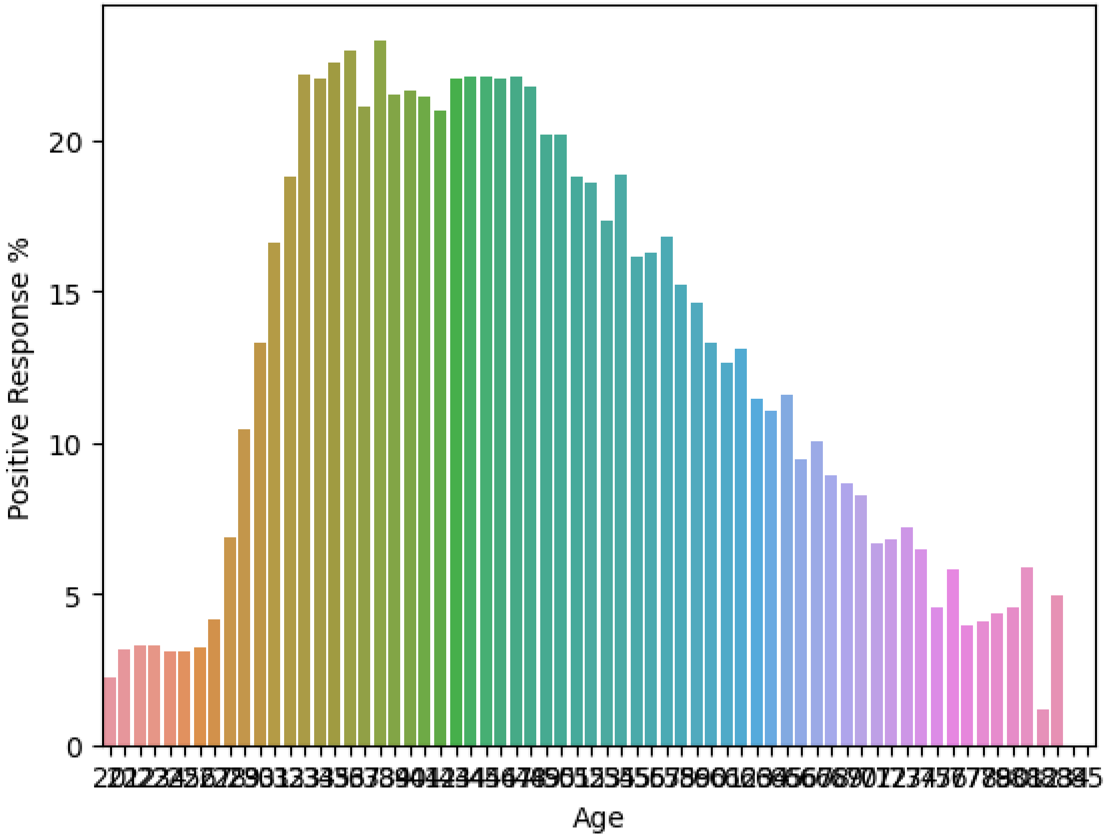


In [6]:
```
# Normalising histogram by age
plt.hist(np.log(train['Age']), bins=200);
```



In [7]:

```
# Analyse and visualise %response by age. Response appears to peak aroun
d 30s, when people are more likely to own/drive a car.
age_pivot = pd.pivot_table(train, index='Response', columns='Age', valu
es='id', aggfunc='count').T
age_pivot['Response%'] = age_pivot[1]/(age_pivot[0]+age_pivot[1])*100
age_pivot
sns.barplot(x=age_pivot.index, y=age_pivot['Response%']).set_ylabel('Po
sitive Response %')
plt.show()
```



In [8]:

