Sai Vasanth Vinay Kudari - 50365626

Part - A

Q1: Explain overfitting and underfitting. What evaluation metrics do we use to check if the regression model fits the data well?

Overfitting: A model which overfits has low generalization performance, which mean it performs well on training data and its performance poorly on unseen data in other words we can say it has high variance and low bias;

Underfitting: A model which underfits performs poorly on both training data and unseen data, which can mean that the model is too simple to fit the data, we can say the model has high bias and low variance

Some of the commonly used evaluation metrics:

- MAE - Mean Abdolute Error
- MSE - Mean Squared Error
- RMSE - Root Mean Squared Error
- RMSLE - Root Mean Squared Log Error
- R-Squared & Adjusted R-Squared

Q2: Differentiate Correlation and Regression. What is R-squared? Discuss the drawback of R-squared in the case of multiple linear regression. What is Adjusted R-squared?

Correlation measures the relationship between two variables whereas regression tell us how one variable will effect the other, generally in regression we represent the effect of a variable using a line equation, unlike correrelation which outputs a single metric

R-squared: It is an evaluation metric, that gives the proportion of variation in target variable explained by the linear regression model. It is obtained by dividing TSS - RSS with TSS. Where TSS is Total Sum of Squares and RSS is Residual Sum of Squares. If the regression line was very close to the actual points. This means the independent variables explain the majority of variation in the target variable. In such a case, we would have a really high R-squared value.

Drawbacks in case of regression with multiple independent variables

- R-squared value never decreases with addition of redundent independent variables, which might be a problem as they might not add any value to our model but this metric cannot point this issue

Adjusted R-squared solves the above problem with multiple linear regression by considering independent variables used for predicting the target variable.

Q3: Explain precision and recall using the confusion matrix. Discuss with examples when precision and recall may consider as more useful classification evaluation metrics over accuracy.

When we are modelling to predict a rare occurance, one naive model which can give great results by simply predicting -ve as this occurance is very frequent our model's accuracy will be pretty high but we know this isnt helpful. Examples where precision and recall are more useful

- Predicting a rare disease such as cancer
- Predicting if a person can get an insurance

Evaluation Metrics :

Actual

Predicted		Correct	Not Correct
	Selected	True +ve	False +ve
	Not Selected	False -ve	True -ve

$$\text{Accuracy} = \frac{(\text{True +ve}) + (\text{True -ve})}{(\text{True +ve}) + (\text{False +ve}) + (\text{False -ve}) + (\text{True -ve})} \quad \Big\} \text{ When equally distributed}$$

Precision = % of predicted items that are correctly predicted

$$= \frac{\text{True +ve}}{(\text{True +ve}) + (\text{False +ve})}$$

Recall = % of correct items that are predicted

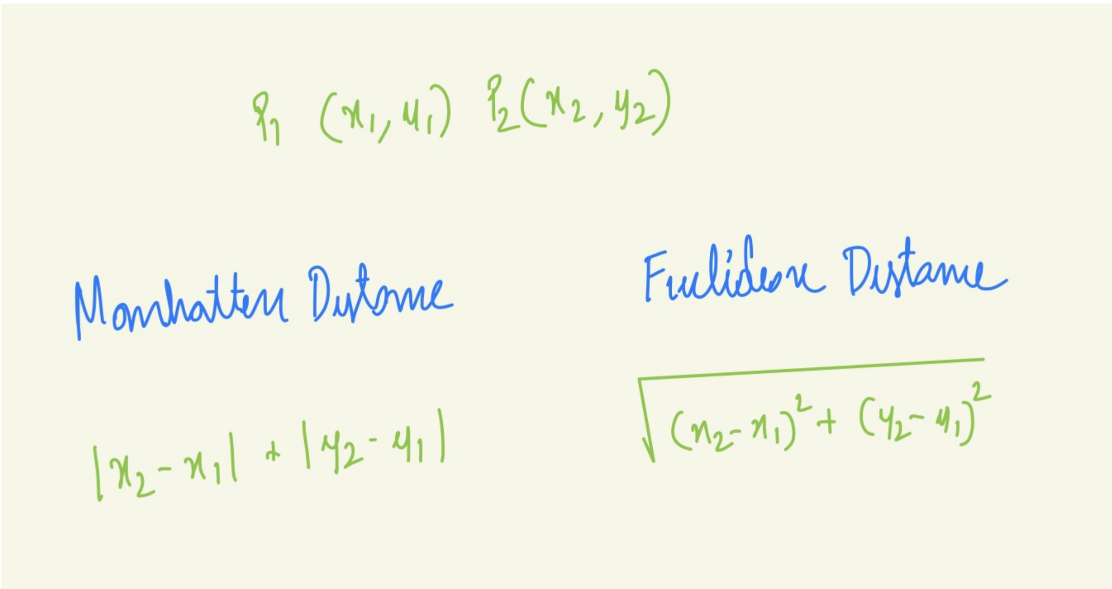
$$= \frac{(\text{True +ve})}{(\text{True +ve}) + (\text{False -ve})}$$

Q4: What is the advantage of K-Means Clustering over Hierarchical Clustering? What are Manhattan Distance and Euclidean Distance in Clustering? Why may we consider different distance measures for clustering?

Both K-Means and Hierarchial clustering are unsupervised algorithms, which aim to recognize the patterns abong data points and cluster them, there are couple of advantages of K-Means over Hierarchial clustering

- K-Means is computationally faster than Hierarchial clustering, especially when no of classes are less. Hence, K-Means is suitable for large datasets
- K-Means produces tigher clusters than Hierarchial clustering
- K-means is linear in the number of data objects i.e. $O(n)$, where n is the number of data objects. The time complexity of most of the hierarchical clustering algorithms is quadratic i.e. $O(n2)$

The choice of distance measure is very important in clustering, this metric will tell us how similar two elements and will influence the shape of clusters. Depending on the type of data we need to select the distance measure, for instance if two points are similar like most variables but differ on one, Euclidean distance will exaggerate the difference, but Manhattan distance would ignore it, being more influenced by the closeness of the other variables.



Manhatten distance is generally preferred over Euclidean when there is high dimentionality in the data

Q5: What is feature scaling? Show an example of why feature scaling is crucial? What are the two standard feature scaling techniques?

Learning algorithms depend on feature values to predict targets, often the features values are in different scales for instance in modelling country GDP the features may include per captia income, population, no of people exceeding certain income level, in this case we can understand that scales are quite different and there might be a chance that our model might give more importance to certain features. Hence, we need to scale the features before feeding into learning algorithm.

Example: In case of K-Means clustering we use some distance metric such as Euclidean/Manhatten distance to find similarity between two data elements, if we dont scale our distance metric could give wrong results when its comparing similarity between two differently scaled elements such as population vector and per captia income vector.

Feature Scaling Techniques:

- Normalization is a scaling technique that shifts and rescales numbers so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.
- Standardization is another scaling strategy, in which the values are centered around the mean and have a unit standard deviation. This means that the attribute's mean becomes zero, and the resulting distribution has a standard deviation of one.

Q6: Discuss collaborative and content-based recommendation techniques with examples

Collaborative recommendations are based on closely related entities preferences, for instance in movie recommendation engine, it will find users similar to the current user and it recommends the movies which are watched/liked by similar users. This type of recommendation works well but it also has a drawback, it has the cold start problem, it cannot work well unless we have some users already in system, which is not in case of content based recommendation systems.

Content based ones solely depend on items and its features to recommend to users like taking preference from other closely associated users. This engine will leverage user item interations to learn the user's preference and model recommendations accordingly, it suffers far less from cold start problem but it will have a problem recommending new items which are contrasting to users preferences but they may be liked by the user.

Often in practice, state of the are recommendation engines uses both content and collaborative engines in tandom

Part - B

Traffic Violation Analysis

After correlation analysis on dataset I found 'Description', 'Contributed To Accident', 'Belts', 'Personal Injury', 'Property Damage' are useful in predicting target accurately. Alongside I've engineered a new feature 'Description Pred' which is result of text classification on 'Description' column which seems to be highly correrelated as expected.

All the models do a decent job at classifying and perform equally good, but Random Forest Classifier is slighty better when compared to other two, it consistently gave better accuracy around 80% and also good sensitivity and specificity scores. Sensitivity score for 'Citation' class using Random Forest and KNN is around 0.90 and using SVM it is around 0.80. We can also observe that 'SERO' class has higher sensitivity (close to 1) and specificity amoung all other models which is due class imbalance, number of samples with SERO class are the least in the dataset and the descriptions are quite similar. Also, our the models have tough time predicting 'Warning' class this can be concluded by lower sensitivity scores.

Amoung all the three models Random Forest gives better accuracy but I found KNN more generalizable even with slightly lower accuracy.

