Pycaret Assignment

- ## Association Rules Mining

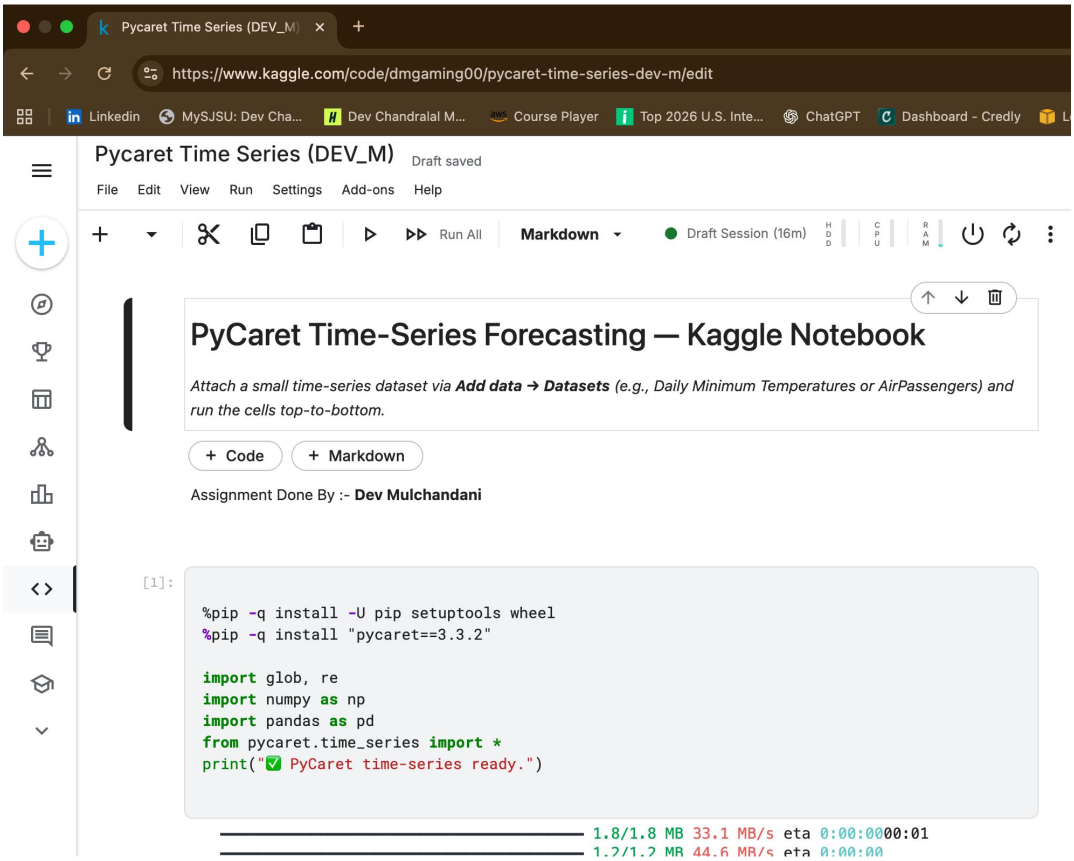
- ## Kaggle Notebook :- Link

- ## Submitted By :- Dev Mulchandani

- ## Overview :-

In my time-series forecasting notebook, I used PyCaret's time-series module to predict future values based on historical data. After loading and cleaning the dataset, I identified the date and target columns, handled missing values, and standardized the data to ensure a consistent time frequency. Using the setup() function, I initialized PyCaret's forecasting environment and compared multiple models with compare_models() to find the best-performing one. Then, I visualized the forecast results and saved both the trained model and the predicted future values. This notebook demonstrated a complete end-to-end time-series workflow — from data preparation to model selection, forecasting, and saving outputs — all done efficiently using PyCaret.

