INT375

PROJECT REPORT

(Project Semester January-April 2025)

*(*Analyzing TLC New Driver License Application Trends and Status Patterns in NYC*)*

Submitted by

Satish Raut

Registration No : **12301144**

Programme and Section : **B.Tech (CSE) & K23ED**

Course Code : **INT375**

Under the Guidance of

Dr. Dhiraj Kapila (UID: 23509)

Discipline of CSE/IT

Lovely School of Computer Science and Engineering

Lovely Professional University, Phagwara

CERTIFICATE

This is to certify that **Satish Raut** bearing Registration no. **12301144** has completed **INT 375** project titled, **"Analyzing TLC New Driver License Application Trends and Status Patterns in NYC"** under my guidance and supervision. To the best of my knowledge, the present work is the result of his/her original development, effort and study.

Signature and Name of the Supervisor

Designation of the Supervisor

School of …….

Lovely Professional University

Phagwara, Punjab.

Date:

DECLARATION

I, Satish Raut, student of B.Tech under CSE/IT Discipline at, Lovely Professional University, Punjab, hereby declare that all the information furnished in this project report is based on my own intensive work and is genuine.

Date: 10 / 04/ 2025 Signature

Registration No. **12301144** **Satish Raut**

1. Introduction

The rise of app-based transportation services in New York City has significantly increased the demand for Taxi and Limousine Commission (TLC) licenses. To meet this demand and regulate the drivers, TLC maintains a detailed application process. This project focuses on analyzing the dataset of new TLC driver license applications in NYC, aiming to understand the trends, timelines, and factors affecting the approval, rejection, or delay of applications. Using Python and libraries such as Pandas, NumPy, Matplotlib, and Seaborn, we carried out a comprehensive exploratory data analysis (EDA) to extract meaningful insights. The ultimate goal is to help stakeholders understand the process dynamics, identify common bottlenecks, and improve the efficiency of driver license processing.

2. Source of dataset

https://data.cityofnewyork.us/Transportation/TLC-New-Driver-Application-Status/dpec-ucu7/about_data

3. EDA Process (Exploratory Data Analysis)

The Exploratory Data Analysis (EDA) process is a crucial step in any data science project.
It involves the following steps:

Step 1: Data Inspection

Step 2: Data Cleaning

Step 3: Data Transformation

Step 4: Univariate and Bivariate Analysis

Step 5: Visualization

Step 6: Interpretation

4. Analysis on dataset (for each analysis)

i. Introduction

To extract actionable insights from the TLC application dataset, we formulated a set of key analytical questions. These questions are designed to uncover application trends, evaluate document processing statuses, and examine patterns related to delays or rejections.

ii. General Description

The dataset contains fields such as application number, type, application date, status, and several columns representing various required documents and steps (like drug test, interview schedule, WAV course, etc.). Each row corresponds to an individual application and captures the state of its progress or completion.

```
      ## "🔍📊 Analyzing TLC New Driver License Application Trends and Status Patterns in NYC 🚀" ##
#  🌀 ──────────────────────────────────────── ★·☆·★ ──────────────────────────────────────── 🌀  #
```

1. Used Libraries and Process of Loading the dataset :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm, ttest_1samp, ttest_ind, ttest_rel

# └➤ 1. ⏳ Load the data
#┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈┈

df = pd.read_excel("TLC_New_Driver_Application_Status.xlsx")
print(df)
```

Output:

```
        App No   ...          Last Updated
0      6107496   ...  2025-04-01 05:00:00
1      6104368   ...  2025-04-01 05:00:00
2      6107004   ...  2025-04-01 05:00:00
3      6103038   ...  2025-04-01 05:00:00
4      6104869   ...  2025-04-01 05:00:00
...        ...   ...                  ...
3195   6107465   ...  2025-04-01 05:00:00
3196   6104313   ...  2025-04-01 05:00:00
3197   6104068   ...  2025-04-01 05:00:00
3198   6106198   ...  2025-04-01 05:00:00
3199   6106613   ...  2025-04-01 05:00:00

[3200 rows x 12 columns]
```

2. 🔎 Basic information about the dataset:

```
# └➤ 2. 🔎 Basic information
#------------------------------------------------------------
print("Dataset Info:\n", df.info)
print("First 5 column from the top: \n", df.head())
print("First 5 column from the bottom: \n", df.tail())

print("Describe: \n",df.describe());
print("Shape: ", df.shape)
print("Columns: \n", df.columns.tolist())
print("Data type of each Column: \n", df.dtypes)
```

Output:

```
Dataset Info:
 <bound method DataFrame.info of        App No  ...        Last Updated
0     6107496  ... 2025-04-01 05:00:00
1     6104368  ... 2025-04-01 05:00:00
2     6107004  ... 2025-04-01 05:00:00
3     6103038  ... 2025-04-01 05:00:00
4     6104869  ... 2025-04-01 05:00:00
...       ...  ...                 ...
3195  6107465  ... 2025-04-01 05:00:00
3196  6104313  ... 2025-04-01 05:00:00
3197  6104068  ... 2025-04-01 05:00:00
3198  6106198  ... 2025-04-01 05:00:00
3199  6106613  ... 2025-04-01 05:00:00

