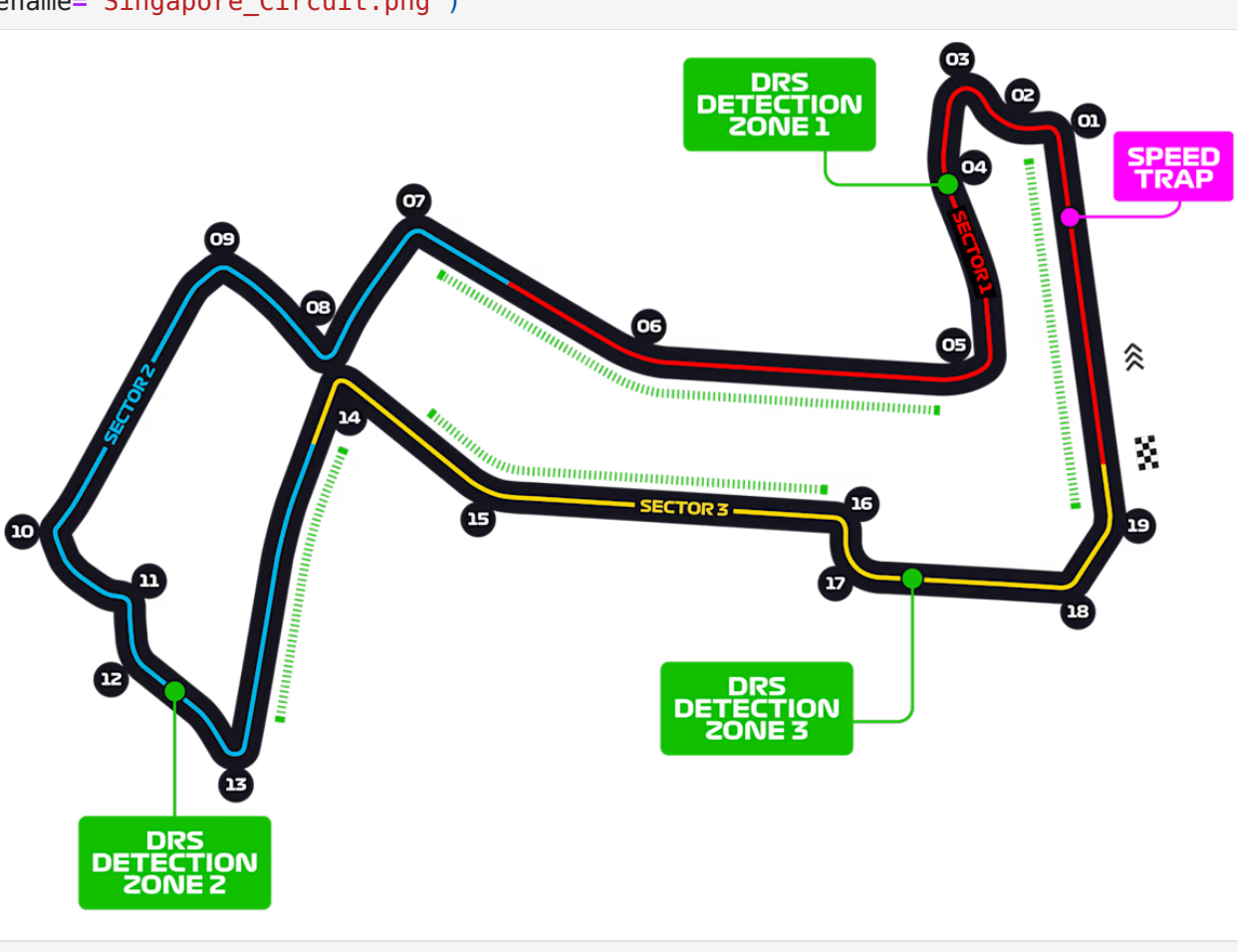
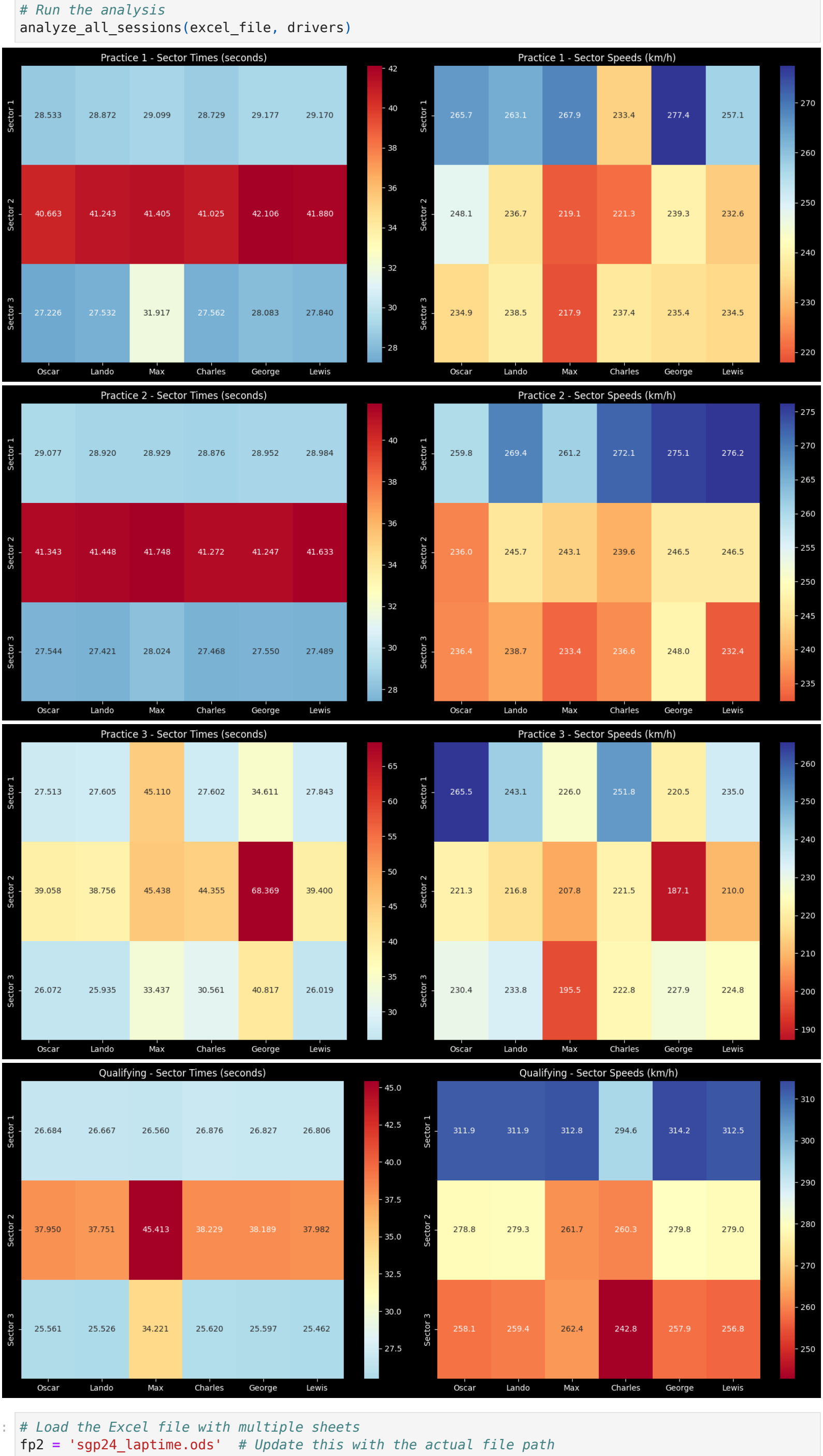
In [48]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import display
import seaborn as sns
```

In [49]:
```python
from IPython.display import Image
Image(filename='Singapore_Circuit.png')
```

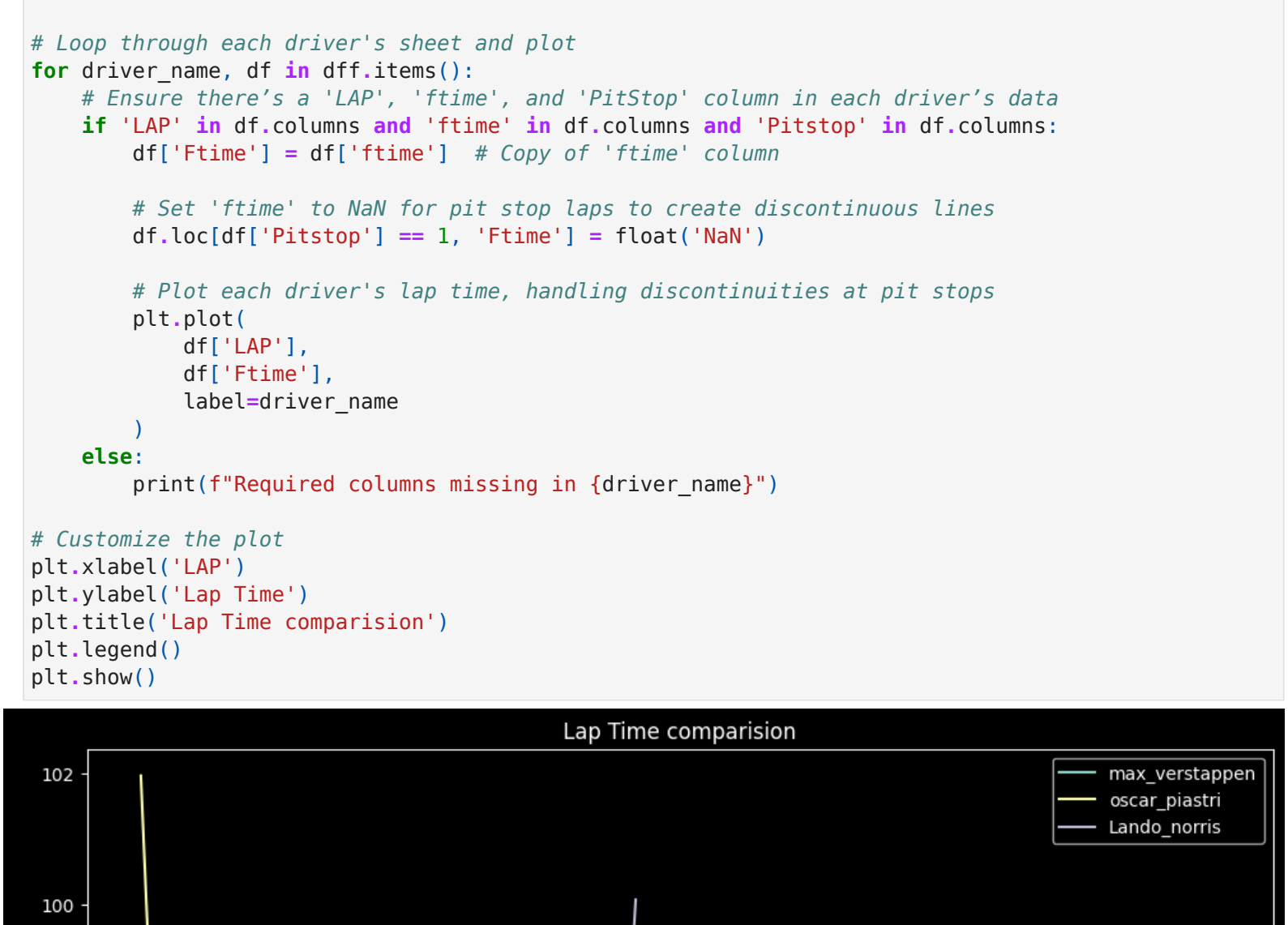
Out[49]:



In [50]:
```python
def clean_session_data(df):
    """
    Clean session data by removing pit stops and empty values
    """
    # Drop rows with any missing values
    df = df.dropna()
    """
    # Calculate z-scores for sector times
    for sector in ['TIME1', 'TIME2', 'TIME3']:
        z_scores = np.abs((df[sector] - df[sector].mean()) / df[sector].std())
        # Mark as pit stop if z-score > 3 (unusually high sector time)
        df = df[z_scores < 3]
    """
    return df