❖ Screenshots :-



[4]:

```python
import pandas as pd

def robust_read_csv(path):
    # Try fast path
    try:
        return pd.read_csv(path)
    except Exception as e1:
        print("Retry with engine='python' & sep=None (infer delimiter) ...", e1)
    # Infer delimiter, tolerate bad lines
    try:
        df = pd.read_csv(
            path,
            sep=None,              # let pandas sniff delimiter
            engine="python",       # more flexible parser
            on_bad_lines="skip"    # drop malformed rows
        )
        if df.columns.str.contains(r"Unnamed", regex=True).any():
            df = df.loc[:, ~df.columns.str.contains(r"^Unnamed")]
        return df
    except Exception as e2:
        print("Retry with common delimiters...", e2)

    # Try common delimiters explicitly
    for sep in [",", ";", "\t", "|"]:
        try:
            df = pd.read_csv(path, sep=sep, engine="python", on_bad_lines="skip")
            if df.columns.str.contains(r"Unnamed", regex=True).any():
                df = df.loc[:, ~df.columns.str.contains(r"^Unnamed")]
            print(f"Loaded with sep='{sep}'")
            return df
        except Exception:
            pass

    # Last resort: be ultra permissive on encoding/quoting
    df = pd.read_csv(
        path,
        sep=None,
        engine="python",
        on_bad_lines="skip",
```

```python
        engine="python",
        on_bad_lines="skip",
        encoding="latin1",     # tolerate odd characters/BOM
        quotechar='"'
    )
    return df

data = robust_read_csv(DATA_PATH)
print("Shape:", data.shape)
display(data.head())
```

```
Retry with engine='python' & sep=None (infer delimiter) ... Error tokenizing data. C error: Expected 2 fields in line 3653, saw 3

Shape: (3650, 2)
```

	Date	Daily minimum temperatures in Melbourne, Australia, 1981-1990
0	1981-01-01	20.7
1	1981-01-02	17.9
2	1981-01-03	18.8
3	1981-01-04	14.6
4	1981-01-05	15.8

[5]:

```python
# 👉 If you know the columns, set them here (else leave as None to auto-detect)
DATE_COL_GUESS   = None  # e.g., 'Date' or 'Month'
TARGET_COL_GUESS = None  # e.g., 'Temp' or 'Passengers'

def guess_date_col(df):
    candidates = [c for c in df.columns if re.search(r"(date|month|time|year)", c, re.I)]
    if candidates:
        return candidates[0]
    obj = [c for c in df.columns if df[c].dtype == 'object']
    return obj[0] if obj else df.columns[0]

def guess_target_col(df, date_col):
    num = df.select_dtypes(include='number').columns.tolist()
    if date_col in num:
        num.remove(date_col)
    if num:
        return num[0]
    for c in df.columns:
        if c == date_col:
            continue
        cleaned = pd.to_numeric(df[c].astype(str).str.replace(r"[^\d\.\-]", "", regex=True), errors="coerce")
        if cleaned.notna().mean() > 0.5:
            df[c] = cleaned
            return c
    return None

DATE_COL   = DATE_COL_GUESS   or guess_date_col(data)
TARGET_COL = TARGET_COL_GUESS or guess_target_col(data, DATE_COL)

print(f"📌 Selected DATE_COL = {DATE_COL}")
print(f"📌 Selected TARGET_COL = {TARGET_COL}")

if TARGET_COL is None:
    raise SystemExit("Could not find a numeric target. Please set TARGET_COL_GUESS to a numeric column name.")
```

```
📌 Selected DATE_COL = Date
📌 Selected TARGET_COL = Daily minimum temperatures in Melbourne, Australia, 1981-1990
```