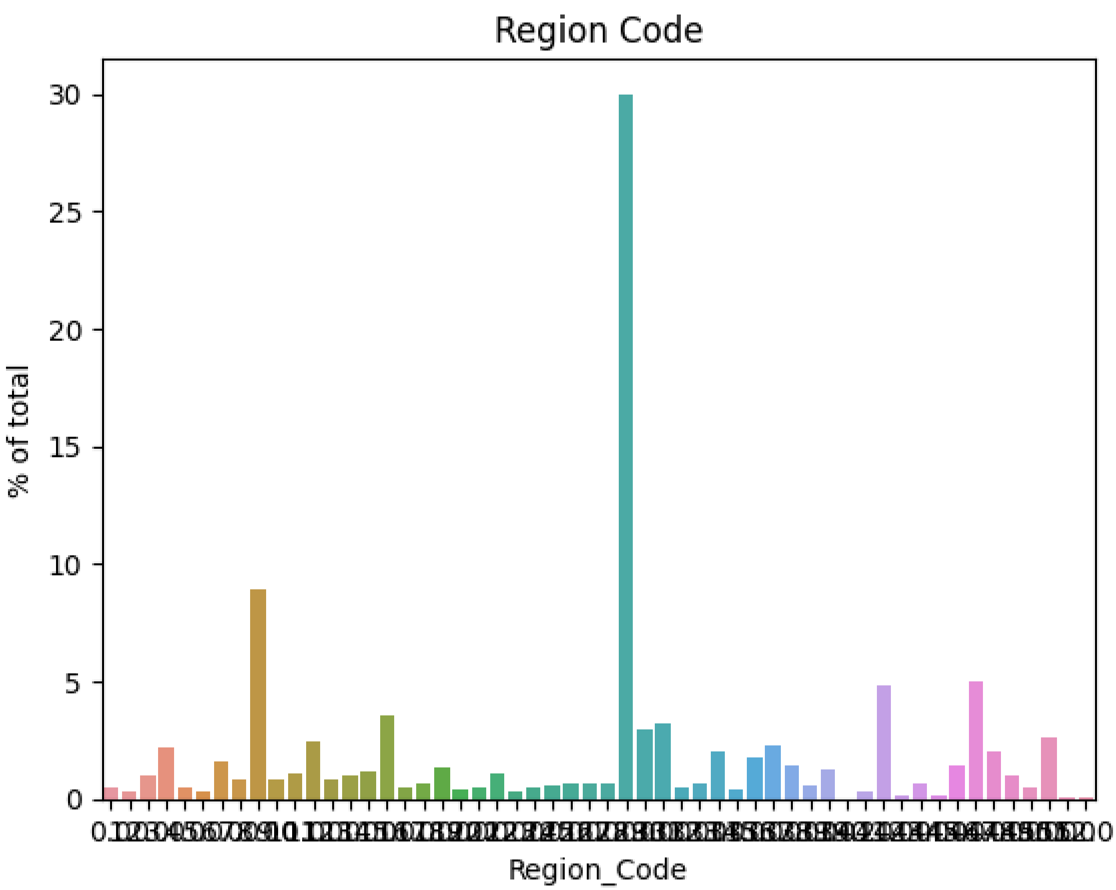
```
# Analyse %response of those with(out) driving license. Those with licen
se are more likely to require insurance.
pd.pivot_table(train, index='Response', columns='Driving_License', valu
es='id', aggfunc='count')
```

Out[8]:

Driving_License	0	1
Response		
0	21502	10068237
1	1255	1413804

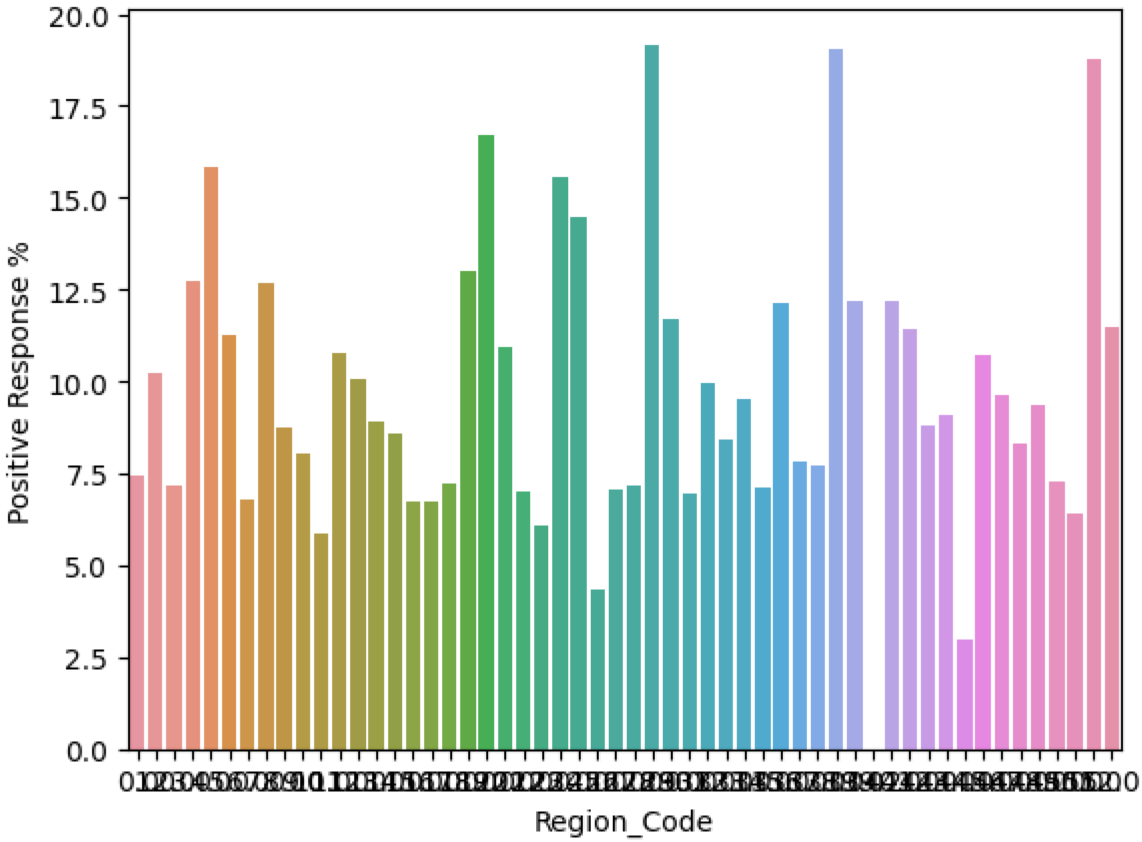
In [9]:

```
# Plotting distribution by region code
region_bar = sns.barplot(x=train['Region_Code'].value_counts().index, y
=train['Region_Code'].value_counts()/11504798*100)
region_bar.set_title('Region Code')
region_bar.set_ylabel('% of total')
plt.show()
```