def combine_session_data(excel_file, drivers, session_type):
    """
    Combine data from individual driver sheets for a specific session
    """
    combined_data = []

    for driver in drivers:
        sheet_name = f"{driver.lower()}_{session_type.lower()}"
        try:
            # Read the specific sheet
            df = pd.read_excel(excel_file, sheet_name=sheet_name)
            # Add driver column
            df['DRIVER'] = driver
            # Clean the data
            df = clean_session_data(df)
            combined_data.append(df)
        except Exception as e:
            print(f"Error reading sheet {sheet_name}: {e}")

    # Combine all drivers' data
    if combined_data:
        return pd.concat(combined_data, ignore_index=True)
    return None

def create_sector_heatmaps(session_data, session_name):
    """
    Create heatmaps for sector times and speeds
    """
    # Calculate average sector times and speeds for each driver
    sector_times = pd.DataFrame()
    sector_speeds = pd.DataFrame()

    drivers = session_data['DRIVER'].unique()

    # Calculate averages for each sector
    for driver in drivers:
        driver_data = session_data[session_data['DRIVER'] == driver]

        # Sector times
        sector_times[driver] = [
            driver_data['TIME1'].median(),
            driver_data['TIME2'].median(),
            driver_data['TIME3'].median()
        ]

        # Sector speeds
        sector_speeds[driver] = [
            driver_data['SPEED1'].mean(),
            driver_data['SPEED2'].mean(),
            driver_data['SPEED3'].mean()
        ]

    sector_times.index = ['Sector 1', 'Sector 2', 'Sector 3']
    sector_speeds.index = ['Sector 1', 'Sector 2', 'Sector 3']

    # Create heatmaps
    plt.figure(figsize=(15, 6))

    # Sector Times Heatmap
    plt.subplot(1, 2, 1)
    sns.heatmap(sector_times, annot=True, fmt='.3f', cmap='RdYlBu_r',
                center=sector_times.mean().mean())
    plt.title(f'{session_name} - Sector Times (seconds)')