[6]:

```python
# Parse dates
dt = pd.to_datetime(data[DATE_COL], errors='coerce', infer_datetime_format=True)
if dt.isna().all():
    for fmt in ("%Y-%m-%d", "%d-%m-%Y", "%m/%d/%Y", "%d/%m/%Y", "%Y/%m/%d"):
        try_dt = pd.to_datetime(data[DATE_COL], format=fmt, errors='coerce')
        if try_dt.notna().sum() > dt.notna().sum():
            dt = try_dt
            print(f"✅ Used explicit date format: {fmt}")
            break

df = data.copy()
df['_dt'] = dt
df = df.dropna(subset=['_dt']).sort_values('_dt')

# Clean target → numeric
if df[TARGET_COL].dtype == 'object':
    df[TARGET_COL] = pd.to_numeric(df[TARGET_COL].astype(str)
                                   .str.replace(r"[^\d\.\-]", "", regex=True),
                                   errors='coerce')
df[TARGET_COL] = pd.to_numeric(df[TARGET_COL], errors='coerce')
dfv = df[['_dt', TARGET_COL]].dropna()

if dfv.empty:
    raise SystemExit("No valid (date, value) rows after cleaning. Check DATE/TARGET column names.")

# Aggregate to daily (helps with multiple rows per day)
y = (dfv.groupby(dfv['_dt'].dt.normalize())[TARGET_COL]
        .mean()
        .sort_index())

print("Length after daily aggregation:", len(y))
display(y.head(10))
```

```
Length after daily aggregation: 3650
_dt
1981-01-01    20.7
1981-01-02    17.9
1981-01-03    18.8
1981-01-04    14.6
1981-01-05    15.8
1981-01-06    15.8
1981-01-07    15.8
1981-01-08    17.4
1981-01-09    21.8
1981-01-10    20.0
Name: Daily minimum temperatures in Melbourne, Australia, 1981-1990, dtype: float64
```

+ Code + Markdown

[9]:

```python
# ----- Force a proper frequency for PyCaret -----
# Try to infer frequency
freq = None
if len(y) >= 3:
    try:
        freq = pd.infer_freq(y.index)
    except Exception:
        pass

# If we still don't have one, ask the user (with a safe default)
if not isinstance(freq, (str, pd.tseries.offsets.BaseOffset, pd.offsets.Tick)):
    print("⚠️ Could not infer a frequency.")
    print("   Common options: D=daily, W=weekly, MS=month-start, M=month-end, Q=quarter-end, Y=year-end")
    user = input("Enter a pandas freq string (press Enter for 'D'): ").strip()
    freq = user or "D"

# Regularize to that freq and fill gaps
y = y.asfreq(freq).interpolate(limit_direction='both')

# Choose a safe forecast horizon: up to 12, but ≤ 1/3 of series length
fh = max(1, min(12, (len(y) // 3 if len(y) >= 3 else 1)))

print(f"✅ Using freq={freq} | fh={fh} | start={y.index.min()} | end={y.index.max()}")
```

```
⚠️ Could not infer a frequency.
   Common options: D=daily, W=weekly, MS=month-start, M=month-end, Q=quarter-end, Y=year-end
Enter a pandas freq string (press Enter for 'D'):
✅ Using freq=D | fh=12 | start=1981-01-01 00:00:00 | end=1990-12-31 00:00:00
```

[10]:

```python
exp = setup(y, fh=fh, fold=3, session_id=123, verbose=False)
best = compare_models()
print("✅ Best model:", best)

# In-sample & future forecast
plot_model(best, plot='forecast')          # combined chart
future_preds = predict_model(best, fh=fh)  # future points only
display(future_preds.head())
```