In [10]:

```
# Analysing positive response % by region
region_pivot = pd.pivot_table(train, index='Response', columns='Region_Code', values='id', aggfunc='count').T
region_pivot['Response%'] = region_pivot[1]/(region_pivot[0]+region_pivot[1])*100
sns.barplot(x=region_pivot.index, y=region_pivot['Response%']).set_ylabel('Positive Response %')
plt.show()
```



In [11]:

```
# Analyse %response of those (not) previously insured. Those who never bought insurance before are more likely to require insurance.
pd.pivot_table(train, index='Response', columns='Previously_Insured', values='id', aggfunc='count')
```

Out[11]:

Previously_Insured	0	1
Response		
0	4766457	5323282
1	1411659	3400

In [12]:

```
# Analyse %response by vehicle age. Respondents with older vehicles more likely to require insurance.
vehicle_age_pivot = pd.pivot_table(train, index='Response', columns='Vehicle_Age', values='id', aggfunc='count')
vehicle_age_pivot = vehicle_age_pivot[['< 1 Year','1-2 Year','> 2 Years']]
vehicle_age_pivot
```

Out[12]:

Vehicle_Age	< 1 Year	1-2 Year	> 2 Years
Response			
0	4835296	4919406	335037
1	208849	1063272	142938

In [13]:

```
# Analyse %response by vehicle damage. Those who have damaged vehicles are more likely to require insurance.
pd.pivot_table(train, index='Response', columns='Vehicle_Damage', values='id', aggfunc='count')
```

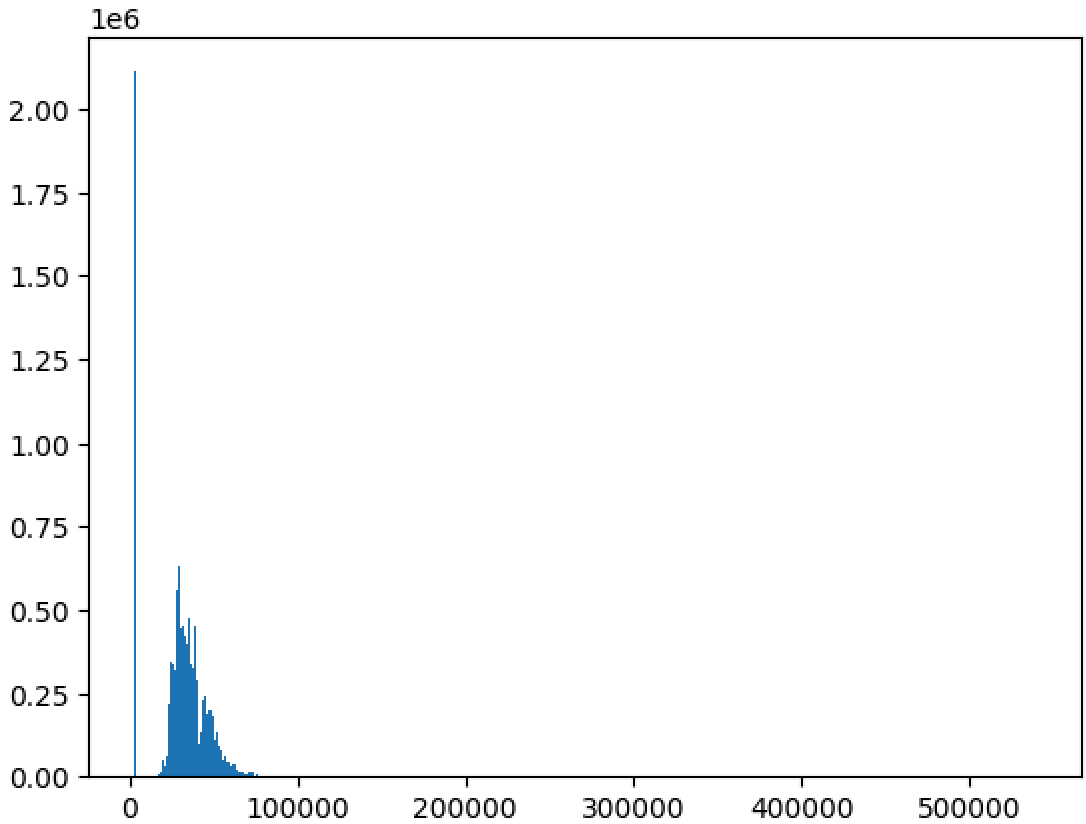
Out[13]:

Vehicle_Damage	No	Yes
Response		
0	5697548	4392191
1	24021	1391038

In [14]:
```
train['Annual_Premium'].value_counts()/11504798*100
```

Out[14]:
```
Annual_Premium
2630.0     18.362435
38287.0     0.055307
39008.0     0.045937
38452.0     0.041035
28861.0     0.040600
              ...
77839.0     0.000009
67126.0     0.000009
15999.0     0.000009
59067.0     0.000009
64538.0     0.000009
Name: count, Length: 51728, dtype: float64
```

In [15]:
```
# Analyse distribution of annual premiums.
plt.hist(train['Annual_Premium'], bins=500);
```



In [16]:

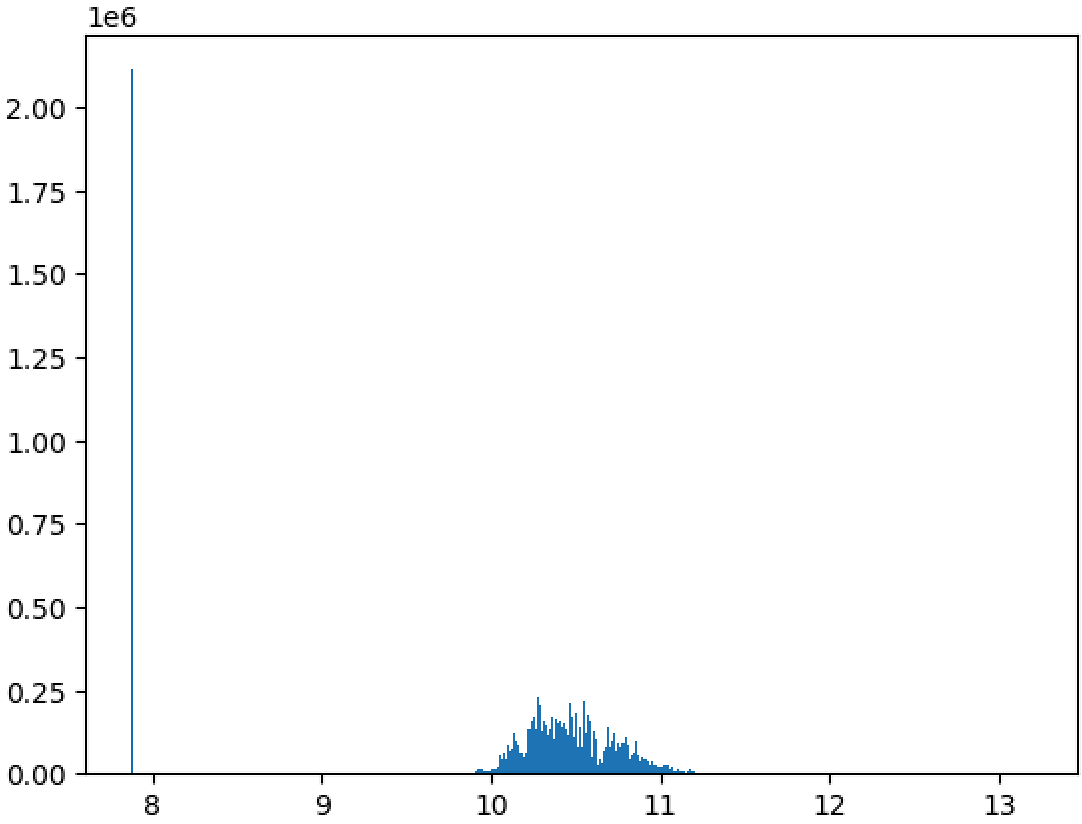
```
# Extracting outlier rows with Annual Premiums exceeding $150,000
train.loc[train['Annual_Premium'] > 150000]
```

Out[16]:

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_A
198	198	Female	39	1	45.0	0	1-2 Year
17133	17133	Female	42	1	3.0	0	1-2 Year
17968	17968	Female	50	1	28.0	0	1-2 Year
38310	38310	Female	21	1	35.0	0	< 1 Year
45301	45301	Female	23	1	14.0	1	< 1 Year
...	...	...	...	...	...	...	...
11462299	11462299	Male	67	1	28.0	0	1-2 Year
11475219	11475219	Male	60	1	28.0	0	> 2 Years
11496884	11496884	Female	22	1	37.0	1	< 1 Year
11502750	11502750	Male	40	1	7.0	0	1-2 Year
11503874	11503874	Female	24	1	29.0	1	< 1 Year

1534 rows × 12 columns

In [17]:
```
# Applying log transformation
plt.hist(np.log(train['Annual_Premium']), bins=500);
```