I would deploy KNN in production for two reasons

- Inference time is faster than other models : In production we should care about faster inferences times as we need to cater for efficiency and scale
- Lower generalization gap : KNN and SVM both generalize well, which mean that traning the test error gap is lower, but since KNN has both accuracy and low generalization gap, I'm inclined toward KNN

Please find below charts and scores which I've used to derive at above conclusions

In [1]:

```
import pandas as pd
import numpy as np
from nltk import *
import matplotlib.pyplot as plt

from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, LabelEncoder
from sklearn.model_selection import train_test_split, cross_validate, learnin
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn.metrics import precision_recall_fscore_support
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix

import seaborn as sns
sns.set(rc={'figure.figsize':(10,8)})
sns.set_style('white')
colors = sns.color_palette('pastel')
```

In [2]:

```
# Took reference from sklearn
# https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_

def plot_learning_curve(
    estimator,
    title,
    X,
    y,
    axes=None,
    ylim=None,
    cv=None,
    n_jobs=None,
    train_sizes=np.linspace(0.1, 1.0, 5),
):
    """
    Generate 3 plots: the test and training learning curve, the training
    samples vs fit times curve, the fit times vs score curve.

    Parameters
    ----------
    estimator : estimator instance
        An estimator instance implementing `fit` and `predict` methods which
        will be cloned for each validation.

    title : str
        Title for the chart.

    X : array-like of shape (n_samples, n_features)
        Training vector, where ``n_samples`` is the number of samples and
        ``n_features`` is the number of features.
```

```
    y : array-like of shape (n_samples) or (n_samples, n_features)
        Target relative to ``X`` for classification or regression;
        None for unsupervised learning.

    axes : array-like of shape (3,), default=None
        Axes to use for plotting the curves.

    ylim : tuple of shape (2,), default=None
        Defines minimum and maximum y-values plotted, e.g. (ymin, ymax).

    cv : int, cross-validation generator or an iterable, default=None
        Determines the cross-validation splitting strategy.
        Possible inputs for cv are:

          - None, to use the default 5-fold cross-validation,
          - integer, to specify the number of folds.
          - :term:`CV splitter`,
          - An iterable yielding (train, test) splits as arrays of indices.

        For integer/None inputs, if ``y`` is binary or multiclass,
        :class:`StratifiedKFold` used. If the estimator is not a classifier
        or if ``y`` is neither binary nor multiclass, :class:`KFold` is used.

        Refer :ref:`User Guide <cross_validation>` for the various
        cross-validators that can be used here.

    n_jobs : int or None, default=None
        Number of jobs to run in parallel.
        ``None`` means 1 unless in a :obj:`joblib.parallel_backend` context.
        ``-1`` means using all processors. See :term:`Glossary <n_jobs>`
        for more details.

    train_sizes : array-like of shape (n_ticks,)
        Relative or absolute numbers of training examples that will be used t
        generate the learning curve. If the ``dtype`` is float, it is regarde
        as a fraction of the maximum size of the training set (that is
        determined by the selected validation method), i.e. it has to be with
        (0, 1]. Otherwise it is interpreted as absolute sizes of the training
        sets. Note that for classification the number of samples usually have
        to be big enough to contain at least one sample from each class.
        (default: np.linspace(0.1, 1.0, 5))
    """
    if axes is None:
        _, axes = plt.subplots(1, 3, figsize=(20, 5))

    axes[0].set_title(title)
    if ylim is not None:
        axes[0].set_ylim(*ylim)
    axes[0].set_xlabel("Training examples")
    axes[0].set_ylabel("Score")

    train_sizes, train_scores, test_scores, fit_times, _ = learning_curve(
        estimator,
```

```
        X,
        y,
        cv=cv,
        n_jobs=n_jobs,
        train_sizes=train_sizes,
        return_times=True,
    )
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)

    # Plot learning curve
    axes[0].grid()
    axes[0].fill_between(
        train_sizes,
        train_scores_mean - train_scores_std,
        train_scores_mean + train_scores_std,
        alpha=0.1,
        color="r",
    )
    axes[0].fill_between(
        train_sizes,
        test_scores_mean - test_scores_std,
        test_scores_mean + test_scores_std,
        alpha=0.1,
        color="g",
    )
    axes[0].plot(
        train_sizes, train_scores_mean, "o-", color="r", label="Training scor
    )
    axes[0].plot(
        train_sizes, test_scores_mean, "o-", color="g", label="Cross-validati
    )
    axes[0].legend(loc="best")

    # Plot n_samples vs fit_times
    axes[1].grid()
    axes[1].plot(train_sizes, fit_times_mean, "o-")
    axes[1].fill_between(
        train_sizes,
        fit_times_mean - fit_times_std,
        fit_times_mean + fit_times_std,
        alpha=0.1,
    )
    axes[1].set_xlabel("Training examples")
    axes[1].set_ylabel("fit_times")
    axes[1].set_title("Scalability of the model")

    # Plot fit_time vs score
    fit_time_argsort = fit_times_mean.argsort()
```

```
    fit_time_sorted = fit_times_mean[fit_time_argsort]
    test_scores_mean_sorted = test_scores_mean[fit_time_argsort]
    test_scores_std_sorted = test_scores_std[fit_time_argsort]
    axes[2].grid()
    axes[2].plot(fit_time_sorted, test_scores_mean_sorted, "o-")
    axes[2].fill_between(
        fit_time_sorted,
        test_scores_mean_sorted - test_scores_std_sorted,
        test_scores_mean_sorted + test_scores_std_sorted,
        alpha=0.1,
    )
    axes[2].set_xlabel("fit_times")
    axes[2].set_ylabel("Score")
    axes[2].set_title("Performance of the model")

    return plt
```

In [3]:

```
data = pd.read_csv('data.csv')
data.head(3)
```

Out[3]:

	Description	Belts	Personal Injury	Property Damage	Commercial License	Commercial Vehicle	State	VehicleType	Yea
0	'DISPLAYING EXPIRED REGISTRATION PLATE ISSUED ...	No	No	No	No	No	NC	'02 - Automobile'	20
1	'DRIVER FAIL TO STOP AT RED TRAFFIC SIGNAL BEF...	No	No	No	No	No	MD	'02 - Automobile'	20
2	'DRIVING UNDER THE INFLUENCE OF ALCOHOL PER SE'	No	No	No	No	No	MD	'02 - Automobile'	20C

In [4]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70340 entries, 0 to 70339
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   Description              70340 non-null  object
 1   Belts                    70340 non-null  object
 2   Personal Injury          70340 non-null  object
 3   Property Damage          70340 non-null  object
 4   Commercial License       70340 non-null  object
 5   Commercial Vehicle       70340 non-null  object
 6   State                    70340 non-null  object
 7   VehicleType              70340 non-null  object
 8   Year                     70340 non-null  object
 9   Make                     70340 non-null  object
 10  Model                    70340 non-null  object
 11  Color                    70340 non-null  object
 12  Contributed To Accident  70340 non-null  object
 13  Driver Race              70340 non-null  object
 14  Gender                   70340 non-null  object
 15  Driver City              70340 non-null  object
 16  Drive State              70340 non-null  object
 17  Violation Type           70340 non-null  object
dtypes: object(18)
memory usage: 9.7+ MB
```

Feature Engineering

In [5]:

```
desc_gp = data.groupby('Violation Type').agg({'Description':lambda x:' '.join
desc_gp.head()
```

Out[5]:

Violation Type	Description
Citation	'DISPLAYING EXPIRED REGISTRATION PLATE ISSUED ...
SERO	'STOP LIGHTS' Headlights 'STOP LIGHTS' 'Wheels...
Warning	'FAILURE OF MV OPER TO PRESENT EVIDENCE OF REQ...

In [6]:

```
def get_counts(n, text):
    text = text.replace('\'', '')
    tokens = word_tokenize(text)
    text = Text(tokens)
    counts = {}

    for i in range(1, n+1):
        counts[i] = FreqDist(ngrams(text, i))

    return counts
```

In [7]:
```
cit_c = get_counts(4, desc_gp.loc['Citation'].values[0])
sero_c = get_counts(4, desc_gp.loc['SERO'].values[0])
war_c = get_counts(4, desc_gp.loc['Warning'].values[0])
```

In [8]:
```
cit_c[2].most_common(5)
```

Out[8]:
```
[(('ON', 'HIGHWAY'), 7897),
 (('FAILURE', 'TO'), 6171),
 (('VEHICLE', 'ON'), 5997),
 (('MOTOR', 'VEHICLE'), 5454),
 (('MPH', 'IN'), 4654)]
```

In [9]:
```
cit_c[4].most_common(10)
```

Out[9]:
```
[(('MPH', 'IN', 'A', 'POSTED'), 4644),
 (('MOTOR', 'VEHICLE', 'ON', 'HIGHWAY'), 3935),
 (('PERSON', 'DRIVING', 'MOTOR', 'VEHICLE'), 3007),
 (('DRIVING', 'MOTOR', 'VEHICLE', 'ON'), 2880),
 (('VEHICLE', 'ON', 'HIGHWAY', 'WITHOUT'), 2096),
 (('PUBLIC', 'USE', 'PROPERTY', 'ON'), 1994),
 (('VEHICLE', 'ON', 'HIGHWAY', 'OR'), 1975),
 (('ON', 'HIGHWAY', 'OR', 'PUBLIC'), 1975),
 (('HIGHWAY', 'OR', 'PUBLIC', 'USE'), 1975),
 (('OR', 'PUBLIC', 'USE', 'PROPERTY'), 1975)]
```

In [10]:
```
sero_c[2].most_common(5)
```

Out[10]:
```
[(('STOP', 'LIGHTS'), 991),
 (('Stop', 'Lights'), 406),
 (('LIGHTS', 'STOP'), 340),
 (('WINDOW', 'TINT'), 314),
 (('TAG', 'LIGHTS'), 206)]
```

In [11]:
```
sero_c[4].most_common(5)
```

Out[11]:
```
[(('STOP', 'LIGHTS', 'STOP', 'LIGHTS'), 277),
 (('STOP', 'LIGHTS', 'Stop', 'Lights'), 129),
 (('Stop', 'Lights', 'STOP', 'LIGHTS'), 110),
 (('LIGHTS', 'STOP', 'LIGHTS', 'STOP'), 98),
 (('WINDOW', 'TINT', 'STOP', 'LIGHTS'), 92)]
```

In [12]:
```
war_c[2].most_common(5)
```

Out[12]:
```
[(('FAILURE', 'TO'), 12498),
 (('DRIVER', 'FAILURE'), 6194),
 (('TO', 'DISPLAY'), 4713),
 (('SPEED', 'LIMIT'), 4415),
 (('LIMIT', 'OF'), 4306)]
```

In [13]:
```
war_c[4].most_common(10)
```

Out[13]:
```
[(('EXCEEDING', 'THE', 'POSTED', 'SPEED'), 4294),
 (('THE', 'POSTED', 'SPEED', 'LIMIT'), 4294),
 (('POSTED', 'SPEED', 'LIMIT', 'OF'), 4294),
 (('DRIVER', 'FAILURE', 'TO', 'OBEY'), 4191),
 (('FAILURE', 'TO', 'OBEY', 'PROPERLY'), 3999),
 (('TO', 'OBEY', 'PROPERLY', 'PLACED'), 3999),
 (('OBEY', 'PROPERLY', 'PLACED', 'TRAFFIC'), 3999),
 (('PROPERLY', 'PLACED', 'TRAFFIC', 'CONTROL'), 3999),
 (('PLACED', 'TRAFFIC', 'CONTROL', 'DEVICE'), 3999),
 (('TRAFFIC', 'CONTROL', 'DEVICE', 'INSTRUCTIONS'), 3999)]
```

Using text classification on decription column to predict class and use that as a feature

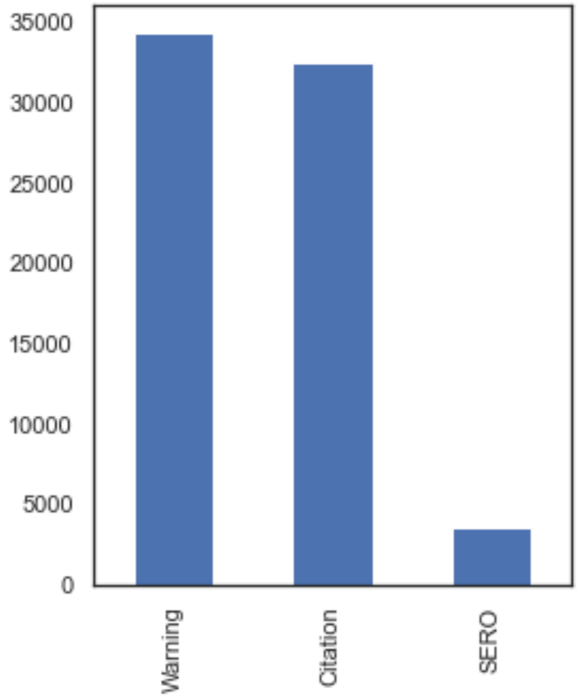
In [14]:
```
mp = {
    'Citation': 0,
    'SERO': 1,
    'Warning': 2,
}

