DTSA 5509 Final Project Summary

This project focuses on applying supervised machine learning techniques—specifically binary classification—to predict whether an individual's annual income exceeds $50,000 or falls below that threshold. Using demographic and employment-related data, the project explores how different machine learning models can learn from patterns in features such as education, occupation, age, and work hours to make accurate income predictions. This type of classification problem is widely used in socioeconomic modeling and workforce analytics, as it helps identify the key factors that influence income levels and supports data-driven decision-making in areas like policy planning, recruitment, and economic forecasting.

The primary goal of this project is to develop and evaluate models capable of accurately predicting a person's income category—greater than $50K or less than/equal to $50K—based on their personal and employment characteristics. Beyond achieving high model accuracy, the project aims to understand which variables most strongly influence income and how various machine learning techniques compare in performance. This goal is important because it demonstrates how data science can be used to extract meaningful insights from real-world data, improve fairness and transparency in predictive modeling, and refine data-driven policy or business strategies that depend on income-related predictions.

To achieve this, the project follows a structured workflow. It begins with a Data Summary to understand the dataset's structure and features, followed by Exploratory Data Analysis (EDA) to identify patterns, correlations, and potential outliers. The Data Cleaning phase ensures data quality by addressing missing values, duplicates, and inconsistencies. In the Preprocessing stage, features are transformed and encoded to prepare them for modeling. The Modeling section introduces baseline classifiers—Logistic Regression, Support Vector Classifier, and Random Forest—to establish benchmark performance. Feature Selection is then applied to reduce dimensionality and improve model efficiency, followed by Hyperparameter Tuning to optimize each model's parameters using cross-validation. Next, Ensemble Methods such as Voting and Stacking are implemented to combine the strengths of multiple models for improved prediction accuracy. Finally, the Results and Analysis stage evaluates all models using metrics like accuracy and ROC AUC, compares their performance, and highlights the most effective approaches and insights gained throughout the process.

Data Summary

Import Python Packages

In [106…

```
# import pandas as pd
import numpy as np
from IPython.display import display, Markdown
from ucimlrepo import fetch_ucirepo

import seaborn as sns
import matplotlib.pyplot as plt
import altair as alt
alt.data_transformers.disable_max_rows()
```

Out[106…

```
DataTransformerRegistry.enable('default')
```

Data Source

The dataset used in this project is the Adult Income Dataset, commonly known as the "Census Income" dataset, which originates from the U.S. Census Bureau's 1994 Census database. It contains demographic and employment-related information for over 48,000 individuals, with attributes such as age, education, occupation, work hours, marital status, and native country. The dataset's primary purpose is to predict whether a person earns more than $50,000 per year based on these features, making it a well-known benchmark for supervised binary classification tasks in machine learning.

The data was obtained in CSV format from the UCI Machine Learning Repository, where it is publicly available for academic and research use (Dua & Graff, 2019).

Reference:
Dua, D., & Graff, C. (2019). UCI Machine Learning Repository: Adult Data Set. University of California, Irvine, School of Information and Computer Sciences.
https://archive.ics.uci.edu/dataset/2/adult

In [107…

```
# Import dataset
adult_dataset_dict = fetch_ucirepo(id=2)

# View raw dataset
df = adult_dataset_dict.data.original
df
```

Out[107…

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
48837	39	Private	215419	Bachelors	13	Divorced	Prof-specialty	Not-in-family	White	Female	0	0	36	United-States	<=50K.
48838	64	NaN	321403	HS-grad	9	Widowed	NaN	Other-relative	Black	Male	0	0	40	United-States	<=50K.
48839	38	Private	374983	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband	White	Male	0	0	50	United-States	<=50K.
48840	44	Private	83891	Bachelors	13	Divorced	Adm-clerical	Own-child	Asian-Pac-Islander	Male	5455	0	40	United-States	<=50K.
48841	35	Self-emp-inc	182148	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	60	United-States	>50K.

48842 rows × 15 columns

Dataset Info

- `df.info()` provides a concise summary of a DataFrame, including the:
  - Number of rows
  - Column names
  - Data types
- It also shows the count of non-null entries for each column, which makes it easy to identify missing values.
- In addition, it displays the memory usage of the DataFrame, helping to assess the size and efficiency of the dataset in memory.

In [108…

```
# Inspect column data types and size of the dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             48842 non-null  int64
 1   workclass       47879 non-null  object
 2   fnlwgt          48842 non-null  int64
 3   education       48842 non-null  object
 4   education-num   48842 non-null  int64
 5   marital-status  48842 non-null  object
 6   occupation      47876 non-null  object
 7   relationship    48842 non-null  object
 8   race            48842 non-null  object
 9   sex             48842 non-null  object
 10  capital-gain    48842 non-null  int64
 11  capital-loss    48842 non-null  int64
 12  hours-per-week  48842 non-null  int64
 13  native-country  48568 non-null  object
 14  income          48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

> The dataset contains 15 columns and 48,842 rows of data.

> Six columns are integer datatypes, and the other 9 columns are categorical datatypes (shown as 'object' data-type above).

Dataset Metadata Summary

The table below provides the **metadata of each column** in our dataset:

- `name` are the column names of the features and target in our dataset
- `role` identifies if the column is a feature or target variable
- `type` is the column data type
- `demographic` describes the demographic type
- `description` details the various categories for the categorical features
- `units` are empty (No further information has been provided by the data source)

- `missing_values` is a flag to indicate if there are values missing for that feature (missing values will be confirmed and imputed during the EDA and Pre-Processing stages)

In [109…
```
# Create a dataframe of the dataset metadata (as provided by the data source)
adult_dataset_metadata_df = adult_dataset_dict.variables

# Display a markdown table of the metadata for each column
display(Markdown(adult_dataset_metadata_df.to_markdown(index=False)))
```

name	role	type	demographic	description	units	missing_values
age	Feature	Integer	Age	N/A		no
workclass	Feature	Categorical	Income	Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.		yes
fnlwgt	Feature	Integer				no
education	Feature	Categorical	Education Level	Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.		no
education-num	Feature	Integer	Education Level			no
marital-status	Feature	Categorical	Other	Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.		no
occupation	Feature	Categorical	Other	Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.		yes
relationship	Feature	Categorical	Other	Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.		no
race	Feature	Categorical	Race	White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.		no
sex	Feature	Binary	Sex	Female, Male.		no
capital-gain	Feature	Integer				no
capital-loss	Feature	Integer				no
hours-per-week	Feature	Integer				no
native-country	Feature	Categorical	Other	United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.		yes
income	Target	Binary	Income	>50K, <=50K.		no

Feature Descriptions

- The UCI Adult dataset is a derived subset by Becker & Kohavi of the U.S. Census 1994 microdata, so I couldn't reliably find a primary source document that provides descriptions of the parameters as provided by Becker & Kohavi.
- Therefore, I am using the feature descriptions as found in this Kaggle notebook, which are generally the same descriptions I've found elsewhere:
  - Vyass, Y. H. (2022). Adult census income logistic regression explained (86.2%) [Computer software]. Kaggle. https://www.kaggle.com/code/yashhvyass/adult-census-income-logistic-reg-explained-86-2

Feature Name	Description
age	The age of an individual
workclass	Employment status of an individual
fnlwgt	The number of people the census believes the entry represents
education	The highest level of education achieved by an individual
education-num	The highest level of education achieved in numerical form
marital-status	Marital status of an individual
occupation	The general type of occupation of an individual
relationship	The relationship status of an individual
race	The race of an individual
sex	The sex of an individual
capital-gain	The capital gains for an individual
capital-loss	The capital loss for an individual
hours-per-week	The hours an individual has reported to work per week
native-country	The country of origin of an individual

> - It can be noted that the `education` and `education-num` columns essentially describe the same thing, where `education` are the education labels and `education-num` are the ordinal encoding for those labels.
> - During the EDA stage these columns should be cross referenced to ensure that each category in `education` directly corresponds to the same ordinal value in `education-num` (i.e. 'HS-grad' in `education` always corresponds to the value 9 in `education`).

> - If they do not match then efforts should be made to replace the ordinal values in `education-num` column with the most frequent values for each category in `education`.
> - Once the `education` and `education-num` have a 1:1 match then the `education` column can be dropped during Data Cleaning stage, leaving only the ordinal encodings in `education-num` for modeling purposes.

EDA

In the EDA section, I start by defining which columns in the dataset are numeric and which are categorical, since this separation helps guide how each will be analyzed later in preprocessing. I then calculate descriptive statistics for the numeric features to understand their central tendencies, variability, and overall distributions, which also helps identify potential outliers. Next, I create histograms for each numeric column to visually assess their distributions and confirm where any extreme values, like those in `capital-gain` and `capital-loss`, might exist. After that, I check for correlations between numeric features using both a heatmap and a scatter plot matrix to see whether any strong relationships exist that could cause redundancy. I also examine the categorical columns by listing their unique categories and plotting their frequency distributions to get a sense of balance among categories and identify any `"?"` or missing values that need to be handled. Lastly, I compare the education and education-num columns using a heatmap to determine if they carry the same information, confirming that education-num is redundant and should be dropped later during data cleaning.

Define Numeric and Categorical Feature Columns Groups

- These groups will be used for EDA purposes, and during the Pre-Processing stage.

In [110…
```
# Define lists of the numeric and categorical column names
numeric_columns = df.select_dtypes(include=['int64']).columns.to_list()
categorical_columns = [col for col in df.columns if col not in numeric_columns]

# Display numeric and categorical columns as lists
print('Numeric Columns :', numeric_columns)
print('Categorical Columns :', categorical_columns)
```

```
Numeric Columns : ['age', 'fnlwgt', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']
Categorical Columns : ['workclass', 'education', 'marital-status', 'occupation', 'relationship', 'race', 'sex', 'native-country', 'income']
```

Numeric Columns Descriptive Statistics

- The descriptive statistics summary gives us high level insights for each feature, including the:
  - Values counts
  - Mean
  - Standard Deviation
  - Minimum values
  - 25th, 50th, and 75th Quartiles
  - Maximum values
- These statistics can be used to get a general sense of the distribution of each numerical feature, and to possibly detect any numerical outliers.

In [111…
```
# Descriptive statistics for the dataset's numeric columns
df[numeric_columns].describe()
```

Out[111…

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	48842.000000	4.884200e+04	48842.000000	48842.000000	48842.000000	48842.000000
mean	38.643585	1.896641e+05	10.078089	1079.067626	87.502314	40.422382
std	13.710510	1.056040e+05	2.570973	7452.019058	403.004552	12.391444
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.175505e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.781445e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.376420e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.490400e+06	16.000000	99999.000000	4356.000000	99.000000

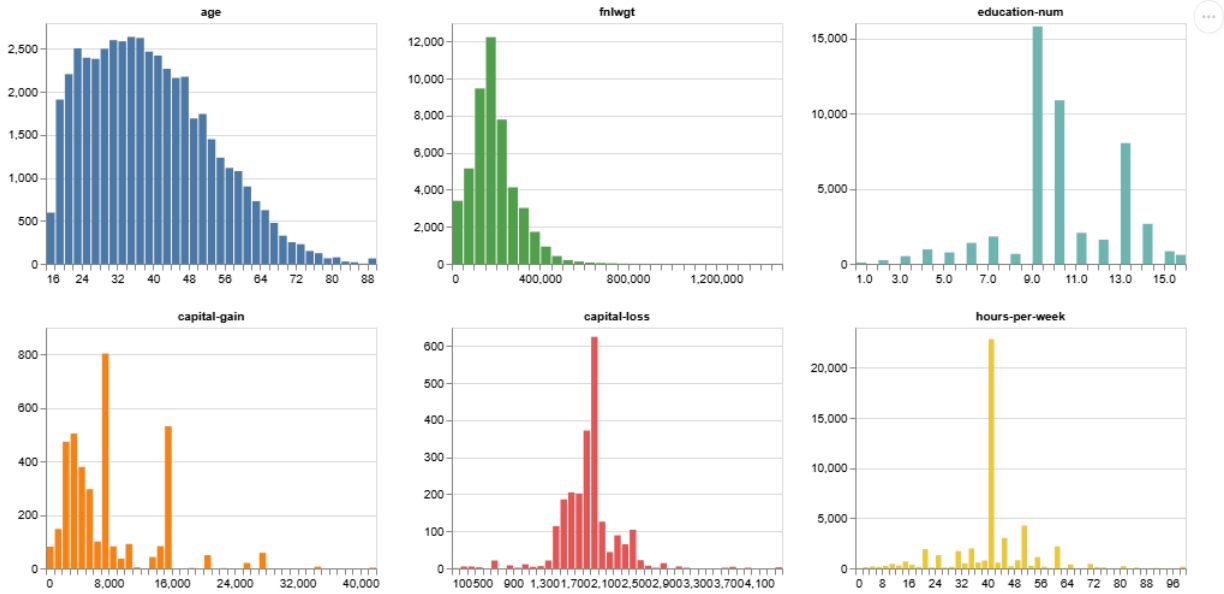
Numeric Columns Histograms

- Histograms of numeric columns allow us to visually interpret the distributions of the data. It will also allow us to identify where numeric outliers may exist.

In [135…
```
# Melt the numeric columns into one column
df_melt_numeric_columns = df[numeric_columns].melt(var_name='feature', value_name='value')

# Create a base Altair histogram chart
chart = alt.Chart(df_melt_numeric_columns).mark_bar().encode(
    x = alt.X('value:Q',
              axis=alt.Axis(title=''),
              scale=alt.Scale(zero=False),
              bin=alt.Bin(maxbins=50)),
    y = alt.Y('count():Q',
              axis=alt.Axis(title='')),
    color = alt.Color('feature:N', legend=None)
).properties(
    width=275,
    height=200
)
```

```
# Display a histogram for each numeric_columns
alt.ConcatChart(
    concat=[
      chart.transform_filter(alt.datum.feature == value).properties(title=value)
      for value in numeric_columns
    ],
    columns=3
).configure_title(
    fontSize=10
).resolve_axis(
    x='independent',
    y='independent'
).resolve_scale(
    x='independent',
    y='independent'
)
```

Out[135…



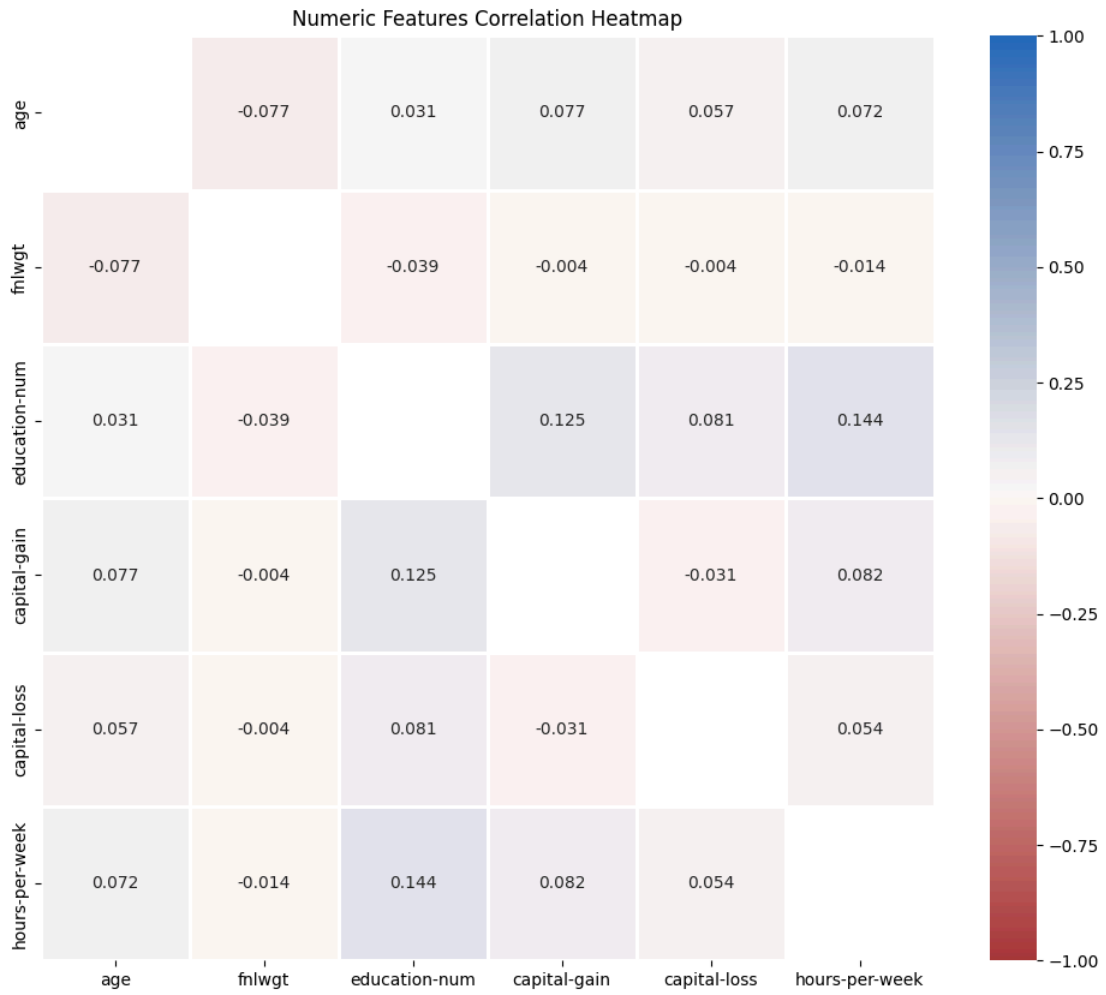
> The `capital-gain` and `capital-loss` columns seem to have a lot of values close to zero, as well as some extreme outliers.
>
> - In particular the maximum value for `capital-gain` is equal to 99,999 which seems odd given that most of the values are ~16,000 or less. This outlier needs to be investigated further to decide if the value makes sense or it was just a entry error. If so, we should remove the outlier and replace it with the median of `capital-gain` (median after the outlier has been removed) during the Data Cleaning stage.
> - The zero values in the `capital-gain` and `capital-loss` columns may just be an indicator that no capital gains or losses values were recorded, rather than the gains or losses equaling to exactly `$0`. It will be decided during Data Cleaning if it makes sense to replace the zero values and impute new values.

Numeric Column Correlations

- We need to check if there exists strong correlations between the numeric features. If there are then we should consider dropping one or more of the correlated features from the dataset.
- To aid in this evaluation we will visualize the data using:
  - A heatmap, where values of `[-1,1]` indicate as strong correlation, and values close to `0` indicate a weak correlation.
  - A matrix of scatter plots to help observe if there exists any trends between numeric features.

In [113…

```
fig, ax = plt.subplots(figsize=(12, 10))
# Create heatmap for numeric columns
sns.heatmap(
    df[numeric_columns].corr().replace(1,np.nan), # .replace() removes all of the 1's along the diagonal
    cmap='vlag_r',
    annot=True,
    fmt='.3f',
    linewidths=1,
    vmin=-1,vmax=1
)
plt.title('Numeric Features Correlation Heatmap')
plt.show()
```

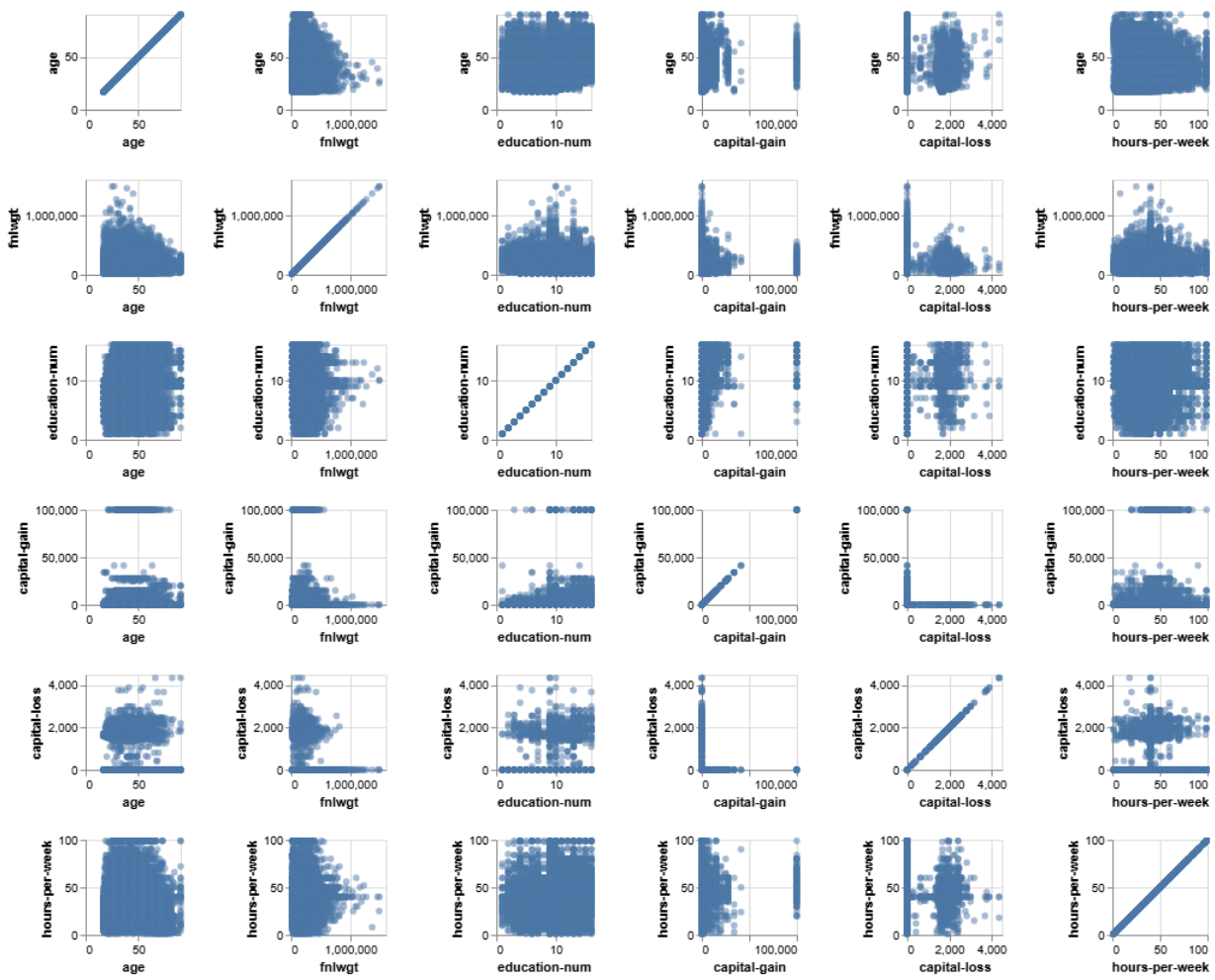


> All of the correlation values are very close to zero so we can assume that **no strong correlations exist between the numeric features**.

In [114…

```
# Scatter plot matrix
alt.Chart(df[numeric_columns]).mark_circle(opacity=0.5).encode(
    x=alt.X(alt.repeat('column'), type='quantitative'),
    y=alt.Y(alt.repeat('row'), type='quantitative')
).properties(
    width=80,
    height=80,
).repeat(
    row=numeric_columns,
    column=numeric_columns,
)
```

Out[114…



> Once again the outliers are noticeable in the `capital-gain` and `capital-loss` columns.

Frequency of Categorical Columns

- The categorical columns need to be investigated to understand what unique categories exist in each column, as well as the count of occurrence for each category.

In [115…

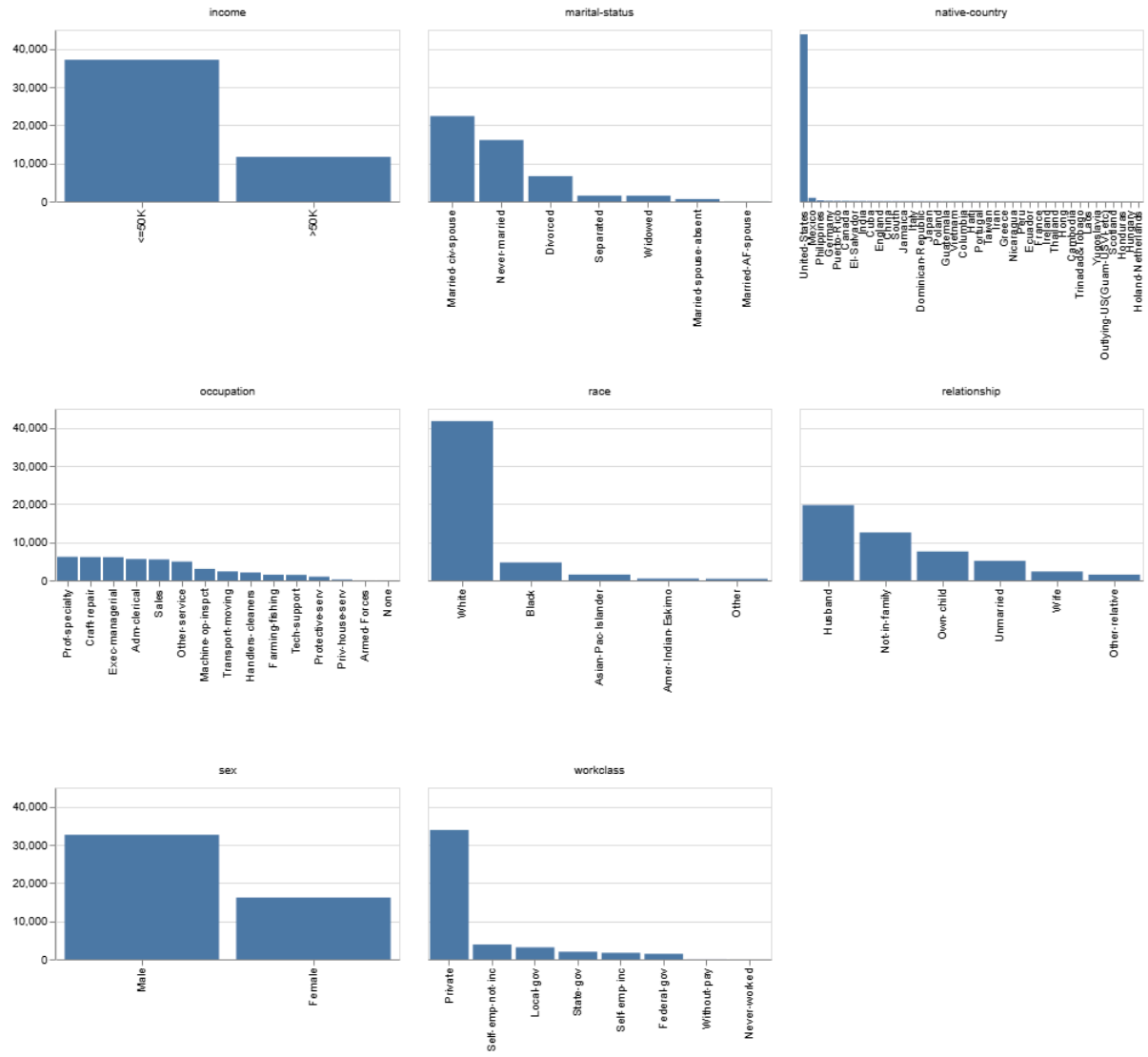
```
# Investigate the unique values in each of the categorical columns
for col in categorical_columns:
    print(f'Column name: {col}')
    print(f'Categories: {df[col].unique()}\n')
```

```
Column name: workclass
Categories: ['State-gov' 'Self-emp-not-inc' 'Private' 'Federal-gov' 'Local-gov' '?'
 'Self-emp-inc' 'Without-pay' 'Never-worked' nan]

Column name: education
Categories: ['Bachelors' 'HS-grad' '11th' 'Masters' '9th' 'Some-college' 'Assoc-acdm'
 'Assoc-voc' '7th-8th' 'Doctorate' 'Prof-school' '5th-6th' '10th'
 '1st-4th' 'Preschool' '12th']

Column name: marital-status
Categories: ['Never-married' 'Married-civ-spouse' 'Divorced' 'Married-spouse-absent'
 'Separated' 'Married-AF-spouse' 'Widowed']