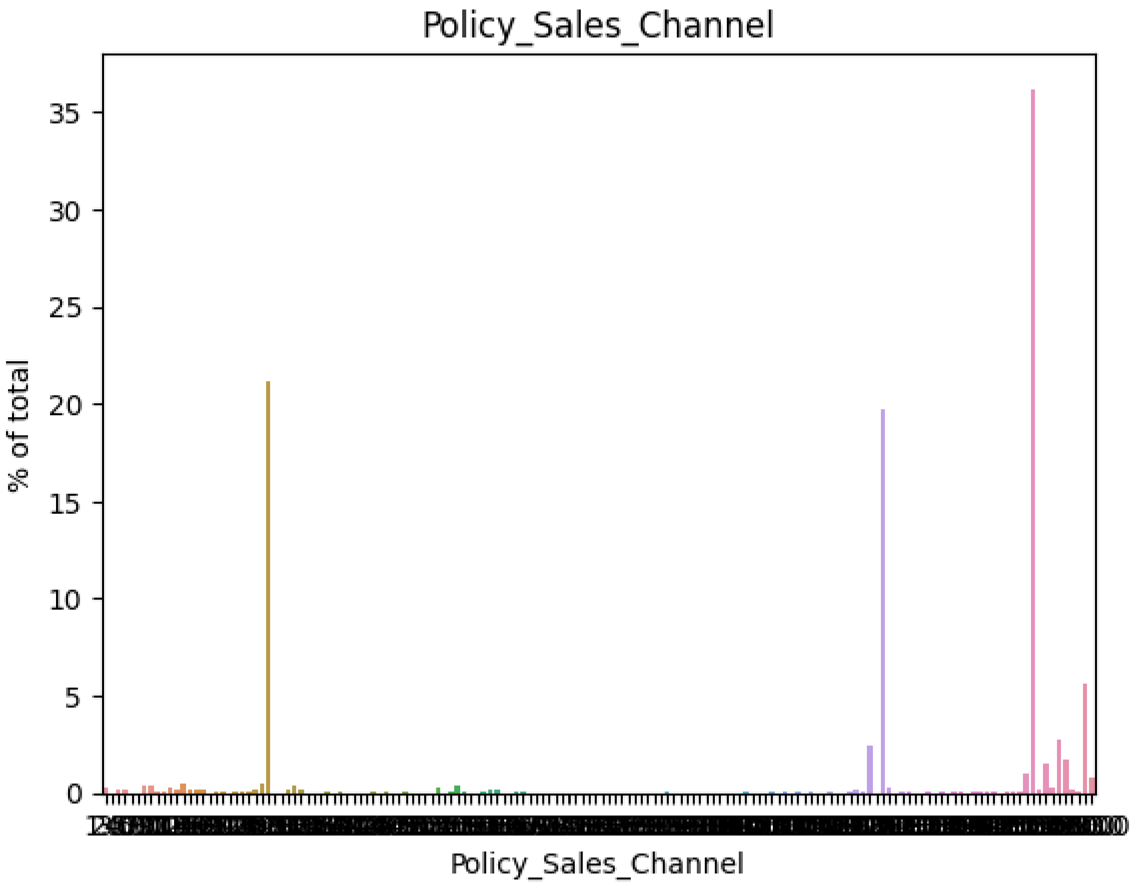


In [18]:
```
train['Policy_Sales_Channel'].value_counts()/11504798*100
```

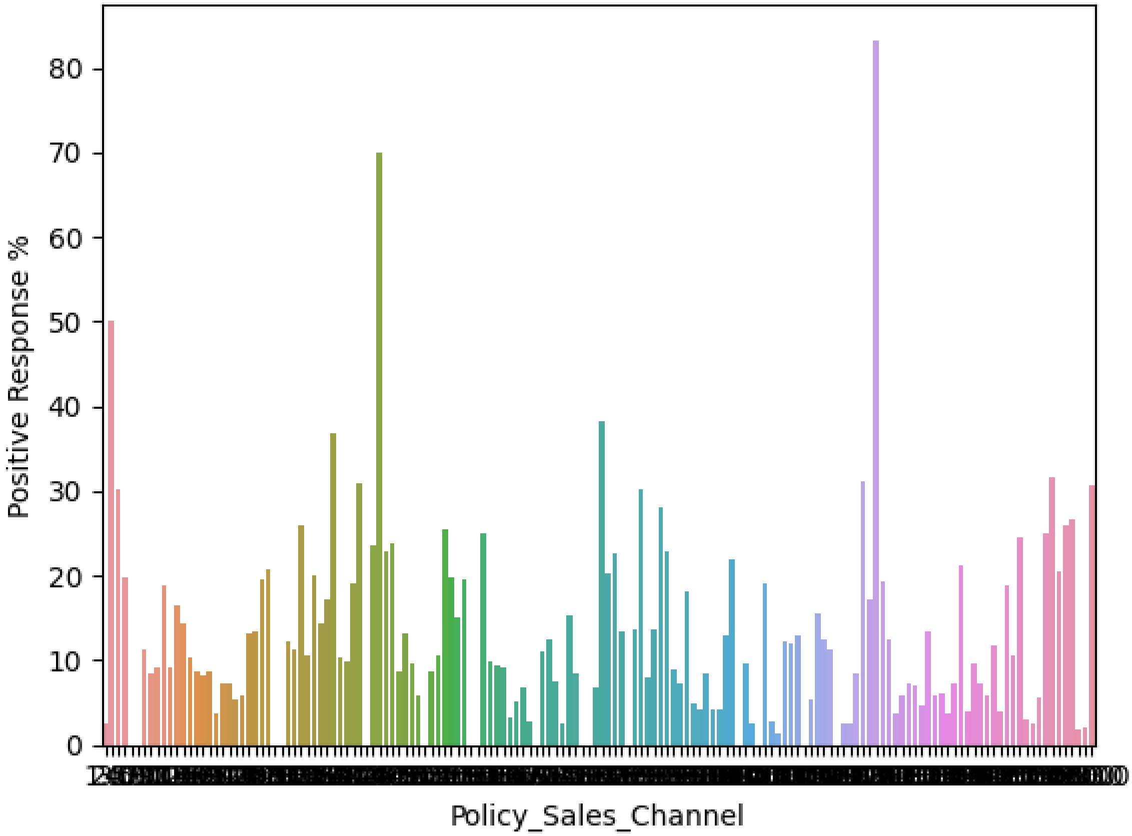
Out[18]:
```
Policy_Sales_Channel
152.0    36.212570
26.0     21.151662
124.0    19.683005
160.0     5.566199
156.0     2.752704
           ...
102.0     0.000035
112.0     0.000026
27.0      0.000017
6.0       0.000009
5.0       0.000009
Name: count, Length: 152, dtype: float64
```