[3200 rows x 12 columns]>
First 5 column from the top:
     App No  ...        Last Updated
0  6107496  ... 2025-04-01 05:00:00
1  6104368  ... 2025-04-01 05:00:00
2  6107004  ... 2025-04-01 05:00:00
3  6103038  ... 2025-04-01 05:00:00
4  6104869  ... 2025-04-01 05:00:00

[5 rows x 12 columns]
First 5 column from the bottom:
        App No  ...        Last Updated
3195  6107465  ... 2025-04-01 05:00:00
3196  6104313  ... 2025-04-01 05:00:00
3197  6104068  ... 2025-04-01 05:00:00
3198  6106198  ... 2025-04-01 05:00:00
3199  6106613  ... 2025-04-01 05:00:00
```

```
Describe:
             App No             App Date        Last Updated
count  3.200000e+03                 3200                3200
mean   6.103729e+06  2025-01-27 11:32:33  2025-04-01 05:00:00
min    5.748366e+06  1997-04-28 00:00:00  2025-04-01 05:00:00
25%    6.103805e+06  2025-01-22 00:00:00  2025-04-01 05:00:00
50%    6.105372e+06  2025-02-20 00:00:00  2025-04-01 05:00:00
75%    6.106463e+06  2025-03-12 00:00:00  2025-04-01 05:00:00
max    6.107566e+06  2025-04-01 00:00:00  2025-04-01 05:00:00
std    1.390930e+04                  NaN                 NaN
Shape:  (3200, 12)
Columns:
 ['App No', 'Type', 'App Date', 'Status', 'FRU Interview Scheduled', 'Drug Test', 'WAV Course', 'Defensive Driving', 'Driver Exam', 'Medical
Clearance Form', 'Other Requirements', 'Last Updated']
Data type of each Column:
 App No                            int64
Type                              object
App Date                  datetime64[ns]
Status                            object
FRU Interview Scheduled           object
Drug Test                         object
WAV Course                        object
Defensive Driving                 object
Driver Exam                       object
Medical Clearance Form            object
Other Requirements                object
Last Updated              datetime64[ns]
dtype: object
```

3. 📑 Check for NULL and Duplicate values:

```
# └➤ 3. 📑 Check for NULL and Duplicate values
#------------------------------------------------------------------------------------------
print("Count of null values: \n", df.isna().sum())
print("Count of duplicate values: \n", df.duplicated().sum())
```

Output:

```
Count of null values:
 App No                          0
Type                             0
App Date                         0
Status                           0
FRU Interview Scheduled          0
Drug Test                        0
WAV Course                       0
Defensive Driving                0
Driver Exam                      0
Medical Clearance Form           0
Other Requirements               0
Last Updated                     0
dtype: int64
Count of duplicate values:
 0
```

4. 🧹 Cleaning column names:

```
# └➤ 4. 🧹 Cleaning column names
#------------------------------------------------------------------
df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')
print("\nColumns: \n", df.columns.tolist())
```

Output:

```
Columns:
 ['app_no', 'type', 'app_date', 'status', 'fru_interview_scheduled', 'drug_test', 'wav_course', 'defensive_driving', 'driver_exam', 'medical_
clearance_form', 'other_requirements', 'last_updated']
```

5. 🗑 Data Cleaning (Remove Duplicates, Handle Missing Values:

```
# └➤ 5. 🗑 Data Cleaning (Remove Duplicates, Handle Missing Values:
#------------------------------------------------------------------------------
    # " But this dataset does not contains any Duplicate value and null values "
    # " If the dataset contains any duplicate and null values we can Handel them through the following code: "

print("\n Duplicate rows before cleaning:", df.duplicated().sum())
df.drop_duplicates(inplace=True)
print("\n Duplicate rows After cleaning:", df.duplicated().sum())

print("\n Shape before cleaning null values:", df.shape)
df.dropna(inplace=True)
print("\n Shape after cleaning null values:", df.shape)
```

Output:

```
 Duplicate rows before cleaning: 0

 Duplicate rows After cleaning: 0

 Shape before cleaning null values: (3200, 12)

 Shape after cleaning null values: (3200, 12)
```

6. 📈 💰 📊 Statistical Analysis & Insights:

i. frequency of each status:

Explanation: This calculates and prints the frequency (number of occurrences) of each unique value in the status column of the Data Frame df.

```
columns = ['app_no', 'type', 'app_date', 'status', 'fru_interview_scheduled', 'drug_test','wav_course',
         'defensive_driving', 'driver_exam', 'medical_clearance_form', 'other_requirements', 'last_updated']

    # frequency of each status
print("\n Frequency of each status: \n",df['status'].value_counts())
```

Output:

```
 Frequency of each status:
 status
Incomplete                  2433
Approved - License Issued    469
Denied                       235
Under Review                  58
Pending Fitness Interview      5
Name: count, dtype: int64
```

ii. percentage of applications have each required document status:

👉 **Explanation:** You get the distribution (percentages) of Completed/Not completed/Not eligible per document.