rev_mp = {
    0: 'Citation',
    1: 'SERO',
    2: 'Warning',
}

X_train, X_test, y_train, y_test = train_test_split(data['Description'], data
count_vect = CountVectorizer(max_features=1000, ngram_range=(1, 3), analyzer 
tfidf_transformer = TfidfTransformer()
X_train_counts = count_vect.fit_transform(X_train)
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
clf = MultinomialNB().fit(X_train_tfidf, y_train)

data['Description Pred'] = data['Description'].apply(lambda x: mp[clf.predict
```

In [15]:
```
X_test_counts = count_vect.fit_transform(X_test)
X_test_tfidf = tfidf_transformer.fit_transform(X_test_counts)
clf.score(X_test_counts, y_test)
```

Out[15]:
```
0.7102644299118567
```

Exploratory Data Analysis

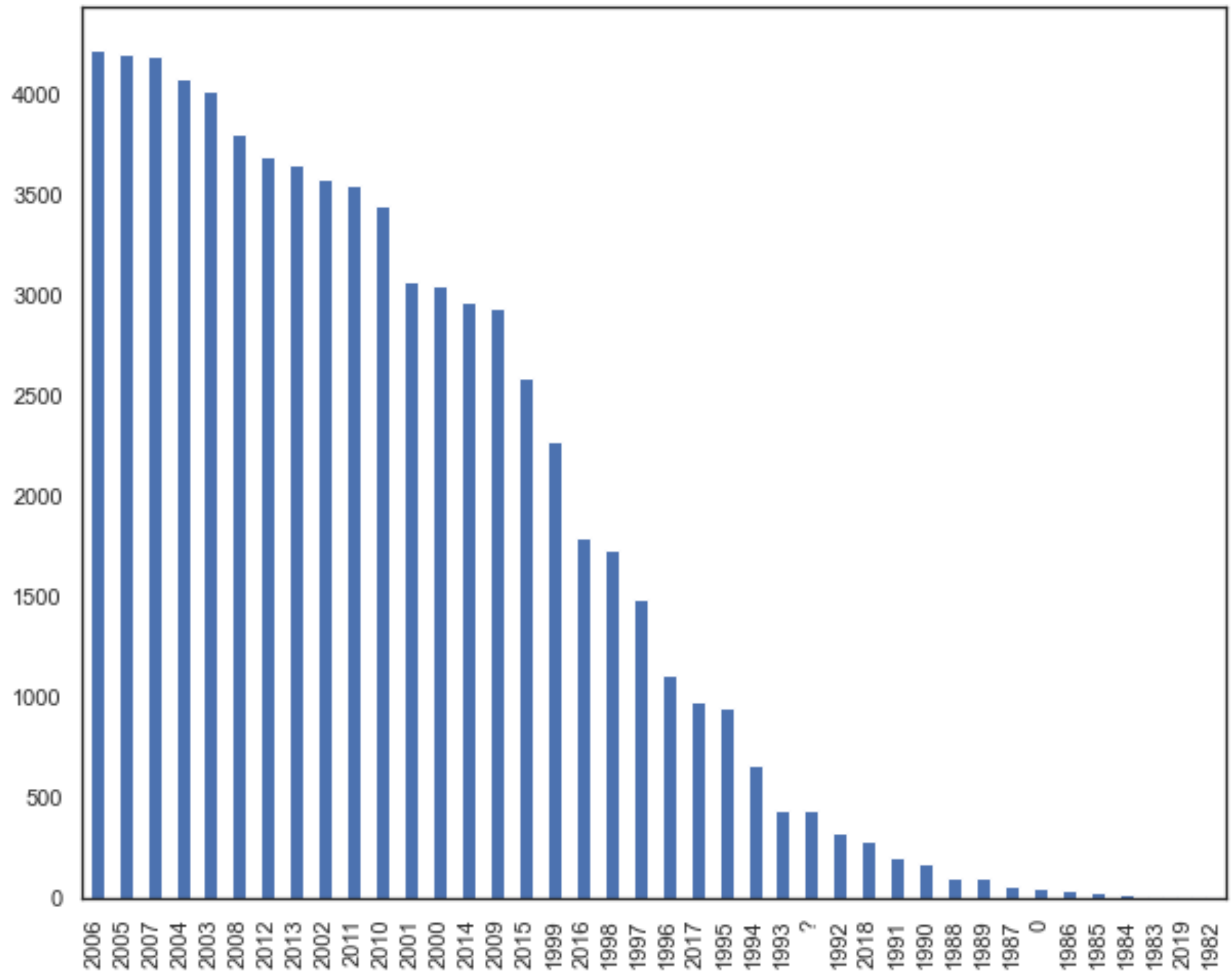
In [16]:
```
plt.figure(figsize=(4, 5))
_ = data['Violation Type'].value_counts().plot.bar()
```



In [17]:
```
data['Driver City'].value_counts()
```

Out[17]:
```
'SILVER SPRING'    17691
GAITHERSBURG        7224
GERMANTOWN          5838
ROCKVILLE           5804
WASHINGTON          2133
                    ...
SPRING                 1
'W HYATTSVILLE'        1
'SN LUIS OBISP'        1
GRUNDY                 1
MEQUON                 1
Name: Driver City, Length: 1890, dtype: int64
```

In [18]:
```
_ = data['Year'].value_counts().head(40).plot.bar()
```



In [19]:

```
data['Model'].value_counts()
```

Out[19]:

```
4S                 7780
TK                 4604
ACCORD             2696
CIVIC              2392
CAMRY              2375
                   ...
'GRAN MARQ'           1
'MILLENIA 4S'         1
'SLK 230'             1
LS450                 1
CONVERT               1
Name: Model, Length: 3828, dtype: int64
```

Data Preprocessing

In [20]:
```
df = data.copy()
ordinal_enc = OrdinalEncoder()

cols = ['Description', 'Belts', 'Personal Injury', 'Property Damage',
        'Commercial License', 'Commercial Vehicle', 'State', 'VehicleType',
        'Year', 'Make', 'Model', 'Color', 'Contributed To Accident',
        'Driver Race', 'Gender', 'Driver City', 'Drive State', 'Violation Type

df[cols] = ordinal_enc.fit_transform(data[cols])