    # Sector Speeds Heatmap
    plt.subplot(1, 2, 2)
    sns.heatmap(sector_speeds, annot=True, fmt='.1f', cmap='RdYlBu',
                center=sector_speeds.mean().mean())
    plt.title(f'{session_name} - Sector Speeds (km/h)')

    plt.tight_layout()
    plt.show()

def analyze_all_sessions(excel_file, drivers):
    """
    Analyze all sessions for all drivers
    """
    sessions = ['P1', 'P2', 'P3', 'Q']

    for session in sessions:
        # Combine data from all drivers for this session
        combined_session_data = combine_session_data(excel_file, drivers, session)

        if combined_session_data is not None:
            # Create heatmaps for this session
            session_names = {
                'P1': 'Practice 1',
                'P2': 'Practice 2',
                'P3': 'Practice 3',
                'Q': 'Qualifying'
            }
            create_sector_heatmaps(combined_session_data, session_names[session])

# Example usage:
# Define your Excel file path
excel_file = 'sgp24_sector.ods'

# List of drivers (adjust according to your data)
drivers = ['Oscar', 'Lando', 'Max', 'Charles', 'George', 'Lewis']  # Add all your drivers

# Run the analysis
analyze_all_sessions(excel_file, drivers)
```



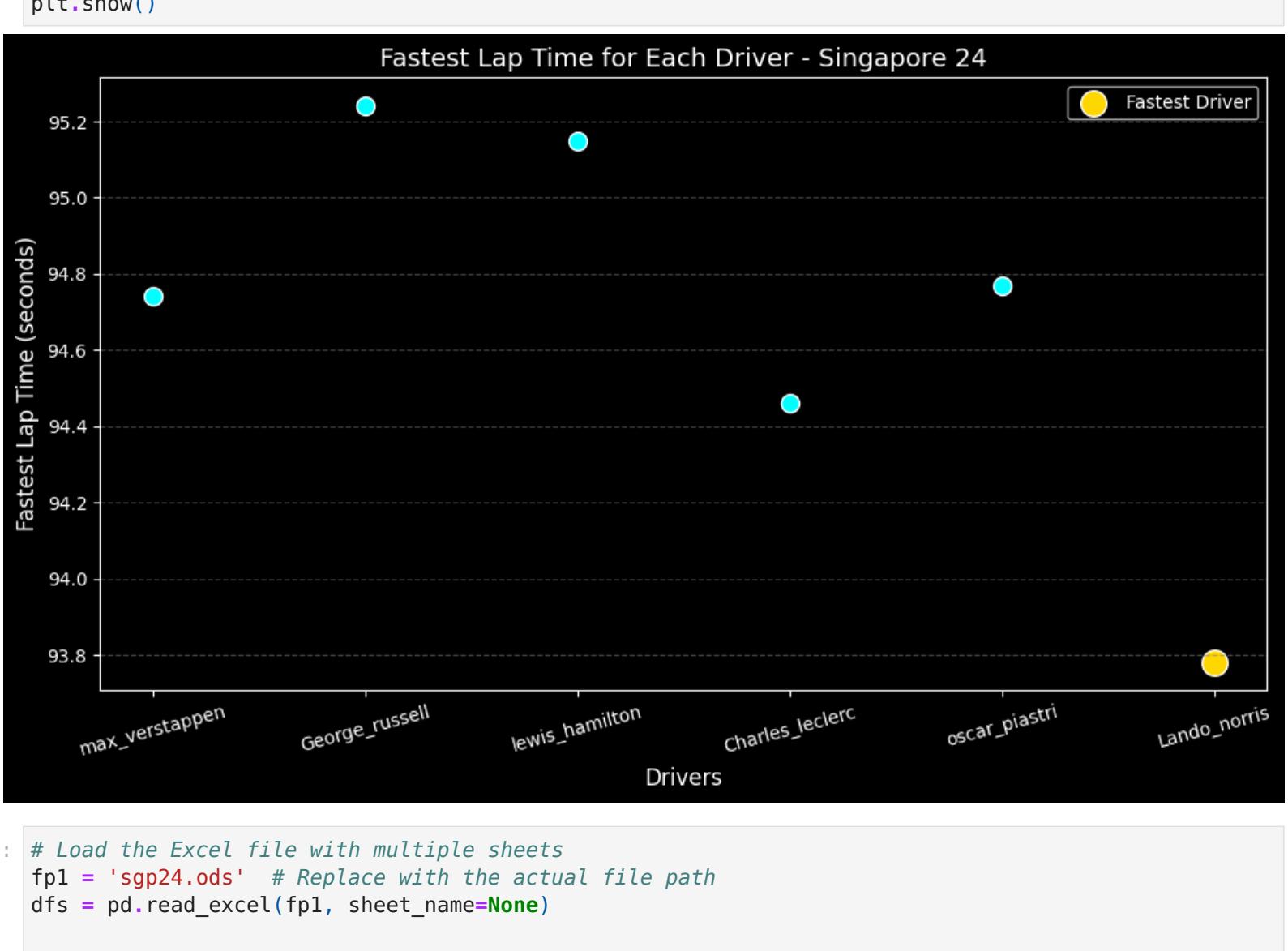
In [51]:
```python
# Load the Excel file with multiple sheets
fp2 = 'sgp24_laptime.ods'  # Update this with the actual file path
dff = pd.read_excel(fp2, sheet_name=None)  # Load all sheets

plt.figure(figsize=(12, 6))  # Set figure size

# Loop through each driver's sheet and plot
for driver_name, df in dff.items():
    # Ensure there's a 'LAP', 'ftime', and 'PitStop' column in each driver's data
    if 'LAP' in df.columns and 'ftime' in df.columns and 'Pitstop' in df.columns:
        df['Ftime'] = df['ftime']  # Copy of 'ftime' column