In [19]:

```
# Plotting distribution by policy sales channel
channel_bar = sns.barplot(x=train['Policy_Sales_Channel'].value_counts
().index, y=train['Policy_Sales_Channel'].value_counts()/11504798*100)
channel_bar.set_title('Policy_Sales_Channel')
channel_bar.set_ylabel('% of total')
plt.show()
```



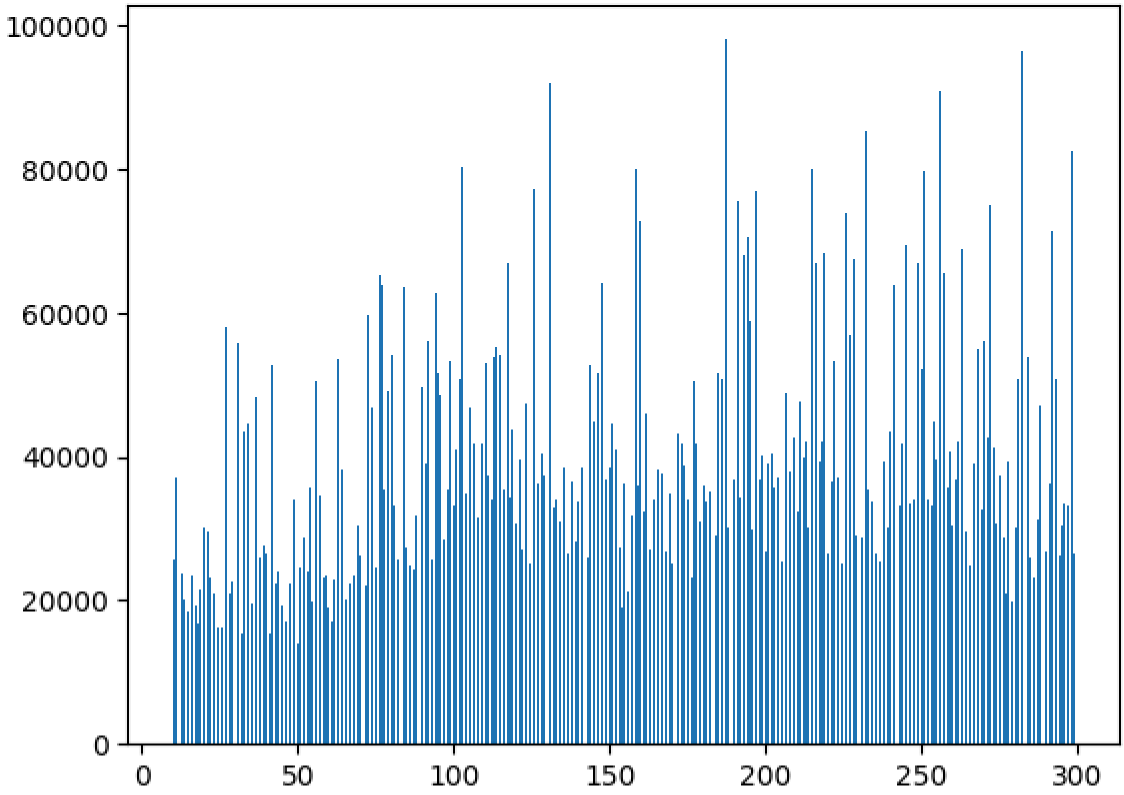
In [20]:
```
# Analysing positive response % by sales channel
channel_pivot = pd.pivot_table(train, index='Response', columns='Policy
_Sales_Channel', values='id', aggfunc='count').T
channel_pivot['Response%'] = channel_pivot[1]/(channel_pivot[0]+channel
_pivot[1])*100
sns.barplot(x=channel_pivot.index, y=channel_pivot['Response%']).set_yl
abel('Positive Response %')
plt.show()
```