	Model	MASE	RMSSE	MAE	RMSE	MAPE	SMAPE	R2	TT (Sec)
auto_arima	Auto ARIMA	0.8976	0.8638	2.4404	3.0243	0.1813	0.1814	-0.5300	23.6733
lightgbm_cds_dt	Light Gradient Boosting w/ Cond. Deseasonalize & Detrending	0.9026	0.8777	2.4540	3.0730	0.1836	0.1788	-0.8064	12.5567
xgboost_cds_dt	Extreme Gradient Boosting w/ Cond. Deseasonalize & Detrending	0.9399	0.9829	2.5553	3.4411	0.2019	0.1800	-1.9251	0.2000
croston	Croston	0.9706	0.9178	2.6389	3.2134	0.1998	0.1955	-0.7803	0.0200
huber_cds_dt	Huber w/ Cond. Deseasonalize & Detrending	0.9843	0.9357	2.6761	3.2760	0.1918	0.1986	-0.8817	0.1433
omp_cds_dt	Orthogonal Matching Pursuit w/ Cond. Deseasonalize & Detrending	0.9844	0.9335	2.6763	3.2683	0.1892	0.1998	-0.8523	0.1467
en_cds_dt	Elastic Net w/ Cond. Deseasonalize & Detrending	0.9875	0.9323	2.6847	3.2639	0.1892	0.2008	-0.8403	0.3100
ridge_cds_dt	Ridge w/ Cond. Deseasonalize & Detrending	0.9888	0.9393	2.6883	3.2884	0.1916	0.1999	-0.8893	0.1400
br_cds_dt	Bayesian Ridge w/ Cond. Deseasonalize & Detrending	0.9888	0.9392	2.6883	3.2884	0.1916	0.1999	-0.8891	0.1433
lr_cds_dt	Linear w/ Cond. Deseasonalize & Detrending	0.9888	0.9393	2.6883	3.2884	0.1916	0.1999	-0.8893	0.3867
llar_cds_dt	Lasso Least Angular Regressor w/ Cond. Deseasonalize & Detrending	0.9895	0.9292	2.6901	3.2533	0.1885	0.2019	-0.8209	0.1433
lasso_cds_dt	Lasso w/ Cond. Deseasonalize & Detrending	0.9895	0.9292	2.6901	3.2533	0.1885	0.2019	-0.8210	0.1400
gbr_cds_dt	Gradient Boosting w/ Cond. Deseasonalize & Detrending	0.9961	0.9548	2.7081	3.3431	0.2111	0.1987	-1.0322	0.3500
rf_cds_dt	Random Forest w/ Cond. Deseasonalize & Detrending	1.0260	0.9940	2.7895	3.4800	0.1968	0.2079	-1.0664	0.7200
ada_cds_dt	AdaBoost w/ Cond. Deseasonalize & Detrending	1.0487	0.9783	2.8511	3.4252	0.2216	0.2096	-1.1085	0.2267
polytrend	Polynomial Trend Forecaster	1.0664	0.9887	2.8992	3.4616	0.1969	0.2207	-1.1112	0.0233
grand_means	Grand Means Forecaster	1.0696	0.9917	2.9080	3.4718	0.1975	0.2215	-1.1311	1.2067
catboost_cds_dt	CatBoost Regressor w/ Cond. Deseasonalize & Detrending	1.1621	1.0939	3.1594	3.8299	0.2448	0.2300	-1.8010	1.7333
et_cds_dt	Extra Trees w/ Cond. Deseasonalize & Detrending	1.1798	1.1058	3.2075	3.8717	0.2458	0.2463	-2.1528	0.4967

ets	ETS	1.3608	1.2047	3.6996	4.2176	0.2802	0.2536	-3.6666	0.1167
theta	Theta Forecaster	1.3874	1.2253	3.7719	4.2895	0.2846	0.2593	-3.7270	0.0367
exp_smooth	Exponential Smoothing	1.3884	1.2260	3.7746	4.2920	0.2847	0.2596	-3.7281	0.3100
naive	Naive Forecaster	1.4048	1.2566	3.8194	4.3990	0.2983	0.2591	-4.2236	1.9767
stlf	STLF	1.5029	1.2963	4.0860	4.5380	0.3041	0.2787	-4.5968	0.0433
snaive	Seasonal Naive Forecaster	1.5132	1.3213	4.1139	4.6258	0.3145	0.2798	-4.7365	0.0433
arima	ARIMA	1.5159	1.3290	4.1214	4.6527	0.3212	0.2805	-4.6732	0.2067

```
✅ Best model: AutoARIMA(random_state=123, sp=2, suppress_warnings=True)
```

	Model	MASE	RMSSE	MAE	RMSE	MAPE	SMAPE	R2
0	Auto ARIMA	0.3531	0.3976	0.9598	1.3910	0.0761	0.0723	-0.0042

	y_pred
1990-12-20	13.9152
1990-12-21	13.7877
1990-12-22	13.8091
1990-12-23	13.5670
1990-12-24	13.7510

+ Code + Markdown

[11]:

```
# Save future forecast and the prepared series
future_preds.to_csv('/kaggle/working/forecast_future.csv', index=True)
y.to_csv('/kaggle/working/series_cleaned.csv', index=True)

# Save PyCaret model
save_model(best, '/kaggle/working/best_ts_model')
print("✅ Saved: forecast_future.csv, series_cleaned.csv, best_ts_model.pkl")
```

```
Transformation Pipeline and Model Successfully Saved
✅ Saved: forecast_future.csv, series_cleaned.csv, best_ts_model.pkl
```