        # Set 'ftime' to NaN for pit stop laps to create discontinuous lines
        df.loc[df['Pitstop'] == 1, 'Ftime'] = float('NaN')

        # Plot each driver's lap time, handling discontinuities at pit stops
        plt.plot(
            df['LAP'],
            df['Ftime'],
            label=driver_name
        )
    else:
        print(f"Required columns missing in {driver_name}")

# Customize the plot
plt.xlabel('LAP')
plt.ylabel('Lap Time')
plt.title('Lap Time comparision')
plt.legend()
plt.show()
```



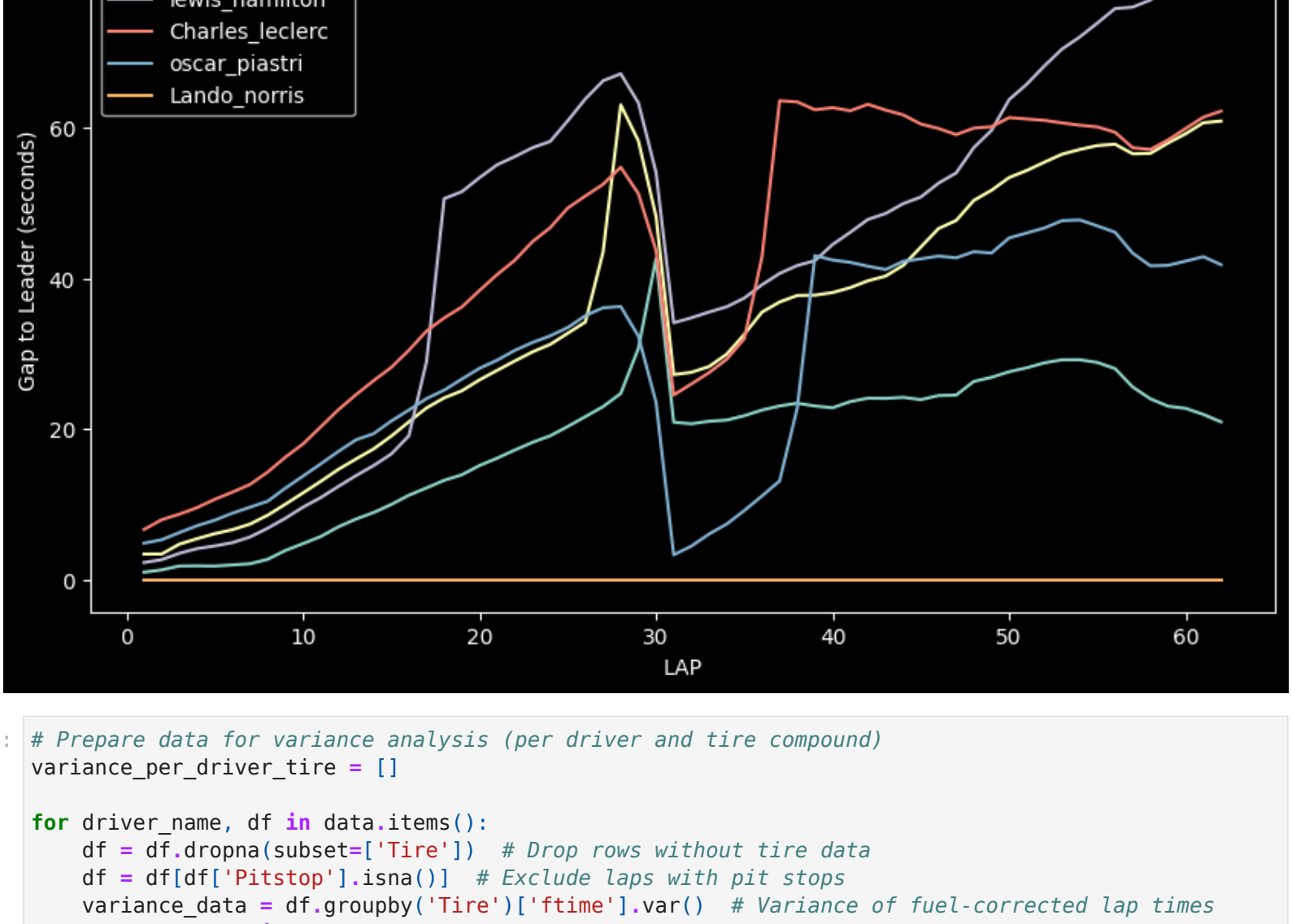
In [52]:
```python
file_path = 'sgp24.ods'
data = pd.read_excel(file_path, sheet_name=None)