df.head()
```

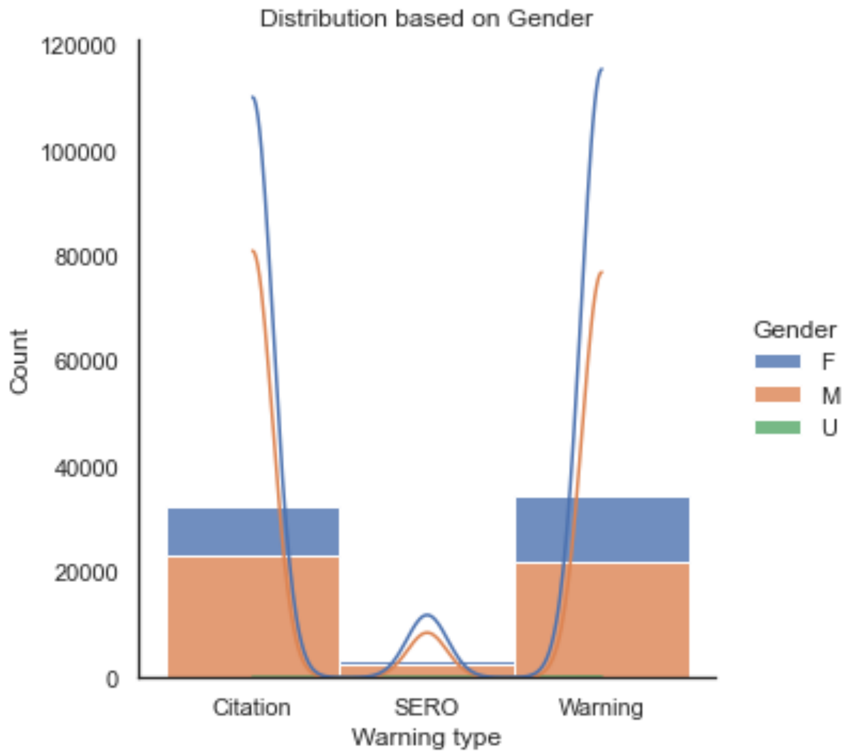
Out[20]:

	Description	Belts	Personal Injury	Property Damage	Commercial License	Commercial Vehicle	State	VehicleType	Year
0	90.0	0.0	0.0	0.0	0.0	0.0	29.0	1.0	64.0
1	150.0	0.0	0.0	0.0	0.0	0.0	22.0	1.0	66.0
2	373.0	0.0	0.0	0.0	0.0	0.0	22.0	1.0	51.0
3	1799.0	0.0	0.0	0.0	0.0	0.0	22.0	1.0	63.0
4	90.0	0.0	0.0	0.0	1.0	0.0	22.0	1.0	61.0

Visualizations

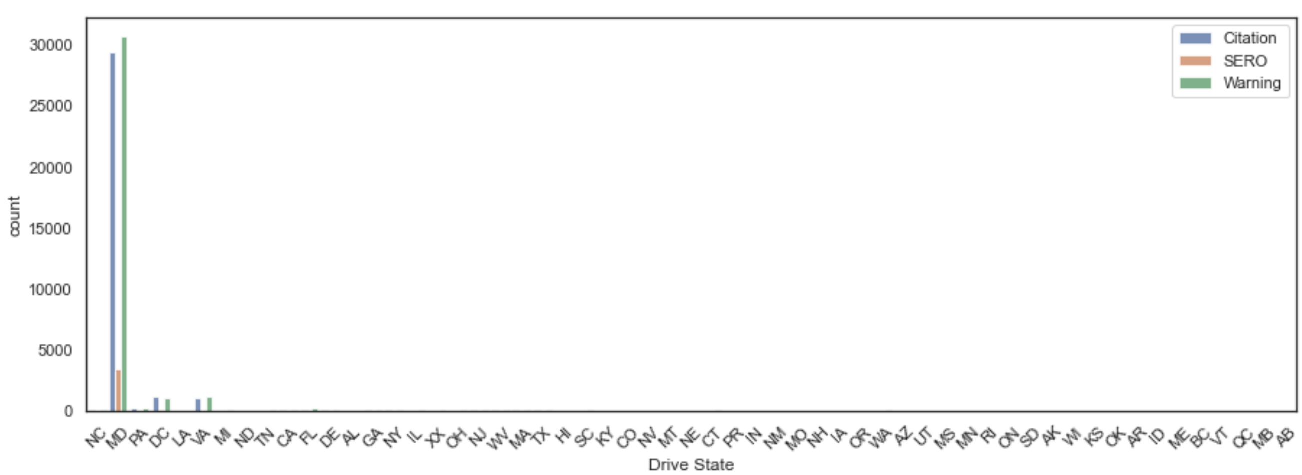
In [21]:
```
_ = sns.displot(
    x = 'Violation Type',
    hue = 'Gender',
    kde = True,
    data = data,
    multiple = 'stack',
    alpha = 0.8,
)
plt.title('Distribution based on Gender')
plt.xlabel('Warning type')
plt.ylabel('Count')
```

Out[21]: Text(-7.273118402777776, 0.5, 'Count')

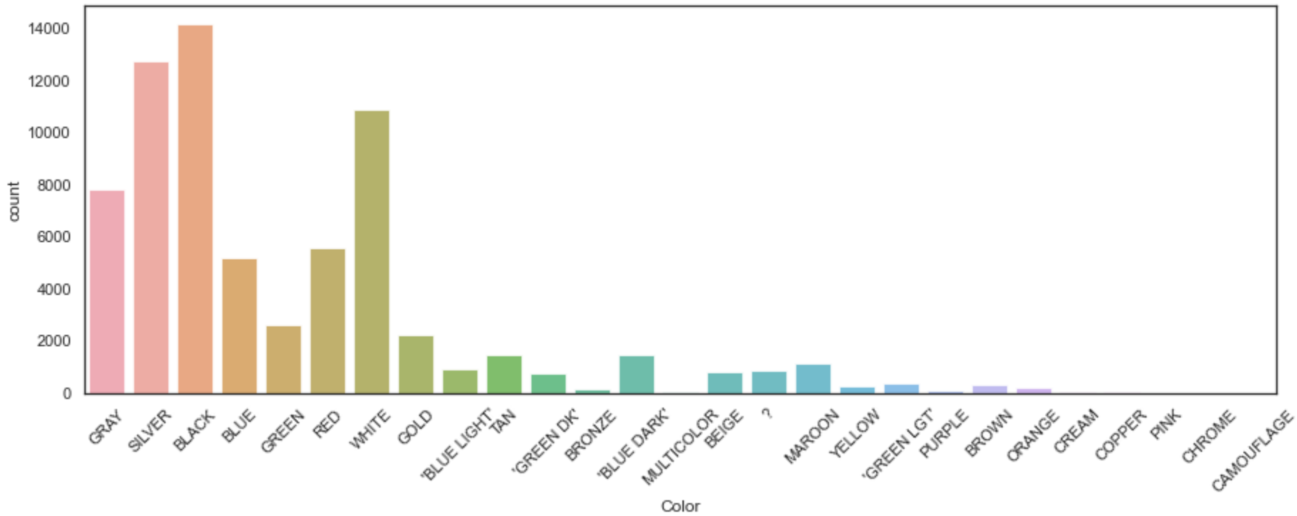


In [22]:

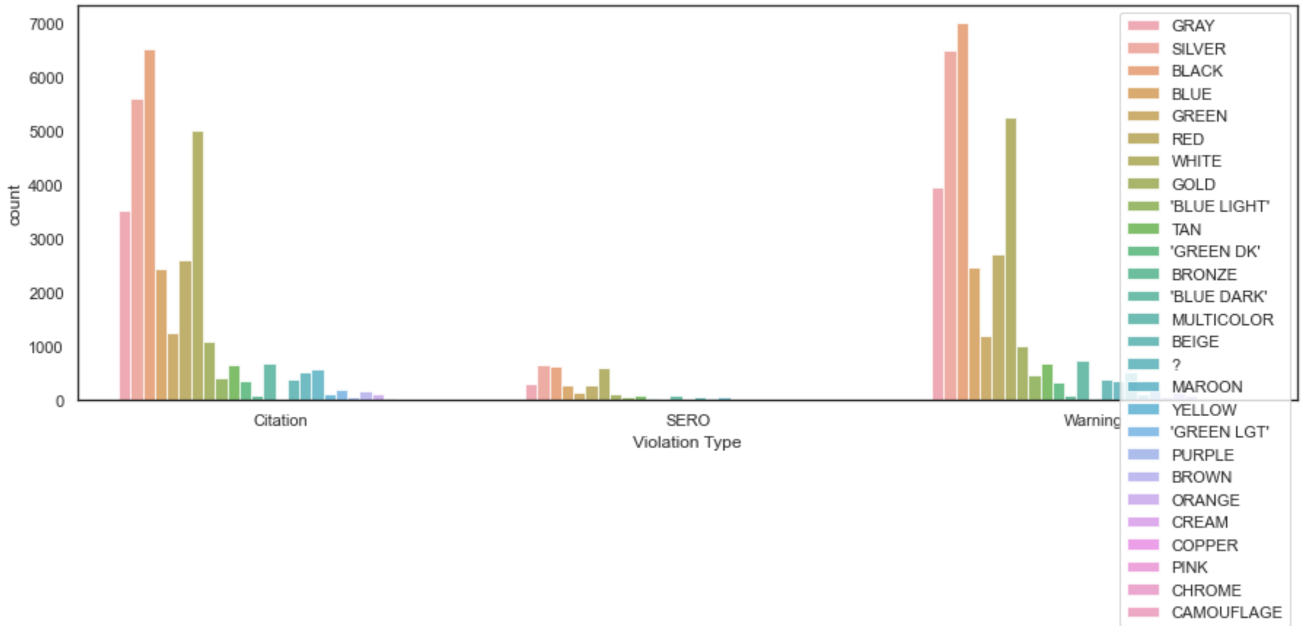
```
plt.figure(figsize=(15, 5))
_ = sns.countplot(
    x = 'Drive State',
    hue = 'Violation Type',
    data = data,
    alpha = 0.8,
)
_ = plt.xticks(rotation=45)
_ = plt.legend(loc=1)
```



In [23]:
```
plt.figure(figsize=(15, 5))
_ = sns.countplot(
    x = 'Color',
    data = data,
    alpha = 0.8,
)
_ = plt.xticks(rotation=45)
```

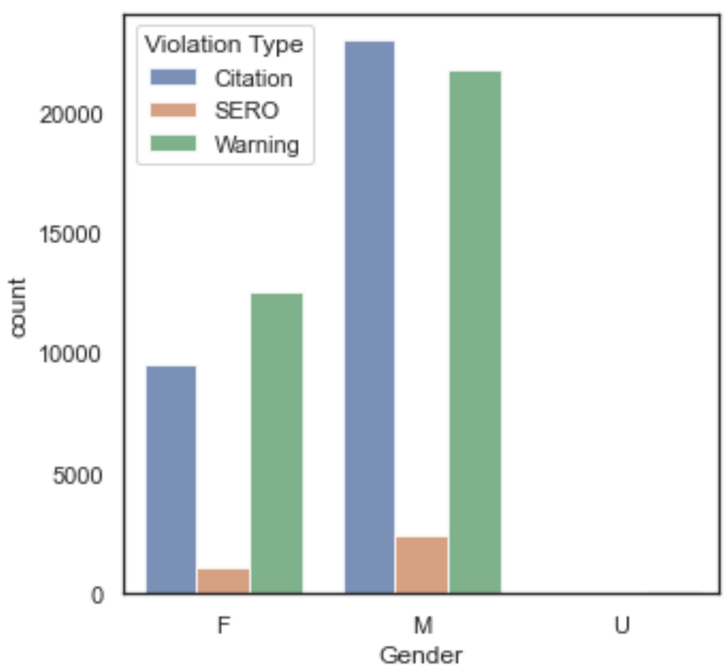


In [24]:
```
plt.figure(figsize=(15, 5))
_ = sns.countplot(
    hue = 'Color',
    x = 'Violation Type',
    data = data,
    alpha = 0.8,
)
_ = plt.legend(loc=1)
```



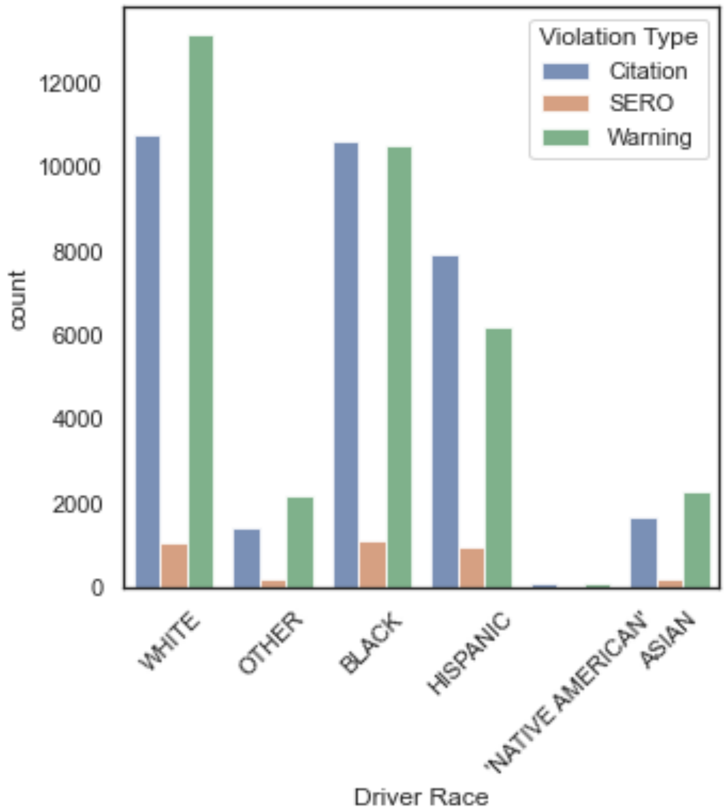
Females have got lesser Citations than Males

In [35]:
```
plt.figure(figsize=(5, 5))
_ = sns.countplot(
    x = 'Gender',
    hue = 'Violation Type',
    data = data,
    alpha = 0.8,
)
```



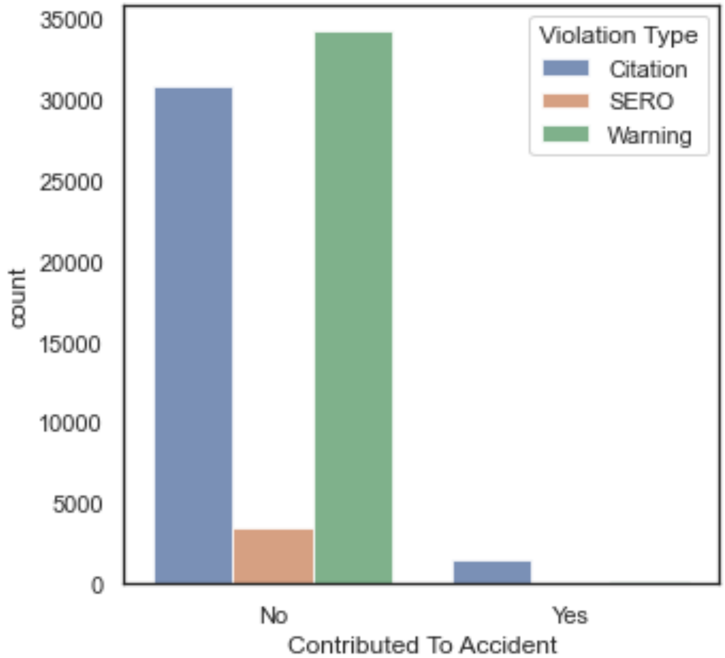
In [26]:

```
plt.figure(figsize=(5, 5))
_ = sns.countplot(
    x = 'Driver Race',
    hue = 'Violation Type',
    data = data,
    alpha = 0.8,
)
_ = plt.xticks(rotation=45)
```



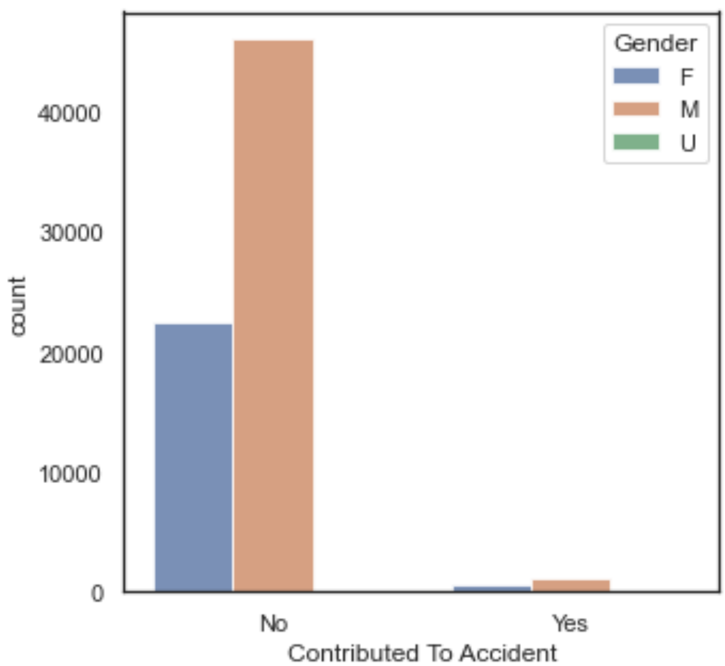
In [39]:

```
plt.figure(figsize=(5, 5))
_ = sns.countplot(
    x = 'Contributed To Accident',
    hue = 'Violation Type',
    data = data,
    alpha = 0.8,
)
```



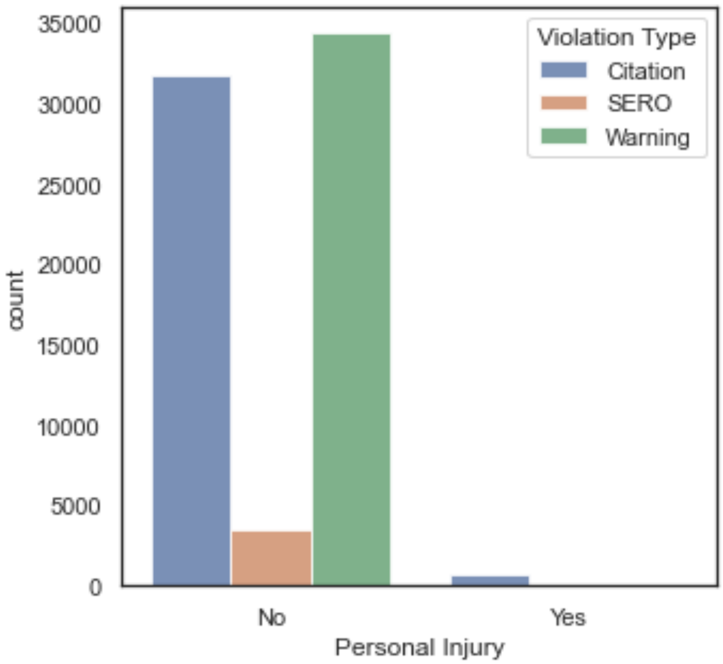
In [38]:

```
plt.figure(figsize=(5, 5))
_ = sns.countplot(
    x = 'Contributed To Accident',
    hue = 'Gender',
    data = data,
    alpha = 0.8,
)
```



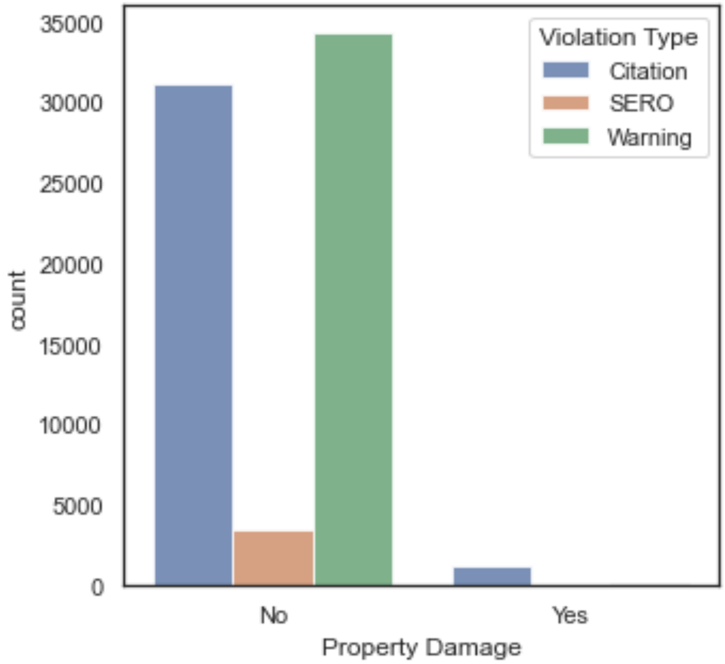
In [41]:

```
plt.figure(figsize=(5, 5))
_ = sns.countplot(
    x = 'Personal Injury',
    hue = 'Violation Type',
    data = data,
    alpha = 0.8,
)
```



In [42]:

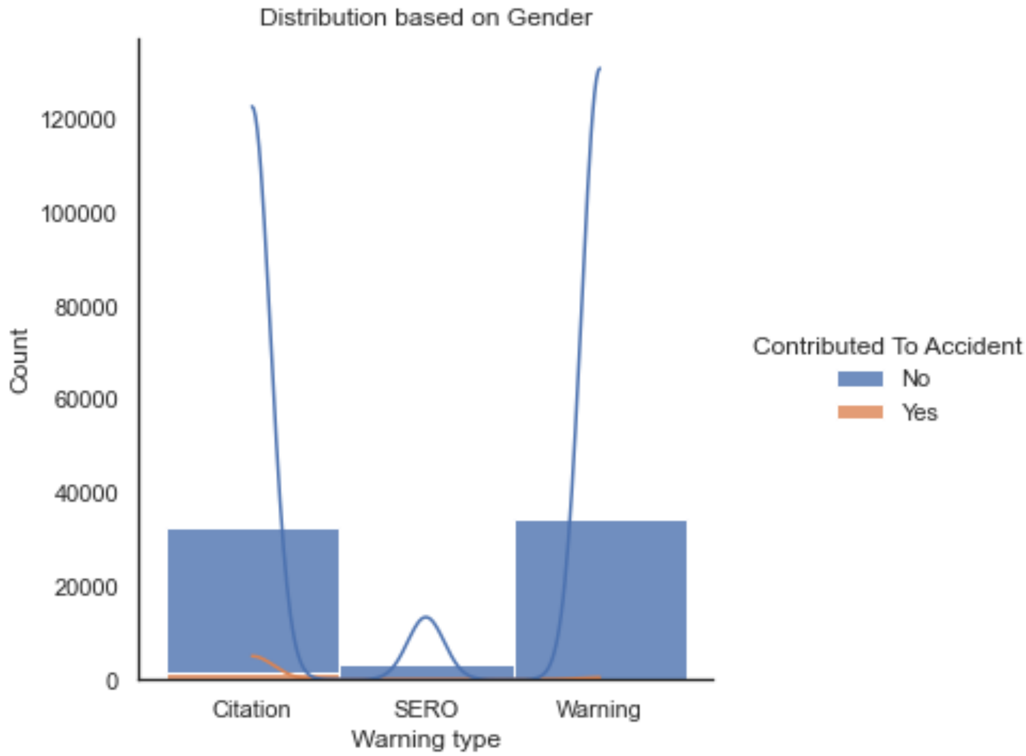
```
plt.figure(figsize=(5, 5))
_ = sns.countplot(
    x = 'Property Damage',
    hue = 'Violation Type',
    data = data,
    alpha = 0.8,
)
```



In [27]:

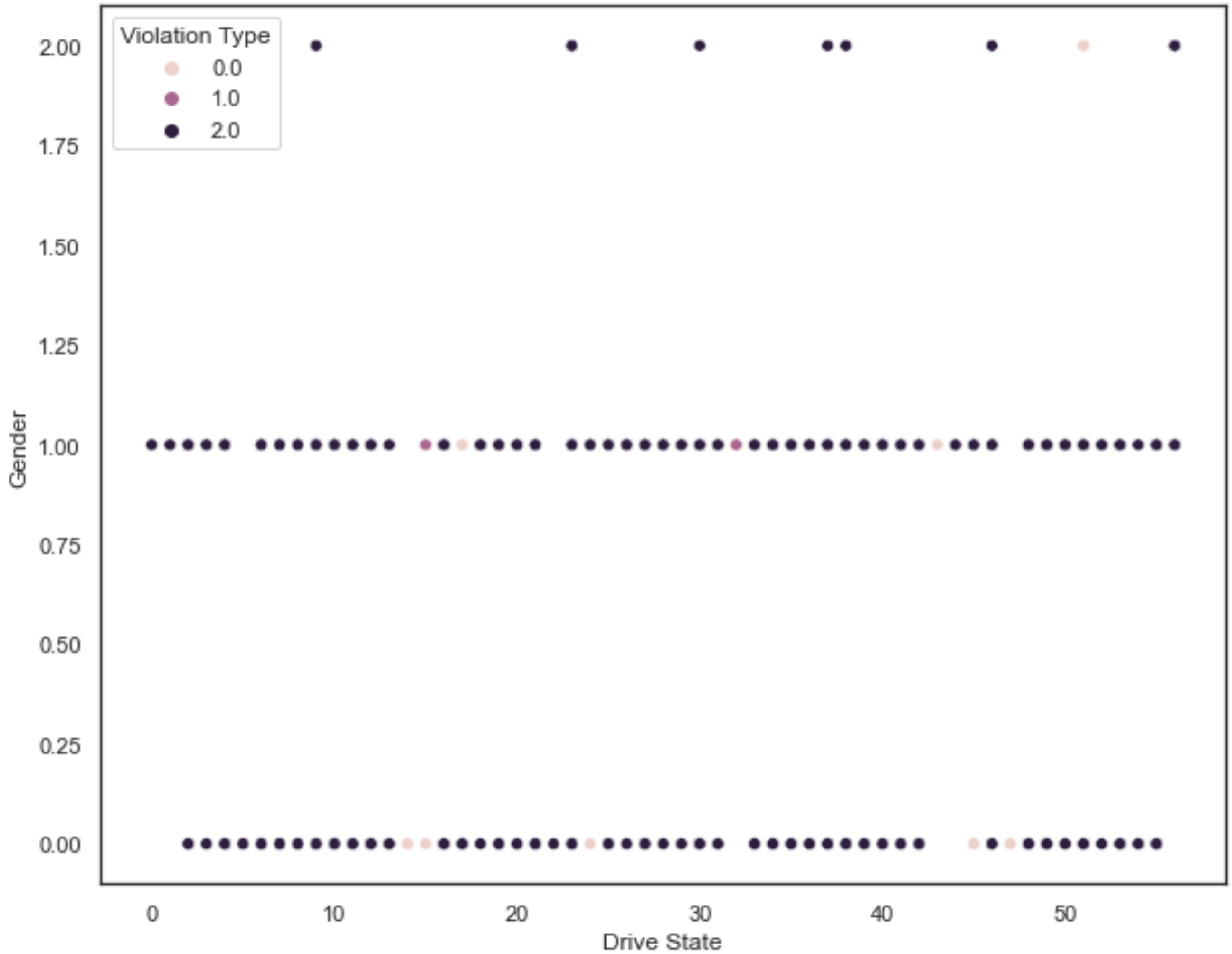
```
_ = plt.figure(figsize=(5, 5))
_ = sns.displot(
    x = 'Violation Type',
    hue = 'Contributed To Accident',
    kde = True,
    data = data,
    multiple = 'stack',
    alpha = 0.8,
)
_ = plt.title('Distribution based on Gender')
_ = plt.xlabel('Warning type')
_ = plt.ylabel('Count')
```

```
<Figure size 360x360 with 0 Axes>
```



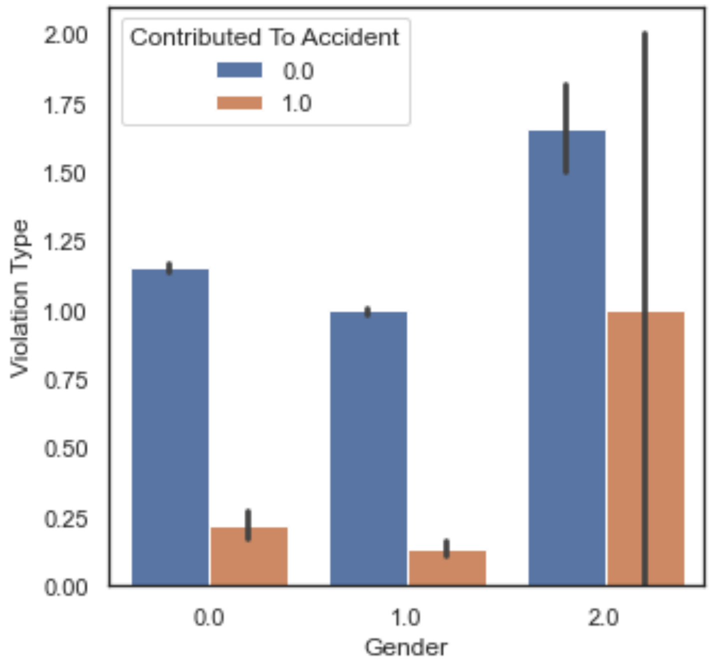
In [28]:

```
_ = sns.scatterplot(
    x = 'Drive State',
    y = 'Gender',
    hue = 'Violation Type',
    data = df,
)
```



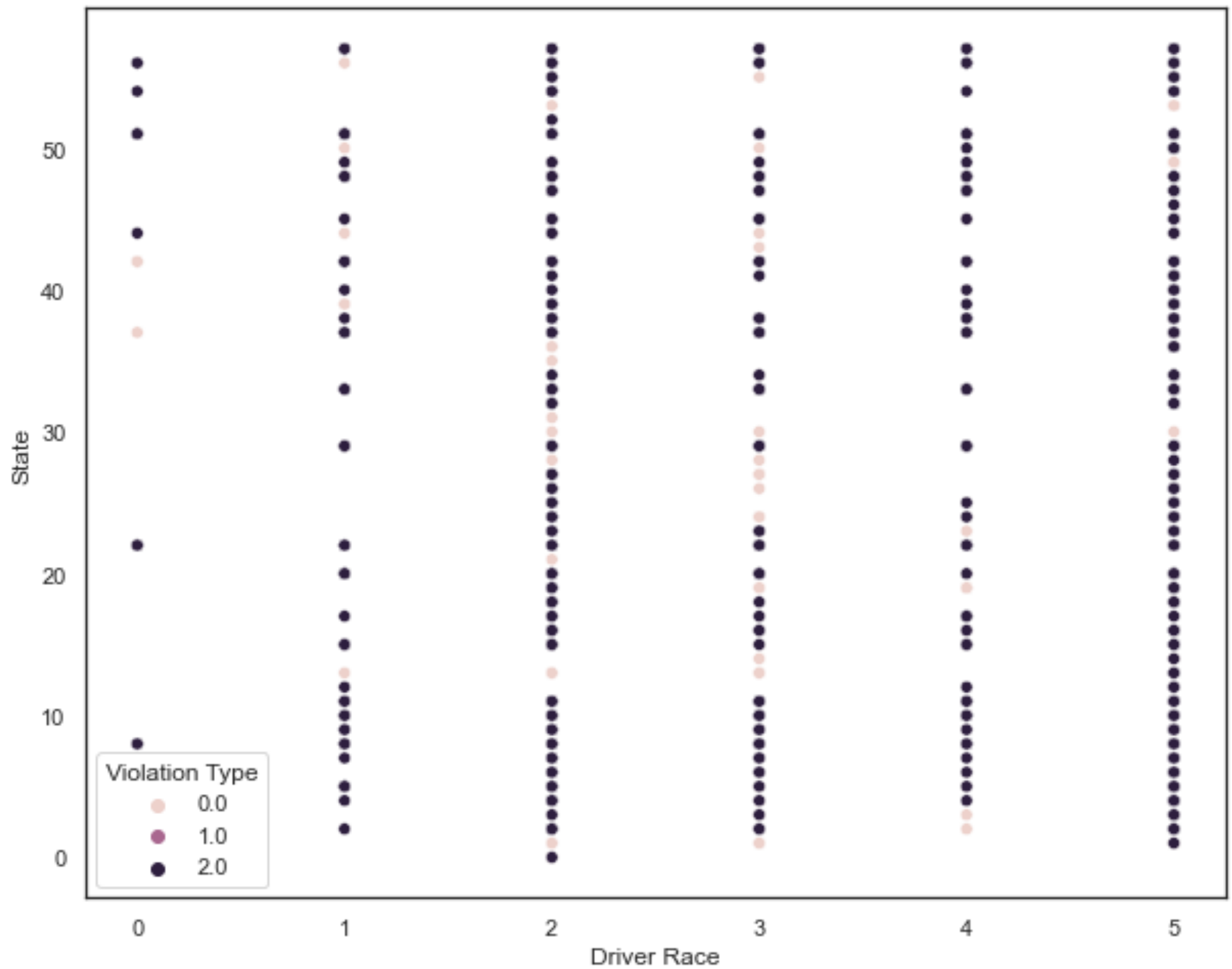
In [29]:

```
plt.figure(figsize=(5, 5))
_ = sns.barplot(
    x = 'Gender',
    y = 'Violation Type',
    hue = 'Contributed To Accident',
    data = df,
)
```

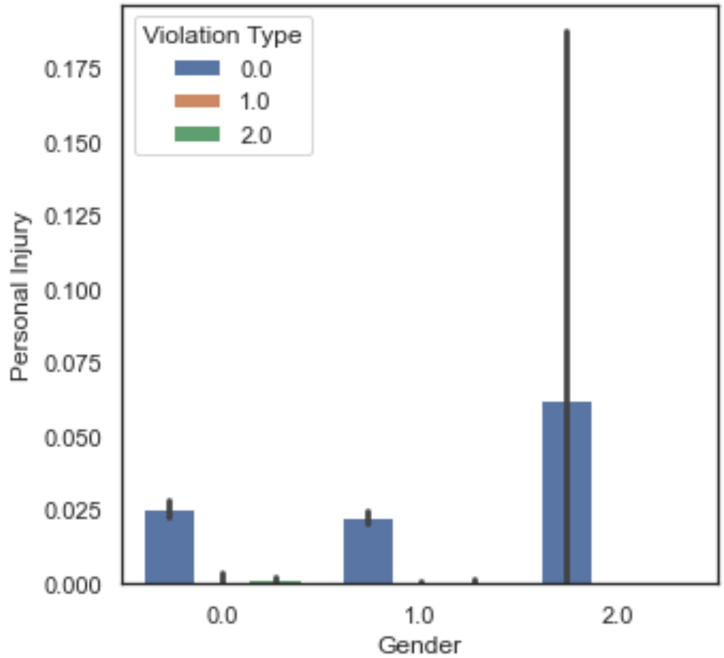


In [30]:

```
_ = sns.scatterplot(
    y = 'State',
    x = 'Driver Race',
    hue = 'Violation Type',
    data = df,
)
```

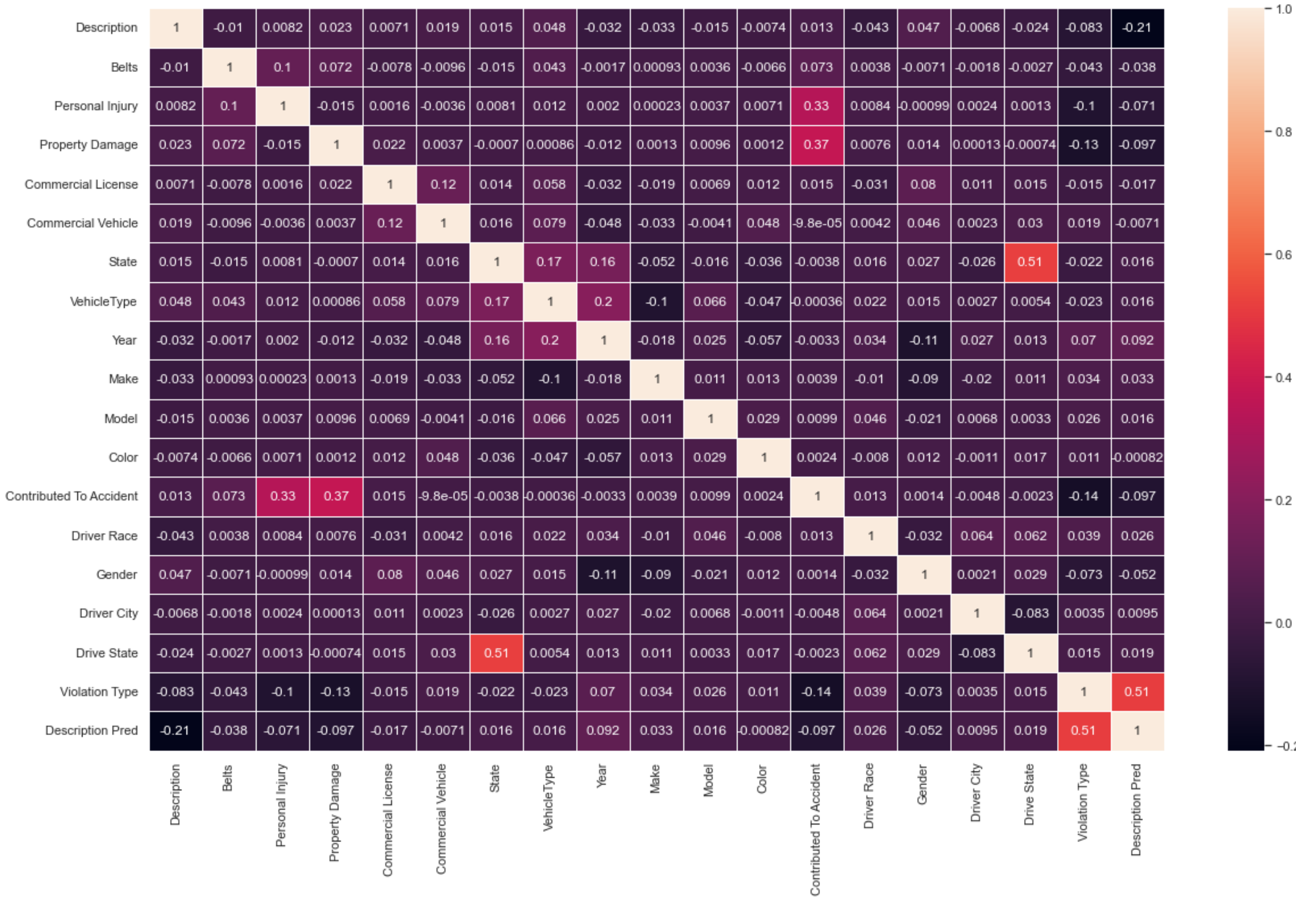


In [31]:
```
plt.figure(figsize=(5, 5))
_ = sns.barplot(
    hue = 'Violation Type',
    y = 'Personal Injury',
    x = 'Gender',
    data = df,
)
```



In [43]:

```
plt.figure(figsize = (20, 12))
corr = df.corr()
sns.heatmap(corr, annot = True, linewidths = 1)
plt.show()
```



In [123…

```
df.corr()['Violation Type'].sort_values()
```

Out[123…

```
Contributed To Accident   -0.139049
Property Damage           -0.125448
Personal Injury           -0.101189
Description               -0.082798
Gender                    -0.072796
Belts                     -0.042997
VehicleType               -0.023455
State                     -0.021977
Commercial License        -0.015411
Driver City                0.003515
Color                      0.010616
Drive State                0.014618
Commercial Vehicle         0.018542
Model                      0.025527
Make                       0.033530
Driver Race                0.039167
Year                       0.070488
Description Pred           0.513349
Violation Type             1.000000
Name: Violation Type, dtype: float64
```

Modelling

In [206…

```
target_col = 'Violation Type'
target = df[target_col]
select_cols = [
    'Contributed To Accident',
    'Description',
    'Belts',
    'Property Damage',
    'Personal Injury',
    'Description Pred',
]
X_train, x_test, y_train, y_test = train_test_split(df[select_cols], target, 
```

Random Forest Classifier

In [207…

```
forest = RandomForestClassifier(n_jobs=-1)
forest.fit(X_train, y_train)
forest_y = forest.predict(x_test)

test_score = forest.score(x_test, y_test)
print(f"Accuracy of the RandomForestClassifier: {test_score:.4f}")
```

```
Accuracy of the RandomForestClassifier: 0.7912
```

In [112…

```
cv_results = cross_validate(forest, df[select_cols], target)
scores = cv_results["test_score"]
print(f"Accuracy score via cross-validation:\n"
      f"{scores.mean():.3f} +/- {scores.std():.3f}")
```

```
Accuracy score via cross-validation:
0.790 +/- 0.003
```

In [113…

```
print(classification_report(y_test, forest_y, target_names=['Citation', 'SERO
```

```
              precision    recall  f1-score   support

    Citation       0.86      0.65      0.74      6456
        SERO       0.99      1.00      0.99       694
     Warning       0.74      0.90      0.81      6918

    accuracy                           0.79     14068
   macro avg       0.86      0.85      0.85     14068
weighted avg       0.81      0.79      0.79     14068
```

In [212…

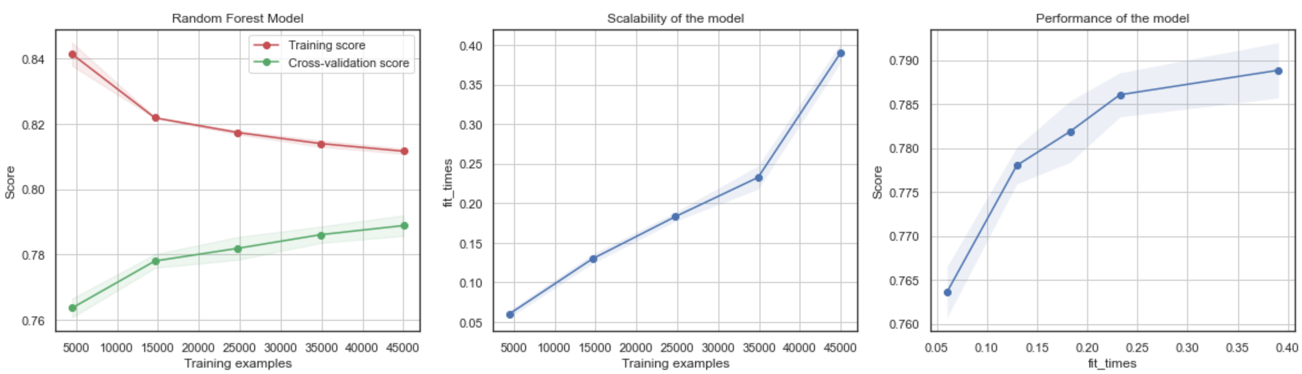
```
res = []
for c in [0, 1, 2]:
    prec,recall,_,_ = precision_recall_fscore_support(
        np.array(y_test) == c,
        np.array(forest_y) == c,
        pos_label=True,
        average=None,
    )
    res.append([rev_mp[c], recall[0], recall[1]])

pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

Out[212…

	class	sensitivity	specificity
0	Citation	0.905281	0.657993
1	SERO	0.999477	0.988489
2	Warning	0.690952	0.895619

In [40]:

```
_ = plot_learning_curve(forest, 'Random Forest Model', X_train, y_train)
```



KNN

In [208…

```
knn = KNeighborsClassifier(100)
knn.fit(X_train, y_train)
knn_y = knn.predict(x_test)

test_score = knn.score(x_test, y_test)
print(f"Accuracy of the KNN: {test_score:.2f}")
```

```
Accuracy of the KNN: 0.78
```

In [209…

```
cv_results = cross_validate(knn, df[select_cols], target)
scores = cv_results["test_score"]
print(f"Accuracy score via cross-validation:\n"
      f"{scores.mean():.3f} +/- {scores.std():.3f}")
```

```
Accuracy score via cross-validation:
0.641 +/- 0.003
```

In [210…

```
print(classification_report(y_test, knn_y, target_names=['Citation', 'SERO',
```

```
              precision    recall  f1-score   support

    Citation       0.85      0.63      0.73      6456
        SERO       0.92      0.94      0.93       695
     Warning       0.72      0.89      0.80      6917

    accuracy                           0.78     14068
   macro avg       0.83      0.82      0.82     14068
weighted avg       0.79      0.78      0.77     14068
```

In [211…

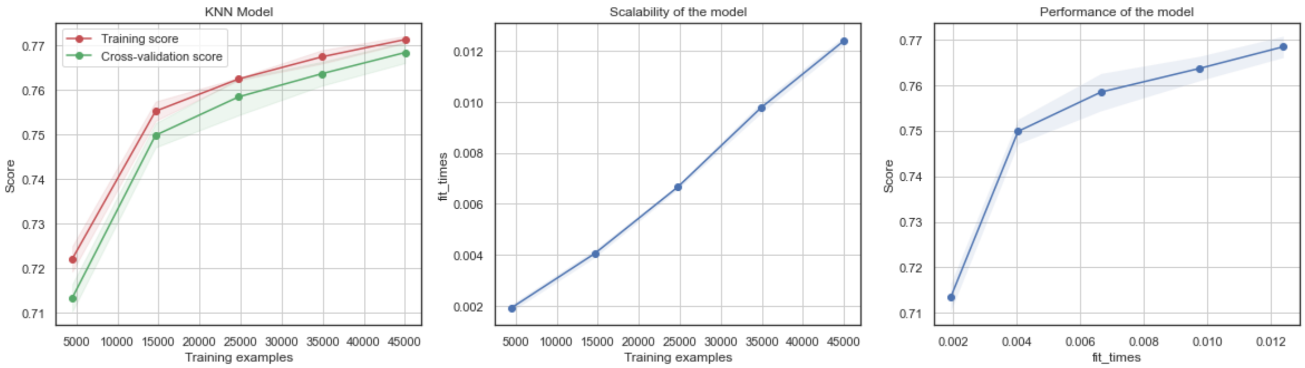
```
res = []
for c in [0, 1, 2]:
    prec,recall,_,_ = precision_recall_fscore_support(
        np.array(y_test) == c,
        np.array(knn_y) == c,
        pos_label=True,
        average=None,
    )
    res.append([rev_mp[c], recall[0], recall[1]])

pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

Out[211…

	class	sensitivity	specificity
0	Citation	0.905413	0.632900
1	SERO	0.996037	0.939568
2	Warning	0.667319	0.893017

In [45]:

```
_ = plot_learning_curve(knn, 'KNN Model', X_train, y_train)
```



SVM

In [46]:

```
svm = SVC()
svm.fit(X_train, y_train)
svm_y = svm.predict(x_test)

test_score = svm.score(x_test, y_test)
print(f"Accuracy of the SVM: {test_score:.2f}")
```

```
Accuracy of the SVM: 0.63
```

In [47]:

```
cv_results = cross_validate(svm, df[select_cols], target)
scores = cv_results["test_score"]
print(f"Accuracy score via cross-validation:\n"
      f"{scores.mean():.3f} +/- {scores.std():.3f}")
```

```
Accuracy score via cross-validation:
0.634 +/- 0.002
```

In [48]:

```
print(classification_report(y_test, svm_y, target_names=['Citation', 'SERO',
```

```
              precision    recall  f1-score   support

    Citation       0.67      0.47      0.55      6495
        SERO       0.68      0.94      0.79       709
     Warning       0.61      0.76      0.67      6864

    accuracy                           0.63     14068
   macro avg       0.66      0.72      0.67     14068
weighted avg       0.64      0.63      0.63     14068
```

In [49]:

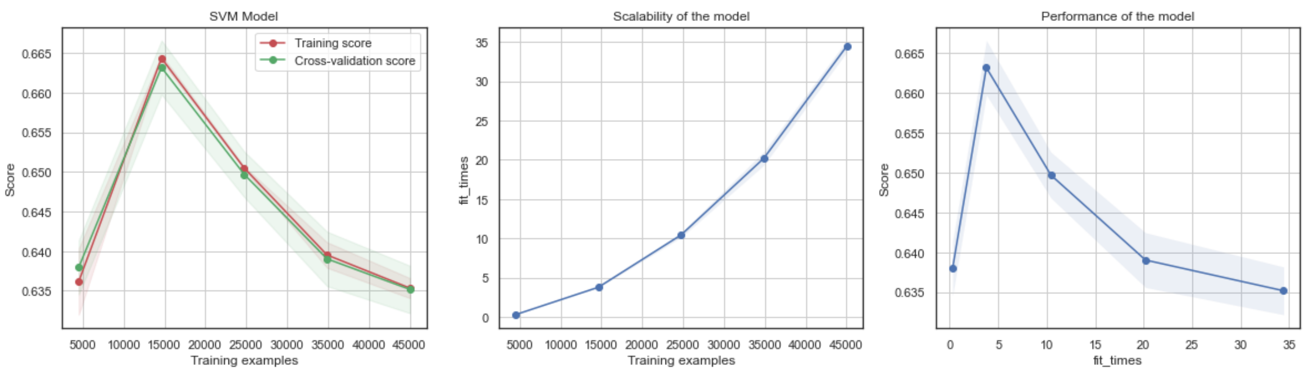
```
res = []
for c in [0, 1, 2]:
    prec,recall,_,_ = precision_recall_fscore_support(
        np.array(y_test) == c,
        np.array(svm_y) == c,
        pos_label=True,
        average=None,
    )
    res.append([rev_mp[c], recall[0], recall[1]])

pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

Out[49]:

	class	sensitivity	specificity
0	Citation	0.804833	0.471286
1	SERO	0.977094	0.935120
2	Warning	0.534148	0.758159

In [50]:

```
_ = plot_learning_curve(svm, 'SVM Model', X_train, y_train)
```



Part - C

"Exascale Computing and Big Data" - by Daniel A. Reed and Jack Dongarra

Motivation

The paper begins by discussing some of the major challenges that are currently being solved by computational models, such as climate modeling and predicting black hole behavior, and then goes on to discuss how solving these problems require some investment in advanced computing research, design, and global collaboration.

Scientific and engineering opportunities with the rise of big data analytics

The authors begin by discussing how big data analytical systems were critical in the discovery of the Higgs boson particle, and they go on to discuss how various fields other than physical sciences are leveraging the potential of big data. It has been reported that some astronomers are now frequently querying existing datasets to discover previously unknown patterns and trends, which is very interesting to know. Big data is also playing an important role in the healthcare domain, with many of these techniques being increasingly used to conduct comparative and longitudinal analyses of cancer treatment regimes. The author also discussed some lesser-known fields where big data analytics is being used; currently, semantic graph visualization tools and recommender systems are being used to identify relevant topics and suggest relevant papers for study.

Technical challenges towards high-end computing

The study explored a wide range of technological obstacles, including software, hardware, and scaling challenges. The author references a survey undertaken by the US Department of Energy, which claims that new designs and technologies are required to reduce the energy requirement to a more manageable and economically reasonable level in order to satisfy the present compute power demand.

Outlining some of the strategies that can address these issues

- Locality-aware algorithms (MapReduce toolkits) and software will be needed to maximize computation performance and reduce energy needs
- New memory technologies, including processor in-memory, stacked memory, and nonvolatile memory approaches are needed to minimize data movement and energy use
- More expressive programming methods are required to deal with the parallelism and frequent errors of exascale computing systems.
- Given the need for systemic resilience in the face of component failures, systems should incorporate a wide range of resilience mechanisms, such as geo-distribution, automated restart and failover, failure injection, and introspective monitoring.