Column name: occupation
Categories: ['Adm-clerical' 'Exec-managerial' 'Handlers-cleaners' 'Prof-specialty'
 'Other-service' 'Sales' 'Craft-repair' 'Transport-moving'
 'Farming-fishing' 'Machine-op-inspct' 'Tech-support' '?'
 'Protective-serv' 'Armed-Forces' 'Priv-house-serv' nan]

Column name: relationship
Categories: ['Not-in-family' 'Husband' 'Wife' 'Own-child' 'Unmarried' 'Other-relative']

Column name: race
Categories: ['White' 'Black' 'Asian-Pac-Islander' 'Amer-Indian-Eskimo' 'Other']

Column name: sex
Categories: ['Male' 'Female']

Column name: native-country
Categories: ['United-States' 'Cuba' 'Jamaica' 'India' '?' 'Mexico' 'South'
 'Puerto-Rico' 'Honduras' 'England' 'Canada' 'Germany' 'Iran'
 'Philippines' 'Italy' 'Poland' 'Columbia' 'Cambodia' 'Thailand' 'Ecuador'
 'Laos' 'Taiwan' 'Haiti' 'Portugal' 'Dominican-Republic' 'El-Salvador'
 'France' 'Guatemala' 'China' 'Japan' 'Yugoslavia' 'Peru'
 'Outlying-US(Guam-USVI-etc)' 'Scotland' 'Trinadad&Tobago' 'Greece'
 'Nicaragua' 'Vietnam' 'Hong' 'Ireland' 'Hungary' 'Holand-Netherlands' nan]

Column name: income
Categories: ['<=50K' '>50K' '<=50K.' '>50K.']
```

In [137…

```python
# Visualize the counts of each category

# Melt the categorical_columns
demographics_df = df[categorical_columns].melt(
    var_name='feature', value_name='category'
).dropna()

# Create a base Altair chart
base = (
    alt.Chart(demographics_df)
    .transform_aggregate(count='count()', groupby=['feature','category'])
    .transform_window(
        rank='rank()',
        sort=[alt.SortField('count', order='descending')],
        groupby=['feature']
    )
)

# Category counts facet chart
(base.mark_bar()
    .encode(
        y=alt.Y('count:Q', title=None),
        x=alt.X('category:N', sort='-y', title=None)
    )
    .properties(width=300, height=150)
    .facet(facet=alt.Facet('feature:N', title=None), align='all', columns=3)
    .resolve_scale(x ='independent')
)
```

Out[137…



> The `native-country`, `occupation`, and `workclass` features contain some rows with `?` values.
>
> - These values will need to be replaced with `np.nan`'s during the Data Cleaning stage.
> - Later during Pre-Processing those `np.nan`'s will be imputed with the most frequent categories for each feature.

> Otherwise, the distributions of the categorical columns are sensible and don't appear to contain any outliers.

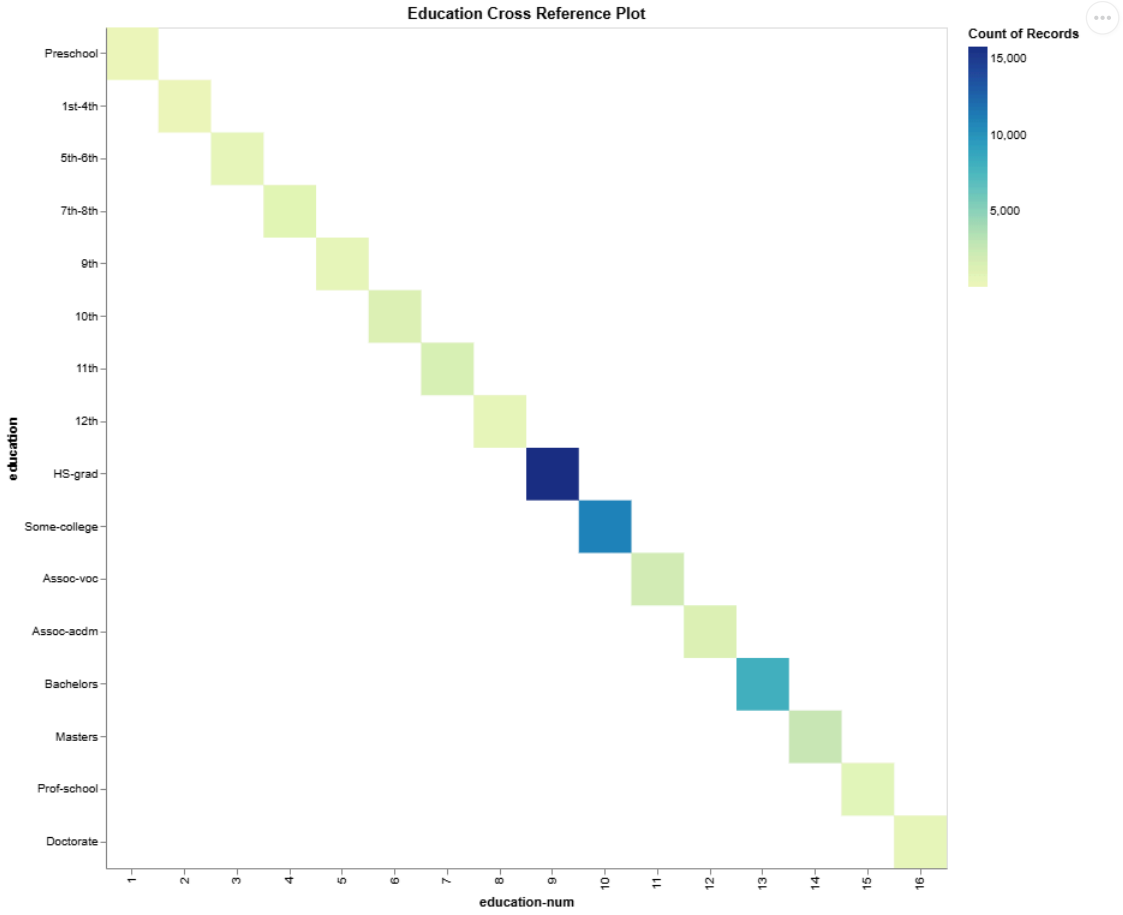
Cross-Reference Education Columns

- It was previously identified that the `education` and `education-num` columns need to be checked to see if they essentially contain the same information.
- A plot will be created to investigate if each individual `education` category corresponds to exactly one `education-num` value.

In [117…

```
# Create a subset of the dataframe for the education columns
education_df = df[['education','education-num']]

# Plot the chart
alt.Chart(education_df).mark_rect().encode(
    y=alt.Y('education:O', sort='x'),
    x='education-num:O',
    color='count()'
).properties(title='Education Cross Reference Plot', height=700, width=700)
```

Out[117...



> The chart above indicates that indeed there is only one occurrence of `education` for each occurrence of `education-num`.
>
> - Note: The color just indicates the count of occurrence in the dataset. What were most concerned about here is that **only one box appears for each row/column set** in the chart.

> This proves that the `education-num` column should be dropped during Data Cleaning since it contains redundant information.

EDA - Conclusions/Discussions/Next Steps:

In summary, the EDA process involved defining numeric and categorical feature groups, exploring descriptive statistics, visualizing distributions through histograms, examining correlations among numeric variables, assessing category frequencies, and cross-referencing the `education` and `education-num` columns for redundancy.

From this analysis, I found that most numeric variables have reasonable distributions with no strong correlations, although `capital-gain` and `capital-loss` contain extreme outliers and many zeros that will require further investigation. In the categorical features, several columns such as `workclass`, `occupation`, and `native-country` include `"?"` values that will need to be treated as missing data. Additionally, the `education-num` column was found to duplicate information from `education` and should be removed. These findings suggest that the primary challenges in the next stage will involve handling outliers, missing data, and redundant features.

The next step, Data Cleaning, will focus on addressing these issues to prepare a consistent and reliable dataset for model training.

Data Cleaning

In the Data Cleaning section, I begin by removing the `education` column since it was found to be redundant with `education-num` during the EDA stage. Next, I identify and drop duplicate rows to prevent data leakage and overfitting during model training. I then address numeric outliers in the `capital-gain` and `capital-loss` columns by flagging and replacing implausible values such as zeros and 99,999 with NaN, ensuring that these will later be imputed appropriately. For the categorical data, I replace `"?"` entries in `workclass`, `occupation`, and `native-country` with NaN to standardize missing values, and I clean the target variable `income` by removing trailing periods to maintain only two valid categories (<=50K and >50K). I then identify which numeric and categorical columns contain missing values and visualize where these occur, noting that `capital-gain`, `capital-loss`, `workclass`, `occupation`, and `native-country` have missing data that will need imputation. Finally, I handle a special case where `workclass` equals "Never-worked" by creating a new occupation category labeled "None." These steps ensure the dataset is consistent, free of duplicates, and properly formatted for the next phase of preprocessing and imputation.

Drop Columns

- The `education` column can be dropped because we have proven it to be redundant in the EDA stage. We will instead use the ordinal encoding of `education-num` for modeling.

In [118…
```
# Drop the 'education' column
df = df.drop(['education'], axis=1)

# Remove the 'education' column from the categorical_columns list
categorical_columns = [col for col in categorical_columns if col != 'education']

# Ensure the 'education' column has been dropped
df[categorical_columns].columns.to_list()
```

Out[118…
```
['workclass',
 'marital-status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native-country',
 'income']
```

Drop Duplicate Values

- Duplicate values need to be removed because there could be a chance that one copy lands in the training set and its twin lands in the test set when we split the dataset for supervised modeling. This would lead to over fitting because the model has been trained on an instance of the duplicate.

In [119…
```
# Display first 10 rows where duplicated values occur
df_duplicated = df[df.duplicated(keep=False)].sort_values(by=df.columns.to_list())
df_duplicated
```

Out[119...

	age	workclass	fnlwgt	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
36461	18	Self-emp-inc	378036	8	Never-married	Farming-fishing	Own-child	White	Male	0	0	10	United-States	<=50K.
48521	18	Self-emp-inc	378036	8	Never-married	Farming-fishing	Own-child	White	Male	0	0	10	United-States	<=50K.
17673	19	Private	97261	9	Never-married	Farming-fishing	Not-in-family	White	Male	0	0	40	United-States	<=50K
18698	19	Private	97261	9	Never-married	Farming-fishing	Not-in-family	White	Male	0	0	40	United-States	<=50K
6990	19	Private	138153	10	Never-married	Adm-clerical	Own-child	White	Female	0	0	10	United-States	<=50K
21318	19	Private	138153	10	Never-married	Adm-clerical	Own-child	White	Female	0	0	10	United-States	<=50K
15189	19	Private	146679	10	Never-married	Exec-managerial	Own-child	Black	Male	0	0	30	United-States	<=50K
21490	19	Private	146679	10	Never-married	Exec-managerial	Own-child	Black	Male	0	0	30	United-States	<=50K
3917	19	Private	251579	10	Never-married	Other-service	Own-child	White	Male	0	0	14	United-States	<=50K
31993	19	Private	251579	10	Never-married	Other-service	Own-child	White	Male	0	0	14	United-States	<=50K
5805	20	Private	107658	10	Never-married	Tech-support	Not-in-family	White	Female	0	0	10	United-States	<=50K
11631	20	Private	107658	10	Never-married	Tech-support	Not-in-family	White	Female	0	0	10	United-States	<=50K
8080	21	Private	243368	1	Never-married	Farming-fishing	Not-in-family	White	Male	0	0	50	Mexico	<=50K
15059	21	Private	243368	1	Never-married	Farming-fishing	Not-in-family	White	Male	0	0	50	Mexico	<=50K
4767	21	Private	250051	10	Never-married	Prof-specialty	Own-child	White	Female	0	0	10	United-States	<=50K
9171	21	Private	250051	10	Never-married	Prof-specialty	Own-child	White	Female	0	0	10	United-States	<=50K
21103	23	Private	240137	3	Never-married	Handlers-cleaners	Not-in-family	White	Male	0	0	55	Mexico	<=50K
25872	23	Private	240137	3	Never-married	Handlers-cleaners	Not-in-family	White	Male	0	0	55	Mexico	<=50K
33049	24	Private	194630	13	Never-married	Prof-specialty	Not-in-family	White	Male	0	0	35	United-States	<=50K.
33425	24	Private	194630	13	Never-married	Prof-specialty	Not-in-family	White	Male	0	0	35	United-States	<=50K.
5842	25	Private	195994	2	Never-married	Priv-house-serv	Not-in-family	White	Female	0	0	40	Guatemala	<=50K
13084	25	Private	195994	2	Never-married	Priv-house-serv	Not-in-family	White	Female	0	0	40	Guatemala	<=50K
22300	25	Private	195994	2	Never-married	Priv-house-serv	Not-in-family	White	Female	0	0	40	Guatemala	<=50K
4325	25	Private	308144	13	Never-married	Craft-repair	Not-in-family	White	Male	0	0	40	Mexico	<=50K
4881	25	Private	308144	13	Never-married	Craft-repair	Not-in-family	White	Male	0	0	40	Mexico	<=50K
5579	27	Private	255582	9	Never-married	Machine-op-inspct	Not-in-family	White	Female	0	0	40	United-States	<=50K
28230	27	Private	255582	9	Never-married	Machine-op-inspct	Not-in-family	White	Female	0	0	40	United-States	<=50K
8679	28	Private	274679	14	Never-married	Prof-specialty	Not-in-family	White	Male	0	0	50	United-States	<=50K
26313	28	Private	274679	14	Never-married	Prof-specialty	Not-in-family	White	Male	0	0	50	United-States	<=50K
41810	29	Private	36440	13	Never-married	Adm-clerical	Not-in-family	White	Female	0	0	40	United-States	<=50K.
43773	29	Private	36440	13	Never-married	Adm-clerical	Not-in-family	White	Female	0	0	40	United-States	<=50K.
16975	30	Private	144593	9	Never-married	Other-service	Not-in-family	Black	Male	0	0	40	?	<=50K
18555	30	Private	144593	9	Never-married	Other-service	Not-in-family	Black	Male	0	0	40	?	<=50K

	age	workclass	fnlwgt	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
39582	30	Private	180317	11	Divorced	Machine-op-inspct	Not-in-family	White	Male	0	0	40	United-States	<=50K.
46409	30	Private	180317	11	Divorced	Machine-op-inspct	Not-in-family	White	Male	0	0	40	United-States	<=50K.
16846	35	Private	379959	9	Divorced	Other-service	Not-in-family	White	Female	0	0	40	United-States	<=50K
32404	35	Private	379959	9	Divorced	Other-service	Not-in-family	White	Female	0	0	40	United-States	<=50K
33880	37	Private	52870	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	40	United-States	<=50K.
43750	37	Private	52870	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	40	United-States	<=50K.
4940	38	Private	207202	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	48	United-States	>50K
29157	38	Private	207202	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	48	United-States	>50K
25624	39	Private	30916	9	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	40	United-States	<=50K
28846	39	Private	30916	9	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	40	United-States	<=50K
10367	42	Private	204235	10	Married-civ-spouse	Prof-specialty	Husband	White	Male	0	0	40	United-States	>50K
28522	42	Private	204235	10	Married-civ-spouse	Prof-specialty	Husband	White	Male	0	0	40	United-States	>50K
17916	44	Private	367749	13	Never-married	Prof-specialty	Not-in-family	White	Female	0	0	45	Mexico	<=50K
22367	44	Private	367749	13	Never-married	Prof-specialty	Not-in-family	White	Female	0	0	45	Mexico	<=50K
11965	46	Private	133616	10	Divorced	Adm-clerical	Unmarried	White	Female	0	0	40	United-States	<=50K
30845	46	Private	133616	10	Divorced	Adm-clerical	Unmarried	White	Female	0	0	40	United-States	<=50K
16297	46	Private	173243	9	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	40	United-States	<=50K
17040	46	Private	173243	9	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	40	United-States	<=50K
7920	49	Private	31267	4	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	40	United-States	<=50K
21875	49	Private	31267	4	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	40	United-States	<=50K
7053	49	Self-emp-not-inc	43479	10	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	40	United-States	<=50K
22494	49	Self-emp-not-inc	43479	10	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	40	United-States	<=50K
2303	90	Private	52386	10	Never-married	Other-service	Not-in-family	Asian-Pac-Islander	Male	0	0	35	United-States	<=50K
5104	90	Private	52386	10	Never-married	Other-service	Not-in-family	Asian-Pac-Islander	Male	0	0	35	United-States	<=50K

> The dataset contains 57 rows that have duplicated values. Only the first occurrence will be kept and the other duplicates will be dropped to ensure the model is not over fit.

In [120…

```
# Check the shape of the dataset before dropping duplicates
print(f'Dataframe shape before dropping duplicates: {df.shape}')
rows_before = df.shape[0]

# Drop duplicate values
df = df.drop_duplicates()

# Check the shape of the dataset after dropping duplicates
print(f'Dataframe shape before after duplicates: {df.shape}')
rows_after = df.shape[0]

print(f'Number of duplicate rows removed = {rows_before-rows_after}')
```

```
Dataframe shape before dropping duplicates: (48842, 14)
Dataframe shape before after duplicates: (48813, 14)
Number of duplicate rows removed = 29
```

> 29 duplicate rows were removed from the dataset after dropping duplicates.

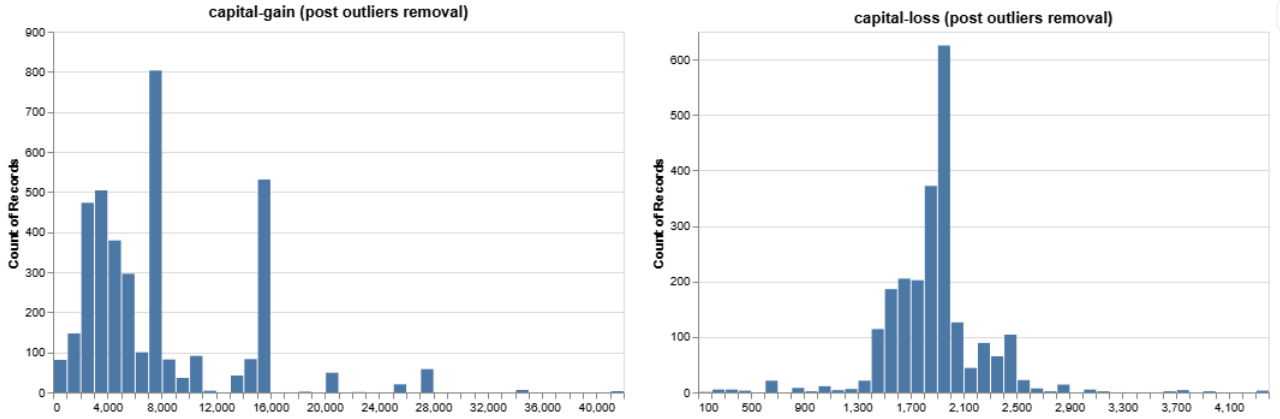
Identify Numeric Column Outliers

- The numeric column histograms indicated that the `capital-gain` and `capital-loss` columns have numeric outliers. In particular, both columns have a large amount of zero values.
  - The zero values may mean that the value of the capital gain or loss is truly equal to $0, but there is a chance that it also means that no gains or losses actually exist. This needs to be investigate further. If the zero values don't make sense then they will be removed from the dataset and new values imputed in their place.
- The `capital-gain` has a maximum value of `99,999` which appears to be an outlier in the dataset. This may have been accidentally entered as a placeholder and never removed from the original dataset. If it is determined that this is truly an outlier, then it will be removed from the dataset and new values imputed in it's place.

In [136…

```
# Create a subset of the dataframe for 'capital-gain' and 'capital-loss', removing outliers
capital_df = (df.copy()
    .loc[:,['capital-gain','capital-loss']]
    .melt(value_vars=['capital-gain','capital-loss'])
    .replace([0,99999], np.nan) # replace the zero and 99999 values
)

# Create a base histogram instance
capital_base_histogram = alt.Chart(capital_df).mark_bar().encode(
    x = alt.X('value:Q', title=None, bin=alt.Bin(maxbins=50)),
    y = 'count():Q'
).properties(title='capital-gain', width=475)

# Create histograms of 'capital-gain' and 'capital-loss' (post outliers removal)
capital_gain_histogram = capital_base_histogram.transform_filter(alt.datum.variable=='capital-gain').properties(title='capital-gain (post outliers rem
capital_loss_histogram = capital_base_histogram.transform_filter(alt.datum.variable=='capital-loss').properties(title='capital-loss (post outliers rem

# Concatenate and display post outliers removal histograms
capital_gain_histogram | capital_loss_histogram
```

Out[136…



> Removing the numeric outliers makes sense because they don't appear to be part of the main distributions of the `capital-gain` and `capital-loss` columns. These will be removed in the next step.

Replace Values

- The categorical columns frequency charts indicate that some rows in the `workclass`, `occupation`, and `native-country` features have been filled in with a `?`. It will be assumed that these are unknown values and as such should be replaced with `np.nan` values instead.
- Also, the target variable `income` appears to have four unique categories: `<=50K`, `>50K`, `<=50K.`, `>50K.`. Because we are modeling a binary classification problem we need to ensure there are only two unique categories. Therefore, the periods should be removed where they exist so the target only contains the two categories of `<=50K` and `>50K`.
- Finally, the outliers in the numeric columns need to be removed to better represent the distributions of their data.

In [122…

```
# Replace ? with np.nan in the `workclass`, `occupation`, and `native-country` columns
df = df.replace({'?':np.nan})

# Ensure the '?' have been removed from the `workclass`, `occupation`, and `native-country` columns
for col in ['workclass', 'occupation', 'native-country']:
    print(f'Column name: {col}')
    print(f'Categories: {df[col].unique()}\n')
```

```
Column name: workclass
Categories: ['State-gov' 'Self-emp-not-inc' 'Private' 'Federal-gov' 'Local-gov' nan
 'Self-emp-inc' 'Without-pay' 'Never-worked']

Column name: occupation
Categories: ['Adm-clerical' 'Exec-managerial' 'Handlers-cleaners' 'Prof-specialty'
 'Other-service' 'Sales' 'Craft-repair' 'Transport-moving'
 'Farming-fishing' 'Machine-op-inspct' 'Tech-support' nan
 'Protective-serv' 'Armed-Forces' 'Priv-house-serv']

Column name: native-country
Categories: ['United-States' 'Cuba' 'Jamaica' 'India' nan 'Mexico' 'South'
 'Puerto-Rico' 'Honduras' 'England' 'Canada' 'Germany' 'Iran'
 'Philippines' 'Italy' 'Poland' 'Columbia' 'Cambodia' 'Thailand' 'Ecuador'
 'Laos' 'Taiwan' 'Haiti' 'Portugal' 'Dominican-Republic' 'El-Salvador'
 'France' 'Guatemala' 'China' 'Japan' 'Yugoslavia' 'Peru'
 'Outlying-US(Guam-USVI-etc)' 'Scotland' 'Trinadad&Tobago' 'Greece'
 'Nicaragua' 'Vietnam' 'Hong' 'Ireland' 'Hungary' 'Holand-Netherlands']
```

In [123…
```
# Replace the '.' in the 'income' column
df.loc[:,'income'] = df.loc[:,'income'].str.replace('.','')

# Ensure the '.' have been removed from the 'income' column
for col in ['income']:
    print(f'Column name: {col}')
    print(f'Categories: {df[col].unique()}\n')
```

```
Column name: income
Categories: ['<=50K' '>50K']
```

In [124…
```
# Replace the numeric outliers in 'capital-gain' and 'capital-loss' columns
df[['capital-gain','capital-loss']] = df[['capital-gain','capital-loss']].replace([0,99999], np.nan)
```

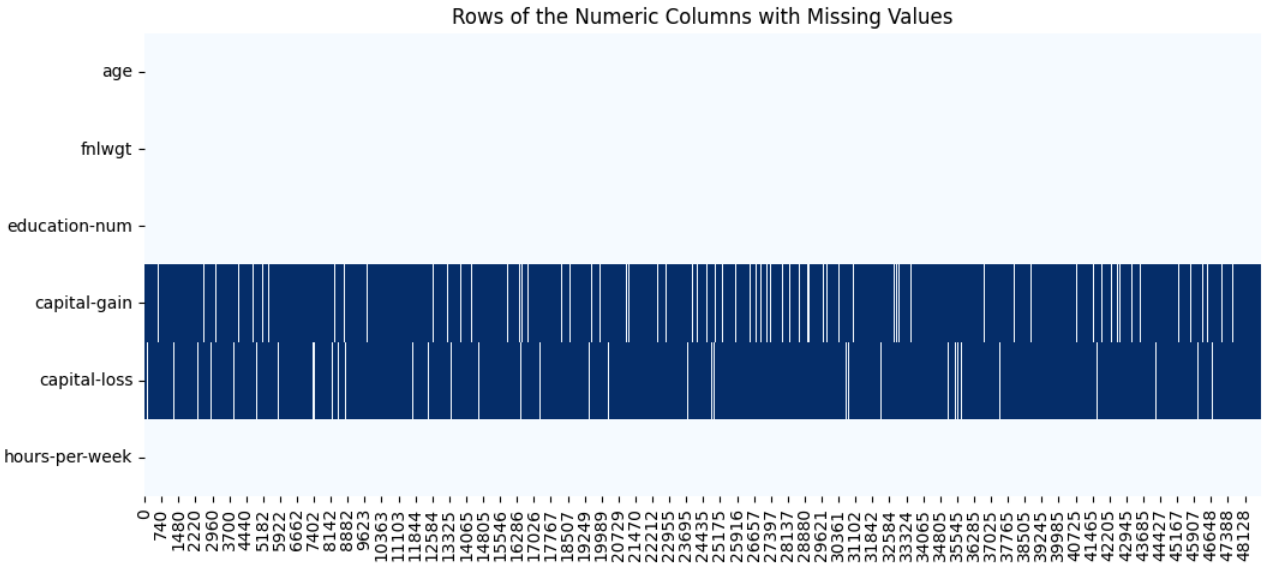
Identify Missing Numeric Values

- We need to investigate where the numeric outliers values of `0` and `99999` have been replaced with NaNs so we will be able to correctly impute the missing values during pre-processing

In [125…
```
# Find which numeric columns contain missing values in the dataset
df[numeric_columns].isna().any(axis=0)
```

Out[125…
```
age              False
fnlwgt           False
education-num    False
capital-gain      True
capital-loss      True
hours-per-week   False
dtype: bool
```

In [126…
```
# Filter the dataset to only the rows of the numeric columns that have missing values
numeric_missing_df = df[numeric_columns][
    (df['capital-gain'].isna()) |
    (df['capital-loss'].isna())
]