# Collect the data for each driver's `ftime` from all sheets
driver_data = {sheet: data[sheet]['ftime'] for sheet in data.keys()}

# Plotting with dark theme
plt.style.use('dark_background')
plt.figure(figsize=(12, 6))
plt.boxplot(driver_data.values(), labels=driver_data.keys(), showmeans=True,
            boxprops=dict(color='white'),
            whiskerprops=dict(color='white'),
            capprops=dict(color='white'),
            medianprops=dict(color='yellow'),
            meanprops=dict(marker='o', markerfacecolor='cyan', markeredgecolor='cyan'))

plt.title('Fuel-Corrected Lap Times (ftime) for Each Driver - Singapore 24', fontsize=14, color='wh
plt.ylabel('Fuel-Corrected Lap Time (seconds)', fontsize=12, color='white')
plt.xlabel('Drivers', fontsize=12, color='white')
plt.grid(axis='y', linestyle='--', alpha=0.5, color='gray')
plt.xticks(rotation=15, color='white')
plt.yticks(color='white')
plt.tight_layout()
plt.show()
```

```
/tmp/ipykernel_7306/3699657016.py:10: MatplotlibDeprecationWarning: The 'labels' parameter of boxpl
ot() has been renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be dropped
in 3.11.
  plt.boxplot(driver_data.values(), labels=driver_data.keys(), showmeans=True,
```

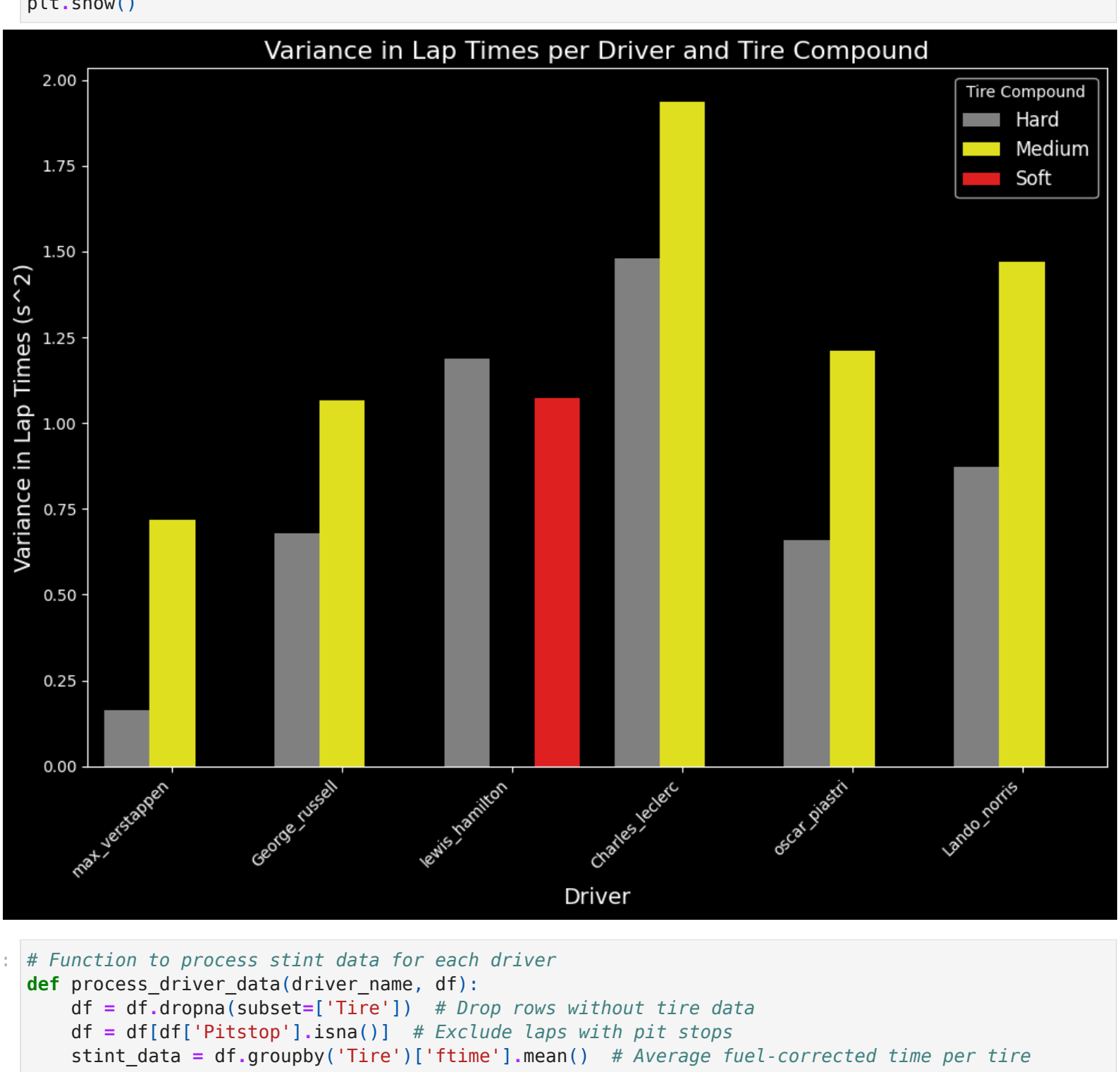
In [53]:
```python
file_path = 'sgp24.ods'
data = pd.read_excel(file_path, sheet_name=None)