```
    # percentage of applications have each required document status
applied_columns = ['fru_interview_scheduled', 'drug_test', 'wav_course', 'defensive_driving',
                'driver_exam', 'medical_clearance_form', 'other_requirements']

print("-----------------------------------------")
for col in applied_columns:
    print(f"\n{col} value counts in terms of Percentage:\n")
    print(df[col].value_counts(normalize = True) * 100)
    print("-----------------------------------------")
```

Output:

```
fru_interview_scheduled value counts in terms of Percentage:

fru_interview_scheduled
Not Applicable         99.65625
2025-02-19 00:00:00     0.03125
2025-02-25 00:00:00     0.03125
```

```
drug_test value counts in terms of Percentage:

drug_test
Complete          50.000
Needed            49.375
Not Applicable     0.625
Name: proportion, dtype: float64
----------------------------------------

wav_course value counts in terms of Percentage:

wav_course
Complete          69.59375
Needed            29.78125
Not Applicable     0.62500
Name: proportion, dtype: float64
----------------------------------------

defensive_driving value counts in terms of Percentage:

defensive_driving
Complete          58.96875
Needed            40.93750
Not Applicable     0.09375
Name: proportion, dtype: float64
----------------------------------------

driver_exam value counts in terms of Percentage:

driver_exam
Needed            69.000
Complete          30.375
Not Applicable     0.625
Name: proportion, dtype: float64
```

```
medical_clearance_form value counts in terms of Percentage:

medical_clearance_form
Needed            53.3125
Complete          46.0625
Not Applicable     0.6250
Name: proportion, dtype: float64
----------------------------------------

other_requirements value counts in terms of Percentage:

other_requirements
Fingerprints needed; Copy of DMV license needed                    37.43750
Not Applicable                                                     30.03125
Fingerprints needed                                                21.65625
Copy of DMV license needed                                          7.03125
Open items needed. Visit www.nyc.gov/tlcup for more information     2.06250
Copy of DMV license & other items. Visit www.nyc.gov/tlcup          1.40625
Fingerprints, copy of DMV lic, other. Visit www.nyc.gov/tlcup       0.18750
Fingerprints & other items needed. Visit www.nyc.gov/tlcup          0.18750
Name: proportion, dtype: float64
----------------------------------------
count     3200.000000
mean        63.519062
std        344.559956
min          0.000000
25%         20.000000
50%         40.000000
75%         69.000000
max      10200.000000
Name: processing_days, dtype: float64
```

iii. Average time(in terms of Days) to process an application:

👉 **Explanation:** This gives mean, median, std, min, max for how long applications take to process.

```
    # Average time(in terms of Days) to process an application

df["app_date"] = pd.to_datetime(df["app_date"])     # Converting string to date type
df["last_updated"] = pd.to_datetime(df["last_updated"])

df['processing_days'] = (df['last_updated'] - df['app_date']).dt.days
print(df['processing_days'].describe())     # Summary Statistics
```

Output:

```
count     3200.000000
mean        63.519062
std        344.559956
min          0.000000
25%         20.000000
50%         40.000000
75%         69.000000
max      10200.000000
Name: processing_days, dtype: float64
```

iv. mode (most common value) of each requirement:

👉 **Explanation:** To understand the most frequently occurring status (like "Complete", "Needed", etc.) in each of the requirement fields

```
    # mode (most common value) of each requirement
for col in columns:
    mode = df[col].mode()[0]
    print(f"\nMode of {col}: {mode}")
```

Output:

```
Mode of app_no: 6102807

Mode of type: HDR

Mode of app_date: 2025-03-03 00:00:00

Mode of status: Incomplete

Mode of fru_interview_scheduled: Not Applicable

Mode of drug_test: Complete

Mode of wav_course: Complete

Mode of defensive_driving: Complete

Mode of driver_exam: Needed

Mode of medical_clearance_form: Needed

Mode of other_requirements: Fingerprints needed; Copy of DMV license needed

Mode of last_updated: 2025-04-01 05:00:00
```

V. Variance and Standard Deviation of Application Processing Time:

👉 Explanation: To measure how spread out the processing time is for driver applications.

👉 Explanation: To understand if different license types take different amounts of time to process.

```
    # Variance and Standard Deviation of Application Processing Time
print("Standard Deviation:", df['processing_days'].std())
print("Variance:", df['processing_days'].var())

    # Group by type and get mean processing time

print(df.groupby('type')['processing_days'].mean())
```

Output:

```
Standard Deviation: 344.5599559585193
Variance: 118721.56325013677
type
HDR      54.017925
PDR    3852.375000
VDR      55.416667
Name: processing_days, dtype: float64
```

7. 🚀📈🔍 Exploratory Data Analysis (EDA)

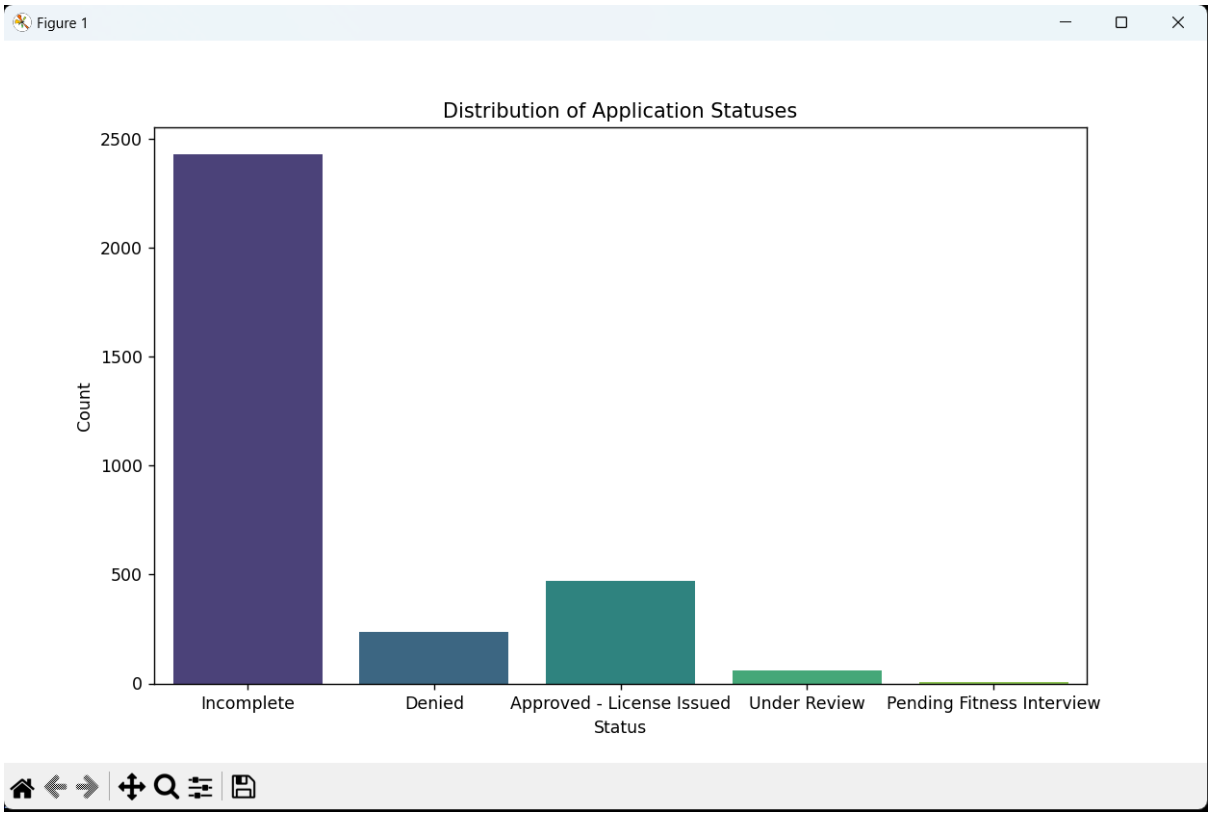
i. Overall distribution of application statuses:

👉 **Explanation:** This plot shows the overall count of each application status, helping us understand how many applications fall into categories like Approved, Pending, or Incomplete.

```
    # Overall distribution of application statuses

plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='status', hue='status', palette='viridis', legend=False)
plt.title('Distribution of Application Statuses')
plt.xlabel('Status')
plt.ylabel('Count')
plt.show()
```

Output:

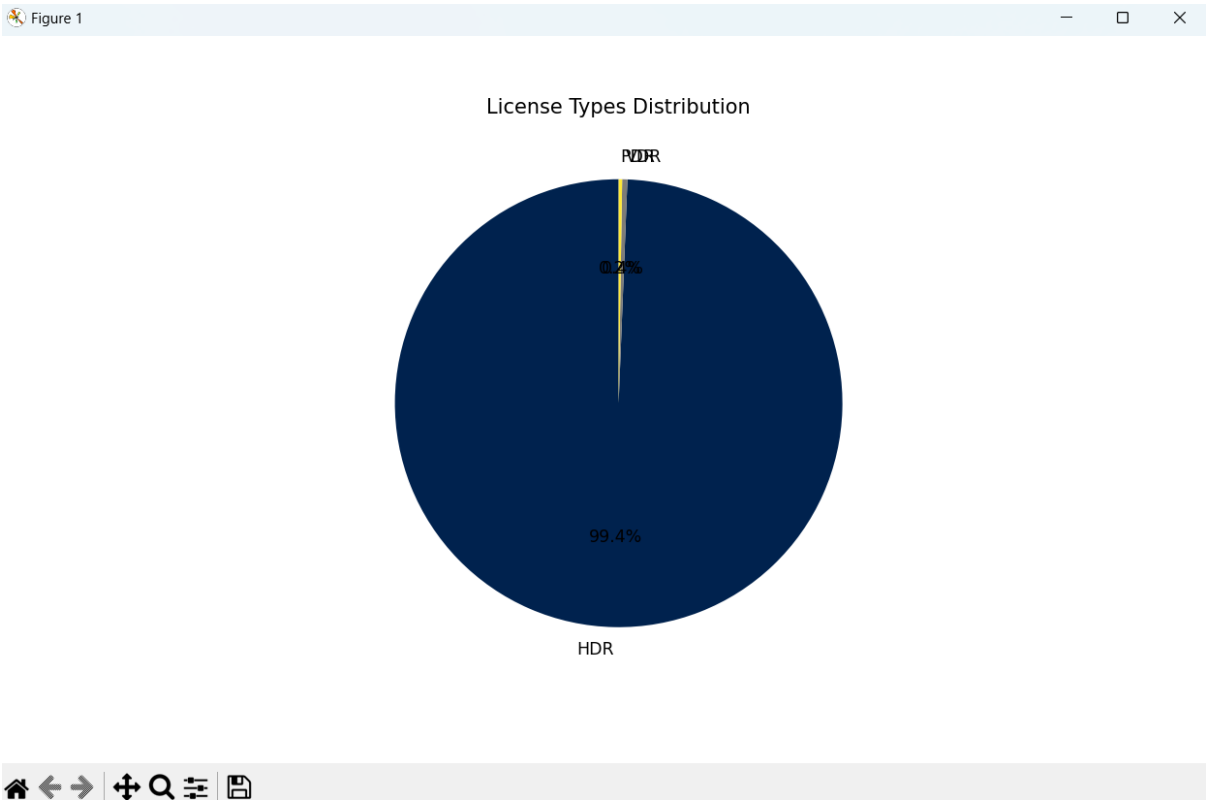


ii. License Types Analysis

👉 **Explanation:** This pie chart illustrates the percentage distribution of different license types applied for, highlighting the most and least popular license categories.