In [21]:
```
channels = train['Policy_Sales_Channel'].value_counts()
channels
main_channels = channels.head(16)
main_channels
main_channels.index
```

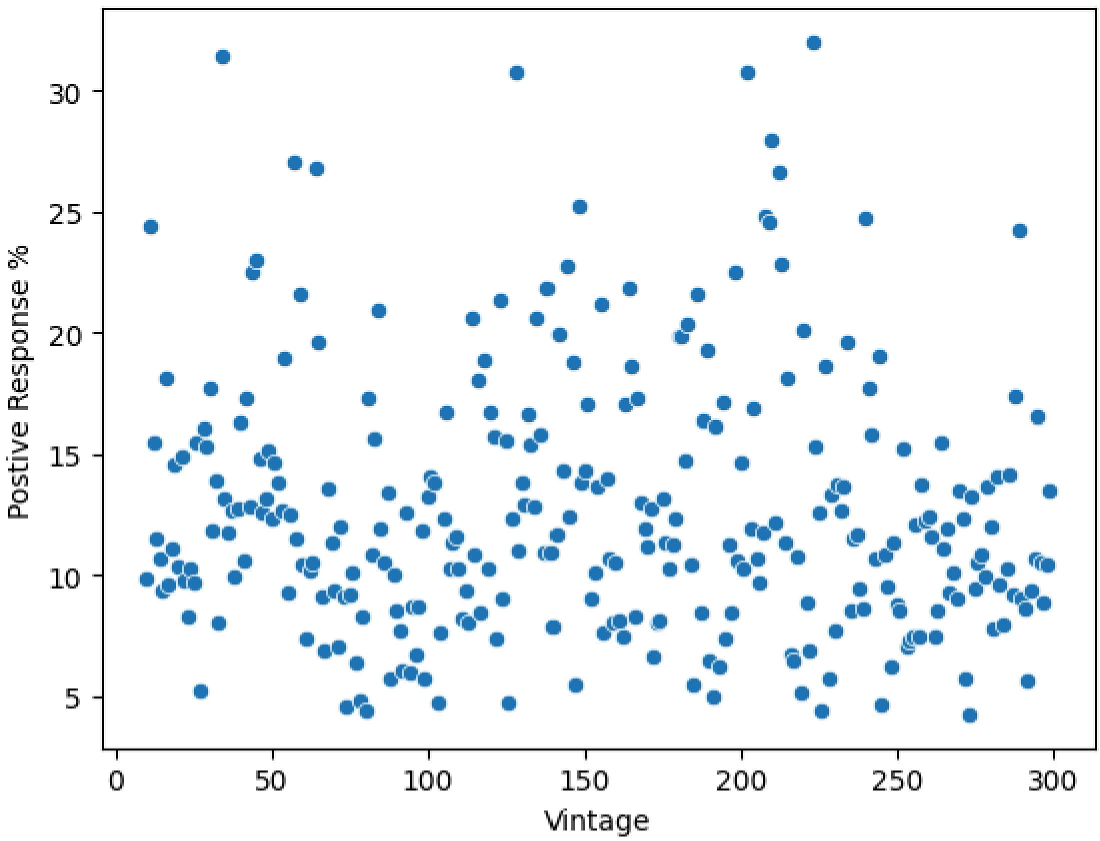
Out[21]:
```
Index([152.0,  26.0, 124.0, 160.0, 156.0, 122.0, 157.0, 154.0, 151.
0, 163.0,
        25.0,  13.0,   7.0,   8.0,  30.0,  55.0],
      dtype='float64', name='Policy_Sales_Channel')
```

In [22]:
```
# Plotting histogram distribution by customer vintage
plt.hist(train['Vintage'], bins=500);
```



In [23]:

```
# Analysing for any correlation between Vintage and positive response ra
te
vintage_pivot = pd.pivot_table(train, index='Response', columns='Vintag
e', values='id', aggfunc='count').T
vintage_pivot['Response%'] = vintage_pivot[1]/(vintage_pivot[0]+vintage
_pivot[1])*100
sns.scatterplot(x=vintage_pivot.index, y=vintage_pivot['Response%']).se
t_ylabel('Postive Response %')
plt.show()
```



Insights gained and observations made

1. Gender

- Males have a slightly higher rate of positive response (Males: 13.97% vs Females: 10.33%).

2. Age

- Age distribution displays right-skewing of the data.
- Positive response rate is highest between age 30 and 72 (middle-aged customers). This suggests that this age group should be the target focus of the insurance company.

3. Driving License

- Most of the data is obtained from those possessing a driving license.
- Additionally, the positive response rate is much higher with customers possessing a driving license, which is to be expected.

4. Region Code

- Distribution of region codes indicate that most of the data has been obtained from a single region (Region 28 represents 30% of the data)

5. Previously Insured

- Those who never bought insurance before are significantly more likely to require insurance.

6. Vehicle Age

- Most of the data is obtained from those possessing vehicles aged 2-years or less.
- Can be observed that the older the vehicle possessed by the customer, the more likely the customer requires insurance.

7. Vehicle Damage

- Can be observed that customers with damaged vehicles are more likely to require insurance.