# Extract the fastest lap time (minimum `ftime`) for each driver
fastest_lap_times = {driver: sheet['ftime'].min() for driver, sheet in data.items()}

# Scatter plot
plt.style.use('dark_background')  # Dark theme
plt.figure(figsize=(10, 6))
plt.scatter(fastest_lap_times.keys(), fastest_lap_times.values(), color='cyan', edgecolor='white',

# Plot annotations
plt.title('Fastest Lap Time for Each Driver - Singapore 24', fontsize=14, color='white')
plt.ylabel('Fastest Lap Time (seconds)', fontsize=12, color='white')
plt.xlabel('Drivers', fontsize=12, color='white')
plt.xticks(rotation=15, color='white')
plt.yticks(color='white')
plt.grid(axis='y', linestyle='--', alpha=0.5, color='gray')

# Highlighting the fastest driver overall
fastest_driver = min(fastest_lap_times, key=fastest_lap_times.get)
fastest_time = fastest_lap_times[fastest_driver]
plt.scatter([fastest_driver], [fastest_time], color='gold', edgecolor='white', s=200, label='Fastes
plt.legend(facecolor='black', fontsize=10)

plt.tight_layout()
plt.show()
```



In [54]:
```python
# Load the Excel file with multiple sheets
fp1 = 'sgp24.ods'  # Replace with the actual file path
dfs = pd.read_excel(fp1, sheet_name=None)

# Assuming each sheet corresponds to a driver
driver_names = list(dfs.keys())  # Get the driver names from the sheet names

# Initialize an empty dictionary to store DataFrames for each driver
data = {}

# Loop through each sheet (driver's data) and calculate the gap to leader
for driver in driver_names:
    data[driver] = dfs[driver]
    # Ensure the columns are appropriately named in your sheets, e.g., 'Lap' and 'Fuel Corrected La
    data[driver]['Ctime'] = data[driver]['ftime'].cumsum()

# Now calculate the gap to the leader at each lap (assuming the first driver is the leader)
leader = driver_names[5]  # Assuming the first driver in the sheet list is the leader
for driver in driver_names:
    data[driver]['GapL'] = data[driver]['Ctime'] - data[leader]['Ctime']

# Plot the data
plt.figure(figsize=(10, 6))

for driver in driver_names:
    plt.plot(data[driver]['LAP'], data[driver]['GapL'], label=driver)