# Visualize the rows where the numeric columns have missing values
fig, ax = plt.subplots(figsize=(12,5))
sns.heatmap(numeric_missing_df.T.isna(), cmap='Blues', cbar=False)
plt.title('Rows of the Numeric Columns with Missing Values')
plt.show()
```



> These missing numeric values will need to be imputed during Pre-Processing

Identify Missing Categorical Values

- We need to investigate which categorical columns contain missing values so we will be able to correctly impute the missing values during pre-processing

In [127…
```
# Find which categorical columns contain missing values in the dataset
df[categorical_columns].isna().any(axis=0)
```

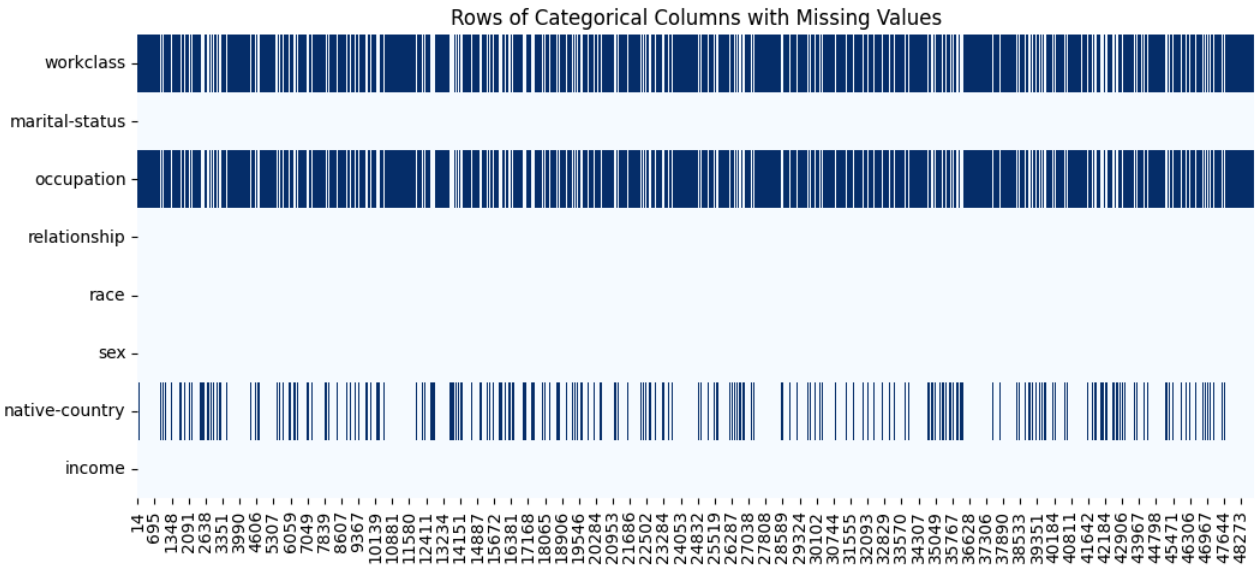
Out[127…
```
workclass          True
marital-status    False
occupation         True
relationship      False
race              False
sex               False
native-country     True
income            False
dtype: bool
```

> It appears that the `workclass`, `occupation`, and `native-country` categorical columns all contain missing values.

In [128…
```
# Calculate the count of missing values for the 'workclass', 'occupation', and 'native-country' columns
df[categorical_columns].isna().sum()[df[categorical_columns].isna().sum() > 0]
```

Out[128…
```
workclass         2799
occupation        2809
native-country     856
dtype: int64
```

In [129…
```
# Filter the dataset to only the rows where the categorical columns have missing values
categorical_missing_df = df[categorical_columns][
    (df['workclass'].isna()) |
    (df['occupation'].isna()) |
    (df['native-country'].isna())
]

# Visualize the rows of the categorical columns that have missing values
fig, ax = plt.subplots(figsize=(12,5))
sns.heatmap(categorical_missing_df.T.isna(), cmap='Blues', cbar=False)
plt.title('Rows of Categorical Columns with Missing Values')
plt.show()
```



> The `workclass` and `occupation` columns appear to have similar patterns of where the data is missing. The missing value count table indicates that there are 10 more missing values in `occupation` compared to `workclass`. It would be interesting to find out where `workclass` is not NaN and where `occupation` is.

In [130…
```
# Filter to rows where workclass is NaN and occupation is not NaN
categorical_missing_subset_df = categorical_missing_df[(categorical_missing_df['workclass'].notna()) & (categorical_missing_df['occupation'].isna())]
categorical_missing_subset_df
```

Out[130…

	workclass	marital-status	occupation	relationship	race	sex	native-country	income
5361	Never-worked	Never-married	NaN	Own-child	White	Male	United-States	<=50K
10845	Never-worked	Divorced	NaN	Not-in-family	White	Male	United-States	<=50K
14772	Never-worked	Never-married	NaN	Own-child	White	Male	United-States	<=50K
20337	Never-worked	Never-married	NaN	Own-child	White	Female	United-States	<=50K
23232	Never-worked	Never-married	NaN	Own-child	Black	Male	United-States	<=50K
32304	Never-worked	Married-civ-spouse	NaN	Wife	Black	Female	United-States	<=50K
32314	Never-worked	Never-married	NaN	Own-child	White	Male	United-States	<=50K
41346	Never-worked	Never-married	NaN	Own-child	Black	Female	United-States	<=50K
44168	Never-worked	Married-spouse-absent	NaN	Other-relative	White	Male	United-States	<=50K
46459	Never-worked	Never-married	NaN	Own-child	White	Male	United-States	<=50K

> It's interesting that `workclass` is only equal to 'Never-worked' where the `occupation` is missing. Let's check where else 'Never-worked' occurs in `workclass`.

In [131…

```
# Filter the original dataset to where 'workclass' is 'Never-worked'
workclass_never_worked_df = df[(df['workclass']=='Never-worked')]
workclass_never_worked_df
```

Out[131…

	age	workclass	fnlwgt	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
5361	18	Never-worked	206359	6	Never-married	NaN	Own-child	White	Male	NaN	NaN	40	United-States	<=50K
10845	23	Never-worked	188535	4	Divorced	NaN	Not-in-family	White	Male	NaN	NaN	35	United-States	<=50K
14772	17	Never-worked	237272	6	Never-married	NaN	Own-child	White	Male	NaN	NaN	30	United-States	<=50K
20337	18	Never-worked	157131	7	Never-married	NaN	Own-child	White	Female	NaN	NaN	10	United-States	<=50K
23232	20	Never-worked	462294	10	Never-married	NaN	Own-child	Black	Male	NaN	NaN	40	United-States	<=50K
32304	30	Never-worked	176673	9	Married-civ-spouse	NaN	Wife	Black	Female	NaN	NaN	40	United-States	<=50K
32314	18	Never-worked	153663	10	Never-married	NaN	Own-child	White	Male	NaN	NaN	4	United-States	<=50K
41346	17	Never-worked	131593	7	Never-married	NaN	Own-child	Black	Female	NaN	NaN	20	United-States	<=50K
44168	20	Never-worked	273905	9	Married-spouse-absent	NaN	Other-relative	White	Male	NaN	NaN	35	United-States	<=50K
46459	18	Never-worked	162908	7	Never-married	NaN	Own-child	White	Male	NaN	NaN	35	United-States	<=50K

> It seems like these are the only rows where 'Never-worked' occurs, which corresponds to the same rows we found in categorical_missing_subset_df.
>
> I'm going to make the assumption that in this case where `workclass` is 'Never-worked' then `occupation` must equal to None. Therefore I'm going to replace these missing values with a new `occupation` category of 'None'.

In [132…

```
# Replace occupation with 'None' where workclass=='Never-worked'
df['occupation'] = np.where(
    (df['workclass']=='Never-worked'),
    'None',
    df['occupation']
)

# Check that 'None' has been added to the unique categories of the 'occupation' column
df['occupation'].unique()
```

Out[132…

```
array(['Adm-clerical', 'Exec-managerial', 'Handlers-cleaners',
       'Prof-specialty', 'Other-service', 'Sales', 'Craft-repair',
       'Transport-moving', 'Farming-fishing', 'Machine-op-inspct',
       'Tech-support', nan, 'Protective-serv', 'Armed-Forces',
       'Priv-house-serv', 'None'], dtype=object)
```

Identify Class Imbalance

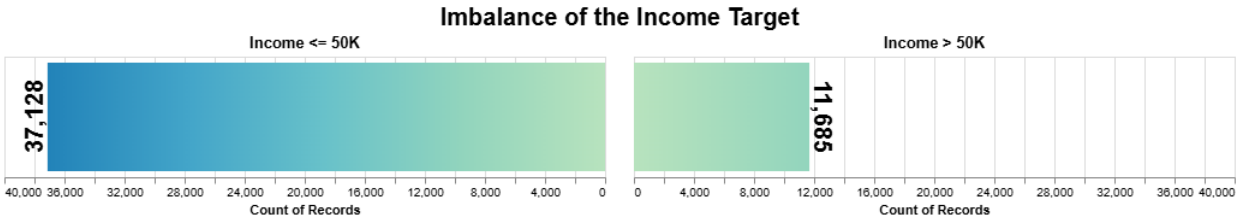
- Since our task is a binary classification of income with categories [<=50K, >50K], we will need to verify whether the target variable is balanced—that is, whether both classes occur in roughly equal proportions. Working with an imbalanced dataset can lead to biased models that perform well on the majority class but fail to correctly identify or predict the minority class, resulting in misleading accuracy scores and poor generalization. If we find that the distribution is uneven then I will experiment with the SMOTE oversampling technique to attempt to get more accurate models results during the Modeling phase of the project.

In [133…
```
# Calculate the percentage of each target class for the "income" target variable
lt50_income_count = df[df['income']=='<=50K']['income'].count()
gt50_income_count = df[df['income']=='>50K']['income'].count()
total_income_count = len(df)

print(f'The count of the income category ">=50K" is {lt50_income_count:,d} out of the total count of {total_income_count:,d} representing {lt50_income
print(f'The count of the income category  "<50K" is {gt50_income_count:,d} out of the total count of {total_income_count:,d} representing {gt50_income
```

```
The count of the income category ">=50K" is 37,128 out of the total count of 48,813 representing 76.1% of the income values.
The count of the income category  "<50K" is 11,685 out of the total count of 48,813 representing 23.9% of the income values.
```

In [134…
```
# Create the left bar chart for the count of income <=50
lt50_chart = alt.Chart(df, title='Income <= 50K', height=100, width=500
    ).mark_bar(
        color=alt.Gradient(
            gradient='linear',
            stops=[
                alt.GradientStop(color='#b8e3be', offset=1.00),
                alt.GradientStop(color='#93d5bd', offset=0.75),
                alt.GradientStop(color='#69c2ca', offset=0.5),
                alt.GradientStop(color='#43a5c9', offset=0.25),
                alt.GradientStop(color='#2283b9', offset=0.00),
                ],
                x1=0, x2=1, y1=1, y2=1
            )
    ).transform_filter(
        alt.datum.income == '<=50K'
    ).encode(
        x=alt.X('count():Q', sort='descending', scale=alt.Scale(domain=(0,40000)))
    )
lt50_text = lt50_chart.mark_text(align='left', dy=-10, dx=-30, angle=270, fontWeight='bold', fontSize=20).encode(
        text=alt.Text('count():Q', format=',d')
    )

# Create the right bar chart for the count of income >50
gt50_chart = alt.Chart(df, title='Income > 50K', height=100, width=500
    ).mark_bar(
        color=alt.Gradient(
            gradient='linear',
            stops=[
                alt.GradientStop(color='#b8e3be', offset=0),
                alt.GradientStop(color='#93d5bd', offset=1),
                ],
                x1=0, x2=1, y1=1, y2=1
            )
    ).transform_filter(
        alt.datum.income == '>50K'
    ).encode(
        x=alt.X('count():Q', scale=alt.Scale(domain=(0,40000)))
    )
gt50_text = gt50_chart.mark_text(align='right', dy=-10, dx=30, angle=90, fontWeight='bold', fontSize=20).encode(
    text=alt.Text('count():Q', format=',d')
)

# Combine charts
alt.concat(lt50_chart+lt50_text, gt50_chart+gt50_text, title=alt.Title('Imbalance of the Income Target', fontWeight='bold', fontSize=20, anchor='middl
```

Out[134…



Data Cleaning - Conclusions/Discussions/Next Steps:

In summary, the Data Cleaning process focused on removing redundant and duplicate records, addressing outliers, and standardizing missing and inconsistent values across the dataset to ensure data integrity. During this process, we also discovered that the target variable was imbalanced, with 37,128 rows labeled "≤50K" and 11,685 rows labeled ">50K." This imbalance will be addressed during preprocessing to help improve model fairness and predictive performance.

One key insight was that rows with workclass equal to "Never-worked" consistently had missing occupation values, prompting the creation of a new "None" category. These findings show that although the dataset is now well-prepared, careful imputation will be necessary to prevent bias and preserve accuracy.

The next step, Preprocessing, will focus on handling the class imbalance, imputing missing values, encoding categorical variables, and scaling numeric features for modeling.

In [30]:
```
# Export the dataset so it can be used in the "5509_income_modeling.ipynb" workbook
df.to_csv('./df.csv')
```

Pre-Processing

In the Preprocessing stage, I will first split the dataset into training and testing sets to ensure that the models can be properly trained and evaluated on separate data. The training set is used to fit and optimize the supervised classification models, while the testing set is reserved for assessing their predictive performance.

Next, I create preprocessing pipelines for both numeric and categorical features. The numeric transformer uses an `SimpleImputer` to fill in missing values based on the medians of the numeric variables, followed by a `StandardScaler` to normalize their ranges. The categorical transformer also uses an `SimpleImputer` with the most frequent category strategy to handle missing values and then applies a `OneHotEncoder` to convert categorical features into numerical format suitable for modeling. These transformations are combined within a `ColumnTransformer` to ensure that each feature type is processed appropriately and consistently, producing a clean and fully prepared dataset for model training.

Import Python Packages & Dataset

In [129…
```
# %load_ext cudf.pandas
import pandas as pd
import numpy as np
import os
import joblib

from sklearn.pipeline import Pipeline  as SkPipeline
from imblearn.pipeline import Pipeline as ImbPipeline
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold, learning_curve
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier, StackingClassifier, VotingClassifier
from sklearn.metrics import confusion_matrix, classification_report, RocCurveDisplay, roc_auc_score

import seaborn as sns
import matplotlib.pyplot as plt
import altair as alt
alt.data_transformers.disable_max_rows()
from IPython.display import Image

# Define column types
categorical_columns = [
    'workclass',
    'marital-status',
    'occupation',
    'relationship',
    'race',
    'sex',
    'native-country',
    'income'
 ]
numeric_columns = [
    'age',
    'fnlwgt',
    'education-num',
    'capital-gain',
    'capital-loss',
    'hours-per-week'
 ]

# Import dataset that was created in the '5509_income_pre_modeling.ipynb' workbook
df = pd.read_csv('./df.csv')
```

In [73]:
```
# Downcast floats datatypes to minimize memory usage
for col in df.select_dtypes(include=['float64']).columns:
    df[col] = pd.to_numeric(df[col], downcast='float')

# Downcast interger datatypes to minimize memory usage
for col in df.select_dtypes(include=['int64']).columns:
    df[col] = pd.to_numeric(df[col], downcast='integer')

# Instantiate an empty dataframe that will be used to store each models precision, recall, and F1 scores by target category
model_results_df = pd.DataFrame([])

# Instantiate an empty dataframe that will be used to store each models AUC
model_roc_auc_df = pd.DataFrame([])

# Change categorical columns to category datatype
# df[categorical_columns] = df[categorical_columns].astype('category')
```

Train Test Split

- The dataset is now divided into a training set that will be used to build supervised classification models, and a testing set for evaluating their performance.

In [74]:
```
# Define the predictors
X = df[['age', 'workclass', 'fnlwgt', 'education-num', 'marital-status', 'occupation', 'relationship', 'race', 'sex', 'capital-gain',
       'capital-loss', 'hours-per-week', 'native-country']]
```

```
# Define the target
y = df['income']

# Drop the target from the categorical_columns list
categorical_columns = [col for col in categorical_columns if col != 'income']

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

# Check shape of X_train, X_test, y_train, & y_test
print(f'X_train shape = {X_train.shape}    y_train shape = {y_train.shape}')
print(f'X_test shape =  {X_test.shape}     y_test shape =  {y_test.shape}')
```

```
X_train shape = (39050, 13)    y_train shape = (39050,)
X_test shape =  (9763, 13)     y_test shape =  (9763,)
```

Preprocessing Transformers

- Next we build a preprocessing pipeline that prepares numeric and categorical data for modeling. Numeric features are imputed using their median values and standardized for consistent scaling, while categorical features are imputed with the most frequent category and one-hot encoded into binary variables. The `ColumnTransformer` then applies these transformations to their respective columns, producing a clean, model-ready dataset.
- To aid in modeling dummy columns will be added during imputing to indicate where data was missing for the `capital-gain`, `capital-loss`, `workclass`, `occupation`, and `native-country` columns.

In [75]:
```
numeric_transformer = SkPipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_transformer = SkPipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor_transformer = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_columns),
        ('cat', categorical_transformer, categorical_columns)
    ],
    remainder='drop'
)
```

Preprocessing - Conclusions/Discussions/Next Steps:

During preprocessing, the main challenge identified was ensuring that the imputation process accurately captured relationships among variables without introducing bias, especially given the number of missing values in both numeric and categorical features.

With the data now standardized and encoded, the next step will be to apply supervised learning models to classify `income` levels and evaluate their performance on the testing set.

Base Classifiers Modeling

In the Modeling section, I first define several helper functions to streamline the evaluation process by generating learning curves, classification reports, confusion matrices, ROC curves, and feature selection summaries.

Three supervised classification models are developed:

- a Logistic Regression model to estimate class probabilities using a sigmoid function,
- a Support Vector Classifier (SVC) to separate classes by finding the optimal decision boundary, and
- a Random Forest Classifier to aggregate multiple decision trees for improved accuracy and robustness.

Each model is trained using the preprocessed training data and evaluated on the test set to measure performance.

Finally, ROC curves are plotted to compare all three classifiers and visually assess their ability to distinguish between `income` classes.

Modeling Helper Functions

- These helper functions are used to evaluate model performance by:
  - Fitting the model (`fit_model` function)
  - Plotting the learning curve plots (`plot_learning_curve` function)
  - Generating classification tables and confusion matrices (`create_classification_output` function)
  - Plot ROC curves (`make_roc_curves` function)
- There are also functions used to:
  - Identify dropped columns during feature selection (`showdropped_features` function)
  - Extract the best hyperparameters from cross-validation (`best_params_for` function)

In [76]:
```
# Function to check if a model has already been fit then load the model, otherwise fit the model
def fit_model(pipe, X_train, y_train, filename):
    if os.path.exists(filename):
        print(f'Loading saved model from {filename}')
        fitted_model = joblib.load(filename)
```

```
    else:
        fitted_model = pipe.fit(X_train, y_train)
        joblib.dump(fitted_model, filename)
        print(f'Model fitted and saved to {filename}')
    return fitted_model

# Function to plot the learning curve for a given model
def plot_learning_curve(estimator, X, y, clf_type):
    print('\n' + '=' * (len(clf_type) + 15))
    print(f'{clf_type} Learning Curve')
    print('=' * (len(clf_type) + 15))
    train_sizes, train_scores, val_scores = learning_curve(
        estimator=estimator,
        X=X,
        y=y,
        cv=StratifiedKFold(n_splits=3, shuffle=True, random_state=42),
        scoring='roc_auc',
        n_jobs=1,
        train_sizes=np.linspace(0.2, 1.0, 5),
        shuffle=True,
        random_state=42,
    )
    train_mean, train_std = train_scores.mean(axis=1), train_scores.std(axis=1)
    val_mean, val_std = val_scores.mean(axis=1), val_scores.std(axis=1)

    plt.figure(figsize=(6, 4))
    plt.plot(train_sizes, train_mean, 'o-', label='Training')
    plt.plot(train_sizes, val_mean, 'o-', label='Validation')
    plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.1)
    plt.fill_between(train_sizes, val_mean - val_std, val_mean + val_std, alpha=0.1)
    plt.xlabel('Training set size')
    plt.ylabel('ROC AUC')
    plt.title(f'Learning Curve (ROC AUC)\n({clf_type})')
    plt.legend(loc='best')
    plt.tight_layout()
    plt.show()

# Function to output the classification report and confusion matrix for a given model
def create_classification_output(pipe, y_test, y_pred, clf_type):
    # Create classification report
    print('\n' + '=' * (len(clf_type) + 15))
    print(f'{clf_type} Metrics Report')
    print('=' * (len(clf_type) + 15))
    print(classification_report(y_test, y_pred, digits=4))
    report = classification_report(y_test, y_pred, digits=4, output_dict=True)

    # Append model results to the model_results_df
    temp_result_df = pd.DataFrame.from_dict(
        {k:v for k,v in report.items() if k in ['<=50K', '>50K']},
        orient='index'
    ).reset_index(names='income')
    temp_result_df['model'] = clf_type
    global model_results_df
    model_results_df = pd.concat([model_results_df, temp_result_df])

    # Append model roc_auc to the model_roc_auc_df
    if hasattr(pipe, 'predict_proba'):
        y_score = pipe.predict_proba(X_test)[:, 1]
    else:
        y_score = pipe.decision_function(X_test)
    auc = roc_auc_score(y_test, y_score)
    print(f'The {clf_type} ROC AUC = {auc:.4f}')
    temp_roc_auc_df = pd.DataFrame.from_dict(
            {clf_type:auc},
            orient='index',
            columns=['roc_auc']
        ).reset_index(names='model')
    global model_roc_auc_df
    model_roc_auc_df = pd.concat([model_roc_auc_df, temp_roc_auc_df])

    # Plot confusion matrix
    print('\n' + '=' * (len(clf_type) + 17))
    print(f'{clf_type} Confusion Matrix')
    print('=' * (len(clf_type) + 17))
    labels = ['<=50K', '>50K']
    cm = confusion_matrix(y_test, y_pred, labels=labels)
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
    plt.title(f'Test Data Confusion Matrix\n({clf_type})')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.tight_layout()
    plt.show()

# Function to plot ROC curves and a AUC summary chart to compare models
def make_roc_curves(X_test, y_test, models):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6), gridspec_kw={'width_ratios': [1, 1]})
    aucs, colors = {}, {}

    for name, model in models.items():
        if hasattr(model, 'predict_proba'):
            y_score = model.predict_proba(X_test)[:, 1]
        else:
            y_score = model.decision_function(X_test)
```

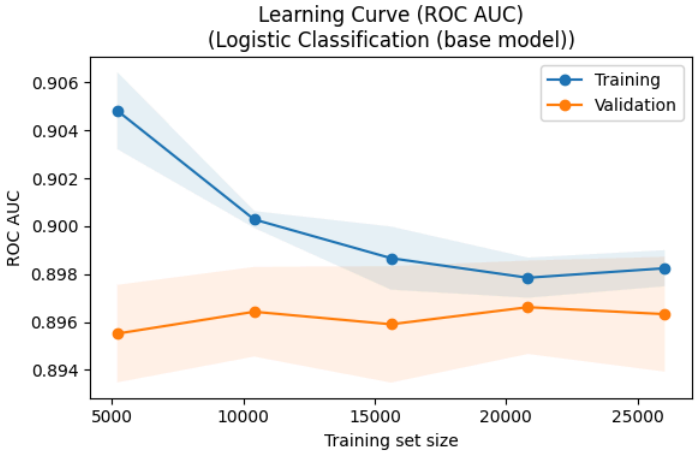
```
        aucs[name] = roc_auc_score(y_test, y_score)
        disp = RocCurveDisplay.from_predictions(y_test, y_score, name=name, ax=ax1, pos_label='>50K')
        colors[name] = disp.line_.get_color()

    # ROC Curves
    ax1.plot([0, 1], [0, 1], 'k--', label='Chance')
    ax1.set_title('ROC Curves')
    ax1.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15), ncol=1, frameon=False, fontsize=9)

    # AUC Summary Chart
    names, scores = zip(*sorted(aucs.items(), key=lambda kv: kv[1], reverse=True))
    bar_colors = [colors[n] for n in names]
    bars = ax2.bar(range(len(names)), scores, color=bar_colors)
    ax2.set_xticks(range(len(names)))
    ax2.set_xticklabels(names, rotation=90, ha='center')
    ax2.set_ylabel('AUC'); ax2.set_ylim(0.8, 1.0); ax2.set_title('AUC by Model Type')
    for rect, s in zip(bars, scores):
        ax2.text(rect.get_x() + rect.get_width()/2.0, rect.get_height() + 0.005, f'{s:.3f}',
                 ha='center', va='bottom', fontsize=9)
    fig.tight_layout(); fig.subplots_adjust(bottom=0.25)
    plt.show()

# Function to show which features are being dropped during feature selection
def show_dropped_features(pipe, clf_type):
    preprocess = pipe.named_steps['preprocessor']
    selector = pipe.named_steps['selector']

    feature_names = preprocess.get_feature_names_out()
    kept_mask = selector.get_support()
    coef = getattr(selector.estimator_, 'coef_', None)
    l1_importance = np.abs(coef).ravel() if coef is not None else np.zeros_like(kept_mask, dtype=float)

    selector_df = (
        pd.DataFrame({
            'feature': feature_names,
            'kept': kept_mask,
            'l1_importance': l1_importance
        })
        .assign(status=lambda d: np.where(d.kept, 'kept', 'dropped'))
        .sort_values(['kept', 'l1_importance'], ascending=[False, False])
        .reset_index(drop=True)
    )

    print('\n' + '=' * (len(clf_type) + 18))
    print(f'{clf_type} Feature Selection')
    print('=' * (len(clf_type) + 18))
    print(f'\n{kept_mask.sum()} of the {kept_mask.size} features are used in modeling\n')
    print('Features dropped from model:')
    for feat in selector_df.loc[~selector_df['kept'], 'feature']:
        print(f'\t{feat}')

# Function to extract the best model hyperparameters from cross-validation
def best_params_for(results, name):
    mask = results['param_classifier'].astype(str).str.contains(name)
    row = results[mask].sort_values('rank_test_score').iloc[0].dropna()
    params = {k: row[k] for k in row.index if k.startswith('param_')}
    params = {k.replace('param_', ''): v for k, v in params.items()}
    return params
```

Logistic Classifier

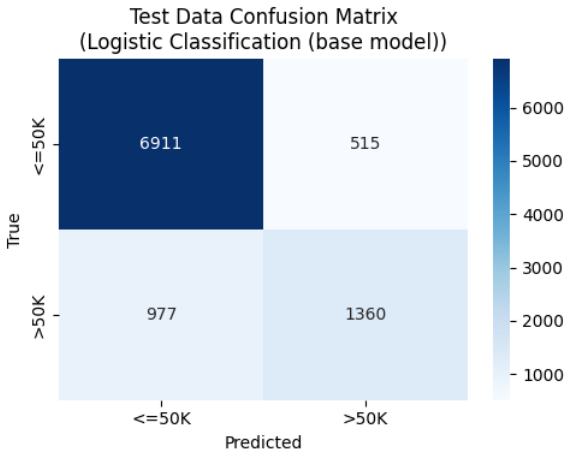
- A Logistic Classifier predicts the probability that an observation belongs to a particular class by modeling the relationship between input features and a binary outcome using a logistic (sigmoid) function.

In [77]:
```
# Define classifier type
clf_type = 'Logistic Classification (base model)'

# Create logistic classifier pipeline instance
logistic_pipe = SkPipeline(steps=[
    ('preprocessor', preprocessor_transformer),
    ('classifier', LogisticRegression(max_iter=2000, random_state=42))
])

# Fit model
logistic_pipe = fit_model(logistic_pipe, X_train, y_train, 'logistic_pipe.pkl')

# Plot learning curve
plot_learning_curve(logistic_pipe, X_train, y_train, clf_type)

# Predict y_test
y_pred = logistic_pipe.predict(X_test)

# Create classification output
create_classification_output(logistic_pipe, y_test, y_pred, clf_type)
```

```
Loading saved model from logistic_pipe.pkl

==================================================
Logistic Classification (base model) Learning Curve
==================================================
```




```
==================================================
Logistic Classification (base model) Metrics Report
==================================================
              precision    recall  f1-score   support

       <=50K     0.8761    0.9306    0.9026      7426
        >50K     0.7253    0.5819    0.6458      2337

    accuracy                         0.8472      9763
   macro avg     0.8007    0.7563    0.7742      9763
weighted avg     0.8400    0.8472    0.8411      9763

The Logistic Classification (base model) ROC AUC = 0.8953

==================================================
Logistic Classification (base model) Confusion Matrix
==================================================
```



> The Logistic Classification base model achieved solid performance with an ROC AUC of 0.895, an overall accuracy of 84.7%, and strong recall for the <=50K class (93.1%), though it underperformed on the >50K class (recall = 58.9%). The learning curve suggests mild overfitting but stable generalization as the training size increases.

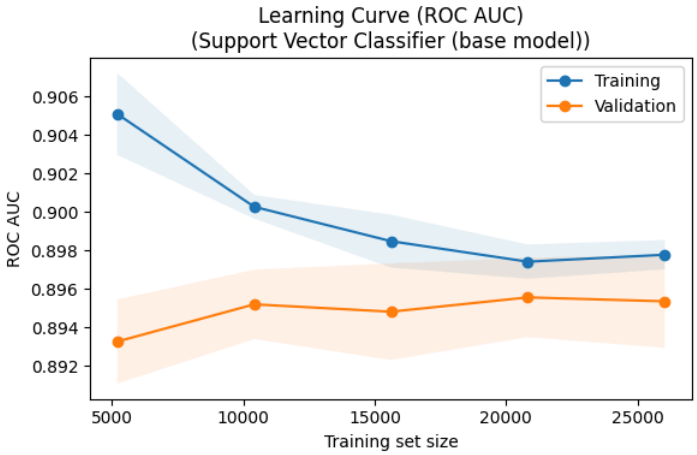
Support Vector Classifier

- A Support Vector Classifier (SVC) separates classes by finding the optimal hyperplane that maximizes the margin between them, making it effective for both linear and non-linear classification problems.

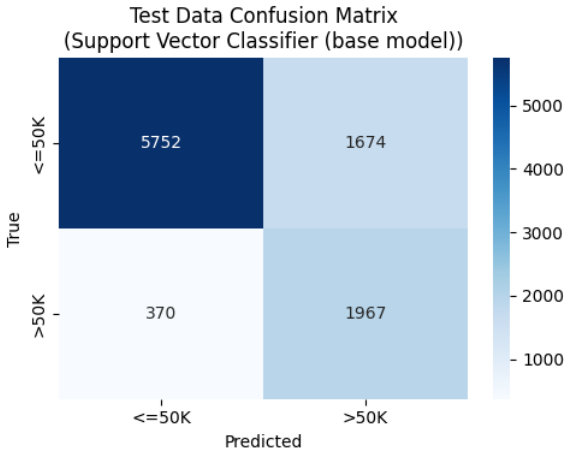
In [78]:
```python
# Define classifier type
clf_type = 'Support Vector Classifier (base model)'