8. Annual Premium

- Annual Premium distribution displays right-skewing of the data, with 18.36% of customers paying 2,630 a year.
- Suggests that most customers pay low premiums, while a tiny proportion of outlier customers pay >100,000 worth of annual premiums.

9. Policy Sales Channel

- Distribution shows that most of the data has been obtained from a handful of sales channels (Channel 152, 26 and 124 make up 77% of the data).

10. Vintage

- Distribution by vintage appears relatively even, which shows that the customers have been with the insurance company for various lengths of time.
- Appears to show almost no correlation with Response.

Model

In [24]:

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OrdinalEncoder, StandardScaler

train_copy = train.copy()

# Using ColumnTransformer to apply ordinal encoding and feature scaling to the respective categorical and numeric columns
preprocessor = ColumnTransformer(transformers=[
    ('ord', OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1), ['Gender','Region_Code','Vehicle_Age','Vehicle_Damage','Policy_Sales_Channel']),
    ('num', StandardScaler(), ['Age','Annual_Premium'])
], remainder = 'passthrough')

# Setting up pipeline
pipeline = Pipeline(steps=[
    ('pre', preprocessor)
])

y = train_copy['Response']
X = train_copy.drop(columns=['Response', 'id'])
X_preprocessed = pipeline.fit_transform(X)
```

In [25]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, KFold, train_test_spl
it
from sklearn.metrics import roc_auc_score

X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y,
test_size=0.2, random_state=42)

# Using Decision Tree and XGBoost classifier models. Random Forest took
too long to train as it does not scale well with large datasets like thi
s.
models = {
    'DecisionTree': DecisionTreeClassifier(random_state=42),
    'XGBoost': XGBClassifier(random_state=42)
}

# Using GridSearchCV to perform hyperparameter tuning and reduce over-fi
tting.
param_grids = {
    'DecisionTree': {
        'max_depth': [10, 15, 20],
        'min_samples_split': [2, 5, 10]
    },
    'XGBoost': {
        'max_depth': [10, 15, 20],
        'min_child_weight': [10, 15, 20],
        'gamma': [2, 4, 6]
    }
}

# 2-fold cross-validation
cv = KFold(n_splits=2, shuffle=True, random_state=42)

# Training prediction and evaluation using ROC AUC
grids = {}
for model_name, model in models.items():
    grids[model_name] = GridSearchCV(estimator=model,
                                    param_grid=param_grids[model_name],
                                    cv=cv,
                                    scoring='roc_auc',
```

```
                                    n_jobs=-1,
                                    verbose=2)
    grids[model_name].fit(X_train, y_train)
    best_params = grids[model_name].best_params_
    best_score = grids[model_name].best_score_

    print(f'Best parameters for {model_name}: {best_params}')
    print(f'Best accuracy for {model_name}: {best_score}\n')
```

```
Fitting 2 folds for each of 9 candidates, totalling 18 fits
Best parameters for DecisionTree: {'max_depth': 15, 'min_samples_sp
lit': 10}
Best accuracy for DecisionTree: 0.8611224093467322

Fitting 2 folds for each of 27 candidates, totalling 54 fits
[CV] END ..................max_depth=10, min_samples_split=5; total
time= 1.0min
[CV] END .................max_depth=10, min_samples_split=10; total
time= 1.3min
[CV] END ..................max_depth=15, min_samples_split=5; total
time= 1.5min
[CV] END ..................max_depth=20, min_samples_split=2; total
time= 1.7min
[CV] END .................max_depth=20, min_samples_split=10; total
time= 1.4min
[CV] END .........gamma=2, max_depth=10, min_child_weight=15; total
time= 2.0min
[CV] END .........gamma=2, max_depth=15, min_child_weight=10; total
time= 1.8min
[CV] END .........gamma=2, max_depth=15, min_child_weight=15; total
time= 1.9min
```

```
/opt/conda/lib/python3.10/site-packages/joblib/externals/loky/proce
ss_executor.py:752: UserWarning: A worker stopped while some jobs w
ere given to the executor. This can be caused by a too short worker
timeout or by a memory leak.
  warnings.warn(
```

```
Best parameters for XGBoost: {'gamma': 2, 'max_depth': 10, 'min_chi
ld_weight': 20}
Best accuracy for XGBoost: 0.8782273286246656
```

Submission

In [26]:

```
test = pd.read_csv("/kaggle/input/playground-series-s4e7/test.csv")
test_id = test['id']
test = test.drop(columns=['id'])
```

In [27]:

```
# Transforming test data
test_preprocessed = pipeline.transform(test)
print(X_preprocessed.shape)
print(test_preprocessed.shape)
```

```
(11504798, 10)
(7669866, 10)
```

In [28]:

```
# Prediction and submission using better-performing model
model = XGBClassifier(max_depth=10, min_child_weight=20, gamma=2).fit(X
_train, y_train)
y_pred = model.predict_proba(test_preprocessed)[:,1]
output = pd.DataFrame({'id': test_id, 'Response': y_pred})
output.to_csv('submission.csv', index=False)
```