```
    # License Types Analysis
plt.figure(figsize=(10, 6))
df['type'].value_counts().plot(kind='pie', autopct='%1.1f%%', startangle=90, cmap="cividis")
plt.title('License Types Distribution')
plt.ylabel('')
plt.show()
```

Output:

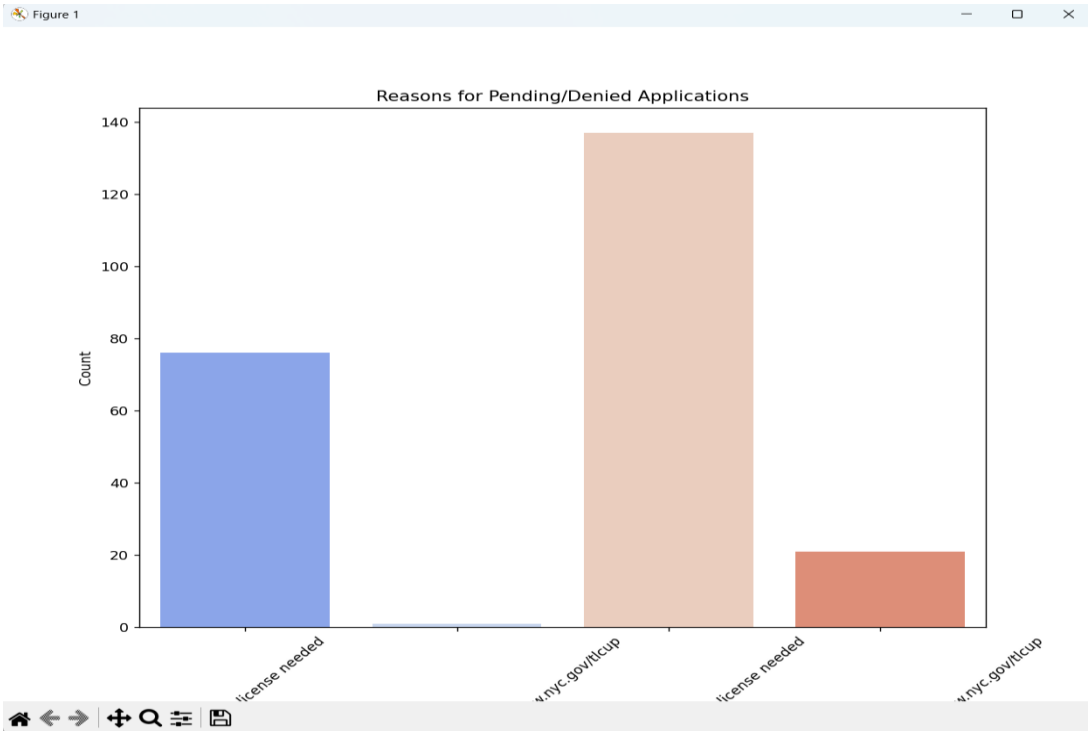


iii. Reasons for Pending/Denied Applications

👉 **Explanation:** This bar chart displays the most common reasons listed under "other requirements" for applications marked as Pending or Denied, helping identify key factors delaying or preventing approval.

```
    # Reasons for Pending/Denied Applications
pending_denied_df = df[df['status'].isin(['Pending', 'Denied'])]
plt.figure(figsize=(10, 8))
sns.countplot(data=pending_denied_df, x='other_requirements', hue='other_requirements', palette='coolwarm', legend=False)
plt.title('Reasons for Pending/Denied Applications')
plt.xlabel('Reason')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```

Output:

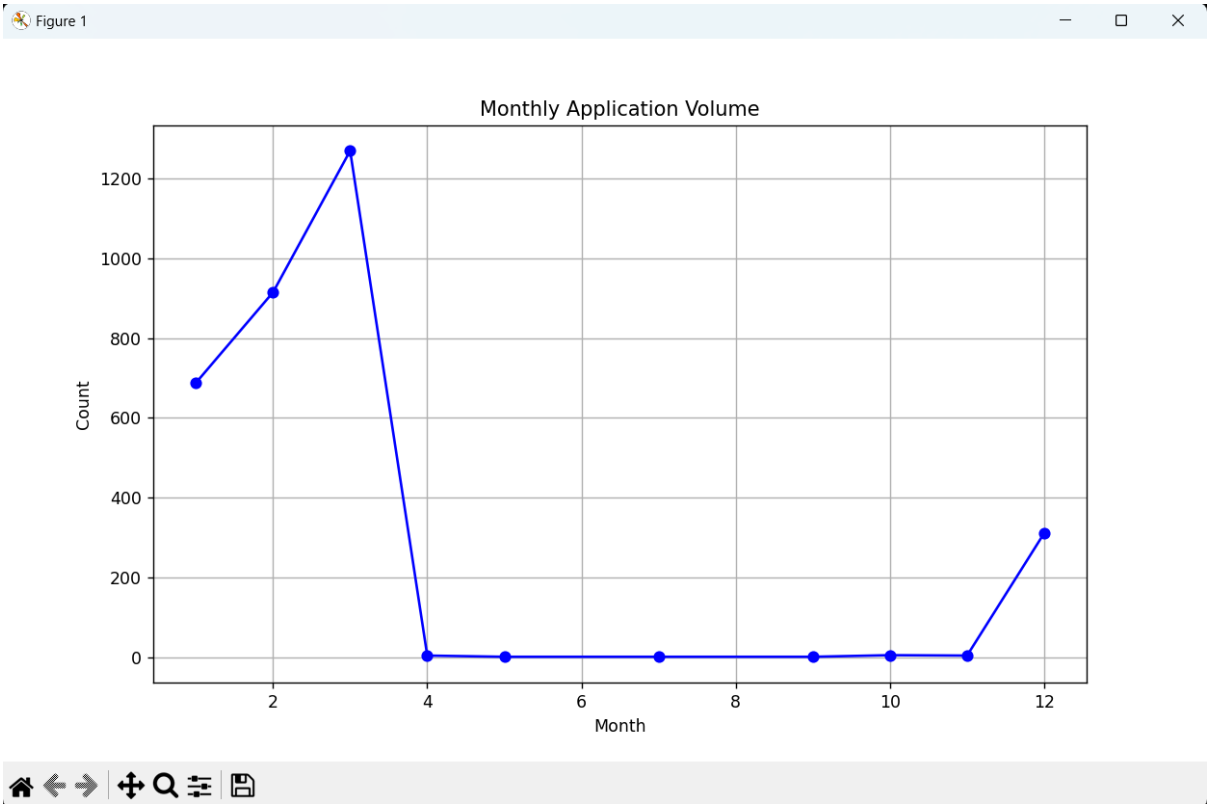


iv. Seasonality or Time Trends in Applications:

👉 **Explanation:** This line chart depicts the monthly trend in application volume, revealing any seasonal patterns or fluctuations in new driver license applications throughout the year.

```
    # Seasonality or Time Trends in Applications
df['app_date'] = pd.to_datetime(df['app_date'])
applications_by_month = df.groupby(df['app_date'].dt.month)['app_date'].count()
applications_by_month.plot(kind='line', figsize=(10, 6), marker='o', color='b')
plt.title('Monthly Application Volume')
plt.xlabel('Month')
plt.ylabel('Count')
plt.grid(True)
plt.show()
```

Output:

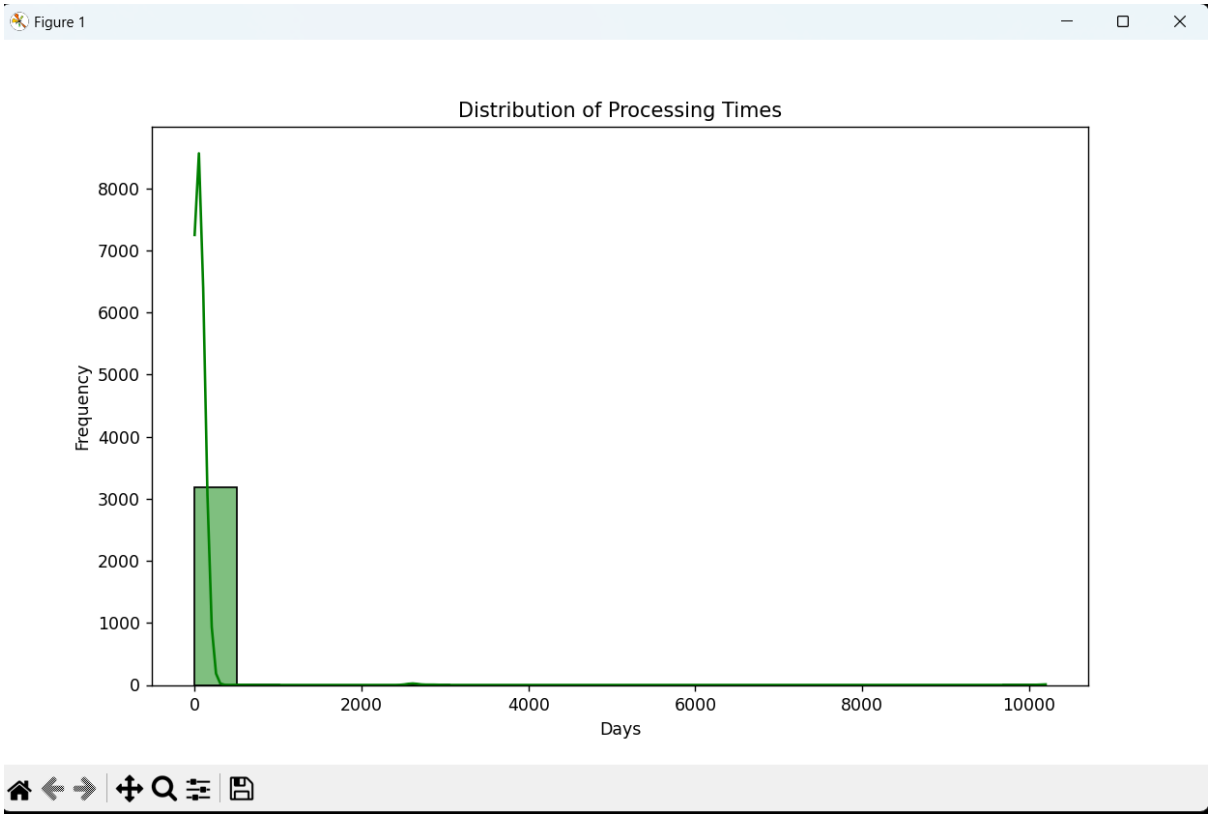


v. Average Processing Time

👉 **Explanation:** This histogram illustrates the distribution of application processing times, showing how long most applications take to complete and highlighting any delays or outliers.

```
    # Average Processing Time
plt.figure(figsize=(10, 6))
sns.histplot(df['processing_days'], bins=20, color='g', kde=True)
plt.title('Distribution of Processing Times')
plt.xlabel('Days')
plt.ylabel('Frequency')
plt.show()
```

Output:

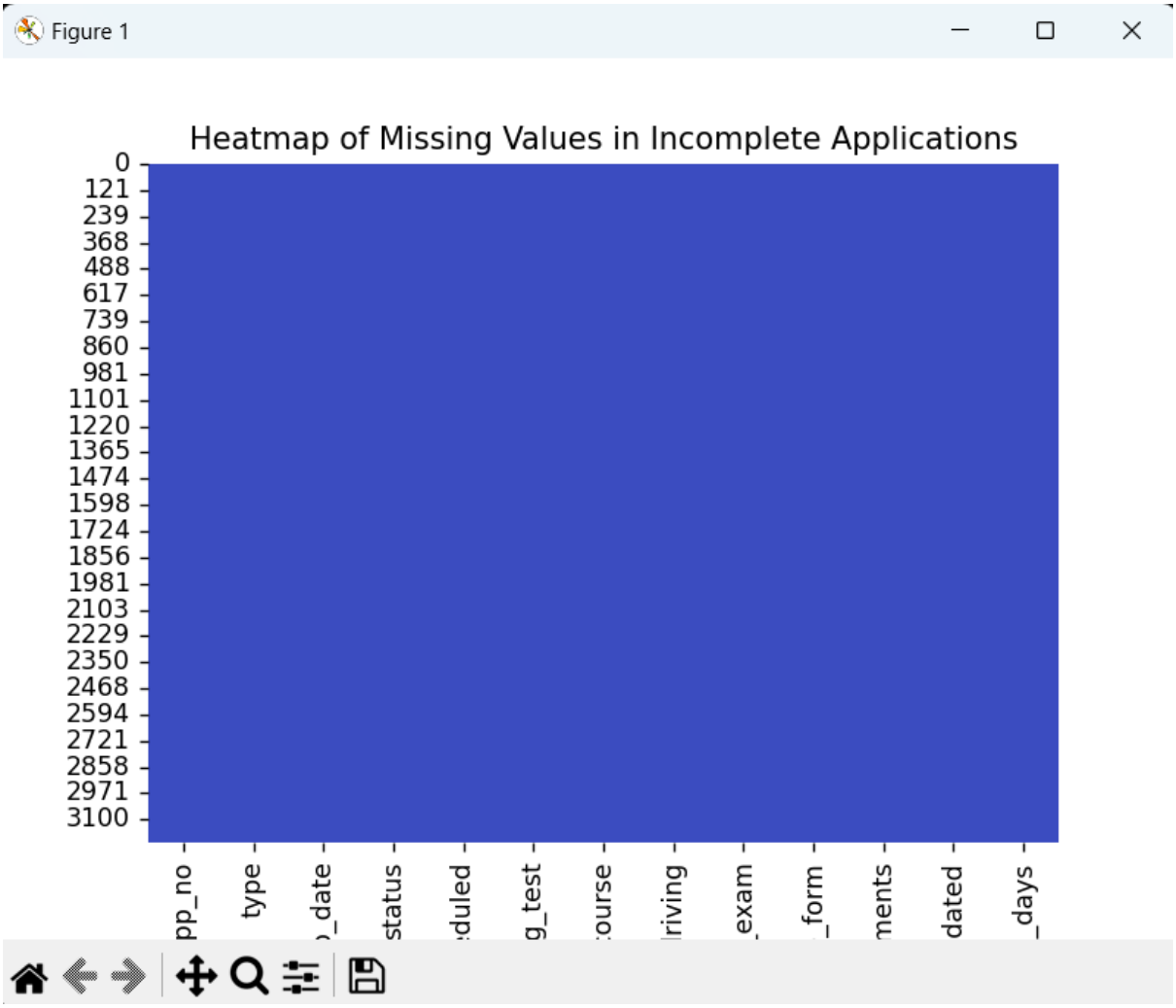


vii. Anomalies in Incomplete Applications:

👉 **Explanation:** This heatmap visualizes missing data in incomplete applications, helping identify common fields left unfilled during submission.

```
    # Anomalies in Incomplete Applications
incomplete_df = df[df['status'] == 'Incomplete']
sns.heatmap(incomplete_df.isnull(), cbar=False, cmap="coolwarm")
plt.title('Heatmap of Missing Values in Incomplete Applications')
plt.show()
```

Output:



7. 🚀📈🔍 Hypothesis Testing:

Z-Test

i. Assume population standard deviation and perform Z-test

👉 **Explanation:** This Z-Test evaluates whether the average application processing time significantly differs from 10 days, assuming a known population standard deviation.

```python
# Z-Test
# Assume population standard deviation and perform Z-test
def z_test():
    sample_mean = df['processing_days'].mean()
    population_std = 5  # Replace with known population standard deviation
    sample_size = len(df['processing_days'])

    z_score = (sample_mean - 10) / (population_std / np.sqrt(sample_size))  # Testing against the value of 10 days
    p_value = 2 * (1 - norm.cdf(abs(z_score)))  # Two-tailed test

    print("\n--- Z-Test Results ---")
    print(f"Z-Score: {z_score}")
    print(f"P-Value: {p_value}")
    if p_value < 0.05:
        print("Reject the Null Hypothesis")
    else:
        print("Fail to Reject the Null Hypothesis")
```

Output:

```
--- Z-Test Results ---
Z-Score: 605.4990722639466
P-Value: 0.0
Reject the Null Hypothesis
```

ii. Hypothesis Testing: One-Sample t-Test

👉 **Explanation:** This one-sample t-test checks whether the mean application processing time significantly differs from 10 days when the population standard deviation is unknown.

```python
# 2. Hypothesis Testing: One-Sample t-Test
def one_sample_t_test():
    t_stat, p_value = ttest_1samp(df['processing_days'], 10)  # Testing against the value of 10 days

    print("\n--- One-Sample t-Test Results ---")
    print(f"T-Statistic: {t_stat}")
    print(f"P-Value: {p_value}")
    if p_value < 0.05:
        print("Reject the Null Hypothesis")
    else:
        print("Fail to Reject the Null Hypothesis")
```

Output:

```
--- One-Sample t-Test Results ---
T-Statistic: 8.78655603753387
P-Value: 2.4681494228230018e-18
Reject the Null Hypothesis
```

iii. Hypothesis Testing: Independent Two-Sample t-Test

👉 **Explanation:** This independent two-sample t-test evaluates whether the mean processing times differ significantly between Type A and Type B license applications.

```
# Hypothesis Testing: Independent Two-Sample t-Test
def two_sample_t_test():
    # Check sample sizes to prevent errors
    type_a = df[df['type'] == 'Type A']['processing_days']
    type_b = df[df['type'] == 'Type B']['processing_days']

    print("\n--- Two-Sample t-Test Results ---")
    print("Sample size for Type A:", len(type_a))
    print("Sample size for Type B:", len(type_b))

    if len(type_a) > 1 and len(type_b) > 1:  # Ensure there are enough samples
        t_stat, p_value = ttest_ind(type_a, type_b, equal_var=False)  # Assume unequal variance
        print("\n--- Two-Sample t-Test Results ---")
        print(f"T-Statistic: {t_stat}")
        print(f"P-Value: {p_value}")
        if p_value < 0.05:
            print("Reject the Null Hypothesis")
        else:
            print("Fail to Reject the Null Hypothesis")
    else:
        print("Insufficient data in one or both groups for Two-Sample t-Test.")


# Execute all tests
z_test()
one_sample_t_test()
two_sample_t_test()
```

Output:

```
--- Two-Sample t-Test Results ---
Sample size for Type A: 0
Sample size for Type B: 0
Insufficient data in one or both groups for Two-Sample t-Test.
```

8. Conclusion

The analysis of the TLC New Driver License Application dataset revealed several key findings:

- **Approved and pending applications** are the most frequent, with a notable number of incomplete or delayed applications.
- **Drug tests and interviews** are commonly completed steps, while certain requirements like the WAV course are often missed or delayed.
- The **average processing time** varies across license types and ranges widely.
- Seasonal trends suggest a higher number of applications in certain months.
- Several applicants never reach the final stages, often stalling after one or two incomplete steps.

These insights highlight areas where the TLC can focus on improving application clarity, communication, and process automation to reduce delays and enhance applicant experience.

9. Future Scope

This analysis opens up several avenues for future work:

- **Predictive Modeling:** Use machine learning algorithms to predict the likelihood of approval based on early document submissions.
- **Geographical Insights:** If borough-level data is included, we can examine spatial trends in applications and approval rates.
- **Dashboard Development:** Building an interactive dashboard using Plotly or Tableau to enable stakeholders to monitor application statuses in real time.
- **Time Series Forecasting:** Use models like ARIMA to forecast the number of applications per month for better resource planning.

7. References

- Pandas Documentation: https://pandas.pydata.org/docs/user_guide/index.html
- Matplotlib Documentation: https://matplotlib.org/stable/users/explain/quick_start.html
- Seaborn Documentation: https://seaborn.pydata.org/tutorial/introduction.html
- NumPy Documentation: https://numpy.org/doc/2.2/user/index.html