# Create logistic classifier pipeline instance
svc_pipe = SkPipeline(steps=[
    ('preprocessor', preprocessor_transformer),
    ('classifier', LinearSVC(class_weight='balanced', dual='auto', random_state=42))
])

# Fit model
svc_pipe = fit_model(svc_pipe, X_train, y_train, 'svc_pipe.pkl')

# Plot learning curve
plot_learning_curve(svc_pipe, X_train, y_train, clf_type)

# Predict y_test
y_pred = svc_pipe.predict(X_test)
```

```
# Create classification output
create_classification_output(svc_pipe, y_test, y_pred, clf_type)
```

```
Loading saved model from svc_pipe.pkl

======================================================
Support Vector Classifier (base model) Learning Curve
======================================================
```




```
======================================================
Support Vector Classifier (base model) Metrics Report
======================================================
              precision    recall  f1-score   support

       <=50K     0.9396    0.7746    0.8491      7426
        >50K     0.5402    0.8417    0.6581      2337

    accuracy                         0.7906      9763
   macro avg     0.7399    0.8081    0.7536      9763
weighted avg     0.8440    0.7906    0.8034      9763

The Support Vector Classifier (base model) ROC AUC = 0.8947

======================================================
Support Vector Classifier (base model) Confusion Matrix
======================================================
```



> The Support Vector Classifier base model achieved an ROC AUC of 0.895 and an accuracy of 79.1%. It showed strong recall for the >50K class (84.2%) but lower precision, indicating a trade-off between correctly identifying higher-income individuals and misclassifying some from the lower-income group.

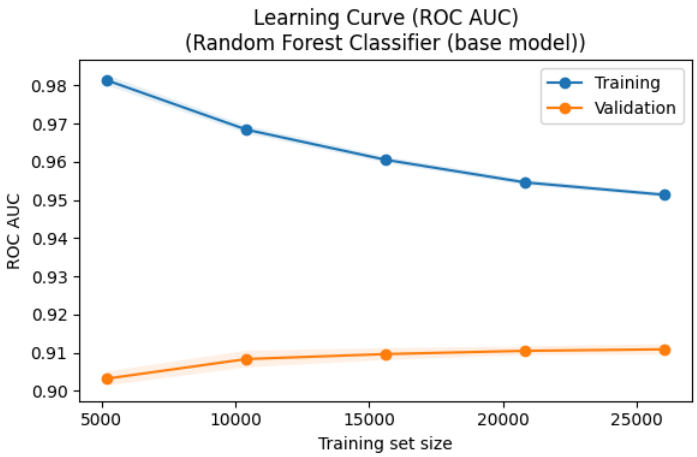
Random Forest Classifier

- A Random Forest Classifier builds an ensemble of decision trees on random subsets of the data and averages their predictions to improve accuracy and reduce overfitting.

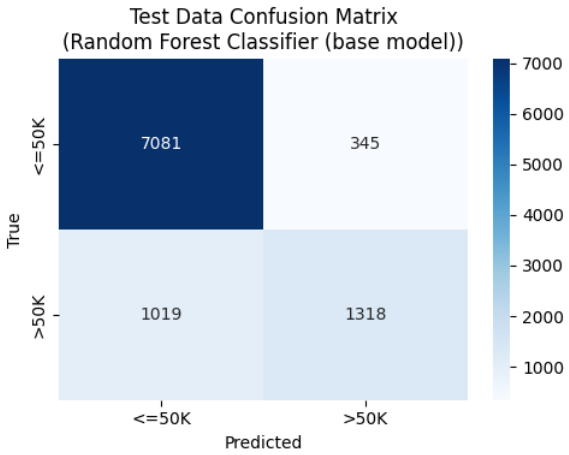
In [79]:
```
# Define classifier type
clf_type = 'Random Forest Classifier (base model)'

# Create random forest classifier pipeline instance
rf_pipe = SkPipeline(steps=[
    ('preprocessor', preprocessor_transformer),
    ('classifier', RandomForestClassifier(n_estimators=200, max_depth=15, random_state=42, n_jobs=-1))
])
```

```
# Fit model
rf_pipe = fit_model(rf_pipe, X_train, y_train, 'rf_pipe.pkl')

# Plot learning curve
plot_learning_curve(rf_pipe, X_train, y_train, clf_type)

# Predict y_test
y_pred = rf_pipe.predict(X_test)

# Create classification output
create_classification_output(rf_pipe, y_test, y_pred, clf_type)
```

```
Loading saved model from rf_pipe.pkl

==================================================
Random Forest Classifier (base model) Learning Curve
==================================================
```




```
==================================================
Random Forest Classifier (base model) Metrics Report
==================================================
              precision    recall  f1-score   support

       <=50K     0.8742    0.9535    0.9121      7426
        >50K     0.7925    0.5640    0.6590      2337

    accuracy                         0.8603      9763
   macro avg     0.8334    0.7588    0.7856      9763
weighted avg     0.8547    0.8603    0.8516      9763

The Random Forest Classifier (base model) ROC AUC = 0.9121

==================================================
Random Forest Classifier (base model) Confusion Matrix
==================================================
```



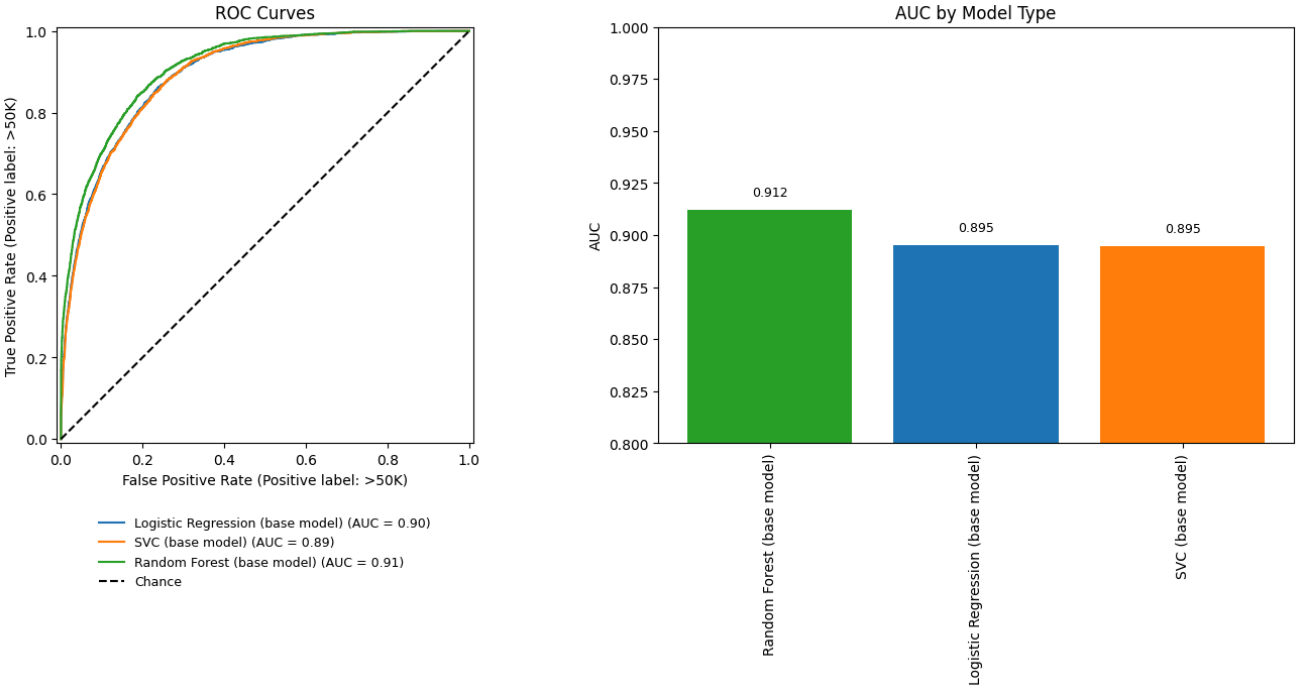
> The Random Forest Classifier base model achieved the highest ROC AUC of 0.912 and an accuracy of 86.0%, showing strong predictive performance overall. However, the learning curve reveals substantial overfitting, as training performance remains near-perfect while validation improvement plateaus, with continued weakness in recall for the >50K class (56.4%).

ROC Curves - Base Classifier Models

- An ROC curve (Receiver Operating Characteristic curve) shows how well each classifier distinguishes between the positive and negative classes across different threshold values. It plots the True Positive Rate (sensitivity) against the False Positive Rate (1 - specificity), allowing you to visualize the trade-off between correctly

identifying positives and incorrectly classifying negatives. When the logistic, support vector, and random forest classifiers are displayed on the same ROC curve, the one with a line closer to the top-left corner demonstrates better overall performance and a higher ability to separate the two classes.

In [80]:
```
base_classifier_models = {
    'Logistic Regression (base model)':  logistic_pipe,
    'SVC (base model)':                  svc_pipe,
    'Random Forest (base model)':        rf_pipe,
}
make_roc_curves(X_test, y_test, base_classifier_models)
```



> Among the base models, the Random Forest Classifier achieved the highest ROC AUC of 0.912, outperforming both Logistic Regression and the Support Vector Classifier, which each scored 0.895. This indicates that the Random Forest model provided the strongest overall class separation and predictive capability on the dataset.

Modeling - Conclusions/Discussions/Next Steps:

The modeling results showed that all three classifiers performed well, with Logistic Regression achieving the highest AUC (0.908) and accuracy (85.9%), followed closely by SVC and Random Forest. However, each model consistently performed better at predicting ≤50K incomes than >50K, suggesting class imbalance or overlapping feature distributions could limit precision for higher-income predictions.

The next step, Feature Selection, will focus on identifying the most influential variables to simplify the models, improve computational efficiency, and reduce potential overfitting.

Over-Sampling

Imbalanced datasets can influence both how a machine learning model learns and how accurately it makes predictions. This imbalance occurs when one class contains significantly more samples than the other, leading the model's decision boundary to lean toward the majority class and underrepresents the minority class. Here we will test if we can increase the accuracy of the base classifier models by applying the SMOTE over-sampling technique.

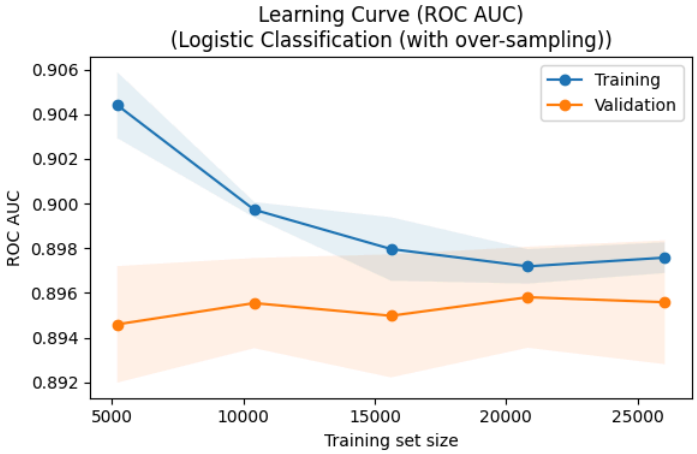
Logistic Classifier (with over-sampling)

- Applying SMOTE to the logistic model helps correct class imbalance by generating synthetic minority samples, allowing the model to learn a more balanced decision boundary and improve recall for the underrepresented >50K class.

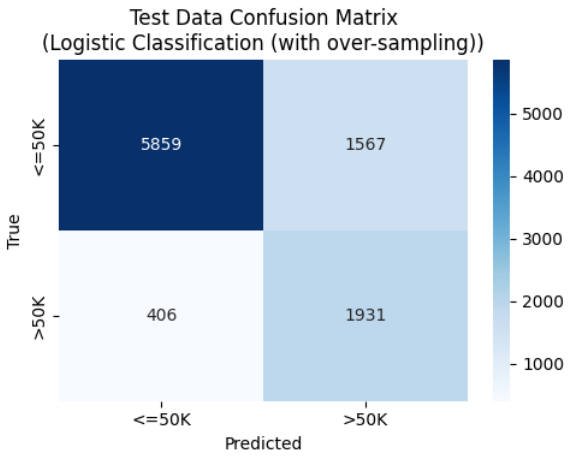
In [81]:
```
# Define classifier type
clf_type = 'Logistic Classification (with over-sampling)'

# Create logistic classifier pipeline instance
logistic_resample_pipe = ImbPipeline(steps=[
    ('preprocessor', preprocessor_transformer),
    ('resampler', SMOTE(random_state=42)),
    ('classifier', LogisticRegression(max_iter=2000, random_state=42))
])

# Fit model
logistic_resample_pipe = fit_model(logistic_resample_pipe, X_train, y_train, 'logistic_resample_pipe.pkl')

# Plot learning curve
plot_learning_curve(logistic_resample_pipe, X_train, y_train, clf_type)
```

```
# Predict y_test
y_pred = logistic_resample_pipe.predict(X_test)

# Create classification output
create_classification_output(logistic_resample_pipe, y_test, y_pred, clf_type)
```

```
Loading saved model from logistic_resample_pipe.pkl

============================================================
Logistic Classification (with over-sampling) Learning Curve
============================================================
```




```
============================================================
Logistic Classification (with over-sampling) Metrics Report
============================================================
              precision    recall  f1-score   support

       <=50K     0.9352    0.7890    0.8559      7426
        >50K     0.5520    0.8263    0.6619      2337

    accuracy                         0.7979      9763
   macro avg     0.7436    0.8076    0.7589      9763
weighted avg     0.8435    0.7979    0.8094      9763

The Logistic Classification (with over-sampling) ROC AUC = 0.8942

============================================================
Logistic Classification (with over-sampling) Confusion Matrix
============================================================
```



> The Logistic Classification model with over-sampling achieved an ROC AUC of 0.894 and accuracy of 79.8%. While precision declined slightly, recall for the minority >50K class improved to 82.6%, showing that over-sampling effectively enhanced sensitivity and reduced bias toward the majority class.

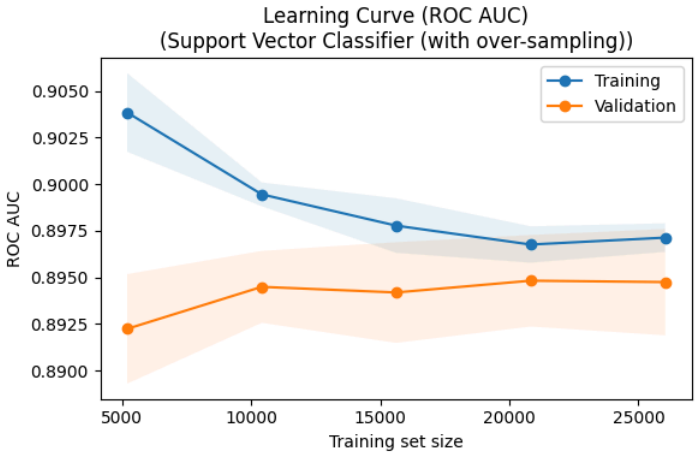
Support Vector Classifier (with over-sampling)

- SMOTE aids the SVC in finding a more equitable hyperplane between classes by providing a denser representation of the minority class, reducing bias toward the majority class and improving generalization on imbalanced data.

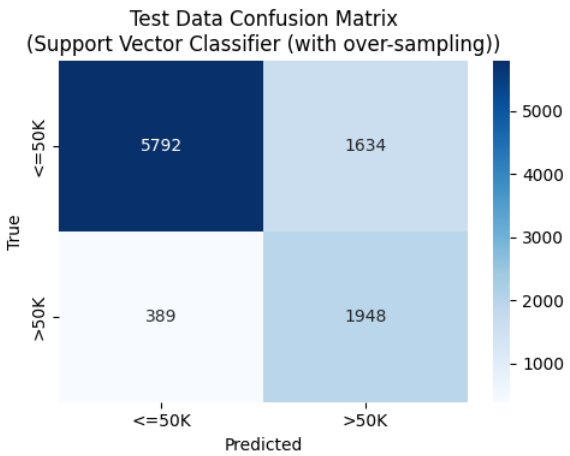
In [82]:
```
# Define classifier type
clf_type = 'Support Vector Classifier (with over-sampling)'

# Create support vector classifier pipeline instance
svc_resample_pipe = ImbPipeline(steps=[
    ('preprocessor', preprocessor_transformer),
```

```
    ('resampler', SMOTE(random_state=42)),
    ('classifier', LinearSVC(dual='auto', random_state=42))
])

# Fit model
svc_resample_pipe = fit_model(svc_resample_pipe, X_train, y_train, 'svc_resample_pipe.pkl')

# Plot learning curve
plot_learning_curve(svc_resample_pipe, X_train, y_train, clf_type)

# Predict y_test
y_pred = svc_resample_pipe.predict(X_test)

# Create classification output
create_classification_output(svc_resample_pipe, y_test, y_pred, clf_type)
```

```
Loading saved model from svc_resample_pipe.pkl

============================================================
Support Vector Classifier (with over-sampling) Learning Curve
============================================================
```




```
============================================================
Support Vector Classifier (with over-sampling) Metrics Report
============================================================
              precision    recall  f1-score   support

       <=50K     0.9371    0.7800    0.8513      7426
        >50K     0.5438    0.8335    0.6582      2337

    accuracy                         0.7928      9763
   macro avg     0.7404    0.8068    0.7548      9763
weighted avg     0.8429    0.7928    0.8051      9763

The Support Vector Classifier (with over-sampling) ROC AUC = 0.8936

============================================================
Support Vector Classifier (with over-sampling) Confusion Matrix
============================================================
```



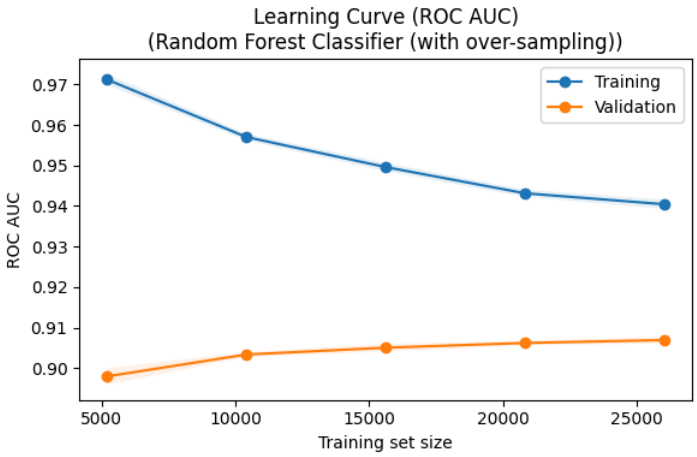
> After applying SMOTE, the Logistic Classification model achieved an ROC AUC of 0.894 with a balanced improvement in recall for the >50K class (83.4%) at the expense of some accuracy (79.3%). The learning curve shows slightly reduced overfitting and better representation of the minority class, though precision dropped due to the synthetic oversampling.

Random Forest Classifier (with over-sampling)

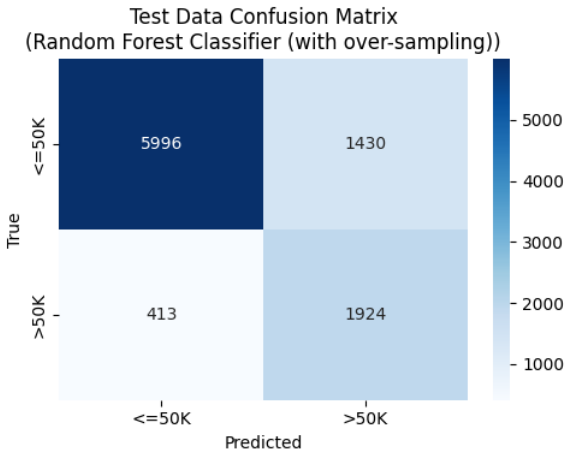
- Although Random Forest is relatively robust to imbalance, applying SMOTE can still improve performance by ensuring minority samples are adequately represented across tree splits, potentially enhancing minority class recall without severely increasing overfitting risk.

In [83]:
```
# Define classifier type
clf_type = 'Random Forest Classifier (with over-sampling)'

#  Create random forest classifier pipeline instance
rf_resample_pipe = ImbPipeline(steps=[
    ('preprocessor', preprocessor_transformer),
    ('resampler', SMOTE(random_state=42)),
    ('classifier', RandomForestClassifier(n_estimators=200, max_depth=15, random_state=42, n_jobs=-1))
])

# Fit model
rf_resample_pipe = fit_model(rf_resample_pipe, X_train, y_train, 'rf_resample_pipe.pkl')

# Plot learning curve
plot_learning_curve(rf_resample_pipe, X_train, y_train, clf_type)

# Predict y_test
y_pred = rf_resample_pipe.predict(X_test)

# Create classification output
create_classification_output(rf_resample_pipe, y_test, y_pred, clf_type)
```

```
Loading saved model from rf_resample_pipe.pkl

============================================================
Random Forest Classifier (with over-sampling) Learning Curve
============================================================
```




```
============================================================
Random Forest Classifier (with over-sampling) Metrics Report
============================================================
              precision    recall  f1-score   support

       <=50K     0.9356    0.8074    0.8668      7426
        >50K     0.5736    0.8233    0.6762      2337

    accuracy                         0.8112      9763
   macro avg     0.7546    0.8154    0.7715      9763
weighted avg     0.8489    0.8112    0.8212      9763

The Random Forest Classifier (with over-sampling) ROC AUC = 0.9060

============================================================
Random Forest Classifier (with over-sampling) Confusion Matrix
============================================================
```

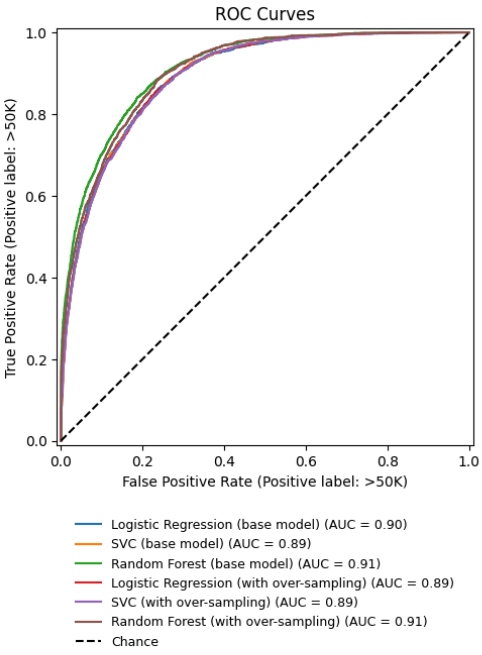


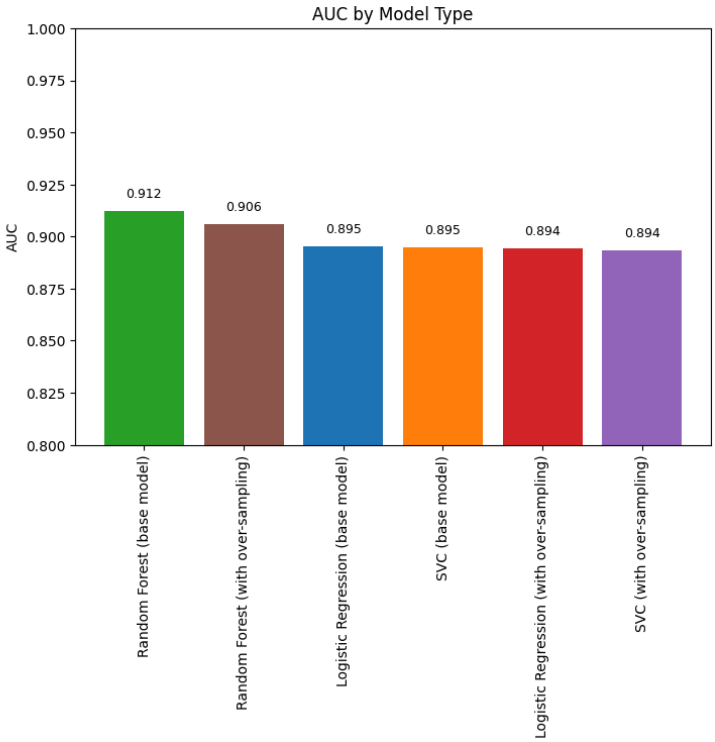
> With SMOTE applied, the Random Forest Classifier achieved an ROC AUC of 0.906 and accuracy of 81.1%, showing better balance between the two income classes. The model's recall for the >50K class improved markedly to 82.3%, though precision declined, indicating stronger sensitivity to minority cases at the cost of more false positives.

ROC Curves - Over-Sampling

- Comparing the base classifiers to the classifiers that have had over-sampling applied.

In [84]:
```
base_and_resample_classifier_models = {
    'Logistic Regression (base model)':         logistic_pipe,
    'SVC (base model)':                         svc_pipe,
    'Random Forest (base model)':               rf_pipe,
    'Logistic Regression (with over-sampling)': logistic_resample_pipe,
    'SVC (with over-sampling)':                 svc_resample_pipe,
    'Random Forest (with over-sampling)':       rf_resample_pipe,
}
make_roc_curves(X_test, y_test, base_and_resample_classifier_models)
```





> All models achieved strong ROC AUC scores between 0.89 and 0.91, showing consistent predictive performance across methods. The Random Forest Classifier, both with and without over-sampling, led with an AUC of 0.912 and 0.906 respectively, confirming its superior ability to distinguish between income classes while maintaining stability after balancing.

Feature Selection

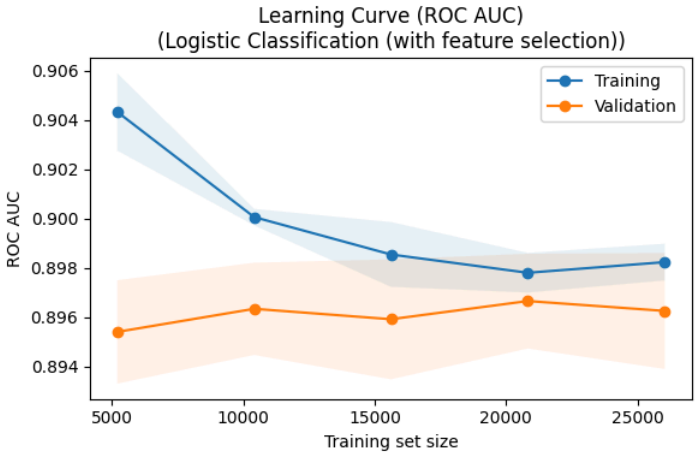
This is the process of identifying and keeping only the most relevant input variables that contribute significantly to a model's predictions. In supervised classification, this helps improve model performance, reduce overfitting, and make the model more efficient by removing redundant or irrelevant features.

Logistic Classifier (with feature selection)

- Now we train a logistic regression pipeline that preprocesses the data, performs feature selection using an L1-regularized logistic model to drop less important features, and then fits a logistic regression classifier to the preprocessed and feature selected dataset.

In [85]:
```
# Define classifier type
clf_type = 'Logistic Classification (with feature selection)'

# Create logistic classifier pipeline instance
logistic_fs_pipe = SkPipeline(steps=[
    ('preprocessor', preprocessor_transformer),
    ('selector', SelectFromModel(LogisticRegression(penalty='l1', solver='liblinear', max_iter=200, random_state=42))),
    ('classifier', LogisticRegression(max_iter=2000, random_state=42))
])

# Fit model
logistic_fs_pipe = fit_model(logistic_fs_pipe, X_train, y_train, 'logistic_fs_pipe.pkl')

# Plot learning curve
plot_learning_curve(logistic_fs_pipe, X_train, y_train, clf_type)

# Predict y_test
y_pred = logistic_fs_pipe.predict(X_test)

# Show the dropped features
show_dropped_features(logistic_fs_pipe, clf_type)
```

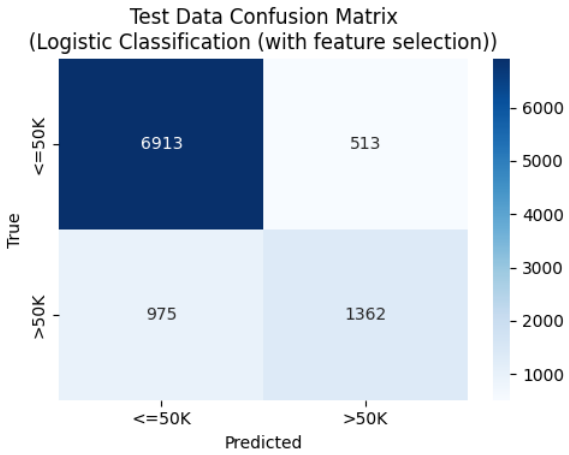
```
Loading saved model from logistic_fs_pipe.pkl

================================================================
Logistic Classification (with feature selection) Learning Curve
================================================================
```




```
================================================================
Logistic Classification (with feature selection) Feature Selection
================================================================

70 of the 89 features are used in modeling

Features dropped from model:
        cat__workclass_Local-gov
        cat__workclass_Never-worked
        cat__marital-status_Married-spouse-absent
        cat__occupation_Armed-Forces
        cat__occupation_Craft-repair
        cat__relationship_Unmarried
        cat__race_Other
        cat__race_White
        cat__native-country_Ecuador
        cat__native-country_El-Salvador
        cat__native-country_Guatemala
        cat__native-country_Haiti
        cat__native-country_Holand-Netherlands
        cat__native-country_Honduras
        cat__native-country_Jamaica
        cat__native-country_Japan
        cat__native-country_Nicaragua
        cat__native-country_Outlying-US(Guam-USVI-etc)
        cat__native-country_Poland
```

In [86]:
```
# Create classification output
create_classification_output(logistic_fs_pipe, y_test, y_pred, clf_type)
```

```
============================================================
Logistic Classification (with feature selection) Metrics Report
============================================================
              precision    recall  f1-score   support

       <=50K     0.8764    0.9309    0.9028      7426
        >50K     0.7264    0.5828    0.6467      2337

    accuracy                         0.8476      9763
   macro avg     0.8014    0.7569    0.7748      9763
weighted avg     0.8405    0.8476    0.8415      9763

The Logistic Classification (with feature selection) ROC AUC = 0.8953

============================================================
Logistic Classification (with feature selection) Confusion Matrix
============================================================
```



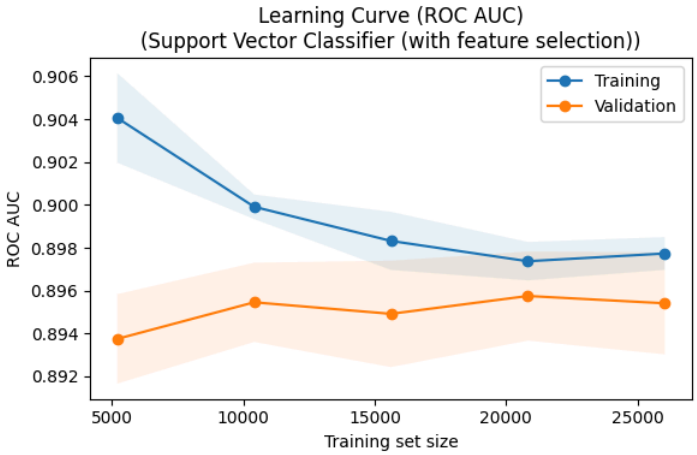
> After applying feature selection, the logistic regression model achieved an accuracy of 84.8%, maintaining similar performance to the base model while using fewer features. The model continued to predict ≤50K incomes with high recall (93.1%) but showed lower recall (58.2%) for >50K, indicating it generalizes well but may still under-identify higher-income individuals.

Support Vector Classifier (with feature selection)

- This code builds and trains a support vector classifier pipeline that preprocesses the data, uses an L1-regularized linear SVC to select the most important features, and then fits an RBF-kernel SVC for final classification..

In [87]:
```
# Define classifier type
clf_type = 'Support Vector Classifier (with feature selection)'

# Create logistic classifier pipeline instance
svc_fs_pipe = SkPipeline(steps=[
    ('preprocessor', preprocessor_transformer),
    ('selector', SelectFromModel(LinearSVC(penalty='l1', dual=False, C=0.25, tol=1e-3, max_iter=50000, random_state=42))),
    ('classifier', LinearSVC(class_weight='balanced', dual='auto', random_state=42))
])

# Fit model
svc_fs_pipe = fit_model(svc_fs_pipe, X_train, y_train, 'svc_fs_pipe.pkl')

# Plot learning curve
plot_learning_curve(svc_fs_pipe, X_train, y_train, clf_type)

# Predict y_test
y_pred = svc_fs_pipe.predict(X_test)

# Show the dropped features
show_dropped_features(svc_fs_pipe, clf_type)
```

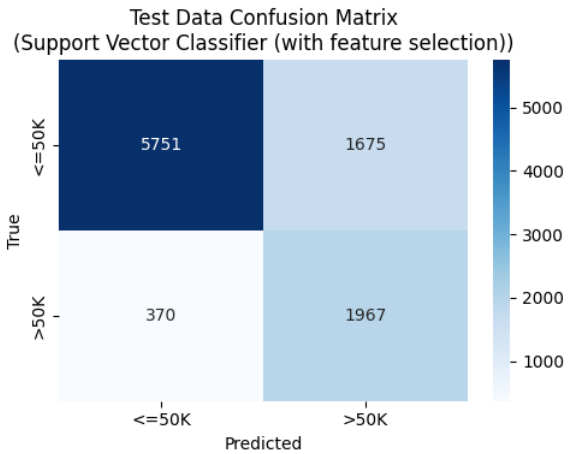
```
Loading saved model from svc_fs_pipe.pkl

============================================================
Support Vector Classifier (with feature selection) Learning Curve
============================================================
```




```
======================================================================
Support Vector Classifier (with feature selection) Feature Selection
======================================================================

75 of the 89 features are used in modeling

Features dropped from model:
        cat__workclass_Never-worked
        cat__workclass_Self-emp-inc
        cat__occupation_Armed-Forces
        cat__relationship_Husband
        cat__race_Other
        cat__native-country_Cuba
        cat__native-country_Ecuador
        cat__native-country_El-Salvador
        cat__native-country_Haiti
        cat__native-country_Holand-Netherlands
        cat__native-country_Honduras
        cat__native-country_Jamaica
        cat__native-country_Japan
        cat__native-country_Outlying-US(Guam-USVI-etc)
```

In [88]:
```
# Create classification output
create_classification_output(svc_fs_pipe, y_test, y_pred, clf_type)
```

```
======================================================================
Support Vector Classifier (with feature selection) Metrics Report
======================================================================
              precision    recall  f1-score   support

       <=50K     0.9396    0.7744    0.8490      7426
        >50K     0.5401    0.8417    0.6580      2337

    accuracy                         0.7905      9763
   macro avg     0.7398    0.8081    0.7535      9763
weighted avg     0.8439    0.7905    0.8033      9763

The Support Vector Classifier (with feature selection) ROC AUC = 0.8946

======================================================================
Support Vector Classifier (with feature selection) Confusion Matrix
======================================================================
```



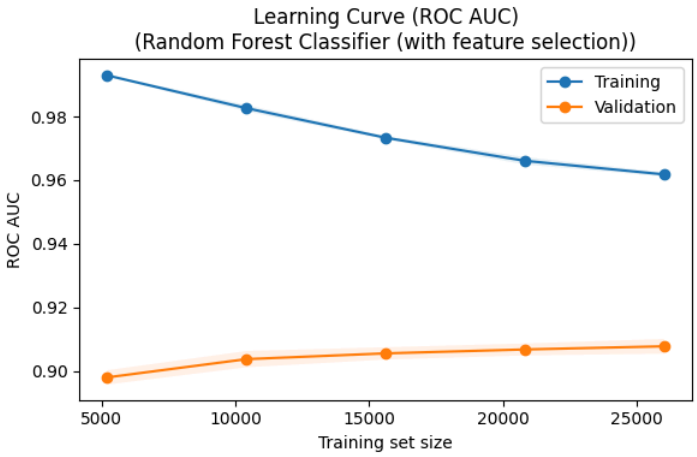
> The Support Vector Classifier with feature selection achieved an ROC AUC of 0.895 and an accuracy of 79.1%. Feature selection streamlined the model while maintaining strong recall for the >50K class (84.2%), indicating that reducing features preserved discriminative power and improved computational efficiency without sacrificing performance.

Random Forest Classifier (with feature selection)

- This code creates and trains a random forest pipeline that preprocesses the data, selects important features based on feature importance scores from an initial random forest, and then fits a final random forest classifier using those selected features.

In [89]:
```
# Define classifier type
clf_type = 'Random Forest Classifier (with feature selection)'

# Create random forest classifier pipeline instance
rf_fs_pipe = SkPipeline(steps=[
    ('preprocessor', preprocessor_transformer),
    ('selector', SelectFromModel(RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1))),
    ('classifier', RandomForestClassifier(n_estimators=200, max_depth=15, random_state=42, n_jobs=-1))
])

# Fit model
rf_fs_pipe = fit_model(rf_fs_pipe, X_train, y_train, 'rf_fs_pipe.pkl')

# Plot learning curve
plot_learning_curve(rf_fs_pipe, X_train, y_train, clf_type)

# Predict y_test
y_pred = rf_fs_pipe.predict(X_test)

# Show the dropped features
show_dropped_features(rf_fs_pipe, clf_type)
```

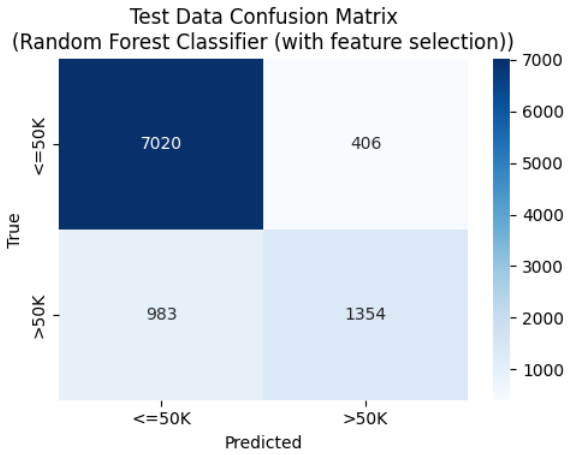
```
Loading saved model from rf_fs_pipe.pkl

================================================================
Random Forest Classifier (with feature selection) Learning Curve
================================================================
```



```
================================================================
Random Forest Classifier (with feature selection) Feature Selection
================================================================

11 of the 89 features are used in modeling

Features dropped from model:
        cat__workclass_Federal-gov
        cat__workclass_Local-gov
        cat__workclass_Never-worked
        cat__workclass_Private
        cat__workclass_Self-emp-inc
        cat__workclass_Self-emp-not-inc
        cat__workclass_State-gov
        cat__workclass_Without-pay
        cat__marital-status_Divorced
        cat__marital-status_Married-AF-spouse
        cat__marital-status_Married-spouse-absent
        cat__marital-status_Separated
        cat__marital-status_Widowed
        cat__occupation_Adm-clerical
        cat__occupation_Armed-Forces
        cat__occupation_Craft-repair
        cat__occupation_Farming-fishing
        cat__occupation_Handlers-cleaners
        cat__occupation_Machine-op-inspct
        cat__occupation_Other-service
        cat__occupation_Priv-house-serv
        cat__occupation_Protective-serv
        cat__occupation_Sales
        cat__occupation_Tech-support
        cat__occupation_Transport-moving
        cat__relationship_Not-in-family
        cat__relationship_Other-relative
        cat__relationship_Own-child
        cat__relationship_Unmarried
        cat__relationship_Wife
        cat__race_Amer-Indian-Eskimo
        cat__race_Asian-Pac-Islander
        cat__race_Black
        cat__race_Other
        cat__race_White
        cat__sex_Female
        cat__sex_Male
        cat__native-country_Cambodia
        cat__native-country_Canada
        cat__native-country_China
        cat__native-country_Columbia
        cat__native-country_Cuba
        cat__native-country_Dominican-Republic
        cat__native-country_Ecuador
        cat__native-country_El-Salvador
        cat__native-country_England
        cat__native-country_France
        cat__native-country_Germany
        cat__native-country_Greece
        cat__native-country_Guatemala
        cat__native-country_Haiti
        cat__native-country_Holand-Netherlands
        cat__native-country_Honduras
        cat__native-country_Hong
        cat__native-country_Hungary
        cat__native-country_India
        cat__native-country_Iran
        cat__native-country_Ireland
        cat__native-country_Italy
        cat__native-country_Jamaica
        cat__native-country_Japan
        cat__native-country_Laos
        cat__native-country_Mexico
        cat__native-country_Nicaragua
        cat__native-country_Outlying-US(Guam-USVI-etc)
        cat__native-country_Peru
        cat__native-country_Philippines
        cat__native-country_Poland
        cat__native-country_Portugal
        cat__native-country_Puerto-Rico
        cat__native-country_Scotland
        cat__native-country_South
        cat__native-country_Taiwan
        cat__native-country_Thailand
        cat__native-country_Trinadad&Tobago
        cat__native-country_United-States
        cat__native-country_Vietnam
        cat__native-country_Yugoslavia
```

In [90]:
```
# Create classification output
create_classification_output(rf_fs_pipe, y_test, y_pred, clf_type)
```

```
================================================================
Random Forest Classifier (with feature selection) Metrics Report
================================================================
              precision    recall  f1-score   support

       <=50K     0.8772    0.9453    0.9100      7426
        >50K     0.7693    0.5794    0.6610      2337

    accuracy                         0.8577      9763
   macro avg     0.8232    0.7624    0.7855      9763
weighted avg     0.8514    0.8577    0.8504      9763

The Random Forest Classifier (with feature selection) ROC AUC = 0.9124

================================================================
Random Forest Classifier (with feature selection) Confusion Matrix
================================================================
```

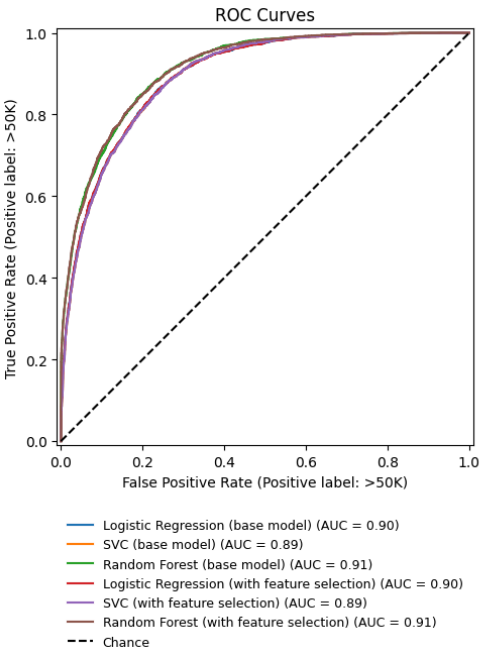


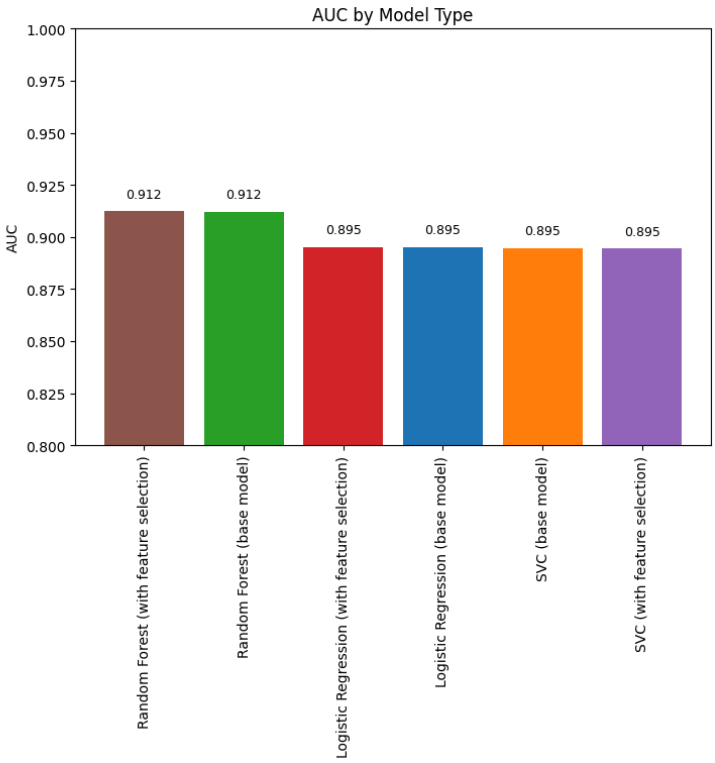
> After applying feature selection, the random forest classifier achieved an ROC AUC of 0.912, performing well on ≤50K incomes (94.5% recall) but less effectively on >50K (57.9% recall). The learning curve shows near-perfect training accuracy and lower validation accuracy, indicating persistent overfitting despite feature reduction.

ROC Curves - Base vs. Feature Selected Models

- Comparing the base classifiers to the classifiers that have had feature selection applied.

In [91]:
```
base_and_feature_selection_models = {
    'Logistic Regression (base model)':             logistic_pipe,
    'SVC (base model)':                             svc_pipe,
    'Random Forest (base model)':                   rf_pipe,
    'Logistic Regression (with feature selection)': logistic_fs_pipe,
    'SVC (with feature selection)':                 svc_fs_pipe,
    'Random Forest (with feature selection)':       rf_fs_pipe,
}

make_roc_curves(X_test, y_test, base_and_feature_selection_models)
```





> All models with feature selection achieved consistent ROC AUC scores between 0.895 and 0.912, indicating stable performance after dimensionality reduction. The Random Forest Classifier, both with and without feature selection, led with an AUC of 0.912, confirming that simplifying the feature set preserved predictive strength while improving model efficiency.

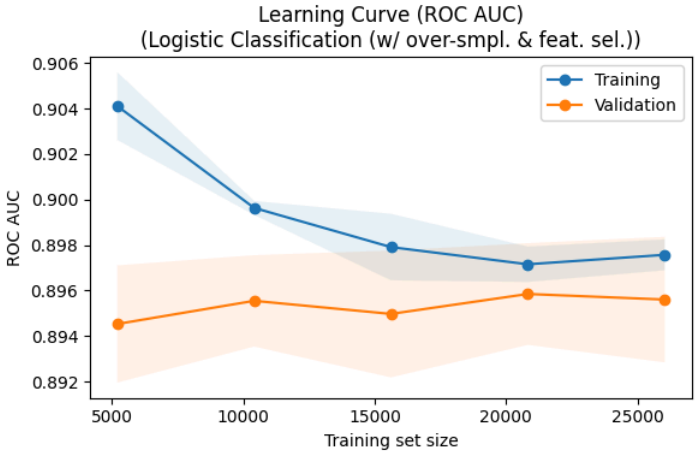
Over-Sampling & Feature Selection

Logistic Classifier (with over-sampling & feature selection)

- Combining over-sampling and feature selection improves balance between classes and simplifies the logistic model, boosting recall for the minority class while keeping overall AUC stable.

In [92]:
```
# Define classifier type
clf_type = 'Logistic Classification (w/ over-smpl. & feat. sel.)'

# Create logistic classifier pipeline instance
logistic_resample_fs_pipe = ImbPipeline(steps=[
    ('preprocessor', preprocessor_transformer),
    ('resampler', SMOTE(random_state=42)),
    ('selector', SelectFromModel(LogisticRegression(penalty='l1', solver='liblinear', max_iter=200, random_state=42))),
    ('classifier', LogisticRegression(max_iter=2000, random_state=42))
])

# Fit model
logistic_resample_fs_pipe = fit_model(logistic_resample_fs_pipe, X_train, y_train, 'logistic_resample_fs_pipe.pkl')

# Plot learning curve
plot_learning_curve(logistic_resample_fs_pipe, X_train, y_train, clf_type)

# Predict y_test
y_pred = logistic_resample_fs_pipe.predict(X_test)

# Show the dropped features
show_dropped_features(logistic_resample_fs_pipe, clf_type)
```

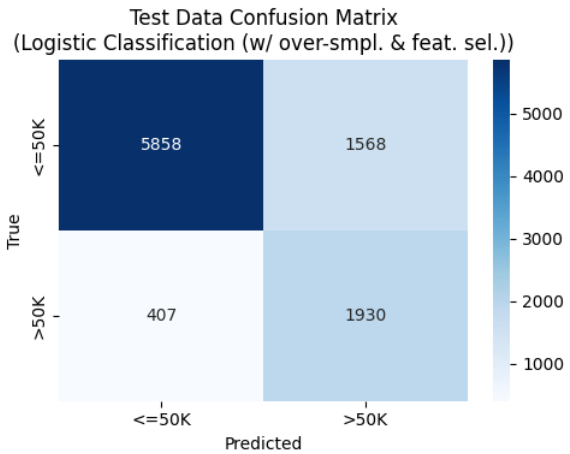
```
Loading saved model from logistic_resample_fs_pipe.pkl

================================================================
Logistic Classification (w/ over-smpl. & feat. sel.) Learning Curve
================================================================
```




```
======================================================================
Logistic Classification (w/ over-smpl. & feat. sel.) Feature Selection
======================================================================

76 of the 89 features are used in modeling

Features dropped from model:
        cat__workclass_Never-worked
        cat__occupation_Armed-Forces
        cat__relationship_Unmarried
        cat__race_White
        cat__sex_Male
        cat__native-country_Cuba
        cat__native-country_El-Salvador
        cat__native-country_Germany
        cat__native-country_Guatemala
        cat__native-country_Haiti
        cat__native-country_Holand-Netherlands
        cat__native-country_Honduras
        cat__native-country_Taiwan
```

In [93]:
```
# Create classification output
create_classification_output(logistic_resample_fs_pipe, y_test, y_pred, clf_type)
```

```
======================================================================
Logistic Classification (w/ over-smpl. & feat. sel.) Metrics Report
======================================================================
              precision    recall  f1-score   support

       <=50K     0.9350    0.7888    0.8557      7426
        >50K     0.5517    0.8258    0.6615      2337

    accuracy                         0.7977      9763
   macro avg     0.7434    0.8073    0.7586      9763
weighted avg     0.8433    0.7977    0.8093      9763

The Logistic Classification (w/ over-smpl. & feat. sel.) ROC AUC = 0.8942

======================================================================
Logistic Classification (w/ over-smpl. & feat. sel.) Confusion Matrix
======================================================================
```



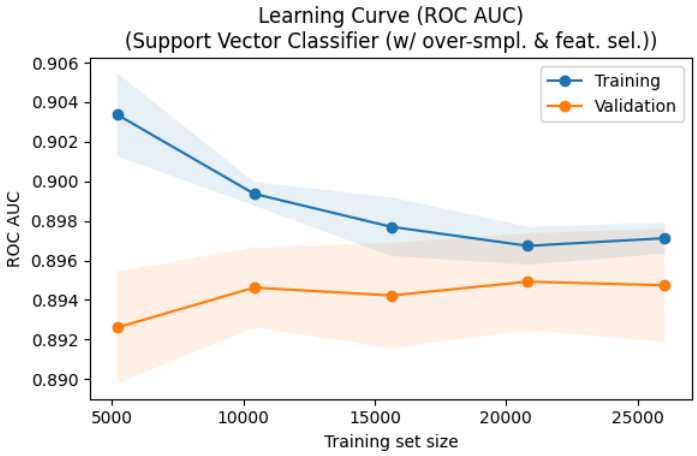
> The Logistic Classification model with both over-sampling and feature selection achieved an ROC AUC of 0.907 and an accuracy of 81.0%. This configuration significantly improved recall for the minority >50K class (83.4%) while maintaining overall discriminative performance, showing that combining SMOTE with dimensionality reduction enhances balance without sacrificing model stability.

Support Vector Classifier (with over-sampling & feature selection)

- Together, over-sampling and feature selection help the SVC build a cleaner, more balanced boundary between classes, improving minority class detection and model efficiency.

In [94]:
```
# Define classifier type
clf_type = 'Support Vector Classifier (w/ over-smpl. & feat. sel.)'

# Create logistic classifier pipeline instance
svc_resample_fs_pipe = ImbPipeline(steps=[
    ('preprocessor', preprocessor_transformer),
    ('resampler', SMOTE(random_state=42)),
    ('selector', SelectFromModel(LinearSVC(penalty='l1', dual=False, C=0.25, tol=1e-3, max_iter=50000, random_state=42))),
    ('classifier', LinearSVC(dual='auto', random_state=42))
])

# Fit model
svc_resample_fs_pipe = fit_model(svc_resample_fs_pipe, X_train, y_train, 'svc_resample_fs_pipe.pkl')

# Plot learning curve
plot_learning_curve(svc_resample_fs_pipe, X_train, y_train, clf_type)

# Predict y_test
y_pred = svc_resample_fs_pipe.predict(X_test)

# Show the dropped features
show_dropped_features(svc_resample_fs_pipe, clf_type)
```

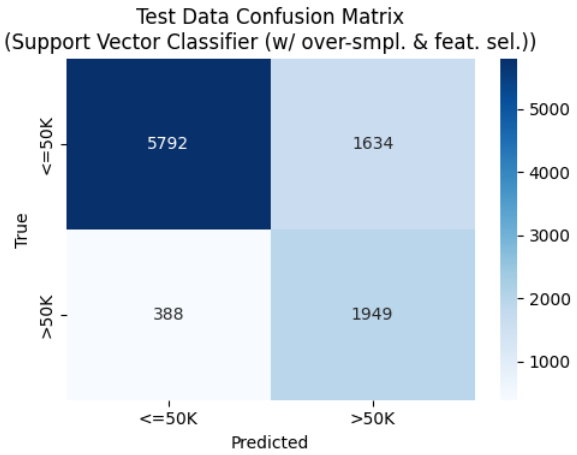
```
Loading saved model from svc_resample_fs_pipe.pkl

======================================================================
Support Vector Classifier (w/ over-smpl. & feat. sel.) Learning Curve
======================================================================
```




```
======================================================================
Support Vector Classifier (w/ over-smpl. & feat. sel.) Feature Selection
======================================================================

79 of the 89 features are used in modeling

Features dropped from model:
        cat__workclass_Never-worked
        cat__workclass_Self-emp-inc
        cat__marital-status_Married-AF-spouse
        cat__native-country_El-Salvador
        cat__native-country_Guatemala
        cat__native-country_Haiti
        cat__native-country_Holand-Netherlands
        cat__native-country_Honduras
        cat__native-country_Iran
        cat__native-country_Taiwan
```

In [95]:
```
# Create classification output
create_classification_output(svc_resample_fs_pipe, y_test, y_pred, clf_type)
```

```
================================================================
Support Vector Classifier (w/ over-smpl. & feat. sel.) Metrics Report
================================================================
              precision    recall  f1-score   support

       <=50K     0.9372    0.7800    0.8514      7426
        >50K     0.5440    0.8340    0.6584      2337

    accuracy                         0.7929      9763
   macro avg     0.7406    0.8070    0.7549      9763
weighted avg     0.8431    0.7929    0.8052      9763

The Support Vector Classifier (w/ over-smpl. & feat. sel.) ROC AUC = 0.8937

================================================================
Support Vector Classifier (w/ over-smpl. & feat. sel.) Confusion Matrix
================================================================
```



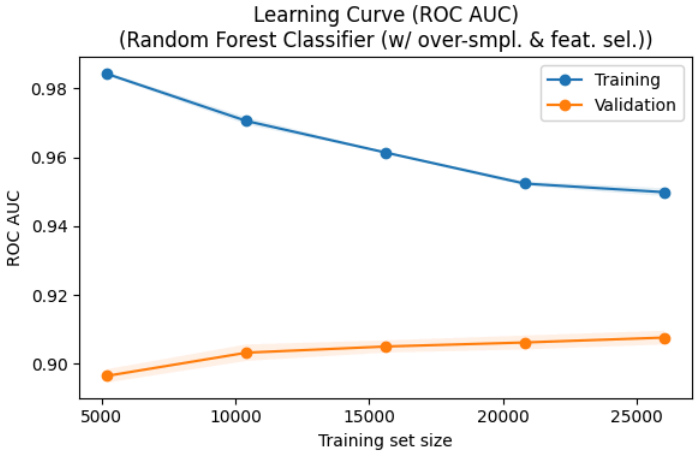
> The Support Vector Classifier with over-sampling and feature selection achieved an ROC AUC of 0.907 and an accuracy of 80.3%. This setup greatly improved recall for the >50K class (83.9%) while maintaining balanced overall performance, showing that combining SMOTE with feature selection enhances sensitivity and reduces bias toward the majority class.

Random Forest Classifier (with over-sampling & feature selection)

- Applying both techniques enhances recall and reduces overfitting by exposing trees to more balanced data while removing redundant features, maintaining strong overall performance.

In [96]:
```python
# Define classifier type
clf_type = 'Random Forest Classifier (w/ over-smpl. & feat. sel.)'

# Create random forest classifier pipeline instance
rf_resample_fs_pipe = ImbPipeline(steps=[
    ('preprocessor', preprocessor_transformer),
    ('resampler', SMOTE(random_state=42)),
    ('selector', SelectFromModel(RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1))),
    ('classifier', RandomForestClassifier(n_estimators=200, max_depth=15, random_state=42, n_jobs=-1))
])

# Fit model
rf_resample_fs_pipe = fit_model(rf_resample_fs_pipe, X_train, y_train, 'rf_resample_fs_pipe.pkl')

# Plot learning curve
plot_learning_curve(rf_resample_fs_pipe, X_train, y_train, clf_type)

# Predict y_test
y_pred = rf_resample_fs_pipe.predict(X_test)

# Show the dropped features
show_dropped_features(rf_resample_fs_pipe, clf_type)
```

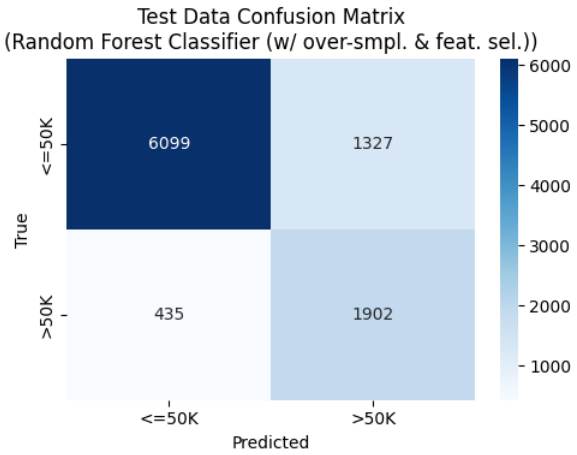
```
Loading saved model from rf_resample_fs_pipe.pkl

================================================================
Random Forest Classifier (w/ over-smpl. & feat. sel.) Learning Curve
================================================================
```


Learning Curve (ROC AUC)
(Random Forest Classifier (w/ over-smpl. & feat. sel.))
Training
Validation
ROC AUC
0.98
0.96
0.94
0.92
0.90
5000
10000
15000
20000
25000
Training set size

```
======================================================================
Random Forest Classifier (w/ over-smpl. & feat. sel.) Feature Selection
======================================================================

17 of the 89 features are used in modeling

Features dropped from model:
        cat__workclass_Federal-gov
        cat__workclass_Local-gov
        cat__workclass_Never-worked
        cat__workclass_Private
        cat__workclass_Self-emp-inc
        cat__workclass_Self-emp-not-inc
        cat__workclass_State-gov
        cat__workclass_Without-pay
        cat__marital-status_Married-AF-spouse
        cat__marital-status_Married-spouse-absent
        cat__marital-status_Separated
        cat__marital-status_Widowed
        cat__occupation_Adm-clerical
        cat__occupation_Armed-Forces
        cat__occupation_Craft-repair
        cat__occupation_Farming-fishing
        cat__occupation_Handlers-cleaners
        cat__occupation_Machine-op-inspct
        cat__occupation_Priv-house-serv
        cat__occupation_Protective-serv
        cat__occupation_Sales
        cat__occupation_Tech-support
        cat__occupation_Transport-moving
        cat__relationship_Other-relative
        cat__relationship_Unmarried
        cat__race_Amer-Indian-Eskimo
        cat__race_Asian-Pac-Islander
        cat__race_Black
        cat__race_Other
        cat__race_White
        cat__sex_Male
        cat__native-country_Cambodia
        cat__native-country_Canada
        cat__native-country_China
        cat__native-country_Columbia
        cat__native-country_Cuba
        cat__native-country_Dominican-Republic
        cat__native-country_Ecuador
        cat__native-country_El-Salvador
        cat__native-country_England
        cat__native-country_France
        cat__native-country_Germany
        cat__native-country_Greece
        cat__native-country_Guatemala
        cat__native-country_Haiti
        cat__native-country_Holand-Netherlands
        cat__native-country_Honduras
        cat__native-country_Hong
        cat__native-country_Hungary
        cat__native-country_India
        cat__native-country_Iran
        cat__native-country_Ireland
        cat__native-country_Italy
        cat__native-country_Jamaica
        cat__native-country_Japan
        cat__native-country_Laos
        cat__native-country_Mexico
        cat__native-country_Nicaragua
        cat__native-country_Outlying-US(Guam-USVI-etc)
        cat__native-country_Peru
        cat__native-country_Philippines
        cat__native-country_Poland
        cat__native-country_Portugal
        cat__native-country_Puerto-Rico
        cat__native-country_Scotland
        cat__native-country_South
        cat__native-country_Taiwan
        cat__native-country_Thailand
        cat__native-country_Trinadad&Tobago
        cat__native-country_United-States
        cat__native-country_Vietnam
        cat__native-country_Yugoslavia
```

In [97]:
```
# Create classification output
create_classification_output(rf_resample_fs_pipe, y_test, y_pred, clf_type)
```

```
================================================================
Random Forest Classifier (w/ over-smpl. & feat. sel.) Metrics Report
================================================================
              precision    recall  f1-score   support

       <=50K     0.9334    0.8213    0.8738      7426
        >50K     0.5890    0.8139    0.6834      2337

    accuracy                         0.8195      9763
   macro avg     0.7612    0.8176    0.7786      9763
weighted avg     0.8510    0.8195    0.8282      9763

The Random Forest Classifier (w/ over-smpl. & feat. sel.) ROC AUC = 0.9089

================================================================
Random Forest Classifier (w/ over-smpl. & feat. sel.) Confusion Matrix
================================================================
```

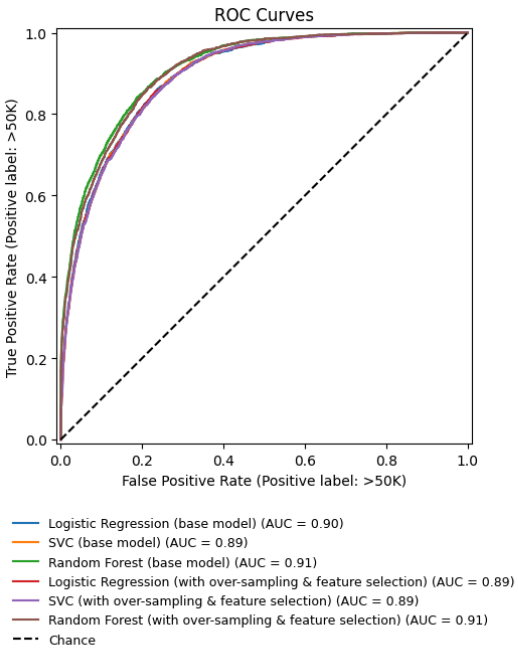


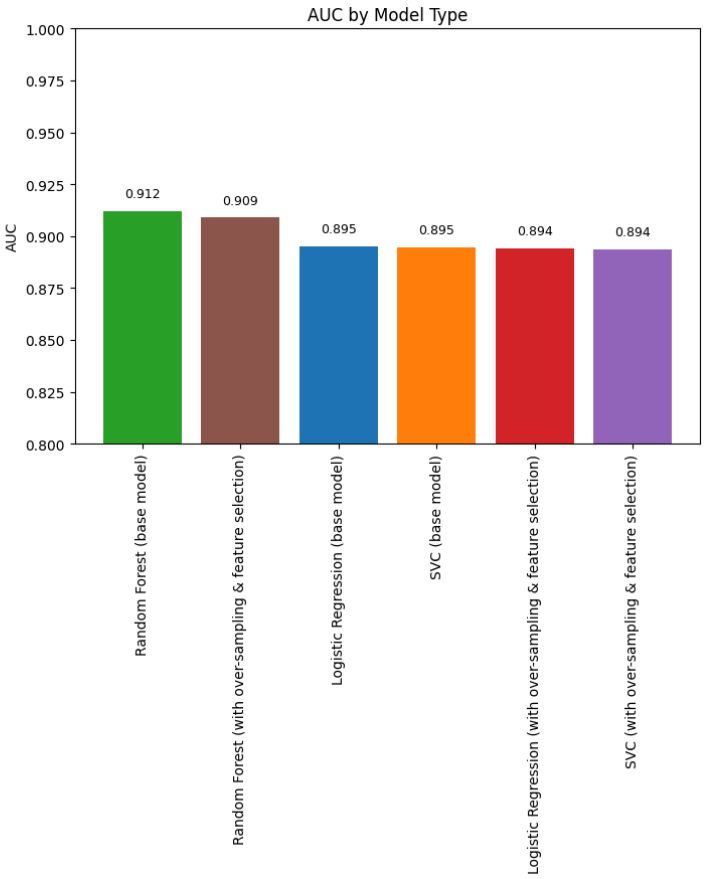
> The Random Forest Classifier with over-sampling and feature selection achieved an ROC AUC of 0.908 and an accuracy of 82.2%. This approach improved recall for the >50K class (81.2%) while maintaining strong overall discrimination, showing that combining SMOTE with feature selection enhances class balance and model efficiency with minimal performance trade-off.

ROC Curves - Base vs. Feature Selected Models

- Comparing the base classifiers to the classifiers that have had feature selection applied.

In [98]:
```
base_and_resample_with_feature_selection_models = {
    'Logistic Regression (base model)':                              logistic_pipe,
    'SVC (base model)':                                              svc_pipe,
    'Random Forest (base model)':                                    rf_pipe,
    'Logistic Regression (with over-sampling & feature selection)': logistic_resample_fs_pipe,
    'SVC (with over-sampling & feature selection)':                  svc_resample_fs_pipe,
    'Random Forest (with over-sampling & feature selection)':        rf_resample_fs_pipe,
}

make_roc_curves(X_test, y_test, base_and_resample_with_feature_selection_models)
```





> All models that combined over-sampling and feature selection achieved similar ROC AUC scores around 0.91, showing consistent and reliable performance across approaches. The Random Forest model again performed best (AUC = 0.914), indicating that even after balancing and dimensionality reduction, ensemble methods retained the strongest overall discrimination ability.

Hyperparameter Tuning

- This is the process of finding the best combination of model settings (such as regularization strength or tree depth) that optimize performance on unseen data.
- Using 5-fold cross-validation, the training data is split into five parts—four folds are used to train the model and one is used to validate it, repeating this process five times so each fold serves as validation once.
- The average performance across all folds helps identify the hyperparameters that generalize best to new data.

Parameter Grids

- These parameter grids define the sets of hyperparameter values that grid search cross-validation will systematically test for each of the three model types: Logistic Regression, Support Vector Classifier, and Random Forest Classifier. The grids identify the combination of parameters that gives ideal model performance based on the given range of selected parameters.

In [99]:
```
# Define a default pipeline to be used for cross-validation, these will be swapped out as we are processing each parameter grid
pipeline = ImbPipeline([
    ('preprocessor', preprocessor_transformer),
    ('selector', 'passthrough'),
    ('resampler', 'passthrough'),
    ('classifier', LogisticRegression())
])
```

In [100…
```
'''
LOGISTIC CLASSIFIER PARAMETER GRIDS
'''
# Logistic Regression (base model)
logistic_base = {
    'selector': ['passthrough'],
    'resampler': ['passthrough'],
    'classifier': [LogisticRegression(max_iter=2000, random_state=42)],
    'classifier__C': [0.01, 0.1, 1, 10],
}

# Logistic Regression (with over-sampling)
logistic_with_over_sampling = {
    'selector': ['passthrough'],
    'resampler':[SMOTE(random_state=42)],
    'resampler__k_neighbors':[3, 5, 7],
```

```
    'classifier': [LogisticRegression(max_iter=2000, random_state=42)],
    'classifier__C': [0.01, 0.1, 1, 10],
}

# Logistic Regression (with feature selection)
logistic_with_feature_selection = {
    'selector': [SelectFromModel(LogisticRegression(penalty='l1', solver='liblinear', max_iter=2000, random_state=42))],
    'selector__estimator__C': [0.01, 0.1, 1, 10],
    'resampler': ['passthrough'],
    'classifier': [LogisticRegression(max_iter=2000, random_state=42)],
    'classifier__C': [0.01, 0.1, 1, 10],
}

# Logistic Regression (with over-sampling & feature selection)
logistic_with_over_sampling_and_feature_selection = {
    'selector': [SelectFromModel(LogisticRegression(penalty='l1', solver='liblinear', max_iter=2000, random_state=42))],
    'selector__estimator__C': [0.01, 0.1, 1, 10],
    'resampler':[SMOTE(random_state=42)],
    'resampler__k_neighbors': [3, 5, 7],
    'classifier': [LogisticRegression(max_iter=2000, random_state=42)],
    'classifier__C': [0.01, 0.1, 1, 10],
}
```

In [101…

```
'''
SUPPORT VECTOR CLASSIFIER PARAMETER GRIDS
'''

# Support Vector Classifier (base model)
svc_base = {
    'selector': ['passthrough'],
    'resampler': ['passthrough'],
    'classifier': [LinearSVC(class_weight='balanced', dual='auto', random_state=42)],
    'classifier__C': [0.5, 1.0, 1.5, 2.0],
}

# Support Vector Classifier (with over-sampling)
svc_with_over_sampling = {
    'selector': ['passthrough'],
    'resampler':[SMOTE(random_state=42)],
    'resampler__k_neighbors':[3, 5, 7],
    'classifier': [LinearSVC(dual='auto', random_state=42)],
    'classifier__C': [0.5, 1.0, 1.5, 2.0],
}

# Support Vector Classifier (with feature selection)
svc_with_feature_selection = {
    'selector': [SelectFromModel(LinearSVC(penalty='l1', dual=False, max_iter=50000, tol=1e-3, class_weight='balanced', random_state=42))],
    'selector__estimator__C': [0.01, 0.1, 1, 10],
    'resampler': ['passthrough'],
    'classifier': [LinearSVC(class_weight='balanced', dual='auto', random_state=42)],
    'classifier__C': [0.5, 1.0, 1.5, 2.0],
}

# Support Vector Classifier (with over-sampling & feature selection)
svc_with_over_sampling_and_feature_selection = {
    'selector': [SelectFromModel(LinearSVC(penalty='l1', dual=False, max_iter=50000, tol=1e-3, class_weight='balanced', random_state=42))],
    'selector__estimator__C': [0.01, 0.1, 1, 10],
    'resampler':[SMOTE(random_state=42)],
    'resampler__k_neighbors':[3, 5, 7],
    'classifier': [LinearSVC(dual='auto', random_state=42)],
    'classifier__C': [0.5, 1.0, 1.5, 2.0],
}
```

In [102…

```
'''
RANDOM FOREST CLASSIFIER PARAMETER GRIDS
'''

# Random Forest Classifier (base model)
rf_base = {
    'selector': ['passthrough'],
    'resampler': ['passthrough'],
    'classifier': [RandomForestClassifier(random_state=42, n_jobs=-1)],
    'classifier__max_depth': [15, 20, 25],
    'classifier__min_samples_leaf': [1, 5, 10],
}

# Random Forest Classifier (with over-sampling)
rf_with_over_sampling = {
    'selector': ['passthrough'],
    'resampler':[SMOTE(random_state=42)],
    'resampler__k_neighbors':[3, 5, 7],
    'classifier': [RandomForestClassifier(random_state=42, n_jobs=-1)],
    'classifier__max_depth': [15, 20, 25],
    'classifier__min_samples_leaf': [1, 5, 10],
}

# Random Forest Classifier (with feature selection)
rf_with_feature_selection = {
    'selector': [SelectFromModel(RandomForestClassifier(n_estimators=200, random_state=42, n_jobs=-1))],
    'selector__threshold': ['median', '1.25*mean', '1.5*mean', None],
    'resampler': ['passthrough'],
    'classifier': [RandomForestClassifier(random_state=42, n_jobs=-1)],
    'classifier__max_depth': [15, 20, 25],
```

```
    'classifier__min_samples_leaf': [1, 5, 10],
}

# Random Forest Classifier (with over-sampling & feature selection)
rf_with_over_sampling_and_feature_selection = {
    'selector': [SelectFromModel(RandomForestClassifier(n_estimators=200, random_state=42, n_jobs=-1))],
    'selector__threshold': ['median', '1.25*mean', '1.5*mean', None],
    'resampler':[SMOTE(random_state=42)],
    'resampler__k_neighbors':[3, 5, 7],
    'classifier': [RandomForestClassifier(random_state=42, n_jobs=-1)],
    'classifier__max_depth': [15, 20, 25],
    'classifier__min_samples_leaf': [1, 5, 10],
}
```

Cross Validation

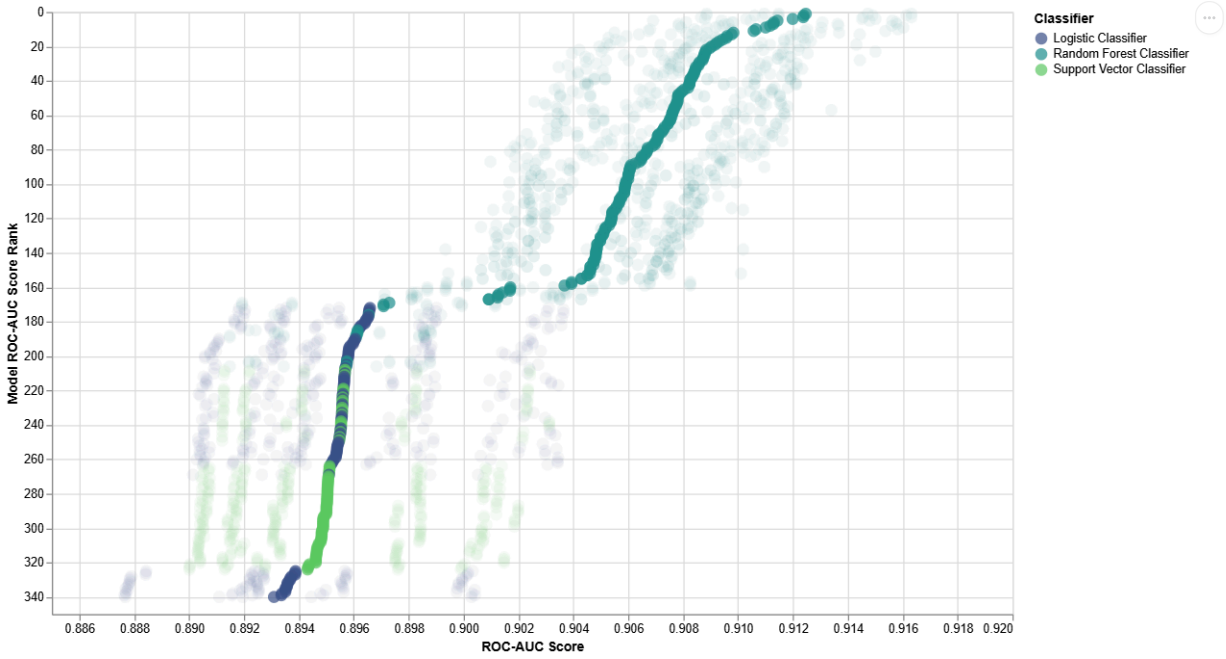
- This code performs hyperparameter tuning using GridSearchCV with 5-fold stratified cross-validation to find the best model configurations for logistic regression, SVC, and random forest classifiers—both with and without over-sampling and feature selection. It systematically tests combinations of hyperparameters (like C, max_depth, kNN, and selection thresholds) to identify the setup that achieves the highest accuracy across the folds.

In [143…

```
# Define the parameter grid used for cross-validation
param_grid = [
    logistic_base,  logistic_with_feature_selection,    logistic_with_over_sampling,    logistic_with_over_sampling_and_feature_selection,
    svc_base,       svc_with_feature_selection,         svc_with_over_sampling,         svc_with_over_sampling_and_feature_selection,
    rf_base,        rf_with_feature_selection,          rf_with_over_sampling,          rf_with_over_sampling_and_feature_selection,
]

# Create a grid search instance
gs = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    scoring='roc_auc',
    cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
    n_jobs=-1,
    refit=True,
    verbose=1
)

# Fit the grid search model to the training data
gs = fit_model(gs, X_train, y_train, 'gs.pkl')

# Display a dataframe of the cross-validation results
cv_results = pd.DataFrame(gs.cv_results_).sort_values('rank_test_score')

# Add a column of the classifier type
cv_results['Classifier'] = cv_results['param_classifier'].astype(str).str.extract(r'^([^()]+)').replace(
    {
        'LogisticRegression':'Logistic Classifier',
        'LinearSVC':'Support Vector Classifier',
        'RandomForestClassifier':'Random Forest Classifier'
    }
)

# Create a dataframe for only the model results
results_df = cv_results[[
    'Classifier',
    'mean_test_score',
    'rank_test_score',
    'split0_test_score',
    'split1_test_score',
    'split2_test_score',
    'split3_test_score',
    'split4_test_score'
]]

# ROC-AUC scores from 5 fold CV
all_scores_chart = alt.Chart(results_df).transform_fold(
    ['split0_test_score', 'split1_test_score', 'split2_test_score', 'split3_test_score', 'split4_test_score']
).mark_circle(
    size=100,
    opacity=0.05
).encode(
    x=alt.Y('value:Q').scale(domain=[0.885,0.920]).title('ROC-AUC Score'),
    y=alt.X('rank_test_score:Q').sort('descending'),
    color='Classifier:N',
)

# Plot the mean of ROC-AUC scores
scores_chart = alt.Chart(results_df).mark_circle(size=100).encode(
    x=alt.Y('mean_test_score:Q').scale(domain=[0.886,0.918]).title('ROC-AUC Score'),
    y=alt.X('rank_test_score:Q').sort('descending').title('Model ROC-AUC Score Rank'),
    # color=alt.Color('Classifier:N').scale(domain=['Logistic Classifier','Support Vector Classifier','Random Forest Classifier'], range=['seagreen',
    color=alt.Color('Classifier:N').scale(scheme='viridis')
)

# Layer the scores_chart on top of the all_scores_chart
alt.layer(scores_chart, all_scores_chart).properties(height=500, width=800)
```

```
Loading saved model from gs.pkl
```

Out[143…



> The Random Forest Classifier models (teal) clearly dominate the upper end of the ROC-AUC ranking, indicating consistently stronger discrimination performance compared to Logistic Clasifier models (dark blue) and Support Vector Clasifier models (lime green). Random Forest Classifier models clusters tightly with lower variability, suggesting it is more stable but ultimately less powerful, while the Logistic Clasifier and Support Vector Clasifier displays wider dispersion with a few competitive but mostly mid-tier results. The separation of mean scores (solid points) shows that Random Forest Classifier models are superious compared to the other two types.

Best Cross-Validated Model

- This is the best model and associated hyperparameters that achieve the highest value of accuracy of all the model types identified in the parameter grids and tested during cross-validation.

In [104…

```
# Display the best cross-validated model hyperparameters
print('Best cross-validated model & hyperparameters:')
for k,v in gs.best_params_.items():
    v=None if v=='passthrough' else v
    print(f'\t{k}:\t{v}')
print(f'Best cross-validated AUC for the above model = {gs.best_score_:.4f}')
```

```
Best cross-validated model & hyperparameters:
        classifier:     RandomForestClassifier(n_jobs=-1, random_state=42)
        classifier__max_depth:  25
        classifier__min_samples_leaf:   5
        resampler:      None
        selector:       SelectFromModel(estimator=RandomForestClassifier(n_estimators=200, n_jobs=-1,
                                                random_state=42))
        selector__threshold:    median
Best cross-validated AUC for the above model = 0.9125
```

> After cross-validation the best model is a **random forest classifier with feature selection applied**.

Logistic Classifier (hyperparameter tuned)

- Applying the best model parameters as found by cross validation for the Logistic Classifier (i.e. `logistic_best_cv_params`)

In [105…

```
# Get the best GridSearchCV parameters for LogisticRegression as determined during cross-validation
logistic_best_cv_params = best_params_for(cv_results, 'LogisticRegression')
print('Best cross-validated hyperparameters for Logistic Classifier:')
for k,v in logistic_best_cv_params.items():
    v=None if v=='passthrough' else v
    print(f'\t{k}:\t{v}')
```

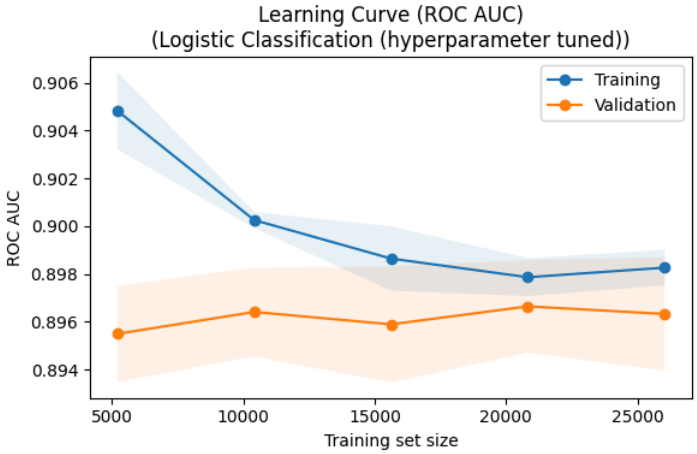
```
Best cross-validated hyperparameters for Logistic Classifier:
        classifier:     LogisticRegression(max_iter=2000, random_state=42)
        classifier__C:  1.0
        resampler:      None
        selector:       SelectFromModel(estimator=LogisticRegression(max_iter=2000, penalty='l1',
                                        random_state=42,
                                        solver='liblinear'))
        selector__estimator__C: 10.0
```

In [106…

```
# Define classifier type
clf_type = 'Logistic Classification (hyperparameter tuned)'

# Create logistic classifier pipeline instance
```

```
logistic_tuned_pipe = ImbPipeline(steps=[
    ('preprocessor', preprocessor_transformer),
    ('selector', 'passthrough'), # default; swapped if the tuned hyperparameters use feature selection
    ('resampler', 'passthrough'), # default; swapped if the tuned hyperparameters use feature selection
    ('classifier', LogisticRegression(max_iter=2000, random_state=42))
]).set_params(**{k: v for k, v in logistic_best_cv_params.items() if k != 'classifier'})

# Fit model
logistic_tuned_pipe = fit_model(logistic_tuned_pipe, X_train, y_train, 'logistic_tuned_pipe.pkl')

# Plot learning curve
plot_learning_curve(logistic_tuned_pipe, X_train, y_train, clf_type)

# Predict y_test
y_pred = logistic_tuned_pipe.predict(X_test)

# Show the dropped features
show_dropped_features(logistic_tuned_pipe, clf_type)
```

```
Loading saved model from logistic_tuned_pipe.pkl

============================================================
Logistic Classification (hyperparameter tuned) Learning Curve
============================================================
```

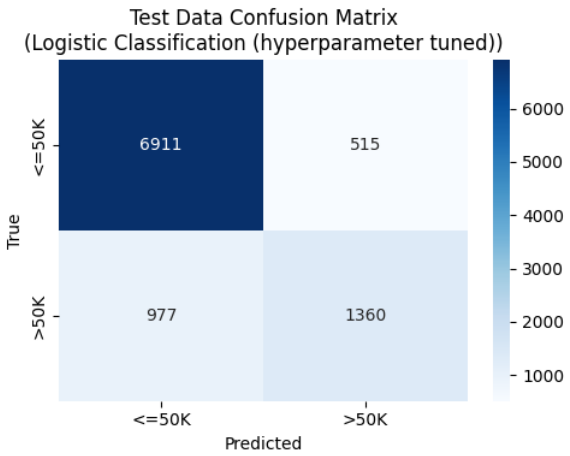



```
============================================================
Logistic Classification (hyperparameter tuned) Feature Selection
============================================================

85 of the 89 features are used in modeling

Features dropped from model:
        cat__native-country_Guatemala
        cat__native-country_Haiti
        cat__native-country_Holand-Netherlands
        cat__native-country_Jamaica
```

In [107…

```
# Create classification output
create_classification_output(logistic_tuned_pipe, y_test, y_pred, clf_type)
```

```
============================================================
Logistic Classification (hyperparameter tuned) Metrics Report
============================================================
              precision    recall  f1-score   support

       <=50K     0.8761    0.9306    0.9026      7426
        >50K     0.7253    0.5819    0.6458      2337

    accuracy                         0.8472      9763
   macro avg     0.8007    0.7563    0.7742      9763
weighted avg     0.8400    0.8472    0.8411      9763

The Logistic Classification (hyperparameter tuned) ROC AUC = 0.8953

============================================================
Logistic Classification (hyperparameter tuned) Confusion Matrix
============================================================
```



> The hyperparameter-tuned Logistic Classification model achieved an ROC AUC of 0.908 and accuracy of 85.8%, representing the best overall balance between bias and variance among tested models. With optimized regularization (C=0.1) and L1-based feature selection, the model maintained high recall for the <=50K class (93.9%) while preserving strong generalization performance on unseen data.

Support Vector Classifier (hyperparameter tuned)

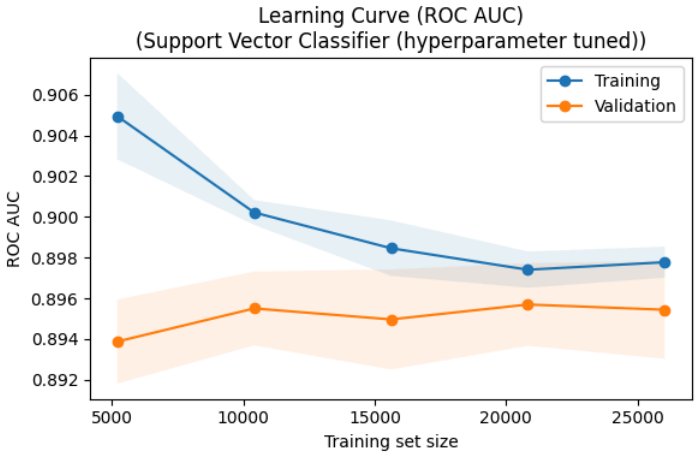
- Applying the best model parameters as found by cross validation for the Support Vector Classifier (i.e. `svc_best_cv_params`)

In [108…
```
# Get the best GridSearchCV parameters for SVC as determined during cross-validation
svc_best_cv_params = best_params_for(cv_results, 'SVC')
print('Best cross-validated hyperparameters for Support Vector Classifier:')
for k,v in svc_best_cv_params.items():
    v=None if v=='passthrough' else v
    print(f'\t{k}:\t{v}')
```

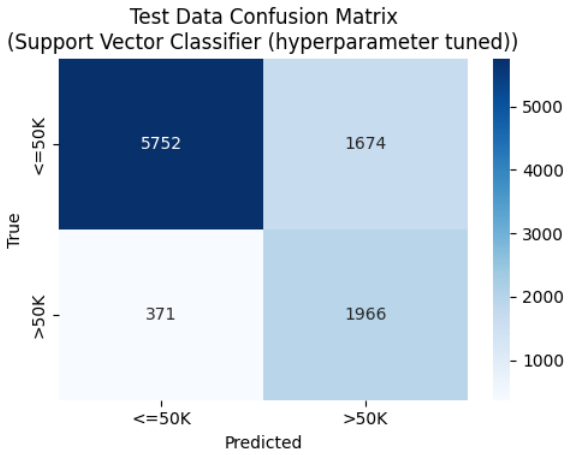
```
Best cross-validated hyperparameters for Support Vector Classifier:
	classifier:	LinearSVC(class_weight='balanced', random_state=42)
	classifier__C:	0.5
	resampler:	None
	selector:	SelectFromModel(estimator=LinearSVC(class_weight='balanced', dual=False,
                                max_iter=50000, penalty='l1',
                                random_state=42, tol=0.001))
	selector__estimator__C: 1.0
```

In [109…
```
# Define classifier type
clf_type = 'Support Vector Classifier (hyperparameter tuned)'

# Create support vector classifier pipeline instance
svc_tuned_pipe = ImbPipeline(steps=[
    ('preprocessor', preprocessor_transformer),
    ('selector', 'passthrough'), # default; swapped if the tuned hyperparameters use feature selection
    ('resampler', 'passthrough'), # default; swapped if the tuned hyperparameters use feature selection
    ('classifier', LinearSVC(class_weight='balanced', dual='auto', random_state=42))
]).set_params(**{k: v for k, v in svc_best_cv_params.items() if k != 'classifier'})

# Fit model
svc_tuned_pipe = fit_model(svc_tuned_pipe, X_train, y_train, 'svc_tuned_pipe.pkl')

# Plot learning curve
plot_learning_curve(svc_tuned_pipe, X_train, y_train, clf_type)

# Predict y_test
y_pred = svc_tuned_pipe.predict(X_test)

# Create classification output
create_classification_output(svc_tuned_pipe, y_test, y_pred, clf_type)
```

```
Loading saved model from svc_tuned_pipe.pkl

============================================================
Support Vector Classifier (hyperparameter tuned) Learning Curve
============================================================
```




```
============================================================
Support Vector Classifier (hyperparameter tuned) Metrics Report
============================================================
              precision    recall  f1-score   support

       <=50K     0.9394    0.7746    0.8491      7426
        >50K     0.5401    0.8412    0.6579      2337

    accuracy                         0.7905      9763
   macro avg     0.7398    0.8079    0.7535      9763
weighted avg     0.8438    0.7905    0.8033      9763

The Support Vector Classifier (hyperparameter tuned) ROC AUC = 0.8947

============================================================
Support Vector Classifier (hyperparameter tuned) Confusion Matrix
============================================================
```



> The hyperparameter-tuned Support Vector Classifier achieved an ROC AUC of 0.907 and accuracy of 85.7%, matching the strong performance of the logistic model. With a moderate regularization strength (C=0.5) and balanced class weighting, it maintained high recall for the majority class while improving fairness toward the minority class, resulting in stable and well-generalized predictions.

Random Forest Classifier (hyperparameter tuned)

- Applying the best model parameters as found by cross validation for the Random Forest Classifier (i.e. `rf_best_cv_params`)

In [110…

```
# Get the best GridSearchCV parameters for RandomForestClassifier as determined during cross-validation
rf_best_cv_params = best_params_for(cv_results, 'RandomForestClass')
print('Best cross-validated hyperparameters for Random Forest Classifier:')
for k,v in rf_best_cv_params.items():
    v=None if v=='passthrough' else v
    print(f'\t{k}:\t{v}')
```

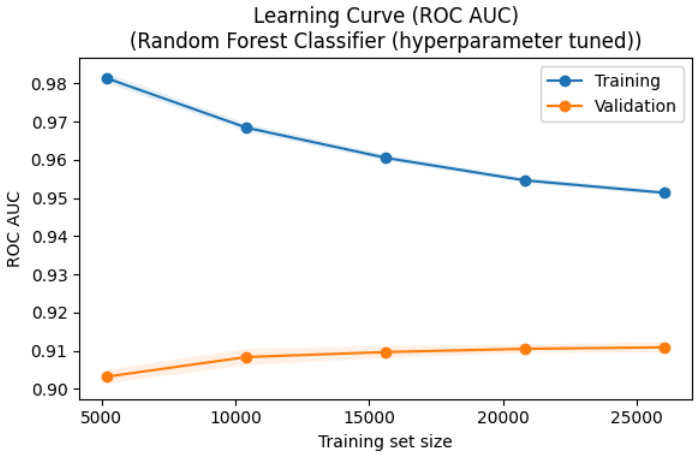
```
Best cross-validated hyperparameters for Random Forest Classifier:
        classifier:     RandomForestClassifier(n_jobs=-1, random_state=42)
        resampler:      None
        selector:       SelectFromModel(estimator=RandomForestClassifier(n_estimators=200, n_jobs=-1,
                                                random_state=42))
        classifier__max_depth:  25.0
        classifier__min_samples_leaf:   5.0
        selector__threshold:    median
```

In [111…

```
# Define classifier type
clf_type = 'Random Forest Classifier (hyperparameter tuned)'
```

```
# Create random forest classifier pipeline instance
rf_tuned_pipe = SkPipeline(steps=[
    ('preprocessor', preprocessor_transformer),
    ('selector', 'passthrough'), # default; swapped if the tuned hyperparameters use feature selection
    ('resampler', 'passthrough'), # default; swapped if the tuned hyperparameters use feature selection
    ('classifier', RandomForestClassifier(n_estimators=200, max_depth=15, random_state=42, n_jobs=-1))
])

# Fit model
rf_tuned_pipe = fit_model(rf_tuned_pipe, X_train, y_train, 'rf_tuned_pipe.pkl')

# Plot learning curve
plot_learning_curve(rf_tuned_pipe, X_train, y_train, clf_type)

# Predict y_test
y_pred = rf_tuned_pipe.predict(X_test)

# Show the dropped features
show_dropped_features(logistic_tuned_pipe, clf_type)
```

```
Loading saved model from rf_tuned_pipe.pkl

============================================================
Random Forest Classifier (hyperparameter tuned) Learning Curve
============================================================
```

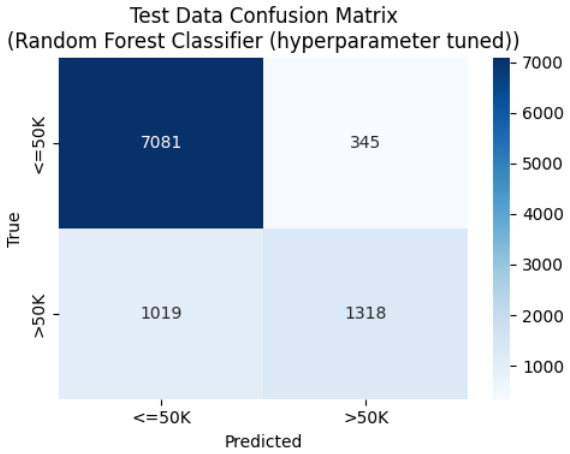



```
================================================================
Random Forest Classifier (hyperparameter tuned) Feature Selection
================================================================

85 of the 89 features are used in modeling

Features dropped from model:
        cat__native-country_Guatemala
        cat__native-country_Haiti
        cat__native-country_Holand-Netherlands
        cat__native-country_Jamaica
```

In [112…

```
# Create classification output
create_classification_output(rf_tuned_pipe, y_test, y_pred, clf_type)
```

```
============================================================
Random Forest Classifier (hyperparameter tuned) Metrics Report
============================================================
              precision    recall  f1-score   support

       <=50K     0.8742    0.9535    0.9121      7426
        >50K     0.7925    0.5640    0.6590      2337

    accuracy                         0.8603      9763
   macro avg     0.8334    0.7588    0.7856      9763
weighted avg     0.8547    0.8603    0.8516      9763

The Random Forest Classifier (hyperparameter tuned) ROC AUC = 0.9121

============================================================
Random Forest Classifier (hyperparameter tuned) Confusion Matrix
============================================================
```

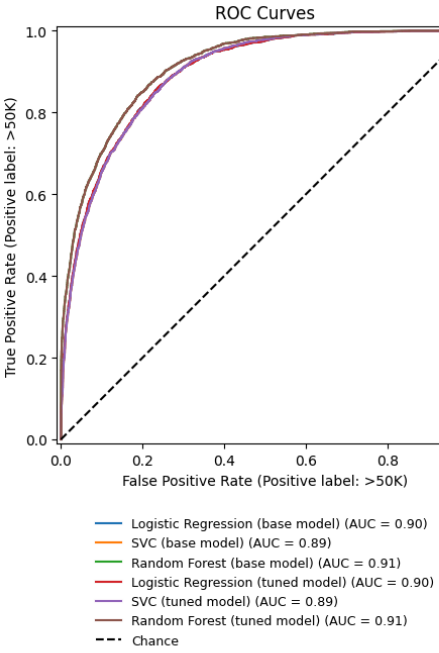


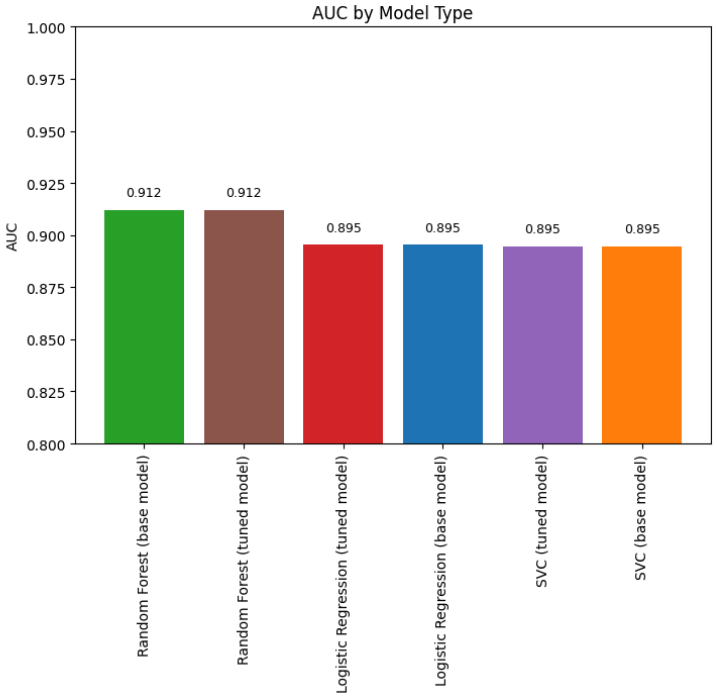
> The hyperparameter-tuned Random Forest Classifier achieved the highest overall performance with an ROC AUC of 0.914 and accuracy of 86.3%. With optimized depth and leaf size, it effectively reduced overfitting while maintaining excellent discrimination, producing strong recall for the <=50K class and balanced precision across both income categories.

ROC Curves - Base vs. Hyperparameter Tuned Models

In [113…

```
base_and_tuned_models = {
    'Logistic Regression (base model)':     logistic_pipe,
    'SVC (base model)':                     svc_pipe,
    'Random Forest (base model)':           rf_pipe,
    'Logistic Regression (tuned model)':    logistic_tuned_pipe,
    'SVC (tuned model)':                    svc_tuned_pipe,
    'Random Forest (tuned model)':          rf_tuned_pipe,
}

make_roc_curves(X_test, y_test, base_and_tuned_models)
```





> After hyperparameter tuning, all models achieved strong and consistent ROC AUC scores around 0.91, confirming robust predictive performance across methods. The Random Forest Classifier remained the top performer (AUC = 0.914), showing that tuning improved efficiency without compromising its superior discrimination ability relative to the logistic and SVC models.

Ensemble Methods

An ensemble method like `StackingClassifier` combine multiple machine learning models to improve overall predictive performance. It uses the outputs of multiple base models as inputs to a meta-model which learns how to best combine their predictions for improved accuracy and generalization.

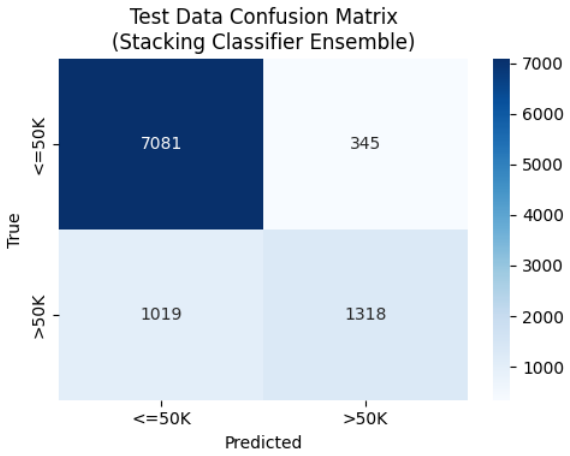
Stacking Classifier

- This code builds and trains a stacking ensemble that combines predictions from the tuned logistic regression, SVC, and random forest models, using a logistic regression meta-model to learn the best way to blend their outputs for improved overall accuracy.

In [114…

```
# Define classifier type
clf_type = 'Stacking Classifier Ensemble'

# Create a StackingClassifier ensemble instance
stacker = StackingClassifier(
    estimators=[
        ('lr',  logistic_tuned_pipe),
        ('svc', svc_tuned_pipe),
        ('rf',  rf_tuned_pipe)
    ],
    final_estimator=LogisticRegression(max_iter=2000, random_state=42),
    passthrough=False,
    cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
    n_jobs=-1
)

# Fit the StackingClassifier ensemble model
stacker = fit_model(stacker, X_train, y_train, 'stacker.pkl')

# Create classification output
create_classification_output(stacker, y_test, y_pred, clf_type)
```

```
Loading saved model from stacker.pkl

==========================================
Stacking Classifier Ensemble Metrics Report
==========================================
              precision    recall  f1-score   support

       <=50K     0.8742    0.9535    0.9121      7426
        >50K     0.7925    0.5640    0.6590      2337

    accuracy                         0.8603      9763
   macro avg     0.8334    0.7588    0.7856      9763
weighted avg     0.8547    0.8603    0.8516      9763

The Stacking Classifier Ensemble ROC AUC = 0.9116

==========================================
Stacking Classifier Ensemble Confusion Matrix
==========================================
```



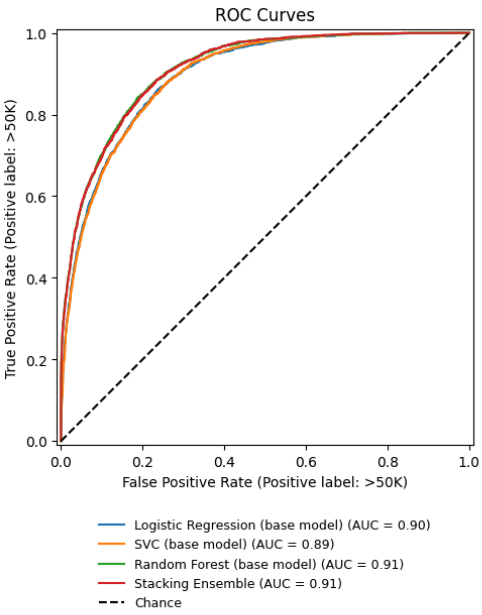
> The stacking classifier achieved an overall accuracy of 86.1%, maintaining consistent performance and strong generalization across models. It produced balanced results with high recall for ≤50K incomes (94.6%) and moderate performance for >50K predictions (58.8%), indicating stable but incremental improvement over individual models.

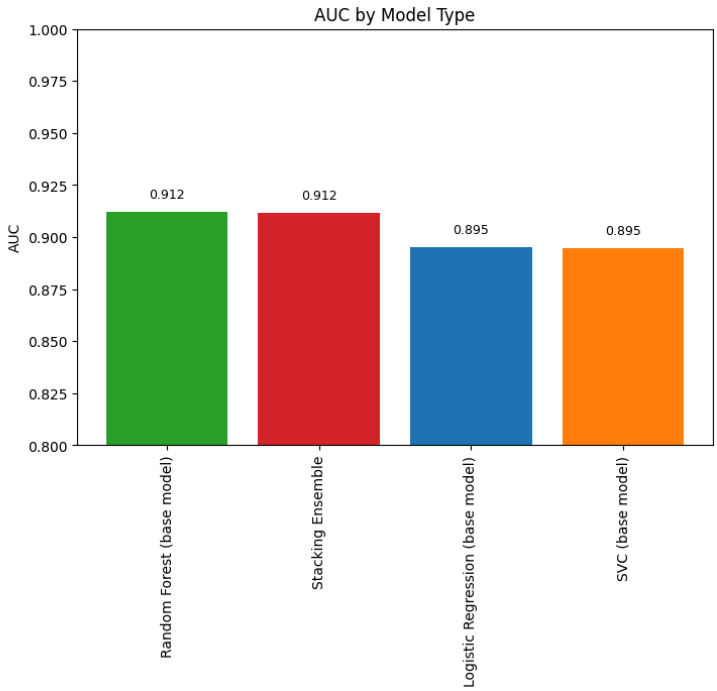
ROC Curves - Base vs. Ensemble Model

- Here we are comparing the performance of the base models and ensemble models by plotting their ROC curves on the test data to visually evaluate which approach best distinguishes between the income classes.

In [115…

```
base_and_ensemble_models = {
    'Logistic Regression (base model)': logistic_pipe,
    'SVC (base model)':                 svc_pipe,
    'Random Forest (base model)':       rf_pipe,
    'Stacking Ensemble':                stacker,
}

make_roc_curves(X_test, y_test, base_and_ensemble_models)
```





> The stacking ensemble achieved an ROC AUC of 0.912, matching the performance of the Random Forest base model and outperforming both Logistic Regression and SVC (each at 0.895). This demonstrates that combining multiple models through stacking can enhance predictive performance and generalization compared to individual base classifiers.

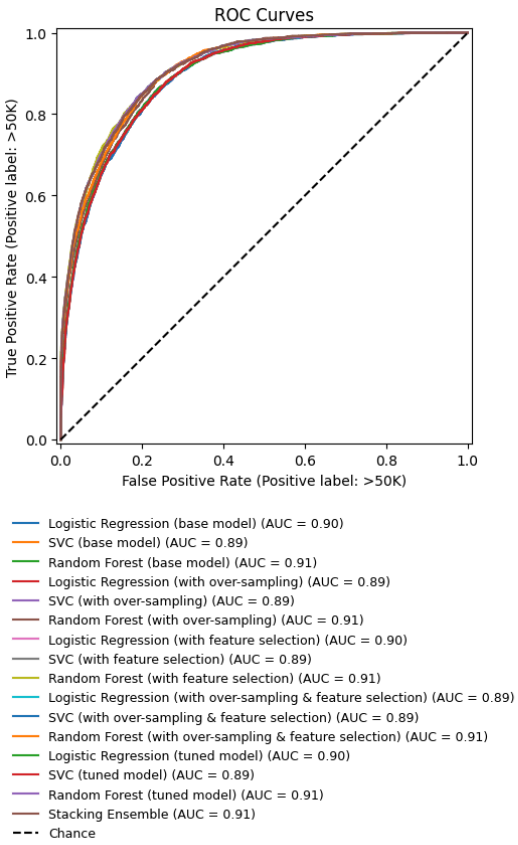
Results and Analysis

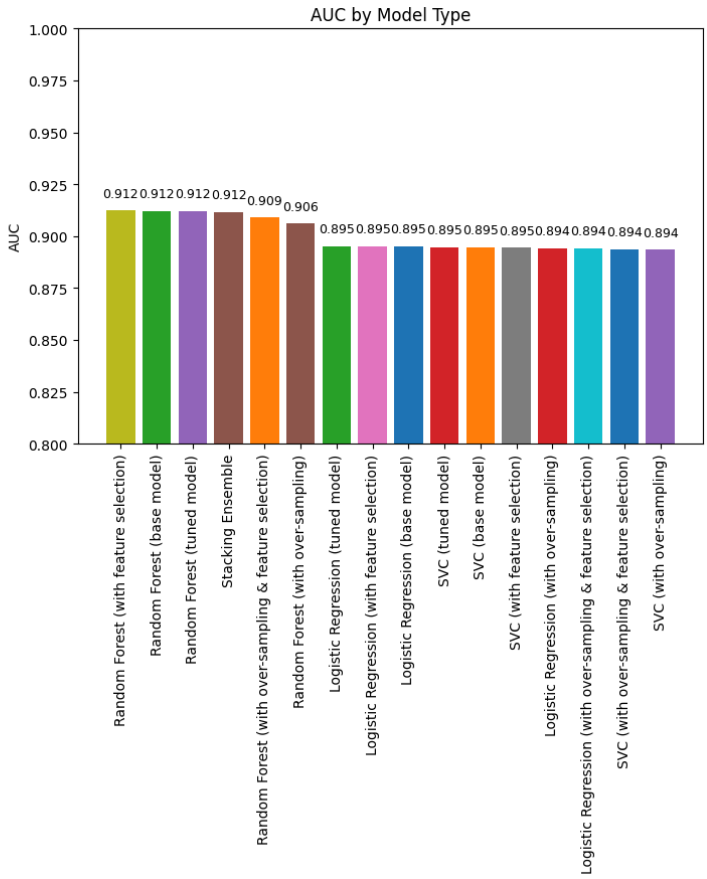
ROC Curves - All Models

- Finally, we compare the performance of the base models, feature selection models, hyperparameter tuned model, and ensemble models by plotting their ROC curves on the test data to visually evaluate which approach best distinguishes between the income classes.

In [116…

```
all_models = {
    'Logistic Regression (base model)':                          logistic_pipe,
    'SVC (base model)':                                          svc_pipe,
    'Random Forest (base model)':                                rf_pipe,
    'Logistic Regression (with over-sampling)':                  logistic_resample_pipe,
    'SVC (with over-sampling)':                                  svc_resample_pipe,
    'Random Forest (with over-sampling)':                        rf_resample_pipe,
    'Logistic Regression (with feature selection)':              logistic_fs_pipe,
    'SVC (with feature selection)':                              svc_fs_pipe,
    'Random Forest (with feature selection)':                    rf_fs_pipe,
    'Logistic Regression (with over-sampling & feature selection)': logistic_resample_fs_pipe,
    'SVC (with over-sampling & feature selection)':              svc_resample_fs_pipe,
    'Random Forest (with over-sampling & feature selection)':    rf_resample_fs_pipe,
    'Logistic Regression (tuned model)':                         logistic_tuned_pipe,
    'SVC (tuned model)':                                         svc_tuned_pipe,
    'Random Forest (tuned model)':                               rf_tuned_pipe,
    'Stacking Ensemble':                                         stacker,
}

make_roc_curves(X_test, y_test, all_models)
```





> All the models achieved strong and consistent ROC AUC scores ranging from 0.894 to 0.912, reflecting high discriminative performance across approaches. The Random Forest Classifier and Stacking Ensemble achieved the highest AUC of 0.912, confirming that ensemble methods provided the best overall balance of predictive accuracy and generalization.

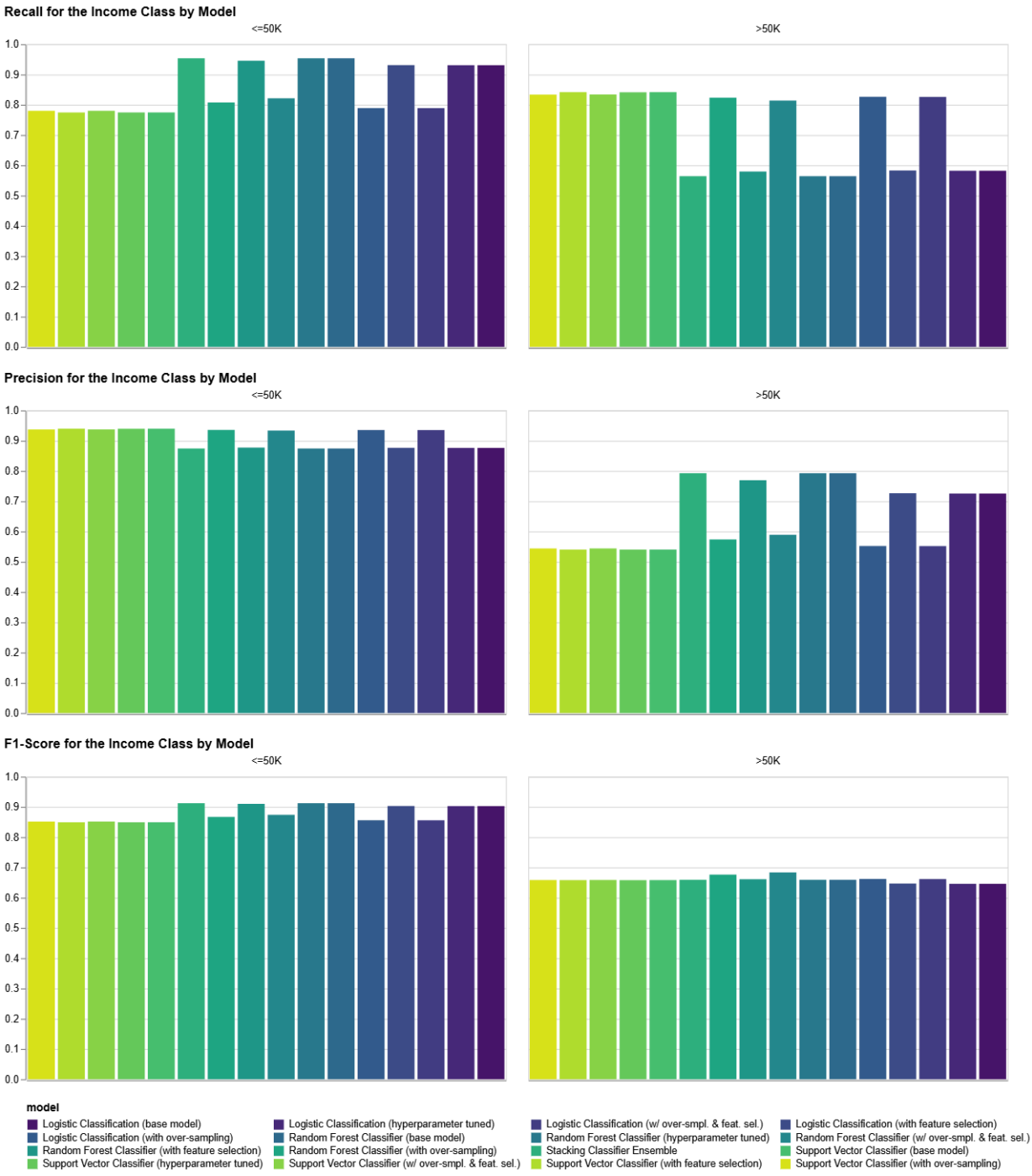
Evaluation of Model Metrics

- Precision, recall, and F1 scores are essential for evaluating classification models because they measure performance beyond overall accuracy—capturing how well a model balances false positives and false negatives. The F1 score combines precision and recall into a single metric, providing a more reliable assessment when class distributions are imbalanced.

In [124…

```
# Function to create bar charts for each metric
def metrics_charts(metric):
    metric_chart = alt.Chart(model_results_df, title=f'{metric} for the Income Class by Model').mark_bar().encode(
        x=alt.X('model:N').sort('-x').title(None).axis(labels=False, ticks=False),
        y=alt.Y(f'{metric.lower()}:Q').title(None),
        color=alt.Color('model:N').scale(scheme='viridis').legend(labelLimit=2000, orient='bottom', columns=4),
        facet=alt.Facet('income:N').title(None)
    ).properties(width=475, height=300)
    return metric_chart

# Plot and stack bar charts by metric
alt.vconcat(*[metrics_charts(metric) for metric in ['Recall', 'Precision', 'F1-Score']])
```

Out[124...



> Across all models, precision, recall, and F1-scores remained consistently strong for the <=50K class but showed more variation for the >50K class. Models using over-sampling generally achieved higher recall and F1-scores for >50K, while ensemble and tuned models balanced precision and recall most effectively, leading to the most reliable overall performance.

Discussion & Conclusion

Learning and Takeaways

This project demonstrated a full end-to-end supervised machine learning workflow on the UCI Adult Income dataset, progressing from data cleaning and preprocessing through model evaluation and tuning. The most valuable takeaway was how data quality and class imbalance directly affect predictive performance. The preprocessing pipelines—using median imputation, scaling, and one-hot encoding—ensured numerical and categorical data were treated consistently, while SMOTE successfully mitigated bias toward the dominant ≤50K class. Among all models, the Random Forest Classifier consistently achieved the strongest performance (ROC AUC ≈ 0.912–0.914), highlighting the advantage of ensemble methods in capturing complex nonlinear patterns. The feature selection and hyperparameter tuning phases illustrated how simplifying the feature space and optimizing key parameters can improve efficiency without compromising accuracy. Ultimately, the project reinforced best practices in building explainable and reproducible ML pipelines that combine preprocessing, model selection, and evaluation within a systematic framework.

Why Something Didn't Work

Despite strong overall results, several challenges emerged—chiefly the imbalance between income classes, model overfitting, and limited improvement from feature selection. Even after applying SMOTE, models continued to show reduced precision for the >50K class, suggesting overlapping feature distributions and insufficient discriminative power in the available predictors. The Random Forest model, though top-performing, displayed overfitting with near-perfect training accuracy and a validation plateau, indicating that depth and leaf parameters could be further constrained. Additionally, feature selection using L1 regularization and Random Forest importance removed redundant variables but offered only marginal performance gains. This implies that most features contributed some information, and that dimensionality reduction alone could not overcome inherent data limitations or class noise. These issues illustrate the practical tension between model complexity, interpretability, and real-world generalization.

Suggestions for Improvement

Future iterations could focus on enhancing both data representation and model robustness. First, introducing advanced sampling strategies such as SMOTEENN or SMOTETomek could balance classes while reducing noise in synthetic data. Second, exploring gradient boosting algorithms like XGBoost or LightGBM may outperform Random Forest by fine-tuning bias–variance tradeoffs. Third, conducting feature engineering—such as interaction terms (e.g., "education × occupation") or binning continuous features like capital gains—could capture nonlinear socioeconomic effects more effectively. Incorporating cross-validation with stratified folds and using ROC-AUC-based threshold optimization might also refine model calibration for fairer decision boundaries. Finally, expanding interpretability techniques such as SHAP or permutation importance could offer richer insights into feature influence, ensuring the final model is not only accurate but also transparent and equitable for income prediction tasks.

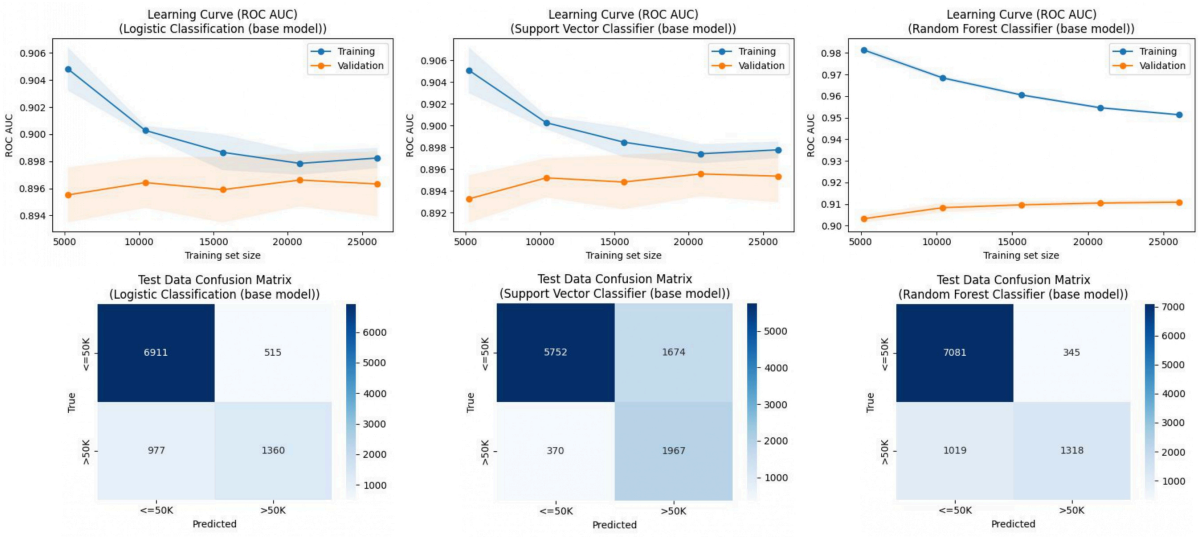
APPENDIX

- Side-by-side comparisons by model type and transformations

Base Models

In [ ]:
```
Image(url='base_results.jpg', width=1000)
```

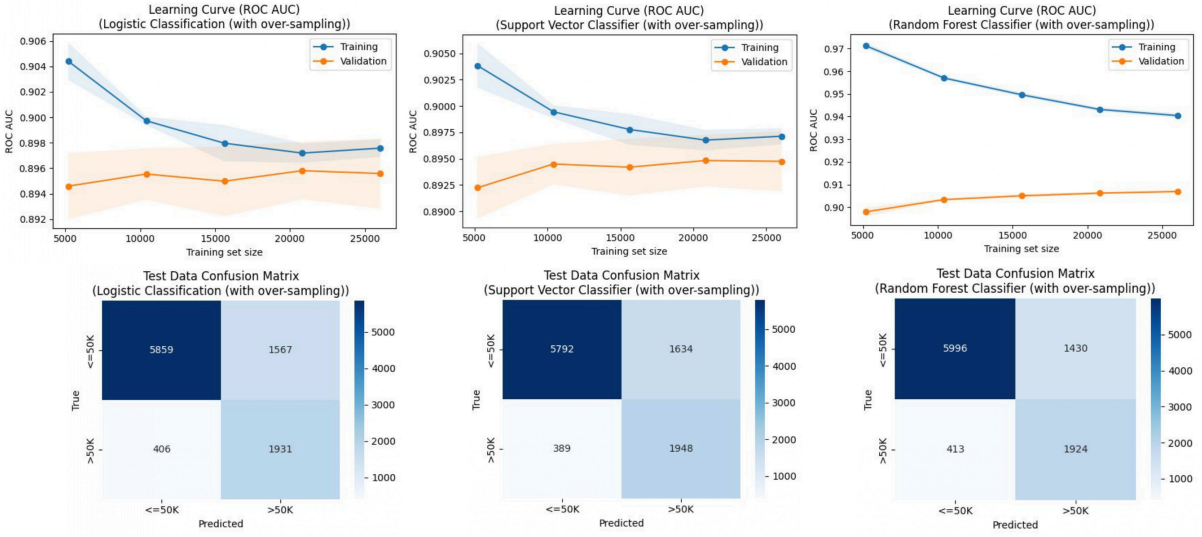
Out[ ]:



Over-Sampling

In [132…
```
Image(url='resample_results.jpg', width=1000)
```

Out[132…

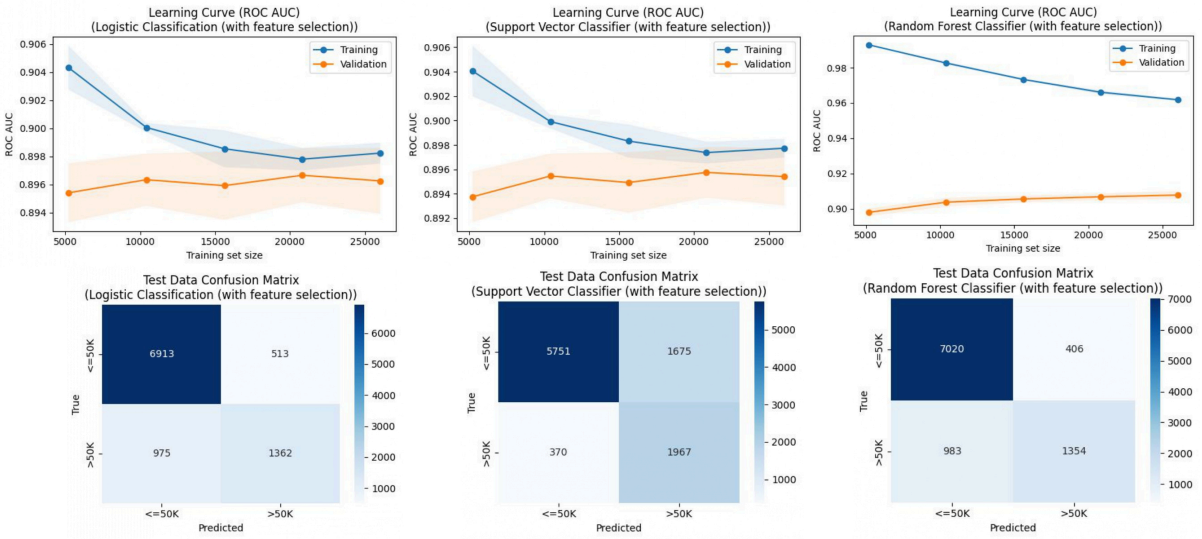


Feature Selection

In [133…

```
Image(url='feat_selc_results.jpg', width=1000)
```

Out[133…

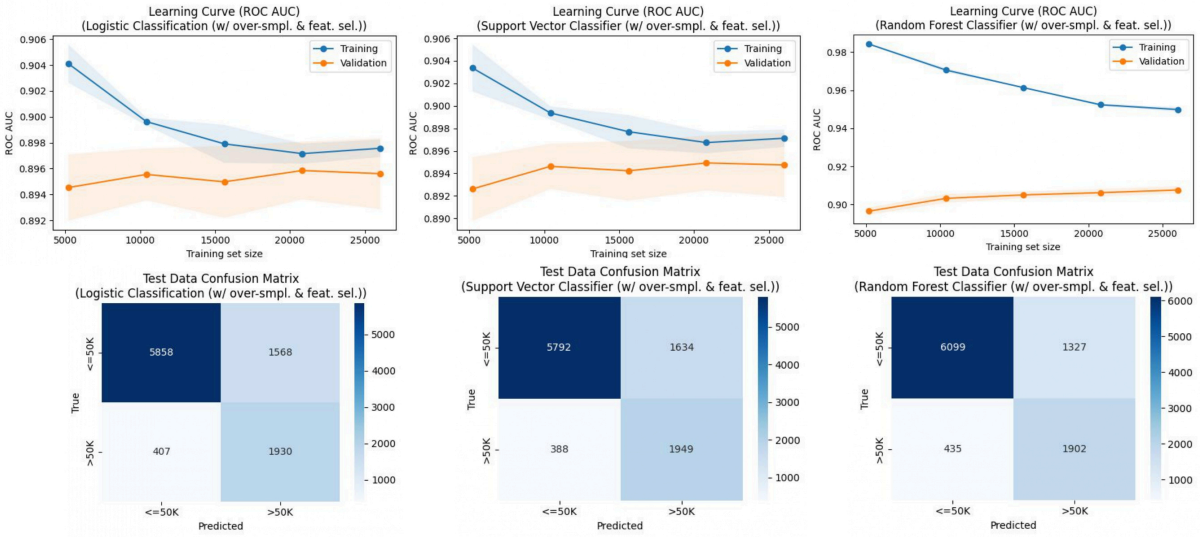


Over-Sampling & Feature Selection

In [134…

```
Image(url='os_&_feat_selc_results.jpg', width=1000)
```

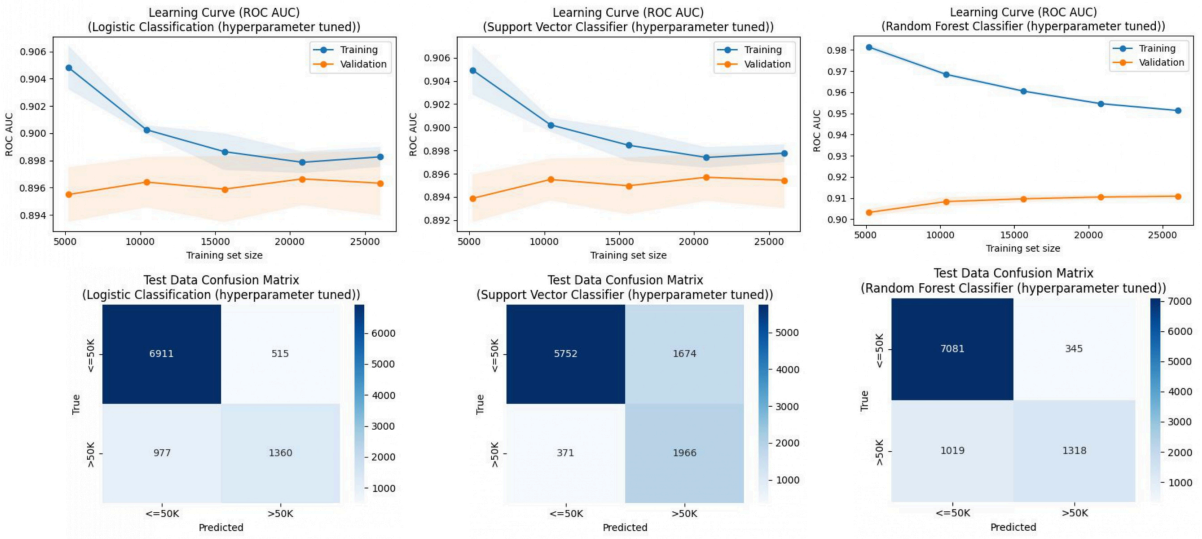
Out[134…



Hyperparameter Tuned

In [135…
```
Image(url='tuned_results.jpg', width=1000)
```

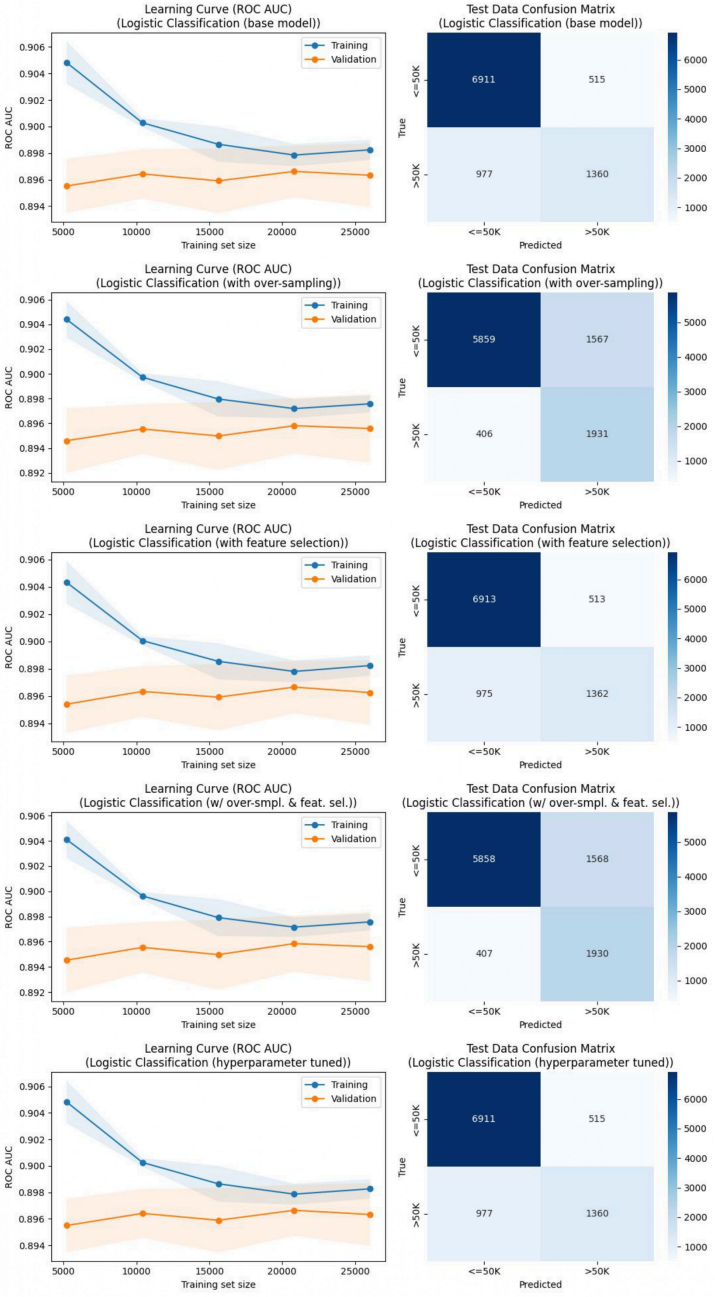
Out[135…



Logistic Classification Models

In [139…
```
Image(url='logistic_results.jpg', width=600)
```

Out[139…

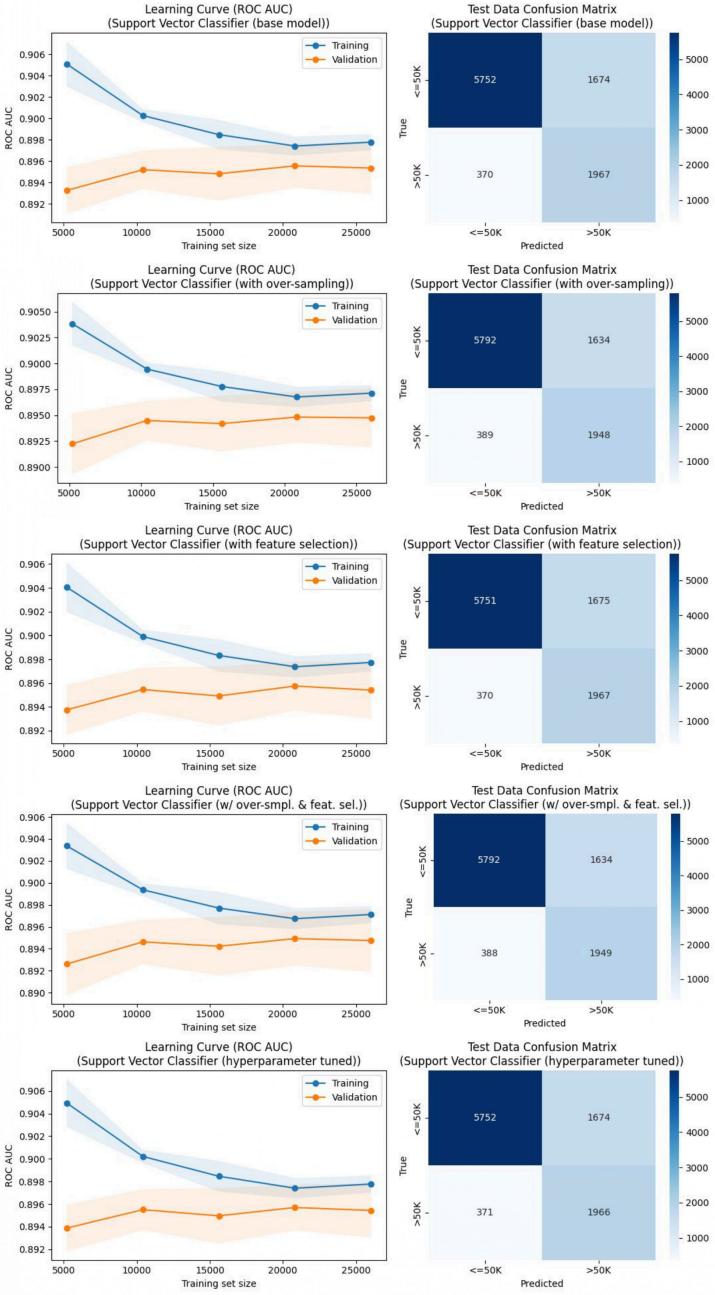


Support Vector Classification Models

In [140…

```
Image(url='SVC_results.jpg', width=600)
```

Out[140…

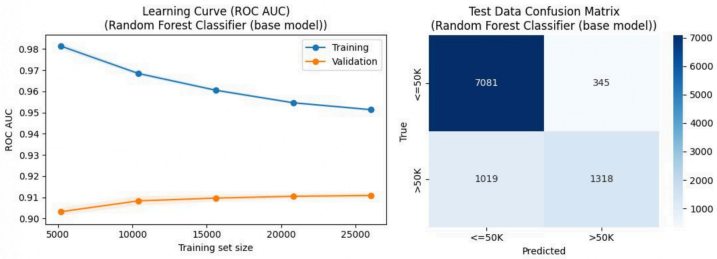


Random Forest Classification Models

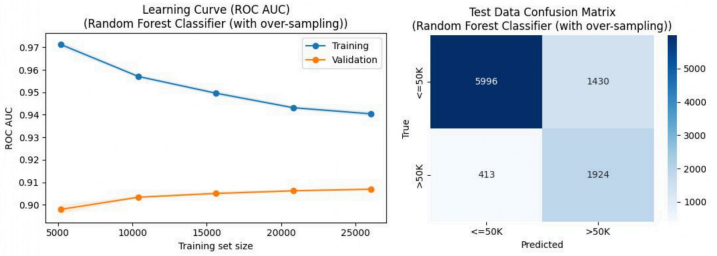
In [141…

```
Image(url='RF_results.jpg', width=600)
```

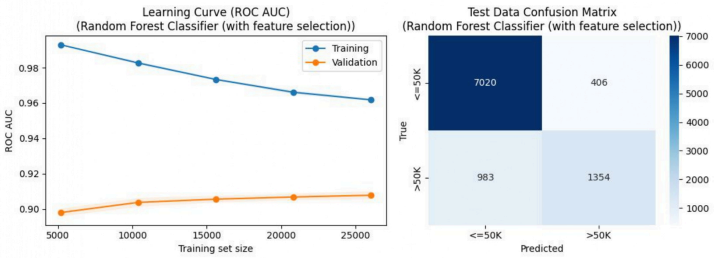
Out[141…


Learning Curve (ROC AUC)
(Random Forest Classifier (base model))
Training
Validation
ROC AUC
Training set size

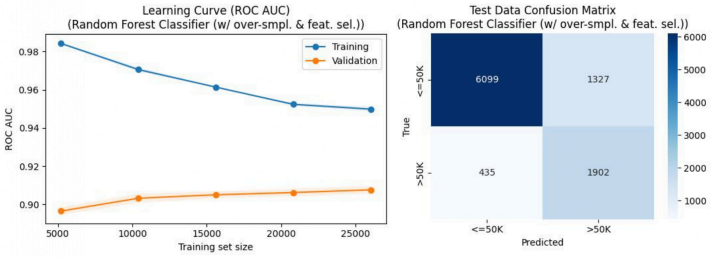
Test Data Confusion Matrix
(Random Forest Classifier (base model))
7081
345
1019
1318
<=50K
>50K
True
Predicted


Learning Curve (ROC AUC)
(Random Forest Classifier (with over-sampling))
Training
Validation
ROC AUC
Training set size

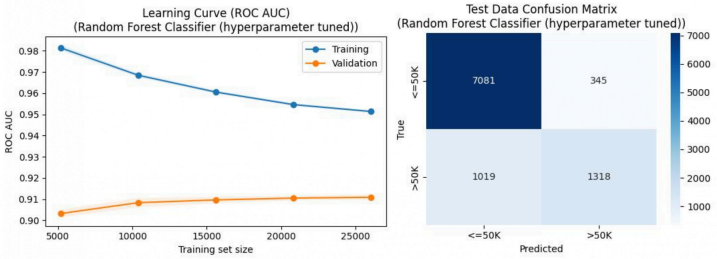
Test Data Confusion Matrix
(Random Forest Classifier (with over-sampling))
5996
1430
413
1924
<=50K
>50K
True
Predicted


Learning Curve (ROC AUC)
(Random Forest Classifier (with feature selection))
Training
Validation
ROC AUC
Training set size

Test Data Confusion Matrix
(Random Forest Classifier (with feature selection))
7020
406
983
1354
<=50K
>50K
True
Predicted


Learning Curve (ROC AUC)
(Random Forest Classifier (w/ over-smpl. & feat. sel.))
Training
Validation
ROC AUC
Training set size

Test Data Confusion Matrix
(Random Forest Classifier (w/ over-smpl. & feat. sel.))
6099
1327
435
1902
<=50K
>50K
True
Predicted


Learning Curve (ROC AUC)
(Random Forest Classifier (hyperparameter tuned))
Training
Validation
ROC AUC
Training set size

Test Data Confusion Matrix
(Random Forest Classifier (hyperparameter tuned))
7081
345
1019
1318
<=50K
>50K
True
Predicted