# Add labels, title, and legend
plt.xlabel('LAP')
plt.ylabel('Gap to Leader (seconds)')
plt.title('Race Gap Chart w.r.t Lando')
plt.legend()
# Show the plot
plt.show()
```

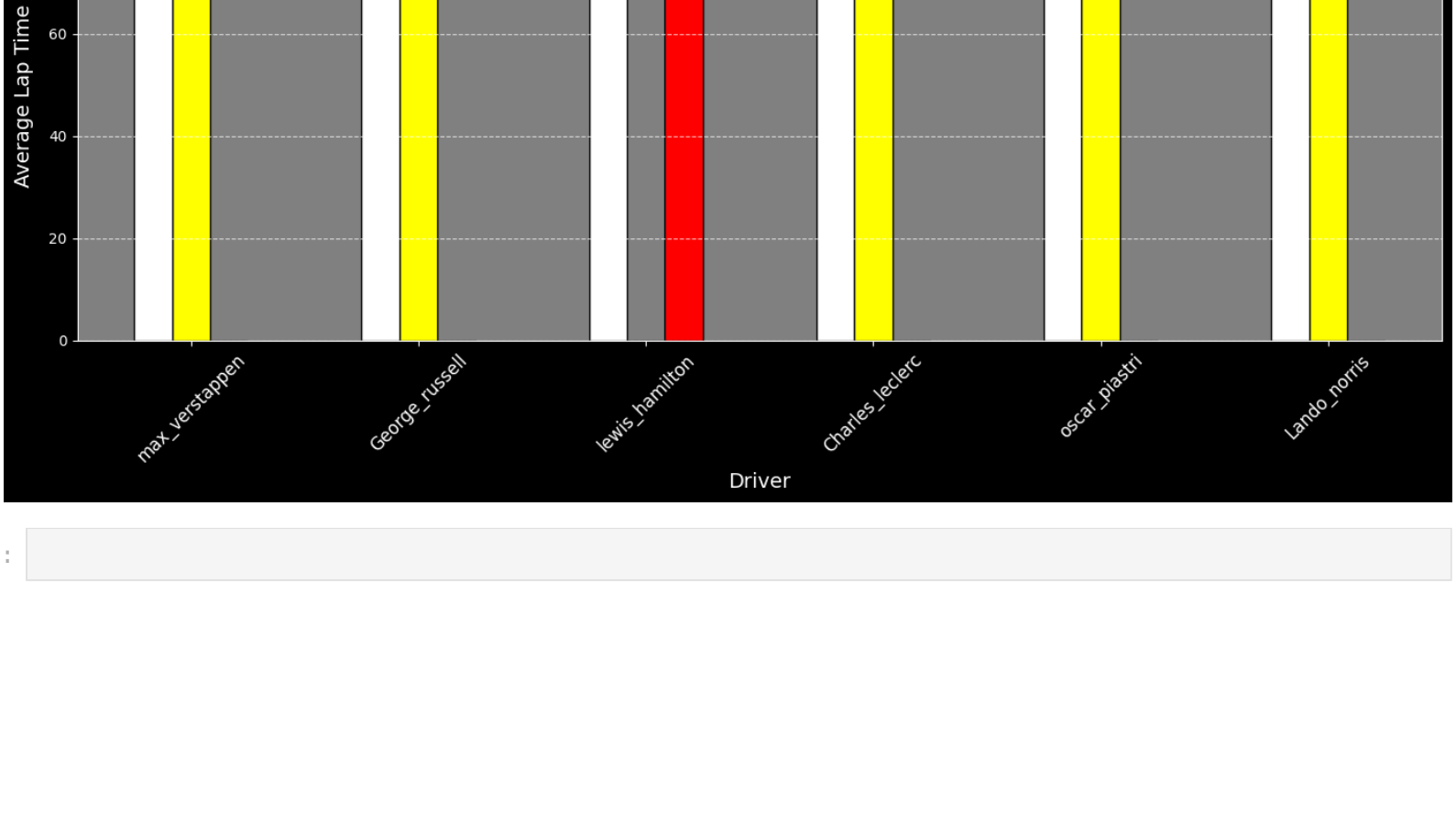


In [55]:
```python
# Prepare data for variance analysis (per driver and tire compound)
variance_per_driver_tire = []

for driver_name, df in data.items():
    df = df.dropna(subset=['Tire'])  # Drop rows without tire data
    df = df[df['Pitstop'].isna()]  # Exclude laps with pit stops
    variance_data = df.groupby('Tire')['ftime'].var()  # Variance of fuel-corrected lap times
    for tire, var in variance_data.items():
        variance_per_driver_tire.append({'Driver': driver_name, 'Tire': tire, 'Variance': var})

# Convert to DataFrame for visualization
variance_df = pd.DataFrame(variance_per_driver_tire)

# Create a grouped bar chart for variance
plt.figure(figsize=(10, 8))
sns.barplot(data=variance_df, x='Driver', y='Variance', hue='Tire', palette=['grey', 'yellow', 'rec
plt.title('Variance in Lap Times per Driver and Tire Compound', fontsize=16)
plt.xlabel('Driver', fontsize=14)
plt.ylabel('Variance in Lap Times (s^2)', fontsize=14)
plt.xticks(rotation=45, ha='right')
plt.legend(title='Tire Compound', fontsize=12)
plt.tight_layout()
plt.show()
```



In [56]:
```python
# Function to process stint data for each driver
def process_driver_data(driver_name, df):
    df = df.dropna(subset=['Tire'])  # Drop rows without tire data
    df = df[df['Pitstop'].isna()]  # Exclude laps with pit stops
    stint_data = df.groupby('Tire')['ftime'].mean()  # Average fuel-corrected time per tire
    return stint_data.rename(driver_name)

# Initialize a DataFrame to store stint analysis results
stint_analysis = pd.DataFrame()

# Process each sheet (driver data)
for driver_name, df in data.items():
    stint_data = process_driver_data(driver_name, df)
    stint_analysis = pd.concat([stint_analysis, stint_data], axis=1)

# Transpose for better readability
stint_analysis = stint_analysis.T

# Plotting the stint analysis
plt.figure(figsize=(10, 6))
stint_analysis.plot(kind='bar', figsize=(14, 7), color=['white', 'yellow', 'red'], edgecolor='black
plt.gca().set_facecolor('grey')
plt.title('Stint Analysis - Fuel Corrected - Singapore GP 2024', fontsize=16)
plt.xlabel('Driver', fontsize=14)
plt.ylabel('Average Lap Time (s)', fontsize=14)
plt.legend(title='Tire Compound', fontsize=12)
plt.xticks(rotation=45, fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



In [ ]: